Patry G. & Girard P. Colloque de l'AFIG'97, Rennes, 3-5 Décembre 1997, pp. 1-10.

Techniques d'interaction : intégrer simplement l'exploration dans des dialogues structurés.

Guillaume Patry, Patrick Girard

LISI/ENSMA
Site du Futuroscope
B.P. 109, 86960 FUTUROSCOPE CEDEX
{patry,girard}@ensma.fr

RÉSUMÉ

La conception technique est un domaine d'applications où les interfaces homme-machine doivent répondre à des besoins spécifiques. Parmi ceux-ci, le support de tâches complexes et structurées est essentiel. Il permet de laisser à l'utilisateur libre cours à son activité de conception. La richesse des dialogues structurés a sa contrepartie : il est généralement impossible d'informer l'utilisateur par des techniques de prévisualisation ou de feed-back proactif. Nous montrons dans ce travail quelles sont les problèmes posés par l'introduction de la technique d'exploration dans un dialogue structuré. Puis, nous proposons quelques règles qui permettent de garder consistance et cohérence au dialogue. Enfin, nous présentons l'implémentation que nous avons réalisée.

MOTS-CLEFS : Techniques d'interaction, Dialogues structurés, Outils de construction d'interface, CAO.

1. INTRODUCTION

Permettre à l'utilisateur d'appréhender le plus tôt possible le résultat de ses actions est un facteur de qualité d'une interface homme-machine [1, 7]. Des techniques d'interaction telles que la visualisation de fantômes lors du déplacement ou du changement de taille d'objets confèrent de ce point de vue à la manipulation directe un avantage certain sur d'autres styles d'interaction. Cependant, ces derniers sont parfois nécessaires, comme nous le verrons par exemple dans le domaine de la conception technique ; il serait souhaitable dans ce cas de pouvoir également disposer de techniques de feed-back proactif.

Dans ce travail, nous visons à incorporer de façon systématique la technique dite d'exploration dans un dialogue structuré. Cette technique consiste à visualiser le résultat d'une action alors que tous ses paramètres ne sont pas complètement fournis. Il s'agit en d'autres termes de présenter à l'utilisateur le résultat potentiel d'une action. L'exemple le plus simple est celui de la construction d'un segment, où l'on dessine un " segment élastique " entre une première position et la position courante de la souris, bouton enfoncé. Le segment définitif est créé à partir de la position de la souris lorsque l'utilisateur relâche le bouton. Cette technique est nommée Rubber Banding [2], ce que l'on peut traduire par " Écho Fantôme " en français.

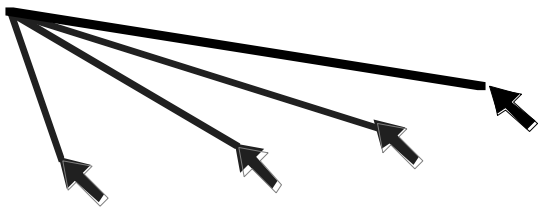

Figure 1: échos fantômes d'un tracé de segment.

Ce comportement est très difficile à mettre en oeuvre dans un dialogue structuré, où chaque paramètre d'une action peut-être obtenu par l'intermédiaire d'une expression plus ou moins complexe. Nous exposerons dans la première partie de ce travail les particularités des dialogues structurés multi-objets, et verrons pourquoi l'intégration de l'exploration y est si difficile. Dans la seconde partie, nous examinerons les différents problèmes posés, et établirons un ensemble de règles permettant d'incorporer la technique d'exploration dans un dialogue structuré tout en respectant sa cohérence. Enfin, nous

présenterons la solution que nous avons réalisée, qui cherche à minimiser le coût d'une telle incorporation pour le concepteur d'applications.

