# Contribution à la définition et à la mise en oeuvre d'un dialogue adaptatif pour la CAO

Isabelle Stémard

# THÈSE

**Présentée à**

**l'UNIVERSITÉ DE METZ**

Pour l'obtention du grade de :
DOCTEUR de l'UNIVERSITÉ DE METZ

**Spécialité : INFORMATIQUE**

Isabelle STÉMART

# CONTRIBUTION À LA DÉFINITION ET À LA MISE EN ŒUVRE D'UN DIALOGUE ADAPTATIF POUR LA CAO

Soutenue à Metz, le 15 janvier 1997

**Composition du jury :**

| | | | |
|---|---|---|---|
| *Directeur de thèse* : | Yvon | GARDAN | (Professeur à l'Université de Metz) |
| *Rapporteurs* : | Philippe | COIFFET | (Professeur et directeur de recherche au CNRS) |
| | Umberto | CUGINI | (Professeur à l'Université de Parme) |
| *Examinateurs* : | Didier | GALMICHE | (Habilité à diriger les recherches à l'Université de Nancy I) |
| | Jean-Pierre | JUNG | (Professeur à l'Université de Metz) |
| | Michel | POTIER-FERRY | (Professeur à l'Université de Metz) |

LABORATOIRE DE RECHERCHE EN INFORMATIQUE DE METZ

# THÈSE

**Présentée à**

## l'UNIVERSITÉ DE METZ

Pour l'obtention du grade de :
DOCTEUR de l'UNIVERSITÉ DE METZ



**Spécialité : INFORMATIQUE**

Isabelle STÉMART

# CONTRIBUTION À LA DÉFINITION ET À LA MISE EN ŒUVRE D'UN DIALOGUE ADAPTATIF POUR LA CAO

Soutenue à Metz, le 15 janvier 1997

**Composition du jury :**

| | | | |
|---|---|---|---|
| *Directeur de thèse* : | Yvon | GARDAN | (Professeur à l'Université de Metz) |
| *Rapporteurs* : | Philippe | COIFFET | (Professeur et directeur de recherche au CNRS) |
| | Umberto | CUGINI | (Professeur à l'Université de Parme) |
| *Examinateurs* : | Didier | GALMICHE | (Habilité à diriger les recherches à l'Université de Nancy I) |
| | Jean-Pierre | JUNG | (Professeur à l'Université de Metz) |
| | Michel | POTIER-FERRY | (Professeur à l'Université de Metz) |

LABORATOIRE DE RECHERCHE EN INFORMATIQUE DE METZ

*À mes parents,*

*À Vincent,*

*« Ainsi, nous savons que compte moins l'œuvre que l'expérience de sa recherche et qu'un artiste est toujours prêt à sacrifier l'accomplissement de son ouvrage à la vérité du mouvement qui y conduit. »*
*Maurice BLANCHOT (1907)*

*« La connaissance est impossible. Mais, je ne peux pas me résigner à ne connaître que les murs de la prison. »*
*Eugène IONESCO (1912-1994)*

## REMERCIEMENTS.

La préparation d'une thèse est un travail de longue haleine qui ne peut aboutir sans l'aide et le soutien d'un grand nombre de personnes. Ainsi, je tiens à remercier dans ces quelques lignes toutes les personnes qui ont pu de près ou de loin m'accompagner dans la préparation de cette thèse.

À Monsieur le professeur Y. GARDAN, qui m'a guidée avec ferveur dans mon travail en tant que directeur de recherche, je ne saurais que trop exprimer ma gratitude non seulement pour les précieux conseils dont il a pu me faire part, mais aussi pour la confiance qu'il m'a accordée.

Je remercie messieurs P. COIFFET, professeur et directeur de recherche au CNRS, et U. CUGINI, professeur à l'Université de Parme, tout d'abord pour avoir accepté d'être rapporteurs, puis pour leurs critiques, car ils ont su porter un autre regard sur mes travaux, ce qui est fructueux dans le domaine de la recherche.

Que messieurs D. GALMICHE, habilité à diriger les recherches à l'Université de Nancy I, J.-P. JUNG, professeur à l'Université de Metz, et M. POTIER-FERRY, professeur à l'Université de Metz, soient remerciés pour leur participation en tant que membres du jury et pour avoir ainsi accepté de juger mon travail.

Je voudrais dire un grand merci à Benoît, membre de l'équipe dialogue du LRIM (Laboratoire de Recherche en Informatique de Metz), pour ses critiques pertinentes et le savoir qu'il m'a transmis pendant nos réunions de travail.

À Estelle, Cathy, Sandrine et Frédéric, je tiens à exprimer ma reconnaissance pour leur soutien, leur disponibilité et leur amitié.

Je remercie également toute l'équipe du LRIM, qui, par l'atmosphère conviviale qui s'y dégage, m'a offert un cadre de travail agréable, et qui a montré une grande disponibilité des personnes pour me faire partager leur expérience.

Enfin, je ne saurais oublier mes proches qui m'ont soutenue dans les moments difficiles et qui m'ont permis de me consacrer entièrement à la préparation de cette thèse.

**TABLE DES MATIÈRES.**

## INTRODUCTION.

La CFAO (Conception et Fabrication Assistées par Ordinateur) fait partie des domaines dans lesquels l'homme est très sollicité : il décrit virtuellement des produits à fabriquer. La technologie qu'elle véhicule le rend certes plus performant, mais souvent elle le commande également. On explique ce phénomène par le fait que les systèmes construits ne disposent pas d'informations suffisantes pour être parés à toute éventualité lors d'un libre échange avec l'utilisateur final. Dans de nombreuses applications, on constate que l'utilisateur doit s'adapter et même parfois renoncer à atteindre tous les buts qu'il s'est fixé.

Ainsi, de nombreux travaux tentent d'inverser la tendance en considérant l'utilisateur tel qu'il est avec son comportement opportuniste et le système tel qu'il devrait être i.e. adaptatif. Nous nous intéressons tout au long de ce mémoire aux techniques à mettre en œuvre pour obtenir un dialogue homme/machine en accord avec cette nouvelle tendance. En d'autres termes, il s'agit d'améliorer les primitives de dialogue pour que l'utilisateur ne soit plus contraint dans ses interactions avec le système : le dialogue est adaptatif dans la mesure où il est adapté à l'utilisateur et il l'aide à atteindre son but (explications ...).

Dans le premier chapitre, nous présentons les différentes évolutions envisageables pour rendre le dialogue homme/machine plus performant. C'est ainsi que nous montrons le grand éventail de possibilités offertes à chaque étape, de la spécification de l'interface à l'utilisation effective de l'application. En effet, de nombreux modèles sont décrits pour cerner au mieux le dialogue et, de manière plus globale, l'interface : modèles d'architecture, modèles de dialogue, modèles d'interactions ... Les techniques de construction d'interfaces sont nombreuses et influent considérablement sur le résultat obtenu. Nous soulevons alors le problème de la capture de la connaissance du concepteur. Il s'agit de savoir qui peut fournir la connaissance (un expert, un utilisateur final ...), quelle est cette connaissance, comment l'acquérir et l'exploiter au mieux. Nous orientons notre propos sur l'IA (Intelligence Artificielle) qui peut apporter des solutions à ce sujet. Ensuite, nous montrons une autre facette dans l'amélioration du dialogue en présentant le bénéfice apporté par l'introduction des interfaces 3D (trois dimensions) que l'on rencontre dans les systèmes de réalité virtuelle.

Devant l'étendue des possibilités pour rendre le dialogue plus adaptatif, nous intervenons tout d'abord, dans le deuxième chapitre, sur la construction du dialogue. Nous proposons une approche duale permettant à des utilisateurs de compétences différentes en matière de dialogue de faire part de leur connaissance. Dans un premier temps, il s'agit de proposer une méthode destinée à des spécialistes qui manipulent aisément les concepts de dialogue. Dans un second temps, nous introduisons une méthode plus intuitive qui déduit le dialogue à partir de connaissances modélisées et acquises d'utilisateurs non expérimentés dans la conception du dialogue. Il s'agit pour l'utilisateur de décrire des comportements sur les objets de son application ; ces comportements sont analysés et interprétés pour modéliser le dialogue.

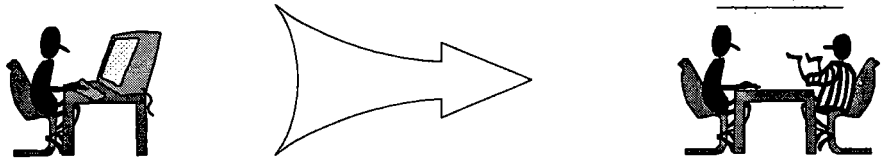
Dans le troisième chapitre, nous nous intéressons à l'amélioration des manipulations proposées à tout opérateur (utilisateur final). Nous sommes partis de la constatation que les outils de manipulations proposés sont si variés qu'il n'existe pas de concept plus général permettant de les centraliser et apportant ainsi une meilleure gestion du dialogue les concernant. Nous introduisons le concept des manipulateurs pour décrire des outils de manipulations adaptés aux opérateurs. Nous soulignons le besoin pour le dialogue de considérer qu'il existe effectivement différents opérateurs.

Les systèmes de CFAO sont en effet accessibles à des opérateurs de compétences différentes, car ils proposent des fonctionnalités très variées. Rappelons que les produits devenant de plus en plus complexes à décrire, leur conception s'effectue souvent par affinages, ce qui peut impliquer l'intervention d'opérateurs spécialisés à chaque étape de conception. De plus, les systèmes de réalité virtuelle entraînent souvent des environnements multi-utilisateurs. Le problème est d'obtenir un dialogue adapté à n'importe quel utilisateur. Le quatrième chapitre apporte une solution en considérant un véritable modèle des opérateurs. Chaque opérateur est perçu comme un objet à part entière de l'application ; on centralise sa connaissance afin de favoriser le dialogue.

Les annexes que nous présentons à la fin de ce mémoire décrivent les différentes implémentations qui nous ont permis d'illustrer nos travaux.

# CHAPITRE 1.

# VERS UN DIALOGUE ADAPTATIF.



**Dialogue homme/machine** **Dialogue homme/homme amélioré**

## 1. INTRODUCTION.

Si l'on se tourne un instant vers le passé, on constate que les applications informatiques étaient loin d'être adaptées à n'importe quel utilisateur. Non seulement, la patience était de rigueur pour attendre une réponse de la part du système, mais le protocole lui-même de communication entre l'utilisateur et le système était de très bas niveau. Un bon nombre de raisons explique ces difficultés de dialogue entre l'utilisateur et le système, parmi lesquelles [PRE 94] : le coût élevé du matériel, l'utilisation quasi exclusive par les spécialistes qui connaissaient, quant à eux, parfaitement le protocole de communication avec le système (les cartes perforées ...) ou encore le manque d'expérience, qui limitait les perspectives de recherche pour pouvoir trouver comment faciliter l'utilisation des systèmes. Les chercheurs et les ingénieurs se préoccupaient de l'amélioration du matériel et des applications proprement dites avant celle du dialogue homme/machine.

De nos jours, les systèmes sont plus performants ; le matériel et les applications évoluent toujours, mais la communication entre l'homme et la machine est devenue également un centre d'intérêt dans la réalisation d'un système. En effet, l'informatique s'est installée, voire imposée, dans de très nombreux domaines (industriel, commercial, artistique, domestique ...), d'où des utilisateurs hétérogènes que l'on ne peut plus négliger.

Le langage informatique est perceptible selon trois niveaux [DIX 93] : le niveau lexical, qui comprend la forme des icônes sur l'écran et les touches réellement sélectionnées ; le niveau syntaxique, qui concerne l'ordre et la structure des entrées/sorties ; le niveau sémantique, qui reflète les sens de la conversation par les effets sur les structures de données internes et/ou sur le monde extérieur. Le dialogue gère la structure de la conversation qui s'établit entre l'utilisateur et le système : il représente le niveau syntaxique du langage informatique et intègre parfois également le niveau lexical. Le problème que l'on rencontre dans la plupart des systèmes actuels est une certaine lourdeur dans le dialogue, car il finit par imposer une structure à l'utilisateur qui perd de sa créativité [EDM 94] : l'utilisateur peut percevoir des éléments qui ne sont pas forcément explicitement représentés et vouloir les manipuler sans passer par une étape de description, puisque pour lui ces éléments sont évidents et naturels. Ce que l'on demande maintenant au dialogue, ce n'est plus seulement le fait d'être fonctionnel, mais c'est surtout d'être à l'écoute de l'utilisateur. On se dirige vers des dialogues adaptatifs. Pour obtenir un tel résultat, ce n'est pas seulement le dialogue qui est concerné, mais c'est la gestion complète de la communication i.e. l'interface, de sa spécification à son utilisation. À tout moment dans l'élaboration et l'utilisation de l'interface, des utilisateurs interviennent pour exprimer des besoins particuliers. La manière de percevoir ces besoins est capitale. Pour y parvenir, des mécanismes toujours plus performants sont mis en jeu.

Dans ce chapitre, nous présentons différentes solutions pour répondre au mieux aux attentes des utilisateurs. Nous les jugeons complémentaires dans la mesure où elles abordent le problème à des niveaux d'abstraction différents. Notre propos s'oriente exclusivement sur les interfaces graphiques interactives, car nous nous intéressons aux dialogues dédiés aux systèmes de CFAO (Conception et Fabrication Assistées par Ordinateur). Pour cela, nous précisons les besoins particuliers requis pour de

tels dialogues. Puis, en montrant comment se définit une interface, nous insistons sur les techniques employées pour bien la construire : des langages les plus rébarbatifs, contraignants pour l'utilisateur, à l'automatisation, voire la génération. Ensuite, nous nous intéressons à l'IA (Intelligence Artificielle) qui procure des techniques de gestion de la connaissance qui, appliquées au dialogue, permettent une meilleure prise en compte de l'utilisateur et facilite ses interventions. Enfin, nous décrivons les nouvelles interfaces qui apparaissent pour répondre toujours à ce même besoin de s'adapter à l'utilisateur.

## 2. L'INTERFACE HOMME/MACHINE ET LA CFAO.

Initialement, on assimile à l'interface homme/machine le traitement de tous les aspects du système auxquels l'utilisateur est confronté [MOR 81]. En dix ans, cette définition s'est précisée : c'est une discipline qui concerne la conception, l'évaluation et l'exécution des systèmes informatiques interactifs pour une utilisation humaine, ainsi que l'étude des phénomènes principaux les entourant [ACM 92].

Les interfaces sont parfois très différentes selon le type d'application considéré : elles ne sont pas abordées de la même façon pour une secrétaire (bureautique) ou pour un concepteur de robots (réalité virtuelle). Nous concentrons notre attention sur le domaine qui nous intéresse tout particulièrement, la CFAO. Ce domaine regroupe les aides informatiques, même très ponctuelles, apportées à l'entreprise jusqu'à la réalisation du produit (pris dans un sens large) ; l'homme et l'ordinateur sont rassemblés pour résoudre des problèmes techniques dans une équipe qui associe étroitement les meilleures qualités de chacun d'eux, de telle sorte que l'équipe travaille mieux que séparément [GAR 91]. Le dialogue (et l'interface en général) rapproche l'homme et la machine pour promouvoir l'efficacité du système.

Tout d'abord, nous étudions les besoins d'une interface et nous insistons sur les besoins spécifiques aux interfaces des systèmes de CFAO. Ensuite, nous nous attachons à expliquer les différents composants d'une interface. Enfin, nous nous intéressons à différentes méthodes de construction d'interfaces. Il ne s'agit pas d'être exhaustif, mais de montrer les efforts pour définir une interface la plus fidèle possible aux attentes de l'utilisateur et tout en restant le plus simple possible.

### 2.1.Les besoins.

Bien souvent, les systèmes ne nous semblent pas très performants alors qu'ils proposent de nombreuses fonctionnalités. Ce problème vient du fait que la communication entre l'homme et la machine n'est pas assez adaptée à l'homme : l'interface reflète plus ou moins la puissance de l'application. En effet, il arrive souvent qu'on renonce à utiliser un système aux détriments de l'application elle-même, à cause de sa mauvaise interface. On cherche donc à bien définir les besoins de l'interface, et surtout ceux requis pour les systèmes de CFAO.

#### *2.1.1.Une interface homme/homme améliorée.*

Nous nous plaçons du côté de l'utilisateur afin de voir comment l'interface doit fonctionner pour être la plus transparente possible. Pour parvenir à une telle analyse, [FOL 90] propose de comparer le

dialogue homme/machine au dialogue homme/homme : l'ordinateur n'est plus un outil, mais un collaborateur [COU 92]. On définit ainsi une liste des caractéristiques du dialogue traditionnel à préserver dans le dialogue homme/machine :

- le partage des connaissances qui se traduit par le principe d'"honnêteté" [ABO 92 ; COU 92 ]. L'interface montre à l'utilisateur tout changement d'état interne pertinent [FOU 94] ;
- la progression dans l'apprentissage de l'utilisation de l'interface ; si une personne veut communiquer avec un étranger, elle doit apprendre sa langue. Par analogie, l'interface peut inclure au départ des règles et un vocabulaire simples, voire des concepts que l'utilisateur connaît déjà pour le laisser se familiariser petit à petit avec le dialogue ;
- une syntaxe naturelle. On ne se préoccupe plus de la syntaxe, mais de la sémantique. L'interface ne doit pas distraire l'utilisateur avec des règles complexes de communication, mais l'aider en le dispensant de certaines tâches. Il s'agit du principe d'"égale opportunité" [ABO 92 ; COU 92], car il montre la faculté pour l'utilisateur de choisir la nature des entrées, le système se chargeant d'effectuer le complément en sortie en conséquence ;
- un langage efficace et complet, car une personne peut exprimer n'importe quelle idée et de manière concise. L'interface doit permettre à un utilisateur de traduire n'importe quelle idée et de transmettre ses commandes rapidement. On introduit le principe de "représentation multiple d'un concept" caractérisant la modularité de l'interface qui associe à chaque représentation un besoin particulier [ABO 92 ; COU 92]. Chaque individu a sa propre représentation d'une idée commune. L'interface doit donc aider le système à dialoguer avec tout type d'individu [ALL 93] ;
- l'extension du langage. L'interface doit fournir à l'utilisateur la possibilité de définir de nouveaux termes en fonction de l'existant (notion de termes génériques). Il s'agit de la dynamique de l'interface qui décrit l'évolution de l'apparence de l'interface au cours du temps ou en réponse aux sollicitations de son environnement [CHA 94c] ;
- la réaction ou "feed-back", car lorsque deux personnes dialoguent et que la réponse n'est pas immédiate, l'interlocuteur va tenter d'attirer l'attention de son auditeur (expressions du visage ...). L'interface ne doit pas être passive ;
- la discontinuité dans la communication. Une personne fait souvent des digressions dans son discours pour donner des explications ou se corriger. L'interface doit permettre à l'utilisateur d'interagir quand il le désire et d'annuler certaines de ses actions. On retrouve le principe de "non préemption" de l'interface qui détermine le caractère opportuniste du dialogue [ABO 92 ; COU 92]. L'utilisateur intervient quand il le désire et le système intervient quand c'est nécessaire.

Nous prenons pour base le dialogue traditionnel dans lequel nous avons de l'expérience. Notre but est de profiter de cette expérience et des possibilités de l'informatique pour proposer un dialogue adaptatif : on tente de simuler un dialogue homme/homme amélioré.

### *2.1.2.Des besoins orientés CFAO.*

Les systèmes de CFAO permettent à un utilisateur final de définir un prototype virtuel d'un produit avant de le fabriquer. Le résultat obtenu est souvent complexe, car les possibilités offertes par le

système sont immenses : l'originalité du résultat et sa complexité ne permettent pas de formaliser le processus complet de conception. Ainsi, pour parvenir à décrire ce prototype, le système met à la disposition de l'utilisateur une interface graphique interactive ainsi que des modèles informatiques de représentations pour gérer le prototype en construction ou final. En favorisant l'intervention de l'utilisateur final par un support graphique, le dialogue offre de nombreuses fonctionnalités, mais les besoins de l'interface deviennent plus rigoureux et contraignants :

- les modèles construits sont complexes, car ils décrivent des objets définis par un ensemble de contraintes de types très variés (contraintes géométriques, tolérances ...). L'interface est à la fois dépendante de ces modèles très particuliers (ce qui contraint sa spécification et sa gestion) et indépendante de ces modèles dans ses réponses avec l'utilisateur final (le dialogue garde une certaine liberté pour être suffisamment général pour convenir à tout type d'utilisateur) ;
- les dialogues sont nombreux, car les systèmes de CFAO manipulent un grand nombre d'objets. La difficulté s'accroît quand on sait que beaucoup d'informations sont données pour ces objets. L'interface doit être non seulement performante mais aussi résistante ;
- les dialogues sont très variés, car l'utilisateur final peut non seulement donner des réponses de type numérique, alphanumérique, mais également grapho-numérique. Une expression grapho-numérique associe à la fois des données numériques, alphanumériques et des données désignées graphiquement [GAR 91 ; STE 94] ;
- la grande liberté laissée à l'utilisateur final impose à l'interface de lever les éventuelles ambiguïtés, voire à capturer des informations à l'insu de l'utilisateur.

De manière plus générale, les interfaces des systèmes de CFAO s'intègrent dans la communauté des interfaces graphiques dans lesquelles les dialogues sont difficiles à concevoir et à gérer du fait de leurs grandes possibilités, parmi lesquelles on peut citer [HUB 89] :

- la diversité des périphériques graphiques d'entrée ;
- des techniques d'interactions très variées ;
- la grande complexité des structures de dialogues ;
- le contrôle dynamique sur la spécification du dialogue ;
- la gestion de fils d'activités multiples qui traduisent les différents chemins empruntés par l'utilisateur final pour dialoguer [COU 90] ;
- des possibilités de sorties puissantes et de gestion des fenêtres ;
- la gestion de l'entrée continue et de la réaction ("feed-back") ;
- la frontière délicate entre l'interface et l'application étant donné les recoupements.

Ces besoins montrent la complexité de décrire l'interface. De plus, il ne faut pas oublier que la liste n'est pas exhaustive et que la description de l'interface doit être suffisamment souple pour permettre de prendre en compte d'autres critères. De plus, trois types d'utilisateurs intervenant dans l'interface sont à considérer : le programmeur (ou concepteur) d'interfaces, qui décrit l'architecture du système sans utiliser un langage informatique, le programmeur d'applications sous-traitant du programmeur d'interfaces qui développe les modules informatiques et l'opérateur (ou utilisateur final), qui utilise le système à l'aide d'un dialogue proche de son langage métier (en particulier le dessin) [GAR 95b].

## 2.2.Les modèles d'architecture.

Une première approche pour spécifier une interface consiste à étudier globalement son fonctionnement vis-à-vis du système complet. Il s'agit de définir les liens entre l'interface et les autres composants dont, en particulier, l'application : le modèle obtenu constitue le modèle d'architecture. [CHA 93 ; MAR 95 ; PAN 93] présentent le modèle d'architecture comme un modèle abstrait capable de fournir une structure générique pour faciliter la conception et le développement des systèmes en identifiant les différents composants susceptibles d'être développés avec une relative indépendance. Les interfaces qui nous intéressent sont les interfaces interactives. La règle de base pour définir un modèle d'architecture mettant en scène un tel type d'interfaces est la séparation entre l'interface et le noyau fonctionnel i.e. l'application [CHA 94a]. Ne pas fondre l'interface dans l'application permet d'en garder un certain contrôle, ce qui simplifie sa modification et sa réutilisation dans d'autres applications [BAR 86].

Les deux modèles d'architecture les plus fréquemment présentés lorsque l'on aborde ce point sont le modèle Seeheim et le modèle PAC [CHA 94a ; COU 90 ; DIX 93]. Le modèle Seeheim (cf. figure I.1) comprend les composants Présentation, Contrôle de dialogue et Interface de l'application. La partie Présentation gère l'apparence de l'interface et inclut les entrées/sorties disponibles pour l'utilisateur ; l'Interface de l'application symbolise la sémantique de l'application ; le Contrôle de dialogue s'occupe de la communication entre les deux autres composants [CHA 94a ; DIX 93 ; WIE 95]. Ce modèle n'est pas censé définir le système interactif tout entier, mais seulement les composants. En conséquence, l'utilisateur et l'application n'y sont pas explicitement représentés. Non seulement ce modèle ne permet pas de répondre aux attentes des systèmes interactifs actuels (communication dense, interactivité ...), mais il ne précise pas la nature du contrôle de dialogue puisqu'il est très dépendant des deux autres composants (toute modification de l'un entraîne une modification de l'autre) [DIX 93 ; CHA 94a ; GAR 95b].

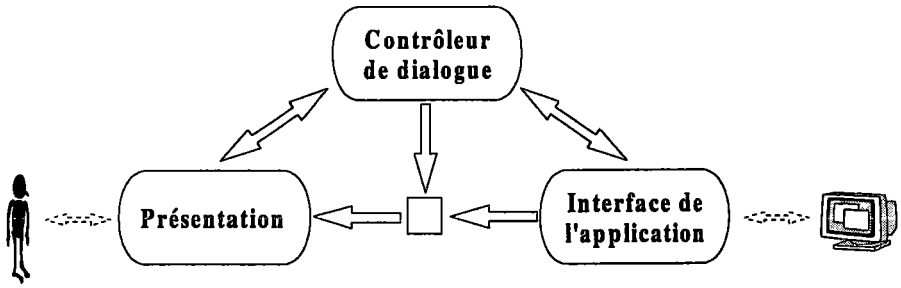



*Figure I.1. Le modèle Seeheim.*

Le modèle PAC, quant à lui, distribue les composants du modèle Seeheim au niveau d'agents chargés de gérer le fonctionnement de l'application [CHA 94a ; COU 90 ; DIX 93 ; HUB 89]. Chaque agent est composé d'une partie Présentation pour gérer les entrées/sorties, d'une partie Abstraction pour représenter la sémantique de l'application dont l'agent a la charge et d'une partie Contrôle pour faire la correspondance entre les deux premières parties (cf. figure I.2). Bien que ce modèle permette d'établir des dialogues en parallèle et facilite la modularité [CHA 94a], la structuration en objets PAC devient

coûteuse en raison du réseau de communication à gérer entre ces agents. Ce modèle représente plutôt une architecture conceptuelle car il est très peu dépendant d'un environnement de programmation [DIX 93].

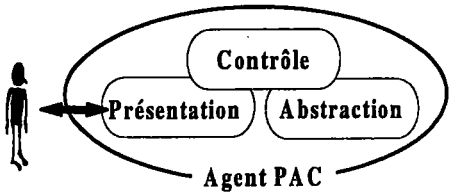



***Figure I.2.** Le modèle PAC.*

Notre but n'est pas de faire une étude complète des modèles d'architecture [CHA 94a; DIX 93], mais simplement de présenter des exemples de modèles pour attirer l'attention sur la difficulté de définir clairement l'interface par rapport à l'application. Dès ce niveau, on peut introduire des notions pertinentes pour garantir un meilleur fonctionnement de l'interface et obtenir un dialogue plus adaptatif. Il est important de bien définir où se situe la séparation entre l'interface et l'application : l'interface ainsi isolée, il est plus facile de se concentrer sur l'amélioration des éléments qui la composent et, en particulier, sur le choix du modèle de dialogue.

## 2.3.Les modèles de dialogue.

Le dialogue fait référence aux échanges d'instructions et d'informations qui se produisent entre l'utilisateur et le système [PRE 94]. À partir de cette considération, on définit un véritable modèle de dialogue qui définit la structure à mettre en place pour gérer ces échanges. Dans cette partie, avant de montrer un exemple de modèle, nous définissons les principaux constituants : les interactions.

### *2.3.1.Les interactions.*

Un modèle de dialogue peut être perçu comme une suite d'échanges entre l'utilisateur et le système. Ces échanges sont gérés par ce que l'on nomme des interactions [BAE 87 ; DIX 93]. Les interactions peuvent apporter des solutions pour répondre le plus fidèlement aux attentes de l'utilisateur. On distingue deux aspects de l'interaction sur lesquels on peut intervenir : le style d'interactions que nous développons dans la partie 2.3.2 et le modèle d'interactions. Le modèle d'interactions régit le fonctionnement de l'interaction. C'est par l'intermédiaire de ce modèle que l'on peut parvenir à préserver la liberté d'intervention de l'utilisateur i.e. le caractère opportuniste de son intervention et son raisonnement non rectiligne [COU 90].

Il existe un modèle d'interactions de référence, car il semble très proche de l'image que l'on a de l'interaction homme/machine : le modèle de Norman [NOR 88]. Il comprend des cycles interactifs qui débutent par une période d'exécution et qui se terminent par une période d'évaluation. De manière plus précise, chaque cycle interactif exécution/évaluation se décompose en sept étapes [DIX 93] : définir le but, former l'intention, spécifier la séquence d'actions, exécuter l'action, percevoir l'état du système, interpréter cet état et évaluer l'état du système en accord avec les buts et les intentions. Chaque étape est une activité de l'utilisateur. En fonction des réactions du système, face au plan d'actions que

l'utilisateur lui fournit de manière plus ou moins intuitive, l'utilisateur peut orienter son plan d'actions suivant.

On retrouve donc une certaine liberté dans l'interaction : on a plus l'impression que l'utilisateur guide le système que l'inverse. Ce modèle permet à l'utilisateur de suivre sa propre logique tout en respectant les impératifs du logiciel. En effet, l'utilisateur donne en quelque sorte ses instructions et le système tente de les satisfaire dans la mesure de ses possibilités. Si l'utilisateur n'est pas satisfait, il peut toujours reformuler son propos. Le système n'impose pas explicitement un plan d'actions comme dans les modèles de dialogue des systèmes classiques i.e. non interactifs. En effet, dans ces systèmes, le dialogue est orienté par la tâche, car des plans sont définis a priori [GRA 94]. Toutefois, le modèle de Norman manque de précisions en ce qui concerne le système lui-même, car il est décrit uniquement par son interface et par l'utilisateur : des extensions de ce modèle ont été développées [ABO 90].

Le modèle d'interactions prend donc toute son importance quand il s'agit de définir le fonctionnement de l'interaction i.e. comment le système inclut dans ses traitements l'intervention de l'utilisateur. Nous avons montré que de cette spécification dépend en partie la liberté d'intervention donnée à l'utilisateur.

### *2.3.2.Les styles d'interactions de base.*

Au delà du fonctionnement même d'une interaction, il convient de s'intéresser à un paramètre essentiel pour obtenir un dialogue adaptatif : le style d'interactions. Il s'agit de déterminer la manière avec laquelle l'utilisateur va pouvoir communiquer ou interagir avec le système [PRE 94].

Les auteurs qui abordent ce point semblent relativement cohérents entre eux sur la taxinomie des styles d'interactions. Ils distinguent entre cinq et six styles de base [DIX 93 ; PRE 94 ; SHN 95] : les menus, les formulaires, les dialogues question/réponse (style qui n'apparaît pas toujours dans les taxinomies), les langages de commande, le langage naturel et la manipulation directe.

Les menus sont des ensembles d'options affichées sur l'écran dont la sélection et l'exécution entraînent le changement d'état de l'interface [PAA 88]. De par la structuration qu'ils imposent sur les options, l'utilisateur se familiarise plus facilement à l'environnement qui lui est proposé et le système dirige les actions de l'utilisateur. Le nombre et la hiérarchie grandissante des menus peuvent entraîner de réels problèmes de compréhension et d'apprentissage pour l'utilisateur, et de gestion pour le système. Des techniques sont proposées pour pallier certains de ces problèmes (menus contextuels ...).

Les guides sont des styles d'interactions qui aident l'utilisateur à donner des informations en imposant la stratégie de capture de ces informations. Il s'agit des formulaires que l'utilisateur se contente de remplir et des dialogues question/réponse pour lesquels l'utilisateur ne fait que répondre à des questions posées par le système. La lourdeur du dialogue fourni va à l'encontre des dialogues adaptatifs ; elle montre, au contraire, la nécessité pour l'utilisateur de s'adapter.

Les langages de commande permettent à l'utilisateur de choisir ses actions en ayant recours à un langage spécialisé. Le système, maîtrisant parfaitement ce langage, répond facilement aux attentes de

l'utilisateur. Toutefois, ce langage nécessite un apprentissage de la part de l'utilisateur. Ainsi, le langage naturel peut être utilisé comme style d'interactions. L'utilisateur n'a plus besoin d'apprendre ou de se familiariser au style d'interactions. Toutefois, le système doit résoudre les ambiguïtés présentes dans le langage naturel.

La manipulation directe est un style d'interactions où tout objet sur lequel l'utilisateur peut interagir est représenté visuellement ; les opérations sont invoquées par des actions accomplies sur des représentations visuelles [FOL 90]. L'utilisateur voit ce qu'il fait ainsi que les conséquences de ses actions. Un autre atout est que l'utilisateur apprend plus vite grâce au support graphique [FRO 93]. Cette approche peut, malgré tout, paraître lente pour des utilisateurs expérimentés et elle ne simplifie pas toutes les opérations des utilisateurs (par exemple, les itérations sont difficiles à réaliser). De plus, le système est plus complexe étant donné les composantes graphiques à gérer.

On peut remarquer une évolution dans les styles d'interactions. En effet, on s'oriente vers des styles plus difficiles à gérer mais donnant moins de contraintes à l'utilisateur. Ceci est dû à la volonté de fournir à l'utilisateur des dialogues de plus en plus adaptatifs. Pour synthétiser, on distingue deux grandes catégories de styles d'interactions :

- les styles orientés système, car c'est l'utilisateur qui doit fournir des efforts et qui est restreint dans ses actions. Ils maintiennent l'utilisateur dans un mode de conduite stricte. On regroupe dans cette catégorie les menus, les guides et les langages de commandes ;
- les styles orientés utilisateur, car c'est le système qui doit s'adapter à l'utilisateur et par conséquent sa gestion en devient plus complexe. Ils offrent plus de liberté à l'utilisateur. Le langage naturel et la manipulation directe constituent des styles représentatifs de cette catégorie.

Nous ne préconisons pas un style d'interactions particulier, car il est clair que, suivant le contexte, il n'est pas utile de requérir des méthodes intuitives comme la manipulation directe. On surchargerait le système inutilement, ce qui entraînerait une baisse de performance certaine : un dialogue trop intuitif ne sert à rien si les temps de réponse du système sont inacceptables. Nous avons voulu montré que l'on peut fournir des efforts sur les styles mêmes d'interactions pour améliorer les dialogues.

### *2.3.3. Un exemple de modèle de dialogue.*

Dans les modèles de dialogue classiques, on rencontre des problèmes de flexibilité, d'extension et de facilité d'utilisation dus à la programmation procédurale et à la structuration par couches traditionnelles (lexical, syntaxique et sémantique). Le modèle IAMBUS présenté par [HUB 89] est un modèle orienté objets qui considère différents niveaux de descriptions pour chaque objet afin de répondre à ces problèmes. Il contient des classes d'interactions de base pouvant servir à l'élaboration de classes plus complexes. Une interaction comprend les éléments suivants :

- un déclencheur, qui définit ce que l'utilisateur doit faire pour exécuter une interaction ;
- une invitation, qui prévient graphiquement que le système attend une interaction ;
- une réaction, qui prévient graphiquement que l'utilisateur vient de déclencher une interaction ;

- une partie sémantique, qui définit l'interface pour l'application ;
- une partie dynamique, qui modifie l'interface pendant que l'utilisateur communique avec le système.

Le modèle de dialogue est constitué d'un graphe topologique montrant le réseau formé par toutes les interactions. Il contient les relations qui existent entre les différentes interactions et contrôle ainsi les différents fils d'activités de l'utilisateur i.e. les différents raisonnements qu'il peut suivre.

### 2.4.Les outils de construction.

Nous avons vu les différents composants pour définir un dialogue et surtout comment utiliser ces composants pour obtenir un dialogue toujours plus proche de l'utilisateur. Le concepteur d'interfaces est chargé de définir l'interface d'une application en accord avec les principes que nous avons évoqués précédemment. Devant la complexité des interfaces à définir, il convient de fournir des outils adaptés permettant au concepteur de transmettre ses connaissances : le concepteur est à son tour un utilisateur à part entière. Les apports de tels outils ne sont plus à prouver [CHA 94a ; MYE 93a]. Par contre, il est intéressant de montrer les différentes techniques mises en œuvre, car :

- elles sont similaires à celles utilisées pour capturer les intentions de l'utilisateur final ; les outils possèdent leur propre interface. Pour garantir que le dialogue décrit est correct avec la description qui en a été faite, il faut que l'outil de construction fournisse une interface de haut niveau pour capturer les intentions du concepteur d'interfaces ;
- elles sont déterminantes pour favoriser la construction d'un dialogue adaptatif.

Dans les trois parties suivantes, nous présentons les grandes catégories de techniques utilisées dans les outils de descriptions d'interfaces.

#### *2.4.1.Les langages.*

Parmi les outils utilisés pour développer une interface, l'utilisation d'un langage est le plus ancien [MYE 93a]. Le choix d'un langage est déterminant puisque chacun apporte des contributions spécifiques [GRA 92b].

Le langage des réseaux à transition d'états est constitué d'états reliés par des arcs orientés indiquant la transition d'un état à l'autre. Cette technique est peu adaptée pour représenter l'interactivité des dialogues qui nous préoccupent : on imagine aisément la complexité du réseau. À noter que les grammaires hors contexte (grammaires basées sur des générateurs d'analyseurs lexical et syntaxique) posent le même problème, car elles correspondent plutôt à des traitements séquentiels [MYE 93a].

Les langages déclaratifs et les langages à événements tentent d'apporter des réponses pour prendre en compte cette interactivité. Tandis que les langages déclaratifs permettent d'alléger la charge du concepteur qui ne décrit que les objets et relations à obtenir, les langages à événements prennent en compte le caractère opportuniste de l'utilisateur. L'inconvénient des langages déclaratifs est qu'ils ne

permettent pas encore de décrire n'importe quel type d'interfaces [MYE 93a]. Quant aux langages à événements, le problème réside dans la difficulté d'établir un réel contrôle.

Les langages de contraintes sont importants, car ils procurent un moyen relativement naturel pour représenter la connaissance contextuelle. Les contraintes sont essentielles dans la construction d'interfaces pour maintenir la consistance non seulement entre les données de base et leur description graphique, mais aussi entre plusieurs visualisations d'une même donnée [BOR 86]. De plus, elles spécifient les événements sur l'animation qui doit se produire suite au déclenchement d'un événement, la manière dont l'information doit être transmise ainsi que les attributs des différents objets dans une animation. Les contraintes ne sont pas forcément statiques, elles peuvent être temporelles, ce qui permet de considérer des actions futures et de décrire une certaine évolution dans le comportement des objets décrits [BOR 86 ; GRA 92a]. Le système GROW est un exemple d'environnements pour développer des interfaces graphiques : il est basé sur la définition d'objets d'interfaces et des contraintes qui les lient [BAR 86]. Les langages de contraintes sont importants, mais la gestion des contraintes décrites est difficile si l'on considère les nombreux choix que le système doit faire pour les résoudre.

### *2.4.2.Les spécifications graphiques.*

L'aspect visuel apporté par des spécifications graphiques permet de se rapprocher du système humain et rend accessible la spécification de l'interface à des non programmeurs. La programmation est réduite et une description basée sur la combinaison de composants est privilégiée [SUK 94]. [MYE 90a] distingue deux approches : la programmation visuelle utilisant des graphiques comme langage et la visualisation de programmes utilisant des graphiques pour illustrer des programmes déjà créés.

Ces concepts se retrouvent dans les outils basés sur la manipulation directe, style d'interactions présenté précédemment [CHA94a ; FOL 90 ; FRO 93]. Comme exemple d'outils de construction d'interfaces utilisant cette technique, nous pouvons évoquer le système Garnet [MYE 90b]. Ce système comprend des outils permettant de décrire des interactions par combinaisons d'interactions de base (les interacteurs) : le concepteur décrit graphiquement son interface et le système interprète les manipulations effectuées, voire déduit des informations (dépendances ...).

### *2.4.3.La génération.*

Fournir des outils graphiques pour décrire les interfaces ne suffit pas : le concepteur se disperse et perd du temps dans sa description. Ainsi, on s'oriente de plus en plus vers une génération des principaux éléments de l'interface. Le concepteur spécifie ce qui le concerne i.e. des connaissances de haut niveau et le système produit automatiquement les composants de bas niveau [JAN 93]. Le concepteur peut se concentrer sur la formulation des connaissances qu'il veut transmettre, ce qui augmente les chances de réussir à décrire un dialogue adaptatif.

Le système UIDE permet la construction automatique d'une interface par la spécification des actions à l'aide de pré conditions et de post conditions [FRA 93 ; SUK 93]. Il utilise un modèle d'application

pour créer automatiquement une interface. Tout d'abord, le concepteur décrit les objets et les actions dans l'application ; il choisit les divers composants fonctionnels d'interfaces correspondant le mieux à l'application et les associe aux actions de l'application. UIDE utilise ces spécifications du modèle pour déduire une interface. La spécification sert aussi à produire de l'aide.

Le problème que l'on rencontre dans la plupart des systèmes qui déduisent l'interface, parmi lesquels le système UIDE, est le manque de standardisation dans la description [JAN 93]. En effet, chacun introduit ses propres notations pour la description de haut niveau.

### 2.5.Synthèse.

Nous avons évoqué le besoin de créer des interfaces répondant aux attentes de l'utilisateur. Le but est d'obtenir des dialogues adaptatifs dans lesquels l'utilisateur intervient librement et accède facilement aux données qu'il peut manipuler avec l'aide éventuelle du système.

Ce phénomène entraîne le développement d'interfaces très complexes. Pour parvenir à de telles interfaces, une structuration hiérarchique de modèles doit désormais être considérée : modèle d'architecture, modèle de dialogue et modèle d'interactions. Nous avons montré que le choix même de ces modèles est capital pour l'obtention d'un dialogue adaptatif. Des connaissances précises sur le fonctionnement de l'interface sont à transmettre pour décrire le plus fidèlement possible ces modèles.

Devant cette complexité, le concepteur d'interfaces ne peut à lui seul décrire convenablement l'interface de son application. Ainsi, des outils destinés à lui simplifier sa tâche sont introduits. Une nouvelle difficulté apparaît : trouver les outils capables de capturer sa connaissance sans trop de pertes.

De plus, la tendance est de créer des interfaces réactives dans lesquelles la manipulation directe prend toute son importance. En conséquence, l'utilisateur doit tout faire et le système perd de son importance. Il semble plus intéressant de se diriger vers des interfaces d'anticipation dans lesquelles l'utilisateur délègue des tâches au système et le système prend des initiatives pour les réaliser [FAR 93]. L'utilisateur n'est plus seul à agir, car le système coopère de manière intelligente (anticipations ...).

Dès la conception de l'interface, on tente de fournir des techniques capables d'obtenir des informations sur l'utilisateur afin de créer un dialogue adaptatif. On se dirige vers une prise en compte plus importante des connaissances des utilisateurs qu'ils soient concepteurs ou opérateurs. Cette constatation nous amène dans la suite à situer le rôle de l'intelligence artificielle dans l'interface.

## 3. L'INTERFACE HOMME/MACHINE ET L'INTELLIGENCE ARTIFICIELLE.

L'IA (Intelligence Artificielle) intègre l'étude des techniques employées pour doter l'ordinateur de capacités habituellement attribuées à l'intelligence humaine : acquisition de connaissances, perception, raisonnement, prise de décisions ... [HAT 89]. Devant les besoins des interfaces que nous avons évoqués précédemment (cf. 2.1. Les besoins) et les notions de connaissances utiles à l'amélioration de l'interface (cf. 2.5. Synthèse), on conçoit que l'IA soit impliquée dans la spécification des interfaces.

Ainsi, nous présentons dans cette partie les éléments qui nous semblent intéressants d'utiliser pour contribuer à la réalisation d'un dialogue homme/machine adaptatif. Dans une première partie, nous précisons quels types de connaissances l'IA préconise de capturer et nous montrons des modes de représentations. Ensuite, nous présentons trois sujets traités en IA et qui peuvent s'intégrer dans le programme d'assistance à l'opérateur à inclure lors de la description d'une interface : le système doit pouvoir apprendre, anticiper et expliquer. Enfin, nous présentons un concept important de l'IA distribuée, les systèmes multi-agents, car il est très souvent utilisé dans les interfaces aussi bien pour décrire l'architecture que le dialogue lui-même.

### 3.1.Une notion de connaissance.

La tendance actuelle en ce qui concerne les interfaces est de ne faire appel à l'utilisateur (tant le concepteur d'interfaces que l'utilisateur final) que pour des décisions de haut niveau [TOL 92]. Ce critère impose au système de disposer des connaissances nécessaires pour intervenir. On peut faire appel à l'IA pour résoudre ce problème. En effet, la représentation des connaissances est un élément-clé dans cette discipline qui traite des systèmes à base de connaissances.

Les systèmes à base de connaissances permettent d'améliorer le raisonnement, de rendre l'application plus souple et d'humaniser le système. Leur rôle est de fournir une aide pour l'opérateur, une aide à la décision pour des systèmes complexes, à l'implémentation et à l'exécution des spécifications, ainsi que de fournir des générateurs, des critiques et des évaluations de structures (plan d'actions ...) [HAY 94].

L'intérêt de ces systèmes réside dans leur capacité à tolérer des erreurs et des incertitudes dans les données et de pouvoir raisonner à différents niveaux d'abstraction [HAY 94]. C'est bien ce que l'on recherche dans le dialogue homme/machine, car non seulement on veut que la machine ait les qualités d'un être humain, mais on veut aussi que l'homme puisse agir en tant que tel (être libre d'agir selon son mode de pensée). On imagine donc l'importance de l'IA dans les interfaces homme/machine. Avant de montrer comment est représentée la connaissance dans les systèmes d'IA, nous précisons quelle est cette connaissance que l'on peut capturer. Enfin, nous évoquons les techniques utilisées pour acquérir cette connaissance.

#### *3.1.1.La connaissance requise.*

[HAY 94] définit la connaissance par toutes les données qui améliorent la résolution de problèmes. Il convient de bien déterminer ce qu'est la connaissance avant de l'implémenter [YEN 93]. Ce n'est pas une tâche facile si l'on considère les multiples classifications qui existent à ce sujet [CHO 91b ; FAR 88 ; HAY 94 ; LIA 91 ; QUE 94 ; TOL 92]. Toutefois, on peut déceler certaines tendances.

La première tendance consiste à définir le type de connaissances par rapport au système et au problème que la connaissance est censée résoudre. Dans, on retrouve très nettement cette analyse dans [BRO 84 ; CHO 91b ; HAY 94 ; TOL 92] ; trois types de connaissances sont mis en valeur :

- la connaissance sur le problème posé (cahier des charges) [BRO 84 ; TOL 92]. [HAY 94] nomme ce type de connaissances les faits, car il représente l'ensemble des propositions valides. De la

même manière, [CHO 91b] définit un niveau descriptif qui regroupe toutes les informations pour amorcer le raisonnement. Bien que l'appellation soit différente selon les auteurs, il est clair que ces connaissances évoquées sont identiques, car elles regroupent les connaissances de base ;

- la connaissance technique et scientifique du domaine de conception [BRO 84 ; TOL 92]. [HAY 94] regroupe les croyances qui expriment des propositions plausibles. C'est une manière plus générale de définir ce que l'on sait sur l'environnement de conception. [CHO 91b] définit un niveau de connaissances d'interprétation qui spécifie les informations de base interprétées en fonction de l'objectif. Ce sont donc également des connaissances dédiées à l'environnement ;
- la connaissance sur la stratégie de conception [BRO 84 ; TOL 92]. [HAY 94] intègre les connaissances heuristiques. Une heuristique peut être vue comme une méthode qui permet d'appliquer un jugement dans des situations pour lesquelles il n'existe généralement pas d'algorithme valide [HAY 94] ou encore comme une technique de résolution de problèmes dans laquelle la solution la plus appropriée est choisie en utilisant des règles [MYE 93b]. Ce type de connaissances permet de définir des stratégies. De même, [CHO 91b] donne comme dernier niveau de connaissances, le niveau de synthèse. Il représente les connaissances permettant d'aboutir au problème ou les connaissances montrant la solution du problème.

Parallèlement à la taxinomie donnée par [BRO 84 ; TOL 92], on constate que les classifications de [CHO 91b ; HAY 94] définissent trois types de connaissances apparemment différents de par leur vocabulaire, mais similaires sur le fond.

La seconde tendance importante sur la classification de la connaissance provient de la nécessité d'incorporer l'expertise humaine [NGU 94]. De cette constatation apparaissent deux types de connaissances : les connaissances profondes qui concernent les connaissances structurelles, fonctionnelles et théoriques du domaine, et les connaissances de surface qui réunissent les connaissances compilées, empiriques, heuristiques ... i.e. le savoir-faire [CHO 91b ; FAR 88 ; LIA 91]. La connaissance de l'expert est essentielle dans les systèmes à base de connaissances. Elle permet de résoudre des problèmes difficiles [HAY 94] : regrouper des faits significatifs, éviter les erreurs courantes, faire des distinctions entre les types de problèmes, éliminer les recherches inutiles, ordonner la recherche, supprimer la redondance et les informations inutiles, réduire les ambiguïtés, exploiter la connaissance de disciplines complémentaires et analyser les problèmes à partir de différentes perspectives ou niveaux d'abstraction.

Il existe d'autres classifications de la connaissance, mais elles sont ponctuelles car elles sont souvent dédiées au traitement. Par exemple, [LEM 92] propose d'isoler la connaissance utilisée pour fournir des explications et de décomposer cette connaissance. On obtient une classification des connaissances qui s'oriente vers les explications car l'application décrite ne se préoccupe que de ces connaissances.

### *3.1.2.La représentation de la connaissance.*

Une fois la connaissance requise définie, il ne s'agit pas de la laisser "envahir" le système n'importe comment. Un système à base de connaissances centralise la connaissance dans un module particulier

qui est dans une large mesure indépendant du reste du système [POM 88]. Ce module définit un mode particulier de représentation de la connaissance. Il existe différentes représentations. Nous en présentons succinctement quelques unes.

Les règles de production constituent la représentation la plus courante. Ce sont des règles de la forme : Si *condition* alors *action*. Comme exemple d'utilisation, nous pouvons citer le système de formation multi-tuteurs présenté par [CHO 91a]. Les règles prennent un caractère impératif si elles indiquent au système comment se comporter, ou plutôt déclaratif si elles décrivent la manière dont les choses fonctionnent sans préciser ce qu'il faut faire [HAY 94]. Ce mode de représentation a bon nombre d'avantages [FAR 88] :

- la concision et la clarté de la règle ;
- la modification est relativement aisée dans la mesure où modifier une règle consiste à modifier automatiquement toutes les chaînes de raisonnement susceptibles de l'utiliser ;
- la détection rapide d'erreurs, car elles sont répercutées dans plusieurs raisonnements ;
- la capacité des systèmes à justifier leur déduction.

Cette représentation n'est toutefois pas idyllique. On imagine parfaitement le problème de concurrence entre les règles dans le cas de systèmes complexes [HAT 89]. De plus, on ne peut représenter les relations invariantes i.e. les connaissances profondes ; ce sont les connaissances de surface qui sont ainsi représentées. [LIA 91] définit différents types de règles selon la connaissance : des règles de début, de propagation et de fin de diagnostic pour les connaissances sur la stratégie de diagnostic et des règles d'orientation du diagnostic et de déduction de nouveaux faits pour les connaissances d'expertises.

Pour représenter les connaissances profondes, les réseaux sémantiques peuvent être utilisés [FAR 88]. Un réseau sémantique est un graphe étiqueté où les nœuds représentent des concepts et les arcs des relations de nature sémantique entre les concepts [HAT 89]. Il permet de traduire les notions d'héritage et de généralisation ainsi que les relations temporelles, causales, spatiales et temporelles [FAR 88].

Il existe d'autres modes de représentation parmi lesquels les objets qui permettent de représenter la connaissance d'éléments physiques ou conceptuels [HAY94], les "frames" qui sont des réseaux sémantiques avec une notion de classes, sous classes et instances, et les représentations logiques. L'IA fournit donc de grandes possibilités en matière de représentation des connaissances si bien qu'on les exploite dans la gestion des interfaces. Par exemple, dans le modèle de dialogue IAMBUS [HUB 89] présenté au 2.3.3., les interactions sont représentées sous forme d'objets.

### *3.1.3.L'acquisition de la connaissance.*

On sait désormais quelles connaissances doivent être prises en compte dans le système et comment les représenter. Bien que l'on puisse modéliser certaines connaissances par une simple analyse du but à atteindre [SAU 94], il n'en est rien pour les connaissances de surface faisant appel au savoir-faire. Tout le problème réside dans l'acquisition de ce savoir-faire. Les systèmes experts ont été introduits pour

rendre cette connaissance disponible : ils intègrent donc des techniques d'acquisition. Ce sont des systèmes à base de connaissances dans lesquels on retrouve des connaissances générales, informatiques et des connaissances d'expertise [CHO 91b ; POM 88].

Lorsqu'un dialogue s'engage, le système doit jouer ce rôle d'expert qui a toutes les connaissances nécessaires pour guider l'utilisateur vers la bonne solution. Ainsi, l'interface doit intégrer des connaissances tant de surface que profondes. Notons que ces connaissances sont aussi bien issues du concepteur d'interfaces, que de l'utilisateur final. Ces deux individus forment les experts réels qui vont permettre à l'interface de s'enrichir de connaissances sur leur comportement et leur savoir-faire.

[CHO 91b] considère qu'il existe deux grandes catégories de méthodes d'acquisition : le prototypage rapide et les méthodes analytiques. Le prototypage rapide traduit la connaissance sans la comprendre : on ne peut pas tout représenter et il n'y a pas de réelle structuration. Les méthodes analytiques capturent la connaissance puis la structurent avant de la traduire dans un langage informatique : ces méthodes sont plus intéressantes que le prototypage, car elles essaient de comprendre la connaissance. En principe, les méthodes analytiques font intervenir un cogniticien chargé des outils de modélisation des connaissances [CHO 91b ; FAR 88].

À titre d'exemple, citons une méthode d'acquisition et de représentation des connaissances dédiée aux applications de CFAO pour la conception de produits mécaniques : DDAM (Deklare Design Analysis Methodology) [SAU 94]. La modélisation du produit est décomposée en différents modèles spécifiques pour lesquels l'acquisition de connaissances est particulière : le modèle physique (les objets, les assemblages ...) est obtenu par des dessins, de la documentation technique et des documents d'expertise alors que le modèle fonctionnel (concepts, solutions techniques, ...) est obtenu grâce à un cogniticien qui se réfère à des documents sur le problème de conception. Il paraît difficile de garantir que le cogniticien appréhende correctement la description du modèle, puisqu'il se base sur des documents qu'il interprète à sa manière.

Sans doute pour répondre à ce problème, [CHO 91b] propose un "pré-captage" de la connaissance avant son acquisition effective et sa structuration. Cette étape consiste à coordonner le cogniticien et l'expert :

- localisation de la connaissance, car le cogniticien réagira différemment selon, par exemple, la provenance des informations (expert humain ou livre) ;
- choix du langage d'expression de l'expertise, car il ne s'agit pas de perdre des connaissances suite à une mauvaise interprétation ;
- familiarisation au domaine de l'expertise, car le but est de créer un dictionnaire de termes communs à l'expert et au cogniticien.

La méthode d'acquisition consiste ensuite à acquérir la connaissance de l'expert par niveau d'abstraction : ceci permet de capturer sa connaissance en cours de raisonnement. Cette méthode d'acquisition permet de décomposer au maximum la connaissance étant donné la structuration incrémentale préconisée, ce qui permet une grande souplesse pour gérer cette connaissance.

L'inconvénient reste dans cette décomposition de la connaissance qui impose à l'expert de déterminer d'une part, l'interprétation qu'il fait de ses données de bas niveau et d'autre part, les paramètres de synthèse qu'il utilise au cours de son raisonnement [CHO 91b].

### 3.2.L'assistance à l'utilisateur.

Nous avons montré que l'IA se préoccupe de modéliser la connaissance, et en particulier, celle qui est moins conventionnelle : le savoir-faire. On dispose ainsi aisément d'un modèle embarqué de l'utilisateur i.e. d'une représentation explicite et dynamique de tous les aspects de l'utilisateur pouvant être utiles au comportement adaptatif du système [COU 92]. En conséquence, le domaine de l'IA ne s'arrête pas simplement à l'étude de la connaissance, mais il est capable d'étudier également des techniques basées sur ces connaissances pour aider l'utilisateur. Dans les trois parties suivantes, nous présentons de telles techniques mettant en valeur la capacité du système à apprendre, à détecter et à expliquer.

#### *3.2.1.L'apprentissage.*

Les utilisateurs évoluent constamment dans leurs actions : ils émettent des restrictions, des adaptations ou des élargissements de la connaissance de base et ceci de manière opportuniste [QUE 94]. Il semble important de disposer de mécanismes pour mettre à jour les connaissances. De plus, si le système comprend les actes de l'utilisateur, il peut l'aider dans ses choix. Or, pour réussir à les comprendre, il faut d'abord qu'il apprenne à les reconnaître [BOU 94]. Des mécanismes d'apprentissage sont introduits pour améliorer, voire déduire les connaissances dont le système a besoin [LIA 91] : le système montre une certaine capacité à raisonner.

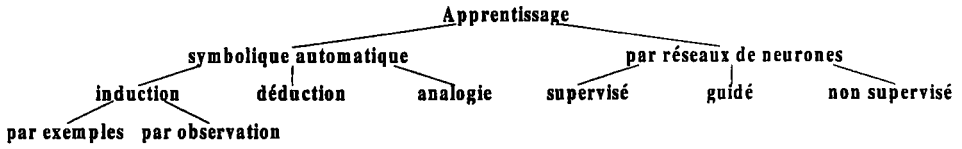



***Figure I.3.*** *Les méthodes d'apprentissage.*

Il existe différentes méthodes d'apprentissage (cf. figure I.3) que l'on peut regrouper en deux grandes classes : l'apprentissage symbolique automatique et l'apprentissage par réseaux de neurones. La classification que nous donnons est une synthèse des diverses classifications rencontrées dans [BON 84 ; BOU 94 ; LIA 91]. L'apprentissage symbolique automatique comprend les apprentissages :

- par induction, qui consiste à acquérir des concepts à partir d'exemples ou de contre-exemples ou à affiner des connaissances acquises en les observant plus attentivement. [BOU 94] présente une méthode d'apprentissage basée sur l'exemple pour simplifier à un utilisateur la description d'une scène (ensemble d'objets graphiques formant le monde manipulé par l'utilisateur). Son but est d'obtenir par apprentissage des filtres pour proposer à l'utilisateur les scènes les plus probables. Pour y parvenir, l'utilisateur fournit des exemples à partir desquels le système d'apprentissage va tenter, par réduction du niveau de détail, de définir les critères de l'utilisateur ;
- par déduction, qui définit de nouvelles connaissances à partir de connaissances existantes ;

- par analogie, qui adapte des connaissances issues de plusieurs domaines pour les appliquer à un domaine. [QUE 94] propose un mécanisme d'apprentissage de généralisation en temps réel par analogie.

La seconde classe d'apprentissage regroupe les techniques d'apprentissage utilisées lorsque le système est représenté sous forme de réseaux de neurones. Un réseau de neurones est un ensemble de nœuds connectés par des liens [LIA 91]. Leur fonctionnement est censé imiter celui du cerveau humain. Ainsi, des règles d'apprentissage sont intégrées pour influencer les poids de connexions qui régissent le bon fonctionnement d'un nœud. Trois types d'apprentissage peuvent intervenir :

- supervisé, si en phase d'apprentissage le réseau reçoit à la fois l'entrée et la sortie désirée de façon à pouvoir calculer la différence entre la sortie effective et la sortie désirée ;
- guidé, si on indique juste si la sortie produite est correcte ou non, ce qui induit une variation sélective des poids des connexions ;
- non supervisé, s'il y a détection automatique des régularités dans les exemples donnés et modification en conséquence des poids des connexions pour que les exemples ayant les mêmes régularités fournissent le même résultat.

Ces deux classes d'apprentissage (symbolique automatique et par réseaux de neurones) ne sont pas exclusives. On peut cumuler différents apprentissages étant donné les apports différents : dans le système de [BOU 94], l'utilisation d'un apprentissage symbolique automatique par l'exemple permet de réduire les solutions proposées à l'utilisateur et l'utilisation d'un apprentissage par réseaux de neurones entraîne la réduction du nombre d'essais à effectuer.

[BON 84] considère deux autres styles d'apprentissage que nous n'avons pas évoqués dans cette classification, car nous ne considérons pas qu'il s'agisse de véritables méthodes d'apprentissage. Le premier style est l'apprentissage par cœur et implantation directe des nouvelles connaissances. Le système se contente de recevoir de nouvelles connaissances et ne les modifie pas. Le second style est l'apprentissage par instruction dans lequel le système acquiert de la connaissance venant de l'extérieur (interactivement par exemple) et l'intègre dans son raisonnement. Ces deux modes d'apprentissage ne montrent aucune capacité du système à raisonner pour acquérir de nouvelles connaissances.

#### *3.2.2.La détection.*

Pouvoir détecter ce que fait l'utilisateur est un atout pour un système : le dialogue n'est plus figé, il réagit en fonction de l'utilisateur. Pour y parvenir, de nombreux stratagèmes issus de l'IA sont utilisés. Nous voulons attirer l'attention sur deux aspects qui nous semblent importants : l'utilisation d'heuristiques (définies au *3.1.1. La connaissance requise*) et l'analyse du comportement.

Les méthodes heuristiques procurent un gain de temps considérable pour l'utilisateur qui n'a pas besoin de tout spécifier. Dans les systèmes de CFAO, l'utilisateur doit souvent définir des contraintes sur les produits qu'il décrit. Bien souvent maintenant, cette tâche est accompagnée d'un processus d'anticipations régi par des heuristiques [ALP 93 ; AND 95 ; ROL 91]. Toutefois, il ne faut pas en

négliger les risques [MYE 93b] : ne pas satisfaire aux attentes de l'utilisateur, manquer de clarté vis-à-vis de l'utilisateur et paraître imprévisible et incontrôlable. Si elles sont correctement définies, les heuristiques constituent un élément de base dans l'optimisation du comportement du système.

L'analyse du comportement permet de détecter les intentions de l'utilisateur et de réagir en conséquence pour améliorer les conditions de l'utilisateur. Nous présentons le module de dialogue dédié à cette analyse proposé dans [GRA 94]. Trois modèles internes au dialogue sont définis :

- modèle d'interprétation thématique, qui compare les chemins empruntés par l'utilisateur à un schéma directeur. Le système indique si l'utilisateur se trouve dans un cas de confirmation, de généralisation ou spécification, de déviation ou de changement de thème. Ce modèle permet de suivre et comprendre les fils d'activités de l'utilisateur ;
- modèle d'analyse intentionnelle, qui définit, à partir du thème abordé, une représentation du dialogue et déduit les actes sous-jacents (affirmation, interrogation ...) ;
- modèle de gestion de l'interaction, qui détecte des erreurs (échec de l'analyse intentionnelle ou impossibilité d'intégrer le message dans un modèle cohérent du discours) et les gère.

Bien que cette méthode analyse un dialogue proche du discours, elle repose sur des bases suffisamment générales pour être interprétée dans le cas d'interfaces graphiques interactives. Toutefois, des difficultés supplémentaires apparaissent surtout pour l'analyse intentionnelle. Il semble plus difficile de déterminer les actes sous-jacents du langage étant donné les différents styles d'interactions qui sont disponibles dans ce type d'interfaces.

#### *3.2.3.L'explication.*

La capacité à fournir des explications est un atout pour le système dans sa gestion du dialogue avec l'utilisateur. Les explications constituent même un élément de base dans les systèmes à base de connaissances [HAY 94]. Parmi les différents sens du mot explication en IA [LEM 92], on retient les explications pour le concepteur et celles pour l'utilisateur final. La distinction qui est faite entre les explications pour ces deux intervenants n'est pas aussi nette : il s'agit de donner des explications à un utilisateur pour comprendre et corriger les réactions du système face à ses interactions.

Le problème des explications à l'utilisateur se décompose en deux temps : la conception de l'explication, qui comprend le mode de déclenchement, et la transmission de l'explication [LEM 92]. Il convient de définir des stratégies différentes d'explications et des critères de choix de ces stratégies. Remarquons qu'une technique telle que l'analyse des comportements que nous avons évoquée dans la partie précédente peut être l'initiatrice de l'explication : le modèle de gestion de l'interaction détecte des dialogues mal engagés qui nécessitent par conséquent des explications.