2. LE DOMAINE D'APPLICATION

La **C**onception **A**ssistée par **O**rdinateur (CAO), et plus généralement les **A**pplications **G**raphiques **I**nteractives de **C**onception **T**echnique (AGI-CT), peuvent être définies comme l'ensemble des aides informatiques au bureaux d'études et de méthodes depuis l'élaboration du cahier des charges jusqu'à l'établissement des documents nécessaires à la fabrication [3]. Une autre définition, plus proche du domaine informatique, consiste à présenter la CAO comme une technique se préoccupant de la création des données qui décrivent un objet en cours de réalisation, de la manipulation de ces données afin d'aboutir à une forme achevée de conception, et de la génération des informations nécessaires à la fabrication de cet objet. Cette dernière définition met en avant les notions importantes de Modèle et de dialogue Homme Machine. Le modèle correspond à la représentation informatique de l'objet en cours de conception, et joue un rôle fondamental en conception technique. C'est au travers du dialogue que devront être utilisées au mieux les qualités de l'homme et de la machine, et que seront exprimées les intentions de l'homme vis-à-vis des données du modèle.

Les applications de conception techniques sont par nature très complexes. Les contraintes qui existent sur l'activité même de conception se retrouvent naturellement dans ces logiciels. La nature des interactions y pose notamment problème. Dans cette section nous en étudierons deux aspects : le caractère multi-objets des interactions, et la structure naturellement hiérarchique des tâches, qui induisent un certain type de dialogue. Nous verrons ensuite quelles sont les conséquences de ce dernier sur le feed-back dans ces applications.

2.1. Interactions Multi-Objets

L'activité de conception technique suppose l'application de règles strictes dépendant du domaine. En cela elle n'a rien à voir avec l'activité beaucoup plus libre de conception 'artistique'. Ainsi au cours du processus de conception, le dessinateur technique connaît-il parfaitement les relations qui doivent exister entre les différents éléments de son modèle. De ce fait, les systèmes CAO permettent d'exprimer explicitement ces relations, sous forme de contraintes ou d'opérateurs géométriques entre objets, et les utilisateurs sont habitués à exprimer ces relations. Un utilisateur peut vouloir construire un cercle tangent à trois autres cercles (action multi-objets basée sur le caractère relationnel de la géométrie), ou construire un noeud de son maillage aligné verticalement avec un noeud donné, et appartenant à une frontière définie par une courbe donnée (action multi-objets basée sur la topologie d'un maillage).

2.2. Interactions Structurées

En plus des relations entre objets, de nombreuses relations existent entre les grandeurs qui caractérisent les objets du modèle. Ces relations sont connues du concepteur, qui souhaite pouvoir les utiliser dans la construction de son modèle. Ceci implique que les tâches doivent donc pouvoir être structurées, c'est à dire que l'utilisateur doit pouvoir utiliser le résultat de certaines tâche pour en accomplir d'autres [10]. Ce point est illustré par l'exemple suivant : " je veux construire un cercle dont le centre est le projeté de *ce* point sur *cette* ligne et le rayon la moitié de la distance entre ce point et l'extrémité de *cette* ligne ". La Figure 1 présente le résultat attendu.

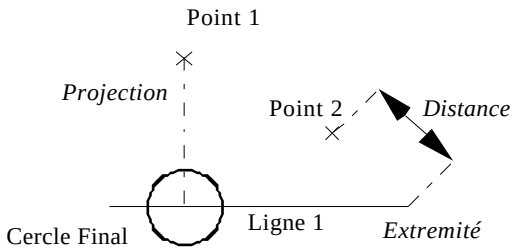

Figure 2 : but de l'utilisateur

Ceci se décompose naturellement et, suivant en cela la théorie de Norman [9] en une arborescence de buts/sous-buts, qui peuvent être directement traduits en tâches dans un système CAO. Ainsi, la tâche *Construire un cercle par centre et rayon* sera-t-il décomposé en deux sous-buts, *Donner le centre* et

Calculer le rayon, eux-mêmes décomposés en sous-buts *Calculer la projection d'un point sur une ligne* et *Calculer l'expression ...*, et ainsi de suite. À chacun de ces sous-buts, correspondent des actions directement exécutables sur le système de conception technique.

Il est cependant important de noter qu'une telle hiérarchisation des buts et des tâches amène à différencier deux types d'actions : les actions de production, qui produisent des éléments pour d'autres actions, et les actions terminales, lesquelles ne produisent rien du point de vue des autres actions. Ainsi par exemple la gestion d'objets du modèle (création, destruction...) ou la gestion de fenêtre (zoom, translation...) appartiennent clairement au domaine des actions terminales. A l'opposé, les expressions grapho-numériques (distance, extrémité, projection...) sont des actions typiques de production.

Les dialogues des systèmes de conception technique sont généralement conçus pour supporter ces deux types de d'actions, permettant ainsi de raccourcir la distance entre le modèle d'action de l'utilisateur et l'utilisation du système.

2.3. Un feed-back limité

La complexité du dialogue se retrouve naturellement dans la programmation des systèmes de conception technique, où elle introduit des effets secondaires fâcheux. Ainsi, la plupart de ces systèmes ne fournissent pas de feed-back dynamique des actions en cours, celles-ci dépendant, la plupart du temps, du résultat d'autres actions, elles-mêmes en attente de paramètres... La Figure 3 présente un exemple de tâche ou une terminale (la création d'un arc de cercle) attendant d'une autre (le calcul de l'extrémité d'un segment la fourniture d'un paramètre de type 'position géométrique'. Cette seconde action, de production, fournit cette donnée à partir d'un paramètre 'segment', lui même fournit via une fonction de désignation. L'extrémité fournie dépend du pointage de désignation de ce segment.

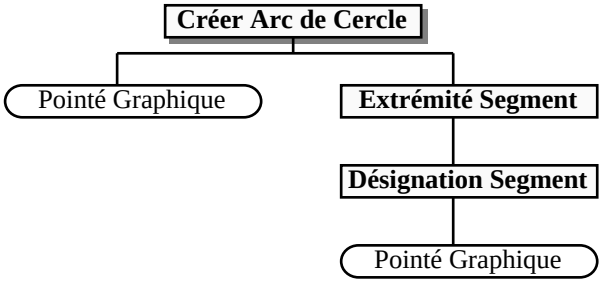

Figure 3 : Création d'un arc de cercle passant par l'extrémité d'un segment

Seules les actions correspondant à la tâche terminale (désignation par exemple) au sein d'une tâche structurée pourront posséder un feed-back proactif. Dans l'exemple précédent, alors que l'on souhaiterait pouvoir visualiser l'arc de cercle final, seul le feed-back proactif de la désignation du segment peut être réalisé, du fait de l'imbrication des tâches.

La Figure 4 présente ainsi le résultat d'une mauvaise sélection de segment : les sous-figures **a)** et **b)** de la figure 4 montrent ainsi respectivement le feed-back lorsque l'utilisateur désigne le segment, et la construction finale (erronée, car l'extrémité finale du segment n'est pas celle voulue) ; en **c)**, la construction espérée est représentée.

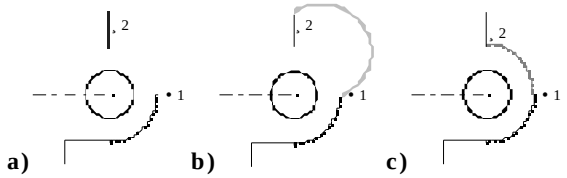

Figure 4: un exemple de sélection de segment (a) conduisant à une construction invalide (b)) au lieu de (c)

Cette absence de feed-back viole les principes élémentaires d'ergonomie : l'utilisateur ne peut vérifier visuellement la validité de son action, et doit réagir *a posteriori* en cas d'erreur, en annulant puis recommençant l'action. Une solution pourrait consister à demander confirmation à l'utilisateur de la création définitive. Elle est cependant inacceptable en regard du nombre considérable d'interactions qu'elle ajouterait à l'utilisation d'un système de conception technique.