### 3.3.Les systèmes multi-agents.

Devant la complexité et la multitude de traitements à effectuer dans les systèmes actuels, il est souvent recommandé de décomposer les tâches à effectuer. On retrouve de plus en plus de systèmes qui utilisent des architectures multi-agents pour répondre à ce problème. De manière plus générale, ces

architectures reposent sur les concepts des systèmes multi-agents : on entre dans le domaine de l'IAD (Intelligence artificielle distribuée) qui consiste à développer des systèmes intelligents en distribuant l'intelligence (ou la compétence) parmi une communauté d'agents intelligents automatisés semi-autonomes [OHA 96]. Dans la première partie, nous définissons le concept d'agent et de système multi-agents. Ensuite, nous présentons la communication entre agents qui est l'un des paramètres essentiels de ces systèmes. Il s'agit de montrer les éléments exploitables pour la communication qui s'établit dans le dialogue homme/machine.

### *3.3.1.Définition.*

Un SMA (Système Multi-Agents) est un système composé des éléments suivants [FER 95] :

- un environnement *E* disposant en général d'une métrique ;
- un ensemble d'objets *O* que l'on sait localiser ; ces objets sont passifs ;
- un ensemble d'agents *A* qui représentent les entités actives du système ;
- un ensemble de relations *R* qui unissent les objets et les agents entre eux ;
- un ensemble d'opérations *Op* des agents *A* sur les objets *O* (produire, transformer, manipuler...) ;
- les lois de l'univers qui traduisent les applications des opérations et les réactions.

Un agent est souvent physiquement et logiquement distinct et capable de raisonner, planifier, communiquer et coopérer [HER 88]. Il s'agit d'une sorte "d'organisme vivant" dont le comportement, qui se résume à communiquer, à agir et, éventuellement, à se reproduire, vise à la satisfaction de ses besoins à partir de tous les autres éléments (perceptions, représentations, actions, communications et ressources) dont il dispose [FER 95].

Pour analyser les besoins d'un SMA, on distingue les différents types d'agents qui interviennent. Ces types sont définis en fonction de la spécialité de chaque agent : un agent a un but et pour y arriver il a recours à une catégorie déterminée d'actions. Le deuxième point à considérer concerne le fonctionnement général des agents : leurs capacités cognitives (leur base de connaissances, ce qu'ils sont capables de réaliser...) et leur position face au travail collectif (travaille seul ou non, avec équipes fixes ou dynamiques...). Ensuite, on se préoccupe plus particulièrement des interactions entre agents. Enfin, on peut étudier l'évolution que peut prendre le SMA par les stratégies mises en œuvre.

### *3.3.2.La communication.*

La particularité de l'approche multi-agents est qu'une grande priorité est donnée à la capacité de communiquer i.e. aux interactions entre agents : les agents sont plus importants pour ce qu'ils font que pour ce qu'ils sont. Notons que les architectures multi-agents que l'on retrouve dans les modèles d'architecture comme le modèle PAC (cf. 2.2 Les modèles d'architecture) sont utilisés avant tout pour leur modularité et leurs interactions.

Pour les systèmes multi-agents, l'interaction est composée de trois points : la nature du but, de l'accès aux ressources et de la compétence des agents. Le premier point concerne les buts des agents. Connaître leurs buts permet d'anticiper et de faciliter leur communication en déterminant dans quelles

situations ils se retrouvent s'ils communiquent : coopération, indifférence, antagonisme. On déclenche ainsi des mécanismes de communication adaptés.

Le deuxième point important de l'interaction concerne les ressources nécessaires à un agent pour agir. Lorsqu'il va chercher à les obtenir, il se peut qu'il ait besoin de communiquer avec d'autres agents. Il s'agit du cas où des ressources sont communes à plusieurs agents : interviennent alors des mécanismes de coordination d'actions et de résolution de conflits. L'analyse des ressources prépare et aide les dialogues qui vont s'engager entre les agents.

Le dernier point de l'interaction concerne la compétence des agents. Il est utile de connaître le fonctionnement d'un agent. Selon sa compétence, l'agent va plus ou moins communiquer avec d'autres agents pour agir. Par exemple, si l'agent de type $\alpha$ peut améliorer son travail avec l'aide d'un agent de type $\beta$, on peut envisager de signaler ou de détecter la présence des agents $\beta$ aux agents $\alpha$.

À partir de ces trois éléments but/ressources/compétence, on peut déterminer le type d'interactions qui peut s'établir entre deux agents et engager un dialogue adapté. Pour aller plus loin, on peut même anticiper les relations qui existent entre deux agents et garantir par la suite une communication adaptée. Pour cela, des mécanismes d'aides sont produits à partir de l'analyse but/ressources/compétence. Pour les mécanismes entraînant la coopération des agents, la plupart des techniques consistent à analyser dans quelles mesures elle a lieu en déterminant des indices caractérisant le degré de coopération. Selon les auteurs, il existe des familles d'indices différents : [BOU 91] en propose six alors que [FER 95] estime qu'il existe trois familles. Nous pensons qu'il n'existe pas une classification, car elle dépend des types d'agents, et plus généralement du système que l'on veut utiliser.

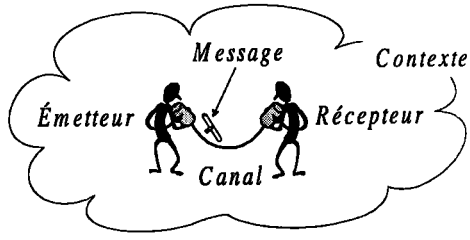



***Figure I.4.*** *Théorie de la communication.*

De manière plus générale, ce qui nous préoccupe c'est la communication entre deux agents. La plupart des théories s'appuient sur la théorie de la communication développée par Shannon et Weaver [SHA 48]. Cette théorie définit la communication comme l'ensemble formé de (cf. figure I.4) : l'émetteur, qui est l'auteur de l'information à transmettre ; le récepteur ou destinataire, qui reçoit l'information ; le message, qui est l'information à transmettre, codée à l'aide d'un langage par l'émetteur et décodée par le destinataire ; le canal ou médium, dans lequel le message transite et qui peut contenir des bruits ; le contexte, qui représente la situation dans laquelle les intervenants se trouvent lors de la communication. Cette définition de la communication donne une vue d'ensemble du problème soulevé par l'acte de communication. Elle aborde le problème dans sa globalité et non comme nous l'avons vu précédemment de manière plus fine en analysant le comportement de l'agent à travers son interaction.

La première relation permettant de déterminer l'acte de communication est la liaison entre l'émetteur et le récepteur. Il est utile de savoir si les deux intervenants se connaissent pour déterminer le type de communication (communication point à point ou diffuse). La deuxième relation dépend de la nature du canal puisqu'il s'agit de définir comment l'émetteur va communiquer : par voie directe, par propagation d'un signal ou par voie d'affichage. Enfin, la dernière relation concerne l'intention de communiquer, la communication n'étant pas toujours intentionnelle.

On constate que les trois relations à considérer pour mieux gérer la communication rejoignent les éléments pris en compte pour déterminer les interactions. En effet, c'est en analysant les buts des agents que l'on détecte une quelconque intention de communiquer. La compétence de l'agent et les ressources dont il a besoin révèle sa capacité à communiquer, et par conséquent son mode de communication. Un agent grâce à sa compétence reconnaît certains agents et dialogue directement avec eux ou encore ses buts à atteindre et/ou les ressources dont il a besoin déterminent si la communication est utile. On retrouve donc également l'intention de communiquer.

Après cette comparaison entre les apports des relations générales sur l'acte de communication et les informations fournies par la décomposition des interactions sur les agents mêmes, on note une certaine redondance. C'est l'analyse des interactions qui nous paraît la plus intéressante à préserver, car non seulement elle apporte des informations plus riches, car plus précises sur la communication, mais elle se base sur une décomposition par agents : sa vision est plus locale.

Lorsqu'une communication s'établit entre deux agents, il est important d'en avoir un certain contrôle afin de garantir un dialogue optimal. Pour cela, il convient de comprendre les différents actes du langage qui interviennent au cours d'une communication. Le problème évident vient dans la manière de décrire ces actes. Il existe différents types d'actes, d'où plusieurs classifications [JAK 63 ; SEA 79 ; VAN 88]. La décomposition que nous présentons est celle qui est apparue dans [AUS 62] et surtout [SEA 69] et qui consiste à décomposer l'acte en trois composantes : une composante locutoire, qui définit le mode d'expression de l'agent (grammaire...), une composante illocutoire, qui permet d'orienter le dialogue (demande, interrogation, affirmation...) et qui contient le contenu du message, et une composante perlocutoire, qui représente l'effet du message sur le destinataire.

En définitive, la modélisation d'un agent comprend non seulement une description destinée à le définir physiquement et fonctionnellement, mais aussi une description de sa communication. Autour de cet agent, des mécanismes de résolution de conflits sont employés. La grande particularité des SMA vient de l'analyse qui consiste à étudier localement chaque agent pour définir sa communication avec l'extérieur. On garantit par cette approche une certaine autonomie pour chaque agent.

### 3.4.Synthèse.

Notre but n'est pas de fournir une analyse complète sur les méthodes et concepts de l'IA, mais d'insister sur l'intérêt d'une telle approche pour la conception d'interfaces interactives de haut niveau. Nous avons montré que l'IA permet de traiter bon nombre de problèmes liés à la prise en compte du comportement humain dans les applications. Non seulement l'IA fournit des mécanismes permettant

d'assurer une certaine souplesse dans le fonctionnement de l'interface (règles, heuristiques, modélisation de la connaissance, architecture multi-agents, ...), mais elle procure également des techniques pour améliorer le dialogue homme/machine (détections, explications, ...).

## 4. LES NOUVELLES INTERFACES.

Au delà des mécanismes pour rendre les interfaces plus adaptées à l'utilisateur, les interfaces elles-mêmes évoluent dans leur philosophie. De nouveaux paramètres sont à considérer. Nous nous intéressons tout particulièrement aux évolutions dans le domaine de la CFAO en mettant en valeur les apports pour l'utilisateur. Notre propos reste toujours le même, à savoir comment les interfaces peuvent évoluer pour être toujours plus à l'écoute de l'utilisateur et rendre transparente la communication homme/machine.

On constate que les interfaces 3D (trois Dimensions) prennent de plus en plus d'importance de nos jours. Une des raisons provient de l'essor de la RV (Réalité Virtuelle). Ainsi dans un premier temps, nous abordons ce point en montrant d'une part, ce qu'est la RV et d'autre part, en introduisant les nouveautés en matière d'interfaces induites par l'avènement de la RV. Dans un deuxième temps, nous expliquons les conséquences en matière d'interactions. Nous montrons notamment quelles sont les interactions à mettre en œuvre pour répondre aux besoins de ces interfaces 3D. Enfin, nous évoquons les environnements multi-utilisateurs qui prennent toute leur importance dans les systèmes de réalité virtuelle et dans lesquels la gestion du dialogue est délicate.

### 4.1.La réalité virtuelle.

Il n'est pas tout à fait exact de montrer la RV comme une nouveauté. Dès les années 1960/1970, des systèmes l'introduisant sont apparus et sont considérés comme les précurseurs [NEW 93]. Toutefois, il semble que ce soit depuis quelques années seulement, sans doute grâce aux progrès du matériel, que les qualités de la RV sont vraiment reconnues et exploitées.

#### *4.1.1.Définition.*

La RV joue un rôle important dans des domaines très variés (mécanique, robotique, artistique ...). Elle permet de réaliser des simulations pour effectuer des tests préliminaires à moindres coûts. Tout le problème est de garantir la fiabilité de la simulation. Des comparaisons ont été effectuées et ont révélé toute la puissance de la RV. Par exemple, [BUT 95] confronte des manipulations réelles, virtuelles et à distance. Les résultats montrent non seulement que la manipulation virtuelle est aussi performante que la manipulation réelle et plus performante que celle à distance, mais qu'elle l'est également pour une manipulation réelle s'appuyant sur une visualisation à deux dimensions.

Les systèmes de RV offrent une haute interactivité qui donne à l'utilisateur l'impression d'avoir un contrôle sur l'environnement [NEW 93]. L'utilisateur prend toute son importance dans ces systèmes : on est loin des interfaces qui imposent à l'utilisateur une ligne de conduite stricte. Pour aller plus loin dans le fonctionnement du couple homme/machine, il s'agit de rendre l'ordinateur si facile

d'utilisation, si familier, si naturel (comme la télévision, le téléphone ou un interrupteur électrique) que l'on puisse en faire abstraction pour se focaliser sur ses fonctions : la communication, l'information et le loisir [PIM 94]. Pour parvenir à une telle souplesse, on demande à l'utilisateur de croire que ce qui n'existe pas existe : c'est là l'ambition de la RV [COI 95].

Il est clair que l'adéquation exacte entre le monde réel et le monde virtuel est impossible à réaliser. Le problème se "restreint" à parvenir à semer le doute entre les deux mondes [LAT 94] : la tâche n'est pas si simple. Certains préconisent même de ne pas forcément représenter fidèlement la réalité, mais un autre monde où la manipulation est aisée [ELK 94 ; HEI 93]. En effet, il ne s'agit pas de garantir une représentation fidèle, mais pouvoir interpréter les phénomènes réels de façon à les rendre optimums pour l'utilisateur. Par exemple, [COI 95] propose de mettre des leurres devant les sens pour convaincre le spectateur et [HER 94] utilise des métaphores pour faire comprendre à l'utilisateur la notion d'espace et comment transposer ses actions réelles en actions virtuelles. Deux axes de recherche se dessinent [COI 95] : la connaissance de l'homme (son système sensoriel, son comportement et ses réactions quand il interagit dans le monde réel) et la connaissance des scènes virtuelles (leur fabrication et les technologies). On ne dispose pas d'un modèle consistant de ce que devrait être la RV [NEW 93]. Toutefois, une règle reconnue existe, la règle des trois I qui préconise pour un système l'Immersion, l'Interaction et l'Imagination [AST 93 ; BUR 93a].

On peut conclure la présentation sur la RV en présentant un point de vue original de [ELK 94]. Il ne présente pas la RV comme une technologie nouvelle, mais une reformulation (voire une reconduite) de problèmes qui se sont posés dans le passé ou dans d'autres domaines comme la philosophie.

#### *4.1.2.Les interfaces multi-modales.*

Pour répondre aux attentes des systèmes de RV, la première idée est d'étendre les possibilités de l'interface en permettant à l'utilisateur de s'exprimer comme il en a l'habitude. Ceci nous amène à définir une interface multi-modale, interface dans laquelle l'interaction se fonde sur l'utilisation simultanée des canaux de communication de l'homme : voix, geste, écriture, vision [COU 94]. Ce type d'interfaces possède des priorités communes aux interfaces qui doivent être gérées dans les systèmes de RV :

- augmenter l'efficacité, par la possibilité d'interagir en parallèle ;
- renforcer la fiabilité, par l'utilisation simultanée de plusieurs canaux. [VEN 93] utilise des sons pour accentuer les interactions (par exemple, une collision entre objets va engendrer un son) ;
- améliorer la prise en compte de la charge cognitive de l'utilisateur. Tout comme on laisse l'utilisateur suivre sa propre logique en gérant les fils d'activités qu'il utilise, on le laisse libre de choisir et de modifier son mode de communication en fonction de ses critères personnels ;
- profiter des capacités sensori-motrices de l'utilisateur, puisque c'est sa façon de communiquer dans le monde réel.

Le manque de retour de force et de sensation tactile lorsqu'un utilisateur manipule un objet graphique est à déplorer dans les systèmes de RV qui préconisent une adéquation presque parfaite avec le monde

réel. De nombreux projets se penchent sur ce problème [ENC 94]. [CAR 90 ; COI 95] conseillent de restituer les informations en s'adressant pour chacune d'elles au bon sens de l'homme : on produit des stimuli pour chaque sens. La représentation des cinq sens est une des préoccupations des systèmes de RV [BAR 92 ; BUR 92 ; BUR 93b ; HER 94 ; NEW 93 ; SUN 92].

Parmi les canaux de communication courants, le geste est un moyen usuel tant dans la réalité que dans le monde virtuel. En effet, depuis l'apparition des interfaces graphiques le système interprète les gestes de l'utilisateur : ces gestes sont certes éloignés des gestes naturels. Ainsi, des systèmes tentent d'apporter des solutions pour utiliser des gestes plus naturels [BAL 94 ; FIG 93]. Les interfaces multi-modales ne sont pas exclusives aux systèmes de RV. Elles mettent en jeu bien d'autres domaines comme la psychologie cognitive, la langue naturelle ... [COU 94]. Ainsi, on peut envisager une évolution rapide de ces interfaces.

#### *4.1.3.Les nouveaux périphériques.*

L'apparition des interfaces multi-modales ainsi que la difficulté de gérer la troisième dimension entraînent l'introduction de nouveaux périphériques. En effet, on ne peut penser à la RV sans penser par exemple aux casques HMD (Head Mounted Display).

L'"haptique" fait partie des technologies de RV actuelles. Ce terme décrit nos réactions physiques avec le monde : sens du toucher, équilibre, position musculaire, résistance au mouvement ... [NEW 93]. L'une des issues de l'"haptique" est de créer des périphériques plus sophistiqués qui fournissent de telles réactions. Par exemple, il s'agit des gants, des casques, des lunettes à obturateurs et, de manière générale, des périphériques à plusieurs degrés de liberté [AST 93 ; ENC 94 ; KAH 94 ; NEW 93 ; VEN 93].

Nous n'envisageons pas de garantir une interface adaptative grâce à ces nouveaux périphériques. En effet, on constate qu'ils posent encore d'énormes problèmes :

- ils sont difficiles à gérer ;
- ils répondent à certains critères aux détriments d'autres. Les casques répondent au besoin d'isoler l'utilisateur dans le monde virtuel et d'oublier ainsi le monde réel, mais ils produisent souvent une mauvaise qualité d'affichage (basse résolution, vitesse d'affichage mauvaise [NEW 93]) ;
- ils sont peu commodes à utiliser. [VEN 93] critiquent l'utilisation d'un crayon optique en 3D, car il est coûteux et fatigant. De manière générale, les outils qui augmentent le degré de liberté ne sont pas très pratiques à utiliser et ne donnent pas une très bonne résolution, ce qui ne facilite pas les interactions [HER 94] ;
- ils doivent répondre à des critères ergonomiques [COI 95]. En effet, le casque ne doit pas gêner l'utilisateur auquel cas il ne pourra être totalement concentré.

Nous pensons qu'il est préférable de définir des techniques indépendantes des périphériques pour améliorer l'interface. Une telle transparence avec les périphériques peut donner plus de puissance à l'interface, car cette dernière devient indépendante des problèmes liés aux périphériques. [FAH 93] préconise l'utilisation d'environnements virtuels pour gérer des environnements multi-utilisateurs

justement parce que l'interface ne doit pas selon lui être liée aux outils multimédia pour communiquer. De même, [VEN 93] préfère fournir des techniques pour permettre à un utilisateur de manipuler des objets en 3D que d'utiliser un périphérique à six degrés de liberté trop contraignant à utiliser. [HER 94] estime qu'il est important de trouver des techniques qui soient indépendantes des périphériques d'entrée/sortie, mais pour certaines tâches uniquement.

Une approche consiste à modéliser l'utilisateur en définissant le fonctionnement d'effecteurs et de capteurs qui sont les seuls éléments directement liés aux périphériques [LAT 94]. Les effecteurs sont chargés de stimuler les perceptions humaines alors que les capteurs sont chargés de détecter toute activité musculaire de l'utilisateur. Un module de confection technique est chargé de faire la relation entre les effecteurs et les capteurs pour l'utilisateur et l'environnement. L'avantage de cette méthode est que les périphériques sont indépendants de l'interface elle-même, car on définit un modèle de représentation de l'utilisateur basé sur des systèmes de perception et des systèmes musculaires qui définissent de manière standard les différents comportements de l'utilisateur. Par exemple, pour le système visuel, le mode d'activité est la vue ; les unités réceptives sont des récepteurs photo ; l'anatomie de l'organe est le mécanisme oculaire comprenant les yeux et le mouvement du corps ; l'activité de l'organe est l'accommodation, l'ajustement de la pupille, la fixation, la convergence et le balayage ; le stimulus est la lumière ; les informations externes sont la taille, la forme, la distance, la localisation, la couleur, la texture et le mouvement.

### 4.2.Les interactions 3D.

Nous nous intéressons dans cette partie à l'interface 3D indépendamment des périphériques utilisés. Dans une première partie, nous évoquons les difficultés à surmonter pour offrir des interactions adaptées à l'utilisateur et à l'interface 3D. Dans une seconde partie, nous présentons les styles d'interactions 3D qui peuvent être proposés à l'utilisateur.

#### *4.2.1.Les besoins.*

L'efficacité d'un environnement virtuel dépend de la qualité de l'interaction [FIG 93]. La difficulté réside dans l'interprétation du monde réel vers le monde virtuel. Il s'agit de bien comprendre et décomposer l'interaction réelle avant de commencer à décrire l'interaction dans l'environnement virtuel [FIG 93 ; HER 94].

En outre, les principales difficultés rencontrées par l'utilisateur dans les interfaces 3D sont essentiellement liées à la visualisation statique trop limitée en 3D [CHU 95] :

- difficulté d'étudier en détail des ensembles d'objets différents dans la scène ;
- problèmes d'occlusion des objets. Ce problème est vite aggravé, car la scène peut posséder un nombre considérable d'objets [HER 94] ;
- différence d'échelle entre les objets parfois considérable. Ceci peut être gênant quand l'utilisateur souhaite étudier un ensemble d'objets de tailles très variées ;
- différence entre ce que l'utilisateur *peut* voir et ce qu'il *voudrait* voir ;

- difficulté de comparer des objets, car ils ne sont pas contigus dans l'espace.

[HER 94] ajoute au problème de la visualisation statique, celui de la visualisation dynamique. En effet, l'utilisateur peut facilement se perdre lorsqu'il navigue dans l'espace virtuel étant donné le manque de repères qui lui sont fournis par rapport au monde réel. De plus, il ne faut pas nécessairement vouloir partir sur les bases de l'interface 2D classique : non seulement les interactions 2D ne sont pas nécessairement adaptées à la 3D, mais le public risque d'être différent et les interactions à offrir ne sont pas forcément identiques [HER 94]. En effet, il y a de fortes chances pour que les utilisateurs des applications 2D continuent à les utiliser pour des raisons très diverses (par exemple, se conformer au standard, coût d'adaptation trop élevé ...) En conséquence, il faut traduire un maximum de connaissances sur l'utilisateur.

*4.2.2.Les techniques d'interactions.*

On constate que les efforts en matière d'interactions 3D concernent deux catégories de techniques : les mécanismes visant à améliorer les conditions d'interactions de l'utilisateur et les styles d'interactions.

D'une part, on retrouve des mécanismes pour que l'utilisateur reconnaisse ses interactions, car elles fonctionnent comme dans le monde réel. On retrouve notamment des mécanismes chargés de détecter les collisions d'objets lors de manipulations [FIG 93].

D'autre part, il s'agit de créer des mécanismes qui ne tentent pas d'être conformes à la réalité. Une première raison est qu'ils tirent partie de ce nouveau monde, pour lequel les règles ne sont pas clairement définies, afin de définir des aides plus adaptées à l'utilisateur que celles du monde réel : le monde virtuel offre la possibilité de corriger les défauts du monde réel. Une autre raison provient d'une limite des mondes virtuels qui n'intègrent pas toutes les fonctionnalités du monde réel : il est donc nécessaire d'inclure des mécanismes propres au monde virtuel. Afin d'améliorer les performances dans la manipulation des objets, des mécanismes tels que des attractions d'objets et la définition de zones d'aimantation sont introduits [KAN 95 ; VEN 93]. Pour résoudre les problèmes d'occlusion, différents mécanismes apparaissent parmi lesquels [CHU 95] :

- transparence momentanée des objets [VEN 93] ;
- élévation de l'ensemble des objets à manipuler ;
- réduction des objets non concernés par les traitements ;
- visualisation des objets à des échelles différentes dans la même scène. Lorsque l'utilisateur manipule un ensemble d'objets, il peut être gêné dans ses interactions sur les objets de cet ensemble parce que certains sont très éloignés et par conséquent quasiment invisibles. On comprend alors l'intérêt de pouvoir modifier l'échelle des objets éloignés.

Quant aux styles d'interactions, ils ont évolué dans deux directions : fournir des outils indépendants du monde virtuel et disposer d'outils graphiques qui interviennent dans le monde virtuel. Parmi le premier type d'outils, nous présentons ceux que l'on assimile à un nouveau type d'interfaces :

l'interface transparente ("The See-Through Interface") [BIE 93]. Ce sont des outils interactifs semi-transparents qui apparaissent entre l'application et l'outil de désignation de l'utilisateur. Il s'agit d'outils graphiques qui n'appartiennent pas à la scène, mais à un niveau supérieur : pour comprendre, il suffit d'imaginer que l'on place une plaque de verre devant la scène et que cette plaque constitue l'espace sur lequel sont définis ces outils. On trouve des exemples de tels outils dans [BIE 93] : on peut citer les Magic Lens™ qui, outre la fonction de loupe qu'elles procurent, permettent de montrer des informations cachées ou dans un autre format. Ce style d'interactions proposé à l'utilisateur n'est pas exclusif aux interfaces 3D, mais il a l'avantage justement de permettre d'inclure dans ces interfaces 3D des outils d'interactions 2D performants. Cette technique apporte de nombreux avantages [BIE 93] :

- place illimitée pour les outils, qui ne se restreint pas à la zone de travail ;
- combinaison de tâches, car non seulement un outil peut regrouper un ensemble de fonctionnalités, mais plusieurs outils peuvent travailler ensemble grâce à leur aspect semi-transparent ;
- interface indépendante de l'affichage ;
- compréhension de l'application simplifiée, car les outils tels que les Magic Lens™ sont des aides.

Toutefois, la manipulation de ces outils révèle une certaine lourdeur. En effet, il arrive que ces outils semi-transparents cachent une partie des objets, donc risquent de gêner l'utilisateur [CHU 95]. Ainsi, le second axe de recherche en matière de style d'interactions concerne la définition d'outils graphiques intégrés au monde virtuel. Ces outils deviennent de véritables objets virtuels dans l'espace 3D aux côtés d'objets d'application [GOB 95]. Il existe plusieurs niveaux de dépendance de ces outils avec les objets d'application.

Tout d'abord, on rencontre une indépendance totale, car les outils sont dédiés à l'utilisateur exclusivement ou à la communication entre plusieurs utilisateurs. Dans DIVE [FAH 93], on trouve, dans l'environnement virtuel, un tableau noir destiné à n'importe quel utilisateur pour réfléchir et s'exprimer librement (joue le rôle d'une feuille de brouillon) ou encore une table de conférence spécialement conçue pour regrouper des utilisateurs virtuels et communiquer.

Ensuite, l'indépendance des outils avec les objets peut être relative : [KAN 95] montre le développement d'outils pour annoter des objets. Ces outils s'appliquent sur des objets mais ne sont pas propres à ces objets.

Enfin, la dépendance peut être totale : il s'agit d'associer à un objet un outil permettant d'interagir sur lui. Le système SDM de [CHU 95] intègre ce type d'outils : les objets peuvent être munis de leviers qui sont des outils permettant de modifier certains de leurs paramètres géométriques (la hauteur par exemple).

Ces outils virtuels nécessitent une gestion particulière. [GOB 95] détermine leur apparence par une hiérarchie de modélisation et leur comportement par un réseau de contraintes internes. Dans la même philosophie, le système SDM de [CHU 95] définit pour des ensembles d'objets un interacteur qui connaît le nombre de paramètres modifiables des objets ainsi que la nature de la transformation : ces interacteurs sont contrôlés par les leviers que l'utilisateur peut manipuler.

Bien souvent, les travaux présentés préconisent un style d'interactions. Il est pourtant préférable de disposer de techniques d'interactions suffisamment souples pour être combinées et résoudre un plus large éventail de problèmes [CHU 95].

### 4.3.Les environnements multi-utilisateurs.

L'apparition de la réalité virtuelle introduit inévitablement celle des environnements multi-utilisateurs : le système Workroom de [PIM 93] en est un exemple, car il permet à deux utilisateurs de construire et décorer une maison dans une simulation de RV. En effet, les systèmes de RV sont le reflet des mondes réels dans lesquels les habitants agissent individuellement, en parallèle ou en communauté. Cette considération a des conséquences sur l'interface qui doit alors gérer les interactions de l'utilisateur concernant la communication avec d'autres utilisateurs. Rendre le dialogue adaptatif devient plus difficile, car il faut que la communication entre utilisateurs soit relativement naturelle.

Le système DIVE présenté dans [FAH 93] permet de créer des environnements virtuels distribués. La gestion de la communication est effectuée via des techniques d'interactions. Le but est de garantir une utilisation transparente, intuitive et naturelle des outils de communication et de coopération. Deux types de techniques sont mis à la disposition des utilisateurs :

- détection de présence d'autres utilisateurs ; ce mécanisme facilite les interactions. Chaque utilisateur contient un englobant appelé "aura" qui définit la région dans laquelle sa présence peut être perçue;
- mise à disposition d'outils virtuels. Ces outils sont soit dédiés aux utilisateurs pour communiquer, soit accessibles à plusieurs utilisateurs.

Le mécanisme des "auras" peut être intéressant pour filtrer les interactions qu'un utilisateur peut avoir avec le reste du monde virtuel. Toutefois, il paraît délicat de définir cette zone de présence. Dans [FAH 93], elle n'est pas très réaliste, voire un peu trop simpliste, car elle est représentée par une sphère et il ne définit aucun critère particulier.

Les outils virtuels permettent de contrôler la communication des utilisateurs. L'avantage est qu'ils ne se contentent pas de proposer de simples fonctionnalités, mais qu'ils réagissent aux interactions des utilisateurs : des "auras" sont également définies pour ces outils. Par exemple, on trouve dans [FAH 93] la définition d'un outil représentant une table de conférence. Son but est de regrouper des utilisateurs pour qu'ils puissent dialoguer tranquillement. Quand un utilisateur s'approche de l'"aura" de la table, cette table considère sa présence en l'incluant à la conférence et en créant automatiquement des canaux de communication avec les autres participants : cet outil virtuel facilite les interactions entre utilisateurs.

En conséquence, on constate que les environnements multi-utilisateurs augmentent la complexité de l'interface. Les outils virtuels deviennent plus difficiles à gérer, car ils doivent intégrer des mécanismes propres à la communication et aux problèmes qu'elle peut entraîner (concurrence ...).

### 4.4.Synthèse.

Les interfaces 3D procurent de par leur nature des dialogues plus adaptés à l'utilisateur, puisque l'un de leur but est de permettre une meilleure interprétation des dialogues réels. Les techniques d'interactions à mettre en œuvre n'en sont que plus difficiles. Nous avons établi une liste non exhaustive de ces techniques. Néanmoins, nous avons voulu insister sur les grandes catégories de techniques qu'il est intéressant de considérer. Nous soulignons l'importance de définir des outils graphiques dédiés au dialogue. Ils respectent le besoin primordial des interfaces interactives qui est d'être les plus transparentes possibles : l'utilisateur est en immersion totale dans l'application, et ceci même pour interagir.

## 5. Conclusion.

Le dialogue homme/machine prend désormais une place importante dans l'élaboration d'un système. Nous avons montré qu'il devient de plus en plus complexe à définir, car on désire maintenant construire des dialogues adaptés à l'utilisateur. C'est l'interface homme/machine toute entière qui est concernée : des modèles d'architecture aux modèles d'interactions comprenant les styles d'interactions, en passant par les outils de construction d'interfaces. Chaque niveau de description est une barrière à franchir pour garantir que le dialogue homme/machine obtenu est satisfaisant.

De manière plus globale, si le dialogue n'est pas aussi naturel qu'on le souhaite, c'est qu'il a été mal décrit. On peut trouver deux raisons : une mauvaise interprétation des connaissances transmises par le concepteur (ou l'expert) et des lacunes dans les mécanismes associés aux interactions de l'utilisateur (par exemple, le manque d'aides pour l'utilisateur). Pour tenter de répondre à ces problèmes, nous avons évoqué l'IA qui est un domaine dont la principale occupation est justement la gestion des connaissances, de l'acquisition à l'exploitation. Les techniques et les méthodes d'analyse que l'IA procure sont d'autant plus intéressantes que nous nous intéressons au dialogue homme/machine pour les systèmes de CFAO. En effet, nous avons montré la forte implication de l'utilisateur dans ces systèmes, surtout si l'on considère les systèmes de RV : le dialogue est complexe à gérer. L'un des problèmes que l'on rencontre dans ces systèmes trouve sa source dans la grande diversité des utilisateurs qui interviennent. Ceci sous-entend qu'il faut tenir compte de l'évolution éventuelle de l'interface toute entière en fonction du niveau de compétence de l'utilisateur.

Dans ce chapitre, nous avons montré comment améliorer les capacités des systèmes de CFAO en les rendant plus performants dans les dialogues proposés aux utilisateurs. Les approches que nous avons analysées font appel à des domaines a priori indépendants (architecture d'interfaces, intelligence artificielle, réalité virtuelle ...). Nous n'avons pas cherché à comparer ces différentes approches dans la mesure où elles sont complémentaires et donnent des solutions sur les besoins dédiés aux systèmes de CFAO.

Dans les chapitres suivants, nous nous efforçons de trouver des solutions pour construire un dialogue homme/machine "idéal" pour la CFAO. Ainsi, nous nous intéressons tout d'abord à la spécification même de l'interface qui demande en général beaucoup trop d'implications de la part du concepteur.

Nous nous basons sur les concepts de construction d'interfaces présentés au début de ce chapitre : modèles d'architecture, modèles de dialogue et modèles d'interactions (cf. 2. L'interface homme/machine et la CFAO). La difficulté est de gérer les compétences différentes qui existent entre les concepteurs et de fournir les bons outils. Nous voulons également tenir compte de l'évolution éventuelle de l'interface. Nous définissons ainsi des outils de construction assez souples pour parvenir à capturer les connaissances des concepteurs.

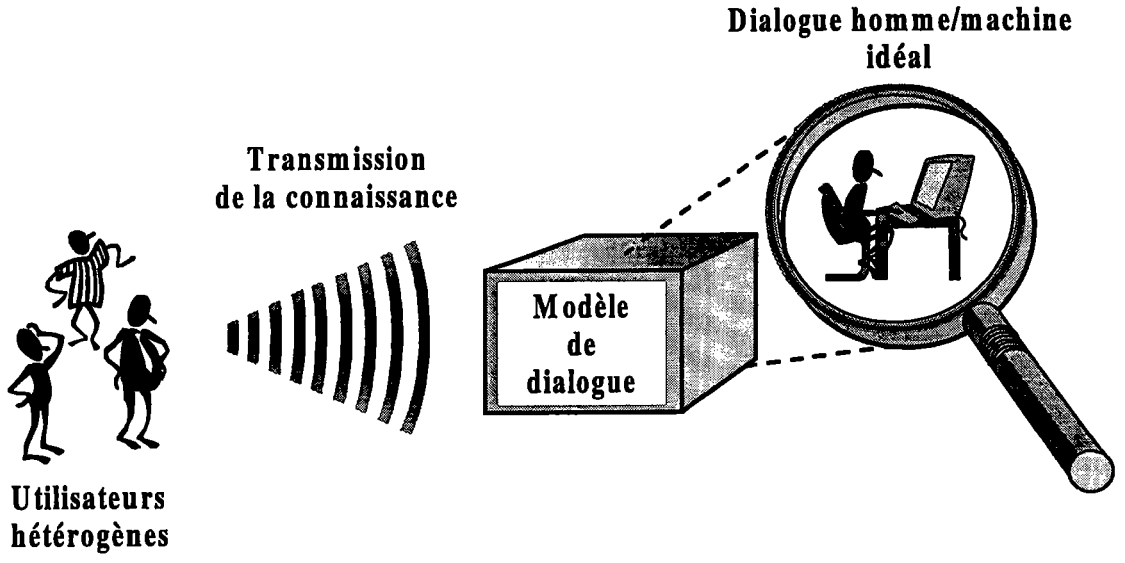
Ensuite, nous nous attachons à définir des outils dédiés à l'utilisateur final pour lui faciliter le dialogue. Nous avons montré qu'il existe beaucoup de techniques d'interactions. Nous avons pu soulever ce problème pour les systèmes de RV pour lesquels il n'existe pas de modèle prédéfini, ce qui a donné naissance à toute une série de tendances. Notre propos consiste à présenter un concept général de description des outils de dialogue 3D pour permettre ainsi de les centraliser pour une meilleure gestion.

Les dialogues actuels tentent de satisfaire l'utilisateur, car les mécanismes montrent le besoin de capturer certaines des connaissances de l'utilisateur pour définir l'interface. Toutefois, aucun ne propose une réelle spécification de l'utilisateur i.e. de considérer l'utilisateur comme une entité instable que le système essaie de maîtriser. Nous proposons, dans le dernier chapitre, une solution à ce problème qui nous paraît essentiel pour parvenir à un dialogue adaptatif : la modélisation de l'opérateur.

Les travaux que nous présentons dans les chapitres suivants intègrent des concepts d'IA pour parvenir à la construction d'un dialogue adaptatif. La raison provient de l'importance toute particulière que l'on a voulu donner aux connaissances des utilisateurs. Les concepts que nous présentons mettent en œuvre, plus ou moins implicitement, des techniques d'acquisition des connaissances.

# CHAPITRE 2.

# CONSTRUCTION D'UN DIALOGUE "IDÉAL" EN CFAO.



## 1. INTRODUCTION.

Lors de la description du dialogue, le concepteur, qu'il soit expert ou non du domaine des interfaces, transmet ses propres connaissances dans la mesure de ses possibilités. D'une part, le concepteur doit pouvoir transmettre aisément ses connaissances : il faut disposer de mécanismes garantissant la qualité de l'interprétation faite par le système. D'autre part, le fonctionnement du dialogue décrit doit offrir une grande liberté à l'opérateur (utilisateur final). Ainsi, la spécification du dialogue doit reposer sur des bases simples, mais qui permettent de décrire des fonctionnalités de haut niveau pour l'opérateur.

Nous avons évoqué dans le chapitre précédent différentes techniques pour construire et représenter le dialogue, mais aucune ne nous satisfait pleinement. En effet, elles ne nous paraissent pas complètes dans la mesure où l'un au moins des critères suivants n'est pas respecté :

- elles ne s'adaptent pas aux différents types de concepteurs qui peuvent intervenir dans l'élaboration du dialogue ;
- elles ne considèrent pas le caractère évolutif de l'interface ;
- elles ne permettent pas de décrire un dialogue interactif de haut niveau ;
- elles ne répondent pas toujours aux besoins des systèmes de CFAO que nous avons mentionnés dans le chapitre précédent (cf. Chapitre 1 *2.1.2. Des besoins orientés CFAO*) .

Par conséquent, nous proposons de décrire nos propres concepts en matière de construction de dialogue afin que ces critères soient respectés. Pour y parvenir, nous décomposons le problème en proposant plusieurs approches pour décrire le dialogue tout en garantissant une certaine complémentarité entre elles. Tout d'abord, nous présentons une première approche qui est dédiée à des utilisateurs expérimentés qui connaissent le modèle de dialogue utilisé. Nous montrons également comment fonctionne ce modèle de dialogue pour répondre à nos critères. Cette première approche constitue la base nécessaire qui nous a permis de développer nos travaux ; notre but n'est donc pas de développer en détail cette approche qui l'est dans [MAR 95], mais de montrer les caractéristiques générales qui nous ont été utiles.

Dans les parties suivantes, nous nous intéressons au développement d'une seconde approche plus intuitive, mais qui repose toujours sur ce même modèle de dialogue. Nous nous attachons à définir les nouveaux concepts nécessaires à l'élaboration d'une telle approche. Ensuite, nous mettons en évidence le mécanisme de construction qui nous permet d'obtenir un dialogue que nous qualifions "d'idéal" du point de vue adaptatif. Enfin, nous insistons sur la puissance des concepts que nous introduisons pour le dialogue en montrant qu'ils répondent à d'autres problèmes de représentation des connaissances.

## 2. UNE APPROCHE POUR UTILISATEUR EXPÉRIMENTÉ.

Le Laboratoire de Recherche en Informatique de Metz (LRIM) a développé depuis 1986 le système SACADO (Système Adaptatif de Conception et d'Aide au Développement par Ordinateur) [GAR 88 ; GAR 93]. Ce système vise à répondre à une première approche en matière de dialogue : établir des concepts solides permettant la description de dialogues complexes i.e. performants. Nous présentons

tout d'abord ces concepts de base établis pour le dialogue. Ensuite, nous poursuivons par la présentation d'une première méthode de spécification directement issue de ces concepts.

## 2.1.Les concepts de base.

SACADO a été conçu pour servir de base de développement pour toutes les applications de CFAO du laboratoire. Sa difficulté, mais aussi son atout, est de fournir un noyau aussi complet que possible pour répondre à tout type d'applications de CFAO et de manière plus que satisfaisante.

Ce qui caractérise les applications ce sont les différentes fonctionnalités qu'elles proposent. SACADO se fonde sur ce principe pour représenter l'architecture du dialogue. Il contient une architecture décrivant toute application par ses fonctionnalités. La décomposition qu'il propose est suffisamment souple pour ne pas imposer des séquences trop strictes dans le dialogue. La figure II.1 illustre cette architecture.

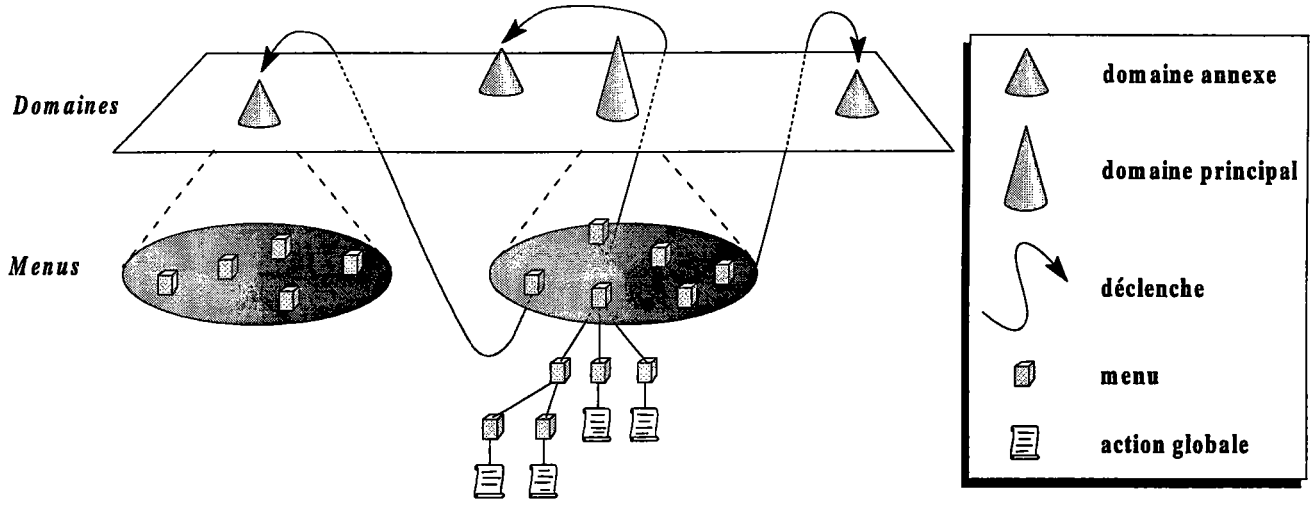



***Figure II.1.** Architecture des fonctionnalités sous SACADO.*

Les fonctionnalités sont regroupées par domaines [MAR 95]. On définit un domaine principal qui renferme les fonctionnalités de base de l'application. Autour de ce domaine est définie une série de domaines annexes. Ces domaines caractérisent des fonctionnalités qui interviennent dans un contexte donné : c'est le domaine principal qui se charge de contrôler les domaines annexes. Chaque domaine est constitué d'un ensemble de hiérarchies de menus spécifiant les chemins pour arriver aux fonctionnalités offertes par l'application. Ceci est une première réponse à la gestion des fils d'activités multiples de l'utilisateur. Tout menu terminal, i.e. se trouvant au plus bas d'une hiérarchie, est associé à une action globale. Une action globale constitue l'ensemble des traitements à effectuer pour réaliser une fonctionnalité [MAR 95].

Les actions globales intègrent les phases de dialogue entre l'utilisateur et le système. La difficulté est de décrire ces actions globales sans imposer un dialogue séquentiel tout en gardant un contrôle sur les actes de l'utilisateur. Ainsi, on représente l'action sous forme d'un graphe orienté (cf. figure II.2.) dont les nœuds sont soit des interactions, soit des actions non interactives et les arcs sont les enchaînements du dialogue que le système peut contrôler.

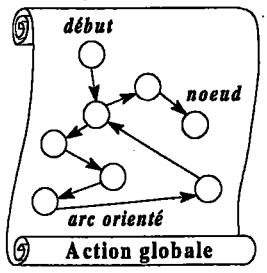



***Figure II.2.** Action globale.*

Les actions non interactives symbolisent les traitements statiques i.e. ne demandant aucune intervention de l'utilisateur : on peut les assimiler à de simples fonctions. Elles assurent une décomposition modulaire logique dans la description de la fonctionnalité représentée par l'action globale. Les nœuds de type interaction indiquent une intervention de l'utilisateur. La particularité du noyau SACADO est qu'une unique primitive d'interactions est définie indépendamment du style d'interactions pour gérer n'importe quel nœud interaction. Ceci permet de centraliser et de contrôler l'ensemble des intentions de l'utilisateur. Cette primitive unique revient à décrire les méthodes requises par le système pour signifier à l'utilisateur son intervention, pour lui fournir les moyens de répondre et pour contrôler les choix de l'opérateur.

Lorsque le système se trouve sur un nœud d'interactions, il attend une réponse de la part de l'utilisateur et c'est alors en fonction de la réponse fournie que le système orientera la poursuite du dialogue. Une interaction, telle qu'elle a été décrite jusqu'à présent, constitue un élément de base pour gérer l'intervention de l'utilisateur. Des opérateurs de composition (*interaction* ET *interaction*, *interaction* OU *interaction*, ENSEMBLE *d'interactions*) ont été introduits pour étendre les possibilités offertes par ces interactions de base en créant des interactions de plus haut niveau. Ces nouvelles interactions, quoique plus complexes, reposent toujours sur la même primitive d'interactions [MAR 95]. Le modèle de dialogue peut devenir beaucoup plus complexe et décrire ainsi des dialogues plus réalistes.

Pour garantir une certaine souplesse dans l'intervention de l'utilisateur, le système permet à ce dernier lors d'une interaction de suivre sa propre logique en autorisant certaines déviations dans ses actions. SACADO gère cette possibilité grâce aux menus sur lesquels il définit des compatibilités entre leur utilisation et l'interaction en attente d'une réponse. Supposons qu'une action globale *A* soit en attente d'une donnée de l'utilisateur ; outre une réponse directe i.e. l'utilisateur fournit la donnée, SACADO permet de gérer trois types de réponses (cf. figure II.3.) :

- l'utilisateur active un menu local qui permet d'effectuer une fonctionnalité (par l'intermédiaire de l'action globale *B*) lui donnant les moyens de répondre à l'interaction de *A* ;
- l'utilisateur active un menu immédiat qui permet de suspendre temporairement *A* pour effectuer une fonctionnalité annexe. Dans le discours naturel, on appelle cette intervention communément "une parenthèse" ;
- l'utilisateur active un menu différé qui permet de modifier son dialogue en interrompant brutalement et définitivement *A*. Le système prend donc en compte les cas où l'utilisateur se trompe ou encore change de stratégie.

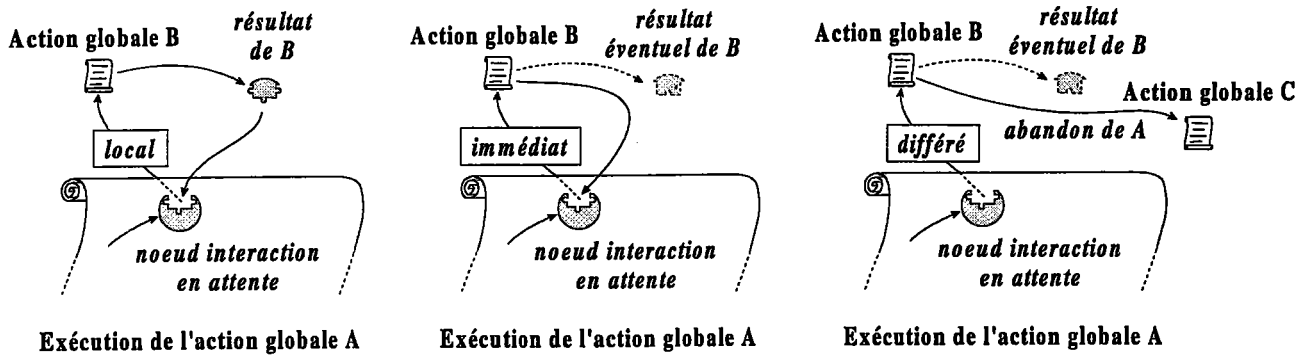



*Figure II.3. Trois types de compatibilités sur les menus.*

Ces compatibilités s'appliquent non seulement entre un menu et une interaction, mais également entre menus. La gestion des compatibilités apporte un certain dynamisme dans le dialogue. En effet, l'opérateur peut suivre sa propre logique, car les dialogues ne sont pas figés : on est loin des dialogues orientés par la tâche (plans définis a priori) évoqués dans [GRA 94]. Le système SACADO permet la gestion des fils d'activités multiples qui caractérisent le dialogue d'un utilisateur. De plus, cette approche garantit un suivi lors des interventions de l'utilisateur : à tout moment, il peut revenir sur ses actions, car SACADO gère les cas d'annulations. Les compatibilités définies sur les menus informent à tout moment le système de la situation dans laquelle l'utilisateur se trouve, ce qui facilite ses réactions pour répondre en accord avec les impératifs du logiciel et de l'utilisateur.

### 2.2.Un formalisme graphique.

Étant donné la complexité que peut prendre le modèle de dialogue sous SACADO, l'utilisation d'un langage textuel s'avère rapidement lourd à manipuler pour l'utilisateur chargé de décrire le dialogue d'une application de CFAO. Une programmation visuelle a été envisagée pour permettre une description plus aisée.

Le formalisme graphique défini pour décrire le dialogue géré par le noyau SACADO est entièrement décrit dans [MAR 95]. Il se base sur une interprétation des réseaux à transitions d'états. De par la décomposition de la description du dialogue en actions globales, le dialogue à modéliser est moins complexe à décrire que dans une approche classique des réseaux à transitions d'états. À chaque action globale va correspondre une formalisation graphique, ce qui apporte une vision plus locale au problème. En fait, le formalisme reprend des réseaux la notion d'états symbolisant un mode particulier et la notion d'arcs pour représenter les différentes dépendances entre les modes. Cependant toutes les dépendances ne sont pas explicitement représentées comme le préconise normalement la représentation sous forme de réseaux, car les interactions en renferment quelques-unes, dont notamment les dépendances dues aux compatibilités.

La figure II.4 illustre les principaux composants du formalisme. On y retrouve les notions d'interactions et d'actions non interactives que nous avons évoquées dans le modèle de SACADO pour décrire les séquences de dialogue d'une action globale. Le formalisme de l'interaction inclut non seulement la description du type d'interactions attendu par l'utilisateur (par exemple un point), mais

aussi l'ensemble des compatibilités qui se trouvent sur les menus pour l'interaction. Une vue sélective i.e. par niveau de description est envisagée afin de simplifier la lecture du formalisme. Par exemple, connaître les menus différés pour une interaction donnée est nécessaire pour décrire complètement le dialogue, mais n'est pas indispensable pour la compréhension globale. De la même manière, les interactions sont représentées plus schématiquement quand on leur applique des opérateurs de composition : ceci évite d'autant plus la surcharge de la représentation que l'interaction pouvant subir de tels opérateurs n'est pas forcément une interaction de base, mais déjà une interaction composée.

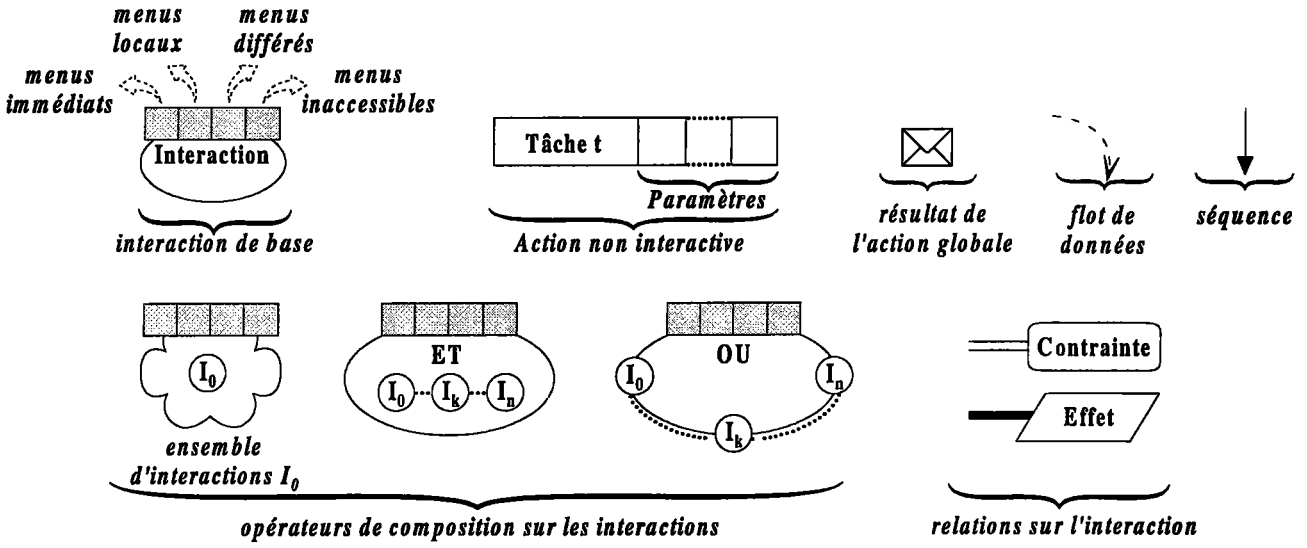



***Figure II.4.** Éléments principaux du formalisme graphique.*

En principe, une action globale décrit une fonctionnalité de l'application, donc fournit souvent un résultat. Par conséquent, le formalisme intègre cette possibilité en représentant le résultat d'une action globale par une enveloppe. Ce composant est associé au dernier nœud de description de l'action.

Chaque interaction peut être soumise à des contraintes particulières : par exemple, le rayon d'un cercle demandé à l'utilisateur ne doit pas excéder 50 cm. De plus, elle peut intégrer des traitements spécifiques (ou effets) au contexte de l'interaction : par exemple, lors de l'attente d'un point correspondant à l'interaction de saisie du second point de création d'un segment, détecter si le segment provisoirement donné par l'utilisateur est horizontal ou vertical. Ces deux relations i.e. la contrainte et l'effet sont ainsi représentées dans le formalisme. Elles ne sont pas internes à l'interaction, car elles sont étroitement liées au contexte dans lequel l'interaction est décrite.

Les derniers composants importants du formalisme sont les liens : la séquence et le flot de données. La séquence lie deux nœuds pour montrer l'enchaînement logique pour parvenir à la réalisation complète d'une action globale. Le flot de données traduit les transferts de données entre les interactions qui capturent des informations et les actions non interactives qui en ont besoin pour leurs traitements.

Nous ne mentionnons pas les différentes règles de cohérence associées à ce formalisme pour garantir une représentation du dialogue conforme au noyau SACADO. Notre propos n'est pas de permettre la manipulation d'un tel formalisme, mais de pouvoir le comprendre et de montrer que cette méthode permet de faciliter la description du dialogue.

### 2.3.Synthèse.

Le modèle de dialogue de SACADO répond aux besoins des systèmes de CFAO. L'utilisateur prend une place importante dans la conception du système : rappelons que, dans les systèmes de CFAO, l'utilisateur intervient très fréquemment. De plus, la gestion de l'interaction sous SACADO est isolée en une unique primitive, ce qui facilite une gestion spécifique et adaptée : on peut se concentrer sur le traitement des différents styles d'interactions plus ou moins complexes qui peuvent se produire, l'utilisateur intervenant de manière fortement graphique. De manière générale, SACADO tente de préserver la séparation entre l'interface et l'application de par l'architecture qu'il propose.

Il existe des modèles se rapprochant du modèle SACADO dont notamment le modèle IAMBUS [HUB 89] (cf. Chapitre 1 *2.3.3. Un exemple de modèle de dialogue*). On retrouve la décomposition du dialogue en interactions de base ainsi que la notion de graphe pour donner le schéma global de fonctionnement du dialogue. Toutefois, l'analyse est différente, car SACADO raisonne en terme d'actions globales, ce qui offre une certaine modularité dans la description du dialogue. De plus, les interactions gérées par SACADO sont plus souples et plus puissantes, car :

- elles reposent sur une unique primitive qui décrit les composants standard de gestion de l'interaction ;
- elles disposent "d'habillages" spécifiques au contexte de chaque action globale sous la forme des contraintes et des effets associés aux interactions, mais qui ne sont pas internes à ces interactions.

On dispose d'un modèle de dialogue complet offrant la possibilité de décrire des dialogues interactifs de haut niveau : on sait qu'on peut arriver à décrire des dialogues adaptatifs avec ce modèle, car il tient compte de l'utilisateur en tant qu'élément instable du système (on entend par instable le caractère opportuniste de l'utilisateur).

Pour bénéficier de toutes les possibilités offertes par ce modèle de dialogue, un formalisme graphique est proposé à l'utilisateur (le concepteur de l'interface) pour en simplifier la description. La programmation visuelle et la manipulation directe permettent d'avoir une vision simplifiée du modèle de dialogue à décrire.

Différents utilisateurs interviennent au cours du développement du dialogue (on envisage même l'intervention de l'opérateur) ; il faut donc être adapté à ces différents intervenants, ce qui n'est pas le cas ici. En effet, le formalisme bien que graphique demande des connaissances sur l'architecture et le modèle de dialogue utilisé, ce qui ne devrait pas faire partie des connaissances de base requises. On ne peut toutefois abandonner cette technique de description puisqu'elle convient parfaitement à des utilisateurs expérimentés. On a besoin de nouveaux concepts pour offrir un mode de description plus adapté et sans remettre en cause les concepts de base du dialogue.

## 3. DE NOUVELLES CONNAISSANCES POUR LE DIALOGUE.

Lorsqu'un concepteur décrit une application de CFAO, il donne une quantité d'informations que le système va exploiter pour construire au mieux l'application désirée. Toute la difficulté pour le système

est l'interprétation de ces informations, car de cette interprétation va dépendre la puissance des mécanismes que le système va pouvoir mettre en œuvre. On s'intéresse plus particulièrement à la capture des connaissances du concepteur pour décrire le dialogue.

Nous avons vu que le problème du dialogue en CFAO n'est pas au niveau de l'architecture même du dialogue : les concepts de base du système SACADO sont suffisamment puissants pour décrire tout type de dialogue. Le risque vient des connaissances mal exploitées qui entraînent une mauvaise utilisation du modèle de dialogue. Dans le chapitre précédent, nous avons soulevé le problème de l'acquisition et de la représentation des connaissances (cf. Chapitre 1 3.1. Une notion de connaissance) : nous avons insisté sur l'importance de la maîtrise des connaissances.

Ainsi, nous proposons une étude du dialogue à travers les connaissances transmises par l'utilisateur, concepteur expérimenté ou non dans le modèle de dialogue. Nous étudions tout d'abord les informations qui sont classiquement représentées dans les modèles de CFAO. Ensuite, en partant de la constatation que les connaissances transmises par l'utilisateur pour décrire ces informations sont mal exploitées, nous introduisons un nouveau concept permettant de mieux cerner toutes les informations : le concept des comportements. Au cours de notre étude, nous insistons également sur le fait que notre méthode de description des connaissances s'attache à ne pas contraindre l'utilisateur.

### 3.1.Les connaissances modélisées.

Pour gérer les objets graphiques manipulés dans les systèmes de CFAO, on utilise des représentations informatiques de ces objets : ce sont des modèles. La tendance actuelle est de définir des modèles adaptés aux traitements : modèle de contraintes, modèle surfacique, modèle de fonctionnement ... La notion de multi-modélisation est très présente en CFAO [GAR 91 ; SUK 93].

Les modèles géométriques sont parmi les modèles incontournables : l'objet y est représenté par un ensemble de composants géométriques et des liens avec d'autres composants parmi lesquels des structures hiérarchiques et d'ingénierie [SUK 93]. Cette définition met en valeur un point important : les relations sur les objets i.e. les propriétés. Elles sont aussi importantes à modéliser que les objets eux-mêmes.

Dans la suite, on considère qu'il existe un modèle générique, base de tous les modèles. Notre propos consiste, à l'aide de cette notion de modèle générique, à montrer de manière synthétique les connaissances réellement modélisées. Nous considérons que les objets et les propriétés sur les objets sont les éléments essentiels du modèle générique. Ainsi, nous résumons, dans les deux parties suivantes, les connaissances décrites pour ces deux éléments.

#### *3.1.1.Les objets.*

Nous appelons objets toutes les entités graphiques qu'un opérateur va pouvoir définir dans son application de CFAO pour concevoir son produit. Nous choisissons de prendre pour exemple le cercle qui est un objet simple couramment utilisé.

De manière générale, un objet est modélisé en utilisant différentes représentations. Nous considérons qu'il existe une représentation générique qui puisse servir de référence pour toutes les autres représentations : c'est la représentation canonique. Nous nous intéressons essentiellement à cette représentation, puisqu'elle symbolise la manière de modéliser la connaissance. Dans la suite, nous utilisons le terme représentation pour désigner la représentation canonique.

La représentation de l'objet doit spécifier de manière concise l'objet pour le système. Elle contient classiquement, dans sa forme la plus simple, des informations précisant les caractéristiques géométriques de l'objet. La figure II.5. illustre les données caractérisant la représentation d'un objet quelconque et également celle du cercle à titre d'exemple.

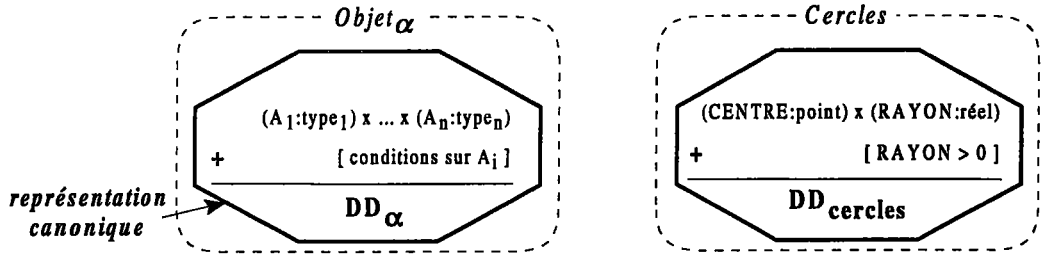



***Figure II.5.** Représentation canonique d'un objet quelconque et exemple pour le cercle.*

Pour un objet de type $\alpha$, on assimile sa représentation à son domaine de définition $DD_\alpha$ que l'on note comme l'ensemble formé [MAR 95 ; GAR 96a] :

- du produit cartésien des attributs $A_i$ caractérisant l'objet ;
- des relations particulières qui peuvent exister sur ces attributs.

Dans le cas de l'objet cercle, on peut prendre comme attributs le définissant de manière unique, son centre et son rayon. Les domaines de définition de ces attributs permettent de mieux cibler l'objet : ici, on définit ces domaines de définition par le type $TYPE_i$ de l'attribut. Considérer uniquement des domaines pour les attributs n'est pas toujours suffisant, il convient d'ajouter des conditions entre attributs (contraintes éventuellement croisées sur les domaines des différents attributs).

### *3.1.2.Les propriétés.*

Dans les systèmes de CFAO, l'utilisateur manipule souvent des contraintes. Ce sont des éléments-clés au même titre que les objets : non seulement, l'opérateur crée des objets pour faire apparaître des contraintes mais aussi des contraintes pour déterminer des objets. Les contraintes sont en fait des propriétés à garantir. Ainsi, ce sont les propriétés qui sont considérées comme des objets à part entière. Des connaissances spécifiques sont par conséquent transmises au système pour les décrire. Cette séparation vis-à-vis des objets autorise la spécification des propriétés par leur sémantique, qui s'appuie sur une description générique indépendante des objets.