3. EXPLORATION ET DIALOGUES STRUCTURÉS

Supposons que nous voulions intégrer l'exploration dans un système permettant de construire des rectangles et des cercles, mais supportant les tâches structurées. Ce système permettrait ainsi une phrase de construction telle que celle-ci : " Je veux créer un rectangle par deux coins dont l'un des coins est le centre d'un cercle et l'autre le centre d'un rectangle (sous-entendu : que je désigne avec ma souris) ". La situation interactive au moment de la création peut se visualiser ainsi (avec en pointillé, le rectangle attendu) :

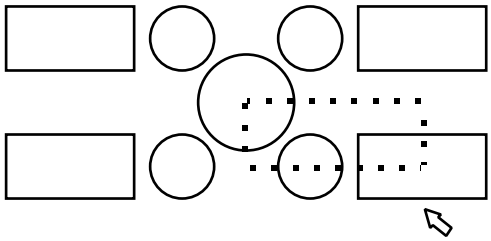

Figure 5 : Situation du système lors de la création

Pour bien comprendre les problèmes posés par l'intégration de l'exploration dans une telle situation, il est nécessaire de préciser quelques notions sur les dialogues structurés. La généralisation des interfaces graphiques (Mac-OS, X-Window, MS-Windows, etc.) a rendu omniprésent le modèle à événements dans les applications. Cependant, nombre de modèles et outils cherchent à élever le niveau d'abstraction de la programmation des interfaces homme-machine. Lorsque l'on exclut la composante " Présentation " des interfaces, décrire le comportement de cette dernière revient à décrire le dialogue homme-machine.

Un modèle très général, qui permet de représenter les tâches structurées évoquées plus haut, mais qui ne présuppose aucune implémentation particulière (événements, automates, Réseaux de Pétri, etc.) peut être décrit comme suit : l'utilisateur produit des **jetons** (événements utilisateurs ou non) qui sont transformés par l'application en fonction de l'état du dialogue homme-machine afin de réaliser des actions. Dans le cas qui nous préoccupe, les actions qui correspondent à des tâches de production **transforment** les jetons. Ainsi, des actions telles que " Extrémité de segment " ou " Centre de cercle " transforment respectivement un segment ou un cercle en une position géométrique. À l'inverse, les actions correspondant aux tâches terminales **consomment** les jetons (après transformation ou non). Les dialogues structurés permettent l'enchaînement dans une même " phrase de dialogue " de plusieurs tâches de production et d'une tâche terminale.

Ainsi, dans notre exemple, et en considérant que les jetons produits par le système sont très élémentaires (positions géométriques souris, numériques), les actions de production nécessaires sont au nombre de quatre : " Désigner un Cercle ", " Désigner un Rectangle ", " Centre de Cercle " et " Centre de Rectangle ". Les deux premières transforment une position géométrique en entité (Cercle ou Rectangle, par proximité) et les deux dernières font le contraire (par valeur caractéristique). L'action terminale est la " Construction d'un Rectangle par Diagonale ", qui consomme deux positions géométriques. La figure suivante représente l'arbre des tâches de l'exemple :

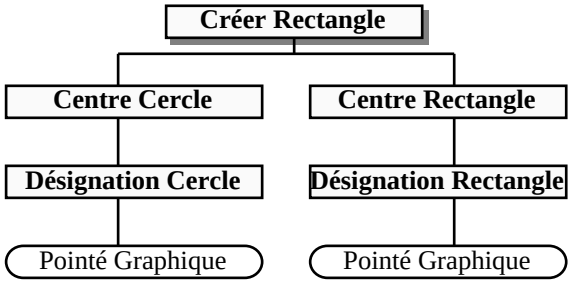

Figure 6 : Arbre des tâches de l'exemple

L'existence d'actions multi-objets nécessite de la part du dialogue de **conserver**, pendant le déroulement de la phrase, certains jetons, qui ne seront consommés que lors du déclenchement de la tâche qui les consomment (terminal ou non). Notons que dans notre exemple, les tâches de production sont mono-objet, mais cela n'est pas général ; ainsi une tâche de projection d'un point sur une droite est multi-objets mais est une tâche de production (elle produit une position géométrique).