Comme pour les objets, une propriété peut être définie de différentes manières. On définit également une représentation canonique à laquelle les autres représentations peuvent se ramener. Ainsi que nous

le montrons à travers la figure II.6., la représentation canonique d'une propriété $\beta$ est définie par le domaine de définition $DD_\beta$, ensemble composé du produit cartésien de données $D_i$ et de conditions sur ces données. Le nombre des données est égal au nombre d'objets concernés par la propriété. Chaque donnée représente les informations liées à un objet : une donnée $D_i$ est le résultat du produit cartésien des attributs $A_{ij}$ dont la mise à jour est liée à un objet concerné par la propriété. Ces attributs sont évalués à partir des attributs de la représentation canonique de l'objet.

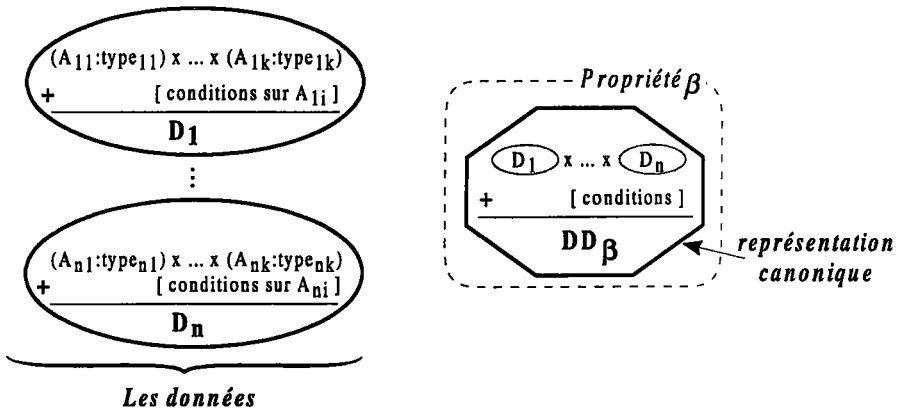



***Figure II.6.** Définition d'une propriété.*

Notons qu'une même donnée peut avoir une évaluation différente d'un objet à l'autre. Prenons par exemple la propriété 3D qui définit si un objet est posé sur un autre. Cette propriété peut être entièrement caractérisée par deux données : les zones de contact des deux objets i.e. la surface posée sur et la surface réceptrice. Une condition est définie sur ces données pour caractériser la propriété : les surfaces doivent être confondues. Dans la majorité des cas, tout objet pouvant se poser sur un autre est capable de fournir la donnée posée sur, mais chaque objet peut en donner une interprétation différente.

Les conditions formant la représentation canonique de la propriété ne sont pas nécessairement des conditions sur les données, mais on peut envisager des conditions sur la propriété même. En effet, une propriété est caractérisée aussi par des règles de manipulations : symétrie, réflexivité ...

La connaissance traduite pour décrire les propriétés peut être assimilée à celle pour les objets si l'on considère les similitudes dans la description des représentations canoniques. En effet, des informations pertinentes sur les caractéristiques des objets et des propriétés sont fournies ainsi qu'un ensemble de conditions particularisant ces informations les unes par rapport aux autres. De manière générale, on peut envisager de traiter une seule famille d'objets dans le système, cette famille regroupant les objets d'application et les propriétés sur ces objets d'application.

### 3.2.Les comportements.

Les connaissances modélisées sont statiques : le modèle obtenu est avant tout descriptif. Nous pensons qu'il se produit une perte d'informations lorsque l'utilisateur décrit le modèle de son application. En effet, l'utilisateur imagine forcément le fonctionnement de l'application avant de la décrire : il sait comment un cercle pourra être créé, mais il va d'abord décrire la représentation du cercle (son modèle), puis il va décrire les mécanismes de dialogue du système pour signifier les modes

de création qu'il envisage. L'architecture du système est un barrage pour garantir que l'utilisateur non expérimenté (en ce qui concerne le système) arrive à traduire exactement le fonctionnement qu'il envisage pour son application. De plus, il se produit une redondance dans les actes de l'utilisateur, car il doit réfléchir au fonctionnement de son application au début de sa spécification pour modéliser correctement les objets, puis au cours de sa spécification pour décrire la suite d'interactions à mettre en œuvre pour obtenir un dialogue adéquat.

Plutôt que de devoir interpréter le fonctionnement qu'il envisage, il est plus judicieux de laisser l'utilisateur décrire le fonctionnement tel qu'il le conçoit. En effet, la description des comportements des objets permet de servir de base pour des développements plus spécifiques au système et que l'utilisateur n'a pas à connaître : ce n'est plus le système qui régit le mode de pensée de l'utilisateur.

Dans cet objectif, nous introduisons le concept des comportements permettant à l'utilisateur de décrire le fonctionnement de son application sans qu'il ait à se soucier de la gestion même du système : le but est de transmettre les connaissances de l'utilisateur à l'état brut et de laisser le système résoudre les problèmes qui lui sont propres et utiliser ses propres concepts. Il s'agit pour l'utilisateur de décrire un objet par un couple données/comportements [GAR 95b ; MAR 95 ; GAR 96a]. Nous distinguons dans les parties suivantes trois types de comportements standard correspondant à trois types de connaissances que l'utilisateur peut transmettre : connaissances sur l'objet face à son entourage, connaissances sur l'évolution de l'objet et connaissances sur la création de l'objet.

### *3.2.1.La traduction.*

Un objet est défini selon une sémantique particulière correspondant à une certaine apparence et un certain état de l'objet qui ne sont pas nécessairement explicites pour certaines manipulations [GAR 96a]. Or lorsque l'utilisateur décrit un objet, que ce soit un objet d'application ou bien une propriété, il décrit uniquement sa représentation alors qu'il possède une connaissance plus étendue sur l'objet quant à son interprétation dans le "monde". C'est cette connaissance que l'on souhaite modéliser désormais. Cette notion est très importante, car elle permet de définir un objet par rapport au contexte dans lequel il se trouve. L'objet n'est pas modifié fondamentalement, mais il est interprété de différentes manières.

Pour décrire ce comportement particulier des objets, nous introduisons la notion de ***réacteur***. Il s'agit d'utiliser ce concept pour fournir à l'utilisateur un mode de représentation des connaissances dédié à l'interprétation contextuelle des objets. On associe les réacteurs aux objets comme on leur associe une représentation. Un objet possède autant de réacteurs que nécessaire.

Un réacteur a un rôle de traducteur d'objet. On différencie deux modes de traduction en fonction du contexte : on interprète un objet pour l'opérateur ou bien pour d'autres objets (objets d'application ou propriétés). Par conséquent, on définit deux types de réacteurs :

- les ***réacteurs extrinsèques***, pour capturer la connaissance sur l'interprétation des objets pour les utilisateurs de l'application ;

- les ***réacteurs intrinsèques***, pour représenter la connaissance sur la traduction d'un objet pour d'autres objets.

Les réacteurs extrinsèques définissent les connaissances sur un objet pour aider un opérateur à comprendre cet objet selon le contexte. Nous illustrons le rôle de ce réacteur à travers la figure II.7. Au moment de décrire un objet de type $\alpha$, le concepteur peut lier à la représentation canonique repérée par son domaine $DD_\alpha$, un réacteur extrinsèque $RE_i$ chargé de décrire, pour l'opérateur, l'objet d'une certaine manière. On représente sur la figure l'objet interprété sous la forme d'une donnée $D_i$ représentant un ensemble d'attributs.

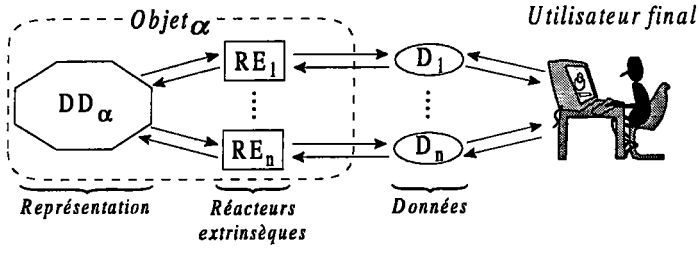



***Figure II.7. Réacteurs extrinsèques.***

Comme on assimile désormais une propriété à un objet (cf. *3.1.2. Propriétés*), on lui associe également des réacteurs extrinsèques. Le but de ces réacteurs est toujours le même, à savoir traduire des informations sur la représentation pour l'utilisateur. Considérons par exemple la propriété de perpendicularité. Lorsqu'une telle propriété est détectée, on peut trouver utile d'en informer l'opérateur. Cette opération revient bien à traduire la propriété en des termes compréhensibles pour l'utilisateur, en utilisant la connaissance d'un réacteur extrinsèque associé à la propriété.

Le réacteur extrinsèque répond au besoin de satisfaire l'opérateur. Cette décomposition permet de regrouper les connaissances utiles à un instant donné : on rencontre trop souvent dans les systèmes de CFAO, des applications qui noient les utilisateurs d'informations peu pertinentes pour eux. Un exemple simple de réacteur extrinsèque peut être la description d'un affichage particulier : le réacteur renferme les éléments nécessaires à l'exploitation de la représentation de l'objet concerné pour afficher l'objet d'une manière appropriée. Nous verrons dans le chapitre 3 que ces réacteurs ne se restreignent pas à la gestion de simples affichages d'informations, mais qu'ils peuvent intégrer des mécanismes d'aides plus puissants (cf. Chapitre 3 3.2. Le lien avec les réacteurs extrinsèques). D'ailleurs, la figure II.7. mentionne des allers et retours entre l'utilisateur et la représentation via le réacteur extrinsèque : il se produit des échanges d'informations.

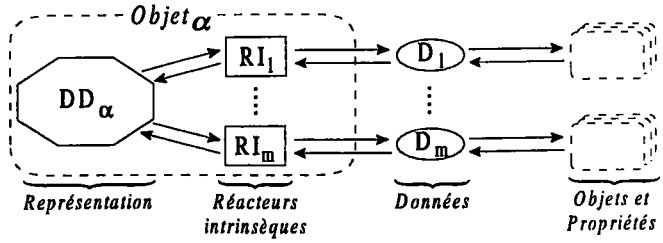



***Figure II.8. Réacteurs intrinsèques.***

Le réacteur intrinsèque a un rôle similaire au réacteur extrinsèque : il se base sur la représentation de l'objet pour construire une donnée. La différence fondamentale est que cette donnée est destinée à un ensemble formé d'objets d'application et de propriétés. Cet ensemble peut n'être constitué que d'un seul élément. La figure II.8. illustre le fonctionnement des réacteurs intrinsèques.

Lorsque le concepteur décrit un objet, il sait quel rôle il va jouer dans son application, et en particulier il connaît les dépendances avec les autres entités de l'application. Cette connaissance particulière sur le comportement d'un objet est capturée par les réacteurs intrinsèques. Un réacteur intrinsèque $RI_j$ traduit la représentation $DD_\alpha$ d'un objet $\alpha$ en une donnée $D_j$. Cette connaissance est essentielle pour :

- aider des objets à être créés. Certains objets ont besoin d'informations d'autres objets pour être eux-mêmes définis. Par exemple, un trou (qui est un objet à part entière) nécessite la donnée de la face à trouer d'un objet ;
- aider des propriétés à fonctionner. Nous avons expliqué au *3.1.2. Propriétés* que les propriétés ont besoin de données fournies par les objets concernés par ces propriétés. Les connaissances acquises par le réacteur intrinsèque préparent l'objet pour la propriété dans le cas de la détection de la propriété et elles interprètent la mise en place de la propriété pour un objet dans le cas de la création [MAR 95 ; GAR 96a].

S'il s'avère essentiel pour l'utilisateur d'être informé des transactions dues aux connaissances d'un réacteur intrinsèque, on peut toujours envisager d'associer à ce réacteur intrinsèque un réacteur extrinsèque chargé de communiquer avec l'utilisateur : les connaissances capturées par le réacteur extrinsèque sont utilisées pour traduire à l'utilisateur la donnée du réacteur intrinsèque. Le réacteur intrinsèque dans ce cas précis fournit un contexte de déclenchement pour le réacteur extrinsèque. Prenons le réacteur intrinsèque $RI_\perp$ d'un segment chargé de définir sa droite support, donnée nécessaire à la propriété de perpendicularité. On peut envisager un réacteur extrinsèque $RE_\perp$ associé à $RI_\perp$ et dont le rôle est de montrer à l'utilisateur cette droite support. Si la propriété est détectée, $RE_\perp$ est utilisé pour informer l'utilisateur de la détection de la perpendicularité.

Comme une propriété est modélisée au même titre qu'un objet d'application, des réacteurs intrinsèques peuvent également lui être associés. Cette approche signifie que des informations propres à une propriété peuvent être traduites à des objets ou à d'autres propriétés. Cette possibilité est tout à fait envisageable. En effet, une propriété peut résulter de la composition de plusieurs autres. Ainsi, elle a besoin d'informations connues de ces autres propriétés, informations que des réacteurs intrinsèques associés à ces propriétés transmettent via la donnée qu'ils gèrent. De plus, un objet peut être défini à l'aide d'une propriété qui serait trop coûteuse à définir en tant que condition de validité sur les attributs de l'objet : par exemple, on peut définir un arrondi grâce à un arc et des propriétés de tangence entre cet arc et des segments. La propriété de tangence peut fournir via un réacteur intrinsèque la donnée nécessaire à l'arrondi.

De manière générale, les réacteurs représentent des passerelles entre les objets pour lesquels ils sont définis et l'environnement qui les entoure i.e. aussi bien les objets et les propriétés qui sont internes à l'application, que l'utilisateur qui peut être considéré comme un élément externe à l'application.

### *3.2.2.L'évolution.*

Les systèmes de CFAO permettent à un utilisateur de définir un prototype virtuel d'un produit. L'utilisateur ne construit que rarement ce prototype en une étape, parce qu'il est trop complexe : sa construction est progressive. Les objets qu'il définit deviennent si précis que leur sémantique change totalement, à tel point que leur représentation initiale n'est plus cohérente : les objets ont évolué. Cette connaissance sur l'évolution des objets, le concepteur de l'application en a conscience, mais il ne la transmet pas explicitement. La conséquence est que le système manque de contrôle sur la validité des objets. Prenons le cas d'une rainure que l'opérateur déplace sur la face supérieure du prototype qu'il est en train de construire et analysons la validité du déplacement (cf. figure II.9.).

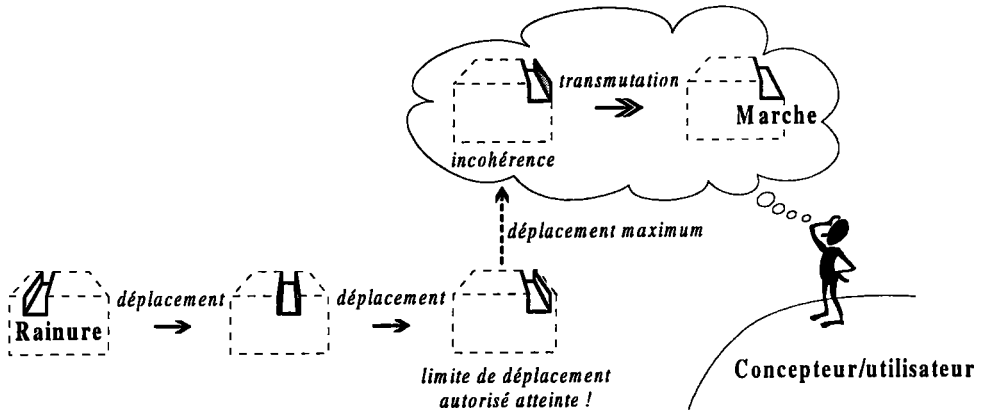



***Figure II.9.** Problème de validité d'une rainure au cours d'un déplacement.*

Le cas où la rainure est déplacée jusqu'au bord de la face latérale du prototype pose problème. La solution courante est d'interdire les débordements dans les déplacements : on définit des tolérances (intervalles cohérents autorisés) sur le déplacement. Tout le problème est de définir des tolérances acceptables pour tout type d'utilisateur. Il serait pourtant plus judicieux et plus puissant de rendre cet objet valide le plus longtemps possible et d'envisager des solutions acceptables dans les cas d'incohérence. Ainsi dans notre exemple, le problème pourrait être résolu en transformant la rainure en une marche, ce qui est alors topologiquement correct. En conséquence, l'objet de type rainure est remplacé dans le modèle de l'application par un objet de type marche. Nous appelons ce phénomène de transformation une transmutation.

On rencontre un second type d'évolution pour un objet lorsque celui-ci se voit muni d'informations plus précises que sa représentation ne peut supporter. Ce comportement est caractéristique des spécialisations dans la sémantique des objets. Par exemple, un arrondi et un congé sont deux objets différents en mécanique et pourtant ce sont initialement de "simples" arcs (cf. figure II.10.).

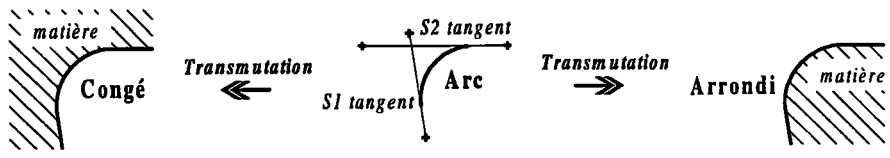



***Figure II.10.** Exemple de spécialisation d'un arc par transmutations.*

Nous introduisons le concept de ***transmuteur*** pour permettre au concepteur de décrire de tels types de comportements. La figure II.11 illustre le fonctionnement des transmuteurs $T_i$ associés à la représentation d'un objet de type $\alpha$. Les transmuteurs renferment toutes les connaissances utiles pour décrire les conditions de modification de la représentation d'un objet pour obtenir un nouveau type d'objet i.e. une nouvelle représentation.

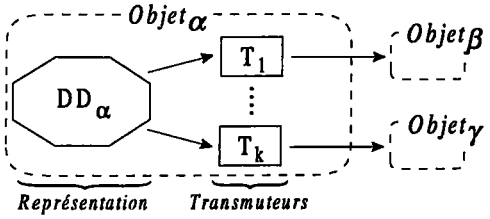



***Figure II.11.*** *Transmuteurs.*

Ici, on introduit un changement fondamental de l'objet : sa représentation n'est plus la même, car de nouvelles données ont été introduites. Soit ces données sont incohérentes avec la sémantique même de la représentation de l'objet, soit la représentation ne peut les prendre en compte car elles sont trop précises. L'objet initial n'est pas nécessairement détruit, mais il ne s'agit pas d'un réacteur qui traduit seulement la représentation d'un objet. Le transmuteur est caractérisé par la création implicite ou contextuelle d'objets. Il apporte des solutions au problème de la multi-modélisation, car à partir d'un objet initial il permet de définir des objets dédiés à des modèles plus spécialisés.

Tout comme un objet d'application, une propriété accepte des transmuteurs dans sa modélisation : on autorise une propriété à évoluer. Par exemple, une propriété de placement varie selon le contexte dans lequel elle se situe. Prenons le cas d'un architecte qui définit une propriété de placement permettant de conditionner le placement de meubles dans une maison. Cette propriété peut évoluer vers des propriétés plus précises selon la pièce qu'il étudie (salle de bain, cuisine ...). Le domaine de définition de la propriété est alors différent : il existe par exemple des données spécifiques au placement dans une cuisine (règle du triangle équilatéral ...).

Le concepteur a les connaissances nécessaires pour décrire les transmuteurs. De plus, au lieu de chercher à restreindre les manipulations de l'opérateur, on laisse la possibilité au concepteur de décrire ce qui se passe et décrire ainsi des solutions acceptables pour l'opérateur. L'avantage est de garantir le plus longtemps possible la validité du système sans que l'opérateur n'ait à s'en préoccuper.

### *3.2.3.La création.*

Un objet d'application est défini par sa représentation pour le système, mais pour l'utilisateur, il l'est par les différentes façons de le créer. En effet, il n'existe pas un unique mode de création : on peut créer un cercle en donnant son centre et son rayon, ou en donnant trois points de passage, ou encore en donnant le centre et un point de passage... La connaissance sur le mode de création est un comportement essentiel pour l'objet, car il se trouve être le moyen de communication entre l'objet et l'opérateur. On introduit le concept de ***producteur*** pour capturer cette information.

Pour créer un objet, l'utilisateur doit donner des attributs qui représentent des éléments essentiels pour définir l'objet. Le système garantit que ces attributs sont correctement définis. L'ensemble formé par ces attributs et les conditions de validité qui peuvent porter dessus forme un domaine de définition de l'objet à créer. Ce domaine est caractéristique du mode de création, mais il ne correspond pas systématiquement à la représentation canonique de l'objet. On associe ce domaine particulier à un producteur. Un objet dispose alors d'autant de producteurs que de modes de création.

Ainsi que le montre la figure II.12., le producteur $P_i$ se charge de récupérer les attributs nécessaires à un mode de création et de transmettre ces informations à la représentation canonique $DD_\alpha$ de l'objet $\alpha$ pour que $DD_\alpha$ soit définie. On distingue deux catégories d'attributs que le producteur définit : des objets d'application donnés par l'utilisateur et des données qui constituent des informations données par des réacteurs intrinsèques ou par l'utilisateur.

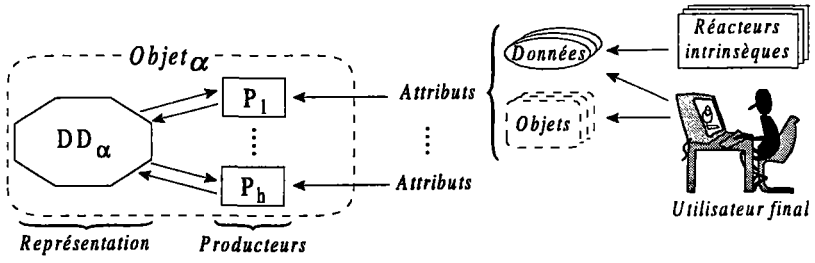



***Figure II.12. Producteurs.***

Tout comme pour les autres comportements, mais de manière plus évidente, les producteurs peuvent être associés à des propriétés. En effet, une propriété peut posséder différents modes de création reposant sur une même représentation (domaine de définition) : on peut définir une tangence en donnant directement le point de tangence ou encore en donnant les deux objets tangents.

Dans les systèmes de CFAO, les modes de création se contentent de moins en moins de saisir seulement les attributs de création. Ils intègrent désormais des mécanismes plus puissants pour faciliter la tâche de l'utilisateur et lui éviter une perte de temps : ces mécanismes sont essentiellement des aides comme des affichages particuliers (affichage en élastique ...) et des détections. On constate que ces informations sont induites par les réacteurs et les transmuteurs. Il en résulte que les producteurs peuvent être liés aux autres comportements afin que leurs connaissances sur les modes de création soient enrichies par les connaissances apportées par ces autres comportements. Le producteur décrit, par cette occasion, un contexte de déclenchement pour les autres comportements.

### 3.3.Synthèse.

Le comportement, qu'il soit réacteur, transmuteur ou producteur agit au moins sur un sous-ensemble du domaine de définition de l'objet. C'est pour cette raison qu'on définit un objet par le couple données/comportements. Devant la multitude de modèles qui peuvent exister pour un objet, on se ramène ainsi à deux ensembles de modèles qui nous paraissent former un couple idéal : les modèles descriptifs et les modèles de fonctionnement [GAR 91 ; MAR 95].

Nous nous sommes attachés à définir des comportements sur des objets. Il peut exister dans les applications des familles d'objets qui agissent selon un mode de fonctionnement commun. Il semble donc que les comportements puissent s'appliquer non seulement à un type d'objet, mais aussi à une famille toute entière d'objets.

Les connaissances décrites par les comportements ne sont pas nouvelles, mais ce qui est primordial, c'est que l'on dispose d'un moyen pour les représenter nécessitant peu d'efforts pour le concepteur, car il respecte son mode de pensée. Le concepteur, utilisateur de l'outil de description de l'application, définit ce qu'est un objet et décrit comment il va se comporter sans se soucier un instant du modèle de dialogue utilisé par le système. On montre une grande utilité des comportements pour l'étape de description du modèle générique. Nous étudions dans la partie suivante leur rôle dans la construction du modèle de dialogue.

## 4. LA GÉNÉRATION DU DIALOGUE.

Un nouveau concept pour capturer la connaissance de l'utilisateur a été défini : le concept des comportements. On peut disposer maintenant d'un modèle générique de description de l'application plus complet. Notre propos consiste à étudier l'impact de ce modèle pour la description du modèle de dialogue de SACADO que nous avons présenté (cf. 2.1. Les concepts de base).

Nous présentons dans la suite comment nous envisageons la conception d'un dialogue "idéal". Nous entendons par "idéal" le fait que le dialogue décrit correspond aux attentes de l'opérateur. Nous expliquons notamment nos intentions en matière de génération du dialogue à partir des descriptions effectuées par le concepteur. Ensuite, nous insistons sur l'utilisation des comportements pour déduire des informations pertinentes sur le dialogue à construire : nous montrons en particulier comment le modèle de dialogue peut être déduit. Enfin, nous présentons un exemple d'implémentation réalisée pour montrer la validité de notre propos.

### 4.1.Une description duale.

Le problème que nous tentons de résoudre concerne la construction d'un dialogue adapté à l'opérateur. Il s'agit de disposer :

- d'un modèle dialogue permettant de tenir compte des opérateurs et de leurs raisonnements ;
- de méthodes d'acquisition et d'interprétation des connaissances du concepteur pour définir le dialogue à mettre en place.

La première approche présentée au début de ce chapitre a permis de définir un modèle de dialogue complet. Le système SACADO en est la preuve, car il intègre ce modèle pour servir de noyau aux différentes applications de CFAO développées au laboratoire parmi lesquelles [GAR 95a ; PIP 95]. Devant la difficulté des dialogues que l'on peut avoir à décrire dans de telles applications, le formalisme de description a été défini comme méthode d'acquisition des connaissances du concepteur sur le dialogue [MAR 95]. Toutefois, cette approche ne nous satisfait pas pleinement dans la mesure où le concepteur doit connaître le modèle de dialogue pour décrire son application.

Par conséquent, une seconde approche est envisagée. Cette étude s'intègre dans le successeur en 1995 de SACADO, le projet REGAIN. Nous orientons nos travaux vers une description duale du dialogue qui correspond à une extension de la première approche (cf. figure II.13). Nous tentons de répondre à un nouveau critère : tenir compte de la diversité des utilisateurs qui peuvent intervenir dans la description du dialogue.

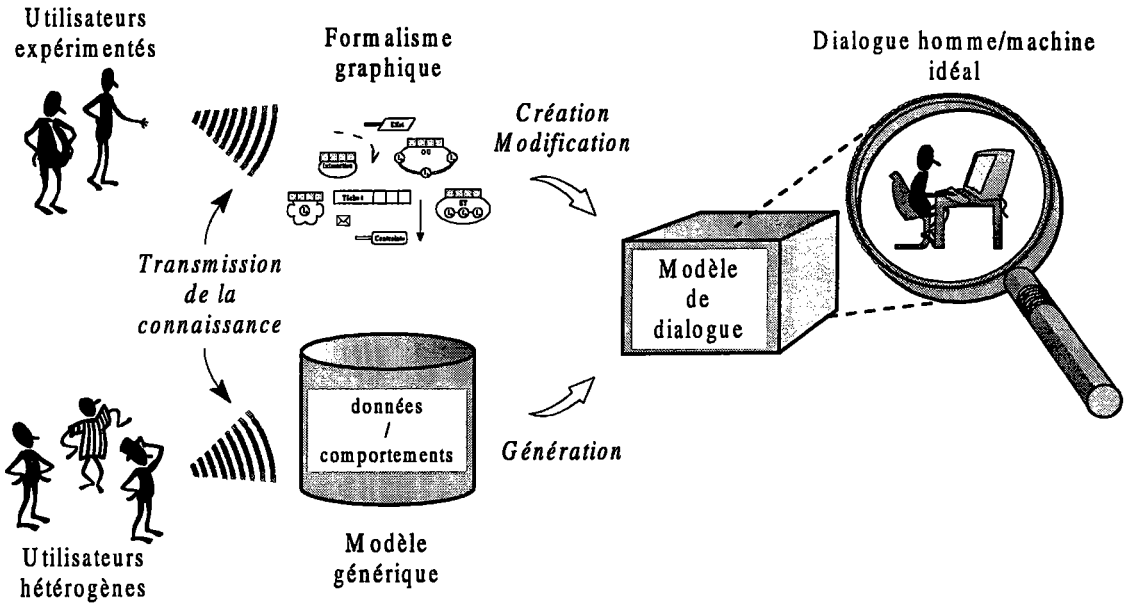



***Figure II.13.*** *Paradigme de la description du dialogue.*

La solution qui nous paraît la plus acceptable consiste à développer deux méthodes en parallèle :

- une méthode dédiée à des utilisateurs expérimentés dans le modèle de dialogue. À l'aide du formalisme graphique, le dialogue est décrit en des termes compréhensibles pour le modèle de dialogue. L'utilisateur peut à son gré modifier le modèle à tout moment via le formalisme. On garantit une mise à jour immédiate du dialogue ;
- une méthode dédiée à des utilisateurs non expérimentés dans le modèle de dialogue. Ces utilisateurs ont besoin d'un mode de description qui leur corresponde. Nous utilisons donc le modèle générique basé sur la description d'un objet par sa représentation et ses comportements. Cette représentation des connaissances que nous avons évoquée dans la partie précédente (cf. 3. De nouvelles connaissances pour le dialogue) nous paraît d'une part, suffisamment complète pour contenir toutes les informations utiles pour le modèle de dialogue et d'autre part, naturelle à décrire par n'importe quel utilisateur. Il s'agit d'interpréter le modèle générique pour déduire le modèle de dialogue.

Les deux approches proposées sont complémentaires. Dans la mesure où le dialogue déduit repose sur le même modèle de dialogue que celui proposé à l'utilisateur expérimenté, il est possible pour cet utilisateur d'intervenir pour adapter le dialogue et l'enrichir pour des besoins particuliers. C'est cette dualité qui nous permet d'envisager que n'importe quel utilisateur puisse concevoir sa propre interface, voire l'architecture complète de son application. La méthode de génération basée sur le modèle générique permet d'apporter des dialogues de base en accord avec l'opérateur et à partir desquels le concepteur d'interfaces expérimenté pourra construire le dialogue complet. Cet utilisateur expérimenté

(le concepteur) est en quelque sorte guidé dans la construction du dialogue, puisqu'un opérateur peut décrire globalement le dialogue qu'il désire utiliser en décrivant les comportements des objets de son application.

### 4.2.L'interprétation des comportements.

Les comportements renferment des connaissances exploitables pour le dialogue. Nous proposons de montrer dans quelles mesures on peut déduire les mécanismes associés au dialogue à partir des producteurs, des réacteurs et des transmuteurs.

L'utilisateur, dans ses manipulations, est guidé par trois types principaux de dialogue : la création, l'aide et la modification. Ce sont les mécanismes décrivant ces dialogues que l'on est capable de déduire. Nous en expliquons les points principaux dans les deux parties suivantes. Nous traitons des dialogues entraînant des modifications des objets à travers les dialogues d'aides (cas des détections).

#### *4.2.1.Les dialogues de création.*

La description des dialogues de création est sans nul doute l'une des primitives de dialogue les plus importantes dans une application, puisque c'est par elle que tout commence. Le modèle de dialogue de SACADO décrit pour chaque création un menu de déclenchement ayant certaines compatibilités et une action globale décrivant la séquence d'interactions et d'actions non interactives à effectuer.

Nous nous intéressons dans cette partie à des dialogues de création simples i.e. ne se ramenant qu'à une capture des informations nécessaires à la définition des objets. Les mécanismes qui peuvent être inclus au processus de création (détections ...) sont des dialogues d'aides particuliers qui sont détaillés dans la partie suivante : nous expliquerons dans quelles mesures ils pourront être intégrés aux dialogues de création.

Nous considérons que la création des objets d'application (cercle, segment, cylindre ...) et la création des propriétés (tangence, assemblages particuliers ...) possèdent un fonctionnement similaire ; par conséquent, la génération du dialogue associé est également identique. Toute la connaissance modélisée à propos des modes de création, que ce soit au sujet des objets d'application ou des propriétés, est décrite explicitement à l'aide des producteurs et plus implicitement à l'aide de réacteurs intrinsèques et de transmuteurs.

Un producteur représente un mode de création particulier. Par conséquent, la première information que l'on peut déduire, c'est la mise en place d'un dialogue de création. Cette étape correspond à la définition d'un menu de création dédié à ce mode de création. Le système sait à ce moment précis que ce menu permet de créer un objet. On peut donc également déduire que ce menu est local à toute interaction demandant ce type d'objet. Il reste à déduire l'action globale associée à ce menu.

Une création est caractérisée par un ensemble d'éléments que l'opérateur doit fournir. Ceci se traduit au niveau du modèle de dialogue de l'action globale par la spécification d'interactions. On peut déduire ces interactions en analysant un producteur. En effet, nous avons montré au 3.2.*3. Création* qu'un

producteur contient l'ensemble des attributs nécessaires à un mode de création particulier. Ainsi sans le savoir, le concepteur non expérimenté décrit des interactions.

De plus, pour décrire précisément les attributs de création, le concepteur donne, toujours par l'intermédiaire du producteur, des conditions de validité sur l'ensemble des attributs. À l'aide d'un analyseur, ces conditions donnent lieu à une série de contraintes propres à la saisie de chaque attribut. Ces contraintes correspondent au niveau dialogue à des contraintes sur les interactions. La figure II.14 donne un exemple de contraintes déduites de conditions de validité d'un producteur.

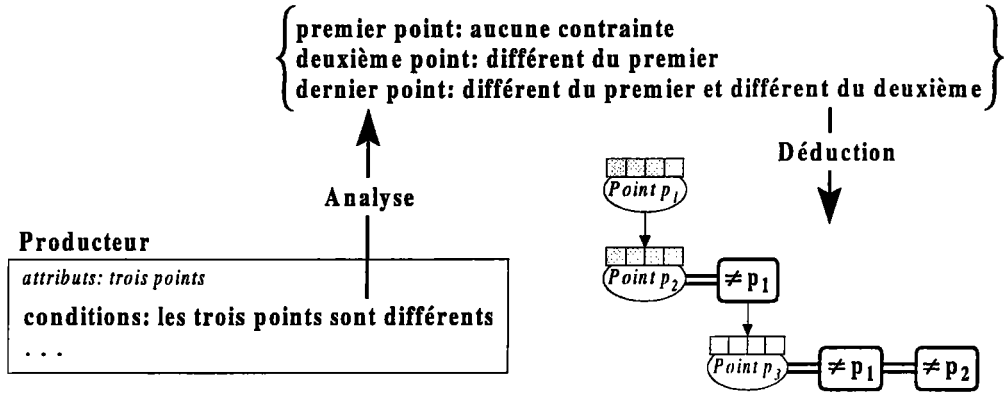



***Figure II.14.*** *Exemple de déduction de contraintes sur les interactions.*

La manière de donner les attributs dans le producteur peut donner une idée de l'ordre dans lequel les interactions doivent s'organiser. Si le concepteur ne donne aucune information, on en déduit qu'il ne souhaite pas donner de priorités particulières. Il est toujours possible de créer des interactions composées à l'aide des opérateurs de composition (ET, OU, ENSEMBLE) pour donner plus de souplesse dans la donnée des attributs.

Le producteur est défini pour décrire le mode de création d'un seul objet : il intègre les liens entre ses attributs propres à la création et ceux de la représentation canonique de l'objet. Cette information constitue le dernier élément de base permettant d'achever la description de l'action globale de création. En effet, nous avons déjà déduit les séquences d'interactions indiquant les interventions de l'utilisateur lors de la création ; il ne reste plus qu'à déduire les actions non interactives à effectuer i.e. les mises à jour de la représentation canonique de l'objet à partir des attributs retournés par les interactions.

Le réacteur intrinsèque joue également un rôle dans la déduction de l'action globale de création d'un objet (objet d'application ou propriété). Son rôle est de contenir les connaissances utiles sur un objet pour d'autres objets. Ainsi, il peut arriver qu'un producteur requiert, pour le mode de création qu'il représente, un attribut qui peut être fourni par un objet via un des réacteurs intrinsèques de cet objet. Il s'ensuit au niveau du dialogue l'ajout d'une interaction demandant l'objet à partir duquel on peut évaluer l'attribut réellement demandé. Cette interaction est suivie d'une action non interactive permettant d'évaluer l'attribut : l'opération est rendue possible, car elle est décrite par le réacteur intrinsèque. Prenons l'exemple de la figure II.15 : le producteur décrit un mode de création d'un objet à partir d'une surface. Globalement, on déduit une interaction demandant une surface. Cette étape de dialogue va être enrichie grâce au réacteur $RI_j$ de l'objet $\alpha$, car il sait traduire un objet $\alpha$ en une surface.

Ainsi, le dialogue demandant une surface peut aussi se ramener à une interaction demandant un objet $\alpha$ qu'une action non interactive transformera en une surface. En définitive, le dialogue déduit revient à une composition OU de deux interactions (équivalentes par leur résultat), ce qui signifie que l'utilisateur a le choix dans sa manière de donner une surface entre deux possibilités.

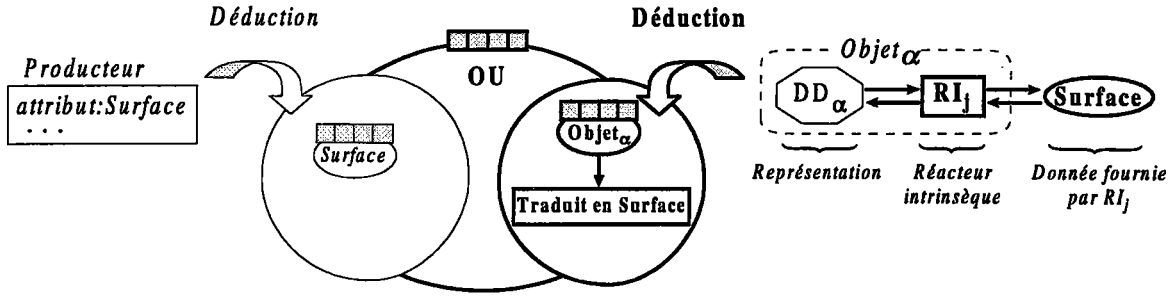



***Figure II.15.*** *Enrichissement du dialogue de création à partir d'un réacteur intrinsèque.*

Le transmuteur décrit à sa manière un mode de création. Il regroupe la connaissance sur l'évolution d'un objet allant jusqu'à entraîner un changement de type : il contient la notion de contexte permettant de décrire avec précision dans quelles conditions un objet peut se transformer en un autre. On peut interpréter cette connaissance pour proposer à l'utilisateur de créer un objet à partir d'un autre objet en modifiant certains paramètres. Le transmuteur contient l'ensemble des attributs associés à la représentation d'un objet $\alpha$ et décrit les modifications de ces attributs ainsi que d'autres attributs, qui ne sont pas nécessairement associés à l'objet $\alpha$, pour obtenir un objet $\beta$. Par conséquent, pour définir le dialogue de création d'un objet $\beta$, on peut déduire les interactions éventuellement contraintes et les actions non interactives nécessaires. Considérons une définition simplifiée d'un transmuteur pour décrire l'évolution d'un arc en arrondi. Ce transmuteur demande qu'à chaque extrémité de l'arc se trouve un segment tangent et que la matière se trouve du côté du cercle support de l'arc (cf. figure II.10 pour voir un exemple d'arrondi). On peut déduire des dialogues à partir de ce transmuteur :

- un dialogue demandant à l'opérateur un arc dont la contrainte est de posséder à chaque extrémité un segment ; il s'agit d'une interaction contrainte. Le système se charge de modifier tous les autres paramètres pour obtenir effectivement un arrondi (rendre les segments tangents et définir la matière), ceci se traduit sous la forme d'actions non interactives ;
- un dialogue dans lequel les paramètres pour créer un arrondi sont demandés à l'opérateur. Il s'agit d'une suite d'interactions contraintes qui se termine par une action non interactive de création.

Ce ne sont que deux exemples très simplifiés des dialogues que l'on peut déduire à partir des connaissances acquises par le transmuteur. La première solution est assez simple du point de vue déduction, puisqu'on part d'une interaction demandant l'objet initial (i.e. avant transmutation) et qu'on ajoute les traitements décrits par le transmuteur sous forme d'actions non interactives. La seconde solution laisse plus de liberté à l'utilisateur, mais la déduction du dialogue demande une analyse plus minutieuse du transmuteur car il faut davantage décomposer sa connaissance.

Nous avons vu qu'on peut déduire des dialogues de création à partir des producteurs, des réacteurs intrinsèques et des transmuteurs. Pour aller plus loin dans la génération, on peut envisager de déduire

des dialogues de création très généraux pour ne pas imposer un mode de création particulier à l'utilisateur. Cette solution consiste à composer les dialogues déduits précédemment (lorsqu'ils concernent la création d'un même objet), d'où le recours notamment aux opérateurs de composition d'interactions. Il existe des cas d'ambiguïtés qui peuvent persister : pour un état du dialogue, on ne pourra pas toujours déterminer clairement le mode de création choisi par l'opérateur (attributs de création communs ...). Dans ce cas, on peut prévoir soit l'intervention d'un concepteur expérimenté pour lever ces cas au moment de la description du dialogue, soit l'intervention de l'opérateur pour préciser ses choix.

#### *4.2.2.Les dialogues d'aides.*

Les dialogues aidant l'opérateur ont une place importante: ils fournissent des dialogues adaptatifs. Les aides peuvent intervenir à tout moment dans les dialogues. L'aide contextuelle, i.e. l'aide intervenant intuitivement pendant l'intervention de l'utilisateur, nous paraît intéressante à exploiter dans la mesure où l'utilisateur est guidé dans ses interactions sans être contraint d'en faire la demande. Du point de vue modélisation du dialogue, ce phénomène d'aides se traduit par des effets associés aux interactions.

On peut déduire ces aides à partir des réacteurs (extrinsèques et intrinsèques) et des transmuteurs. La première information à déduire est le contexte de déclenchement de l'aide. Il faut pouvoir déterminer durant quelle interaction un effet va fonctionner. Ce sont les associations entre comportements qui contiennent ces informations. Par exemple, prenons un producteur auquel un réacteur extrinsèque a été associé. Comme à partir du producteur on déduit un dialogue de création, on en déduit que le réacteur extrinsèque apporte une aide pendant la création. À chaque intervention de l'utilisateur, l'effet déduit du réacteur va tenter de se déclencher grâce à la représentation canonique de l'objet définie provisoirement par les informations temporaires provenant de l'interaction. C'est le réacteur lui-même qui va permettre de définir avec précision à quel moment il intervient dans le dialogue de création. Il agit sur la représentation de l'objet (sur un ensemble de ses attributs). Par conséquent, si les attributs qu'il utilise ne sont pas définis (même provisoirement) pour une interaction donnée, l'aide ne pourra avoir lieu pendant cette interaction.

Transmuteurs, réacteurs intrinsèques et réacteurs extrinsèques fonctionnent de manière identique : ils utilisent la représentation canonique de l'objet auquel ils sont associés, même si elle est provisoire, pour décrire des traitements. Par contre, c'est le type d'effet déduit dans le dialogue qui est différent :

- détection de changement de type de l'objet en construction grâce au transmuteur ; comme il traite de l'évolution des objets, le transmuteur contient toutes les conditions sur le contexte et sur la représentation de l'objet pour définir à quel moment l'objet change. Proposer une telle détection pendant le dialogue de création permet à l'utilisateur de construire des objets complexes de manière simple et implicite sans avoir recours à des menus complexes ;
- détection de propriétés sur l'objet en construction grâce au réacteur intrinsèque. Dans ce cas, le réacteur traduit un objet pour gérer une propriété. Pendant une interaction, l'effet consulte la représentation provisoire de l'objet et tente d'évaluer la donnée nécessaire à la gestion de la propriété. Comme pour le dialogue déduit du transmuteur, l'utilisateur voit sa tâche se simplifier ;

- informations sur l'objet grâce au réacteur extrinsèque. Ce dernier traduit un objet pour l'utilisateur. Il contient les traitements nécessaires à la mise en évidence d'informations pertinentes pour l'utilisateur. L'effet déduit apporte à l'utilisateur une aide qui s'oriente plus vers un effort pour expliquer que pour fournir des traitements automatiques.

L'aide déduite et destinée à l'utilisateur peut se retrouver également dans des actions non interactives. Par exemple, lorsqu'un objet est créé, la visualisation qui en est faite est une aide pour l'utilisateur puisqu'il peut voir de manière claire ce qu'il a construit. Cette aide constitue l'étape finale du dialogue de création. C'est l'association d'un réacteur extrinsèque à un producteur qui permet cette déduction. Le concepteur décrit un producteur pour indiquer comment un objet est créé et peut préciser qu'à la fin cet objet soit visualisé en utilisant l'interprétation décrite par un réacteur donné.

### 4.3.Implémentation.

Afin de valider nos travaux, nous avons réalisé une application en C++ sur stations de travail SUN [GAR 96a]. Dans cette implémentation, nous avons voulu concevoir une application graphique interactive qui permet à un opérateur de manipuler un formalisme graphique de description de dialogues (cf. figure II.17 pour un exemple de manipulations). L'opérateur peut créer interactivement des objets géométriques symbolisant les différents composants de ce formalisme. Pour plus de simplicité dans la définition de la représentation de ces composants, nous avons choisi de reprendre comme objets géométriques ceux utilisés pour symboliser les composants du formalisme graphique de description du dialogue de SACADO (cf. figure II.4).

L'implémentation doit mettre en évidence la méthode de conception de l'application. Il s'agit de montrer comment un concepteur peut parvenir à spécifier cette application sans posséder de connaissances particulières sur la gestion du dialogue. Nous sommes partis de cette spécification pour mettre en œuvre une déduction des concepts inhérents au dialogue.

L'application prise pour exemple n'a que très peu d'importance pour valider notre approche en matière de génération du dialogue. En réalité, elle aurait pu porter sur n'importe quel domaine (réalisation de pièces paramétrées, description de circuits électroniques ...) : notre méthodologie est indépendante de son usage. Toutefois, le choix d'une application interactive est plus intéressant, car le dialogue à déduire n'en est que plus complexe à construire : c'est un bon moyen de valoriser la génération du dialogue.

Nous ne présentons pas dans cette partie le processus de génération, mais nous insistons sur la simplicité de la spécification du modèle face aux dialogues déduits (cf. Annexe A pour des compléments d'informations au sujet de la description informatique).

#### *4.3.1.Spécification du modèle.*

L'application choisie est intéressante de par le dialogue interactif qu'elle nécessite : créations d'objets graphiques, intervention de l'opérateur pendant les créations d'objets et aides à l'opérateur par des

mises en évidences (affichages ...) et des détections (de propriétés et de changements de type). Pour parvenir à un tel type de dialogue, nous avons utilisé un modèle générique dans lequel le concepteur non informaticien spécifie chaque type d'objets défini dans son application. Cette spécification consiste à décrire comment sont représentés les objets (représentation canonique), puis à leur ajouter tous les comportements que le concepteur souhaite leur voir attribuer i.e. comment ils peuvent :

- être créés ; ceci constitue la définition de producteurs ;
- se transformer ; il s'agit de décrire des transmuteurs ;
- recevoir des propriétés ; ce sont les connaissances capturées par les réacteurs intrinsèques ;
- être compréhensibles pour l'opérateur ; les réacteurs extrinsèques représentent ces informations.

Le concepteur est capable non seulement de décrire ces comportements, mais aussi d'indiquer à quel moment ces traitements (en particulier pour les trois derniers cas) peuvent se produire : il s'agit pour le concepteur de réaliser des associations entre les comportements.

De manière plus concrète, nous nous proposons de montrer la spécification du modèle générique à travers un exemple. Prenons le lien non typé, qui se trouve être un objet de l'application assez complet, car il intègre tous les types de comportements. Nous nous basons sur la figure II.16. pour illustrer cet exemple. Le lien non typé est un objet de l'application dont le but est d'effectuer une liaison entre deux objets du formalisme. On peut définir sa représentation canonique, à savoir les attributs nécessaires (deux objets d'application) et les conditions de validité sur les types d'objets mis en relation. Pour créer ce lien, on n'envisage qu'un mode de création correspondant directement à la représentation canonique : on définit le producteur $P_1$. Il décrit les objets nécessaires que l'opérateur doit fournir pour créer le lien : les objets qui ne sont pas des liens.

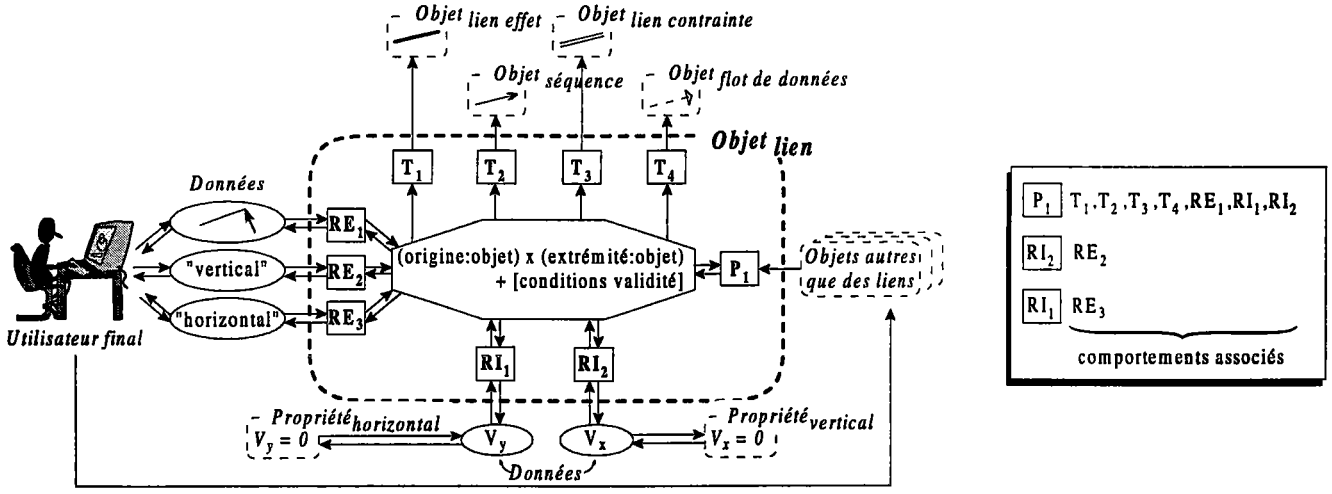



***Figure II.16.** Spécification du lien non typé.*

Rappelons que cette application permet de créer un outil graphique de manipulations du formalisme simplifié de SACADO. Ainsi, l'application propose différents types de lien en fonction des objets mis en relation : la séquence, le flot de données, le lien contrainte et le lien effet. Le lien non typé est un lien plus général qui ne tient pas compte du type de liaison. Nous voulons en fait pouvoir détecter automatiquement le type d'une liaison lorsque l'opérateur crée le lien non typé : ceci revient à

transformer le lien non typé en un lien typé. Ce comportement que l'on vient de décrire correspond exactement à la définition de transmuteurs. On définit quatre transmuteurs ($T_1$, $T_2$, $T_3$ et $T_4$) qui contiennent chacun les conditions de transformation vers les objets typés cités précédemment. Pour permettre une réelle détection de ces transmutations pendant la création du lien, on doit associer ces transmuteurs au producteur (cf. figure II.16).

De la même manière, on associe au producteur $P_1$ le réacteur extrinsèque $RE_1$ qui décrit un affichage en élastique et les deux réacteurs intrinsèques $RI_1$ et $RI_2$ pour gérer la détection de propriétés pendant la création du lien. Le réacteur $RE_1$ permet de traduire le lien à l'opérateur en cours de création. Le réacteur $RI_1$ traduit le lien en l'ordonnée $V_y$ du vecteur directeur du segment qui le représente pour permettre de gérer la propriété horizontal sur le lien. De la même manière, le réacteur $RI_2$ traduit le lien en l'abscisse $V_x$ du vecteur directeur du segment support pour gérer la propriété vertical.

Les réacteurs extrinsèques $RE_2$ et $RE_3$ sont définis avec le lien et associés respectivement à $RI_1$ et $RI_2$ pour visualiser les deux propriétés horizontal et vertical. Cette association des réacteurs extrinsèques aux réacteurs intrinsèques n'est pas correcte : elle a été réalisée uniquement en raison d'une restriction dans l'implémentation. En effet, nous avons considéré les propriétés comme des objets à part entière, mais seuls les objets d'application ont été décrits selon l'approche données/comportements. Notre objectif est de nous occuper des comportements pour les objets de l'application dans une première approche, puis d'étendre la méthode aux propriétés si l'approche est convaincante. Les propriétés sont caractérisées uniquement par leur représentation canonique i.e. un ensemble de données et des conditions sur cet ensemble pour valider la propriété (cf. figure II.6). La visualisation d'une propriété pose problème, car dans notre approche, cette connaissance devrait être à la charge d'un réacteur extrinsèque associé à la propriété. Or, nous ne définissons aucun comportement pour les propriétés dans cette implémentation. Pour contourner ce problème, nous avons choisi d'associer de tels réacteurs extrinsèques aux objets concernés par la propriété ($RE_2$ et $RE_3$ dans l'exemple de la figure II.16). Pour que ces réacteurs représentent un traitement associé aux propriétés, ils sont liés aux réacteurs intrinsèques décrivant les données nécessaires à la propriété ($RI_1$ et $RI_2$ dans l'exemple).

Le lien est ainsi défini par un couple données/comportements qui intègre des connaissances pertinentes sur le dialogue associé à sa manipulation. Il est défini tant du point de vue classique pour la modélisation, que du point de vue pratique pour l'utilisation. En résumé, le lien est décrit par :

- ses données : c'est un segment avec des conditions particulières sur l'origine et l'extrémité ;
- ses comportements : la création du lien requiert la saisie d'une origine et d'une extrémité cohérente avec l'origine ; pendant la création, interviennent des aides sous forme d'affichage en élastique, de détections de propriétés qui seront indiquées à l'opérateur et de détections de spécialisation de lien qui transformeront le lien en un lien spécialisé.

#### *4.3.2. Un exemple de dialogue déduit.*

Suite à la description du modèle générique, l'application est déduite sans que le concepteur ait eu recours à une description du modèle de dialogue. C'est tout d'abord une véritable hiérarchie de menus

qui est déduite. Cette déduction est due aux producteurs. À partir du moment où un mode de création est décrit, on déduit un menu de création pour l'objet. Dans le cadre de cette implémentation, nous nous sommes restreints à ne déduire que les dialogues de création des objets d'application à partir des comportements. Chaque dialogue de création produit correspond en fait à la génération d'une action globale. L'application est opérationnelle. Nous avons capturé deux étapes consécutives pendant son utilisation pour mettre en valeur le dialogue de création du lien dont nous avons étudié la spécification dans la partie précédente. Ces captures d'écran sont visibles dans la figure II.17.

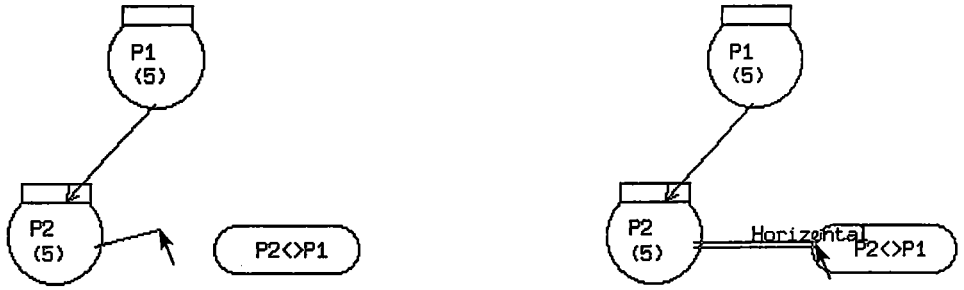



***Figure II.17.** Création d'un lien non typé.*

L'opérateur est en train de construire un lien non typé (producteur $P_1$) : il peut voir un résultat provisoire pendant sa création grâce à un affichage en élastique (réacteur extrinsèque $RE_1$). Puis, le lien est détecté horizontal (réacteur intrinsèque $RI_1$) et l'opérateur en est informé (réacteur extrinsèque $RE_2$). De plus, la liaison que l'opérateur tente d'effectuer permet au système de transformer le lien non typé en un lien contrainte (transmuteur $T_3$) et d'afficher ce nouveau type de lien (réacteur extrinsèque associé au lien contrainte). En conséquence, les comportements que l'on souhaitait voir sur le lien et qui ont été décrits dans le modèle générique sont inclus dans le dialogue déduit.

*4.3.3.Bilan.*

Le dialogue produit est plus que satisfaisant : on obtient un dialogue interactif de haut niveau. L'énorme atout est qu'on est assuré que le dialogue correspond aux attentes du concepteur non expérimenté dans la modélisation du dialogue, puisque ce dernier a pu intervenir dans l'étape de construction. Notons que ce concepteur peut être l'opérateur. Il se produit une réelle génération du dialogue. Il ne s'agit pas d'une réécriture de bas niveau, mais il se produit une interprétation : déduction des menus de créations, mise en place du modèle de dialogue ... Outre une réelle génération du dialogue, l'implémentation prouve qu'on peut éviter de contraindre l'utilisateur (le concepteur, voire l'opérateur) à apprendre un langage "barbare" pour décrire son application : on reste suffisamment éloigné des concepts du dialogue à utiliser tout en restant proche de la représentation que l'utilisateur se fait du dialogue.

Le travail qui a été réalisé a permis également de montrer :

- la hiérarchie d'utilisation des comportements. Nous avons constaté, par exemple, que le producteur se distingue des autres comportements contrairement au réacteur extrinsèque qui est accessible à tous ;

- l'importance de la représentation des objets. Les comportements sont étroitement liés à la représentation. Il faut donc bénéficier d'un accès privilégié à la représentation. La détection intervient pendant la création d'un objet avant sa validation finale. On utilise des attributs candidats à la création de l'objet. Il faut donc disposer d'une représentation provisoire de l'objet. Le problème est de maintenir cette représentation temporaire ;
- le manque d'outils adaptés pour décrire les comportements. Ne disposant pas de tels outils, le caractère naturel dans la description des comportements n'est pas mis en valeur avec autant d'évidence que nous l'aurions souhaité.

### 4.4.Synthèse.

Nous avons décrit dans cette partie comment envisager une génération d'un dialogue adaptatif. Grâce au concept des comportements, l'utilisateur non expérimenté dans la modélisation du dialogue peut intégrer ses connaissances au système pour décrire le type d'application qu'il souhaite utiliser. On arrive à déduire en grande partie le dialogue ainsi décrit : on obtient aisément un prototype du dialogue et surtout qui répond aux attentes du concepteur (opérateur éventuel) non expérimenté dans la modélisation du dialogue. De plus, l'ajout et la modification des objets ou comportements entraînent une mise à jour automatique du dialogue, ce qui donne une plus grande souplesse dans la spécification du dialogue.

Afin de garantir un dialogue optimal, l'utilisateur expérimenté peut intervenir sur la description par l'intermédiaire du formalisme graphique. Nous avons en effet montré que le dialogue est créé en accord avec le modèle de dialogue de SACADO, et par conséquent directement interprété selon le formalisme. Par ailleurs, les résultats de l'implémentation sont prometteurs, car, avec peu de moyens, le dialogue déduit est interactif : il simule bien à petite échelle le dialogue que l'on rencontre dans les systèmes de CFAO actuels et même plus, car on peut désormais ajouter facilement des connaissances de surface (le savoir-faire).

## 5. LA PORTÉE DES COMPORTEMENTS.

Notre recherche a abouti à l'intégration de connaissances dans le modèle sous une forme permettant de les exploiter. Les comportements permettent non seulement de structurer la connaissance, mais aussi d'acquérir avec une plus grande fiabilité les connaissances faisant appel au savoir-faire de l'utilisateur. Devant la puissance de représentation des connaissances apportées par les comportements pour déduire un dialogue plus que convenable, il ne s'agit pas de garder l'exclusivité de ce concept pour le dialogue.

De manière générale, si l'on observe le processus complet de développement d'un produit, on se rend compte que la connaissance est présente mais mal exploitée et que la fabrication du produit n'est pas optimum : la séparation entre les connaissances sur la conception et sur la fabrication pose un problème de communication dans le processus de développement [KRI 95]. Nous nous intéressons à ce problème dans cette partie. Nous présentons tout d'abord une nouvelle méthodologie qui tente de résoudre les problèmes de fabrication liés à la conception : la DFM (Design For Manufacturing). Nous

insistons essentiellement sur l'origine et la philosophie de cette approche afin de montrer dans un second temps comment notre concept de modèle générique décrit par des couples données/comportements peut s'inscrire dans les axes de cette méthodologie DFM.

### 5.1.Introduction à la méthode DFM.

Lorsqu'un opérateur interagit, il n'a pas forcément conscience si le produit qu'il décrit peut être fabriqué, ce qui fait que les choix qu'il prend ne sont pas toujours en accord avec la fabrication du produit. On peut envisager plusieurs raisons parmi lesquelles le manque de connaissances sur le processus de fabrication [DEL 94]. On se ramène toujours au même problème d'acquisition des connaissances (cf. Chapitre 1 *3.1.3. L'acquisition de la connaissance*).

En conséquence, on aurait tout intérêt à fournir des systèmes de conception mieux adaptés au résultat final i.e. la fabrication effective du produit. Pour y parvenir, il semble essentiel d'introduire des connaissances pertinentes sur les conditions de fabrication dès l'étape de conception pour prévenir d'éventuelles erreurs. Cette solution correspond à la méthodologie DFM (Design For Manufacturing) qui préconise de remonter au niveau de l'étape de conception des contraintes et connaissances liées à l'étape de fabrication [JOO 95 ; KRI 95 ; MON 94].

De nombreux systèmes intègrent dans leur structure cette notion de DFM. On peut citer le système de [JOO 95] qui développe des modèles statistiques à partir d'expérience de fabrication pour mettre à jour les règles de conception. Quant au système de [RON 95], il cherche à améliorer la qualité des produits modélisés tout en simplifiant le processus de conception. Pour cela, il utilise un modèle générique dans lequel la conception reconnaît les besoins, mais reste indépendante de tout type de méthodologie de fabrication. Cette approche utilise surtout des bibliothèques de composants comme connaissances de fabrication. À travers ces exemples, on souligne la volonté, dans l'approche DFM, de concevoir le meilleur produit [CHA 94b].

Les méthodes proposées ont tendance à confiner l'utilisateur dans un mode de pensée (la fabrication) si bien qu'il n'est plus libre d'agir. En effet, considérer la fabrication est une chose, mais cette connaissance ne doit pas contraindre l'utilisateur à une utilisation trop restrictive. En conséquence, nous proposons d'utiliser notre technique de capture des connaissances sur le dialogue pour intégrer des connaissances sur la fabrication, car notre méthode permet à l'utilisateur de s'exprimer selon son mode de pensée.

### 5.2.Une solution via les comportements.

Étant donné les caractéristiques des méthodes utilisant le principe DFM, le concept des comportements peut être exploité dans cet axe [GAR 96b]. En effet, nous avons utilisé jusqu'à présent les comportements pour inclure des notions proches du dialogue pour adapter le dialogue aux opérateurs et aux objets. De la même manière, nous voulons adapter la fabrication aux opérateurs et aux objets de sorte que les objets incluent des notions de fabrication et que les opérateurs accèdent à la fabrication, en comprennent les notions essentielles et les adaptent selon leurs propres critères.

Chaque type de comportement apporte une contribution particulière. Le réacteur extrinsèque permet de traduire un objet en des termes de fabrication pour l'opérateur. En définitive, il décrit des informations dédiées à la fabrication que le système pourra transmettre à l'opérateur au cours du dialogue. Les réacteurs extrinsèques décrivent de véritables aides pour faire comprendre des notions de fabrication à l'opérateur : par exemple, montrer les différents outils qui peuvent être utilisés pour construire le produit (cf. figure II.18).

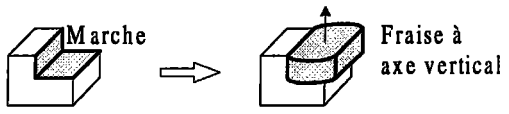



***Figure II.18.*** *Utilisation d'un réacteur extrinsèque pour traduire une marche par son outil.*

On peut traduire la représentation d'un objet pour fournir des données liées à la fabrication en utilisant des réacteurs intrinsèques. Ces données peuvent alors être utilisées par exemple pour gérer des propriétés propres à la fabrication (la propriété de vissage) ou encore pour préciser des propriétés selon des critères de fabrication. On peut également envisager que des objets d'application aient besoin de ces données propres à la fabrication.

Les transmuteurs gèrent les évolutions des objets. Par conséquent, ils peuvent inclure des connaissances sur la fabrication pour préciser ces changements : des critères de fabrication permettent d'empêcher l'évolution vers des objets que l'on ne peut raisonnablement pas fabriquer. De plus, les transmuteurs peuvent décrire de nouvelles évolutions des objets : des changements pour spécialiser l'objet selon un modèle de fabrication. Dans l'exemple de la figure II.19, une rainure est perçue dans un modèle géométrique classique de type CSG (Constructive Solid Geometry) comme une différence alors qu'elle devient une véritable entité dans un modèle orienté fabrication [GAR 96c]. Les transmuteurs décrivent les passerelles entre les différents modèles.

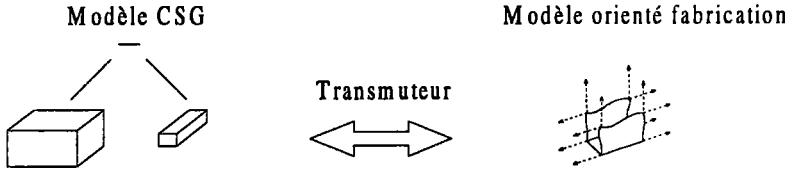



***Figure II.19.*** *Multi-modélisation d'une rainure grâce aux transmuteurs.*

Les producteurs décrivent des modes de création particuliers pour les objets. Ils peuvent donc inclure des connaissances spécifiques à la fabrication pour considérer ces informations dès la création des objets. Nous pensons notamment à l'ajout de contraintes de validité sur les attributs qui prennent en compte la fabrication potentielle de l'objet à créer : l'opérateur est aidé dans la conception des objets. Parallèlement aux nouvelles contraintes, de nouveaux attributs dédiés à la fabrication sur les objets peuvent être aussi décrits.

### 5.3.Synthèse.

Les besoins pour la fabrication sont comparables sur de nombreux points à ceux pour le dialogue. Ainsi, comme les comportements parviennent à capturer les connaissances pour simplifier la

conception du dialogue, il nous a paru naturel de les utiliser pour améliorer la fabrication. Nous avons montré que l'utilisation des comportements dans le cadre de la DFM est intéressante au moins pour deux raisons :

- la connaissance induite par les comportements guide l'opérateur pendant l'étape de conception du produit en accord avec l'étape de fabrication ;
- les comportements favorisent la multi-modélisation, car ils définissent avec précision les passages entre le modèle générique et un modèle dédié à la fabrication.

Cette extension que nous proposons sur les comportements renforce le concept initial, à savoir produire des dialogues adaptatifs. En effet, en incluant des connaissances sur la fabrication, les dialogues déduits deviennent de plus en plus spécialisés et adaptés à l'opérateur.

## 6. Conclusion.

Construire le dialogue d'une application n'est pas une tâche simple ainsi que nous l'avons montré dans le premier chapitre. C'est pour cette raison que nous avons choisi de décrire dans ce deuxième chapitre le mode de construction qui nous permet d'obtenir un dialogue "idéal" dans la mesure où il devient adaptatif. Nous avons insisté sur la notion de dualité dans la description du dialogue. Cette approche garantit que le dialogue spécifié répond aux attentes de l'opérateur, puisque tout utilisateur, expérimenté ou non, peut ajouter ses propres connaissances pour décrire l'application la plus complète possible. Toutefois, il ne faut pas négliger la gestion délicate de la cohérence entre les dialogues construits par chaque approche. Nous ne nous sommes pas préoccupé de cet aspect dans la mesure où nous nous sommes tout d'abord intéressé aux méthodes d'acquisition des connaissances pour améliorer la spécification du dialogue.

Le travail d'équipe est prédominant dans notre approche. Les utilisateurs expérimentés spécifient le dialogue en utilisant un formalisme graphique reprenant les concepts du modèle de dialogue. Il faut donc une grande maîtrise de l'architecture et du modèle de dialogue utilisé pour décrire aisément tout type de dialogue à l'aide du formalisme. En prenant conscience que ces concepteurs manquent d'expérience en matière d'utilisation des applications, nous avons introduit une méthode permettant à des utilisateurs non expérimentés, et en particulier l'opérateur, de définir à leur tour des connaissances plus subtiles dont le concepteur n'a pas conscience. En effet, l'opérateur manipule les applications tant et si bien qu'il devient expert dans leur manipulation. Il acquiert des connaissances sur le dialogue auxquelles le concepteur ne pense pas forcément. De plus, l'opérateur a un mode de pensée tout à fait différent du concepteur, car il a ses propres critères en matière de dialogue, critères qui sont bien souvent liés à son domaine d'activité.

Le concept des comportements représente une technique d'acquisition des connaissances de l'utilisateur en matière de dialogue : l'utilisateur est l'expert dans l'art de dialoguer. En IA, pour construire des systèmes experts, c'est un cogniticien qui est chargé de traduire les connaissances de surface provenant de l'expert [CHO 91b ; FAR 88]. Nous avons choisi de déléguer ce rôle aux comportements dont la préoccupation essentielle est de traduire les connaissances de l'utilisateur.

Les comportements imposent une certaine discipline dans la description de par leur décomposition en producteurs, réacteurs et transmuteurs. Toutefois, ils respectent la pensée de l'utilisateur. On dispose désormais d'une méthode simple pour acquérir et représenter la connaissance. L'utilisateur peut se concentrer sur la modélisation. De plus, peu de pertes d'informations sont à déplorer, car tout est centralisé dans le modèle générique.

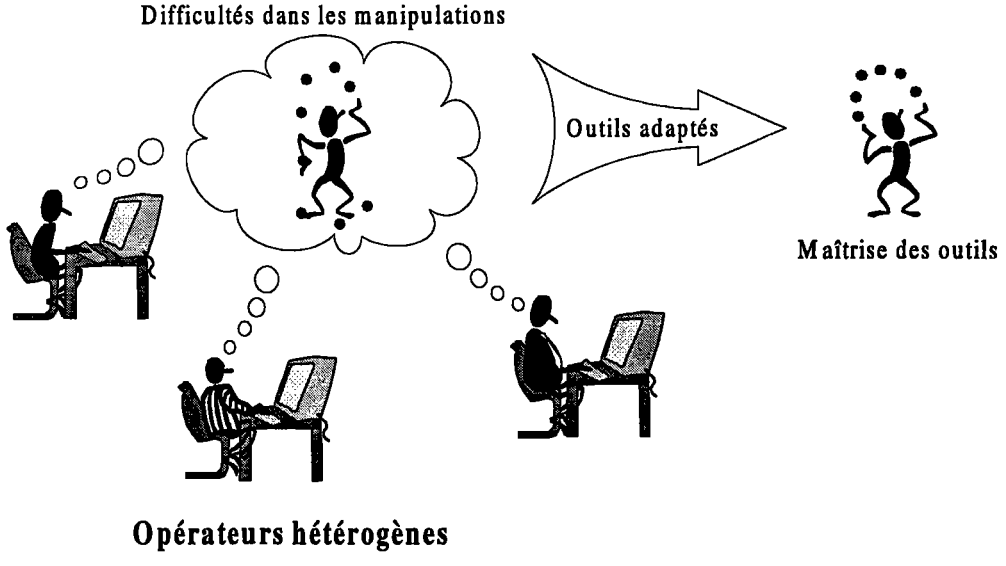
Nous avons montré que le système déduit le dialogue à partir des comportements et ceci en accord avec le modèle de dialogue utilisé par le concepteur. L'implémentation que nous avons réalisée est la preuve que ce n'est pas une utopie. Quoi qu'il arrive, les utilisateurs expérimentés ou non ont la possibilité d'intervenir sur le dialogue déduit soit en utilisant à nouveau les comportements, soit le formalisme graphique. De par notre méthode de construction, nous prenons conscience qu'il n'existe pas une unique personne qui possède toutes les informations nécessaires à la description d'un dialogue adaptatif.

Pour donner encore plus de poids aux comportements, nous avons montré que ce concept est également un moyen pour acquérir des connaissances qui ne sont pas a priori destinées au dialogue : nous avons pris comme exemple les connaissances sur la fabrication. L'avantage de notre technique de modélisation par les comportements est qu'elle permet de prendre en compte ces connaissances également pour le dialogue.

Dans les chapitres suivants, nous ne nous intéressons plus à la manière d'acquérir les connaissances pour spécifier le dialogue, mais plutôt au fonctionnement même du dialogue. Notre but est toujours de construire un dialogue adaptatif. Après nous être intéressé à la construction du dialogue en respectant les critères des différents utilisateurs, nous nous penchons sur les fonctionnalités que le dialogue apporte à l'opérateur pour rendre l'application encore plus performante tout en restant en accord avec l'opérateur.

# CHAPITRE 3.

# OUTILS ADAPTÉS AU DIALOGUE EN CFAO.



## 1. INTRODUCTION.

Le chapitre précédent était consacré à l'étude d'une nouvelle approche de description du dialogue. Nous disposons maintenant d'une technique de construction du dialogue qui permet de capturer les connaissances du concepteur qu'il soit utilisateur final ou non de l'application. Les conditions sont favorables pour fournir un dialogue qui réponde aux attentes de l'utilisateur.

La qualité du dialogue est évaluée en fonction de la capacité d'un opérateur à accomplir sa tâche. Ce ne sont pas seulement les fonctionnalités proposées, mais ce sont aussi les outils permettant d'effectuer ces fonctionnalités qui sont déterminants. Jusqu'à présent, nous avons surtout insisté sur la description du dialogue par les fonctionnalités à proposer dans une application. Grâce à la simplicité de la méthode de description, les fonctionnalités deviennent de plus en plus complexes. Il s'ensuit une évolution des outils à mettre en place pour réaliser ces fonctionnalités. Nous considérons que les outils sont des aides intégrées au dialogue. Leur rôle est délicat, car ils doivent être accessibles et faciliter les interactions de l'opérateur sans pour autant le ralentir, la compétitivité demeurant l'un des principaux critères pour une application.

Dans ce chapitre, nous nous intéressons à ces outils dédiés aux manipulations dans une application. Bien que le concepteur puisse décrire des aides à intégrer au dialogue en utilisant la méthode duale de description, lui laisser une liberté totale ne nous paraît pas adéquat. Il n'a pas forcément conscience des capacités du système et donc une maîtrise totale des outils qu'il peut mettre à la disposition de l'opérateur. De plus, comme chaque concepteur peut définir ses propres outils, on obtient une multitude d'outils plus ou moins distincts. Il s'agit de regrouper l'ensemble de ces outils afin de parvenir à une gestion plus adaptée et surtout afin d'aboutir à une standardisation.

Tout d'abord, nous concentrons notre attention sur le dialogue proposé à l'opérateur. Nous insistons sur les possibilités qui lui sont offertes ainsi que sur les limites de la définition de nouveaux outils d'aides à la manipulation. Cette analyse nous permet d'introduire ensuite le concept de manipulateurs, qui constitue notre proposition pour spécifier l'ensemble des outils. Enfin, nous montrons les différentes propriétés qui renforcent ce concept. Cette étude nous permet de présenter le nouveau fonctionnement du dialogue.

## 2. LES POSSIBILITÉS DU DIALOGUE.

Nous nous plaçons du côté de l'opérateur. Imaginons qu'une application ait été décrite et soit opérationnelle. L'opérateur se trouve dans un certain contexte d'utilisation qu'il nous importe de cerner pour comprendre les différents types de manipulations qu'on peut lui fournir. Nous introduisons la notion de contexte d'utilisation dans une application. Nous définissons le contexte d'utilisation à un instant *t*, *C(t)*, par le triplet formé d'un opérateur *op* qui interagit, d'un sous-ensemble *em* de manipulations proposées à *op* et d'un sous-ensemble d'actions *act* que *em* permet d'accomplir (cf. figure III.1.). Ainsi, $\forall t\ \forall op \in OP\ \exists em \subset EM\ \exists act \subset ACT \mid C(t) = (Op, em, act)$.

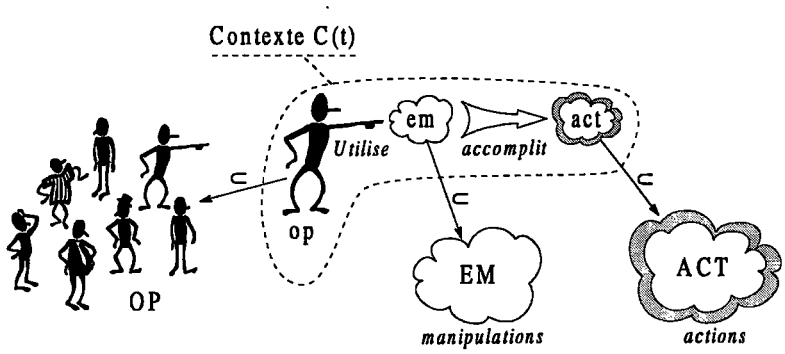



***Figure III.1.*** *Contexte d'utilisation à un instant t.*

Le contexte d'utilisation n'a pas toujours le même impact sur l'opérateur. On distingue :

- les contextes de base, qui définissent un ensemble d'outils de manipulations permettant d'accomplir des actions correspondant à des fonctionnalités de base. On entend par fonctionnalité de base toute opération directement liée à la gestion du modèle des objets (création, modification, destruction ...). Les outils de manipulations proposés sont alors souvent assimilables à l'utilisation de styles d'interactions tels que les menus ou les boîtes de dialogue (cf. Chapitre 1 *2.3.2. Les styles d'interactions de base.*) ;
- les contextes d'aides, qui comprennent des outils de manipulations de plus haut niveau. On ne se contente plus de répondre strictement aux fonctionnalités de base. Les actions proposées aident l'opérateur à atteindre son but. Ainsi, les outils de manipulations sont plus complexes. Ils allient des styles d'interactions à des mécanismes d'aides pour proposer des fonctionnalités plus adaptées à l'opérateur (détections, zoom ...). Le dialogue devient plus adaptatif.

Les contextes d'aides caractérisent les systèmes actuels. Nous concentrons notre attention sur les outils de manipulations des ensembles *em* associés aux contextes d'aides, car ils apportent de nouvelles possibilités pour le dialogue. Dans la partie suivante, nous insistons sur l'importance de contrôler ces nouveaux outils en montrant les multiples possibilités offertes par la RV (réalité virtuelle) en ce qui concerne les manipulations. Plus concrètement, nous présentons dans la seconde partie un exemple de tels outils de manipulations : une calculatrice d'expressions grapho-numériques. Nous posons le problème de la spécification de cet outil dans le modèle de dialogue de SACADO.

### 2.1.Les manipulations en réalité virtuelle.

Nous avons présenté dans le chapitre 1 l'importance des interfaces 3D ; l'exemple le plus marquant concerne le domaine de la RV (cf. Chapitre 1 4. Les nouvelles interfaces). L'univers virtuel entraîne des changements dans les manipulations. Certains outils de manipulations qui ont fait leur preuve en 2D ne sont pas adaptés en 3D : les styles d'interactions ne conviennent parfois plus. Par exemple, on dénote une certaine lourdeur dans l'utilisation des menus dans les environnements virtuels. Une des raisons provient des dialogues qui tendent à devenir plus naturels et intuitifs : on essaie de déduire un maximum d'actions de l'opérateur. Les anciens outils de manipulations ont tout intérêt à évoluer en incluant des mécanismes qui tiennent compte de ce caractère intuitif. Les outils deviennent plus complexes à gérer.

La RV offre de nouvelles possibilités dans les manipulations. Ce phénomène s'accentue avec l'apparition des interfaces multi-modales (cf. Chapitre 1 *4.1.2. Les interfaces multi-modales*). Les nouveaux outils ont tendance à se fondre dans l'environnement : c'est dans la scène 3D que l'on introduit des outils (par exemple, des manettes associées aux objets pour les modifier). L'opérateur peut effectuer la plupart de ses manipulations sans quitter l'environnement virtuel ; il reste ainsi concentré sur ce qui le préoccupe. De plus, directement liées aux nouvelles possibilités, des difficultés supplémentaires sont à surmonter. On peut évoquer le problème de la navigation dans le monde virtuel. Les déplacements de l'opérateur amènent des problèmes tant sur la manière d'évoluer que sur celle de reconnaître l'information (visualisation ...) et de l'utiliser (désignation ...). Les outils de manipulations se diversifient pour faire face à ces nouveaux problèmes. Il faut définir un contrôle sur ces outils.

## 2.2.La description d'un nouvel outil.

Les manipulations deviennent de plus en plus évoluées et variées. Dès qu'une fonctionnalité de base est décrite dans une application, se pose le problème de l'outil de manipulations à mettre en place pour placer l'opérateur dans un contexte d'aides *C(t)*. Le dialogue à décrire pour l'utilisation de ces outils peut être conséquent. Nous nous proposons d'étudier l'exemple de la définition d'une calculatrice dédiée à la manipulation de contraintes. Cet outil fait partie d'une implémentation basée sur le noyau SACADO [GAR 95a ; STE 94]. Nous précisons tout d'abord les raisons qui nous ont poussés à définir cet outil. Puis, nous décrivons l'outil lui-même, car il ne correspond pas simplement à un style d'interactions. Enfin, nous montrons comment il a pu s'intégrer dans le modèle de dialogue de SACADO.

### *2.2.1.Les motivations.*

Dans la plupart des systèmes de CFAO, les objets modélisés sont créés par affinages successifs. Cette technique consiste à définir l'objet globalement, puis à lui ajouter des contraintes afin de l'identifier de manière plus précise. Si elles ne sont pas détectées, les contraintes sont spécifiées à l'aide d'expressions alphanumériques. L'opérateur donne une telle expression à l'aide d'un langage textuel et indique par des cotations toutes les données graphiques qu'il mentionne dans l'expression de manière alphanumérique (cf. figure III.2).

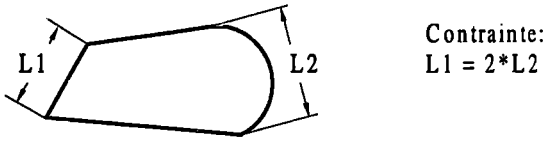



***Figure III.2.** Définition d'expressions alphanumériques.*

Cette manipulation, bien que standard, demeure contraignante pour l'opérateur :

- l'utilisation des cotations est rendue difficile à cause des conventions à respecter pour les représenter. La mise en place de ces conventions complique la gestion de l'expression;
- l'opérateur est contraint par la définition des étiquettes symbolisant chacune des données graphiques repérées par des cotations (sur la figure III.2, il s'agit de *L1* et *L2*) ;

- la saisie de l'expression alphanumérique ("*L1 = 2*L2*") peut être contraignante selon les styles d'interactions utilisés (boîte de dialogue avec saisie au clavier ...) ;
- l'utilisation de cotations soulève le problème de la lisibilité. On se retrouve très vite avec une quantité de cotations surchargeant le modèle visualisé.

L'outil de manipulations utilisé pour décrire une expression ne donne pas forcément entière satisfaction : le contexte d'aides fourni par le système des cotations n'est pas "idéal". On est encore trop proche d'un contexte de base, car l'outil de manipulations proposé fournit des actions correspondant à des fonctionnalités de base de l'application (création de l'expression en saisissant les éléments la caractérisant).

Plutôt que d'améliorer cet outil de manipulations, nous introduisons un nouvel outil : une calculatrice d'expressions grapho-numériques. En effet, les manipulations proposées à l'opérateur demandent des améliorations : le dialogue est peu adapté à un opérateur non expérimenté dans le système des cotations et il lui délègue beaucoup de tâches. Nous présentons la calculatrice dans la partie suivante en montrant les éléments qui permettent de placer l'opérateur dans un contexte d'aides.

### *2.2.2.La calculatrice d'expressions grapho-numériques.*

La calculatrice que nous introduisons permet de créer une contrainte sous la forme d'une expression. La particularité de la contrainte n'est pas dans sa forme, mais dans sa manière d'être créée i.e. dans le dialogue mis en place. Le principe de base de la manipulation revient à donner l'expression en une unique étape i.e. sans avoir recours au préalable à des spécifications (par cotations) des données graphiques intervenant dans la contrainte. L'outil de manipulations permet à l'opérateur de définir les données graphiques aussi naturellement que les données numériques pendant la définition de l'expression : on ne traite plus des expressions alphanumériques, mais réellement des expressions grapho-numériques. Pour parvenir à ce type de manipulations, il suffit de laisser l'opérateur montrer graphiquement les données au cours de la création i.e. quand il a besoin d'introduire une donnée graphique dans l'expression. La calculatrice est un outil de manipulations qui intègre des mécanismes et des styles d'interactions propres. On lui associe deux mécanismes chargés de :

- la reconnaissance des données désignées graphiquement. L'opérateur ne mentionne pas le type de donnée (par exemple, la hauteur d'un cylindre). Il s'agit de mettre en place des règles de détection des données. C'est ici qu'intervient l'IA, car la technique utilisée pour détecter les données graphiques s'apparente à celles des systèmes à base de connaissances (base règles ...) [STE 94] ;
- l'introduction de la donnée graphique reconnue dans l'expression de telle sorte que l'opérateur puisse faire le lien entre sa désignation et la représentation de la donnée de l'expression. On n'impose plus à l'opérateur la gestion d'étiquettes.

De même, on associe des styles d'interactions pour former une calculatrice, soulageant ainsi l'opérateur d'un langage textuel compliqué pour saisir les données numériques. La gestion de cette calculatrice appartient entièrement à l'outil de manipulations. La figure III.3 montre la calculatrice lorsqu'une expression est créée.

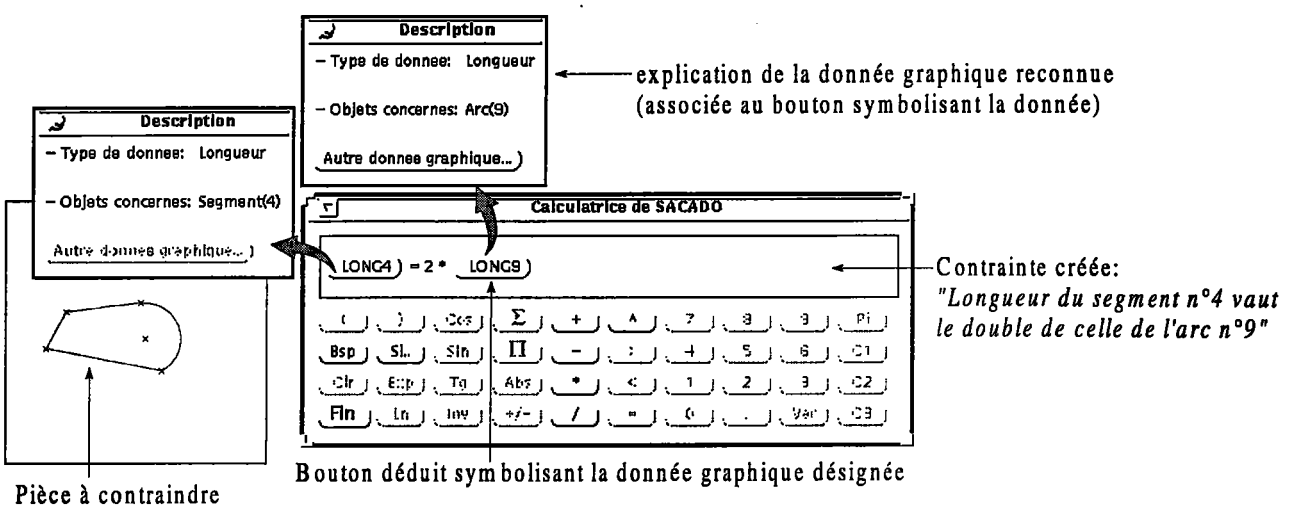



***Figure III.3.*** *Calculatrice d'expressions grapho-numériques.*

L'outil de manipulations défini a un caractère plus intuitif que le type de manipulations classiquement utilisé pour créer des contraintes : il s'adapte à l'opérateur, car les actions que l'outil décrit ne correspondent pas forcément aux fonctionnalités de base de l'application (reconnaissance des données graphiques ...). Ainsi, la calculatrice représente une entité à part entière du modèle de dialogue, composée de styles d'interactions et de mécanismes de haut niveau parmi lesquels des mécanismes d'IA. La difficulté est d'intégrer cet outil dans le modèle de dialogue en respectant cette représentation.

### *2.2.3.L'intégration au modèle de dialogue.*

Nous avons implémenté la calculatrice d'expressions grapho-numériques en prenant comme modèle de dialogue celui décrit dans le noyau SACADO. La calculatrice est considérée comme un outil de manipulations qui constitue une entité à part entière dans le dialogue. Ainsi, elle doit apparaître en tant que tel dans le modèle. En ce sens, une première étape a consisté à spécifier l'ensemble des mécanismes et des styles d'interactions comme un tout, indépendamment de l'architecture complète de l'application. Dans la seconde étape, nous nous sommes préoccupés de l'intégration de la calculatrice dans l'architecture (cf. Chapitre 2 figure II.1 pour voir la définition de l'architecture de SACADO). Pour préserver cette notion d'entité à part entière, la solution choisie considère la calculatrice comme un domaine annexe. Chaque bouton de la calculatrice est assimilé à un menu de ce domaine annexe. Le domaine principal donne le contrôle du dialogue au domaine annexe dès qu'une interaction demande une expression grapho-numérique ou bien dès que le menu du domaine principal concernant la création d'une expression est activé.

Le domaine annexe nous a servis à regrouper les éléments communs à la manipulation pour créer une expression grapho-numérique. Il représente dans l'application un outil de manipulations. Bien que le résultat soit satisfaisant dans la mesure où la calculatrice fonctionne en accord avec sa spécification, l'utilisation du domaine annexe en tant qu'outil de manipulations est discutable. En effet, à l'origine, le domaine annexe représente des fonctionnalités qui interviennent dans un contexte donné. Il en résulte que la calculatrice est perçue dans l'application comme une fonctionnalité particulière et non comme un outil de manipulations répondant à une fonctionnalité. Ceci signifie qu'à chaque nouvel outil de

manipulations, un nouveau domaine annexe est créé. Il n'est pourtant pas cohérent de séparer des outils qui répondent à la même fonctionnalité. De plus, si un outil intervient dans des contextes différents, il faudrait le représenter par plusieurs domaines annexes (autant que de contextes différents) : l'outil n'aurait plus sa place comme une entité à part entière.

En outre, on comprend aisément que définir des boutons de la calculatrice comme les menus du domaine annexe n'est qu'une solution temporaire pour intégrer la calculatrice dans le modèle de dialogue de SACADO. Il est impossible de généraliser cette méthode d'intégration d'un outil au modèle de dialogue. Deux raisons majeures s'ajoutent à celles énoncées précédemment sur l'assimilation entre le domaine annexe et l'outil :

- certains boutons ne sont pas considérés comme des menus dans la calculatrice : il s'agit des boutons apparaissant dans l'expression en création et symbolisant les données graphiques (cf. figure III.3.). Ces boutons sont déduits des données graphiques détectées : on ne peut les définir comme des menus du domaine annexe dans l'architecture puisqu'il est impossible de les répertorier au préalable ;
- les styles d'interactions ne sont pas tous assimilables à des menus.

L'intégration de la calculatrice dans le modèle de dialogue de SACADO n'est pas idéale. Dans ce travail d'implémentation, nous avons concentré nos efforts sur le fonctionnement de la calculatrice, son intégration dans le modèle n'était pas notre préoccupation première. Le résultat obtenu montre que le dialogue géré par le noyau SACADO peut tenir compte de cet outil de manipulations moyennant quelques adaptations. La méthode d'intégration utilisée ne peut pas être généralisée à l'ensemble des outils de manipulations. Nous définissons dans le paragraphe 3 le concept des manipulateurs qui représente un concept général sur les outils de manipulations.

### 2.3.Synthèse.

Le modèle de dialogue de SACADO ne permet pas une intégration standard des outils de manipulations en tant qu'entités à part entière. Nous pensons que le problème provient d'abord des outils eux-mêmes, car il n'existe pas de modèle définissant clairement l'ensemble des outils de manipulations disponibles. On rencontre une multitude d'outils qui peuvent être très distincts les uns des autres : il peut tout aussi bien s'agir de loupes proposant des filtres de visualisation [BIE 93] que de manettes liées aux objets pour modifier leur hauteur [CHU 95]. Une fois formalisés, les outils sont mieux cernés et on peut les inclure au modèle de dialogue de manière générale. En outre, les systèmes de RV manifestent le besoin d'une standardisation de ses outils de manipulations : nous situons notre approche dans le contexte des interfaces 3D.

## 3. LES MANIPULATEURS.

Parce que les outils de manipulations sont très nombreux et créés souvent indépendamment les uns des autres (sans connaissance des outils existants), ils possèdent leur propre définition. Notre propos consiste à formuler un concept suffisamment général pour regrouper l'ensemble des outils de

manipulations existants et à venir. Nos travaux de recherche ont abouti à l'introduction du concept des manipulateurs dans la modélisation du dialogue.

Dans les parties suivantes, nous portons notre attention sur la spécification des manipulateurs. Tout d'abord, nous donnons une définition du manipulateur. Des exemples de manipulateurs sont présentés afin de montrer que la définition proposée est suffisamment générale pour inclure tout type d'outils de manipulations. Ensuite, nous présentons le lien qui existe entre les manipulateurs et les réacteurs extrinsèques. Ce lien nous permet de situer les manipulateurs par rapport à la spécification du modèle générique présenté dans le chapitre 2. Puis, nous établissons une classification des manipulateurs afin de montrer la puissance de représentation d'un tel concept. Enfin, nous définissons les manipulateurs généraux qui décrivent des manipulateurs dont la représentation permet d'influer sur le fonctionnement d'autres manipulateurs. Nous montrons comment utiliser la représentation du manipulateur pour obtenir un tel manipulateur.

### 3.1.Définition et exemples.

Un manipulateur est un concept formalisant un outil quelconque de manipulations. Il représente des aides mises à la disposition de l'environnement animé et qui œuvrent ensemble pour satisfaire un même but. Nous appelons environnement animé aussi bien l'opérateur intervenant virtuellement que des objets qui ont une certaine autonomie (nous pensons tout particulièrement aux objets simulant des robots). La figure III.4. donne la représentation d'un manipulateur. Afin de mieux comprendre cette représentation, nous décrivons au fur et à mesure de l'explication un outil de manipulations permettant à un type d'opérateurs de modifier la hauteur d'un cylindre : le manipulateur manette.

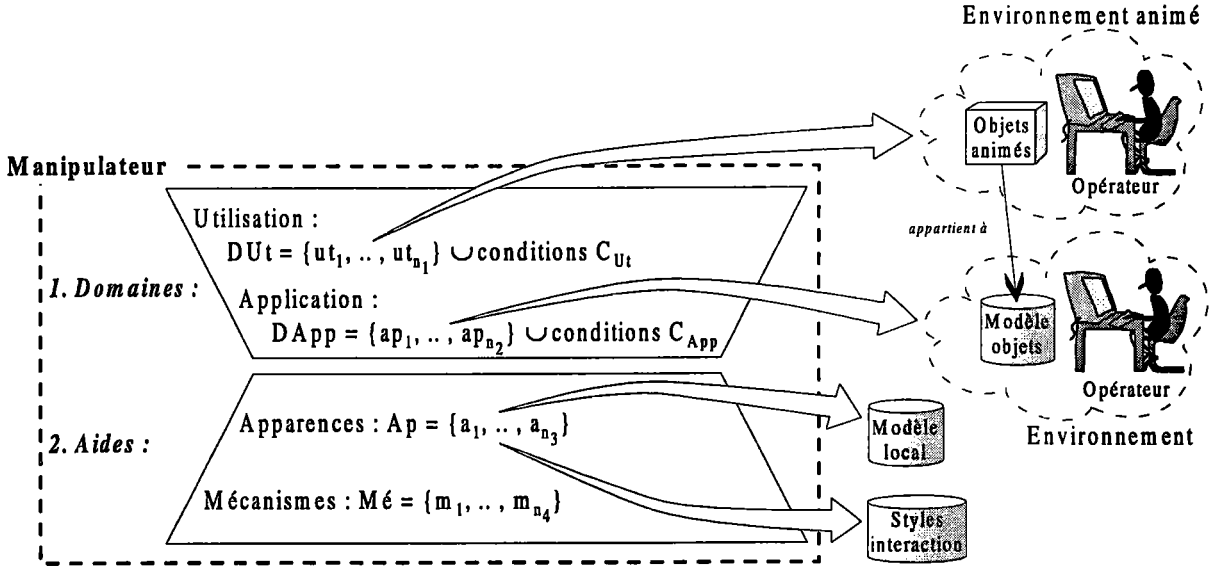



***Figure III.4. Définition d'un manipulateur.***

On définit le manipulateur comme une entité à part entière du système. Il est défini par la donnée du couple : (Domaines, Aides). La partie Domaines contient les domaines caractérisant le manipulateur :

- *DUt*, le domaine d'utilisation qui définit l'ensemble des entités animées $ut_h$ (objets ou opérateurs) qui vont utiliser le manipulateur. On inclut également dans ce domaine les conditions $C_{Ut}$ portant

sur les entités $ut_h$. Elles définissent les conditions d'utilisation du manipulateur. Dans le cas du manipulateur manette, *DUt* est constitué d'un type d'opérateurs (seuls certains opérateurs ont accès à la manette) et d'une condition indiquant à quel moment la manette peut être utilisée (par exemple, condition qui définit une distance entre l'opérateur et le cylindre à partir de laquelle le manipulateur manette peut être proposé à l'opérateur) ;

- *DApp*, le domaine d'application qui définit l'ensemble des entités $ap_k$ de l'environnement (animé ou non) sur lesquelles le manipulateur agit et pour lesquelles on définit éventuellement des conditions $C_{App}$. Il s'agit du cylindre pour le manipulateur manette.

La partie Aides du manipulateur décrit le cœur du manipulateur : elle lui donne sa sémantique en décrivant les aides au dialogue apportées par le manipulateur. Nous distinguons deux types de description d'aides intégrés au manipulateur :

- *Ap*, l'ensemble des apparences qui définit les différents modes de représentation du manipulateur. On définit une apparence $a_i$ comme un ensemble de composants correspondant à des styles d'interactions (menus, boîte de dialogue ...) ou à une représentation géométrique du manipulateur (cas d'un manipulateur tel que la manette qui intervient dans l'environnement virtuel) ;
- *Mé*, l'ensemble des mécanismes qui décrit le manipulateur pendant son utilisation. Il s'agit de l'élément moteur du manipulateur. Un mécanisme $m_j$ représente un fonctionnement du manipulateur. Il peut être étroitement lié à une apparence $a_i$ dans le cas où le manipulateur capture des informations grâce à l'utilisation de $a_i$. Le mécanisme inclut les traitements nécessaires à l'exploitation de ces informations en accord avec la manipulation. Par exemple, lorsque le manipulateur manette est utilisé (grâce à une de ses apparences), il peut évaluer la nouvelle hauteur du cylindre ; cette tâche est réservée à l'un des mécanismes du manipulateur manette. Des mécanismes plus évolués sont également intégrés au manipulateur. Ils dépendent plus de l'environnement que des apparences du manipulateur. Il s'agit, par exemple, du mécanisme de reconnaissance des données graphiques pour le manipulateur gérant la calculatrice d'expressions grapho-numériques.

Un manipulateur peut avoir plusieurs apparences. Elles sont le reflet du manipulateur et influent sur l'aide qu'il véhicule. Selon le contexte dans lequel il se trouve pendant le dialogue, on autorise de modifier l'apparence du manipulateur, ce qui permet de le rendre plus compréhensible puisqu'à chaque apparence on peut associer une étape de la manipulation. Dans le cas de la manette, on peut décrire une apparence lorsqu'elle est en attente d'utilisation et une apparence lorsqu'elle est utilisée. Le manipulateur est concerné par la multi-modélisation car il peut posséder plusieurs représentations selon le contexte.

D'après la définition du manipulateur, une apparence $a_i$ peut correspondre à une représentation géométrique. On définit $a_i$ dans un modèle local à l'ensemble des manipulateurs. Dans l'exemple du manipulateur manette, on définit, par exemple, une de ses apparences comme un parallélépipède symbolisant une manette et que l'on représente dans la scène avec le cylindre à modifier : ce parallélépipède est un objet modélisé localement pour le manipulateur manette.

Considérer un modèle local pour l'ensemble des manipulateurs est une perspective intéressante, car on garantit l'indépendance entre l'application et le dialogue et on facilite les traitements. En effet, les manipulateurs sont des éléments propres au dialogue, il n'y a pas lieu de les considérer comme des objets d'application. De plus, on a besoin d'un modèle local aux manipulateurs pour gérer les différentes représentations géométriques qui peuvent les caractériser : on répond au problème de la prise en compte de la multi-modélisation pour les manipulateurs. On répond également à des problèmes qui surviennent actuellement entre les éléments de dialogue créés et les objets d'application créés. Parmi ces problèmes, on peut citer celui lié au rafraîchissement de l'environnement : il s'agit d'effacer la scène et d'afficher tous les objets d'application modélisés. En raison de l'indépendance entre le dialogue et l'application, on ne disposait pas de réels moyens pour afficher les éléments propres au dialogue qui étaient déjà affichés. Grâce au modèle local, on peut rafraîchir également l'ensemble des apparences des manipulateurs en action au moment du rafraîchissement. En considérant le modèle local aux manipulateurs, on pose malgré tout le problème de la cohérence entre les différents modèles. Il s'agit notamment de modifier les algorithmes de visualisation réaliste (i.e. avec élimination des parties cachées) basés sur un unique modèle.

La définition que nous avons donnée du manipulateur est générale, ce qui est une condition nécessaire pour inclure tous les outils de manipulations. L'un des atouts de ce concept est qu'on identifie des éléments qui composaient l'application jusqu'à présent sans réelle appartenance à un ensemble : par exemple, les fenêtres graphiques ou les curseurs. En fait, ce sont des outils de manipulations. Par exemple, pour un manipulateur fenêtre, on peut définir :

- pour son domaine d'utilisation *DUt*, tout opérateur, car la fenêtre est un outil de manipulations chargé de proposer à l'opérateur un mode de visualisation des objets ;
- pour son domaine d'application *DApp*, l'ensemble des objets d'application créés ;
- pour ses apparences *Ap*, une fenêtre graphique et une icône correspondant à la fenêtre réduite mise en attente (allusion aux fenêtres utilisées sous Microsoft® Windows®) ;
- pour ses mécanismes *Mé*, la gestion de l'affichage dans l'apparence du manipulateur i.e. la fenêtre graphique et la gestion des événements qui se produisent dans cette apparence (désignations ...).

### 3.2.Le lien avec les réacteurs extrinsèques.

Fondamentalement, un manipulateur repose sur des informations qui le caractérisent et qui permettent de définir le couple (Domaines, Aides). Il s'agit de la hauteur du cylindre pour le manipulateur manette, car sans cette information, ce manipulateur n'existe pas. Les informations peuvent être plus abstraites comme dans le cas de la calculatrice d'expressions grapho-numériques pour laquelle il s'agit de la capacité de l'opérateur à définir des contraintes. On définit une donnée *D* comme l'ensemble des informations provenant d'une entité quelconque (objet, objet animé ou opérateur) et capables d'induire un manipulateur (cf. figure III.5) : *D* représente les informations sans lesquelles le manipulateur n'a pas d'existence. De plus, l'utilisation du manipulateur peut provoquer des modifications de *D* : dans le cas de la manette, son utilisation entraîne une mise à jour de la hauteur du cylindre. Par ailleurs, dans la

suite de notre propos, si l'on définit un manipulateur pour une entité, il s'agit d'une association faite par transitivité sur *D* (cf. figure III.5), car en réalité le manipulateur est directement lié à *D*.

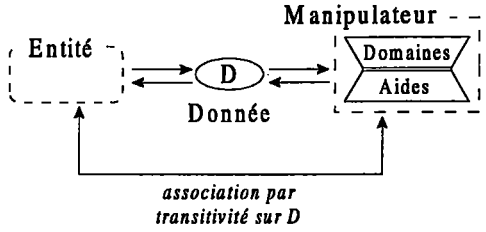



***Figure III.5.** Origine du manipulateur.*

Cette précision sur l'origine du manipulateur nous permet de définir exactement l'intégration du manipulateur dans le système. Ainsi que le montre la figure III.6, le manipulateur est lié aux comportements, car la donnée sur laquelle il repose est fournie par un réacteur extrinsèque.

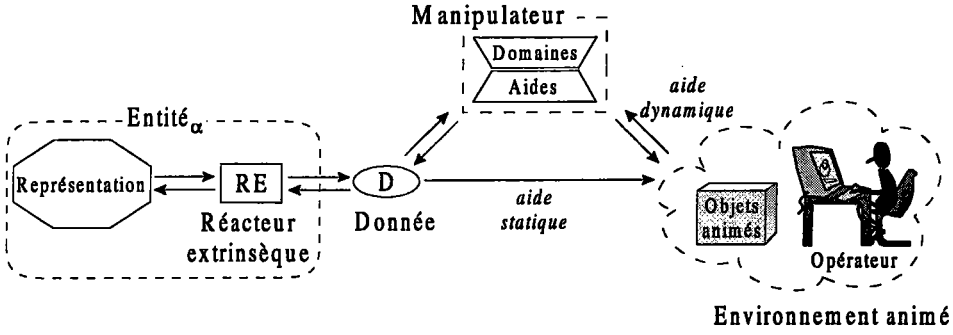



***Figure III.6.** Lien entre le manipulateur et le réacteur extrinsèque.*

En effet, le réacteur extrinsèque est un comportement chargé de traduire, pour un opérateur, l'objet auquel il est associé, en une donnée *D*. De plus, par extension, nous définissons le réacteur dans le cadre des environnements animés. Par conséquent, il est défini pour une entité quelconque (objet, objet animé ou opérateur) et il est destiné à fournir une aide pour une entité animée. Ainsi, la donnée du réacteur extrinsèque correspond à des informations pouvant servir de base à un manipulateur.

Le manipulateur prend toute son importance en l'associant à un réacteur extrinsèque, car son rôle est de se servir de la donnée *D* pour apporter une aide dynamique. En effet, dans le chapitre 2, nous n'avons exploité la donnée du réacteur extrinsèque que pour fournir des aides statiques telles que des affichages (cf. Chapitre 2 *3.2.1. La traduction*). Si l'on reprend l'exemple de la manette, un réacteur extrinsèque *RE* est associé à un cylindre pour le traduire par sa hauteur (donnée *D*). Cette donnée sert de base pour ce manipulateur dont le rôle est d'ajouter des informations réactives : on montre la donnée hauteur grâce à une manette et on autorise sa modification. La traduction que le manipulateur opère sur la donnée permet aux entités de l'environnement animé, concernées par le manipulateur, d'interagir sur cette donnée.

En outre, cette association des manipulateurs avec les réacteurs extrinsèques permet de mieux appréhender l'intégration des manipulateurs dans le modèle de dialogue. En effet, les réacteurs sont définis dans le modèle générique en tant que comportements sur les entités. Nous avons montré que le

dialogue les concernant peut être déduit. Ainsi, il est tout à fait envisageable de décrire les manipulateurs et de déduire le dialogue associé. Il suffit de comprendre les manipulateurs comme une extension des réacteurs : l'utilisateur décrit comment un objet est interprété dans une situation donnée et ajoute à cette spécification l'outil de manipulations associé.

En définitive, le manipulateur est un sous-traitant du réacteur extrinsèque : le réacteur délègue au manipulateur toute la partie interactive de l'aide qu'il doit éventuellement fournir. Ainsi, on va plus loin dans l'interaction, car on transmet un message véhiculé par la donnée et on permet de travailler avec cette donnée. Il n'est pas obligatoire d'associer un manipulateur à chaque réacteur extrinsèque. Si l'aide revêt un caractère statique, un outil de manipulations n'a pas lieu d'être. Ainsi, les réacteurs extrinsèques présentés dans le chapitre 2 sont toujours valides. Par ailleurs, on peut envisager plus tard de transformer ces aides statiques fournies en aides dynamiques par l'introduction de manipulateurs.

## 3.3.Une classification.

La définition que nous avons donnée d'un manipulateur permet d'envisager a priori une grande quantité, pour ne pas dire la totalité, des outils en tant que manipulateurs. Le problème est que l'on se retrouve très vite avec un ensemble de manipulateurs très conséquent. Des outils tels que la calculatrice d'expressions grapho-numériques et la manette de modification de hauteur sont tous traités comme des manipulateurs alors qu'ils sont différents dans leur mode de fonctionnement. Pour apporter une plus grande maîtrise sur l'ensemble des manipulateurs et le rendre plus homogène, nous définissons la classification illustrée par la figure III.7.

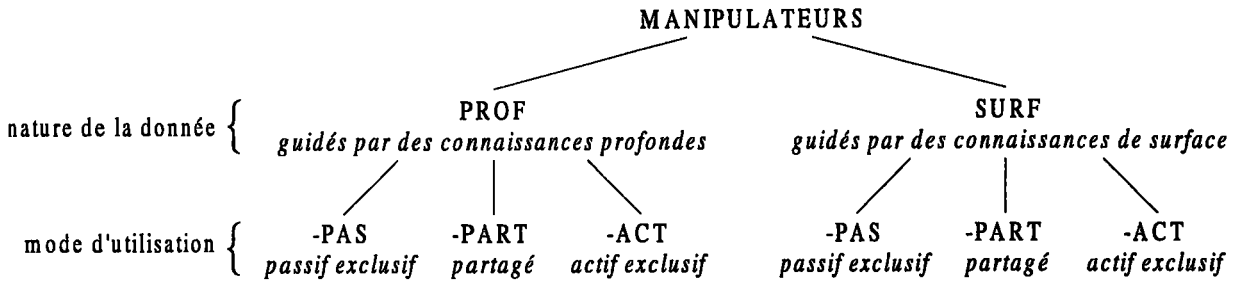



***Figure III.7.*** *Classification des manipulateurs.*

Nous définissons deux critères de classification : la nature de la donnée *D* et le mode d'utilisation du manipulateur. Dans les deux premières parties, nous développons ces critères et nous montrons dans quelles mesures nous les jugeons appropriés. Ensuite, nous évoquons les raisons qui nous ont poussés à abandonner l'idée d'identifier les manipulateurs tout naturellement par leur fonctionnalité.

### *3.3.1.La nature de la donnée.*

Le manipulateur s'articule autour d'une donnée (cf. 3.2. Le lien avec les réacteurs extrinsèques) et tout ce que le manipulateur va définir dépend de cette donnée : la nature de la donnée influe sur le comportement du manipulateur. On définit deux natures possibles pour la donnée : des connaissances profondes (*PROF*) et des connaissances de surface (*SURF*). Pour parvenir à une telle conclusion, il suffit d'analyser le réacteur extrinsèque qui fournit la donnée : son rôle est de traduire l'entité à laquelle

il appartient sous la forme d'une donnée, ce qui revient à donner des connaissances particulières sur l'entité. Or, il existe deux manières de représenter les connaissances : les connaissances profondes et les connaissances de surface [CHO 91b ; FAR 88 ; LIA 91]. Nous avons déjà présenté ces deux types de connaissances en abordant l'IA dans le chapitre 1 (cf. Chapitre 1 *3.1.1 La connaissance requise*).

La classe *PROF* comprend des manipulateurs dont la donnée repose essentiellement sur des connaissances profondes de l'entité d'où la donnée est issue. La donnée utilise des informations concrètes, modélisées sur l'objet ou une combinaison de ces informations pour se déterminer : ces informations sont issues de la représentation canonique de l'entité. À titre d'exemple, le manipulateur manette fait partie de la classe *PROF*, car il se base sur la hauteur du cylindre pour fonctionner et qui est une donnée directement issue de la représentation canonique du cylindre.

Quant à la classe *SURF*, elle comprend les manipulateurs dont la donnée repose principalement sur des connaissances de surface à propos de l'entité. Il s'agit d'informations plus abstraites (dans le sens moins mesurables) qui reflètent souvent des notions de savoir-faire. Ces données symbolisent une capacité de l'entité. Le manipulateur est utilisé dans ce cas pour rendre possible cette capacité. La calculatrice d'expressions grapho-numériques est un manipulateur de la classe *SURF*. Rappelons que c'est un manipulateur associé à un opérateur et qui définit des contraintes entre objets. Pour déterminer avec précision cette donnée, il suffit d'imaginer ce qu'il se produirait si la calculatrice n'existait pas : l'opérateur n'aurait aucun moyen de définir des contraintes sous forme d'expressions grapho-numériques. On en déduit que la donnée sur laquelle la calculatrice repose est la capacité de l'opérateur à définir des contraintes. Le manipulateur fournit alors un moyen de créer des contraintes de manière dynamique grâce au procédé des expressions grapho-numériques. La calculatrice est bien définie grâce à une donnée symbolisant une connaissance de surface.

Ce premier critère de classification concernant la nature de la donnée est consistant, car il se base sur une classification des connaissances qui est déjà établie [CHO 91b ; FAR 88 ; LIA 91] : il ne peut exister de manipulateurs qui ne correspondent pas à l'une des deux classes.

### *3.3.2. Une notion d'activité.*

Le manipulateur est associé implicitement (par transitivité) à une entité par l'intermédiaire de la donnée (cf. figure III.5). Tantôt le manipulateur est lié à l'entité à laquelle il s'applique, tantôt il est lié à l'entité qui l'utilise. Le lien entre l'entité et son manipulateur détermine un certain degré d'activité entre ces deux intervenants. Ainsi, on différencie les manipulateurs par leur mode d'utilisation avec les entités auxquelles ils sont associés et on définit, pour chaque classe *PROF* et *SURF*, les sous-classes :

- *-PAS* passif exclusif, qui indique que le manipulateur ne peut pas être utilisé par l'entité. Le manipulateur manette est passif exclusif pour le cylindre, car ce dernier ne peut utiliser sa manette. Ainsi, la manette que nous avons décrite jusqu'à présent est donc *PROF-PAS* ;
- *-ACT* actif exclusif, qui au contraire précise que c'est l'entité qui utilise son manipulateur. Le manipulateur calculatrice est actif exclusif, puisque seul l'opérateur qui le possède l'utilise. Avec le critère sur la nature de la donnée, la calculatrice est donc *SURF-ACT* ;

- *-PART* partagé, qui combine les deux types précédents de liaison. Le manipulateur est utilisé à la fois par l'entité qui le possède et par d'autres entités. Par exemple, prenons un objet animé de type robot possédant un manipulateur lui permettant d'enregistrer certaines de ses actions. On définit ce manipulateur partagé pour qu'un opérateur ou un autre robot puissent également l'utiliser afin d'écouter les enregistrements effectués par le robot. On retrouve cette notion de partage dans l'utilisation du manipulateur. De plus, ce manipulateur se base sur la capacité de l'objet à enregistrer, donc il fait partie de la classe *SURF-PART*. Cette classe de manipulateurs demande une gestion supplémentaire en ce qui concerne l'accès à ses manipulateurs. En effet, en envisageant qu'un manipulateur puisse être utilisé par plusieurs entités, on risque d'être confronté à des conflits dans son utilisation.

### *3.3.3.Le rejet du critère sur les fonctionnalités.*

Les manipulateurs reflètent des outils de manipulations. Ils sont donc caractérisés par des fonctionnalités. Ce critère de sélection nous a paru intéressant dans un premier temps, car il permet de classer les manipulateurs par leur rôle et d'envisager ainsi des traitements adaptés pour chaque classe de manipulateurs. Pourtant, ce n'est pas un critère fiable ainsi que nous allons le montrer.

Toute la difficulté est de recenser les différentes fonctionnalités qui peuvent définir les manipulateurs. Nous partons d'une application quelconque de CFAO et nous analysons les différentes fonctionnalités qui interviennent "naturellement" au cours du dialogue. À chacune de ces fonctionnalités, on associe une classe de manipulateurs. On distingue les fonctionnalités suivantes :

- le déplacement, qui concerne aussi bien l'opérateur que les objets animés ou non. En effet, on définit dans cette classe des outils de manipulations concernant les modes de déplacement direct (ascenseurs, portes de téléportation ...) ou indirect (plateau pour faire tourner les objets, manette de déplacement d'un objet ...) ;
- la modification, qui peut porter sur l'environnement (outils pour créer des objets, pour créer un fichier de sauvegarde d'objets ...) ou sur les objets (manette de modification de hauteur, calculatrice d'expressions grapho-numériques pour contraindre les objets ...) ;
- l'information, qui regroupe tous les outils permettant d'aider l'environnement animé à comprendre l'environnement virtuel. On inclut aussi bien les zooms, les plans, tous les types de repères pour les problèmes de localisation, que les outils mettant en évidence les objets (explications, visualisation ...) ;
- la désignation, qui définit tous les outils permettant de sélectionner des objets. On peut citer les différents curseurs et les filtres de sélection.

Cette classification soulève un problème. Elle restreint le manipulateur à ne représenter qu'une seule fonctionnalité. Pourtant, il existe de nombreux outils qui combinent plusieurs fonctionnalités : on peut évoquer les Magic Lens™ qui permettent de désigner des objets, de les modifier localement et d'en extraire des informations cachées ou dans un autre format [BIE 93]. Certains outils de manipulations ne trouvent pas clairement leur place dans cette classification.

De plus, on risque de se retrouver avec une multitude d'outils à cause de cette restriction, puisque la moindre fonctionnalité impose alors l'utilisation d'un outil différent. Il s'ensuit des dialogues peu conviviaux, car l'opérateur doit se familiariser avec trop d'outils : il perd un temps considérable par rapport au temps d'utilisation de chaque outil.

Si l'on commence à effectuer des associations de fonctionnalités pour définir de nouvelles classes de manipulateurs, on obtient des dépendances entre les différentes classes qui ne parviennent pas à isoler de manière précise les manipulateurs. De plus, on s'expose à la création d'un nombre considérable de classes de manipulateurs, ce qui n'a pas d'intérêt. On démontre ainsi la fragilité de la classification qui est incapable d'être stable et concise, critères importants pour une bonne classification.

### 3.4.Les manipulateurs généraux.

Parmi les différents manipulateurs que l'on peut construire, on définit des manipulateurs généraux qui se basent sur la définition des manipulateurs (cf. figure III.4), mais qui se distinguent par leur mode de fonctionnement. En effet, on appelle manipulateur général tout manipulateur capable de capturer des informations à l'aide de ses apparences *Ap* pour d'autres manipulateurs.

Par exemple, le manipulateur fenêtre que avons décrit auparavant (cf. 3.1. Définition et exemples) devient un manipulateur général si l'on considère les informations qu'il capture grâce à ses apparences (fenêtre graphique ...) alors qu'elles sont exploitables par d'autres manipulateurs : c'est le cas des désignations dans la fenêtre graphique. Le manipulateur fenêtre détermine le manipulateur (ou un ensemble de manipulateurs) adéquat pour répondre à la situation. Une désignation dans la fenêtre graphique est utile, par exemple, à la calculatrice d'expressions grapho-numériques pour ajouter une donnée graphique à l'expression en construction ou encore au manipulateur manette pour gérer l'utilisation de l'apparence de la manette (si l'opérateur la manipule via la fenêtre graphique).

Le manipulateur général intègre des données sur des manipulateurs, mais aussi des données propres. En effet, il peut posséder un fonctionnement particulier parallèlement au fait de pouvoir déclencher d'autres manipulateurs. Nous nommons composants propres du manipulateur général, tout composant de ses Domaines ou de ses Aides le caractérisant indépendamment des autres manipulateurs.

Soit *MG* un manipulateur général. On suppose que son utilisation peut entraîner celle de manipulateurs parmi les manipulateurs $M_1, .., M_k$. La figure III.8 illustre la définition de *MG*. Son domaine d'utilisation *DUt(MG)* comprend un sous-domaine définissant le domaine d'utilisation propre à *MG* et des domaines d'utilisation $DUt'(M_i)$ déterminés à partir des domaines d'utilisations $DUt(M_i)$ des manipulateurs $M_i$. $DUt'(M_i)$ décrit les entités qui peuvent utiliser *MG* ainsi que des conditions d'utilisation de *MG* dans le but implicite d'utiliser $M_i$ : c'est pourquoi $DUt'(M_i) \subset DUt(M_i)$.

Le domaine d'application de *MG*, *DApp(MG)*, est défini comme *DUt(MG)* puisqu'il contient un sous-domaine propre et des domaines $DApp'(M_i)$ issus de $DApp(M_i)$. Le manipulateur *MG* s'applique à des entités pour son propre compte ou pour le compte de $M_i$ (entités définies dans $DApp'(M_i)$).

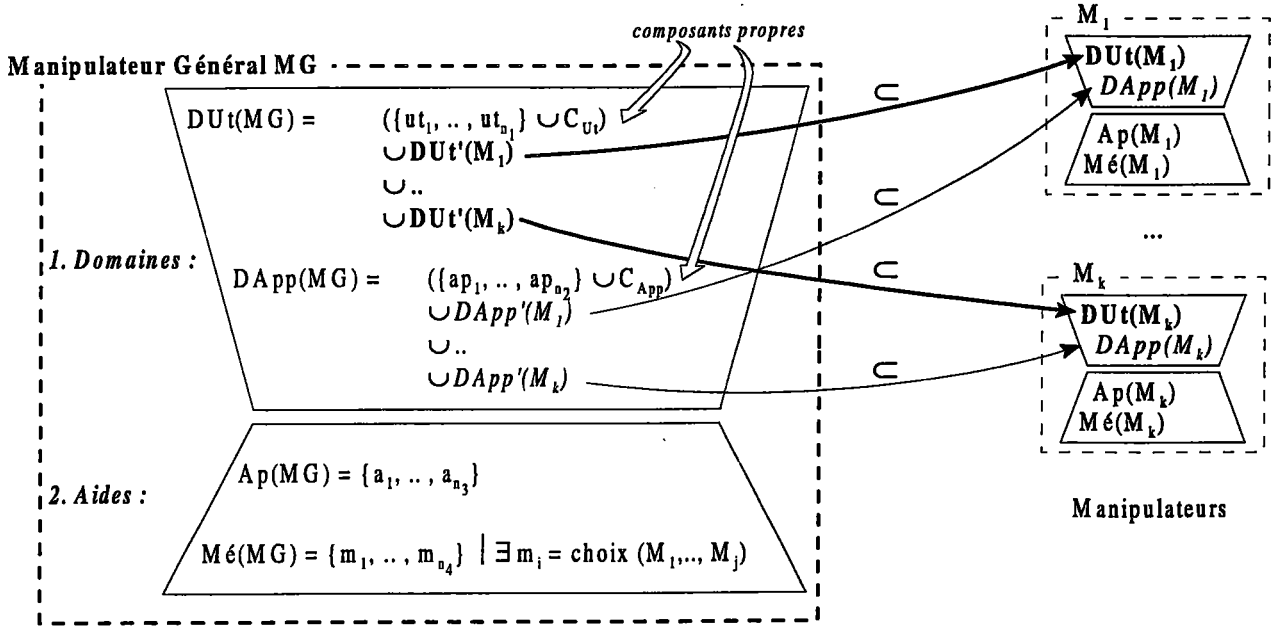



***Figure III.8.** Définition d'un manipulateur général.*

Les apparences de *MG*, *Ap(MG)* sont des composants propres à *MG* et ne proviennent donc pas des manipulateurs $M_i$ qu'il déclenche. La raison essentielle est que seul *MG* est capable de gérer complètement ces apparences : cas de la fenêtre graphique pour le manipulateur général fenêtre.

Tout comme n'importe quel manipulateur, *MG* possède des mécanismes *Mé(MG)* pour décrire son fonctionnement. Parmi ces mécanismes, on définit des mécanismes $m_i$ décrivant les choix que *MG* doit effectuer pour déterminer à quel manipulateur $M_i$ les informations capturées sont destinées.

En définitive, le principe du manipulateur général consiste à définir un manipulateur capable de contrôler d'autres manipulateurs. Le manipulateur général centralise et contrôle des connaissances communes à plusieurs manipulateurs. On peut ainsi envisager grâce à ce manipulateur de partager des connaissances entre manipulateurs, voire d'obtenir des fonctionnements de manipulateurs en parallèle.

### 3.5.Synthèse.

Le concept des manipulateurs que nous avons introduit permet de rassembler les différents outils de manipulations que l'on peut rencontrer dans une application. Pour permettre un plus grand contrôle de ces outils, nous avons défini une méthode de classification des manipulateurs.

De plus, nous avons montré que les manipulateurs permettent de consolider le concept des comportements. En effet, combiné au manipulateur, le réacteur extrinsèque devient plus complet. Jusqu'à présent, son rôle apparaissait sous une forme relativement restreinte ; il permettait de décrire des aides statiques alors qu'elles peuvent devenir maintenant dynamiques grâce aux manipulateurs.

La représentation composée du couple (Domaines, Aides) apporte une contribution en ce qui concerne l'adaptation des outils au dialogue. On définit les outils en fonction de l'environnement (qu'il s'agisse des objets ou des opérateurs) ; les manipulateurs généraux reflètent cet aspect puisqu'ils

déterminent selon la situation des manipulateurs qui interviennent à un instant donné. On se dirige vers des dialogues plus adaptés, car les manipulateurs permettent de décrire aisément des contextes d'aides grâce à sa représentation avec ses Domaines (d'utilisation et d'application) et ses Aides.

En ce qui concerne l'utilisation des manipulateurs au cours du dialogue, on pressent des modifications conceptuelles notoires du modèle de dialogue. D'après la définition même du manipulateur, il apparaît que des phases complètes de dialogue sont incluses dans le manipulateur. Nous concentrons notre attention sur ce point dans le chapitre suivant.

## 4. LES PROPRIÉTÉS DES MANIPULATEURS.

Les manipulateurs regroupent de nombreux outils de manipulations. Le but recherché n'est pas seulement de parvenir à une définition commune, mais on cherche également à déléguer aux manipulateurs la gestion du dialogue qui caractérise chacun des outils qu'ils décrivent. Grâce à sa spécification, le manipulateur possède des propriétés qui permettent de rendre les outils adaptés au dialogue et fournir ainsi un dialogue adaptatif.

Avant de développer, dans la dernière partie, le rôle des manipulateurs dans le modèle de dialogue, nous présentons dans les quatre premières les propriétés qui assurent la fiabilité du concept ainsi qu'une grande souplesse. Nous montrons tout d'abord le caractère intuitif des manipulateurs. Puis, nous expliquons dans quelles mesures ils gardent leur autonomie vis-à-vis des outils physiques. Ensuite, nous insistons sur le caractère adaptatif des manipulateurs, car c'est une propriété essentielle pour préserver la fiabilité du concept. Enfin, la dernière propriété présentée concerne la possibilité de définir des compatibilités sur les manipulateurs afin de maintenir la cohérence des dialogues.

### 4.1.Le caractère intuitif.

On considère que l'on se dirige vers des dialogues adaptatifs dès lors que le système peut déduire un maximum de cas et donne ainsi plus de liberté à l'opérateur. De ce fait, le manipulateur est conçu de sorte qu'il représente un composant intuitif dans le système.

Un composant *c* du système est dit intuitif s'il possède au moins un élément *elt* qui permette d'obtenir des informations sur une entité animée *e* utilisant *c* sans que *e* ait eu besoin d'intervenir (cf. figure III.9). On appelle ces informations, des informations implicites sur *e*, puisqu'elles sont obtenues sans que *e* soit sollicité.

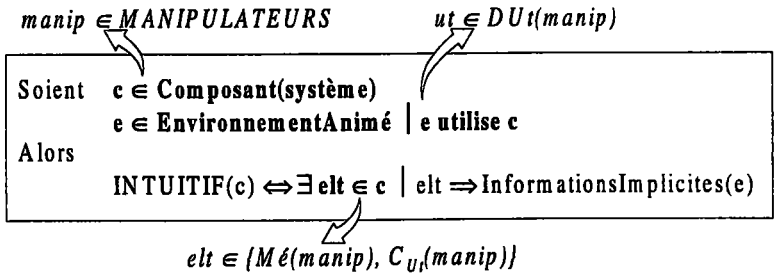



***Figure III.9.** Application du caractère intuitif d'un outil à un manipulateur.*

Tout manipulateur *manip* peut faire partie des composants *c*. En effet, les éléments qui permettent de le rendre intuitif sont :

- ses conditions d'utilisation $C_{ut}(manip)$, qui définissent l'activation implicite du manipulateur *manip*. Il s'agit d'une anticipation des intentions d'une entité du domaine d'utilisation pour utiliser le manipulateur ;
- ses mécanismes *Mé(manip)*, qui incluent la description d'aides implicites aux entités *ut* utilisant *manip*. La calculatrice d'expressions grapho-numériques contient un mécanisme de reconnaissance des données graphiques désignées par l'opérateur. L'opérateur ne spécifie pas quelle donnée graphique il désigne ; le manipulateur reconnaît implicitement les informations sur le type de la donnée graphique désignée. Pour ce manipulateur, on peut même intégrer des mécanismes d'apprentissage de la technique de reconnaissance des données graphiques. Tous ces mécanismes pour rendre le manipulateur intuitif utilisent des techniques d'IA.