Les différentes règles que nous donneront dans la suite de cette section vont s'exprimer par rapport à la nature des jetons et aux notions de production/transformation/consommation de ces jetons.

3.1. L'exploration par rapport à l'exemple

On souhaite donc pouvoir pré-visualiser la construction du rectangle dès que cela est possible, i.e. lorsque l'on enfonce le bouton de la souris, lors du deuxième pointage de désignation. Si l'on suppose que les sous-tâches s'effectuent dans l'ordre *Centre Cercle* puis *Centre Rectangle*, il s'agit donc de pré-visualiser le *rectangle final* lors de la désignation du rectangle dont on souhaite utiliser le centre. Le centre du cercle est déjà calculé (son écho peut d'ailleurs être visualisé) mais les actions de désignation du rectangle et de calcul de son centre doivent pouvoir s'exécuter de façon à fournir à l'action de création du rectangle son deuxième paramètre. Bien évidemment, tout cela doit se faire sous forme " fantôme "...

3.2. La notion de jeton fugitif

Nous avons implicitement décrit, dans ce qui précède, la nature des jetons de bas-niveau nécessaires à l'exécution " d'échos fantômes ". Il s'agit des positions données lorsqu'un bouton de la souris est enfoncé. En X-Window par exemple, l'événement correspondant est *Motion Notify*. Mais cette définition n'est pas suffisante : Dans notre modèle, tout jeton peut être transformé (par exemple, une position en désignation d'objet, elle-même transformée en valeur caractéristique). Or, dans un feed-back proactif étendu à l'arbre des tâches, il est nécessaire que ces jetons transformés soient identifiés comme susceptibles de générer des échos fantômes. Nous proposons la définition pour ces jetons de la notion de " caractère fugitif ", ce qui donne la première règle :

Règle 1 : Les jetons qui transitent dans le dialogue sont caractérisés par un drapeau indiquant s'ils sont 'fugitifs' ou 'non fugitifs'. Le caractère 'fugitif' ou 'non fugitif' est déterminé au plus bas niveau de l'interaction

Notons que le caractère 'fugitif' n'est pas obligatoirement du niveau de l'événement 'Motion Notify' mentionné plus haut. De la même manière que la différence entre position géométrique fugitive et non fugitive, on peut définir une valeur numérique fugitive on non, fournie au moyen d'un widget du type ascenseur.

3.3. Actions fugitives

Les actions qui utilisent les jetons fugitifs doivent assurer un comportement particulier (effacement de situation précédente, puis affichage de la nouvelle situation).

Règle 2 : les actions utilisant des jetons 'fugitifs' sont dénommées 'actions fugitives'.

Ces actions, que nous nommerons actions fugitives, peuvent, comme les actions 'courantes', comporter plusieurs paramètres. Comme ces dernières, elles ne peuvent s'exécuter que lorsque tous leurs paramètres leur sont fournis. Il en résulte que le comportement 'écho fantôme' ne pourra se manifester que par rapport au dernier paramètre qui seul pourra être fugitif. Si le dialogue permet à l'utilisateur de choisir l'ordre des paramètres fournis à une action, le paramètre fugitif sera le dernier dans l'ordre chronologique. La figure 7 montre les deux cas d'écho fantôme pour un cercle par centre et rayon, selon l'ordre des paramètres :

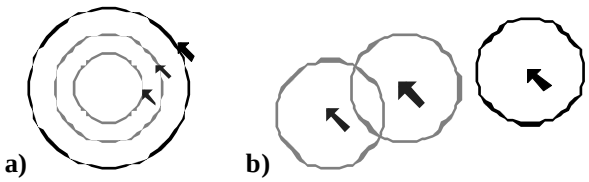

Figure 7 : Écho fugitif de la construction d'un cercle avec (a) le centre fixé, ou (b) le rayon fixé.