Le manipulateur est intuitif dans son activation et dans son fonctionnement en proposant diverses anticipations. De plus, les manipulateurs généraux permettent d'étendre la notion de manipulateur intuitif. En effet, un manipulateur *manip* est intuitif également s'il existe un élément *elt* qui provient d'un manipulateur général *MG* et qui entraîne l'utilisation implicite de *manip*. Par conséquent, l'anticipation d'un manipulateur *manip* est régie d'abord de manière globale par les mécanismes de choix d'un manipulateur général *MG*, puis de manière plus précise par les conditions du domaine d'utilisation du manipulateur *manip*. Le manipulateur est rendu intuitif grâce à un autre manipulateur.

On constate que le manipulateur possède un déclenchement contextuel comme un menu classique. Il est important de préciser qu'un manipulateur n'est pas assimilable à un menu, car il ne s'agit pas seulement d'un style d'interactions. Le menu n'est qu'une apparence possible pour un manipulateur.

### 4.2.L'indépendance avec les outils physiques.

On définit la propriété d'indépendance entre les outils physiques (outils réels proposés à l'opérateur pour interagir) et les manipulateurs en ce qui concerne leur spécification. On dit qu'un ensemble $SB \subset B$ est indépendant de $SA \subset A$ s'il n'existe pas de fonctions de spécification d'éléments de *SB* qui utilisent comme connaissances de base des éléments de *SA* bien qu'elles utilisent des connaissances de *A*. Cette propriété s'applique aux outils physiques et aux manipulateurs (cf. figure III.10).

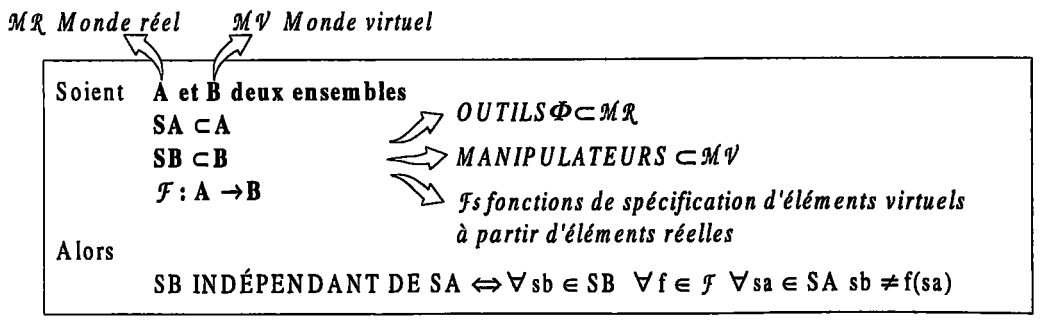



***Figure III.10.*** *Application de l'indépendance au couple (manipulateurs, outils physiques).*

Les manipulateurs *MANIPULATEURS* sont des éléments du monde virtuel $\mathcal{MV}$ et les outils physiques *OUTILS*$\Phi$ sont des éléments du monde réel $\mathcal{MR}$. De manière générale, on note $\mathcal{Fs}$ l'ensemble des fonctions de spécification d'éléments de $\mathcal{MV}$ à partir d'éléments de $\mathcal{MR}$. Spécifier un manipulateur demande des informations sur $\mathcal{MR}$ donc il existe des fonctions $f$ de $\mathcal{Fs}$ dédiées à la spécification des manipulateurs. Toutefois, aucune de ces fonctions $f$ ne se basent sur des outils physiques, ce qui rend les manipulateurs totalement indépendants des outils physiques.

Dans le chapitre 1, nous avons insisté sur l'indépendance entre les techniques de dialogue et les outils physiques (cf. Chapitre 1 *4.1.3. Les nouveaux périphériques*). Cette volonté d'indépendance s'est affirmée avec la RV dont l'un des objectifs est de combler le manque provoqué par l'absence des cinq sens dans le monde virtuel. Cela ne signifie pas forcément recréer le monde réel, mais exploiter les nouvelles capacités offertes. Or, considérer les outils physiques dans la gestion du dialogue implique de garder un lien trop fort avec la réalité et d'encourir le risque d'être restreint par le monde réel.

Le concept des manipulateurs respecte l'idée d'indépendance avec les outils physiques bien qu'à l'origine ce ne soit pas si évident. Le manipulateur représente un outil virtuel, car il fournit un outil de manipulations dans le monde virtuel $\mathcal{MV}$. Il est naturel de penser qu'un outil physique lui correspond. Pourtant, il est plus productif que le manipulateur s'adapte à n'importe quel outil physique. De plus, ceci se confirme si l'on songe à l'instabilité des outils physiques due à leurs progrès continuels.

Bien qu'au moment de leur spécification les manipulateurs soient indépendants de quelconques outils physiques, il se produit tout de même un lien indirect. Le manipulateur fonctionne grâce à des mécanismes et des apparences qui se basent sur des événements. Ces événements représentent les interactions qui peuvent se produire sur l'environnement virtuel. Que ces événements soient dus, par exemple, à l'utilisation d'un casque HMD (Head Mounted Display) ou d'un gant n'a pas d'importance, car ils sont définis en fonction des manipulations possibles dans le monde virtuel et non en fonction des manipulations possibles avec les outils physiques. En fait, on considère que l'utilisation d'un quelconque outil physique donne lieu à une interprétation dans un format standard représentant un type d'événements que le manipulateur sait exploiter (cf. figure III.11). Le manipulateur active alors la manipulation virtuelle correspondante sans avoir eu connaissance a priori de l'outil physique : le lien entre outils physiques et manipulateurs est indirect.

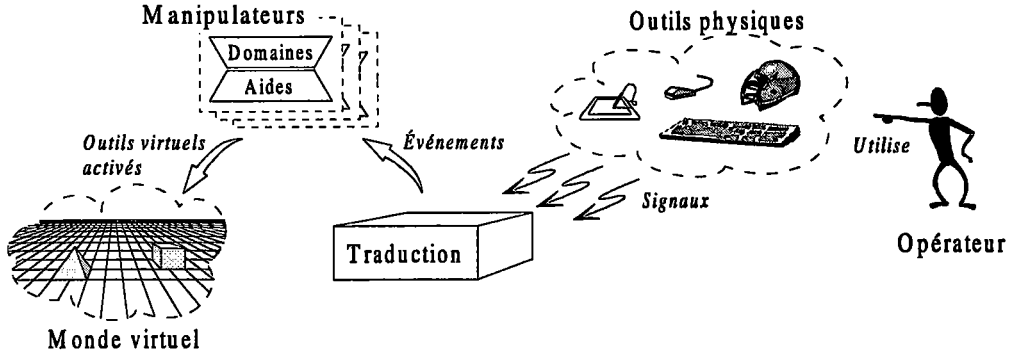



***Figure III.11.*** *Lien indirect entre manipulateurs et outils physiques.*

### 4.3.Le caractère adaptatif.

Notre objectif est de fournir un dialogue adaptatif. Ainsi, il est important de garantir le caractère adaptatif de chaque élément intervenant dans la réalisation du dialogue. Un élément *a* d'un ensemble *A* est adaptatif si et seulement si il peut être enrichi de nouvelles informations (cf. figure III.12).

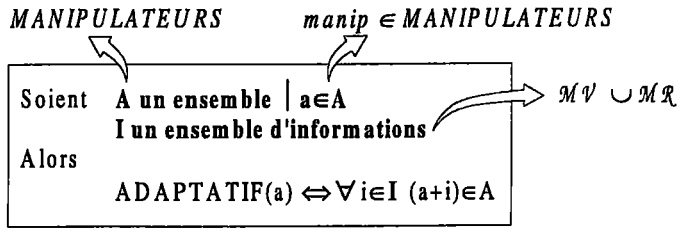



***Figure III.12.*** *Application du caractère adaptatif aux manipulateurs.*

Le caractère adaptatif s'applique aux manipulateurs. Des informations *i* provenant du monde virtuel $\mathcal{MV}$ ou du monde réel $\mathcal{MR}$ peuvent s'ajouter à un manipulateur pour former un manipulateur mieux adapté au contexte *C(t)*. Nous supposons qu'un contrôle s'effectue sur les informations *i* afin de garantir qu'elles n'interfèrent pas avec des informations existantes : nous ne nous préoccupons pas du problème de cohérence bien que nous ayons conscience de la difficulté de gérer un tel problème.

Dans les trois parties suivantes, nous montrons quelles sont ces informations *i* qui permettent à un manipulateur d'évoluer en insistant sur les facultés du manipulateur à s'adapter. Nous expliquons tout d'abord dans quelles mesures des informations *i* peuvent adapter les composants du manipulateur. Puis, nous mettons en valeur le caractère générique du manipulateur. Enfin, nous montrons comment un manipulateur peut s'appliquer à une entité résultant de la composition de plusieurs entités.

#### *4.3.1.Les composants.*

Les informations *i* permettant d'adapter un manipulateur modifient les composants mêmes du manipulateur. En effet, aucun composant du manipulateur n'est réellement figé, qu'il s'agisse de la partie Domaines (domaine d'utilisation *DUt* et domaine d'application *DApp*) ou bien de la partie Aides (mécanismes *Mé* et apparences *Ap*). On dispose d'une grande souplesse dans la spécification du manipulateur grâce à sa décomposition modulaire.

La partie Domaines d'un manipulateur *manip* peut évoluer. Une nouvelle entité peut tout aussi bien être définie pour modifier les entités pouvant utiliser *manip* (i.e. modifier son domaine d'utilisation *DUt(manip)*) ou étendre le domaine d'application de *manip* (i.e. *DApp(manip)*). Notons que l'évolution sur laquelle nous nous basons exprime un changement : il ne s'agit pas forcément d'une extension du manipulateur, il peut également s'agir d'une restriction de sa définition. En outre, la possibilité de définir explicitement *DUt* pour un manipulateur permet d'obtenir des outils de manipulations adaptés à chaque situation. Le principe d'anticipation du manipulateur raisonne en ce sens, car les informations *i* que l'on peut ajouter au manipulateur concernent également les conditions d'utilisation $C_{ut}$ du manipulateur. Ces conditions $C_{ut}$ rendent plus souples *DUt*, car elles l'affinent en apportant la possibilité d'autoriser des détections plus ou moins différentes selon l'entité animée qui interagit.

Les Aides décrites pour le manipulateur sont également flexibles. Tant les mécanismes *Mé* que les apparences *Ap* sont concernés par l'ajout de nouvelles informations *i*. Les mécanismes gérant le fonctionnement du manipulateur sont assimilables souvent à des règles de production (cf. Chapitre 1 *3.1.2. La représentation de la connaissance*). En effet, ils décrivent des opérations que le manipulateur doit effectuer en fonction d'une certaine situation : on est bien dans le cas "Si *condition* alors *action*". Or, il est tout à fait concevable d'autoriser l'ajout, la modification ou la suppression de règles : les informations *i* provoquent des changements concernant la règle dans sa globalité ou de manière plus locale les conditions ou encore les actions. Pour renforcer cette idée, on rappelle que les apparences *Ap* capturent des informations utiles pour certains mécanismes : ces informations constituent des conditions de déclenchements pour les mécanismes. Or, les apparences sont elles aussi adaptables, ce qui entraîne irrémédiablement une évolution des mécanismes. En résumé, les mécanismes ont une capacité d'adaptation explicite (modification des règles globalement ou localement) et implicite (modification des apparences impliquant une éventuelle adaptation des mécanismes associés).

Envisager la modification de l'ensemble des apparences *Ap* d'un manipulateur est certes révélateur d'une capacité d'extension du manipulateur. Toutefois, il est plus intéressant d'insister sur la possibilité pour un manipulateur de posséder plusieurs apparences. Cela signifie d'une part, qu'un manipulateur peut avoir plusieurs fonctionnalités et d'autre part que, pour une même fonctionnalité, des apparences différentes peuvent être proposées en fonction du contexte *C(t)* (cf. 2. Les possibilités du dialogue pour retrouver la définition du contexte). Prenons le manipulateur associé à l'outil de construction d'expressions grapho-numériques. Pour saisir l'expression, il utilise comme apparence un certain type de calculatrice. En réalité, le manipulateur pourrait disposer d'apparences différentes pour saisir l'expression : calculatrice scientifique, standard, boîtes de saisie textuelle (pas de boutons pour saisir les données alphanumériques) ...

Le manipulateur possède la propriété de s'adapter dans le mesure où l'utilisateur peut l'enrichir à tout moment de nouvelles informations. Tout comme les comportements permettent d'adapter les objets, le manipulateur permet d'adapter les outils de manipulations. Il s'ensuit un dialogue plus adaptatif, car toutes les entités intervenant dans l'application s'adaptent les unes aux autres.

#### *4.3.2.Les manipulateurs génériques.*

Tout manipulateur est adaptatif et peut donc servir de base à d'autres manipulateurs. On définit une bibliothèque contenant des manipulateurs qui représentent des manipulations couramment utilisées et qui sont décrits de manière générique. On appelle ces manipulateurs, des manipulateurs génériques.

À partir de la bibliothèque, tout utilisateur qu'il soit concepteur de dialogue ou opérateur peut spécifier ses propres manipulateurs (cf. figure III.13) ; il est aidé dans sa spécification puisqu'il dispose d'une base de travail. Il peut s'attacher à ce qui le préoccupe vraiment i.e. créer un dialogue adapté s'il s'agit d'un concepteur et manipuler l'application s'il s'agit d'un opérateur. On évite par la même occasion de voir une quantité d'outils de manipulations similaires encombrer l'application.

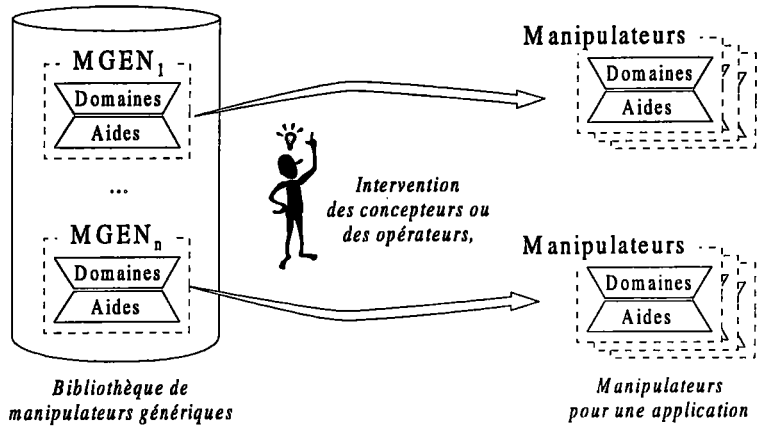



***Figure III.13.** Manipulateurs génériques.*

On se dirige vers une indépendance de l'entrée des données. L'entité animée appartenant au domaine d'utilisation *DUt* du manipulateur aura même la possibilité de choisir l'apparence qu'elle souhaite utiliser pour interagir et le manipulateur pourra ainsi capturer des informations pertinentes. Cette possibilité ouvre une voie vers la spécification du manipulateur pendant le fonctionnement de l'application : les opérateurs apportent leurs propres connaissances en matière de manipulations.

Nous tenons à préciser que les notions de manipulateur générique et de manipulateur général sont deux notions distinctes. Le manipulateur général est un manipulateur dont le fonctionnement est particulier, car il consiste à capturer des informations pour résoudre les cas de conflits entre manipulateurs communs par leur mode d'activation : il dirige le dialogue vers le (ou les) bon(s) manipulateur(s). Le manipulateur générique, quant à lui, appartient à une bibliothèque et représente un support de description de manipulateurs. Ainsi, il peut servir à décrire un manipulateur général.

*4.3.3.La composition d'entités.*

Parmi les informations *i* qui entraînent une adaptation des manipulateurs, on distingue la composition d'entités (cf. figure III.14). Supposons qu'on dispose d'un manipulateur *manip* qui définit un type de manipulations sur une entité *e*.

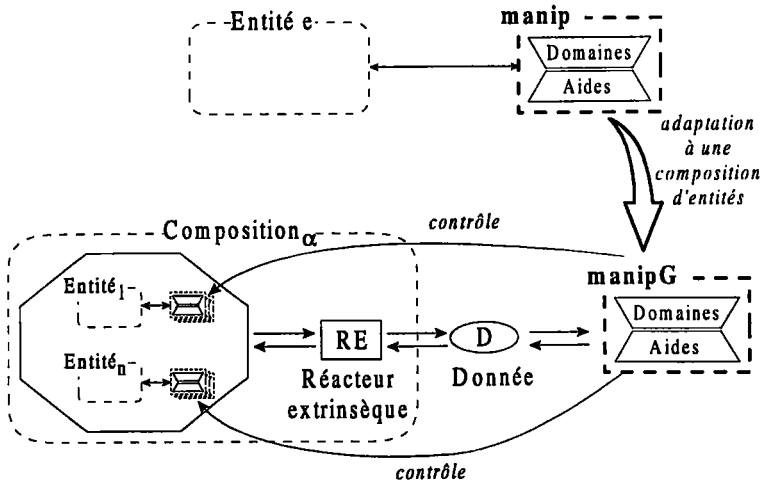



***Figure III.14.** Adaptation d'un manipulateur à une composition d'entités.*

L'information *i* décrivant une composition d'entités entraîne la création d'un manipulateur général *manipG* reposant sur *manip* dans le cas où on désire créer le même type de manipulations que *manip* sur la composition : on considère qu'il s'agit d'une adaptation d'un manipulateur *manip*. Deux caractéristiques concernent *manipG* : il associé à une composition d'entités et il est général.

La première caractéristique se justifie par la possibilité pour un réacteur extrinsèque *RE* de s'appliquer à une famille d'entités $Composition_{\alpha}$ (cf. Chapitre 2 3.3. Synthèse). *RE* traduit cet ensemble en une donnée *D* exploitée par un manipulateur *manipG* pour la rendre plus dynamique et en offrir une réelle manipulation. Il s'ensuit que le manipulateur *manipG* est associé à une composition d'entités qui est considérée comme une entité du système.

La seconde caractéristique de *manipG* répond aux problèmes de conflits qui pourraient se produire entre *manipG* et les manipulateurs associés aux différentes entités de $Composition_{\alpha}$. Considérons le manipulateur manette de modification de la hauteur du cylindre. On envisage maintenant un manipulateur offrant un outil identique mais s'appliquant sur un objet poutre issu de la composition de trois objets : deux supports cylindriques et une barre horizontale parallélépipédique. Utiliser le manipulateur de la poutre modifie la hauteur de la poutre, ce qui revient à modifier la hauteur des deux cylindres et à descendre le parallélépipède. Le manipulateur de la poutre contrôle donc l'ensemble des entités : il désactive le manipulateur manette de chaque support cylindrique. Ce manipulateur a donc agi sur le déclenchement des manipulateurs liés aux objets composants la poutre. Il se comporte donc comme un manipulateur général en contrôlant l'utilisation des éventuels manipulateurs liés aux entités.

Un manipulateur *M* appartenant à une entité $Entité_i$ de la composition risque d'interférer sur le fonctionnement de *manipG* si ses domaines *DUt* et *DApp* ont des points communs avec ceux de *manipG*. En d'autres termes, *M* et *manipG* peuvent se déclencher en même temps et agir sur des entités communes, ce qui peut provoquer des conflits dans le dialogue à proposer à l'opérateur. Ce problème peut être détecté au moment de la spécification de *manipG*. En effet, connaissant la composition à laquelle *manipG* est associé, le système peut détecter les risques de conflits en comparant les domaines de *manipG* avec ceux des manipulateurs définis pour chaque entité de la composition et faire intervenir l'utilisateur pour ajouter à *manipG* des mécanismes définissant la stratégie à adopter : *manipG* établit un contrôle sur ces manipulateurs.

La détection des conflits entre *manipG* et un manipulateur *M* peut devenir très vite complexe et entraîner une réaction en chaîne pour lever des ambiguïtés, dans les cas où :

- *M* est lui-même un manipulateur associé à une composition. Il faut définir la position de *manipG* face aux manipulateurs contrôlés par *M* ;
- un manipulateur général déclenche *M*. Il faut définir la position de ce manipulateur général face à *manipG* i.e. définir s'il doit, par exemple, déclencher *manipG* à la place de *M*.

Le système aide l'utilisateur à définir *manipG* en prévenant les ambiguïtés, car cet utilisateur n'a pas forcément connaissance de tous les manipulateurs déjà spécifiés dans la mesure où plusieurs utilisateurs (concepteurs ou opérateurs) peuvent spécifier des manipulateurs.

Tout comme les objets peuvent être créés dans les applications de manière progressive par assemblages, les manipulateurs peuvent évoluer vers des manipulateurs généraux qui étendent leur champ d'actions sur des ensembles d'entités.

### 4.4.Le maintien de la cohérence du dialogue.

On définit la cohérence d'un dialogue par l'adéquation entre les interactions de l'opérateur et l'interprétation que le système en fait. Proposer un dialogue homme/machine cohérent revient pour le système à disposer de mécanismes capables de comprendre l'opérateur à tout moment. Le système peut mieux aider l'opérateur.

La cohérence dans le dialogue est un paramètre difficile à maîtriser si l'on considère les fils d'activités multiples de l'opérateur. Le système n'a pas un contrôle total sur le dialogue, car il ne connaît pas a priori les chemins empruntés par l'opérateur pour atteindre son but. Nous présentons dans cette partie le rôle que peut tenir le manipulateur. Dans un premier temps, nous insistons sur le contrôle que l'on peut apporter sur le dialogue grâce aux manipulateurs. Puis, dans un second temps, nous expliquons une nouvelle technique permettant d'appréhender le manipulateur sous un autre angle pour favoriser un dialogue aidant toujours plus l'opérateur : nous présentons les compatibilités sur les manipulateurs.

#### *4.4.1.Le contrôle du dialogue.*

Un dialogue devient adaptatif dès lors que le système est capable d'aider l'opérateur, notamment en donnant des explications. Le système doit d'abord comprendre lui-même ce qui se passe. Il s'agit pour lui de réussir à contrôler les interactions de l'opérateur. Pour y parvenir, une solution est de prendre le problème à sa source : les interactions. Ces dernières se produisent lorsque l'opérateur effectue des manipulations. Ce sont donc les outils de manipulations, par l'intermédiaire des manipulateurs, qui peuvent apporter des réponses.

De par leur définition, les manipulateurs apportent un contrôle au dialogue. Ils sont chargés de guider l'opérateur dans ses manipulations. Leur spécification est donc capitale. Quoi qu'il arrive, les chances pour obtenir un manipulateur en accord avec l'opérateur sont importantes étant donné les possibilités d'adaptation du manipulateur.

Disposer d'un concept tel que celui des manipulateurs pour gérer les outils de manipulations est d'autant plus intéressant qu'on peut mieux gérer ce qui se passe pendant le dialogue. En effet, on utilise le manipulateur pour effectuer des traitements implicites favorisant l'étude de la cohérence dans le dialogue. Ainsi, on peut étudier les raisons qui poussent un opérateur (ou un objet animé) à utiliser un outil de manipulations. Nous nous étudions la méthode à utiliser dans la partie suivante.

#### *4.4.2.Les thèmes et les compatibilités.*

La technique de contrôle de cohérence que nous introduisons a pour but d'analyser la situation présente lorsqu'un manipulateur est utilisé et d'aboutir à d'éventuels aménagements de

l'environnement pour aider l'opérateur. Elle repose sur deux principes : une analyse thématique inspirée du modèle de dialogue de [GRA 94] présenté dans le chapitre 1 (cf. Chapitre 1 *3.2.2. La détection*) et une gestion de compatibilités sur les manipulateurs qui reprend le principe des compatibilités sur les menus utilisé dans SACADO (cf. Chapitre 2 2.1. Les concepts de base).

L'analyse thématique consiste à contrôler la cohérence d'un propos à l'autre et d'en déduire un indice caractérisant la situation dans laquelle l'opérateur se trouve par rapport au contexte. À partir de cet indice, le système peut réagir et fournir des aides adaptées au contexte. L'avantage de la méthode est qu'on ne demande aucune intervention explicite de la part de l'opérateur : tout est implicite et s'effectue pendant les interactions de l'opérateur. En définitive, il se produit une réelle anticipation d'explications sur le comportement de l'opérateur. Cette technique assure un contrôle sur les fils d'activités multiples de l'opérateur, voire permet de les réduire sans contrainte. En effet, grâce à l'aide implicite apportée, l'opérateur devient plus expérimenté dans l'utilisation du système, il atteint son but plus rapidement.

Pour établir effectivement un lien entre les interactions de l'opérateur, il faut comprendre ce qu'il fait et ce qui le motive. Ainsi, en accord avec l'analyse thématique, nous définissons des thèmes $T_i$ pour mieux cerner l'opérateur. Il s'agit de regrouper les actions globales en fonction des types de manipulations qu'un opérateur peut effectuer sur l'environnement (cf. figure III.15). Le thème $T_i$ poursuivi par un opérateur (ou un objet animé) connaît à tout instant $t$ l'état du dialogue le concernant i.e. l'état de l'action globale qui est en train de s'effectuer.

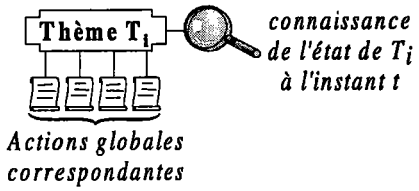



***Figure III.15. Définition d'un thème.***

L'ensemble de ces thèmes constitue les buts possibles de l'opérateur. Les thèmes dans la décomposition des actions globales qu'ils proposent sont assimilables à des menus terminaux ou non du modèle de dialogue de SACADO auxquels on associe directement les actions globales. Nous insistons sur le fait qu'un thème n'est pas un menu, son rôle est tout à fait différent ; on note simplement des similitudes dans la capacité à ordonner les actions globales. Les thèmes ont un caractère très général ; il peut s'agir par exemple d'un thème de création pour regrouper toutes les actions globales concernant des créations d'objets ou encore d'un thème de modification pour décrire toutes les actions globales effectuant des modifications. La définition des thèmes est laissée au concepteur. Ainsi, il peut adapter la portée du contrôle de cohérence à sa manière. Il faut savoir que plus les thèmes sont précis (i.e. proches d'une action globale), plus le contrôle de cohérence sera précis. Toutefois, si l'on définit un thème par action globale, cela signifie que chaque but de l'opérateur se résume à une action globale. Ainsi, on perd le contrôle de cohérence global permettant de cerner des buts plus complexes (plusieurs actions globales pour un thème) et permettant de capturer des informations de plus haut niveau sur les stratégies de raisonnement de l'opérateur.

Lorsqu'un opérateur interagit, il se trouve dans un thème particulier. L'analyse thématique intervient dès que le thème en cours risque d'être modifié i.e. dès que l'opérateur intervient. Elle va définir à cet instant précis un indice en fonction de la situation présente (le thème en cours) et en fonction du thème qui va se poursuivre suite à l'intervention de l'opérateur. On distingue trois types d'indices:

- indice de poursuite, qui indique que le thème engagé se poursuit normalement. On se dirige vers une spécialisation du thème. L'opérateur est cohérent dans ses actes, car il continue ce qu'il a commencé ;
- indice de déviation, qui indique que le thème engagé est momentanément interrompu. L'opérateur est cohérent avec lui-même, car il exprime le besoin d'effectuer une action annexe, soit parce qu'il l'a oubliée, soit parce qu'elle l'aide dans son dialogue. Cet aspect reflète le discours classique dans lequel l'interlocuteur peut faire des parenthèses pour exprimer une idée annexe qu'il juge utile pour la circonstance. Ce type de comportement peut révéler des hésitations de la part de l'opérateur pour atteindre son but. Le système apporte dans ce cas des aides, voire des explications ;
- indice d'incohérence, qui indique un changement de thème radical. L'opérateur s'est apparemment trompé dans ses manipulations, son but est maintenant différent. Le système va devoir opérer une annulation du thème engagé, ce qui revient à annuler les actions globales que l'opérateur a déjà effectuées dans ce thème. De plus, il propose des aides adaptées, car le dialogue semble mal engagé.

Pour définir l'indice thématique, il faut arriver à faire correspondre les thèmes et les interactions. Pour établir cette correspondance, nous utilisons les manipulateurs : ils contiennent suffisamment de sémantique sur les manipulations de l'opérateur pour déterminer avec précision le thème abordé.

Les manipulateurs décrivent souvent plusieurs fonctionnalités (cf. *3.3.3. Le rejet du critère sur les fonctionnalités*). Cela signifie qu'un manipulateur peut agir dans des thèmes différents. De plus, l'indice thématique ne peut être défini au préalable i.e. on ne peut dire que lorsque le manipulateur *M* est activé dans le thème *T*, l'indice thématique est *I*. Ainsi, on définit des compatibilités entre les manipulateurs et les thèmes pour permettre de déduire aisément l'indice thématique. En fonction du thème en cours et de l'utilisation faite du manipulateur, on définit le manipulateur comme :

- local, s'il permet de poursuivre le thème. Il s'agit de l'indice de poursuite ;
- immédiat, s'il permet d'effectuer une manipulation annexe. On en déduit un indice de déviation ;
- différé, s'il permet d'abandonner le thème actuel. On en déduit un indice d'incohérence.

Non seulement, la compatibilité sur le manipulateur permet de définir l'indice thématique et de faire réagir le système en conséquence, mais elle procure un moyen pour le système d'obtenir des informations plus précises sur les manipulations. Par exemple, si le manipulateur est différé, le système peut utiliser le manipulateur pour fournir des explications plus pertinentes.

Nous avons repris la philosophie des compatibilités des menus du modèle de dialogue de SACADO pour l'appliquer aux manipulateurs. Toutefois, il existe une différence fondamentale qui concerne la

connaissance de la compatibilité. Contrairement au manipulateur, le système connaît la compatibilité d'un menu avant même de l'utiliser ; les compatibilités sur les manipulateurs sont conditionnées. En effet, selon le thème en cours, le résultat du manipulateur (i.e. la fonctionnalité qu'il propose à cet instant) initie la compatibilité. Le manipulateur intègre dans ses mécanismes *Mé* des règles permettant de la déterminer.

Pour montrer cet aspect des compatibilités, prenons l'exemple d'un manipulateur plan dont l'apparence est une fenêtre graphique visualisant la scène projetée sur un plan et représentée de manière synthétique. L'outil proposé est assimilable à une carte routière virtuelle. Ce manipulateur donne une vue globale de la scène et permet les manipulations suivantes : sélection d'un objet, désignation d'un emplacement pour créer un nouvel objet et désignation de la nouvelle position à laquelle l'opérateur veut se rendre dans la scène 3D. Supposons que le système possède un thème concernant les créations d'objets. Pour ce thème donné, le manipulateur plan contient un mécanisme détectant sa compatibilité en fonction de son utilisation :

- il est local, si l'opérateur a désigné un emplacement de construction disponible pour créer un objet ;
- il est immédiat, si l'opérateur a désigné un emplacement accessible pour lui. L'opérateur entre dans un thème de déplacement avant de revenir au thème de création ;
- il est immédiat, si l'opérateur désigne un objet existant. L'opérateur entre dans un thème de modification d'objet, le thème de création est interrompu momentanément.

Le contrôle de cohérence que nous proposons s'applique à l'opérateur. Toutefois, il peut également concerner un objet animé, car ce dernier peut représenter une entité instable du système tout comme un opérateur. En effet, par exemple dans les systèmes de simulation de robots, on tente de laisser une certaine autonomie aux robots, ce qui implique des risques d'erreurs dans les interactions avec ces robots : le contrôle de cohérence est d'autant plus souhaitable qu'il respecte le robot en tant qu'individu, car il est implicite et ne restreint pas ses interactions.

### 4.5.Le fonctionnement du dialogue.

Le modèle de dialogue utilisé dans SACADO n'intègre pas le concept des manipulateurs. Notre propos consiste à présenter comment combiner les manipulateurs aux éléments caractéristiques du modèle de dialogue. Il ne s'agit pas de remettre en cause les fondements du modèle, car non seulement nous en avons montré les atouts (génération, complétude dans la description ...), mais aussi nous les avons utilisés comme base implicite des manipulateurs en introduisant les réacteurs extrinsèques comme origine des manipulateurs. Ainsi, nous développons le fonctionnement du dialogue intégrant les manipulateurs ; nous illustrons cette analyse à l'aide de la figure III.6.

Le dialogue a évolué ; il n'est plus dirigé par les interactions en attente d'une intervention de l'opérateur, mais il est entièrement contrôlé par les manipulateurs. La raison est simple : les manipulateurs sont au cœur des manipulations, ce sont les premiers éléments avertis d'un quelconque changement du système.

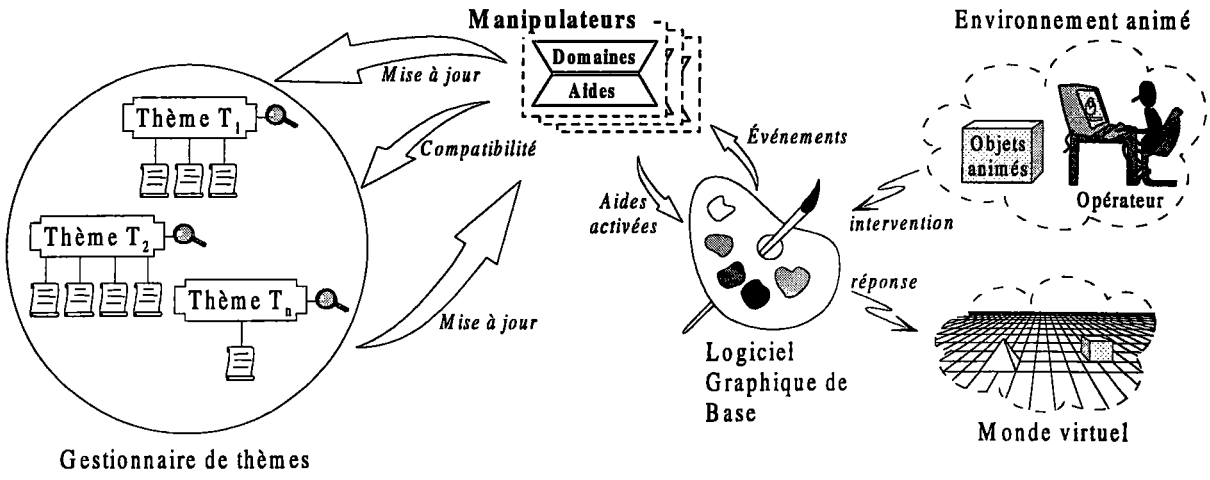



***Figure III.6.*** *Dialogue contrôlé par les manipulateurs.*

Lorsqu'une entité animée intervient, c'est pour effectuer une manipulation ; le LGB (Logiciel Graphique de Base) transmet le signal émis (dépendant des outils physiques) au manipulateur *M* concerné sous forme d'un événement standard. Le manipulateur gère l'information grâce à ses mécanismes *Mé*. Il peut proposer des réactions immédiates en activant ses aides (apparences *Ap* et mécanismes *Mé*) dans le cas où l'intervention de l'entité animée concerne des éléments contrôlés par le manipulateur : par exemple, il s'agit du déclenchement du manipulateur qui entraîne la visualisation d'une de ses apparences. Notons que toute réponse est donnée via le LGB qui connaît le matériel utilisé et est donc capable de transmettre la réponse au monde virtuel. Une fois l'information interprétée par le manipulateur, ce dernier a suffisamment d'informations pour évaluer sa compatibilité par rapport au thème abordé par l'opérateur (le thème courant) et transmettre l'information à l'application si elle peut l'exploiter.

Pour assurer une étroite coopération entre les manipulateurs et les actions globales contrôlées par les thèmes, un gestionnaire de thèmes est instauré : il constitue une passerelle entre les manipulateurs et le thème courant. Son rôle consiste à :

- récupérer la compatibilité du manipulateur *M* utilisé ;
- déduire l'indice thématique en fonction de la compatibilité ;
- engager les éventuelles aides induites par ce contrôle de cohérence, ce qui revient à activer des manipulateurs particuliers ou mettre à jour l'état de ces manipulateurs (activer des mécanismes et/ou des apparences) ;
- gérer le thème courant de l'opérateur. Suite au contrôle de cohérence, le thème a peut-être changé momentanément ou non. Le gestionnaire se charge de l'annulation et de la restauration des thèmes.
- récupérer l'information capturée par le manipulateur *M* ;
- transmettre cette information au thème courant pour qu'il se poursuive (mise à jour de l'application) ;
- mettre le thème courant en attente lorsque ce dernier a fini d'exploiter l'information ;
- transmettre d'éventuelles informations utiles à des manipulateurs.

Poursuivre le thème courant signifie qu'une de ses actions globales va pouvoir continuer à s'exécuter grâce à l'information transmise par le manipulateur et ceci, jusqu'à ce que cette information ne soit plus suffisante pour l'action globale. Ainsi, l'action globale est toujours perçue comme une suite d'états à effectuer : dès qu'elle se met en attente d'informations, l'état correspond à une interaction. Le concept de base du modèle de dialogue est donc respecté. La différence est que les actions globales sont déchargées des états propres à la manipulation. Chaque manipulateur possède son propre dialogue, car il est dédié à la manipulation.

Le problème que nous avons soulevé (cf. *2.2.3. L'intégration au modèle de dialogue*) pour l'intégration de la calculatrice d'expressions grapho-numériques ne se pose plus : le dialogue concernant la manipulation de l'outil (i.e. la saisie de l'expression) a sa place dans le modèle de dialogue sous la forme d'un manipulateur. Ce n'est plus la peine de créer un simulacre de domaine annexe dans lequel on était par exemple obligé de définir certains boutons de la calculatrice comme des menus du domaine : grâce au manipulateur ces boutons ont une place à part entière, car ils constituent une partie de ses apparences.

## 5. Conclusion.

Les interfaces devenant toujours plus performantes, les manipulations proposées à l'opérateur ne se contentent plus d'effectuer strictement les fonctionnalités de l'application. Elles intègrent des fonctionnalités propres qui leur permettent d'offrir des aides adaptées aux opérateurs. Ainsi, le succès d'une application devient dépendant de la performance des outils de manipulations qu'elle propose.

Devant l'importance prise par les outils de manipulations, un contrôle est nécessaire. Dans ce chapitre, nous avons présenté un concept permettant de décrire tout type d'outils : le manipulateur. On assure une gestion contrôlée suffisamment souple pour considérer des outils existants [BIE 93 ; CHU 95] et permettre la définition d'outils de haut niveau. L'annexe B décrit des exemples de manipulateurs dans une application ; il s'agit de la spécification d'un travail d'implémentation que nous avons réalisé.

Nous avons sans cesse gardé à l'esprit que le manipulateur doit favoriser la conception d'un dialogue adaptatif. C'est pour cette raison qu'une grande partie de ce chapitre a été consacrée à la description des propriétés des manipulateurs. On peut affirmer que le manipulateur contribue à la création de dialogues adaptatifs et ceci de différentes manières. Tout d'abord, son comportement lui permet d'intégrer des aides pertinentes et implicites pour l'opérateur : adaptations, détections, anticipations, contrôle de cohérence. En outre, les manipulateurs gèrent eux-mêmes leurs propres conflits : nous avons introduit le concept de manipulateur général qui choisit parmi un ensemble de manipulateurs candidats à une manipulation le (ou les plus) approprié(s). Par conséquent, le concept que nous proposons apporte des solutions sérieuses pour répondre au paradigme qui préconise de "fournir les bons outils au bon moment".

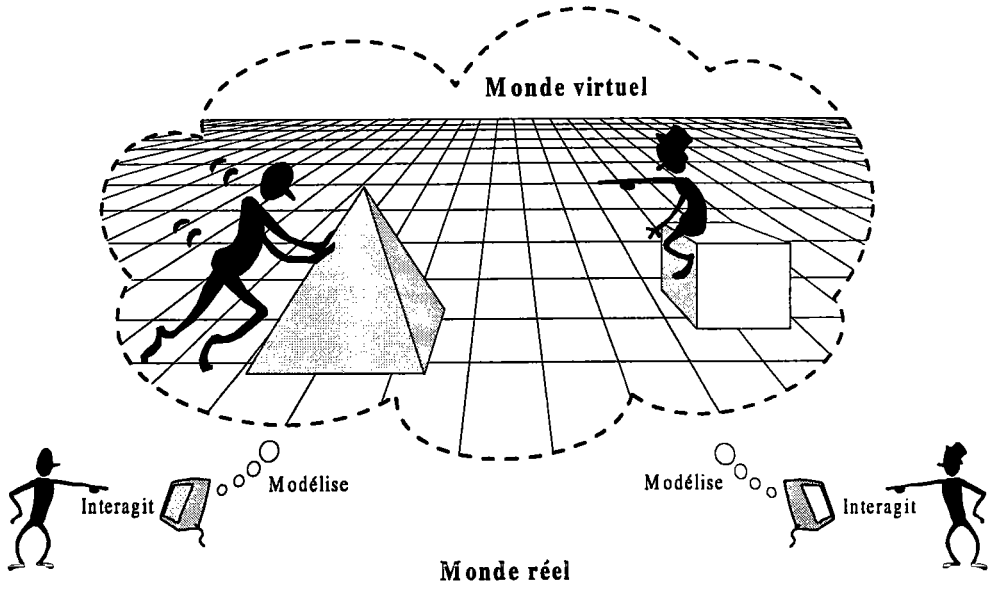
De plus, nous avons montré comment concevoir le dialogue en considérant les manipulateurs. Les bases du modèle de dialogue présenté au chapitre 2 n'ont pas été remises en question. Le concept des

manipulateurs s'intègre au modèle de dialogue assez naturellement, ce qui renforce notre étude. Nous avons tout d'abord montré le lien avec les comportements. Non seulement, les manipulateurs s'accordent sans équivoque aux réacteurs extrinsèques, mais ils nous ont permis de préciser ce type de comportements qui a pu prendre toute son importance. Nous avons ensuite montré le fonctionnement de dialogue comprenant les manipulateurs. Ils permettent de soulager les actions globales des actions propres aux outils de manipulations. La modularité apportée permet une grande souplesse dans l'application. Si le manipulateur ne convient pas, on le modifie ou on le change sans que les actions globales en soient affectées : l'application ne change pas fondamentalement. Ainsi, les manipulateurs marquent encore plus l'indépendance entre l'application et l'interface.

L'opérateur n'est pas gêné par les outils de manipulations i.e. il n'est pas préoccupé par ce qui le pousse à les utiliser, il a un but à atteindre et est guidé par ce but. Il n'est pas contraint par les manipulateurs, bien au contraire ils lui apportent des aides considérables. Les manipulateurs sont le reflet des capacités des opérateurs et du système. Ainsi, on oriente le système complètement en fonction des actes de l'opérateur, ce qui nous permet de souligner l'importance de l'opérateur. Dans le chapitre suivant nous portons justement une attention toute particulière sur l'opérateur en tant qu'entité à part entière pour le dialogue.

# CHAPITRE 4.

# MODÉLISATION DE L'OPÉRATEUR.



## 1. INTRODUCTION.

Parmi les nombreuses définitions de la CAO, se distinguent deux définitions qui ont la particularité de mettre en avant l'aspect informatique [GAR 91] :

- technique dans laquelle l'homme et l'ordinateur sont rassemblés pour résoudre des problèmes techniques dans une équipe qui associe étroitement les meilleures qualités de chacun d'eux, de telle sorte que l'équipe travaille mieux que séparément ;
- technique qui s'occupe de la création des données qui décrivent l'objet à concevoir, et de la génération des informations nécessaires à la fabrication de cet objet à partir de ces données.

En se basant sur ces deux définitions, on constate que les éléments-clés de la CAO sont les objets et l'opérateur. En effet, ces deux éléments constituent les pôles de la CAO : d'un côté, on a l'opérateur qui va utiliser le système et de l'autre côté, on a les objets qu'il va créer par l'intermédiaire du système.

Les objets font partie intégrante du système, car ils sont représentés informatiquement à travers un modèle. Ce modèle permet de stocker toutes les informations concernant les objets tout au long de leur évolution. Nous avons montré dans les chapitres 2 et 3 comment améliorer leur modélisation afin de favoriser le dialogue : description des comportements et des manipulateurs dans un modèle générique.

Jusqu'à présent, l'opérateur est perçu par le système à travers le dialogue. C'est l'amélioration dans la modélisation des objets qui nous a permis de mieux comprendre l'opérateur. Toutefois, avec l'introduction du concept des manipulateurs, on va au delà d'une prise en compte de l'opérateur par les objets. En effet, les manipulateurs ne sont pas toujours caractéristiques d'objets ; certains sont définis spécialement pour des opérateurs. La calculatrice d'expressions grapho-numériques en est un exemple, puisqu'elle n'est pas dédiée à un objet, mais réellement à l'opérateur qui peut l'utiliser. On arrive à considérer de plus en plus d'informations concernant l'opérateur tant et si bien que les objets ne suffisent plus pour intégrer ces données. Nous sommes partis de cette constatation pour poser le problème de la modélisation de l'opérateur en tant qu'entité à part entière du système.

Afin de mieux cerner l'opérateur et de mettre en évidence toute la connaissance qu'il peut apporter au système par sa modélisation, la première partie est consacrée aux différentes implications de l'opérateur dans le système. Ensuite, nous proposons un modèle de représentation des opérateurs en accord avec les concepts de dialogue décrits dans les chapitres précédents. Puis, nous procédons à une analogie entre les opérateurs et les agents rencontrés dans les systèmes multi-agents afin d'améliorer la modélisation des opérateurs. Nous nous attardons tout particulièrement sur la façon de gérer la communication entre opérateurs. Enfin, nous présentons l'implémentation que nous avons réalisée afin d'illustrer nos travaux sur la modélisation de l'opérateur ainsi que sur les manipulateurs.

## 2. LES IMPLICATIONS DE L'OPÉRATEUR.

Jusqu'à présent, nous n'avons cessé d'insister sur l'importance du dialogue homme/machine : les systèmes tentent de fournir des dialogues toujours plus cohérents et adaptés à l'opérateur. Devant ce

phénomène, l'opérateur voit son rôle évoluer. On ne le considère plus comme le simple utilisateur qui va manipuler l'application, mais véritablement comme une personne qui possède des connaissances qui lui sont propres et qu'elle va utiliser pour interagir. Le système intègre donc des mécanismes pour tenter de capturer et comprendre ces connaissances afin de mieux maîtriser l'opérateur. Ainsi, l'opérateur est impliqué dans le système dès la spécification de l'application.

Nous montrons les implications de l'opérateur dans l'obtention d'un dialogue adaptatif afin de souligner l'intérêt de définir un véritable modèle de l'opérateur. Nous mettons en évidence, dans les trois premières parties, le rôle de l'opérateur dans les mécanismes mis en jeu ; nous montrons dans quelles mesures ce rôle revêt un caractère moins explicite que le simple fait d'utiliser l'application. Pour finir, nous présentons l'opérateur dans les mondes virtuels dans lesquels il tient un rôle important.

### 2.1.La construction du dialogue.

Dans le chapitre 2, nous avons présenté une technique duale pour construire le dialogue. Il s'agit de permettre à des personnes, même inexpérimentées dans la modélisation du dialogue, de spécifier le dialogue qu'elles envisagent pour leur application. Ainsi, l'opérateur peut intervenir dès l'étape de construction du dialogue. Il peut décrire lui-même ce qu'il désire tout en bénéficiant d'un contrôle dans sa description puisqu'un concepteur expérimenté peut toujours intervenir par la suite.

L'opérateur donne des informations à propos des comportements des objets dans l'application. Le système considère donc l'opérateur à travers les objets et leurs comportements. L'avantage de la méthode est que chaque opérateur peut apporter des comportements différents sur les objets, ce qui permet de traduire le fait que chacun a des buts particuliers qui entraînent des comportements particuliers pour les objets. Le système est capable d'intégrer des connaissances sur l'opérateur, mais qui sont étroitement liées aux objets de l'application : l'opérateur ne peut pas décrire toutes les connaissances qui le concernent. Ainsi, on ressent le besoin de décrire un modèle dédié à l'opérateur.

### 2.2.Les anticipations des actes de l'opérateur.

Pour garantir un dialogue de haut niveau avec l'opérateur, les systèmes utilisent des mécanismes pour anticiper ses intentions et l'aider ainsi dans ses interactions. Si l'on s'intéresse aux anticipations, on constate que réaliser des anticipations pertinentes demande d'introduire des connaissances sur l'opérateur puisque l'anticipation doit :

- être suffisamment complète. Il ne s'agit pas de déduire des bribes d'informations, mais de proposer des informations utiles à l'opérateur. On note le besoin de connaître ses buts ;
- être compréhensible. L'opérateur ne doit pas perdre son temps à essayer de comprendre ce que le système a anticipé sinon la détection n'a pas d'intérêt puisqu'elle est censée réduire la tâche de l'opérateur. Il faut connaître le langage de l'opérateur ou plutôt son niveau de compétence ;
- convenir à l'opérateur. Le système propose parmi un ensemble de solutions possibles celle que l'opérateur aurait probablement choisie. On retrouve la notion de buts à atteindre. De plus, il s'agit de connaître les stratégies de choix de l'opérateur en fonction du contexte ;

- faire intervenir le moins possible l'opérateur, l'intérêt de l'anticipation étant le gain de temps apporté. Pour cela, il faut connaître un maximum d'informations sur l'opérateur.

Bien que les objets manipulés dans l'application soient utilisés pour étudier les anticipations, on constate que, pour répondre aux critères énumérés précédemment, il faut disposer également d'informations sur l'opérateur. L'implication de l'opérateur dépasse largement le niveau de l'utilisation pure et simple du système. Nous pensons donc modéliser l'opérateur pour regrouper toutes les connaissances le concernant et qui peuvent servir lors des anticipations : le système se réfère à ce modèle pour obtenir des informations exclusives à l'opérateur.

### 2.3.L'évolution de l'opérateur.

L'opérateur ne peut en aucun cas être considéré comme une entité figée. De par sa nature, l'homme est capable d'apprendre et de modifier son comportement selon les circonstances. Il n'est pas normal de lui proposer une application qui ne tienne pas compte de ce caractère évolutif.

L'exemple le plus marquant concerne l'opérateur inexpérimenté dans l'utilisation d'une application. Un dialogue adaptatif doit aider cet opérateur dans ses interactions. Les anticipations fournies par le système apportent un maximum d'informations à l'opérateur afin qu'il comprenne le fonctionnement du système. Peu à peu, l'opérateur va se familiariser avec l'application s'il désire vraiment l'utiliser. Il en résulte que certaines aides apportées par le système n'ont plus de réel intérêt, au contraire elles peuvent même provoquer l'effet inverse et aller jusqu'à importuner l'opérateur, voire le ralentir dans ses interactions. Ainsi, les aides deviennent plus concises et le système peut même modifier les fonctionnalités qu'il propose : par exemple, mettre à la disposition de l'opérateur les actions plus complexes uniquement lorsqu'il est suffisamment expérimenté pour les réaliser.

Ainsi, quand l'opérateur utilise une application, il acquiert de l'expérience et le système peut évoluer. Pour y parvenir, il s'agit pour le système de réviser les connaissances qu'il possède sur l'opérateur. Le système apprend à connaître l'opérateur, ce qui nous fait penser aux techniques d'apprentissage (cf. Chapitre 1 *3.2.1. L'apprentissage*). Or, ces techniques se basent sur des connaissances qui, dans le cas présent, concernent directement l'opérateur : on a besoin à tout moment de connaître l'opérateur, d'où l'intérêt de le modéliser.

### 2.4.Les mondes virtuels.

L'apparition de la RV permet de confirmer l'importance prise par l'opérateur. Une des raisons provient sans doute de la définition elle-même de la RV qui tente de faire oublier à l'opérateur qu'il se trouve devant un écran. Pour y parvenir, l'opérateur doit être intégré à l'application.

Les mondes virtuels créés renferment des nombreuses possibilités en matière de dialogue, ce qui permet d'accroître l'implication de l'opérateur. Tout d'abord, nous évoquons les environnements multi-utilisateurs qui prennent tout leur sens dans les mondes virtuels. Nous montrons dans quelles mesures ils laissent à penser qu'un opérateur est tout aussi important qu'un objet dans l'application.

Puis, nous insistons dans la seconde partie sur d'autres possibilités offertes par la RV et qui influent sur l'opérateur (dans ce qu'il fait et dans ce qu'il est).

### *2.4.1.Les environnements multi-utilisateurs.*

Les environnements multi-utilisateurs sont liés aux environnements virtuels (cf. Chapitre 1 4.3. Les environnements multi-utilisateurs). On rencontre de plus en plus ce type d'environnements afin d'obtenir une meilleure adéquation avec le monde réel dans lequel l'activité et l'intelligence sont distribuées. En effet, si l'on observe les problèmes à traiter, on constate que [FER 95] :

- ils sont physiquement distribués ; les entités de l'environnement sont souvent indépendantes ;
- ils sont fonctionnellement distribués et hétérogènes ; on fait appel à des spécialistes pour chaque niveau de difficulté ;
- leur complexité impose une vision locale.

Chaque opérateur a des tâches précises à effectuer seul ou en collaboration. L'opérateur ne communique plus seulement avec le système, mais aussi avec d'autres opérateurs. On tend à visualiser l'opérateur directement dans l'environnement afin de mieux simuler la présence des différents opérateurs et de permettre la communication entre ces différents intervenants. Le dialogue évolue, car il ne gère plus uniquement le dialogue entre l'opérateur et le système, mais aussi celui entre opérateurs.

De plus, ces environnements sont difficiles à contrôler dans la mesure où de nombreux opérateurs peuvent intervenir. Le système doit exercer un contrôle sur cet ensemble d'opérateurs. Il s'agit de veiller à ce que les différents intervenants interagissent en harmonie les uns avec les autres. Le système doit connaître chaque type d'opérateurs. Cette connaissance devient d'autant plus indispensable que le dialogue en a besoin pour s'adapter aux différences de compétence existant entre opérateurs.

### *2.4.2.Les nouvelles possibilités.*

Les mondes virtuels offrent de nouvelles possibilités en matière d'interactions. Nous avons d'ailleurs évoqué ce point dans les chapitres 1 et 3 (cf. Chapitre 1 4.2. Les interactions 3D et Chapitre 3 2.1. Les manipulations en réalité virtuelle). Parmi les nouvelles données que le dialogue doit gérer, on distingue des données liées à l'opérateur.

On peut représenter l'opérateur grâce à l'ajout de la troisième dimension : afin qu'il ait la sensation de saisir les objets qu'il manipule, on peut visualiser par exemple ses mains "virtuelles" lorsqu'il prend les objets. Ainsi, l'opérateur est visualisé comme un objet graphique. Par conséquent, il faut connaître le mode de représentation à utiliser pour l'opérateur. De plus, chaque opérateur est différent ; il possède sa propre représentation graphique.

Étroitement liés à l'opérateur, des dialogues sur les comportements de son "corps virtuel" apparaissent. Nous pensons tout particulièrement aux différents types de déplacements. Ce sont des manipulations qui sont propres à un type particulier d'opérateurs. On va centraliser ces données dans un modèle.

### 2.5.Synthèse.

Nous avons mis en évidence l'importance croissante prise par l'opérateur dans le système en montrant ses différentes implications. Les systèmes tentent de se rapprocher de l'opérateur. Cette tendance se confirme par la volonté d'utiliser avant tout un langage métier. Nous avons pu constater que l'opérateur a une place prépondérante en CFAO. Pourtant, il n'apparaît pas clairement dans le système.

Se pose alors le délicat problème de la conceptualisation de la connaissance. Chaque opérateur a sa propre autonomie : il a certains buts à atteindre et c'est son savoir-faire qui va l'aider à y parvenir. Outre ce savoir-faire, l'opérateur agit selon un mode de pensée qui le caractérise : on peut considérer qu'un ensemble de lois régit ses comportements. À titre d'exemple, on peut citer les notions de secret qui apparaissent lorsqu'il interagit : le besoin de garder une certaine confidentialité devient indispensable pour certains opérateurs. Toutes ces données sont riches en informations sur l'opérateur pour permettre au système de se rapprocher de lui (résolution des incertitudes en matière d'anticipation ...). Pour exploiter ces données, on est amené à introduire un véritable modèle de l'opérateur. Les systèmes de RV corroborent l'idée de modéliser l'opérateur, puisque ce dernier est présent dans le monde virtuel. Il faut donc des moyens pour gérer l'opérateur virtuel.

## 3. LA MODÉLISATION.

Il est de plus en plus utile de modéliser l'opérateur. Dans le chapitre 2, nous avons proposé un modèle générique chargé de modéliser les objets de l'application. Il est apparu que les propriétés qui interviennent sur les objets en CFAO sont assimilables à des objets du point de vue de modélisation : c'est leur place importante prise dans les applications qui nous a poussés à envisager leur prise en compte dans la modélisation. L'opérateur peut à son tour être considéré comme un objet et, au même titre, faire partie du modèle générique. Nous montrons une unité dans les concepts de modélisation.

Nous nous proposons d'étudier la modélisation des opérateurs en tenant compte des concepts de modélisation établis dans les chapitres précédents. Nous voulons aboutir à un dialogue adaptatif et la prise en compte de l'opérateur dans la modélisation doit y contribuer. Nous insistons donc sur la modélisation des données pour favoriser le dialogue avec l'opérateur.

Ainsi, nous analysons tout d'abord les paramètres que l'on peut modéliser au sujet de l'opérateur et qui constituent la représentation de l'opérateur. Ensuite, nous poursuivons l'adéquation de la modélisation des objets avec celle des opérateurs en montrant comment utiliser les comportements (les producteurs, les réacteurs et les transmuteurs) pour décrire des connaissances sur l'opérateur. Enfin, nous situons la position des manipulateurs face aux opérateurs.

### 3.1.Les paramètres génériques.

Pour obtenir une gestion adaptée du dialogue, le modèle de l'opérateur va inclure des données traduisant des connaissances sur l'opérateur qui permettent de mener à bien ses actions. Ces données constituent la représentation canonique de l'opérateur. Nous considérons qu'il existe, pour chaque type

d'opérateurs, une représentation canonique déterminée par un domaine de définition *DD* (cf. figure IV.1). Ce domaine contient l'ensemble des paramètres génériques caractérisant l'opérateur. Il s'agit d'un ensemble d'attributs et de conditions portant sur un ou plusieurs attributs. Nous appelons ces données les paramètres génériques, car elles font partie du modèle générique qui est considéré comme le modèle de base de tous les modèles (cf. Chapitre 2 3.1. Les connaissances modélisées).

Les paramètres génériques des opérateurs nous préoccupent plus que ceux des objets d'application dont la présentation au chapitre 2 est restée sommaire. On ne se contente pas de considérer qu'ils existent. En effet, il s'agit de définir les informations que l'on va pouvoir intégrer au modèle en tant que paramètres génériques d'un opérateur.

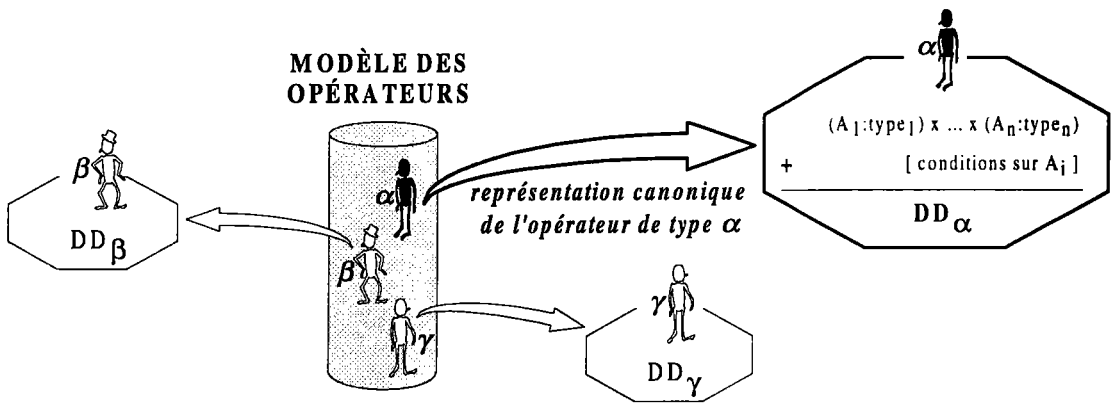



***Figure IV.1.*** *Représentation canonique de l'opérateur.*

Nous distinguons trois types d'informations. Il s'agit des paramètres concernant :

- la représentation de l'opérateur dans le monde virtuel. Il s'agit de définir son corps virtuel ;
- les capacités de l'opérateur. Il s'agit de définir les connaissances qui représentent sa compétence ;
- les influences de l'opérateur sur l'environnement. L'opérateur est une entité particulière du système qui a des connaissances difficiles à maîtriser tellement elles sont nombreuses et très variées. Il s'ensuit qu'il possède ses propres idées en matière d'interactions. Ainsi, nous déterminons également comment prendre en compte de tels critères.

Nous détaillons ces trois types d'informations dans les parties suivantes.

### *3.1.1.Le corps virtuel.*

L'opérateur est présent dans le monde virtuel : il se déplace, manipule des objets et communique avec des entités animées (opérateurs ou objets). Connaître sa position dans l'espace n'est plus suffisant, car on a besoin de le visualiser. Nous avons évoqué ce besoin précédemment (cf. *2.4.2. Les nouvelles possibilités*). Ainsi, on dédie une partie des paramètres génériques de l'opérateur à la représentation de son corps virtuel. On note $CV_\alpha$ l'unique sous-domaine inclus dans le domaine de définition $DD_\alpha$ d'un opérateur $\alpha$ et qui décrit le corps virtuel de l'opérateur : $\forall DD_\alpha \exists! CV_\alpha \mid CV_\alpha \subset DD_\alpha$.

On éprouve plus de facilités à représenter un objet de manière réaliste qu'un opérateur, car il possède, de manière générale, une représentation géométrique bien déterminée. Cette représentation n'est pas

toujours sous une forme très simple à visualiser, mais l'avantage est qu'il existe au moins une représentation mathématique adaptée. La représentation de l'opérateur est plus délicate à spécifier. Le problème consiste à définir s'il est utile de représenter dans le monde virtuel une copie conforme à l'opérateur i.e. de définir un modèle exact de l'opérateur. On nomme modèle exact, un modèle dans lequel *CV* contient tous les paramètres caractéristiques de l'opérateur de sorte qu'on ait suffisamment d'informations pour visualiser son image réelle en virtuelle.

Disposer d'un modèle exact de l'opérateur semble une bonne hypothèse pour favoriser le dialogue. L'opérateur n'a pas à se familiariser avec une nouvelle apparence : il se reconnaît dans ses interactions et reconnaît les autres opérateurs. À l'heure actuelle, on est capable de réaliser une modélisation exacte. Il existe une technique qui, à partir de découpes d'un corps humain congelé, construit une représentation exacte [USN 96]. On ne peut pas appliquer cette technique étant donné l'issue fatale de l'opérateur à modéliser. Toutefois, les corps déjà modélisés peuvent servir de base pour construire des opérateurs types.

En fait, il faut avoir conscience de l'intérêt donné à l'opérateur. Dès lors qu'il entre dans le monde virtuel, il joue le rôle d'un habitant de ce monde. Il n'est pas indispensable de donner à l'opérateur virtuel un corps identique à son correspondant réel. Cette constatation permet d'envisager plusieurs représentations pour l'opérateur selon les circonstances et de tirer profit des possibilités du monde virtuel. On définit pour l'opérateur un corps virtuel avant tout fonctionnel. Les données inutiles à l'application n'ont pas d'intérêt à être modélisée : le but est de soulager le système d'une gestion inutile et de simplifier l'opérateur dans sa compréhension du monde virtuel. Quant au risque pour les opérateurs d'avoir des difficultés à s'adapter à des apparences virtuelles différentes de l'apparence réelle, nous pensons qu'il est quasi inexistant dans la mesure où elles facilitent la compréhension du monde virtuel et les interactions des opérateurs.

En définitive, le modèle d'un opérateur $\alpha$ contient un domaine de définition $CV_\alpha$ à partir duquel on détermine son corps virtuel qui peut éventuellement correspondre à une représentation graphique exacte. On n'impose pas le mode de représentation de l'opérateur (i.e. exact ou non), puisqu'il est étroitement lié aux types de manipulations réservées à l'opérateur.

### *3.1.2.Les capacités.*

L'opérateur ne se distingue pas uniquement par son corps virtuel, mais il se caractérise également par ses buts à atteindre et par sa compétence à interagir dans l'environnement virtuel. Nous avons d'ailleurs noté auparavant leur importance dans la gestion d'un dialogue adaptatif (cf. 2.2. Les anticipations des actes de l'opérateur). On définit un unique sous-domaine $CA_\alpha$ du domaine de définition $DD_\alpha$ d'un opérateur afin d'y représenter les paramètres génériques sur les capacités de l'opérateur : $\forall DD_\alpha \; \exists! CA_\alpha \mid CA_\alpha \subset DD_\alpha$.