3.4. Constance de l'état du dialogue

Un écho fugitif comportant un nombre quelconque d'appels (jusqu'à ce que l'utilisateur relâche la souris), les paramètres non fugitifs doivent être conservés entre chaque appel de l'action fugitive, et ce jusqu'à l'appel final. D'un point de vue plus général, c'est tout l'état du dialogue qui est concerné :

Règle 3: Un jeton fugitif peut déclencher une action fugitive, mais ne doit en aucun cas changer l'état du dialogue

Ceci implique également qu'une action utilisant une donnée fugitive ne doit pas détruire les paramètres qui lui sont transmis. Ceux ci doivent rester valide après appel de cette fonction. On notera par extension que les actions ne modifiant pas le modèle géométrique ne sont pas différentes lorsque le jeton est fugitif ou lorsqu'il ne l'est pas. Le centre d'un cercle reste le même, que ce cercle soit fugitif ou non. Dans de nombreux cas, l'action appelée sera identique quel que soit le caractère du jeton. En revanche, les actions agissant sur le modèle géométrique présentent une différence : l'action finale modifie effectivement le modèle, alors que l'action fugitive se contente d'afficher un écho de ce que sera l'action finale...

3.5. Transmission du caractère fugitif

Dans notre modèle, les jetons sont transmis de tâche en tâche, et interprétés au sein de différentes fonctions. Si une action recevant un jeton fugitif retournait un jeton non fugitif, la tâche de niveau supérieur en attente de ce jeton exécuterait son action 'finale', alors que l'utilisateur n'a pas terminé son interaction. Il est donc important de ne pas perdre le caractère fugitif des données lors des transformations. Tant que l'utilisateur 'émet' des jetons fugitifs, toutes les tâches actives doivent recevoir des jetons fugitifs, quelle que soit leur position dans cette hiérarchie.

Règle 4 : Tout jeton produit par une action fugitive est lui même fugitif.

3.6. Conséquences sur l'écho

Par le biais de ces quatre règles, il devient possible de réaliser un système où toute expression peut avoir un écho permettant l'exploration, et ce quel que soit le nombre de sous-tâches entrant en jeu. En outre cet écho étant géré par chaque tâche, on dispose de la possibilité de réaliser un écho multi-niveau : si l'on reprend comme exemple la construction d'un rectangle dont le second coin est le centre d'un autre rectangle (cf Figure 8), on peut disposer de trois niveaux d'écho : écho de la sélection du second rectangle, écho du centre de ce rectangle, et écho du futur rectangle passant par ce point. Lors du déplacement de la souris, on obtient à chaque instant l'écho du rectangle le plus proche, du centre de ce rectangle et l'écho du futur rectangle

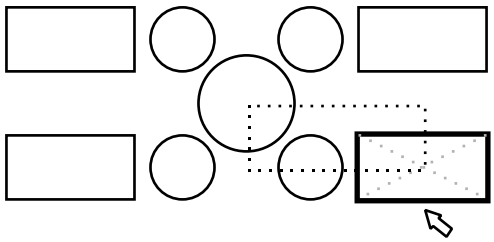

Figure 8 : **Échos multi-niveau lors de l'exploration d'une construction:**

4. MISE EN OEUVRE

Dans cette section, nous allons montrer comment l'application des règles décrites ci-dessus à un a pu être réalisée pour obtenir un système supportant à la fois les tâches structurées et l'exploration. Pour cela, nous sommes partis du modèle issu des travaux de Laurent Guittet [5, 6], le modèle des Interacteurs Hiérarchisés.

Ce modèle a été introduit pour prendre en compte dans l'architecture d'une application graphique interactive de conception technique la nécessité de dialogues structurés aussi libres que possibles. Dans ce modèle, le contrôleur de dialogue dispose d'une hiérarchie d'agents réactifs, nommés **Interacteurs**. Ceux-ci correspondent aux différents niveaux de tâches identifiés dans l'application. Chaque interacteur est associé à des types d'informations d'entrée définis par la tâche qu'il permet d'accomplir, et produit des informations de sortie qui représentent le résultat de cette tâche.

La transmission d'informations entre interacteurs est assurée par un module indépendant, nommé **Moniteur**, ayant accès à tous les interacteurs. Il assure ainsi leur masquage les uns vis-à-vis des autres. L'unité de communication est le jeton. Il s'agit d'un élément correspondant à des commandes (messages) ou des paramètres (données) pouvant circuler au sein du système. Un jeton disponible est

transmis par le moniteur au premier interacteur supérieur dans la hiérarchie en attente d'un jeton de ce type. Le moniteur assure ainsi la communication et le contrôle entre les différents interacteurs. Ceux-ci sont donc indépendants et il ne peut y avoir de communication directe entre eux.

La figure suivante illustre le modèle sur le petit exemple de la section précédente. On dispose de trois interacteurs : l'interacteur de Création, de plus haut niveau est chargé d'ajouter des objets au modèle. L'interacteur d'Information est à même de fournir des informations sur un objet que l'on lui fournit. Enfin, l'interacteur de Désignation, de plus bas niveau peut transformer un pointage en un objet une catégorie donnée (cf. Figure 9).

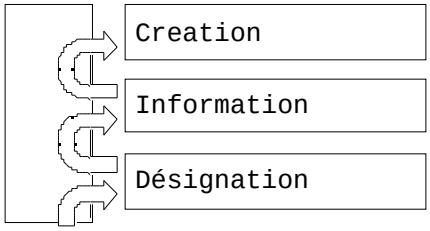

Figure 9 : Hiérarchie des Interacteurs pour un mini système CAO

4.1. État initial

Le modèle a servi de base à la réalisation de quelques applications interactives, dont une qui est actuellement en phase industrielle [4]. Des outils ont été créés, permettant, à partir d'une spécification formelle du dialogue, de générer l'application interactive correspondant à ce dialogue. Nous donnerons ci-dessous un aperçu du langage de description, afin de montrer par la suite les transformations apportées.

Chaque interacteur est décrit sous forme d'un automate augmenté (ATN [11]). Les automates sont décrits dans un langage spécialisé, nommé LSI (**L**angage de **S**pécification d'**I**nteracteur). Sommairement, on donne pour chaque transition l'état de départ, l'état d'arrivée, et le paramètre validant la transition. On peut y ajouter l'action à réaliser lors du passage de la transition. Par exemple, la définition, au sein de l'interacteur 'Création', de la suite de paramètre amenant à la création d'un rectangle est donnée ci-dessous. Elle débute par la réception d'une commande 'Rectangle', se poursuit avec la réception de deux positions et l'appel de l'action à réaliser. On retourne ensuite à l'état initial.

```
#Interacteur
   Creation
#Etats
   Initial
   Rectangle
   Point_1
#Transitions
   Initial   > Rectangle : 'Rectangle'
   Rectangle > Point_1   : Pointe;
   Point_1   > Initial   : Pointe :
      Creer_Rectangle (Pointe, Pointe);
```

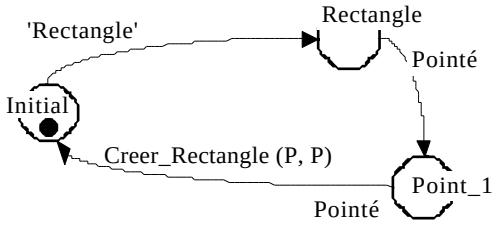

Figure 10 : Automate de l'interacteur de création (partiel)

4.2. Transformations réalisées

Le modèle des interacteurs hiérarchisés est particulièrement bien adapté au traitement des tâches structurées multi-objets, mais ne résout pas directement le problème de l'exploration : un interacteur ne reçoit ses paramètres que lorsque les interacteurs inférieurs ont terminé de traiter les leurs. Cependant, il est conforme au modèle plus général que nous avons utilisé dans la section précédente pour établir

nos règles, et, en particulier, les fonctions sont indépendantes de la transmission de paramètre. Nous détaillons ci-dessous les quelques modifications apportées au système pour réaliser notre but.