Pour définir les buts d'un opérateur, on conserve dans $CA_\alpha$ les thèmes qui le préoccupent (cf. figure IV.2). Ces thèmes sont ceux décrits dans le chapitre précédent (cf. Chapitre 3 *4.4.2. Les*

*thèmes et les compatibilités*). Ils constituent des attributs du domaine de définition de l'opérateur. On définit des conditions sur les thèmes, au même titre que les attributs plus communs tels que les données géométriques (position ...). Grâce aux conditions, on définit des priorités sur les thèmes de l'opérateur : par exemple, définition du thème principal à considérer par défaut au début des dialogues. Ces paramètres génériques sont très utiles pour le gestionnaire de thèmes pour gérer la cohérence dans les interventions de l'opérateur et réaliser des anticipations dans le dialogue (cf. Chapitre 3 4.4. Le maintien de la cohérence du dialogue).

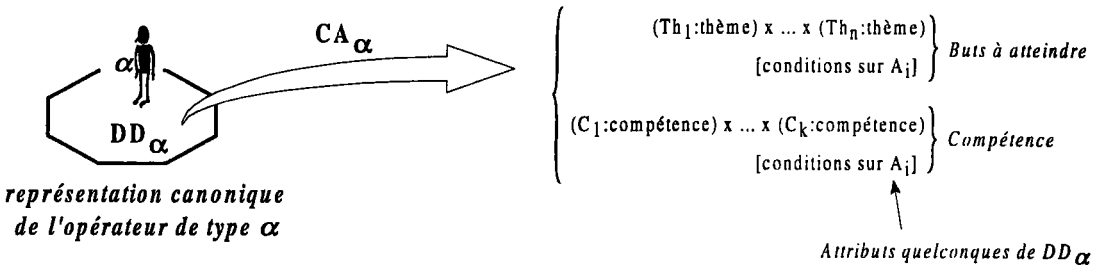



***Figure IV.2.*** *Les paramètres génériques sur les capacités d'un opérateur.*

Soient $Op\alpha$ et $Op\beta$ deux types d'opérateurs. Supposons qu'ils possèdent le même thème $T$, par exemple un thème sur la modification des objets de type $O$. Chaque opérateur peut donner sa propre définition du thème $T$ (cf. figure IV.3). Cette définition est un attribut du sous-domaine $CA$ de chaque opérateur.

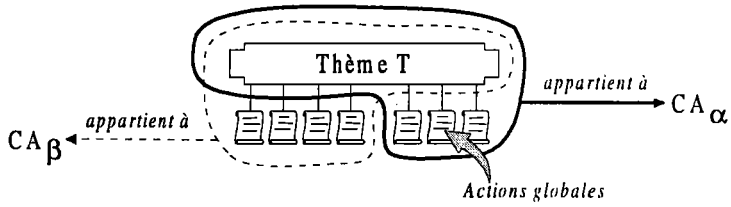



***Figure IV.3.*** *Thème commun à plusieurs opérateurs.*

L'intérêt de conserver un thème identique alors que les actions globales associées sont différentes se trouve dans la gestion globale des opérateurs et en particulier dans la gestion de la communication entre opérateurs. En effet, on peut détecter plus facilement les cas de conflit ou de coopérations possibles entre les opérateurs quand leur thème est commun. Dans l'exemple précédent, si $Op\alpha$ et $Op\beta$ sont tous les deux dans le thème $T$ et qu'ils modifient effectivement le même objet de type $O$, des problèmes peuvent survenir, car fondamentalement l'objet est modifié, que ce soit d'une façon ou d'une autre.

Le sous-domaine $CA_\alpha$ contient également des paramètres génériques décrivant la compétence de l'opérateur de type $\alpha$ (cf. figure IV.2). On détermine la compétence d'un opérateur sous deux aspects :

- l'opérateur dans l'utilisation de l'application. Il s'agit de décrire ses expériences pour réaliser les fonctionnalités proposées par l'application. Ces informations sont capitales pour fournir des aides adaptées à chaque opérateur dans le cas où le dialogue est mal engagé. Les conditions décrites sur les compétences de l'opérateur rendent possible la gestion d'un historique de ses interactions

permettant de mettre à jour le niveau d'expérience de l'opérateur et de faciliter les anticipations (analyse des choix que l'opérateur a fait auparavant et qui se trouvent dans l'historique) ;

- l'opérateur dans le monde virtuel. Il s'agit de décrire les compétences relatives à son travail. Ces informations facilitent les interactions dans le monde virtuel en ce qui concerne leur faisabilité. Par exemple, certaines manipulations telles que les simulations sont contraintes par les capacités de l'opérateur qui les effectue. De même, la communication entre deux opérateurs peut être favorisée si l'on connaît les compétences de chacun d'eux.

Notons que les conditions associées au sous-domaine $CA_\alpha$ d'un opérateur $\alpha$ ne portent pas exclusivement sur ses propres attributs (thèmes ou compétences), mais sur l'ensemble des attributs du domaine de définition $DD_\alpha$.

### *3.1.3.Les influences sur l'environnement.*

On définit pour l'opérateur $\alpha$ des paramètres génériques qui représentent des influences particulières de l'opérateur sur l'environnement. Ces paramètres sont définis dans un unique sous-domaine $IN_\alpha$ du domaine de définition $DD_\alpha$ : $\forall \mathrm{DD}_\alpha \; \exists! \mathrm{IN}_\alpha \mid \mathrm{IN}_\alpha \subset \mathrm{DD}_\alpha$. On distingue deux types de paramètres génériques pour le sous-domaine $IN_\alpha$ : des critères ergonomiques et des méta concepts sur le fonctionnement du système vis-à-vis de l'opérateur.

Si l'opérateur a des préférences particulières, on les considère comme des paramètres génériques de son modèle : elles constituent des critères ergonomiques. On souligne ici l'avantage de posséder un modèle de chaque opérateur, car il permet au système de mieux s'adapter à chaque opérateur.

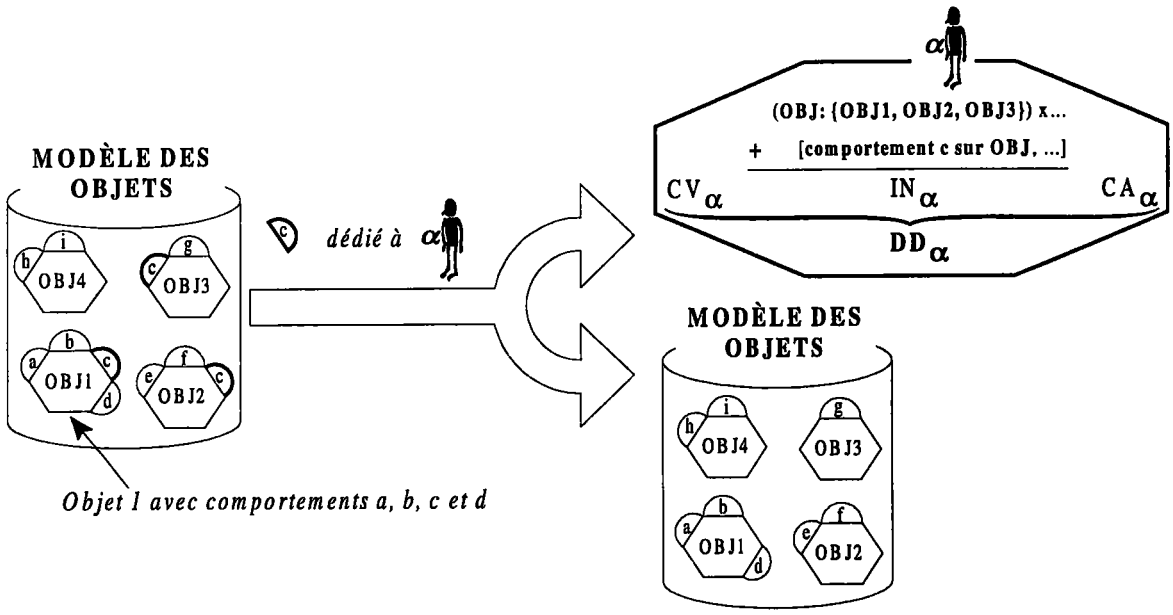



***Figure IV.4.** Méta concept comme paramètre générique du domaine des influences d'un opérateur.*

Les méta concepts répondent au besoin de regrouper certains objets d'application en famille d'objets dans les cas où les objets ont des comportements similaires et qui sont liés à un type d'opérateurs. La figure IV.4 en donne un exemple. On suppose que le modèle des objets d'application contient la

description des objets *OBJ1*, *OBJ2*, *OBJ3* et *OBJ4*. Chacun est défini en fonction de sa représentation canonique et d'un ensemble de comportements (producteurs, réacteurs, transmuteurs) : par exemple, l'objet *OBJ1* a les comportements *a*, *b*, *c* et *d*. Il se trouve que le comportement *c* s'applique aussi bien aux objets *OBJ1*, que *OBJ2* ou encore *OBJ3* et qu'il est utile aux opérateurs de type $\alpha$. En fait, le comportement *c* est défini sur une famille d'objets constituée des objets *OBJ1*, *OBJ2* et *OBJ3*. Pour décrire cette famille dans le modèle, on la considère comme un attribut *OBJ* de la représentation des opérateurs $\alpha$. Cet attribut est conditionné par le fait qu'il possède le comportement *c*.

En d'autres termes, cela signifie que lorsque l'opérateur de type $\alpha$ manipule l'application, les objets de la famille d'objets *OBJ* possèdent le comportement *c*. L'attribut *OBJ* et sa condition symbolisent une influence de l'opérateur de type $\alpha$ sur l'environnement ; ils appartiennent au sous-domaine $IN_\alpha$. En effet, l'opérateur de type $\alpha$ impose, par sa présence, des critères particuliers sur l'environnement. On ne restreint toutefois pas l'application du comportement *c* à la famille *OBJ* uniquement dans le cas des opérateurs de type $\alpha$ : on conçoit tout à fait de définir ces paramètres génériques pour d'autres types d'opérateurs possédant également ce type d'influence.

### 3.2.La description par les comportements.

On autorise la description des comportements sur les opérateurs. Ainsi, un opérateur peut posséder des producteurs, des réacteurs intrinsèques et extrinsèques, et des transmuteurs. Nous avons montré dans le chapitre 2 les grandes possibilités offertes par la description des objets d'application à l'aide de leurs comportements en ce qui concerne le dialogue et, plus généralement, la capture de connaissances. Plusieurs raisons nous ont poussés à définir des comportements sur les opérateurs modélisés.

Nous sommes partis de la constatation que l'opérateur est une entité à part entière du système, car il y est maintenant modélisé au même titre qu'un objet. Devant cette prise en compte de l'opérateur, on s'est interrogé sur le dialogue qui lui est associé et sa spécification.

Hormis les dialogues dépendant des objets d'application, des dialogues propres à l'opérateur vont s'engager : par exemple, les dialogues pour gérer les déplacements de l'opérateur ou encore la communication entre opérateurs. De plus, des données propres à certains types d'opérateurs risquent d'influer sur le dialogue portant sur des objets d'application : par exemple, des phénomènes particuliers à un type d'opérateur à déclencher pendant l'interaction sur un objet.

Ainsi, l'opérateur possède des informations utiles pour définir le dialogue d'une application. La construction du dialogue par la méthode duale décrite au chapitre 2 n'est pas suffisante. Le concepteur non expérimenté dans le modèle de dialogue ne peut transmettre toutes ses connaissances sur l'opérateur en décrivant le modèle générique, car ce dernier ne modélise pas l'opérateur. Seul le concepteur expérimenté peut inclure quelques connaissances sur l'opérateur avec le formalisme graphique. Toutefois, on ne dispose pas de moyen de stocker ces informations. La déduction complète du dialogue est donc impossible. Nous avons d'ailleurs précisé au chapitre 2 que tous les dialogues ne peuvent être déduits et en particulier ceux concernant exclusivement les opérateurs.

En définitive, comme on est capable de déduire le dialogue à partir du concept des comportements, nous avons inclus dans la modélisation des opérateurs la définition des comportements. Ainsi, la déduction du dialogue s'effectue par analyse des comportements non seulement sur les objets (objet d'application et propriétés), mais aussi sur les opérateurs. Une plus grande quantité d'informations est fournie pour obtenir un dialogue plus complet : le modèle générique est composé d'objets d'application, de propriétés et d'opérateurs qui sont tous les trois décrits par une représentation canonique et des comportements.

Dans les trois parties suivantes, nous définissons ce que représente chaque type de comportement pour un opérateur. Nous nous attachons à préserver le concept de base des comportements (cf. Chapitre 2 3.2. Les comportements).

*3.2.1.Les producteurs.*

La création d'un opérateur est un comportement qui a toujours été implicite dans une application puisque l'opérateur n'est jamais apparu comme un objet modélisé. Désormais, on s'intéresse à la manière dont l'opérateur réel peut manifester sa présence dans le mode virtuel.

Les producteurs sont chargés de décrire les différents modes de création des opérateurs. La définition du producteur est similaire à celle pour les objets (cf. figure IV.5) :

- il contient la définition de ses attributs et conditions propres à son mode de création. Il décrit donc sa propre représentation de l'opérateur ;
- il se base sur des informations provenant de réacteurs ou d'opérateurs ;
- il est capable de transmettre sa représentation à la représentation canonique de l'opérateur concerné. Une passerelle est nécessaire pour effectuer la traduction entre les deux représentations.

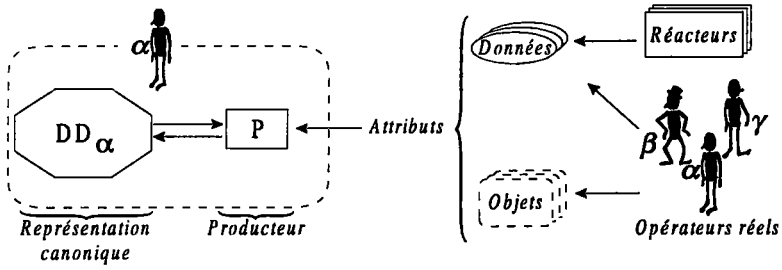



***Figure IV.5.** Producteur pour un opérateur α modélisé.*

Bien que la définition du producteur soit identique pour les objets et les opérateurs, de nouvelles caractéristiques sont mises en évidence, car nous considérons des objets animés et des environnements multi-utilisateurs. Les attributs définis par le producteur sont éventuellement fournis par différents opérateurs. Ainsi, un opérateur virtuel peut être défini par l'opérateur réel correspondant ou par d'autres opérateurs, ce qui n'est toutefois pas une solution très courante.

Nous avons montré la complexité que peuvent prendre les paramètres génériques définissant la représentation canonique d'un opérateur (cf. 3.1. Les paramètres génériques). Le producteur renferme donc des informations de haut niveau pour parvenir à décrire un opérateur. Il représente une technique

de description des paramètres génériques sous une forme accessible à tout type d'utilisateurs chargés de spécifier les modes de création. En outre, le producteur peut contenir la description complète d'attributs de manière figée i.e. pendant le dialogue déduit du producteur, ces attributs sont connus et ne proviennent donc ni de réacteurs ni d'opérateurs.

Étant donné les attributs que le producteur doit décrire pour compléter la représentation d'un opérateur, la déduction du dialogue ne se contente pas de fournir uniquement le dialogue de création associé au producteur, mais elle complète les dialogues sur d'autres objets modélisés. Le producteur inclut la description des attributs destinés au sous-domaine des influences *IN* du domaine de définition *DD* d'un opérateur (cf. *3.1.3. Les influences sur l'environnement*). Ces attributs peuvent révéler des dialogues particuliers sur l'environnement. Si l'on reprend l'exemple de la figure IV.4, on peut décrire un producteur qui permet de définir la famille *OBJ* d'objets qui contient le comportement *c*. Lors de la déduction du dialogue, le système déduit, pour chaque objet de la famille *OBJ*, le dialogue associé au comportement *c* : le producteur d'un opérateur devient l'instigateur de dialogues sur d'autres objets.

### *3.2.2.Les transmuteurs.*

Un opérateur possède des transmuteurs, ce qui peut a priori paraître étrange. En effet, ce comportement décrit l'évolution d'un objet de type *t* vers un objet de type *t'*. On autorise donc un opérateur à changer de type. Les transmuteurs gèrent l'évolution (cf. figure IV.6) :

- vers un autre opérateur comme le transmuteur $T_1$. Définir une telle évolution permet de traiter les cas où l'opérateur change de compétences. L'opérateur contient dans son sous-domaine des capacités *CA* des informations très précises sur sa compétence. Le transmuteur détermine à quel moment la compétence de l'opérateur change au point de modifier complètement l'opérateur ;
- vers un objet d'application animé ou non comme le transmuteur $T_2$. Ce type de transmuteur procure des fonctionnalités intéressantes. Par exemple, on peut imaginer qu'un opérateur virtuel se transforme en un robot virtuel, car les interactions qu'il doit réaliser sont répétitives. L'opérateur interagit et simultanément le système enregistre ses actions jusqu'à un certain moment décrit par le transmuteur. Ensuite, l'opérateur virtuel se transforme en robot, car le transmuteur a reconnu que les actions effectuées correspondent à celle du robot. Le transmuteur décrit également une transformation vers un objet non animé. Par exemple, un opérateur peut se transformer en un objet non animé lorsqu'il a atteint son but ou encore lorsque d'autres opérateurs ont agi sur lui.

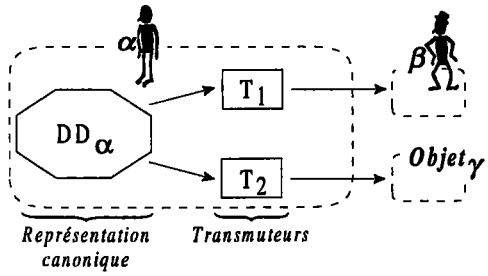



***Figure IV.6.*** *Transmuteurs pour un opérateur α.*

Le transmuteur appliqué à l'opérateur apporte ainsi de nouveaux dialogues. L'exemple du robot montre qu'un apprentissage peut s'effectuer pour automatiser les interactions grâce à l'opérateur qui montre ce qu'il faut faire et grâce au transmuteur qui reconnaît le type de robot. À ce niveau, on ne se contente plus de décrire l'opérateur virtuel comme l'opérateur réel ; on va au delà de ses capacités réelles afin de profiter au maximum des possibilités informatiques présentes.

*3.2.3.Les réacteurs.*

On définit des réacteurs pour un opérateur afin de l'interpréter sous une forme compréhensible pour l'environnement. La figure IV.7 montre l'intégration de réacteurs à la définition d'un opérateur $\alpha$.

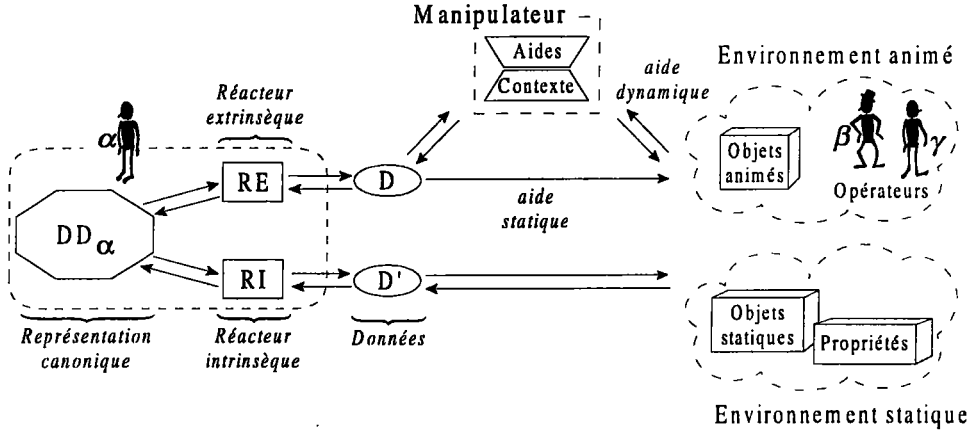



***Figure IV.7.** Réacteurs pour un opérateur $\alpha$.*

Comme dans le cas des objets, on distingue les réacteurs extrinsèques et les réacteurs intrinsèques. Chacun traduit l'opérateur en une donnée, ensemble d'attributs. La différence entre ces deux types s'effectue au niveau des récepteurs de la donnée : pour le réacteur extrinsèque, c'est une entité faisant partie de l'environnement animé i.e. un opérateur ou un objet animé (robot) alors que pour le réacteur intrinsèque c'est une entité non animée i.e. un objet d'application statique (cube ...) ou une propriété.

Le réacteur intrinsèque fournit des informations utiles pour gérer des propriétés. Ainsi, on arrive à décrire des propriétés dans lesquelles l'opérateur intervient. De plus, si on a besoin d'informations sur un opérateur $\alpha$ pour créer un objet non animé $O$, on définit pour l'opérateur $\alpha$ un réacteur intrinsèque chargé d'obtenir la donnée correspondant aux informations nécessaires à la création de $O$. Pour la création d'un objet animé ou d'un opérateur, c'est un réacteur extrinsèque qui définit la donnée nécessaire.

Le réacteur extrinsèque procure des aides statiques si la donnée est transmise telle quelle à l'environnement animé, ou dynamiques si la donnée est traitée par un manipulateur. Parmi les aides statiques que le réacteur extrinsèque peut induire, on trouve la visualisation du corps virtuel de l'opérateur. Ainsi, en définissant plusieurs réacteurs extrinsèques, on peut donner des apparences différentes pour un opérateur en fonction du contexte ; ces réacteurs se basent sur la représentation de l'opérateur, modélisée par le sous-domaine corps virtuel *CV* du domaine de définition *DD* de

l'opérateur. Des réacteurs extrinsèques d'un opérateur peuvent être associés à des manipulateurs. Nous préférons consacrer la partie suivante à ce point, car il mérite une étude plus approfondie.

## 3.3.Les manipulateurs.

Nous précisons dans cette partie la position des opérateurs face aux manipulateurs. Tout d'abord, nous analysons les classes de manipulateurs concernées par les opérateurs. Ensuite, nous montrons les possibilités apportées par le couple opérateur/manipulateur en matière d'interactions.

### *3.3.1.Les classes de manipulateurs utiles pour l'opérateur.*

Les manipulateurs associés aux opérateurs par l'intermédiaire des réacteurs extrinsèques ne sont pas exclusifs à une classe particulière de manipulateurs (cf. Chapitre 3 3.3. Une classification) : la donnée transmise par le réacteur extrinsèque peut tout aussi bien révéler un savoir-faire de l'opérateur (connaissance de surface) ou une caractéristique plus structurée (connaissance profonde). Quant à la notion d'activité, il n'y a aucune restriction particulière : le manipulateur associé à un opérateur peut être actif exclusif, partagé ou passif exclusif.

Bien que toute classe de manipulateurs s'accorde à un opérateur, on distingue une classe privilégiée : le manipulateur actif guidé par une connaissance de surface i.e. la classe *SURF-ACT*. La calculatrice d'expressions grapho-numériques en est un exemple. Elle reflète la capacité, pour l'opérateur auquel elle est associée, de définir des contraintes sur l'environnement. En d'autres termes, on modélise les outils dont il a besoin pour réaliser ses interactions dans le monde virtuel. On évite ainsi de proposer à un opérateur des outils de manipulations qui ne lui correspondent pas.

L'opérateur utilise également des manipulateurs appartenant à d'autres entités (objets ou opérateurs), puisqu'il peut faire partie du domaine d'utilisation *DUt* d'un quelconque manipulateur : par exemple, le domaine d'utilisation de la manette de modification de la hauteur d'un cylindre. Modéliser l'opérateur prend toute son importance ici. Le système parvient plus aisément à retrouver et reconnaître les opérateurs concernés par un manipulateur (i.e. appartenant au domaine d'utilisation *DUt* ou domaine d'application *DApp*) puisque ces opérateurs sont modélisés.

### *3.3.2.Vers une autre dimension de l'interaction.*

Les manipulateurs offrent des possibilités d'interactions inespérées, car ils permettent d'annihiler les restrictions apportées par l'opérateur réel et d'obtenir un opérateur virtuel plus performant. Pour mettre en évidence cet aspect, nous présentons un exemple de manipulateur associé à des opérateurs de type $\alpha$ pour les rendre plus performant : le manipulateur clone. Le clone permet à un opérateur de se dédoubler virtuellement afin de déléguer des travaux à son double pendant qu'il poursuit ses interactions. Non seulement l'opérateur gagne du temps, mais il peut réaliser des tâches en parallèle.

Le clone est un manipulateur, car il appartient à la classe *PROF-PART* (guidé par des connaissances profondes, utilisation partagée) de manipulateurs. Le clone existe grâce à un réacteur extrinsèque *RE* associé aux opérateurs $\alpha$. *RE* transmet sous forme d'une donnée tous les attributs utiles à la réalisation

du clone : la donnée représente essentiellement une connaissance profonde, d'où l'appartenance du manipulateur à la classe *PROF*. Le clone est manipulé par l'opérateur qui le possède, mais il peut aussi l'être par d'autres opérateurs (cas où, par exemple, un opérateur de type $\beta$ communique avec le clone) donc il est partagé. Ainsi, le clone appartient bien à la classe *PROF-PART*. Nous analysons maintenant les différents composants qui permettent la définition du clone :

- son domaine d'utilisation *DUt* est constitué de l'opérateur $\alpha$ auquel le clone est associé et de tous les opérateurs, voire les objets animés, qui sont capables d'utiliser le clone pour réaliser leurs propres interactions. Ces entités animées (opérateurs ou objets) ont la possibilité de communiquer avec l'opérateur $\alpha$ ou d'obtenir des informations à son sujet via le clone ;
- son domaine d'application *DApp* dépend des fonctionnalités que le clone est capable d'effectuer. Si l'on considère que le clone possède les mêmes fonctionnalités que l'opérateur $\alpha$, *DApp* représente l'union de tous les domaines d'application des manipulateurs de l'opérateur $\alpha$. On peut envisager des fonctionnalités propres au clone et, par conséquent, de nouveaux objets à définir dans *DApp* ;
- ses apparences *Ap* reposent sur le sous-domaine corps virtuel *CV* de la représentation de l'opérateur $\alpha$. On peut éventuellement décrire dans *Ap* des données propres au clone afin de le distinguer dans le monde virtuel, mais ce n'est pas une obligation. En effet, il n'est peut-être pas nécessaire de distinguer le clone de son opérateur $\alpha$ dans l'environnement animé ;
- ses mécanismes *Mé* reprennent des mécanismes propres et des mécanismes issus de l'opérateur $\alpha$ puisque le clone peut effectuer des interactions à la place de l'opérateur. Les mécanismes propres au clone sont très complexes, car ils gèrent les différents problèmes qui se produisent lorsque le clone interagit avec l'environnement. La difficulté provient du fait que le clone n'est pas autonome, il dépend de l'opérateur. Il peut se trouver dans des cas de conflits avec les objets qu'il tente de manipuler ou avec des opérateurs ou d'autres clones. Par exemple, lorsqu'il est utilisé pour effectuer les mêmes actions que l'opérateur $\alpha$ à une position différente dans l'environnement, il risque de se retrouver avec un objet qu'il ne peut manipuler car il est contraint alors que l'opérateur dispose d'un objet non contraint. Il a besoin des connaissances de l'opérateur pour résoudre ce cas.

Ainsi, le clone est un manipulateur très puissant de par les possibilités d'interactions qu'il offre à l'opérateur auquel il est associé. Toutefois, si du côté manipulations on gagne en simplicité, du côté modélisation les mécanismes à mettre en œuvre sont de plus en plus complexes. Grâce à la modélisation des opérateurs, on dispose de plus d'informations, ce qui tend toutefois à simplifier la gestion des mécanismes.

### 3.4.Synthèse.

Nous avons montré que l'opérateur peut être modélisé utilement en décrivant tout d'abord sa représentation, puis en ajoutant des comportements, ce qui favorise la construction du dialogue. Ainsi, l'opérateur est perçu par un couple données/comportements au même titre qu'un objet d'application ou une propriété. L'assimilation de l'opérateur avec un objet n'est réalisable que si l'on considère qu'il

existe des objets animés, auquel cas on définit l'opérateur comme tel, les notions de réactivité et de buts à atteindre caractérisant un opérateur ne pouvant être mises à l'écart.

De plus, la modélisation de l'opérateur améliore nettement les dialogues non seulement grâce aux données pertinentes que l'on modélise (buts, compétences ...), mais aussi grâce aux manipulateurs qu'on arrive à associer à l'opérateur. On démontre que la RV ne se restreint pas à la traduction pure et simple du monde réel mais qu'elle exploite toutes les capacités offertes par la modélisation pour proposer des dialogues adaptatifs.

## 4. LES OPÉRATEURS ET LES AGENTS.

L'opérateur modélisé est perçu comme un objet animé du système qui interagit sur l'environnement et éventuellement communique avec d'autres objets animés. La modélisation de l'opérateur favorise des environnements multi-utilisateurs et amène le problème de la communication entre les différents opérateurs.

Cette représentation du système semble s'apparenter à celle que l'on retrouve dans les SMA (Systèmes Multi-Agents) que nous avons évoqués au chapitre 1 (cf. Chapitre 1 3.3. Les systèmes multi-agents). On s'interroge sur le lien éventuel entre les opérateurs et les agents que l'on rencontre dans ces systèmes afin de résoudre le problème de la communication.

Tout d'abord, nous comparons les deux approches afin de montrer que la correspondance qui semble exister entre les opérateurs et les agents est exploitable. Ensuite, nous présentons comment gérer la communication entre opérateurs en nous basant sur les techniques des SMA.

### 4.1.La comparaison.

Nous concentrons notre attention sur les agents dans les SMA et les opérateurs dans les systèmes de RV. Il s'agit de mettre en évidence les similitudes entre ces deux notions afin de bénéficier de l'expérience des SMA pour proposer un modèle des opérateurs plus complet. Pour établir la frontière entre ces deux types d'entités, nous nous basons sur la présentation des SMA faites dans le chapitre 1 (cf. Chapitre 1 3.3. Les systèmes multi-agents). Nous sommes arrivés à la conclusion que les liens entre les agents et les opérateurs sont indéniables bien qu'il existe quelques différences notables ainsi que le montre le tableau récapitulatif :

| CRITÈRES DE COMPARAISON | AGENTS | OPÉRATEURS |
|---|---|---|
| ①**Caractéristique principale** | interagir | idem |
| ②**Mécanismes essentiels** | communication entre agents,<br><br>puis, manipulations | manipulations,<br><br>puis, communication entre opérateurs |

| | | |
|---|---|---|
| **③Notion d'entités à part entière** | distinction physique et logique,<br><br>compétences différentes | représentation canonique définie par : $DD = CV \cup CA \cup IN$<br>*où DD : Domaine de définition*<br>*CV : Corps Virtuel*<br>*CA : Capacités*<br>*IN : Influences* |
| **④Capacités** | raisonner et agir,<br><br>planifier, catégorie déterminée d'actions,<br><br>satisfaire des buts | raisonner et agir,<br><br>suivre leur propre logique, fils d'activités multiples,<br><br>satisfaire des buts (thèmes) |
| **⑤Position face à l'environnement** | perception limitée,<br><br>représentation partielle stockée | *en théorie*, perception limitée (restriction physique et fonctionnelle),<br>*en pratique*, dépend des outils mis à leur disposition<br><br>*en théorie*, représentation partielle stockée<br>*en pratique*, dépend des outils mis à leur disposition |

Les agents sont des entités qui sont utilisées pour leur capacité à interagir (critère ① du tableau). Ils sont plus importants pour ce qu'ils font que pour ce qu'ils sont. Les opérateurs sont également perçus de cette manière. Si l'opérateur intervient dans une application, c'est dans un but précis : pour y parvenir, il va réaliser une ou plusieurs interactions. Ce sont ses actes (interactions) qui vont être importants puisqu'ils vont permettre d'agir dans l'environnement (création ...) et de guider le dialogue (génération d'aides ...). Les opérateurs sont présents pour interagir avec l'environnement. On donne donc de l'importance aux interactions des opérateurs comme pour les agents.

L'une des plus grandes priorités des agents concerne la communication entre agents : c'est l'interaction entre agents qui le caractérise vraiment (critère ②). L'opérateur est dirigé plutôt par les interactions qu'il effectue sur les objets d'application, car on modélise avant tout ses comportements et ses outils de manipulations (manipulateurs). La communication entre opérateurs revêt un caractère plus secondaire ; elle est peu utilisée dans les systèmes actuels, car les environnements multi-utilisateurs ne sont pas encore assez développés, mais elle peut devenir fondamentale par la suite. Elle facilite certaines interactions de l'opérateur qui peut par exemple demander de l'aide à d'autres opérateurs.

Si l'on s'intéresse à la définition même d'un agent (cf. Chapitre 1 3.3. Les systèmes multi-agents), on constate de grandes similitudes avec un opérateur (critère ③). L'agent est souvent physiquement et logiquement distinct, ce qui est également une caractéristique de l'opérateur. On distingue d'ailleurs dans le domaine de définition *DD* d'un opérateur des sous-domaines *CV* (corps virtuel), *CA* (capacités) et *IN* (influences sur l'environnement) : on met ainsi en valeur cette distinction physique et logique.

Les capacités à raisonner, à planifier et à agir sont également des éléments caractéristiques d'un agent (critère ④). De par sa nature, l'opérateur dispose à son gré de telles capacités. L'opérateur et l'agent exploitent ces capacités pour satisfaire un but. L'agent a recours à une catégorie déterminée d'actions pour y parvenir ; l'opérateur est moins maîtrisable, car il suit sa propre logique qui ne correspond pas forcément à un plan défini (notion de fils d'activités multiples). On modélise à l'aide des thèmes les buts à atteindre de chaque opérateur : on apporte un contrôle sur les fils d'activités multiples.

En outre, un agent est capable de percevoir son environnement, mais de manière limitée et n'en possède qu'une représentation partielle [FER 95] (critère ⑤). Dans la réalité, l'opérateur répond à ce critère. Il ne perçoit que de manière limitée son environnement pour deux raisons majeures :

- restriction physique due à son champ de vision limité et à la composition de l'environnement qui entraîne des zones cachées ;
- restriction fonctionnelle due à sa compétence. Il ne peut pas percevoir ce qu'il ne connaît pas.

A priori, l'opérateur est également limité dans la connaissance de son environnement. Toutefois, les comportements et les manipulateurs qu'on associe à son modèle lui procurent une connaissance plus étendue sur l'environnement. Par exemple, il peut posséder un champ de vision total (360°) en définissant un réacteur extrinsèque et un manipulateur adaptés. Par conséquent, l'opérateur tel qu'il est modélisé n'est pas aussi limité qu'un agent sur la connaissance de son environnement. Quant à sa capacité à conserver les informations qu'il perçoit sur l'environnement, elle dépend de son aptitude à mémoriser et des outils qu'il peut utiliser pour stocker ce dont il a besoin. Ainsi, selon la définition donnée pour l'opérateur (i.e. sa modélisation), on peut arriver à une connaissance sur l'environnement beaucoup moins stricte que pour l'agent.

De manière plus globale, le SMA et le système de RV dans lequel on modélise des opérateurs sont liés. Dans le chapitre 1 (cf. Chapitre 1 *3.3.1.Définition*), on peut trouver une définition d'un SMA dans laquelle on cite les différents éléments caractéristiques : un environnement *E*, un ensemble *O* d'objets, un ensemble *A* d'agents, un ensemble *R* de relations, un ensemble *Op* d'opérations et des lois *L* de l'univers. On constate qu'à chacun de ces éléments correspond un composant d'un système de RV :

- *E* est assimilable à l'environnement virtuel ;
- *O* correspond aux objets d'application constituant *E*. Dans un SMA, *O* représente des objets passifs que l'on sait localiser. Par conséquent, *O* n'inclut pas les objets d'applications animés (des robots par exemple) que nous assimilons aux opérateurs (cf. 3.4. Synthèse) ;
- *A* représente les entités actives du système. Il s'agit des opérateurs et des objets animés i.e. l'environnement animé ;

- *R* constitue les liens entre les entités de l'environnement virtuel i.e. les objets (animés ou non) ou les opérateurs. Ces liens sont très divers. Il peut s'agir d'un lien hiérarchique afin de définir des familles d'entités ou encore de contraintes afin de préciser les entités les unes par rapport aux autres. Le sous-domaine des influences *IN* de la représentation d'un opérateur définit des liens ;
- *Op* correspond aux capacités des opérateurs et objets animés pour interagir. *Op* est entièrement décrit dans le modèle par la représentation canonique et les comportements ;
- *L* décrit tous les mécanismes liés au contexte et qui permettent d'appliquer les opérations de *Op*. Ces mécanismes incluent les aides apportées aux agents pour interagir. Les comportements et les manipulateurs décrivent *L*.

### 4.2.La communication entre opérateurs.

La communication est une fonctionnalité qui prend toute son importance à partir du moment où l'on choisit de modéliser l'opérateur. Sa gestion est confiée tout naturellement à des manipulateurs. En effet, quelle que soit la gestion effectuée, elle se base sur une donnée (ensemble d'attributs) de l'opérateur qui communique ; cette donnée est transmise par un réacteur extrinsèque et entraîne le processus de communication que l'on confie donc à un manipulateur. Il n'y a pas unicité du manipulateur qui gère la communication : par exemple, le clone peut servir à communiquer au même titre qu'une zone de message interactive. La raison provient de la possibilité pour un manipulateur de combiner plusieurs fonctionnalités. On se propose de décrire globalement un processus de communication. On définit donc un manipulateur générique *M-Com* chargé de la communication d'un opérateur de type $\alpha$.

Dans les SMA, on essaie de programmer l'intelligence des agents en essayant de spécifier leur autonomie. L'exemple le plus parlant est sans doute la définition d'un SMA où les agents sont des robots qui doivent chacun accomplir une tâche en tenant compte de l'environnement. Pour spécifier leur autonomie, on doit être capable de situer et de gérer la position de chaque agent face aux autres agents. Ce qui nous préoccupe tout particulièrement dans le concept d'agent, c'est cette faculté de gérer les communications entre agents.

Étant donné les similitudes entre agents et opérateurs, on s'interroge sur les aides que l'on peut apporter à l'opérateur pour définir le manipulateur *M-Com* en utilisant des concepts issus des SMA. Ces aides apparaissent à deux moments :

- quand l'opérateur interagit ; le système peut chercher à détecter si une communication avec d'autres opérateurs est bénéfique, nécessaire ou inutile. Ce moment correspond exactement à des connaissances des Domaines du manipulateur *M-Com* formés du domaine d'utilisation *DUt* et du domaine d'application *DApp* (cf. figure IV.8) ;
- quand l'opérateur communique ; la compréhension entre les deux intervenants n'est pas simple, mais le système peut les aider. Il s'agit des Aides de *M-Com* formées des apparences *Ap* et des mécanismes *Mé* (cf. figure IV.8).

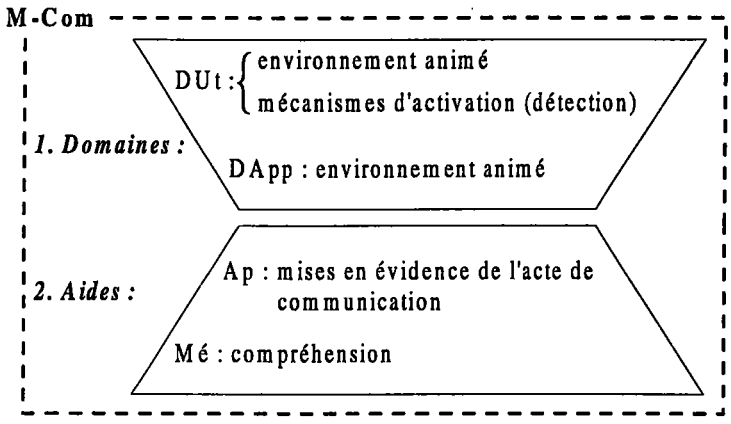



***Figure IV.8.*** *Manipulateur générique M-Com dédié à la communication.*

Nous étudions dans les deux parties suivantes les apports des SMA en ce qui concerne ces deux aides particulières ; on montre comment les techniques des SMA interviennent pour décrire les Domaines et les Aides du manipulateur *M-Com.* Tout d'abord, il s'agit de définir dans quelles conditions le dialogue peut détecter de manière naturelle des communications entre opérateurs. Ensuite, on suppose qu'une communication s'établit et on montre comment faciliter la communication entre opérateurs.

*4.2.1.La détection.*

Lorsque l'opérateur effectue des manipulations dans l'environnement virtuel, il peut éprouver certaines difficultés à les réaliser ou ne pas les réaliser de manière optimale. Le système propose des aides pour faciliter la tâche de l'opérateur. Parmi les aides qu'il peut fournir, on s'intéresse à la détection de la communication entre opérateurs i.e. à l'activation implicite du manipulateur *M-Com.* Il s'agit de décrire le mécanisme d'activation sur le domaine d'utilisation *DUt* de *M-Com* (cf. figure IV.8).

Les SMA confrontent les buts, les ressources et les compétences des agents pour détecter les éventuels conflits ou collaborations possibles entre agents donnant lieu à une anticipation de la communication (cf. Chapitre 1 *3.3.2. La communication*). Nous nous inspirons de cette technique pour définir notre propre technique de détection adaptée aux opérateurs en proposant la méthode suivante :

*Algorithme* Détection_de_la_communication ;
*Début*
    *Pendant* interactions de l'opérateur α *faire*
*{étape 1}*  B ← ChercherButsPoursuivis ;
*{étape 2}*  R-Nec ← ÉvaluerRessourcesNécessaires(B) ;
    R-Eff ← ÉvaluerRessourcesEffectives(R-Nec, Environnement) ;
*{étape 3}*  Niv-C ← ÉvaluerNiveauDeCompétence(R-Nec, R-Eff) ;
*{étape 4}*  ÉtatInteraction ← (B, R-Nec, R-Eff, C) ;
*{étape 5}*  *si* (ÉtatInteraction = Dialogue mal engagé) *alors*
        GénérerAidesClassiques(ÉtatInteraction) ;
    *fsi*

```
{étape 6}     Situation ← DéduireSituationCommunication(ÉtatInteraction) ;
{étape 7}     si (Situation = Communication recommandée) alors
                    Filtre ← FiltreCommunication(Situation) ;
                    Op ← RechercherOpérateurPourCommuniquer(Filtre) ;
                    ActiverCommunication(Op) ;
              fsi
        fpendant
Fin
```

L'étape 1 consiste à définir le ou les buts *B* que l'opérateur de type α essaie d'atteindre à l'instant où l'on effectue la détection de la communication. Grâce à la modélisation de l'opérateur et des manipulateurs, on parvient très facilement à obtenir *B*. En effet, on connaît déjà l'ensemble des buts que l'opérateur $\alpha$ peut avoir, car son modèle contient cette information dans le sous-domaine des capacités $CA_\alpha$ de son domaine de définition $DD_\alpha$ sous la forme des thèmes $T_i$. À partir de ces informations, le gestionnaire de thèmes associé aux manipulateurs peut fonctionner pour déterminer les thèmes poursuivis à un moment donné (cf. Chapitre 3 4.4. Le maintien de la cohérence du dialogue). Ainsi, on peut déterminer *B*.

L'étape 2 détermine les ressources concernant les données indispensables à l'opérateur $\alpha$ pour effectuer *B*. On distingue deux types de ressources :

- les ressources nécessaires *R-Nec*, qui représentent ce dont l'opérateur a besoin pour terminer l'action qu'il a commencée. Elles constituent les ressources théoriques. Il s'agit de décrire les entités de l'environnement que l'opérateur doit utiliser (objets et opérateurs). Les thèmes décrivent les actions nécessaires pour les accomplir, donc a fortiori les éléments indispensables pour réaliser ces actions ;
- les ressources effectives *R-Eff*, qui représentent ce qui est mis à la disposition de l'opérateur. Elles ne sont pas toujours suffisantes pour accomplir convenablement les actions de l'opérateur. On assimile ces ressources aux ressources pratiques. Connaissant *R-Nec*, il suffit de vérifier si l'environnement peut fournir les informations nécessaires à la réalisation de *B*, ceci est rendu possible grâce à la modélisation de l'ensemble des objets animés ou non (opérateurs inclus).

L'étape 3 définit le niveau de compétences *Niv-C* de l'opérateur $\alpha$ pour utiliser les ressources effectives *R-Eff* et pour créer les ressources qui vont lui manquer (i.e. *R-Nec* - *R-Eff*) : ce sont les compétences modélisées et sans cesse mises à jour qui donnent *Niv-C*. En effet, le sous-domaine $CA_\alpha$ du domaine de définition $DD_\alpha$ de l'opérateur $\alpha$ contient parmi ses attributs et conditions une description de ses compétences (cf. *3.1.2. Les capacités*). De plus, le sous-domaine corps virtuel *CV* peut apporter des compléments d'informations sur le niveau de compétence dans le cas où des paramètres propres à l'apparence physique de l'opérateur sont mis en jeu dans les ressources à obtenir.

L'étape 4 définit l'état d'interactions *ÉtatInteraction* de l'opérateur $\alpha$ à partir du quadruplet (*B*, *R-Nec*, *R-Eff*, *Niv-C*). On détermine si l'opérateur va se trouver dans un cas de dialogue mal engagé. Le

cas échéant, le système peut déclencher des aides adaptées. L'étape 5 reflète les aides classiques i.e. qui ne se préoccupent que de l'opérateur $\alpha$ alors que les étapes 6 et 7 décrivent une aide qui fait intervenir les autres opérateurs (la détection de la communication). Nous détaillons ces deux dernières étapes.

L'étape 6 consiste à définir la situation de communication *Situation* à partir du quadruplet (*B*, *R-Nec*, *R-Eff*, *Niv-C*). On détermine si l'opérateur a tout intérêt à communiquer avec d'autres opérateurs. Trois cas sont envisageables :

- communication pour bénéficier de l'aide d'un autre opérateur ;
- communication pour prévenir les risques de conflits avec d'autres opérateurs ;
- communication inutile.

En utilisant le modèle des opérateurs, on obtient des informations pertinentes sur les capacités des opérateurs si bien qu'on arrive à déterminer s'il existe des opérateurs pouvant aider ou gêner l'opérateur de type $\alpha$ à réaliser ses buts *B* étant donné *R-Nec*, *R-Eff* et *Niv-C*.

L'étape 7 concerne la mise en place de la communication dans le cas où la situation de communication détectée (*Situation*) la préconise. On définit avec *Situation* un filtre permettant d'optimiser la recherche des opérateurs présents dans l'environnement et avec lesquels l'opérateur doit communiquer. Le manipulateur de communication *M-Com* connaît les types d'opérateurs avec lesquels l'opérateur $\alpha$ peut communiquer grâce à son domaine d'application *DApp*, ce qui permet de préciser le filtre. Ensuite, on effectue, au moyen du filtre, la recherche d'un opérateur susceptible d'entrer en communication avec l'opérateur $\alpha$.

L'utilisation et le choix d'un filtre sont importants. En effet, la méthode de recherche des opérateurs utilisée influe sur la qualité du processus de détection de la communication. La première solution revient à analyser tous les opérateurs présents dans l'environnement. Pour améliorer l'analyse, il convient de définir un filtre de recherche d'opérateurs. Cette méthode pour détecter la présence d'opérateurs nous fait penser à la définition de "l'aura" de [FAH 93] que nous avons présentée dans le chapitre 1 (cf. Chapitre 1 4.3. Les environnements multi-utilisateurs) et qui définit la région dans laquelle la présence d'un opérateur peut être perçue.

On choisit de définir comme filtre une zone de communication autour de l'opérateur en difficulté (i.e. celui pour lequel le système cherche des interlocuteurs). Cette zone définit le périmètre dans lequel il est utile de rechercher un opérateur pour communiquer ; on préfère cette zone à "l'aura", car le processus de détection de la communication se base sur l'opérateur qui interagit, tout s'articule autour de lui pour détecter la présence des autres opérateurs et non l'inverse. Le problème réside dans la définition optimale de la zone. Contrairement à la méthode de "l'aura", cette zone à utiliser s'adapte à chaque situation de détection en utilisant l'information *Situation* basée sur le quadruplet (*B*, *R-Nec*, *R-Eff*, *Niv-C*).

On suppose pour l'instant que *M-Com* choisit un seul opérateur *Op* qu'il propose à l'opérateur $\alpha$ pour communiquer. Toutefois, on pourrait aller plus loin dans la recherche de communication, car, en

réalité, il peut y avoir plusieurs opérateurs susceptibles d'entrer en communication avec l'opérateur $\alpha$. Ainsi, il faudrait décrire des mécanismes supplémentaires pour gérer les cas où il est nécessaire de proposer plusieurs communications. Une fois l'opérateur *Op* déterminé, les conditions d'utilisation du manipulateur *M-Com* sont remplies, *M-Com* peut fonctionner pour gérer l'acte de communication.

### *4.2.2.La compréhension.*

On suppose maintenant qu'une communication s'établit entre deux opérateurs suite à une détection ou à une demande explicite de l'un des intervenants. En d'autres termes, le manipulateur *M-Com* est activé et grâce à une partie de ses mécanismes et de ses apparences il est capable de transmettre les messages tel un véritable canal de communication. Indépendamment des moyens matériels à mettre en œuvre, la communication peut poser des problèmes. De même que le dialogue homme/machine requiert des mécanismes adaptés pour que la communication ait un sens, le dialogue homme virtuel/homme virtuel a besoin de l'aide du système. Ce sont les aides du manipulateur *M-Com* qui vont être mises en œuvre i.e. ses apparences *Ap* et ses mécanismes *Mé* (cf. figure IV.8).

Les opérateurs qui interviennent dans le monde virtuel ont des compétences et des caractéristiques différentes les uns par rapport aux autres ; c'est l'une des raisons qui nous ont poussés à modéliser les opérateurs. Il se pose le problème de la possible incompréhension entre les opérateurs pendant qu'ils communiquent. Ce problème est courant dans tous les domaines, y compris en CFAO. Par exemple, prenons le cas d'opérateurs spécialisés dans les assemblages d'objets. Certains de ces assemblages sont réalisés par opérations booléennes. Le langage utilisé pour décrire ces opérations peut varier selon la compétence de l'opérateur. La figure IV.9. illustre ce problème en montrant deux manières différentes de percevoir les opérations booléennes en CFAO : l'une est orientée conception géométrique, l'autre concerne plutôt la fabrication.

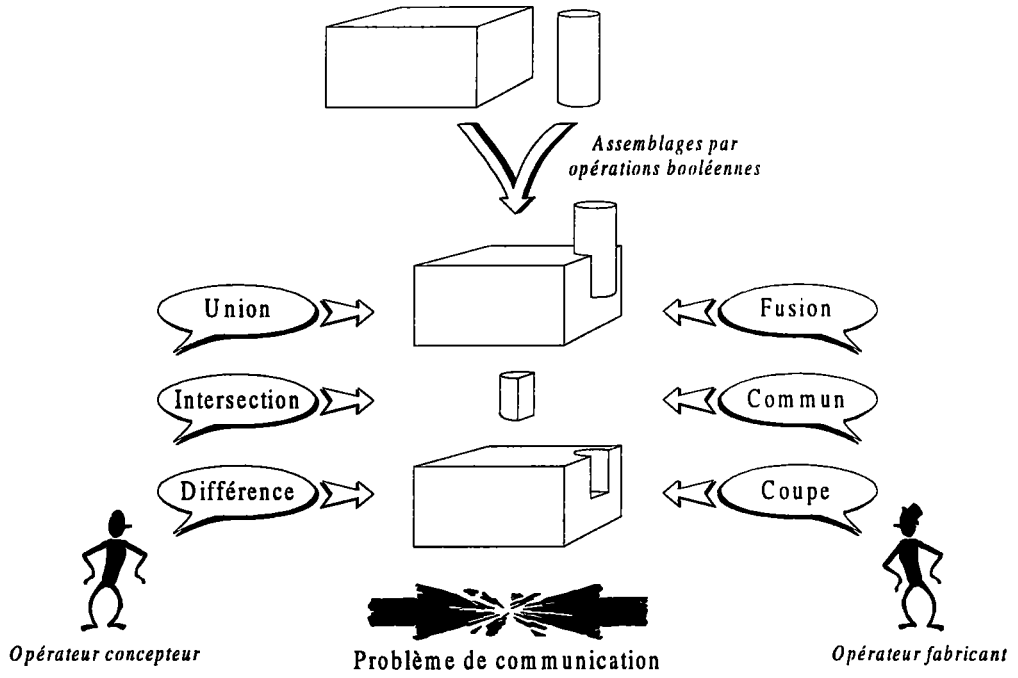



***Figure IV.9.*** *Compétences différentes entre opérateurs.*

Pour résoudre ce problème d'incompréhension, il faut définir des aides pendant le dialogue entre les opérateurs virtuels. Nous présentons deux approches que le manipulateur peut intégrer dans ses aides pour répondre à cette attente : la première est issue des techniques des SMA, la seconde est propre à la modélisation en CFAO. Ces deux approches ne s'opposent nullement, au contraire, on peut les considérer comme deux méthodes complémentaires.

Les agents communiquent entre eux en limitant les problèmes d'incompréhension grâce à un contrôle soutenu de chaque propos échangé. La méthode consiste à décomposer l'acte de communication afin d'en comprendre toutes les finalités. Nous nous basons sur la méthode de décomposition présentée au chapitre 1 (cf. Chapitre 1 *3.3.2. La communication*) et dans laquelle l'acte de communication comprend une composante locutoire, illocutoire et perlocutoire. Il s'agit par la forme locutoire de connaître le langage de communication de l'opérateur. Cette information peut être modélisée pour chaque opérateur dans le cas où le vocabulaire employé est propre à la compétence de l'opérateur : c'est le sous-domaine des capacités *CA* du domaine de définition de l'opérateur qui contient ce type d'informations et que le manipulateur *M-Com* va exploiter. La forme illocutoire est très importante, car elle permet de donner un ton au discours (interrogation, exclamation ...). L'application de cette forme peut revenir à modifier l'apparence de l'opérateur i.e. son corps virtuel : l'opérateur adopte un comportement virtuel différent. Ceci implique de la part du manipulateur *M-Com* de disposer de mécanismes et d'apparences gérant la forme illocutoire du discours de l'opérateur et dont le but est d'influer sur l'apparence virtuelle de l'opérateur qui est modélisée par le sous-domaine corps virtuel *CV* du domaine de définition de l'opérateur.

Dans les SMA, l'acte de communication contient une troisième forme : la forme perlocutoire. Elle représente l'effet du message sur le destinataire. Dans le modèle de l'opérateur, on interprète ce cas de la même manière que pour la forme illocutoire : le manipulateur *M-Com* se charge de capturer et de transmettre le message de l'opérateur. Il ne nous paraît pas intéressant de distinguer une quelconque forme perlocutoire pour l'opérateur dans la mesure où les traitements en modélisation sont identiques.

Dans le cas d'incompréhension tel que celui illustré par la figure IV.9, il existe une approche plus adaptée au domaine de la CFAO. En effet, le problème d'incompréhension entre deux opérateurs correspond parfois au fait qu'en CFAO il existe plusieurs modèles de représentation des objets : c'est le concept de multi-modélisation qui est à l'origine du problème. En effet, les opérateurs éprouvent des difficultés à se comprendre tout simplement parce qu'ils ne se basent pas sur le même modèle. Une solution pour éviter cette incompréhension est de tirer partie du modèle générique qui est la base de tous les modèles. Il existe des passerelles entre les différents modèles à partir de ce modèle générique. Nous avons d'ailleurs montré que les transmuteurs sont des comportements qui décrivent de telles passerelles pour les objets d'application, les propriétés et même les opérateurs. Le manipulateur *M-Com* intègre des mécanismes complexes chargés d'exploiter les connaissances modélisées pour faciliter la communication. Pour y parvenir, il s'agit de :

- capturer le message de l'interlocuteur. Le manipulateur doit avoir un contrôle suffisamment important pour reconnaître un maximum de données ;

- décomposer le message en connaissances de base. La difficulté de cette opération dépend de la technique employée par le manipulateur pour saisir le message (langage naturel, langage par mots-clés ...) ; notre propos n'est pas de décrire une telle technique, mais de décrire globalement la gestion de la communication via le manipulateur générique *M-Com* ;
- traduire ces connaissances dans le modèle sur lequel se base l'opérateur récepteur du message. Le manipulateur transforme ces connaissances dans le modèle générique, puis dans celui du récepteur, grâce aux informations décrites par des transmuteurs associés à l'opérateur. On sous-entend que le manipulateur est capable de reconnaître le modèle de description des connaissances de base déduites du message.

Les mécanismes chargés de gérer la compréhension des opérateurs se révèlent très complexes. Nous avons voulu présenter de manière globale le problème en insistant sur le rôle joué par le manipulateur.

### 4.3.Synthèse.

Nous avons montré qu'il existe de grandes similitudes entre les agents et les opérateurs modélisés dans le monde virtuel. Cette constatation nous a permis de consolider notre approche de la modélisation des opérateurs. En effet, la plupart des données indispensables pour les agents sont prises en compte dans la modélisation des opérateurs (compétence, buts à atteindre ...). Cette étude nous a permis d'envisager une exploitation des techniques propres aux agents pour les opérateurs. En particulier, nous avons montré grâce aux concepts issus des SMA comment traiter la communication entre opérateurs. La solution proposée permet de continuer à fournir un dialogue adaptatif. La communication entre opérateurs constitue un type d'interactions que le système contrôle via le modèle de l'opérateur sans le contraindre grâce à l'anticipation de l'acte de communication et l'aide apportée pour résoudre les problèmes d'incompréhension. Cette composante intégrée au modèle de l'opérateur par l'intermédiaire d'un manipulateur montre toute la difficulté et la puissance apportées par la modélisation des opérateurs.

## 5. IMPLÉMENTATION.

Après avoir réalisé une implémentation pour montrer la génération du dialogue (cf. Chapitre 2 4.3. Implémentation), nous avons entrepris une seconde implémentation pour valider nos travaux sur les manipulateurs (Chapitre 3) et la modélisation de l'opérateur (Chapitre 4). Devant l'ampleur des concepts à considérer dans ce travail d'implémentation, nous avons choisi un exemple capable d'illustrer les principaux aspects sans pour autant obtenir une application réellement fonctionnelle. Il s'agit de réaliser une application permettant de concevoir un parcours de santé. Le développement de cette application s'est déroulé sur PC en Microsoft® Visual C++.

Dans la première partie, nous présentons l'application par l'ensemble des fonctionnalités et des objets qu'elle inclut. Nous montrons que cette application peut être assimilée à une application de CFAO de par son organisation générale. Ensuite, nous montrons la méthode suivie pour spécifier cette application en accord avec la modélisation des opérateurs et des manipulateurs.

## 5.1.Présentation.

L'application que nous avons voulu réaliser se situe dans le cadre de la RV. Elle propose la description d'un parcours de santé dans le monde virtuel. On suppose qu'on dispose d'une aire de parcours sur laquelle se trouve un sentier qui simule le chemin emprunté par le sportif. Le but de l'application est de placer des ateliers d'exercices de part et d'autre du sentier.

On définit une bibliothèque de type d'ateliers pour faciliter les traitements. On peut ainsi définir des poutres, des portiques et une série de plots. Il suffit de choisir un type d'atelier, le placer sur le parcours et le particulariser (par le type d'exercice associé ou par des contraintes). Pour parvenir à une telle description, nous introduisons trois types d'opérateurs pouvant intervenir :

- un opérateur jouant le rôle du concepteur d'ateliers. Il a à sa charge la spécification d'un exercice i.e. la construction éventuellement contrainte de l'atelier ;
- un opérateur dont le rôle est celui d'un fabricant. Il spécialise les ateliers en terme de fabrication (définition du matériau par exemple) ;
- un opérateur qui simule le sportif ou plutôt le client, contrôleur du parcours créé.

Ces trois types d'opérateurs sont différents par leur compétence et les données exploitées à leur sujet. Par exemple, on a besoin de connaître des caractéristiques physiques sur le client pour savoir s'il peut effectuer les exercices proposés. Ainsi, dans cette application on peut modéliser des opérateurs.

De plus, les opérateurs doivent pouvoir communiquer entre eux dans l'application, ce qui permet d'envisager la mise en place d'une détection de communication et d'un processus d'aides à la compréhension entre deux opérateurs (nous proposons dans ce travail d'implémentation la détection de communication).

En outre, pour faire fonctionner l'application, on définit des manipulateurs tant sur les ateliers que sur les opérateurs. Il s'agit de décrire des manettes pour modifier certains ateliers et des manipulateurs adaptés aux différentes capacités des opérateurs.

En définitive, le but de l'implémentation est de mettre en valeur la spécification des manipulateurs et des opérateurs dans une application quelconque de CFAO en RV. L'application choisie renferme des notions générales caractérisant les applications de CFAO : conception d'objets, intervention de différents opérateurs, notion de fabrication et simulation. L'annexe B donne des compléments d'informations quant aux différents types d'objets utilisés.

## 5.2.Spécification.

On dispose d'un modèle de description des objets de l'application (les ateliers et le sentier), ainsi que d'un modèle des opérateurs. Chaque objet, quel que soit son modèle d'appartenance, a été spécifié selon l'approche du modèle générique i.e. par un couple données/comportements (cf. Annexe B). Toutefois, nous n'avons pas déduit le dialogue, car décrire le processus de génération n'était pas le but recherché dans cette implémentation ; nous avons juste voulu montrer que l'on pouvait l'envisager.

Différents types de manipulateurs ont été définis afin de montrer que, quelle que soit sa classe d'appartenance, le manipulateur s'accorde avec la représentation sous la forme (Domaines, Aides) : par exemple, des manettes modifiant les hauteurs des ateliers, une calculatrice de création de contraintes sur les ateliers, un plan du parcours pour le concepteur afin qu'il place les nouveaux ateliers ...(cf. Annexe B). Chaque manipulateur possède des apparences particulières : pour le plan, il s'agit d'une fenêtre 2D et d'un bouton de mise en attente. Il inclut également des mécanismes propres aux manipulations qu'il propose : le plan, par exemple, identifie la désignation effectuée par le concepteur et l'interprète par une nouvelle position du concepteur sur le sentier, un atelier à modifier ou une position pour placer un atelier. Les Domaines des manipulateurs sont représentés plus implicitement par les liaisons entre les manipulateurs et les objets ou les opérateurs, et par le résultat de la manipulation effectuée. En outre, nous avons décrit la fenêtre 3D de visualisation du parcours comme un manipulateur général ainsi que nous l'avons préconisé dans le chapitre précédent (cf. Chapitre 3 3.4. Les manipulateurs généraux).

L'un des éléments moteurs de l'application est la gestion des thèmes abordés par les différents opérateurs. Globalement, nous avons défini cinq thèmes : création d'un atelier, modification d'un atelier, déplacement de l'opérateur, communication entre opérateurs, information sur un atelier.

D'une part, nous avons pu spécifier les buts de chaque opérateur en définissant les thèmes qui le préoccupent et en décrivant comment ils sont abordés. Rappelons que ces informations représentent une partie du sous-domaine des capacités *CA* de l'opérateur (cf. *3.1.2.Les capacités*). En outre, on représente la notion de thème commun entre plusieurs opérateurs (cf. figure IV.3). Par exemple, un concepteur et un fabricant sont tous les deux concernés par le thème de modification d'un atelier, mais ils n'effectuent pas les mêmes modifications : le dialogue n'est pas le même.

D'autre part, la gestion des thèmes nous a permis d'intégrer les manipulateurs dans le contrôle de cohérence de l'application. Dans le cadre de l'implémentation, nous nous sommes contentés de détecter les différents indices de situation et de les indiquer. En effet, notre but est avant tout de montrer le rôle dual du manipulateur i.e. sa capacité à gérer des outils de manipulations centralisant une partie du dialogue et sa capacité à suivre les fils d'activités multiples de l'opérateur grâce à ses compatibilités sur les thèmes.

Nous avons envisagé d'appliquer la détection de la communication au client. On n'effectue pas la détection à chaque interaction comme au *4.2.1.Détection*, mais dans le cas où le client s'approche d'un atelier. On propose de détecter s'il est capable d'effectuer l'exercice de l'atelier et de chercher à communiquer avec un concepteur si l'atelier ne lui convient pas. Si l'on reprend l'algorithme de détection de la communication (cf. *4.2.1. La détection*), les étapes sont simplifiées dans notre cas :

- *B* représente le thème d'information d'un atelier. Le client a pour but à cet instant de se renseigner sur l'exercice à effectuer pour voir s'il lui convient ;
- *R-Nec* représente l'atelier qu'il doit utiliser pour effectuer l'exercice et plus particulièrement les contraintes portant sur l'atelier et mentionnant la faisabilité de l'exercice (contrainte sur la hauteur de l'atelier par rapport au client, compétence sportive requise ...) ;

- *R-Eff* représente les données sur l'atelier (quand la contrainte porte sur sa hauteur par exemple) ;
- *Niv-C* représente des données propres au client et qui sont nécessaires à *R-Nec*. Il s'agit par exemple de ses compétences sportives (débutant ou confirmé) ;
- *ÉtatInteraction* représente le résultat de l'évaluation des contraintes entre l'atelier et le client.

Si le client ne peut effectivement pas accomplir l'exercice proposé, le système va rechercher un concepteur présent sur le parcours et le faire entrer en communication avec le client. Nous ne nous sommes pas penchés précisément sur la recherche du concepteur (utilisation des filtres variables ...) dans cette implémentation.

### 5.3.Bilan.

Les résultats obtenus sur cette première implémentation sont intéressants. Nous avons soulevé l'importance des manipulateurs dans la gestion du dialogue. Leur spécification a montré qu'ils s'intègrent parfaitement dans le système et plus encore, qu'une partie du dialogue est déléguée aux manipulateurs. De plus, la formalisation que nous avons apportée avec l'introduction des manipulateurs montre qu'il existe un mode de représentation commun à des outils de manipulations hétérogènes (manette, calculatrice, fenêtre ...) : le concept des manipulateurs demeure un atout essentiel pour décrire tout type d'outils de manipulations.

L'implémentation a mis en évidence l'intérêt de modéliser les opérateurs. Nous avons modélisé des données différentes selon le type d'opérateur (concepteur, fabricant, client). L'exemple le plus marquant concerne les buts (thèmes) différents qui les caractérisent ainsi que les manipulateurs qui leur sont propres. On ne procure pas à l'opérateur des outils inadaptés à sa compétence.

La gestion de la communication s'est avérée très délicate à gérer et demande une étude plus poussée. Toutefois, nous avons constaté que la modélisation des opérateurs joue un rôle important étant donné les informations qu'elle apporte sur les opérateurs.

## 6. CONCLUSION.

Notre objectif est celui de construire un dialogue adaptatif. Pour y parvenir, nous nous sommes intéressés dans ce chapitre à l'opérateur. Il est apparu que ce dernier intervient de manière plus ou moins explicite dès lors que l'on tente d'améliorer le dialogue. Les connaissances le concernant nous sont apparues primordiales tant elles sont exploitées dans le système. Ainsi, nous avons été amenés à proposer la modélisation de l'opérateur.

Bien que l'opérateur semble plus complexe à modéliser qu'un objet, nous nous sommes attachés à conserver la même approche de modélisation. Ainsi, l'opérateur est décrit dans un modèle générique par un couple données/comportements. On tend à dire que l'opérateur est un objet particulier. L'atout essentiel pour le dialogue est que l'on est désormais capable de décrire plus de connaissances. Il s'ensuit que le dialogue que l'on va pouvoir déduire devient plus complet. Des dialogues très courants

tels que celui associé au zoom peuvent être déduits en spécifiant pour un opérateur des comportements auxquels des manipulateurs sont éventuellement associés.

La modélisation de l'opérateur a également permis une meilleure gestion des environnements multi-utilisateurs. Pour y parvenir, nous avons étudié les analogies possibles avec les systèmes multi-agents. De cette analyse, nous avons cherché des éléments de réponses pour favoriser la communication entre opérateurs sans contraindre le dialogue homme/machine. Ainsi, nous sommes arrivés à tourner à notre avantage le problème de la communication en proposant une anticipation de l'acte de communication pour aider les cas de dialogue mal engagé. En effet, ce processus d'anticipation a pour but d'aider les opérateurs à interagir en évitant les conflits et en encourageant l'aide entre opérateurs. Nous avons également donné des éléments de réponses concernant les risques d'incompréhension entre opérateurs.

Ainsi, nous avons mis en évidence dans ce chapitre l'intérêt de modéliser l'opérateur. Non seulement, on peut maîtriser plus facilement les connaissances de l'opérateur et améliorer le dialogue existant, mais on arrive à proposer à l'opérateur une nouvelle dimension dans ses interactions (le clone par exemple).

Nos travaux ont abouti au développement d'une application dans laquelle les opérateurs et les manipulateurs jouent un rôle essentiel : plusieurs types d'opérateurs ont été spécifiés ainsi qu'un nombre représentatif de manipulateurs afin de montrer la gestion de la cohérence du dialogue d'un opérateur (gestion des thèmes). Le résultat obtenu montre les grandes capacités offertes par les manipulateurs et le modèle des opérateurs. De plus, sans vouloir réellement déduire le dialogue de l'application, nous avons spécifié chaque objet et chaque opérateur selon l'approche données/comportements (cf. Annexe B). Ainsi, on laisse entendre que le dialogue aurait pu être déduit grâce à cette spécification du modèle générique. Les connaissances acquises dans ce modèle sont de plus en plus pertinentes dans la mesure où elles ne s'attachent plus seulement à préciser les objets, mais également tous les autres éléments principaux de l'application i.e. les propriétés et les opérateurs. En conséquence, la spécification du modèle générique nous apparaît une bonne solution pour proposer un dialogue adaptatif.

## CONCLUSION.

Les travaux que nous avons présentés dans ce mémoire contribuent à l'amélioration du dialogue homme/machine. Nous nous sommes intéressés à la méthode d'acquisition des connaissances des utilisateurs concepteurs de dialogue, ainsi qu'à la représentation de connaissances pertinentes pour aider les utilisateurs opérateurs dans l'application. On se dirige vers un dialogue plus adaptatif dans lequel les utilisateurs (concepteurs de dialogue et opérateurs) peuvent se concentrer sur leurs propres objectifs.

Les besoins des interfaces évoqués dans le premier chapitre sont en bonne voie pour être résolu : l'utilisateur peut suivre ses fils d'activités multiples sans perturber le système, les outils qui lui sont proposés sont adaptés à ses compétences et, de manière générale, il est aidé dans ses dialogues. De plus, dans le cas où il n'est pas satisfait, il peut intervenir pour adapter les connaissances du système à ses propres connaissances grâce à la souplesse des concepts présentés (qu'il s'agisse de la méthode duale de description du dialogue, des comportements, des manipulateurs ou du modèle des opérateurs).

Le deuxième chapitre montre comment acquérir les connaissances nécessaires à la construction d'un dialogue adaptatif pour la CFAO. Nous nous sommes confrontés à des problèmes d'acquisition des connaissances similaires à ceux rencontrés dans les systèmes à base de connaissances présentés dans le premier chapitre. Nous avons dû définir une méthode duale d'acquisition afin de faire face aux compétences variables des concepteurs de dialogue :

- la première méthode est destinée à des concepteurs expérimentés dans la modélisation du dialogue telle que nous la percevons en CFAO ;
- la seconde méthode est destinée à tout utilisateur capable de décrire le dialogue par une description proche de l'utilisation de l'application. L'utilisateur décrit les comportements des objets manipulés dans l'application.

La seconde méthode reflète la volonté d'adapter le dialogue aux opérateurs, car on permet à un plus large public (voire aux opérateurs eux-mêmes) d'adapter la construction du dialogue. L'atout de cette méthode d'acquisition des connaissances est de déduire les éléments inhérents au dialogue à partir d'une description (couple données/comportements) relativement naturelle pour un utilisateur quelconque. L'implémentation que nous avons réalisée représente une preuve que la déduction du dialogue est possible. Au delà de nos espérances, nous sommes parvenus à décrire, par l'intermédiaire des comportements, un concept capable de capturer des connaissances beaucoup plus variées que celles concernant uniquement le dialogue. Nous avons montré à titre d'exemple que les comportements permettent de décrire des données sur la fabrication, ce qui est important pour les applications de CFAO. En effet, le système peut alors aider l'opérateur à décrire un produit que l'on sera capable de fabriquer. Bien que les bases de la méthode duale aient été posées et que chacune des deux approches fonctionne, nous n'avons pas encore étudié les problèmes de cohérence entre ces deux approches. Le système devrait pouvoir contrôler si l'ajout de connaissances par l'une des deux approches interfère avec le dialogue déjà mis en place.

Le troisième chapitre aborde le dialogue par les outils de manipulations mis à la disposition de l'opérateur : nous avons proposé un mode de représentation des connaissances sur les manipulations. Il nous a paru intéressant de définir le concept des manipulateurs pour représenter de manière générale un outil de manipulations. Cette approche nous a permis de montrer comment adapter le dialogue aux opérateurs par la description de leurs outils. Pour aller plus loin, nous avons tiré profit de ce nouveau mode de représentation des connaissances afin de fournir un dialogue toujours plus adaptatif. En effet, nous avons proposé d'utiliser ces connaissances sur les manipulations pour exercer un contrôle sur la cohérence de l'opérateur dans son dialogue et lui fournir des aides si son dialogue est mal engagé. En outre, les manipulateurs ont une place d'autant plus légitime dans le dialogue qu'ils sont une suite logique au concept des comportements. Nous proposons d'étudier maintenant comment déduire le dialogue sur les outils de manipulations à partir du modèle générique qui intègre des réacteurs extrinsèques associés à des manipulateurs.

Dans le quatrième chapitre, nous avons souligné l'importance de modéliser l'opérateur pour obtenir un dialogue adaptatif. En effet, nous avons montré l'intérêt d'un tel modèle pour l'acquisition des connaissances des opérateurs. Dans le premier chapitre, nous n'avons que trop souvent insisté sur le fait que les applications manquent de rigueur dans les dialogues proposés dans la mesure où ils ne sont pas en parfaite adéquation avec les capacités de l'opérateur. Nous avons montré que l'on pouvait associer aux opérateurs des manipulateurs proposant des fonctionnalités de haut niveau (le manipulateur clone ...), car, grâce au modèle proposé, on dispose d'informations suffisantes sur l'opérateur pour parvenir à les gérer convenablement. De plus, avec les environnements multi-utilisateurs, le modèle de l'opérateur permet de mieux gérer la communication entre opérateurs virtuels et d'apporter des aides. Il est clair que nous avons posé les bases de ce modèle et qu'un travail considérable reste à faire : la détection de la communication entre opérateurs avec la notion de filtres pour rechercher les opérateurs capables de communiquer, l'utilisation du modèle pour gérer le clone, la déduction du dialogue de l'opérateur à partir d'une description par les comportements. Nous souhaitons étudier de nouvelles exploitations des connaissances du modèle pour améliorer le dialogue.

Dans ce mémoire, nous avons montré une certaine unité dans les concepts développés, car chacun contribue de manière complémentaire à l'amélioration des dialogues. En définitive, nous sommes parvenus à obtenir un modèle générique centralisant de nombreuses connaissances utiles à la gestion d'un dialogue adaptatif : les objets (les propriétés, les objets d'application animés ou non et les opérateurs) décrits par un couple données/comportements et la description des outils de manipulations (manipulateurs associés aux réacteurs extrinsèques). L'étude que nous avons proposée nous a amenés à définir de nombreux concepts. Toutefois, nous sommes parvenus à en valider une partie dans deux implémentations qui ont demandé un important travail de développement.

Parallèlement à toutes les extensions de recherche que nous venons de proposer, nous sommes intéressés pas un problème majeure en CFAO : la multi-modélisation, que nous avons sans cesse évoquée dans ce mémoire. En effet, les concepts que nous avons établis (comportements, manipulateurs et modèle des opérateurs) renferment des connaissances qui sont, à notre avis, capables d'apporter des réponses au problème de l'interprétation des connaissances pour des modèles dédiés.

# RÉFÉRENCES BIBLIOGRAPHIQUES.

[ABO 90] G. D. ABOWD and R. BEALE. Users, systems and interfaces : A unifying framework for interaction. *HCI'91 : People and Computers VI*, D. Diaper and N. Hammond editors, Cambridge University Press, Cambridge, pages 73-87, 1991.

[ABO 92] G. ABOWD, J. COUTAZ and L. NIGAY. Structuring the Space of Interactive System Propeties. *IFIP TC2/WG2.7 Working Conference on Engineering for Human Computer Interaction*, Elsevier publisher, Finland, Ellivuori, august, 1992.

[ACM 92] ACM SIGCHI. *Curriculum for Human-Computer Interaction*. ACM Special Interest Group on Computer-Human Interaction Curriculum Development Group, USA, New York, 1992.

[ALL 93] C. D. ALLEN, D. BALLMAN, V. BEGG, H. H. MILLER-FACOBS, M. MULLER, J. NIELSEN and J. SPOOL. Users Involvement In The Design Process : Why, When & How ?. *INTERCHI'93*, ACM Press, pages 251-254, 24-29 april, 1993.

[ALP 93] S. R. ALPERT. Graceful Interaction with Graphical Constraints. *IEEE Computer Graphics & Applications*, pages 82-91, march, 1993.

[AND 95] R. ANDERL and R. MENDGEN. Parametric Design and its Impact on Solid Modeling Applications. *Proceeding of Third Symposium on Solid Modeling and Applications*, C. Hoffmann and J. Rossignac editors, ACM press, USA, Utah, Salt Lake City, pages 1-12, 17-19 may, 1995.

[AST 93] P. ASTHEIMER, W. FELGER and S. MÜLLER. Virtual design : a generic VR system for industrial applications. *Comput. & Graphics*, Pergamon Press, Great Britain, vol. 17, n°6, pages 671-677, 1993.

[AUS 62] J. L. AUSTIN. *How to Do Things With Words omput*. 1962, Clarendon Press, traduit en Français : Quand dire c'est faire, Le Seuil, 1970.

[BAE 87] R. M. BAECKER and W. A. S. BUXTON. *Readings in Human-Computer : A Multi-disciplinary Approach*. Los Altos, CA : Morgan Kaufmann, 1987.

[BAL 94] O. BALET, V. GAILDRAT and R. CAUBET. Le geste comme moyen d'expression. *Infographie Interactive et Intelligence Artificielle, Actes du congrés, Journée 3IA'94*, France, Limoges, pages 177-185, 6-7 avril, 1994.

[BAR 86] P. S. BARTH. An Object-Oriented Approach to Graphical Interfaces. *ACM Transactions on Graphics*, vol. 5, n°2, april, pages 142-172, 1986.

[BAR 92] I. BARDOT, L. BOCHEREAU, P. BOURGINE, B. HEYD, J. HOSSENLOPP, N. MARTIN, M. ROGEAUX and G.. TRYSTAM. Cuisiner Artificial : Un Automate Pour la Formulation Sensorielle de Produits Alimentaires. *Proceedings of Interface to Real and Virtual Worlds Conference*, France, Montpellier, pages 463-472, march, 1992.

[BIE 93] E. A. BIER, M. C. STONE, K. PIER, W. BUXTON and T. D. DEROSE. Toolglass and Magic Lenses : The See-Through Interface. *Computer graphics proceedings, Annual Conference Series, SIGGRAPH 93*, ACM Press, pages 73-80, 1-6 august, 1993.

[BON 84] A. BONNET. *L'intelligence artificielle : Promesses et réalités.* InterEditions, France, Paris, 1984.

[BOR 86] A. BORNING and R. DUISBERG. Constraint-Based Tools for Building User Interfaces. *ACM Transactions on Graphics*, vol. 5, n°4, october, pages 345-374, 1986.

[BOU 91] T. BOURON. What Architecture for Multi-Agent Systems. *In AAAI'91 Workshop on Cooperation among heterogeneous intelligent systems*, 1991.

[BOU 94] M. BOUGHANEM and D. PLEMENOS. Techniques d'apprentissage en modélisation déclarative de scènes par décomposition hiérarchique. *Infographie Interactive et Intelligence Artificielle, Actes du congrés, Journée 3IA'94*, France, Limoges, pages 99-120, 6-7 avril, 1994.

[BRO 84] D. C. BROWN and B. CHANDRASEKARAN. Expert Systems for a class of Mechanical Design Activity. *IFIP WG5.2*, september, 1984.

[BUR 92] G. BURDEA, J. ZHUANG, E. ROSKOS, D. SILVER and N. LANGRANA. A Portable Dextrous Master with Force Feedback. *Presence-Teleoperators and Virtual Environments*, vol. 1, n°1, pages 18-27, march, 1992.

[BUR 93a] G. BURDEA. Virtual Reality Systems and Applications. *Electro'93 International Conference*, Short Course, Edison, 164 pages, 28 april, 1993.

[BUR 93b] G. BURDEA et P. COIFFET. *La Réalité Virtuelle.* Éditions Hermès, 402 pages, 1993.

[BUT 95] P. BUTTOLO, D. KUNG and B. HANNAFORD. Manipulation in Real, Virtual and Remote Environments. *Proceedings IEEE Conference on System, Man and Cybernetics*, Canada, Vancouver, pages 4656-4661, 1995.

[CAR 90] L. CARRABINE. Plugging into the computer to sense. *Computer-Aided Eng.*, pages 16-26, june, 1990.

[CHA 93] B. CHABRIER. *Interfaces par contraintes graphiques.* Thèse d'université, France, Nice-Sophia Antipolis, 7 octobre, 1993.

[CHA 94a] B. CHABRIER et P. FRANCHI-ZANNETTACI. Délégation sémantique par contraintes réactives pour les interfaces graphiques. *Technique et science informatiques,* éditions Hermès, vol. 13, n°4, pages 539-566, 1994.

[CHA 94b] P. CHARVET et G. BERDAT. DFM Design for manufacturing. *Revue d'automatique et de productique appliquées*, éditions Hermès, vol. 7, n°1, pages 117-125, 1994.

[CHA 94c] S. CHATTY. De la manipulation directe à l'animation : la dynamique de l'interface. *Technique et science informatiques,* éditions Hermès, vol. 13, n°1, pages 63-78, 1994.

[CHO 91a] C. CHOQUET, A. PÉNINOU, G. GOUARDERES et B. MALSALLEZ. Un système de formation multituteurs : représentation et manipulation des connaissances. *Actes de la Convention IA 91*, éditions Hermès, pages 203-255, 1991.

[CHO 91b] C. CHOULET, A. HAURAT, F. SANDOZ et M. TEBAA. Méthode générale d'acquisition de la connaissance experte. *Actes de la Convention IA 91*, éditions Hermès, pages 113-126, 1991.

[CHU 95] M. C. CHUAH, S. F. ROTH, J. MATTIS and J. KOLOJEJCHICK. SDM : Selective Dynamic Manipulation of Visualizations. *Proceedings of UIST'95, ACM Symposium on User Interface Software and Technology*, ACM Press, USA, Pennsylvania, Pittsburgh, pages 61-70, 14-17 november, 1995.

[COI 95] P. COIFFET. *Mondes imaginaires*. Éditions Hermès, France, Paris, 127 pages, 1995.

[COU 90] J. COUTAZ. *Interfaces hommes-ordinateur, Conception et Réalisation*. Dunod informatique, France, Paris, 1990.

[COU 92] J. COUTAZ, J. CROWLEY and C. LAUGIER. Intelligence artificielle et interfaces homme-machine. *Actes des 4èmes Journées Nationales PRC-GDR Intelligence Artificielle*, France, Marseille, pages 167-187, 19-21 octobre, 1992.

[COU 94] J. COUTAZ et J. CAELEN. Les interfaces multimodales et la communauté française. *Technique et science informatiques*, vol. 13, n°1, pages 131-134, 1994.

[DEL 94] A. DELCHAMBRE et D. DUFOUR. Integrated Design for Assembly and Manufacturing (IDAM). *Revue d'automatique et de productique appliquées*, éditions Hermès, vol. 7, n°1, pages 99-106, 1994.

[DIX 93] A. DIX, J. FINLAY, G. ABOWD and R. BEALE. *Human-Computer Interaction*. Prentice Hall International editors, Cambridge University Press, Cambridge, 570 pages, 1993.

[EDM 94] E. EDMONDS, D. RIECKEN, R. SATHERLEY, K. STENNING and W. VISSER. Computers and Creative Thought. *ECAI'94 11th European Conference on Artificial Intelligence*, A. Cohn editor, J. Wiley & Sons publishers, pages 779-784, 1994.

[ELK 94] J. ELKINS. There are No Philosophic Problems Raised by Virtual Reality. *Computer Graphics*, vol. 28, n°4, november, pages 250-254, 1994.

[ENC 94] J. ENCARCAÇÃO, M. GÖBEL and L. ROSENBLUM. European Activities in Virtual Reality. *IEEE Computer Graphics & Applications*, pages 66-74, january, 1994.

[FAH 93] L. E. FAHLÉN, C. G. BROWN, O. STÅHL and C. CARLSSON. A Space Based Model for User Interaction in Shared Synthetic Environments. *INTERCHI'93*, ACM Press, pages 43-48, 24-29 april, 1993.

[FAR 88] X. FARGEAS et F. FRYDMAN. *Les systèmes esperts en médecine*. Collection Technologies de pointe, éditions Hermès, France, Paris, 1988.

[FAR 93] A. B. FARRAND, M. ROCHKIND, J. M. CHAUVET, B. T. TOGNAZZINI and D. C. SMITH. Common Elements in Today's Graphical User Interfaces : The Good, the Bad, and the Ugly. *INTERCHI'93*, ACM Press, pages 470-473, 24-29 april, 1993.

[FER 95] J. FERBER. *Les Systèmes Multi-Agents : vers une intelligence collective.* InterÉdition, France, Paris, 522 pages, 1995.

[FIG 93] M. FIGUEIREDO, K. BÖHM and J. TEIXEIRA. Advanced Interaction Techniques in Virtual Environments. *Comput. & Graphics*, Pergamon Press, Great Britain, vol. 17, n°6, pages 655-661, 1993.

[FOL 90] J. D. FOLEY, A. VAN DAM, S. K. FEINER and J. F. HUGHES. *Computer Graphics, principles and practice.* Second edition, Addison-Wesley publishing company, 1175 pages, 1990.

[FOU 94] J. P. FOURNIER. Vers des programmes dociles. *Technique et science informatiques,* éditions Hermès, vol. 13, n°1, pages 79-104, 1994.

[FRA 93] M. R. FRANK and J. D. FOLEY. Model-based user interface design by example and by interview. *Proceedings of UIST'93, ACM Symposium on User Interface Software and Technology*, pages 1-9, november, 1993.

[FRO 93 ] D. M. FROHLICH. The history and future of direct manipulation. *Behaviour & information technology*, vol. 12, n°6, pages 315-329, 1993.

[GAR 88] Y. GARDAN, J. P. JUNG, J. N. KOLOPP, C. MINICH et W. TOTINO. Une approche nouvelle de la convivialité dans un système de CAO : les principes du dialogue dans SACADO. *Actes de MICAD 88*, éditions Hermès, France, Paris, pages 281-296, 21-25 mars, 1988.

[GAR 91] Y. GARDAN. *La CFAO introduction, techniques, et mise en œuvre.* Éditions Hermès, France, Paris, 418 pages, décembre, 1991.

[GAR 93] Y. GARDAN, J. P. JUNG and B. MARTIN. An end-User oriented approach to design man-machine interface for CAD/CAM. *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, France, Le Touquet, pages 525-530, 17-20 october, 1993.

[GAR 95a] Y. GARDAN, J. P. JUNG, S. LEINEN, B. MARTIN, C. MINICH, C. POINSIGNON et I. STÉMART. *Conception et réalisation d'une maquette illustrant une approche du dialogue, de la gestion des contraintes et de la modélisation en CAO.* Rapport de recherche, LRIM, France, Metz, n°95-01, 102 pages, mars, 1995.

[GAR 95b] Y. GARDAN, B. MARTIN et I. STÉMART. *Modélisation des comportements : un lien entre modèles et dialogues.* Rapport de recherche, LRIM, France, Metz, n°95-02, 33 pages, avril, 1995.

[GAR 96a] Y. GARDAN, B. MARTIN et I. STÉMART. Utilisation des connaissances du modèle en CFAO : application à la génération du dialogue. *MICAD'96, actes dans la Revue Internationale de CFAO et d'informatique graphique*, éditions Hermès, France, Paris, vol. 11, n°1-2, pages 71-89, 13-15 février, 1996.

[GAR 96b] Y. GARDAN, B. MARTIN et I. STÉMART. Behaviors : from man-machine interface to design for manufacturing. *Proceeding of CESA'96 IMACS Multiconference, Computational Engineering in Systems Applications, symposium on Discrete Events and*

*Manufacturing Systems, IEEE-SMC*, P. Borne, J. C. Gentina, E. Craye and S. El Khattabi editors, France, Lille, pages 530-535, 9-12 july, 1996.

[GAR 96c] Y. GARDAN, C. MINICH et C. POINSIGNON. Propositions pour un modèle produit. *Proceeding of First International Conference IDMME'96*, France, Nantes, tome 2, pages 827-836, 15-17 april, 1996.

[GOB 95] E. GOBBETTI and J. F. BALAGUER. An Integrated Environment to Visually Construct 3D Animations. *Computer Graphics Proceedings, Annual Conference Series, SIGGRAPH 95*, ACM Press, USA, California, Los Angeles, pages 395-398, 6-11 august, 1995.

[GRA 92a] T. C. N. GRAHAM. Constructing User Interfaces with Functions and Temporal Constraints. *Languages for Developping User Interfaces*, B. A. Myers editor, J. Bartlett publisher, pages 279-299, 1992.

[GRA 92b] T. C. N. GRAHAM. Future Research Issues in Langages for Developing User Interfaces. *Languages for Developping User Interfaces*, B. A. Myers editor, J. Bartlett publisher, pages 401-419, 1992.

[GRA 94] B. GRAU, G. SABAH et A. VILNAT. Pragmatique et dialogue homme-machine. *Technique et science informatiques,* éditions Hermès, vol. 13, n°4, pages 9-30, 1994.

[HAT 89] J.P. HATON et M.C. HATON. *L'Intelligence Artificielle*. Que sais-je?, presses Universitaires de France, 127 pages, avril, 1989

[HAY 94] F. HAYES-ROTH and N. JACOBSTEIN. The State of Knowledge-Based Systems. *Communications of the ACM*, vol. 37, n°3, pages 27-39, 1994.

[HEI 93] HEIM. *Metaphysics of Virtual Reality*. Oxford and New York : Oxford University Press, 91 pages, 1993.

[HER 88] L. E. C. HERN. On Distributed Artificial Intelligence. *The Knowledge Engineering Review*, Cambridge University Press, vol. 3, n°1, january, 1988.

[HER 94] K. P. HERNDON, A . VAN DAM and M. GLEICHER. The Challenges of 3D Interaction. *SIGCHI Bulletin,* vol. 26, n°4, pages 36-43, october, 1994.

[HUB 89] W. HÜBNER. Two Object-Oriented Models to Design Graphical User Interfaces. *EUROGRAPHICS'89*, W. Hansmann, F. R. A. Hopgood and W. Stasser editors, Elsevier Science publishers B. V., North Holland, pages 63-75, 1989.

[JAK 63] R. JAKOBSON. *Essai de linguistique générale*. Éditions de Minuit, 1963.

[JAN 93] C. JANSSEN, A. WEISBECKER and J. ZIEGLER. Generating User Interfaces from Data Models and Dialogue Net Specifications. *INTERCHI'93*, ACM Press, pages 418-423, 24-29 april, 1993.

[JOO 95] Y. JOO and R. R. BARTON. Integrated product and process design through feedback of manufacturing experience. *Computers & Industrial Engineering*, Pergamon Press, vol. 28, n°3, pages 561-573, 1995.

[KAH 94] D. KAHANER. Japanese Activities in Virutal Reality. *IEEE Computer Graphics & Applications,* pages 75-78, january, 1994.

[KAN 95] E. KAN. Réalité virtuelle et CAO : bientôt la maturité. *Le Monde Informatique, Informatique industrielle*, supplément n°654, pages 6-10, 1995.

[KRI 95] S. KRISHNAN and K. SRIHARI. A Knowledge-Based Object Oriented DFM Advisor for Surface Mount PCB Assembly. *The International Journal of Advanced Manufacturing Technology*, Great Britain, vol. 10, n°5, pages 317-329, 1995.

[LAT 94] J. N. LATTA and D. J. OBERG. A Conceptuel Virtual Reality Model. *IEEE Computer Graphics & Applications,* pages 23-29, january, 1994.

[LEM 92] B. LEMAIRE. Construction d'explications : utilisation d'une architecture de tableau noir. *Actes des 4èmes Journéees Nationales PRC-GDR Intelligence Artificielle*, France, Marseille, pages 471-494, 19-21 octobre, 1992.

[LIA 91] Y. LIANG, B. HOURIEZ, R. DUPAS et P. MILLOT. Un modèle d'apprentissage pour un système expert de diagnostic. *Convention IA 91*, éditions Hermès, pages 259-276, 1991.

[MAR 95] B. MARTIN. *Contribution pour une nouvelle approche du dialogue homme-machine en C.F.A.O.* Thèse d'université, France, Metz, 189 pages, 22 décembre, 1995.

[MON 94] C. MONY. DFM : enjeux, tendances et état de l'art. *Revue d'automatique et de productique appliquées*, éditions Hermès, vol. 7, n°1, pages 15-25, 1994.

[MOR 81] T. P. MORAN. The command langage grammar : a representation for the user interface of interactive systems. *International Journal of Man-Machine Studies*, pages 3-50, 1981.

[MYE 90a] B. A. MYERS. Taxonomies of Visual Programming and Program Visualization. *Journal of Visual Langages and Computing*, Academic Press, vol. 1, pages 97-123, 1990.

[MYE 90b] B. A. MYERS, D. A. GIUSE, R. B. DANNENBERG, B. VANDER ZANDEN, D. S. KOSBIE, E. PERVIN, A. MICKISH and P. MARCHAL. Garnet : Comprehensive Support for Graphical Highly Interactive User Interfaces. *Readings in Human-Computer Interaction : Toward the Year 2000*, R. M. Baecker, J. Grudin, W. A. S. Buxton and S. Greenberg writers and editors, second edition, M. Kaufmann publishers, USA, California, San Francisco, pages 357-371, 1995.

[MYE 93a] B. A. MYERS. State of the Art in User Interface Software Tools. *Readings in Human-Computer Interaction : Toward the Year 2000*, R. M. Baecker, J. Grudin, W. A. S. Buxton and S. Greenberg writers and editors, second edition, M. Kaufmann publishers, USA, California, San Francisco, pages 323-343, 1995.

[MYE 93b] B. A. MYERS, R. WOLF, K. POTOSNAK and C. GRAHAM. Heuristics in Real User Interfaces. *INTERCHI'93*, ACM Press, pages 304-307, 24-29 april, 1993.

[NEW 93] G. B. NEWBY. Virtual Reality. *Annual review of information science and technology (ARIST)*, USA, vol.28, pages 187-229, 1993.

[NGU 94] C. NGUYEN. CAO*I* Une modélisation 3D par contraintes. *Infographie Interactive et Intelligence Artificielle, Actes du congrés, Journée 3IA'94*, France, Limoges, pages 167-176, 6-7 avril, 1994.

[NOR 88] D. A. NORMAN. *The Psychology of Everyday Things.* Basic Books, 1988.

[OHA 96] G. M. P. O'HARE, S. ABBAS and R. PIKE. The MODSET System, an Engineering Design Environment for Subsea Technologies : An Experiment in Agent-Oriented Design. *Proceeding of CESA'96 IMACS Multiconference, Computational Engineering in Systems Applications, symposium on Robotics and Cybernetics, IEEE-SMC*, P. Borne, R. Soenen, Y. Sallez and S. El Khattabi editors, France, Lille, pages 256-261, 9-12 july, 1996.

[PAA 88] K. R. PAAP and R.J. ROSKE-HOFSTRAND. Design of menus. *In Handbook of Human-Computer Interaction*, M. Helander editor, North-Holland, Amsterdam, 1988.

[PAN 93] G. J. PANGOLAS. Designing the user interface, *Computers in Industry*, n° 22, pages 193-200, 1993.

[PIM 93] K. PIMENTEL and B. BLAU. System Architecture Issues Related to Multiple-User VR Systems : Teaching Your System to Share. *Proceedings of VR Systems'93 Conference*, USA, New York, pages 125-133, march, 1993.

[PIM 94] K. PIMENTEL et K. TEIXEIRA. *La réalité virtuelle...de l'autre côté du miroir.* Addison-Wesley publishing company,1994.

[PIP 95] T. PIPIERI. *Implémentation d'un nouveau dialogue dans le système SACADO.* Mémoire d'ingénieur CNAM, LRIM, février, 1995.

[POM 88] J. C. POMEROL. *Les systèmes experts.* Collection Technologies de pointe, éditions Hermès, France, Paris, 63 pages, 1988.

[PRE 94] J. PREECE, Y. ROGERS, H. SHARP, D. BENYON, S. HOLLAND and T. CAREY. *Human-Computer Interaction.* Addison-Wesley publishing company, 775 pages, 1994.

[QUE 94] P. QUENCEZ et M. SPADONI. Un modèle de représentation des connaissances et de l'apprentissage pour la conception d'objets technologiques. *Proceedings of IFIP 94 international conference*, France, Valenciennes, pages 243-258, 1994.

[ROC 97] J. P. ROCHÉ. *Définition et mise en œuvre de Logiciels Graphiques de Base en C++.* Mémoire d'ingénieur CNAM, LRIM, à paraître, 1997.

[ROL 91] D. ROLLER. An Approach to computer-aided parametric design. *Computer-Aided Design*, vol. 23, n°5, pages 385-391, june, 1991.

[SAU 94] A. SAUCIER, C. VARGAS, P. COURT, P. ALBERT and P. A. YVARS. Compared application of two knowledge modelisation methodologies on a car engine cylinder head design problem. *Proceedings of IFIP 94 international conference*, France, Valenciennes, pages 59-78, 1994.

[SEA 69] J. R. SEARLE. *Speechs Acts.* Cambridge University Press, 1969.

[SEA 79] J. R. SEARLE. *Expression and Meaning.* Cambridge University Press, 1979.

[SHA 48] C. SHANNON and W. WEAVER. *The Mathematical theory of Communication.* University of Illinois Press, USA, Urbana, 1948.

[SHN 95] B. SHNEIDERMAN. A taxonomy and rules base for the selection of interaction styles. *Readings in Human-Computer Interaction : Toward the Year 2000*, R. M. Baecker, J. Grudin, W. A. S. Buxton and S. Greenberg writers and editors, second edition, M. Kaufmann publishers, USA, California, San Francisco, pages 401-410, 1995.

[STE 94] I. STÉMART. *Définition et mise en œuvre d'une nouvelle primitive de dialogue pour les contraintes en C.A.O.* Mémoire de DEA, CRIN-LRIM, France, Metz/Nancy, 48 pages, 9 septembre, 1994.

[SUK 93] P. N. SUKAVIRIYIA, J. D. FOLEY and T. GRIFFITH. A Second Generation User Interface Design Environment : The Model and The Runtime Architecture. *INTERCHI'93*, ACM Press, pages 375-382, 24-29 april, 1993.

[SUK 94] N. SUKAVIRIYA, S. KOVACEVIC, J. D. FOLEY, B. A. MYERS, D. R. OLSEN JR. and M. SCHNEIDER-HUFSCHMIDT. Model-Based User Interfaces : What are They and Why Should We Care ? *Proceedings of UIST'94, the ACM Symposium on User Interface Software and Technology*, ACM Press, USA, California, Marina del Rey, pages 133-135, 2-4 november, 1994.

[SUN 92] H. SUNDGREN, F. WINQUIST and I. LUNDSTROM. Artificial Olfactory System Based on Field Effect Devices. *Proceedings of the Interface to Real and Virtual Worlds*, France, Montpellier, pages 463-472, march, 1992.

[TOL 92] M. TOLLENAERE et R. CHAMBON. Un exemple de système CAO intelligent. *MICAD 92*, éditions Hermès, France, vol. 2, pages 359-378, 1992.

[USN 96] U.S. NATIONAL LIBRARY OF MEDECINE. The Visible Human Project. *Site Internet www.nlm.nih.gov/research/visible/visible-human.html*, 1996.

[VAN 88] D. VANDERVEKEN. *Les actes de discours.* Pierre Mardaga, 1988.

[VEN 93] D. VENOLIA. Facile 3D Direct Manipulation. *INTERCHI'93*, ACM Press, pages 31-36, 24-29 april, 1993.

[WIE 95] C. WIECHA, W. BENNETT, S. BOIES, J. GOULD and S. GREENE. ITS : A Tool for Rapidly Developing Interactive Applications. *Readings in Human-Computer Interaction : Toward the Year 2000*, R. M. Baecker, J. Grudin, W. A. S. Buxton and S. Greenberg writers and editors, second edition, M. Kaufmann publishers, USA, California, San Francisco, pages 373-389, 1995.

[YEN 93] J. YEN and J. LEE. A Task-Based Methodology for Specifying Expert Systems. *IEEE Expert*, pages 8-15, february, 1993.

# ANNEXE A.

# IMPLÉMENTATION DES COMPORTEMENTS

## 1. INTRODUCTION.

Dans cette annexe, nous détaillons la méthode de développement suivie pour réaliser l'implémentation sur la génération du dialogue que nous avons présentée dans le chapitre 2 (cf. Chapitre 2 4.3. Implémentation).

Rappelons que cette implémentation montre comment décrire une application graphique interactive en utilisant la description données/comportements, afin de déduire le dialogue complet. L'application que nous décrivons est un outil de manipulations d'un formalisme de description de dialogue s'apparentant au formalisme de SACADO. Elle comprend trois notions importantes :

- création d'objets graphiques. La figure A.1 indique l'ensemble des objets graphiques que l'opérateur pourra manipuler dans l'application à travers deux fenêtres d'affichage (l'une pour permettre à l'opérateur de décrire une hiérarchie de menus, l'autre pour décrire une action globale associée à un menu terminal) ;

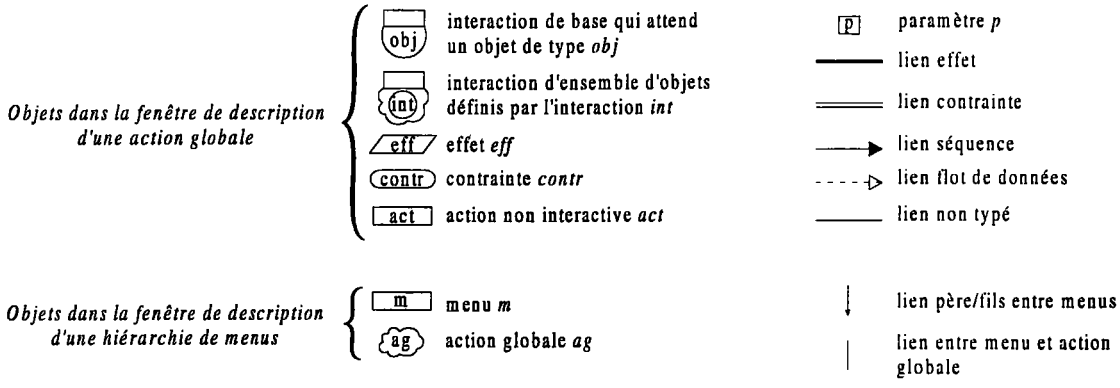



***Figure A.1.*** *Les objets de l'application.*

- intervention de l'opérateur pendant la création. La création d'un objet n'est pas seulement le résultat d'un appel à un menu, l'opérateur peut fournir des informations complémentaires ;
- aides à l'opérateur par des mises en évidence (affichages en élastiques pendant les créations, aides textuelles) et des détections (détections des propriétés horizontal, vertical et collé contre, et détections de changements de type). La figure A.2. montre un aperçu des aides obtenues pour l'application.

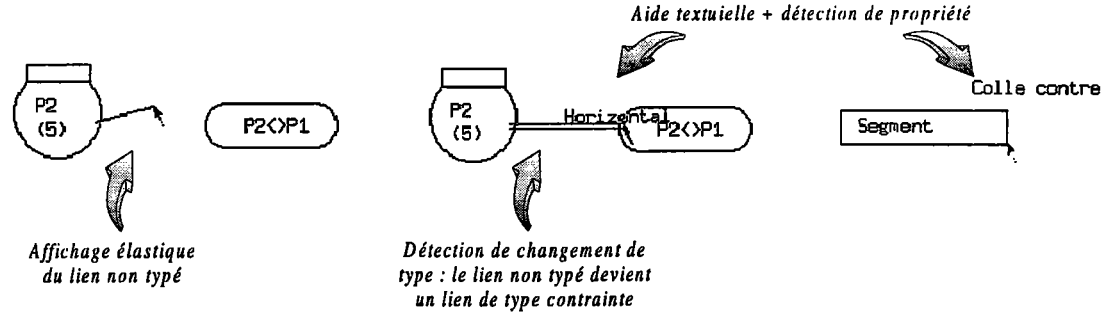



***Figure A.2.*** *Exemple d'aides apportées à l'opérateur pendant la manipulation de l'application.*

L'implémentation a été effectuée selon une approche objet. Ainsi, nous décrivons dans la première partie les différentes classes d'objets permettant de décrire le modèle générique selon l'approche

données/comportements. Ensuite, nous montrons comment un objet d'application quelconque doit être spécifié. Enfin, nous présentons le processus de génération du dialogue à travers les algorithmes mis en place ainsi que les fichiers interprétés décrivant le dialogue et qui sont issus de la génération.

## 2. LES CLASSES.

Le modèle générique est composé de deux modèles : le modèle décrivant les propriétés gérées dans l'application et le modèle décrivant les objets manipulés dans l'application. La figure A.3. montre la hiérarchie globale de classes, utilisée pour décrire le modèle générique.

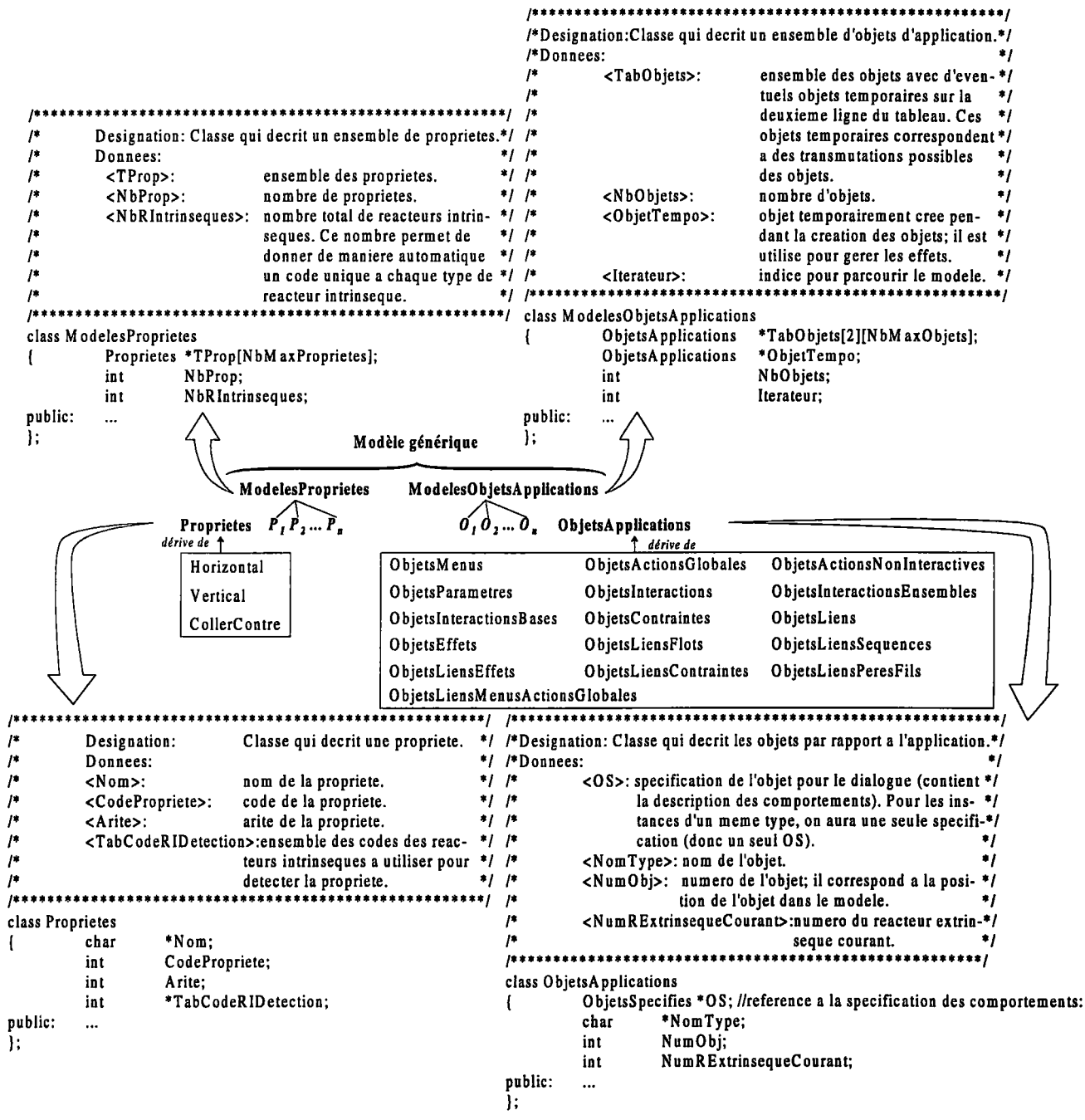



*Figure A.3. Hiérarchie globale des classes décrivant le modèle générique.*

Les propriétés ne sont pas décrites par des comportements dans cette implémentation. Toutefois, elles représentent des entités à part entière dans le modèle générique. De manière générale, nous avons

défini un objet *Propriétés* duquel chaque propriété effective hérite. Cet objet regroupe les informations de base d'une propriété i.e. son mode d'identification (nom, code), son arité et les réacteurs intrinsèques qui permettent de fournir les données pour l'évaluer.

Chaque objet d'application dérive d'un objet plus général *ObjetsAplications*. Cet objet contient, outre l'identifiant de l'objet d'application (nom, numéro), l'ensemble des comportements qui lui sont associés. Afin de bien distinguer le couple données/comportements dans le modèle, cette description des comportements est représentée par un objet *ObjetsSpecifies* (cf. figure A.4).

```
/*************************************************************************/
/*      Designation:        Classe qui decrit les objets pour le dialogue.   */
/*      Donnees:                                                             */
/*      <TabProducteurs>:   ensemble des producteurs de l'objet.             */
/*      <NbProducteurs>:    nombre de producteurs associes a l'objet.        */
/*      <TabRIntrinseques>: ensemble des reacteurs intrinseques utilises par l'objet. */
/*      <NbRIntrinseques>:  nombre de reacteurs intrinseques associes a l'objet. */
/*      <TabRExtrinseques>: ensemble des reacteurs extrinseques utilises par l'objet. */
/*      <NbRExtrinseques>:  nombre de reacteurs extrinseques associes a l'objet. */
/*      <TabTransmuteurs>:  ensemble des transmuteurs utilises par l'objet.  */
/*      <NbTransmuteurs>:   nombre de transmuteurs associes a l'objet.       */
/*      <NumType>:          type de l'objet specifie (depend de sa position dans le */
/*                          modele des objets specifies.                     */
/*************************************************************************/
class ObjetsSpecifies
{
        Producteurs         *TabProducteurs[NbMaxProducteurs];
        RIntrinseques       *TabRIntrinseques[NbMaxRIntrinseques];
        RExtrinseques       *TabRExtrinseques[NbMaxRExtrinseques];
        Transmuteurs        *TabTransmuteurs[NbMaxTransmuteurs];
        int  NbProducteurs, NbRIntrinseques, NbRExtrinseques, NbTransmuteurs;
        int  NumType;

public:  ...
};
```

***Figure A.4.** Classe représentant l'objet d'application par ses comportements.*

### 2.1.Les objets d'application.

On décrit, pour chaque type particulier d'objets intervenant dans l'application, une classe dérivant de la classe *ObjetsApplications*. La classe obtenue permet de décrire les données i.e. le domaine de définition de l'objet d'application particulier. Dans ce qui suit, nous énumérons les différentes classes créées en montrant les données modélisées pour chacune d'elles :

```
/*********************************************/
/*      Designation: Classe qui decrit les menus.   */
/*      Donnees:                                    */
/*      <Position>:position du menu dans la fenetre,*/
/*                 le coin haut gauche de la boite. */
/*      <NomInstance>:nom du menu.                  */
/*********************************************/
class ObjetsMenus: public ObjetsApplications
{
        Points    *Position;
        char      *NomInstance;
public:  ...
};
```

```
/*********************************************/
/*      Designation: Classe qui decrit les interac-  */
/*                   tions de base.                  */
/*      Donnees:                                     */
/*      <NomInstance>: nom de l'interaction.         */
/*      <Type>: type attendu par l'interaction.      */
/*      <Position>: position de l'interaction dans la */
/*                  fenetre.                         */
/*********************************************/
class ObjetsInteractionsBases: public ObjetsApplications
{
        char      *NomInstance;
        int       Type;
        Points    *Position;
public:  ...
};
```

```
/*************************************************/
/*      Designation: Classe qui decrit les actions   */
/*                   globales.                       */
/*      Donnees:                                     */
/*      <Position>:position de l'action globale dans */
/*                 la fenetre, le centre du "nuage". */
/*      <NomInstance>:nom de l'action globale.       */
/*      <TabCentres>:centres des cercles pour re-    */
/*                   presenter l'objet.              */
/*************************************************/
class ObjetsActionsGlobales: public ObjetsApplications
{
        Points      *Position;
        char        *NomInstance;
        Points      *TabCentres[7];
public: ...
};

/*************************************************/
/*      Designation: Classe qui decrit les interac-  */
/*                   tions au sens general, i.e. l'en-*/
/*                   semble constitue d'une iteration*/
/*                   de base ou d'ensemble et de ses*/
/*                   eventuels contraintes et effets. */
/*      Donnees:                                     */
/*      <Position>: position de l'interaction, centre */
/*                   de l'objet.                     */
/*      <NomInstance>: nom de l'interaction.         */
/*      <TabObj>: ensemble des objets constituant    */
/*                   l'interaction.                  */
/*      <NbObj>:  nombre d'objets constituant l'in-  */
/*                   teraction.                      */
/*************************************************/
class ObjetsInteractions: public ObjetsApplications
{
        Points      *Position;
        char        *NomInstance;
        int         TabObj[NbMaxObjets];
        int         NbObj;
public: ...
};

/*************************************************/
/*      Designation: Classe qui decrit les effets.   */
/*      Donnees:                                     */
/*      <NomInstance>: nom de l'effet.               */
/*      <Position>:position de l'effet dans la fenetre.*/
/*      <TabSommets>:sommets du parallelogram-       */
/*                   me representant l'effet.        */
/*************************************************/
class ObjetsEffets: public ObjetsApplications
{
        char        *NomInstance;
        Points      *Position;
        Points      *TabSommets[5];
public: ...
};

/*************************************************/
/*      Designation: Classe qui decrit les actions   */
/*                   non interactives.               */
/*      Donnees:                                     */
/*      <NomInstance>: nom de l'action.              */
/*      <Position>: position de l'action non interac- */
/*                   tive dans la fenetre.           */
/*************************************************/
class ObjetsActionsNonInteractives: public ObjetsApplications
{
        char        *NomInstance;
        Points      *Position;
public: ...
};

/*************************************************/
/*      Designation: Classe qui decrit les parametres*/
/*      Donnees:                                     */
/*      <NomInstance>: nom de l'action.              */
/*      <Position>: position du parametres dans la   */
/*                   fenetre.                        */
/*************************************************/
class ObjetsParametres: public ObjetsApplications
{
        char        *NomInstance;
        Points      *Position;
public: ...
};
```

```
/*************************************************/
/*      Designation: Classe qui decrit les interac-  */
/*                   tions d'ensemble.               */
/*      Donnees:                                     */
/*      <NomInstance>: nom de l'interaction.         */
/*      <InterGenerale>: interaction generale defi-  */
/*                   nissant les objets attendus     */
/*                   par l'ensemble.                 */
/*      <TabCentres>:  centres des cercles pour re-  */
/*                   presenter l'objet.              */
/*************************************************/
class ObjetsInteractionsEnsembles: public ObjetsApplications
{
        char                *NomInstance;
        ObjetsInteractions  *InterGenerale;
        Points              *TabCentres[6];
public: ...
};

/*************************************************/
/*      Designation: Classe qui decrit les contrain-  */
/*                   tes.                            */
/*      Donnees:                                     */
/*      <NomInstance>: nom de la contrainte.         */
/*      <Position>: position de la contrainte dans la */
/*                   fenetre, le coin haut gauche du */
/*                   rectangle utilise pour la repre-*/
/*                   senter.                         */
/*      <Exp>: arbre contenant la contrainte.        */
/*************************************************/
class ObjetsContraintes: public ObjetsApplications
{
        char                *NomInstance;
        Points              *Position;
        ExpressionsGenerales *Exp;
public: ...
};

/*************************************************/
/*      Designation: Classe qui decrit les liens de  */
/*                   type sequence.                  */
/*      Donnees:                                     */
/*      <ObjetOrigine>: numero de l'objet qui con-   */
/*                   tient l'origine du lien.        */
/*      <ObjetExtremite>: numero de l'objet qui      */
/*                   contient l'extremite du lien.*/
/*      <PointOrigine>: point origine du lien.       */
/*      <PointExtremite>: point extremite du lien.   */
/*      <TabSommets>: sommets du parallelogram-      */
/*                   me representant le lien.        */
/*************************************************/
class ObjetsLiensSequences: public ObjetsApplications
{
        int         ObjetOrigine, ObjetExtremite;
        Points      *PointOrigine, *PointExtremite;
public: ...
};

/*************************************************/
/*      Designation: Classe qui decrit les liens de  */
/*                   type flot.                      */
/*      Donnees:                                     */
/*      <ObjetOrigine>: numero de l'objet qui con-   */
/*                   tient l'origine du lien.        */
/*      <ObjetExtremite>: numero de l'objet qui      */
/*                   contient l'extremite du lien.*/
/*      <PointOrigine>: point origine du lien.       */
/*      <PointExtremite>: point extremite du lien.   */
/*      <TabSommets>: sommets du parallelogram-      */
/*                   me representant le lien.        */
/*************************************************/
class ObjetsLiensFlots: public ObjetsApplications
{
        int         ObjetOrigine, ObjetExtremite;
        Points      *PointOrigine, *PointExtremite;
public: ...
};
```

```
/*************************************************/
/*        Designation: Classe qui decrit les liens non  */
/*                     types.                    */
/*        Donnees:                                */
/*        <ObjetOrigine>: numero de l'objet qui con-  */
/*                     tient l'origine du lien.  */
/*        <ObjetExtremite>: numero de l'objet qui  */
/*                     contient l'extremite du lien.*/
/*        <PointOrigine>: point origine du lien.  */
/*        <PointExtremite>: point extremite du lien.  */
/*        <TabSommets>: sommets du parallelogram-*/
/*                     me representant le lien.  */
/*************************************************/
class ObjetsLiens: public ObjetsApplications
{
        int       ObjetOrigine, ObjetExtremite;
        Points    *PointOrigine, *PointExtremite;
public:   ...
};

/*************************************************/
/*        Designation: Classe qui decrit les liens de  */
/*                     type effet.               */
/*        Donnees:                                */
/*        <ObjetOrigine>: numero de l'objet qui con-  */
/*                     tient l'origine du lien.  */
/*        <ObjetExtremite>: numero de l'objet qui  */
/*                     contient l'extremite du lien.*/
/*        <PointOrigine>: point origine du lien.  */
/*        <PointExtremite>: point extremite du lien.  */
/*        <TabSommets>: sommets du parallelogram-*/
/*                     me representant le lien.  */
/*************************************************/
class ObjetsLiensEffets: public ObjetsApplications
{
        int       ObjetOrigine, ObjetExtremite;
        Points    *PointOrigine, *PointExtremite;
        Points    *TabSommets[5];
public:   ...
};
```

```
/*************************************************/
/*        Designation: Classe qui decrit les liens de  */
/*                     type contrainte.          */
/*        Donnees:                                */
/*        <ObjetOrigine>: numero de l'objet qui con-  */
/*                     tient l'origine du lien.  */
/*        <ObjetExtremite>: numero de l'objet qui  */
/*                     contient l'extremite du lien.*/
/*        <PointOrigine>: point origine du lien.  */
/*        <PointExtremite>: point extremite du lien.  */
/*        <TabSommets>: sommets du parallelogram-*/
/*                     me representant le lien.  */
/*************************************************/
class ObjetsLiensContraintes: public ObjetsApplications
{
        int       ObjetOrigine, ObjetExtremite;
        Points    *PointOrigine, *PointExtremite;
        Points    *TabSommets[5];
public:   ...
};

/*************************************************/
/*        Designation: Classe qui decrit les liens entre*/
/*                     un menu et une action globale.*/
/*        Donnees:                                */
/*        <ObjetOrigine>: numero de l'objet qui con-  */
/*                     tient l'origine du lien.  */
/*        <ObjetExtremite>: numero de l'objet qui  */
/*                     contient l'extremite du lien.*/
/*        <PointOrigine>: point origine du lien.  */
/*        <PointExtremite>: point extremite du lien.  */
/*************************************************/
class ObjetsLiensMenusActionsGlobales: public ObjetsApplications
{
        int       ObjetOrigine, ObjetExtremite;
        Points    *PointOrigine, *PointExtremite;
public:   ...
};
```

```
/*************************************************/
/*        Designation: Classe qui decrit les liens entre*/
/*                     deux menus.               */
/*        Donnees:                                */
/*        <ObjetOrigine>: numero de l'objet qui con-  */
/*                     tient l'origine du lien.  */
/*        <ObjetExtremite>: numero de l'objet qui  */
/*                     contient l'extremite du   */
/*        <PointOrigine>: point origine du lien.  */
/*        <PointExtremite>: point extremite du lien.  */
/*************************************************/
class ObjetsLiensPeresFils: public ObjetsApplications
{
        int       ObjetOrigine, ObjetExtremite;
        Points    *PointOrigine, *PointExtremite;
public:   ...
};
```

## 2.2.Les propriétés.

Chaque propriété dérive d'un objet de la classe *Proprietes*. On décrit dans chaque classe dérivée, les données propres à la définition de la propriété i.e. son domaine de définition. Nous avons défini trois propriétés dans l'application (horizontal, vertical et poser contre). Ainsi, les classes suivantes ont été définies :

```
/*********************************************/
/*      Designation: Classe qui decrit la proprietе  */
/*                   d'horizontalite.                */
/*      Donnees:                                     */
/*      <V>: vecteur directeur utilise pour tester la */
/*           proprietе; un objet sera horizontal si  */
/*           son vecteur directeur a une ordonnee    */
/*           nulle.                                  */
/*********************************************/
class Horizontal: public Proprietes
{
        Vecteurs  *V;
public: ...
};
```

```
/*********************************************/
/*      Designation: Classe qui decrit la proprietе  */
/*                   de verticalite.                 */
/*      Donnees:                                     */
/*      <V>: vecteur directeur utilise pour tester la */
/*           proprietе; un objet sera vertical si son */
/*           vecteur directeur a une abscisse nulle. */
/*********************************************/
class Vertical: public Proprietes
{
        Vecteurs  *V;
public: ...
};
```

```
/*********************************************/
/*      Designation: Classe qui decrit la proprietе  */
/*                   de coller un objet contre un    */
/*                   autre.                          */
/*      Donnees:                                     */
/*      <p11>, <p12>: deux points de contact de      */
/*                   l'objet qui va etre colle       */
/*                   contre.                         */
/*      <p21>, <p22>: deux points de contact de      */
/*                   l'objet contre lequel un objet  */
/*                   va etre colle.                  */
/*********************************************/
class CollerContre: public Proprietes
{
        Points    *P11, *P12;
        Points    *P21, *P22;
public: ...
};
```

## 2.3.Les comportements.

Chaque comportement donne lieu à la définition d'une classe d'objets particulière. Ainsi, on définit les classes *RExtrinseques*, *RIntrinseques*, *Transmuteurs* et *Producteurs*.

Dans cette implémentation, le réacteur extrinsèque permet de traduire un objet d'application sous forme graphique i.e. il définit un mode d'affichage. Par conséquent, on définit le réacteur extrinsèque par le code caractérisant une fonction d'affichage (cf. figure A.5).

```
/*********************************************/
/*      Designation: Classe qui decrit les reacteurs */
/*                   extrinseques.                   */
/*      Donnees:                                     */
/*      <CodeReacteur>: code permettant de retrou-   */
/*                   ver la fonction de declen-      */
/*                   chement du reacteur.            */
/*      <Nom>: nom identifiant le reacteur.          */
/*********************************************/
class RExtrinseques
{       int       CodeReacteur;
        char*     Nom;
public: ...
};
```

```
/*********************************************/
/*      Designation: Classe qui decrit les reacteurs */
/*                   intrinseques.                   */
/*      Donnees:                                     */
/*      <CodeReacteur>: code connue par la pro-      */
/*           prietе a laquelle le reacteur est       */
/*           associe.                                */
/*      <Nom>: nom identifiant le reacteur.          */
/*      <TabRefRExtrinseques>: ensembles des re-     */
/*           ferences des reacteurs extrinseques     */
/*           utilises pendant l'execution du reac-   */
/*           teur intrinseque.                       */
/*      <NbRefRExtrinseques>: nombre de reac-        */
/*           teurs extrinseques associes au reac-    */
/*           teur intrinseque.                       */
/*      <FonctionParametres>: fonction permettant    */
/*           d'evaluer des parametres du critere     */
/*           de la proprietе utilisant le reacteur.  */
/*********************************************/
class RIntrinseques
{       int       CodeReacteur;
        char      *Nom;
        RExtrinseques  *TabRefRExtrinseques[NbMaxRExtrinseques];
        int       NbRefRExtrinseques;
        void      (*FonctionParametres)(ObjetsApplications*, Proprietes*);
public: ...
};
```

***Figure A.5. Classes définissant les réacteurs.***

Le réacteur intrinsèque traduit un objet en une donnée. Ainsi, sa classe contient comme donnée membre une fonction capable d'effectuer cette traduction. De plus, on y décrit également les liens éventuels avec des réacteurs extrinsèques (cf. figure A.5).

Le transmuteur gère la modification d'un objet entraînant un changement de type. Pour parvenir à une telle gestion, on définit dans la classe *Transmuteurs* les informations essentielles à la modification i.e. la connaissance sur le nouveau type, les conditions de transmutations et le traitement à effectuer pour réaliser la transformation (cf. figure A.6).

```
/*************************************************************************/
/*      Designation: Classe qui decrit les transmuteurs.                 */
/*      Donnees:                                                         */
/*      <Code>: code utilise par les comportements (producteurs ...) pour inclure le */
/*              transmuteur dans leur fonctionnement.                    */
/*      <Nom>: nom du transmuteur.                                       */
/*      <NomTypeTransmute>: nom du type des objets transmutes.           */
/*      <FonctionTesteTransmutation>: fonction contenant la description du critere */
/*                                    de transmutation.                  */
/*      <FonctionCreeObjTransmute>: fonction a appeler pour creer l'objet transmu*/
/*                                    te temporairement dans le cas d'une detec- */
/*                                    tion de transmutation.             */
/*************************************************************************/
class Transmuteurs
{       int     Code;
        char    *Nom;
        char    *NomTypeTransmute;
        Boolean (*FonctionTesteTransmutation)(ObjetsApplications*, ModelesObjetsApplications*);
        ObjetsApplications  *(*FonctionCreeObjTransmute)(ObjetsApplications*, ModelesObjetsApplications*, ModelesObjetsApplications*);
public::...
};
```

***Figure A.6.** Classe définissant le transmuteur.*

```
/*************************************************/
/*      Designation: Classe qui decrit un attribut */
/*                   utilise pour creer un objet geo- */
/*                   metrique.                   */
/*      Donnees:                                 */
/*      <Nom>: nom de l'attribut.                */
/*      <NbType>: nombre de types autorises pour */
/*              l'attribut.                      */
/*      <Type>: tableau dynamique contenant l'en- */
/*              semble des types que peut prendre */
/*              l'attribut.                      */
/*      <Contrainte>: contrainte de validite de l'at- */
/*              tribut.                          */
/*************************************************/
class Attributs
{       char                    *Nom;
        int                     NbType;
        int                     *Type;
        ExpressionsContraintes  *Contrainte;
public: ...
};
```

```
/*****************************************************************/
/*      Designation: Classe qui decrit les producteurs.          */
/*      Donnees:                                                 */
/*      <Nom>: nom du producteur.                                */
/*      <TabAttribut>: ensemble des attributs de creation de l'objet pour */
/*              lequel le producteur est defini.                 */
/*      <TabRefRIntrinseques>: ensemble des references des reacteurs in- */
/*              trinseques a utiliser lors de l'execution du producteur.*/
/*      <NbRefRIntrinseques>: nombre de reacteurs intrinseques associes */
/*              au producteur.                                   */
/*      <TabRefRExtrinseques>: ensemble des references des reacteurs */
/*              extrinseques a utiliser lors de l'execution du produc- */
/*              teur.                                            */
/*      <NbRefRExtrinseques>: nombre de reacteurs extrinseques asso- */
/*              cies au producteur.                              */
/*      <TabRefTransmuteurs>: ensemble des references des transmuteurs*/
/*              a utiliser lors de l'execution du producteur.    */
/*      <NbRefTransmuteurs>: nombre de transmuteurs associes au pro- */
/*              ducteur.                                         */
/*      <CodeMAJ>: code permettant d'identifier la fonction de mise a */
/*              jour de la representation de l'objet.            */
/*      <CodeInitObjet>: code permettant d'identifier la fonction d'initia- */
/*              lisation de la representation de l'objet.        */
/*****************************************************************/
class Producteurs
{       char            *Nom;
        TabAttributs    *TabAttribut;
        RIntrinseques   *TabRefRIntrinseques[NbMaxRIntrinseques];
        int             NbRefRIntrinseques;
        RExtrinseques   *TabRefRExtrinseques[NbMaxRExtrinseques];
        int             NbRefRExtrinseques;
        Transmuteurs    *TabRefTransmuteurs[NbMaxTransmuteurs];
        int             NbRefTransmuteurs;
        int             CodeMAJ, CodeInitObjet;
public: ...
};
```

***Figure A.7.** Classes définissant le producteur et ses attributs.*

Le producteur représente un mode de création d'un objet d'application. Ainsi, la classe Producteur est définie par un ensemble d'attributs de création ainsi que des liens vers d'autres comportements dans le cas où le mode de création intègre des aides (détections et affichages) (cf. figure A.7). Un attribut est représenté par une classe Attribut dans laquelle on définit son domaine de définition.

### 3. UN EXEMPLE TYPE POUR SPÉCIFIER OBJET.

Une fois les différentes classes établies, on peut décrire pour chaque type d'objets d'application ses comportements. Ne disposant pas d'outil graphique de spécification du modèle générique, cette description a été réalisée sous forme de programme. Ainsi, pour montrer son fonctionnement, nous présentons comment spécifier un objet d'application quelconque par ses comportements : nous appelons cet objet *objetTest*. Les classes et primitives données ci-dessous représentent des modèles pour spécifier un objet quelconque. Nous avons pris comme convention de mettre en évidence les informations variables d'un objet à l'autre entre les signes < et >. Par exemple, dans la figure A.8, <NOM_CLASSE_OBJETTEST> signifie qu'à cet endroit on doit donner le nom de la classe décrivant les données du nouveau type d'objet à considérer dans l'application.

```
/*********************************************************/
/*          Designation: Classe qui decrit un objet de l'application. */
/*                       Cette classe doit servir de modele pour    */
/*                       tout nouvel objet.                         */
/*********************************************************/
class <NOM_CLASSE_OBJETTEST>: public ObjetsApplications
{
   <Donnees decrivant l'objet: REPRESENTATION CANONIQUE>
public:    ...
};

Boolean CreerObjetTest(ModelesObjetsApplications *ModeleInstanceParType, ObjetsTests **<OBJETTEST>)
/*****************************************************************************************
Fonction:  cree une instance d'un objet quelconque dans le modele dedie au dialogue et retourne TRUE si succes;
---------- cette instance est creee sans ses comportements; cette fonction doit etre utilisee comme modele; les
           noms entre <..> sont a completer.
Entree/Sortie:
---------------
        <ModeleInstanceParType>: modele dedie au dialogue dans lequel on va creer un nouvel objet
Sortie:
-------
        <<OBJETTEST>>:   instance d'un objet quelconque.
*****************************************************************************************/
{
        (*<OBJETTEST>) = new <NOM_CLASSE_OBJETTEST>(new ObjetsSpecifies());
        return (ModeleInstanceParType->AjouteObjet(*<OBJETTEST>, TRUE));
}
```

***Figure A.8.*** *Classe et instance définissant un objet d'application objetTest.*

Avant de décrire les comportements de *objetTest*, on décrit tout d'abord ses données à travers une classe dérivant d'un objet *ObjetsApplications* (cf. figure A.8). Le modèle générique est constitué d'une instance de chaque classe d'objets. Chacune de ces instances symbolise un type d'objet ; c'est à cette instance que l'on associe des comportements. Le modèle est ainsi dédié au dialogue, car il décrit chaque type d'objets selon l'approche données/comportements. Par conséquent, on définit une instance de *objetTest* afin d'en décrire ses comportements (cf. figure A.8).

Ensuite, on peut définir les comportements de l'objet. Si parmi eux, se trouvent des réacteurs intrinsèques permettant de gérer des propriétés, on a besoin de connaître ces propriétés, car il faut leur

spécifier que des réacteurs supplémentaires peuvent leur apporter désormais des données utiles (cf. figure A.4.). Nous montrons dans la figure A.9 l'allure de la procédure décrivant les comportements d'un objet quelconque.

```
Boolean CreerComportementsObjetTest(ModelesObjetsApplications *, Proprietes *<PROPRIETE>, ObjetsTests *<OBJETTEST>)
/*****************************************************************************************

Fonction: cree l'ensemble des comportements pour un objet de type quelconque; cette specification est definie
          une fois pour l'instance representant l'objet (c'est l'instance definie dans le modele dediee au dialogue);
          puis lors de l'execution de l'application, toute nouvelle instance fera reference a cette specification;
          cette fonction sert de modele a tout nouvel objet; les comportements fournis en exemple sont:
                  - un reacteur extrinseque pour afficher l'objet;
                  - un reacteur extrinseque pour afficher une aide pendant la creation de l'objet;
                  - un reacteur extrinseque pour afficher la propriete de l'objet quand elle est detectee;
                  - un reacteur intrinseque pour detecter une propriete sur l'objet.
                  - un transmuteur pour decrire un changement de type pendant la creation.
                  - un producteur pour decrire la creation de l'objet.
          Toutes les donnees entre <..> sont a remplacer.

Entree:       <<PROPRIETE>>: propriete sur l'objet.

Entree/Sortie: <<OBJETTEST>>: objet auquel on ajoute les comportements.

*****************************************************************************************/
{
        Boolean             succes = TRUE;
        Producteurs         *<PRODUCTEUR_OBJETTEST>;
        RExtrinseques       *<R_AFFICHE>, *<R_AIDE_CREATION>, *<R_AFFICHE_PROPRIETE>;
        RIntrinseques       *<R_PROPRIETE>;
        Transmuteurs        *<TRANSMUTEUR>;
        int                 <CODE_RP>;

création des réacteurs extrinsèques
        <R_AFFICHE> = new RExtrinseques(<NUMERO_TRAITEMENT_ASSOCIE>, <NOM_R_AFFICHE>);
        <R_AIDE_CREATION> = new RExtrinseques(<NUMERO_TRAITEMENT_ASSOCIE>, <NOM_R_AIDE_CREATION>);
        <R_AFFICHE_PROPRIETE> = new RExtrinseques(<NUMERO_TRAITEMENT_ASSOCIE>, <NOM_R_AFFICHE_PROPRIETE>);

création d'un réacteur intrinsèque
        <CODE_RP> = <PROPRIETE>->CodeRIDetection(1);
        <R_PROPRIETE> = new RIntrinseques(<CODE_RP>, <NOM_R_PROPRIETE>, <NOM_FONCTION_CALCUL_PARAMETRE>);

création d'un transmuteur
        <TRANSMUTEUR> = new Transmuteurs( <NOM_TRANSMUTATION>, <NOM_TYPE_OBJET_TRANSMUTE>,
                                          <NUMERO_TRANSMUTATION>, <NOM_FONCTION_TESTE_TRANSMUTATION>,
                                          <NOM_FONCTION_CREE_TRANSMUTATION_TEMPORAIRE>);

association de ces cinq comportements à l'objet
        succes = (Boolean)(  <R_PROPRIETE>->AjouteRExtrinseque(<R_AFFICHE_PROPRIETE>)
                          && <OBJETTEST>->LitOS()->AjouteRExtrinseque(<R_AFFICHE>)
                          && <OBJETTEST>->LitOS()->AjouteRExtrinseque(<R_AIDE_CREATION>)
                          && <OBJETTEST>->LitOS()->AjouteRExtrinseque(<R_AFFICHE_PROPRIETE>)
                          && <OBJETTEST>->LitOS()->AjouteRIntrinseque(<R_PROPRIETE>)
                          && <OBJETTEST>->LitOS()->AjouteTransmuteur(<TRANSMUTEUR>));
        if (succes)
        {
          <PRODUCTEUR_OBJETTEST> = new Producteurs(<NOM_PRODUCTEUR>,
                                                   <NOM_FONCTION_MISE_A_JOUR_REPRESENTATION>,
                                                   <NOM_FONCTION_INITIALISATION_REPRESENTATION>);
          succes = (Boolean)(<PRODUCTEUR_OBJETTEST>->AjouteAttribut(
                          <NOM_ATTRIBUT_1>,
                          "N_<N°_FONCTION>_<NOM_FONCTION_CONTRAINTE>(<ATTRIBUTS_CONCERNES>)",
                          ET,
                          <NOMBRE_TYPES_AUTORISES_POUR_ATTRIBUTS>,
                          <N°_TYPE_1>,
création d'un producteur
                          ...,
                          <N°_DERNIER_TYPE>)
                       && <PRODUCTEUR_OBJETTEST>->AjouteAttribut(
                          <NOM_ATTRIBUT_2>,
                          "N_<N°_FONCTION>_<NOM_FONCTION_CONTRAINTE>(<ATTRIBUTS_CONCERNES>)",
                          <ORDRE_ATTRIBUT>,
                          <NOMBRE_TYPES_AUTORISES>,
                          <NUMERO_TYPE_1>,
                          ...,
                          <NUMERO_DERNIER_TYPE>)
associations: de comportements au producteur; du producteur à l'objet
                       && <PRODUCTEUR_OBJETTEST>->AjouteRExtrinseque(<R_AIDE_CREATION>)
                       && <PRODUCTEUR_OBJETTEST>->AjouteRIntrinseque(<R_PROPRIETE>)
                       && <PRODUCTEUR_OBJETTEST>->AjouteTransmuteur(<TRANSMUTEUR>)
                       && <OBJETTEST>->LitOS()->AjouteProducteur(<PRODUCTEUR_OBJETTEST>));
        }
        return succes;
}
```

***Figure A.9.*** *Description des comportements d'un nouvel objet.*

## 4. LA GÉNÉRATION DU DIALOGUE.

Lorsque le modèle générique est spécifié, on peut déduire le dialogue. La déduction se résume à la création de fichiers interprétés de description du dialogue qui seront utilisés au moment de l'exécution de l'application pour créer les données propres au dialogue.

```
/*********************************************************************/
/*   Designation: Classe qui decrit le processus de generation du dialogue a partir */
/*                du modele dedie au dialogue.                         */
/*   Donnees:                                                          */
/*   <ModeleRepresentants>: modele contenant un representant de chaque objet  */
/*        manipule dans l'application avec leurs comportements.        */
/*   <ModeleProprietes>: modele contenant un representant de chaque propriete  */
/*        traitee dans l'application.                                  */
/*   <FInterface>: fichier texte contenant la description complete de l'interface de */
/*        l'application.                                               */
/*   <FCompatibilite>: fichier texte des compatibilites locales sur les interactions.*/
/*********************************************************************/
class Generations
{       ModelesObjetsApplications    *ModeleRepresentants;
        ModelesProprietes            *ModeleProprietes;
        FILE                         *FInterface;
        FILE                         *FCompatibilite;
public: ...
};
```

*Figure A.10. Classe définissant le processus de génération du dialogue.*

La génération est perçue dans l'implémentation comme une instance d'une classe *Generations*. La raison fondamentale de ce choix provient du fait que la génération utilise des données propres et différentes primitives de déduction : elle constitue un composant à part entière dans l'implémentation. La figure A.10 montre la classe *Generations* ainsi décrite.

Nous décrivons tout d'abord les primitives mises en place pour déduire le dialogue. Ensuite, nous montrons des fichiers de description du dialogue qui ont été déduits.

### 4.1.Les algorithmes.

La génération du dialogue est associée au succès de la description du modèle générique. Ainsi, on crée les modèles de données sur les objets et les propriétés, puis on crée les comportements associés à chaque objet d'application ; la génération du dialogue peut ensuite s'effectuer. Nous avons ainsi défini la primitive *DecrireDialogue* :

```
Boolean DecrireDialogue()
/*********************************************************************************************
Fonction:  generation du dialogue associe a l'application suite a la description des elements manipules dans
----------  l'application et retourne TRUE si succes.
*********************************************************************************************/{
        Generations         *Generation;
        Boolean             succes = FALSE;

        ModeleProp = CreerModelePropriete();
        ModeleInstanceParType = CreerModeleObjetApplication();
        succes = (CreerProprietes() && CreerObjets() && CreerComportementsObjets());
        if (succes)
        {       Generation = new Generations(ModeleInstanceParType, ModeleProp);
                Generation->ActiveGeneration();
        }
        return succes;
}
```

La primitive *ActiveGeneration* représente la primitive principale de génération du dialogue :

```
void Generations::ActiveGeneration()
/*****************************************************************************
Fonction:  genere le dialogue de l'application; parcourt le modele des representants de chaque
---------  objet manipule par l'application; pour chaque objet:
              - on cree dans le fichier de specification des menus les menus de creation de
                l'objet (un menu par producteur).
              - on cree le fichier decrivant l'action globale liee a la creation

*****************************************************************************/
{
        ObjetsApplications   *Objet;
        ObjetsSpecifies      *ObjetS;
        int                  nbProducteurs, i;
        Producteurs          *Producteur;
        char                 nompere[NbMaxChaineFichier],
                             NomActionGlobale[NbMaxChaineFichier],
                             ch[NbMaxChaineFichier];




        if ((OuvrirFichier("menu", "acr", FInterface)) && (OuvrirFichier("compatibilite", "acr", FCompatibilite)))
        {
                CreerEnteteFInterface();
                CreerGroupeGeneralFInterface();
                CreerGroupeApplicationFInterface();
                CreerEnteteFCompatibilite();
                ModeleRepresentants->PasseAuPremier();
                while (!ModeleRepresentants->ToutParcouru())
                {
                        Objet = ModeleRepresentants->LitObjet();
                        ObjetS = Objet->LitOS();
                        nbProducteurs = ObjetS->LitNbProducteurs();
                        if (nbProducteurs != 0)
                        {
                                AjouteElement(      "OPTION MENU", Objet->LitNomType(), "Creation",
                                                    "VIDE", "VIDE", FInterface);
   on définit une compatibilité locale entre
    le menu de création d'un type d'objet et {AjouteCompatibilite(Objet->LitNomType(), Objet->NumType());
toute interaction demandant ce type d'objet  strcpy(nompere, Objet->LitNomType());
                                for (i=0;i<nbProducteurs;i++)
                                {
                                    Producteur = ObjetS->LitProducteur(i);
                                    GenereActionGlobaleCreation(Objet, Producteur);
                                    sprintf(    NomActionGlobale,"%s_%s", nompere,
                                                Producteur->LitNom());
                                    RetireEspace(NomActionGlobale, ch);
                                    sprintf(NomActionGlobale,"\"%s\"", ch);
                                    AjouteElement(      "OPTION MENU", Producteur->LitNom(),
                                                        nompere, NomActionGlobale, "VIDE",
                                                        FInterface);
                                }
                        }
                        ModeleRepresentants->PasseAuSuivant();
                }
                CreerFenetres();
                CreerZonesSaisie();
                CreerCompatibilitesMenus();
                AjouteCompatibilite("Coord OK", TypeCoordonnees2D);
                AjouteCompatibilite("Entier OK", TypeEntiers);
                AjouteCompatibilite("Chaine OK", TypeChaines);
                FinCompatibilite();
                FermerFichier(FCompatibilite);
                FermerFichier(FInterface);
        }
}
```

*on définit le fichier de description d'un mode de création pour l'objet Objet en analysant l'un de ses producteurs*

*on ajoute un menu associé au mode de création ainsi décrit*

Chaque producteur est analysé pour donner lieu à un fichier de description du dialogue d'un mode de création de l'objet auquel il est associé. On analyse les différents attributs caractérisant le producteur afin de définir les séquences d'interactions correspondantes. On associe à chaque interaction déduite

des effets particuliers déduits des comportements associés au producteur : affichages en élastique, détections de contraintes, et détection de transmutation. Il s'agit de la procédure *AjouteEffet* :

```
void Generations::AjouteEffets(Producteurs *Prod, int NumInter, int &CompteurNumElt, FILE *&F)
/*****************************************************************************
Fonction: ajoute dans le fichier de description d'une action globale pour une interaction ses effets.
Entree:
        <Prod>:       producteur de l'objet.
        <NumInter>: numero de l'interaction a laquelle on ajoute des effets.
Entree/Sortie:
        <CompteurNumElt>: numero du prochain element a traiter dans le fichier.
        <F>:              fichier de description de l'action globale.
*****************************************************************************/
{
        int             i;
        Transmuteurs    *T;
        RExtrinseques   *RE;
        RIntrinseques   *RI;
        Proprietes      *P;

                for (i=0;i<Prod->LitNbRIntrinseques();i++)
                {
                        RI = Prod->LitRIntrinseques(i);
                        if (RI->EstRIDeclenchement(ModeleProprietes, P))
                        {
                                fprintf(F, "COMPORTEMENT : (%i)     \"%s\"     CODE : %i\tPARAMETRE : 2\tTYPE : %i VERS (%i) ;\n",
                                        CompteurNumElt, RI->LitNom(), P->LitCodePropriete(), TypeCoordonnees2D, NumInter);
                                AjouteFlot(1, CompteurNumElt, F);
                                CompteurNumElt++;
                        }
                }
                for (i=0;i<Prod->LitNbTransmuteurs();i++)
                {
                        T = Prod->LitTransmuteurs(i);
                        fprintf(F, "COMPORTEMENT : (%i)     \"%s\"     CODE : %i\tPARAMETRE : 2\tTYPE : %i VERS (%i) ;\n",
                                CompteurNumElt, T->LitNom(), T->LitCode(), TypeCoordonnees2D, NumInter);
                        AjouteFlot(1, CompteurNumElt, F);
                        CompteurNumElt++;
                }
                for (i=0;i<Prod->LitNbRExtrinseques();i++)
                {
                        RE = Prod->LitRExtrinseques(i);
                        fprintf(F, "COMPORTEMENT : (%i)     \"%s\"     CODE : %i\tPARAMETRE : 2\tTYPE : %i VERS (%i) ;\n",
                                CompteurNumElt, RE->LitNom(), RE->LitCode(), TypeCoordonnees2D, NumInter);
                        AjouteFlot(1, CompteurNumElt, F);
                        CompteurNumElt++;
                }
}
```

*on ajoute des détections de propriétés*

*on ajoute des détections de transmutation*

*on ajoute des affichages en élastique*

## 4.2.Un exemple de fichiers déduits.

L'analyse des comportements entraîne la création de fichiers interprétés décrivant des éléments du modèle de dialogue. Prenons l'exemple de l'objet d'application paramètre (classe Objets*Parametres*). Il a été défini avec les comportements suivants :

- un réacteur extrinsèque pour afficher l'objet ;
- un réacteur extrinsèque pour afficher l'objet en élastique pendant sa création ;
- un réacteur intrinsèque pour traduire l'objet paramètre en une donnée permettant de détecter la pose de cet objet contre un objet action non interactive ou paramètre ;
- un réacteur intrinsèque pour traduire l'objet paramètre en une donnée permettant de détecter la pose d'un objet paramètre contre cet objet.

- un producteur pour décrire la création d'un paramètre à partir de sa position dans la fenêtre graphique et de son nom. Les trois derniers réacteurs sont associés au producteur pour définir ces comportements sur l'objet pendant sa création.

L'analyse des comportements donne lieu à la déduction d'un menu de création propre à l'objet paramètre. Cette déduction apparaît dans le fichier contenant la description de l'ensemble des menus de l'application par l'ajout des lignes suivantes :

*menu de création d'un objet paramètre*

```
OPTION MENU     : "Parametre"       FILS DE "Creation" [VIDE][VIDE];
OPTION MENU     : "Position/Nom"    FILS DE "Parametre" ["Parametre_PositionNom"][VIDE];
```

*menu proposant un mode de création particulier*

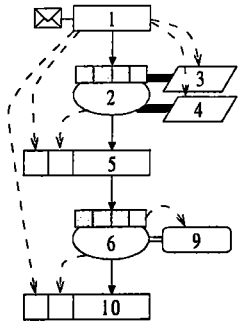
Ensuite, une compatibilité locale est déduite sur le menu de création général (i.e. n'importe quel mode de création) de l'objet paramètre pour toute interaction demandant un objet paramètre. Cette déduction apparaît dans le fichier des compatibilités par l'ajout de la ligne :

```
COMPATIBILITE : INTERFACE "Parametre" EST Local DE INTERACTION 18 ;
```

Enfin, chaque producteur donne lieu à la création d'un fichier de description du dialogue d'un mode de création. Pour l'objet paramètre, il n'y a qu'un seul producteur qui permet de déduire la création d'un paramètre par sa position et son nom.

*Dialogue de création d'un objet paramètre:* ☐

*dialogue décrit*




*fichier interprété du dialogue déduit des comportements*

```
NOMBRE DE PARAMETRE : 0;

ANIAR : (1) "Initialisation representation" CODE : 53 PARAMETRE : 0 ;

INTERACTION :(2) "Position" TYPE : 2;
EFFET : (3) "RCollerContre" CODE : 68 PARAMETRE : 2 TYPE : 2 VERS (2) ;
FLOT DE DONNEE : (1) VERS (3) ;
EFFET : (4) "Affiche parametre en elastique" CODE : 25 PARAMETRE : 2 TYPE : 2 VERS (2) ;
FLOT DE DONNEE : (1) VERS (4) ;
ANI : (5) "Mise a jour representation" CODE : 36 PARAMETRE : 2 ;
FLOT DE DONNEE : (1) VERS (5) ;
SEQUENCE : (2) VERS (5) ;
FLOT DE DONNEE : (2) VERS (5) ;
SEQUENCE : (1) VERS (2) ;

INTERACTION :(6) "Nom" TYPE : 8;
EFFET : (7) "RCollerContre" CODE : 68 PARAMETRE : 2 TYPE : 2 VERS (6) ;
FLOT DE DONNEE : (1) VERS (7) ;
EFFET : (8) "Affiche parametre en elastique" CODE : 25 PARAMETRE : 2 TYPE : 2 VERS (6) ;
FLOT DE DONNEE : (1) VERS (8) ;
CONTRAINTE : (9) CODE : 101 PARAMETRE : 1 VERS (6) ;
FLOT DE DONNEE : (6) VERS (9) ;
ANI : (10) "Mise a jour representation" CODE : 36 PARAMETRE : 2 ;
FLOT DE DONNEE : (1) VERS (10) ;
SEQUENCE : (9) VERS (10) ;
FLOT DE DONNEE : (6) VERS (10) ;
SEQUENCE : (5) VERS (6) ;

DEPART : (1);
RESULTAT : (1);
BOUCLAGE : FAUX;
```

*les effets sont déduits pour chaque interaction; mais ne sont effectués que si la représentation de l'objet le permet, ce qui n'est pas le cas ici*

***Figure A.11.*** *Dialogue de création d'un objet paramètre.*

Dans la figure A.11, nous montrons le fichier de description du dialogue qui a été déduit de l'analyse des comportements, ainsi que la description du dialogue par le formalisme graphique de SACADO. Sans la description de l'objet paramètre par ses comportements, il aurait fallu décrire le dialogue à l'aide du formalisme graphique, ce qui demande une certaine pratique du modèle de dialogue établi. De plus, notons que le résultat montré par le formalisme n'est pas complet pour ne pas surcharger la figure (il manque par exemple, les noms des éléments, les types et les compatibilités avec les menus).

## 5. CONCLUSION.

Nous avons montré dans le chapitre 2 les résultats de la génération du dialogue en montrant un exemple de dialogue pendant l'utilisation de l'application (cf. Chapitre 2 figure II.17) : le dialogue déduit est plus que satisfaisant. La méthode utilisée est d'autant plus efficace que les comportements fournissent des bases solides à partir desquelles on arrive facilement à déduire le modèle de dialogue. Les algorithmes de génération ont été réalisés de manière relativement automatique. En outre, la spécification du modèle générique reste simple bien que l'on ait à regretter un outil convivial de spécification des comportements.

# ANNEXE B.

# IMPLÉMENTATION DES MANIPULATEURS ET DES OPÉRATEURS

## 1. INTRODUCTION.

Dans cette annexe, nous présentons les points essentiels sur la mise en œuvre de l'implémentation évoquée dans le chapitre 4 (cf. Chapitre 4 5. Implémentation). Son rôle est de valider le concept des manipulateurs et la modélisation des opérateurs.

L'application que nous avons choisie de réaliser concerne la conception d'un parcours de santé. On suppose que l'on dispose initialement d'une aire dans laquelle se trouve un sentier (cf. figure A.1). Ensuite, il s'agit de construire des ateliers pour définir des exercices sportifs, puis préciser ces ateliers en terme de fabrication et enfin, tester la faisabilité des exercices proposés. Ainsi, l'application demande l'intervention de différents types d'opérateurs : des concepteurs, des fabricants et des clients. On définit un modèle d'opérateurs pour gérer les différents intervenants. De plus, pour concevoir ce parcours de santé, on définit différents manipulateurs afin de proposer aux opérateurs des dialogues adaptés.

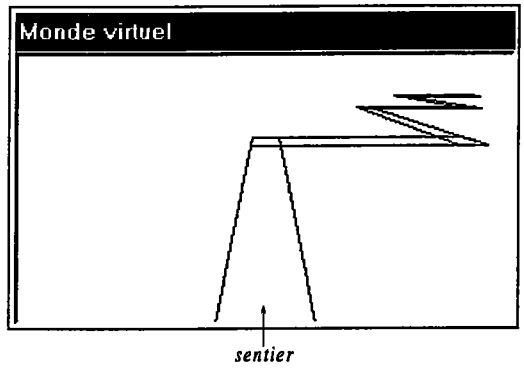



***Figure B.1.** Parcours de santé initial.*

Dans un premier temps, nous présentons les différents objets (opérateurs ou non) composants l'application et nous montrons comment spécifier chaque objet en utilisant la description du modèle générique i.e. par un couple données/comportements. Ensuite, nous décrivons les manipulateurs qui interviennent en insistant sur la capacité à regrouper en un seul concept une grande diversité d'outils. Enfin, nous montrons comment s'effectue la gestion du dialogue en utilisant les manipulateurs comme élément-clé pour déterminer la cohérence de chaque opérateur dans ses dialogues.

L'application a été développée selon une approche objet en Visual C++ sur PC sous l'environnement Microsoft® Windows® 95. Elle utilise un LGB (Logiciel Graphique de Base) développé au laboratoire par Jean-Paul ROCHÉ dans le cadre d'une thèse d'ingénieur CNAM [ROC 97].

## 2. LA SPÉCIFICATION DES OBJETS PAR LES COMPORTEMENTS.

On distingue trois classes principales d'objets dans l'application : les objets d'application, les opérateurs et les propriétés. Nous n'avons considéré de modèles que pour les deux premières classes. En effet, aucun modèle n'est défini pour les propriétés, car il se trouve que l'on ne définit qu'une seule propriété : la faisabilité d'un exercice pour le sportif. De plus, cette dernière est utilisée par les mécanismes du manipulateur gérant la détection de la communication.

Nous montrons dans les deux parties suivantes comment les objets peuvent être spécifiés à l'aide des comportements. Il ne s'agit pas dans cette implémentation de déduire le dialogue de l'application à partir d'une telle spécification. Nous avons toutefois voulu montrer que l'on pouvait l'envisager. L'unique propriété gérée possède également sa propre spécification par les comportements. Nous donnons sa description dans la seconde partie.

### 2.1.Les objets d'application.

Le modèle des objets d'application est composé d'un sentier et de différents ateliers. On dispose de trois types d'ateliers parmi lesquels le concepteur choisit celui qu'il désire construire. La figure B.2 montre trois ateliers créés et qui se basent chacun sur l'un des trois types.

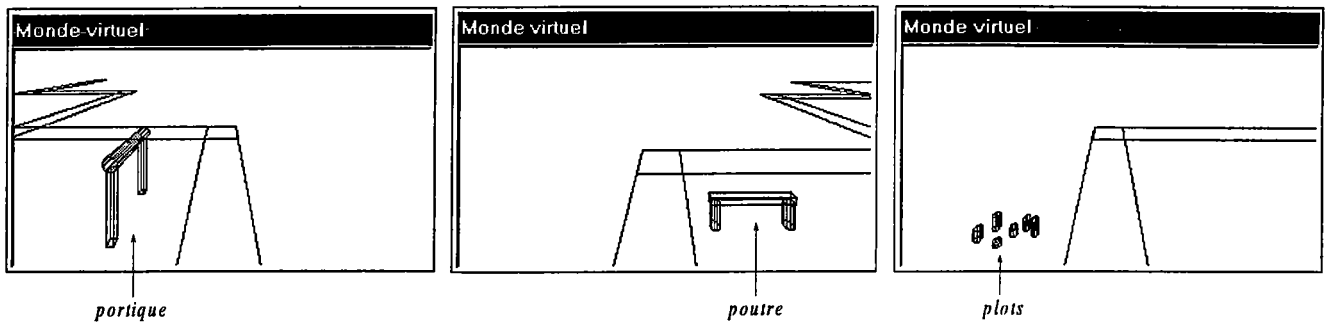



***Figure B.2.** Trois types d'ateliers.*

On peut spécifier un atelier par sa représentation canonique et un ensemble de comportements. Dans notre application, la représentation est la même quel que soit le type d'ateliers. Elle comprend :

- un ensemble d'objets primitifs (cylindres et parallélépipèdes) composant la construction sportive (portique, poutre ou plot) ;
- un ensemble d'expressions grapho-numériques définissant des contraintes entre l'atelier et le sportif. Ces contraintes sont données par un concepteur et représentent la faisabilité de l'exercice en fonction du type de sportif ;
- la position de l'atelier sur le parcours. Un atelier ne sera jamais construit sur le sentier ;
- un identifiant ;
- un message, qui contient l'explication de l'exercice sportif associé à l'atelier.

On ne définit qu'un seul producteur pour un atelier, noté $P_{atelier}$, car l'application ne propose qu'une façon de créer un atelier. Les attributs caractérisant $P_{atelier}$ sont :

- la position de l'atelier, donnée par identification de son emplacement sur un plan du parcours ;
- un modèle d'atelier (portique, poutre ou plots) ;
- un message expliquant l'exercice.

Il n'y a pas d'expressions grapho-numériques à la création d'un atelier et l'identifiant est donné.

Un ensemble de réacteurs extrinsèques $REi_{atelier}$ sont nécessaires pour traduire l'atelier à l'environnement animé. On définit ainsi :

- $RE1_{atelier}$, pour traduire l'atelier afin de le visualiser. Il s'agit de déclencher les réacteurs extrinsèques des objets primitifs constituant l'atelier et spécifiant leur mode d'affichage ;
- $RE2_{atelier}$, pour traduire l'atelier afin de le visualiser en élastique sur le parcours ;
- $RE3_{atelier}$, pour traduire l'atelier sur le plan du parcours. Dans l'application, il s'agit de définir une fonction d'affichage qui symbolise l'atelier par un losange ;
- $RE4_{atelier}$, pour traduire l'atelier par ses contraintes et l'explication de l'exercice. On associe à cette donnée un manipulateur appelé "Décodeur" pour fournir des informations sur l'atelier ;
- $RE5_{atelier}$, pour traduire le matériau de l'atelier. On associe à cette donnée un manipulateur appelé "Matériau" pour définir interactivement le matériau à utiliser ;
- $RE6_{portique}$, pour traduire la hauteur de l'atelier. On associe à cette donnée un manipulateur appelé "Manette" pour agir interactivement sur la hauteur. Ce réacteur (et donc aussi ce manipulateur) n'a été défini que pour les ateliers de type portique.

Enfin, on définit un réacteur intrinsèque $RI1_{atelier}$ pour traduire un atelier par ses ressources nécessaires et effectives pour tester la propriété de faisabilité d'un exercice.

### 2.2.Les opérateurs.

On définit cinq thèmes dans l'application : créer un atelier, adapter un atelier, se déplacer, communiquer, et s'informer. Chaque opérateur possède une combinaison de ces thèmes afin de définir ses buts. On décrit trois classes d'opérateurs : les concepteurs, les fabricants et les clients. Les thèmes sont répartis de la façon suivante :

- pour le concepteur : créer atelier, adapter atelier, se déplacer, communiquer.
- pour le fabricant : adapter atelier, se déplacer ;
- pour le client : s'informer, se déplacer, communiquer.

Bien qu'ils possèdent des données communes telles que les thèmes (cf. figure B.3), les opérateurs se distinguent par d'autres données et par leurs comportements.

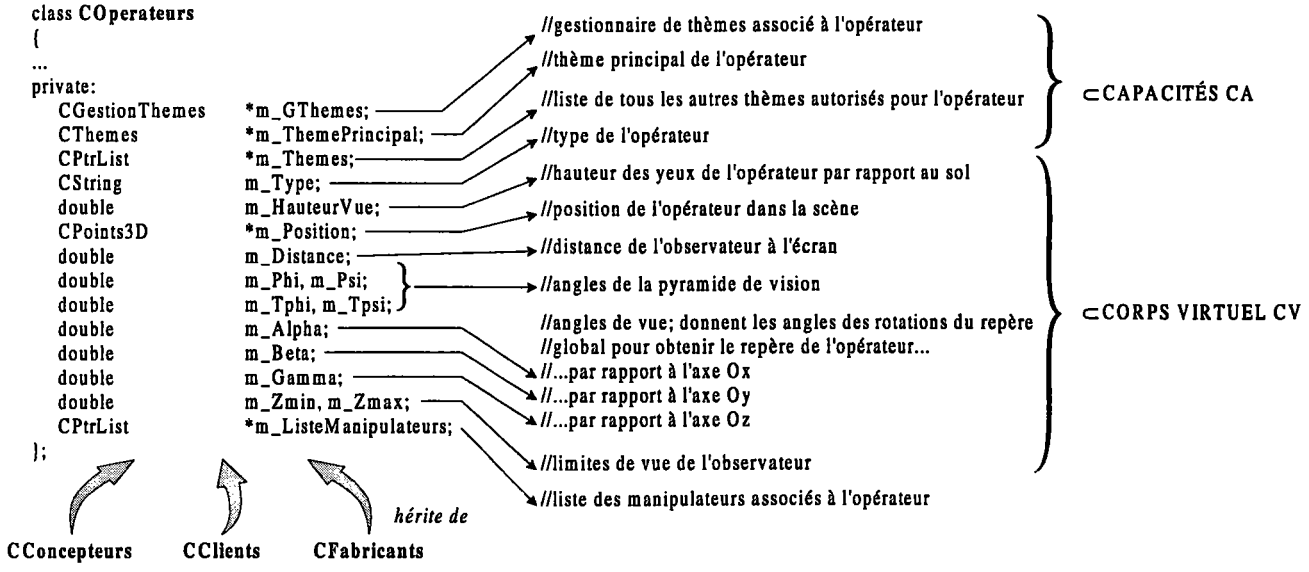



***Figure B.3.** Données communes aux opérateurs.*

Le concepteur ne possède pas de données propres i.e. définies dans la classe CConcepteurs. Il ne possède qu'un seul producteur, $P_{concepteur}$, car on n'envisage qu'un mode de création. Lorsqu'un concepteur utilise l'application, il doit s'identifier pour permettre de le distinguer des autres concepteurs. Tous les autres paramètres sont donnés par défaut. On ne définit aucun réacteur intrinsèque, ce qui n'est pas le cas des réacteurs extrinsèques. Le concepteur possède l'ensemble de réacteurs $REi_{concepteur}$ suivant :

- $RE1_{concepteur}$, pour traduire la représentation virtuelle du concepteur sur le parcours pour les autres utilisateurs ;
- $RE2_{concepteur}$, pour traduire sa position. On construit autour de cette donnée deux manipulateurs, "Plan Concepteur" qui fournit un plan 2D du parcours pour se repérer et placer des ateliers et "Fenêtre 3D" qui gère le monde virtuel que l'opérateur peut voir de sa position ;
- $RE3_{concepteur}$, pour traduire la capacité du concepteur à créer un atelier. Un manipulateur appelé "Bibliothèque" est associé à cette donnée pour permettre au concepteur de créer interactivement des ateliers grâce à une bibliothèque en ligne ;
- $RE4_{concepteur}$, pour traduire la capacité de l'opérateur à définir des contraintes de faisabilité des ateliers. Un manipulateur appelé "Calculatrice" est construit autour de cette donnée pour fournir au concepteur un outil de saisie de contraintes sous forme d'expressions grapho-numériques ;
- $RE5_{concepteur}$, pour traduire la capacité de l'opérateur à expliquer des exercices. Un manipulateur appelé "Notice" est construit autour de cette donnée pour fournir au concepteur un outil de saisie d'explications.

Le concepteur a la possibilité de modifier la hauteur des portiques créés. En d'autres termes, il est capable d'utiliser les manettes de modification des hauteurs associés à ces ateliers. Il fait donc partie du domaine d'utilisation du manipulateur "Manette".

Quant au fabricant, il possède un unique producteur, $P_{fabricant}$, qui définit son intervention dans le monde virtuel. En effet, comme il n'a pas de données particulières (non décrites dans la classe COperateurs), on ne lui associe qu'un seul producteur pour s'identifier. De plus, on ne lui associe que deux réacteurs extrinsèques :

- $RE1_{fabricant}$, pour la représentation virtuelle du fabricant sur le parcours pour les autres utilisateurs ;
- $RE2_{fabricant}$, pour traduire la position de l'opérateur. On construit autour de cette donnée deux manipulateurs, "Plan Fabricant" permettant au fabricant de se déplacer d'atelier en atelier et "Fenêtre 3D" qui gère le monde virtuel que l'opérateur peut voir de sa position. Le manipulateur "Plan Fabricant " n'est pas identique au manipulateur "Plan Concepteur", car il ne propose pas les mêmes fonctionnalités. Par exemple, le fabricant ne peut disposer de nouveaux ateliers à l'aide de son plan.

La particularité de cet opérateur réside dans la modification des ateliers. Contrairement au concepteur, il est capable de modifier le matériau utilisé pour construire les ateliers. Ainsi, il appartient au domaine d'utilisation du manipulateur "Matériau".

Le client diffère des deux autres types d'opérateurs non seulement par ses comportements, mais aussi par ses données. En effet, sa représentation inclut deux informations supplémentaires : sa taille et son niveau de sportivité (débutant ou confirmé). Ces données sont utilisées pour tester la faisabilité des exercices proposés par les ateliers. On définit le producteur, $P_{client}$ qui permet de modéliser un client grâce à la donnée d'identifiant, sa hauteur et son niveau de sportivité.

Deux réacteurs extrinsèques sont associés au client :

- $RE1_{client}$, pour traduire la représentation virtuelle du client sur le parcours pour les autres utilisateurs ;
- $RE2_{client}$, pour traduire la capacité du client à communiquer. On associe à cette donnée le manipulateur "M-Com" qui gère la communication du client en proposant une détection de la communication quand elle est nécessaire (i.e. quand le client se trouve face à un atelier dont il ne peut effectuer l'exercice) ;
- $RE3_{client}$, pour traduire la position de l'opérateur. On construit autour de cette donnée deux manipulateurs, "Plan Client" permettant au client de se déplacer d'atelier en atelier et "Fenêtre 3D" qui gère le monde virtuel que l'opérateur peut voir de sa position. Le manipulateur "Plan de Client " n'est pas identique au manipulateur "Plan Concepteur", car il ne propose pas les mêmes fonctionnalités. Par contre, c'est un manipulateur similaire au "Plan Fabricant".

Un réacteur intrinsèque $RI1_{client}$ est également associé au client. Il permet de traduire le client par son niveau de compétence (taille, niveau de sportivité) afin de tester la propriété de faisabilité de l'exercice proposé par un atelier. On introduit en effet, une telle propriété dans l'application. Elle est utilisée lors de la détection de la communication entre le client et le concepteur que nous avons décrite au chapitre 4 (cf. Chapitre 4 5.2. Spécification).

Pour créer cette propriété, on définit un producteur. Le producteur permet de définir les données nécessaires pour gérer cette propriété. Ces données proviennent des objets concernés par la propriété. Ici, la propriété est binaire : elle a besoin du client et de l'atelier. Le producteur est donc défini par le réacteur intrinsèque du client (qui donne son niveau et sa taille) et celui de l'atelier (qui donne les contraintes de faisabilité sous forme d'expressions grapho-numériques). La propriété est gérée en évaluant les expressions avec les données du client et des ateliers.

En définitive, on parvient à décrire le modèle des opérateurs en utilisant l'approche par des couples données/comportements. Outre le fait que l'on peut envisager une déduction du dialogue, on montre également l'intérêt de modéliser les opérateurs. En effet, comme chaque opérateur possède des comportements différents, le modéliser permet de mieux le considérer dans la gestion du dialogue.

### 3. LES MANIPULATEURS.

L'application choisie nous a permis de définir une grande variété de manipulateurs qui reposent sur une définition commune décrite par la classe CManipulateurs (cf. figure B.4).

```
class CManipulateurs: public CObjetCommunication
{ ...
private:
        //compatibilité du manipulateur a un instant donné
        TypeCompatibilites   m_Compatibilite;

        //informations capturées par le manipulateur
        CInformations        *m_Informations;

        //nom du thème dans lequel le manipulateur permet d'entrer suite à son
        //utilisation
        TypeNomTheme         m_NomNouveauTheme;

        //TRUE si le manipulateur est actif
        BOOL                 m_Actif;

        //entité animé (opérateur) utilisant le manipulateur à un instant donné
        // (NULL sinon)
        COperateurs          *m_Utilisateur;

        //vrai si les événements associés au manipulateur ont changé
        BOOL                 m_ChangementEvenement;

        //TRUE si l'apparence correspondant au mode d'attente de déclenchement
        //est activée
        BOOL                 m_ModeAttente;

        //TRUE si l'apparence activée par le mode d'attente est activée
        BOOL                 m_ApparenceActive;

        //numéro de l'apparence dans le modèle local des manipulateurs si le
        //manipulateur définit son apparence dans le modèle local; sinon -1
        int                  m_NumApparence;

        //Gestion du mode d'attente d'un manipulateur ...
        //...les boutons de la barre d'attente
        CPtrList             *m_ListeOutilsAttentes;
        //...numéro de l'image dans le fichier des images des boutons
        int                  m_PositionImageAttente;

        //vraie si le manipulateur a modifié la scène (objets ou opérateurs) et est
        //utilisé pour savoir si l'utilisation de ce manipulateur demande une mise à
        //jour des autres manipulateurs (en particulier du manipulateur fenêtre3D)
        BOOL                 m_SceneModifiee;
};
```

Données utilisées par les mécanismes communs à tous les manipulateurs

Données communes à plusieurs manipulateurs sur les Apparences

*hérite de*

```
class CPlanConcepteurs: public CManipulateurs
{ ...
private:
      //APPARENCES...
      //...fenetre 2D activée quand le manipulateur
      //est déclenché
      IDENT        m_Fenetre;
      IDENT        m_Espace;
      IDENT        m_Cloture;

      //...structure de scène de la fenêtre
      IDENT        m_Scene2D;

      //correspondance entre les figures de la
      //structure de scène et les faces des objets
      //du modèle
      CPtrArray    *m_TabFiguresFaces;
};
```

Données pour l'apparences propre au PlanConcepteur:

```
class CBibliotheques: public CManipulateurs
{ ...
private:
      //APPARENCES...
      int              m_BoutonRadioPortique;
      int              m_BoutonRadioPoutre;
      int              m_BoutonRadioPlot;
      CBoiteDialogue   *m_BoiteDialogue;
      int              m_XBoite, m_YBoite;
};
```

Données pour l'apparence propre à la Bibliothèque

***Figure B.4.*** *Données de la classe CManipulateurs, CPlanConcepteurs et CBibliotheques.*

Parmi les manipulateurs implémentés, on peut citer les plus pertinents :

- "Fenêtre 3D", qui représente un manipulateur général associé au modèle des opérateurs. Il appartient à la classe PROF-ACT, car il se base sur la position de l'opérateur pour fonctionner et est utilisé par cet opérateur. Ce manipulateur nous permet de mettre en évidence le fonctionnement d'un manipulateur général ainsi que de montrer la possibilité de définir un

manipulateur sur un ensemble d'objets. Chaque opérateur a à sa disposition un tel manipulateur, puisque ce dernier est associé au modèle des opérateurs. Dès qu'une désignation s'effectue dans son apparence, "Fenêtre 3D" traite cette information et active le manipulateur qui en a besoin à l'instant donné ;

- "Plan Concepteur", qui est défini pour le concepteur. Il appartient à la classe PROF-ACT, car il se base sur la représentation du concepteur et est utilisé par ce dernier. Ses mécanismes consistent à afficher sur un plan 2D le parcours de santé ainsi que la position du concepteur, et à gérer les désignations effectuées sur ce plan. Certains mécanismes déclenchent les réacteurs extrinsèques des ateliers et du concepteur qui permettent de les afficher de manière symbolique en 2D ; d'autres permettent au concepteur de désigner un emplacement pour un nouvel atelier, de désigner un atelier déjà créé et de s'y rendre ou encore de positionner le concepteur à un endroit sur le sentier ;
- "Manette", qui est défini pour un atelier portique. Il appartient à la classe PROF-PAS, car il repose sur la hauteur du portique et est utilisé par le concepteur. Lorsque la manette (apparence du manipulateur) est activée, le manipulateur va automatiquement modifier la hauteur du portique et utiliser le réacteur extrinsèque $RE2_{atelier}$ de cet objet pour l'afficher en mode élastique ;
- "Bibliothèque", qui est associée au concepteur. Il appartient à la classe SURF-ACT, car il se base sur la capacité de création d'atelier d'un concepteur et est utilisé par ce dernier.

On montre dans les figures B.4 et B.5 les apparences caractérisant les manipulateurs "Plan Concepteur" et "Bibliothèque" associé au concepteur. Ils ont plusieurs apparences : une pour indiquer au concepteur que l'outil géré par le manipulateur est disponible (mode d'attente) et une pendant l'utilisation de l'outil.

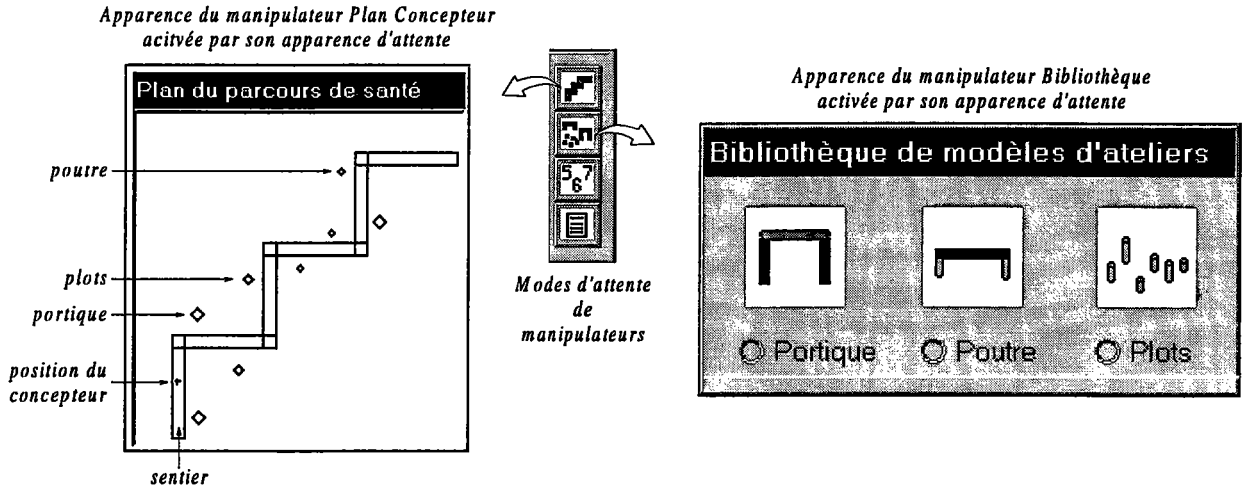



***Figure B.5.** Apparences des manipulateurs Plan Concepteur et Bibliothèque.*

## 4. LA GESTION DU DIALOGUE.

Les manipulateurs contrôlent toutes les interactions des opérateurs, puisqu'ils symbolisent les outils de manipulations disponibles dans l'application. Nous avons mis en place les bases de développement du contrôle de cohérence du dialogue d'un opérateur à partir de ses thèmes et des compatibilités des manipulateurs. Nous montrons dans un premier temps à titre d'exemple les compatibilités définies sur

les manipulateurs "Plan Concepteur" et "Bibliothèque" pour les thèmes du concepteur. Ensuite, nous détaillons le processus de gestion du dialogue à travers les primitives mises en œuvre.

### 4.1.Les compatibilités des manipulateurs.

Chaque manipulateur possède des compatibilités particulières selon son l'utilisation i.e. les informations qu'il a capturées et le thème abordé par l'opérateur. La figure B.6 montre les primitives d'évaluation de la compatibilité du manipulateur plan défini pour le concepteur. La première primitive *CalculeCompatibiliteCreerAtelier()* reprend la description des compatibilités du plan pour le thème de création d'atelier, donnée dans le chapitre 3 (cf. Chapitre 3 *4.4.2. Les thèmes et les compatibilités*).

```
//évaluation de la compatiblité
/*virtual*/ void CPlanConcepteurs::CalculeCompatibiliteCreerAtelier()
{
  if ((Informations()->Vide())||(Informations()->EstPosition()))
  {
        Compatibilite(LOCAL);
        NomNouveauTheme(CREER_ATELIER);
  }
  else
  {// désignation d'un objet sentier ou atelier
        int       ident = ((CIdentifiants*)Informations())->Identifiant();
        CObjets   *objet = (theApp.ModeleObjets())->Objet(ident);

        if (objet->EstSentier())
        {
              Compatibilite(IMMEDIAT);
              NomNouveauTheme(SE_DEPLACER);
        }
        else //atelier désigné sur le plan
        {
              Compatibilite(IMMEDIAT);
              NomNouveauTheme(ADAPTER_ATELIER);
        }
  }
}

/*virtual*/ void CPlanConcepteurs::CalculeCompatibiliteAdapterAtelier()
{
  if (Informations()->Vide())
  {
        Compatibilite(LOCAL);
        NomNouveauTheme(ADAPTER_ATELIER);
  }
  else if (Informations()->EstPosition())
  {//désigne une position pour créer un atelier
        Compatibilite(DIFFERE);
        NomNouveauTheme(CREER_ATELIER);
  }
  else
  {// désignation d'un objet sentier ou atelier
        int       ident = ((CIdentifiants*)Informations())->Identifiant();
        CObjets   *objet = (theApp.ModeleObjets())->Objet(ident);

        if (objet->EstSentier())
        {
              Compatibilite(IMMEDIAT);
              NomNouveauTheme(SE_DEPLACER);
        }
        else //atelier désigné sur le plan
        {
              Compatibilite(LOCAL);
              NomNouveauTheme(ADAPTER_ATELIER);
        }
  }
}
```

```
/*virtual*/ void CPlanConcepteurs::CalculeCompatibiliteSeDeplacer()
{
  if (Informations()->Vide())
  {
        Compatibilite(LOCAL);
        NomNouveauTheme(SE_DEPLACER);
  }
  else if (Informations()->EstPosition())
  {//désigne une position pour créer un atelier
        Compatibilite(DIFFERE);
        NomNouveauTheme(CREER_ATELIER);
  }
  else
  {// désignation d'un objet sentier ou atelier
        int       ident = ((CIdentifiants*)Informations())->Identifiant();
        CObjets   *objet = (theApp.ModeleObjets())->Objet(ident);

        if (objet->EstSentier())
        {
              Compatibilite(LOCAL);
              NomNouveauTheme(SE_DEPLACER);
        }
        else //atelier désigné sur le plan
        {
              Compatibilite(DIFFERE);
              NomNouveauTheme(ADAPTER_ATELIER);
        }
  }
}

/*virtual*/ void CPlanConcepteurs::CalculeCompatibiliteCommuniquer()
{
  Compatibilite(DIFFERE);
  if (Informations()->Vide())
  {
        NomNouveauTheme(CREER_ATELIER);
  }
  else if (Informations()->EstPosition())
  {//désigne une position pour créer un atelier
        NomNouveauTheme(CREER_ATELIER);
  }
  else
  {// désignation d'un objet sentier ou atelier
        int       ident = ((CIdentifiants*)Informations())->Identifiant();
        CObjets   *objet = (theApp.ModeleObjets())->Objet(ident);

        if (objet->EstSentier())
        {
              NomNouveauTheme(SE_DEPLACER);
        }
        else //atelier désigné sur le plan
        {
              NomNouveauTheme(ADAPTER_ATELIER);
        }
  }
}
```

***Figure B.6.*** *Définition des compatibilités pour le manipulateur Plan Concepteur.*

Pour chaque nouvelle compatibilité, on définit le nouveau thème dans lequel se trouve l'opérateur. On peut ainsi définir la cohérence du dialogue en comparant le thème courant avant l'utilisation du manipulateur et le nouveau thème introduit par ce manipulateur. Dans tous les cas, si le manipulateur est utilisé pour être désactivé, sa compatibilité reste locale au thème courant. On indique ainsi que le manipulateur n'influe pas sur le thème courant.

### 4.2.La mise en place du processus de gestion du dialogue.

Le processus de gestion mis en place repose sur le schéma décrit dans le chapitre 3 (cf. Chapitre 3 figure III.6). Toute interaction d'un opérateur se traduit dans un premier temps par un événement. Cet événement est capturé par le LGB qui le transmet au manipulateur adéquat : le LGB active le manipulateur en appelant sa primitive *Traiter()* (cf. figure B.7). Grâce à ses mécanismes, le manipulateur interprète l'événement en un ensemble d'informations, puis transmet cet ensemble au gestionnaire de thèmes de l'opérateur actif (primitive *InformeEvolution()*).

```
//utilisation du manipulateur
void CManipulateurs::Traiter()
{
   CModelesOperateurs *modele = theApp.ModeleOperateurs();

   //on indique au manipulateur l'opérateur qui l'utilise
   Utilisateur(modele->OperateurMaitre());

   //le manipulateur n'a encore rien modifié dans la scène
   m_SceneModifiee = FALSE;

   //mécanismes du manipulateur activés
   m_Informations = CaptureInformations();

   // le manipulateur transmet les informations éventuelles au
   // gestionnaire de dialogue
   InformeEvolution();
}
```

*exemple pour le Plan Concepteur*

```
//mécanismes de traitement des informations
/*virtual*/ CInformations *CPlanConcepteurs::CaptureInformations()
{
        switch (CodeEvenement())
        {// on traite les événements capturés par le LGB et qui
         // concernent le manipulateur
                case E_SELECTION_OUTIL:
                        if (ApparenceActive())
                        {//on ferme la fenêtre 2D
                                DesactiverApparence();
                        }
                        else
                        {//le manipulateur n'est plus en attente
                                ActiverApparence();
                        }
                        Informations(new CInformations());
                        break;
                case E_APPUI_BOUTON_GAUCHE:
                        Informations(GereDesignation());
                        break;
                default:  break; //événement non traité
        }
        return (Informations());
}
```

***Figure B.7.*** *Contrôle du dialogue par le manipulateur.*

Le gestionnaire de thèmes évalue la compatibilité du manipulateur utilisé avec le thème courant et contrôle la cohérence du dialogue engagé : il s'agit de la primitive *EtatTheme* (cf. figure B.8). Cette primitive exécute en fonction du thème courant l'une des primitives de calcul des compatibilités : c'est la primitive *CalculeCompatibilite(theme)*, qui va à son tour exécuter l'une des primitives de calcul des compatibilités en fonction d'un thème. Dans le cas du manipulateur Plan Concepteur, il s'agit des primitives présentées dans la figure B.6.

Une fois le thème actualisé (i.e. modifié ou non), le manipulateur transmet les informations qu'il a éventuellement capturées pour poursuivre le dialogue : primitive *PoursuivreDialogue()* (cf. figure B.8). Il s'agit de transmettre ces informations aux modèles (modèle des objets d'application et/ou modèle des opérateurs) afin de réaliser les traitements correspondant aux manipulations de l'opérateur. De

manière plus synthétique, ceci revient à effectuer une étape du thème abordé, sachant que chaque thème est décomposé en étapes de dialogue.

```
//transmission des informations au gestionnaire de thèmes
/*virtual*/ void CManipulateurs::InformeEvolution()
{
        m_GestionThemes->GereSuiteDialogue(this);
}


//Gestion du dialogue
void CGestionThemes::GereSuiteDialogue(CManipulateurs *manip)
{
        CInformations        *infos = manip->Informations();

        if (!infos->Vide())
        {//cas où le manipulateur a des informations à transmettre
                m_Informations = infos;
                EtatTheme(manip, ThemeCourant());
                PoursuivreDialogue();
        }
        else
        {//on vérifie juste les éventuels changements au niveau
         // des thèmes
                EtatTheme(manip, ThemeCourant());
        }
        MiseAJourManipulateurs(manip);
}
```

```
void CGestionThemes::EtatTheme(CManipulateurs *manip, CThemes *theme)
{
        TypeEtatTheme       typeEtat;
        CThemes             *themeManip;

        manip->CalculeCompatibilite(theme);
        typeEtat = manip->EtatTheme(theme, &themeManip);
        switch (typeEtat)
        {
                case poursuite:  PoursuivreTheme();
                                 break;
                case suspension:SuspendreTheme(themeManip);
                                 break;
                case arret:      AnnulerTheme(manip, themeManip);
                                 break;
                default:         break;
        }
}
void CGestionThemes::PoursuivreDialogue()
{
        ThemeCourant()->Continue(this, m_Informations);
}
```

***Figure B.8.*** *Gestion de la poursuite du dialogue.*

Lorsque l'étape est accomplie, le thème est à nouveau en attente d'informations pour accomplir la prochaine étape. Enfin, avant de laisser l'opérateur continuer à interagir, le gestionnaire de thèmes informe tous les manipulateurs de l'évolution du dialogue afin qu'ils effectuent les mises à jour nécessaires. Il s'agit, par exemple, de vérifier les conditions d'utilisation des manipulateurs "Manettes" des portiques créés afin de détecter et proposer leur activation.

## 5. CONCLUSION.

L'implémentation que nous avons réalisée a donné des résultats intéressants sur la gestion du dialogue en considérant les manipulateurs et le modèle des opérateurs. Il est apparu que les manipulateurs apportent effectivement un contrôle implicite sur les interactions de l'opérateur. De plus, nous avons réussi à regrouper en un concept unique des outils de manipulations variés (outils de visualisation, de modification, de déplacement ...).

En outre, les bases de la modélisation des opérateurs que nous avons posées dans cette implémentation révèlent déjà des atouts importants pour le dialogue. Grâce aux connaissances modélisées, on parvient à offrir un dialogue plus adapté à l'opérateur. Nous avons modélisé trois types d'opérateurs afin de montrer par la modélisation :

- du concepteur, l'utilisation de manipulateurs variés (plan, calculatrice, manettes ...) ;
- du fabricant, la définition d'un autre type d'opérateurs et la mise en évidence que chaque type d'opérateur a des capacités propres ;
- du client, la détection de la communication ainsi qu'une représentation canonique différente des autres opérateurs.

Il est clair que cette implémentation n'a pas pour but de modéliser entièrement ces trois types d'opérateurs, mais de les modéliser de manière symbolique. Il est vrai, par exemple, que le fabricant n'a que très peu d'autonomie. De nombreuses extensions sont envisageables, comme la communication entre le concepteur et le fabricant, ce qui demande la mise en place d'un processus de compréhension du dialogue entre les deux intervenants.

Nous nous sommes attachés à spécifier l'application en utilisant les comportements. Nous avons ainsi montré qu'une application complexe peut être décrite de cette manière et que l'on peut envisager une déduction du dialogue.

Parallèlement aux apports propres à nos travaux, l'implémentation a permis de valider le LGB développé par [ROC 97] pour montrer une corrélation entre l'environnement Windows®, qui impose un ensemble de concepts de base parmi lesquels la notion de multi-fenêtrage, et l'environnement de CFAO, qui contient ses propres concepts de développement parmi lesquels la notion de fenêtres et de clôtures.

# CONTRIBUTION À LA DÉFINITION ET À LA MISE EN ŒUVRE D'UN DIALOGUE ADAPTATIF POUR LA CAO


Parce que de nombreux utilisateurs interviennent dans les applications, fournir un dialogue adaptatif est une tâche désormais difficile. Tout au long de ce mémoire, nous nous intéressons à l'amélioration des dialogues pour les applications de CFAO (Conception et Fabrication Assistées par Ordinateur).

Nous montrons tout d'abord les efforts que l'on peut fournir à travers une étude des besoins et des techniques utilisées plus ou moins indirectement pour construire des dialogues adaptatifs. Nous soulignons l'importance de domaines tels que l'IA (Intelligence Artificielle) pour apporter des méthodes de gestion des connaissances sur les utilisateurs. Pour répondre aux besoins des systèmes de CFAO et proposer une aide au développement, nous présentons une méthode duale d'acquisition des connaissances du concepteur de dialogue. Cette méthode nous permet d'offrir la possibilité à un concepteur inexpérimenté dans la modélisation du dialogue en CFAO de décrire le dialogue de son application. Nous nous attachons à proposer des solutions qui demandent un minimum d'efforts de la part du concepteur.

Pour assurer un meilleur contrôle sur les manipulations de l'opérateur, nous introduisons un concept permettant de les centraliser de manière unique : les manipulateurs. Nous expliquons leurs propriétés afin de souligner leur importance dans l'obtention d'un dialogue adaptatif. Nous montrons comment les manipulateurs s'intègrent dans le processus de construction du dialogue.

Nous voulons aller plus loin que nous intéresser uniquement aux manipulations de l'opérateur pour rendre le dialogue adaptatif. Nous proposons de modéliser l'opérateur afin de représenter un maximum de connaissances à son sujet. Nous montrons que cette approche prend toute son importance dans les systèmes de CFAO qui utilisent la RV (Réalité Virtuelle). La modélisation de l'opérateur associée au concept des manipulateurs nous permet d'envisager de nouveaux types d'aides.



**Mots-clés :** CFAO, dialogue homme/machine, modélisation, manipulations, opérateurs, connaissances.

# CONTRIBUTION À LA DÉFINITION ET À LA MISE EN ŒUVRE D'UN DIALOGUE ADAPTATIF POUR LA CAO


Parce que de nombreux utilisateurs interviennent dans les applications, fournir un dialogue adaptatif est une tâche désormais difficile. Tout au long de ce mémoire, nous nous intéressons à l'amélioration des dialogues pour les applications de CFAO (Conception et Fabrication Assistées par Ordinateur).

Nous montrons tout d'abord les efforts que l'on peut fournir à travers une étude des besoins et des techniques utilisées plus ou moins indirectement pour construire des dialogues adaptatifs. Nous soulignons l'importance de domaines tels que l'IA (Intelligence Artificielle) pour apporter des méthodes de gestion des connaissances sur les utilisateurs. Pour répondre aux besoins des systèmes de CFAO et proposer une aide au développement, nous présentons une méthode duale d'acquisition des connaissances du concepteur de dialogue. Cette méthode nous permet d'offrir la possibilité à un concepteur inexpérimenté dans la modélisation du dialogue en CFAO de décrire le dialogue de son application. Nous nous attachons à proposer des solutions qui demandent un minimum d'efforts de la part du concepteur.

Pour assurer un meilleur contrôle sur les manipulations de l'opérateur, nous introduisons un concept permettant de les centraliser de manière unique : les manipulateurs. Nous expliquons leurs propriétés afin de souligner leur importance dans l'obtention d'un dialogue adaptatif. Nous montrons comment les manipulateurs s'intègrent dans le processus de construction du dialogue.

Nous voulons aller plus loin que nous intéresser uniquement aux manipulations de l'opérateur pour rendre le dialogue adaptatif. Nous proposons de modéliser l'opérateur afin de représenter un maximum de connaissances à son sujet. Nous montrons que cette approche prend toute son importance dans les systèmes de CFAO qui utilisent la RV (Réalité Virtuelle). La modélisation de l'opérateur associée au concept des manipulateurs nous permet d'envisager de nouveaux types d'aides.



**Mots-clés :** CFAO, dialogue homme/machine, modélisation, manipulations, opérateurs, connaissances.