Comportement du moniteur/interacteur

Au sein du contrôleur de dialogue, aucun composant ne dispose d'une vue globale de l'action en cours. Chaque interacteur connaît uniquement son état, et les actions qu'il doit réaliser en fonction des paramètres qui lui sont transmis. Il ignore d'où proviennent ces paramètres. Le moniteur quant à lui ne s'occupe que de transmettre les jetons d'interacteur en interacteur. Il ignore quelles actions particulières sont réalisées, et quels sont les paramètres nécessaires. L'exploration doit donc être étudiée au niveau de l'interacteur et non au niveau plus général du moniteur. Ceci a pour conséquence que le traitement de l'exploration doit se faire action par action, indépendamment de la provenance des paramètres, puisque celle ci est inconnue.

Ajout " d'actions fugitives "

Afin de gérer l'exploration d'une action via le caractère fugitif d'un paramètre, une spécialisation des transitions a été introduite : les transitions fugitives.

Ces transitions ne sont validées que sur un paramètre (jeton) fugitif; De plus, à toute transition fugitive doit être associée une transition non fugitive validée par le même type de paramètre. Cette contrainte existe pour éviter qu'un interacteur ne bloque des informations fugitives, mais laisse passer le jeton standard correspondant. On aurait alors un comportement différent des interacteurs supérieurs selon que les jetons transmis sont fugitifs ou non.

En outre, le système assure que les règles décrites plus haut sont vérifiées à tout instant. Notamment qu'une transition fugitive part d'un état pour arriver au même état; et que les jetons stockés au sein d'un interacteur sont conservées après l'appel à l'action fugitive (application de la règle 3).

Au niveau de la description des interacteurs, la notation concernant les actions fugitives a pu, grâce aux règles définies précédemment, être considérablement simplifiée. Une transition décrite en LSI peut éventuellement comporter deux actions. La première correspond à l'action standard. La seconde correspond à l'action en cas de jeton fugitif (action fugitive). L'existence de cette action entraîne automatiquement la création d'une transition fugitive au sein du système. La figure suivante illustre cette notation, et donne une visualisation graphique de l'automate avec la transition fugitive.

```
#Interacteur
   Creation
#Etats
   Initial
   Rectangle
   Point_1
#Transitions
   Initial   > Rectangle : 'Rectangle'
   Rectangle > Point_1   : Pointe;
   Point_1   > Initial   : Pointe :
      Creer_Rectangle     (Pointe, Pointe)
    / Creer_Rectangle_Tmp (Pointe, Pointe);
```

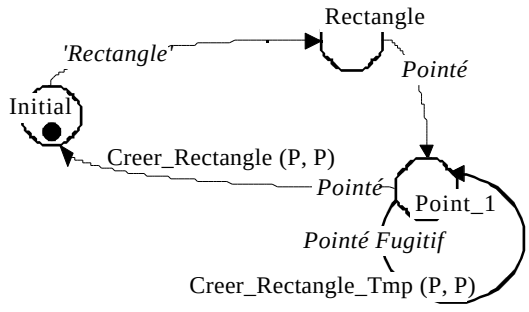

Figure 11 : transition fugitive

Cas des interacteurs d'expression

Certaines actions correspondent à des tâches d'expressions, i.e. d'interrogation, d'information ou de transformation, sans que le modèle géométrique ne soit modifié. L'exécution de ces action est indépendante du caractère fugitif de leurs paramètres, alors que des actions modifiant le modèle ne le

sont pas. Par exemple, le calcul de centre d'un cercle est identique que ce cercle soit fugitif ou non. A l'opposé l'action de création de rectangle à une conséquence variable selon le caractère fugitif des paramètre : l'action standard modifie effectivement le modèle, alors que l'action fugitive se contente d'afficher un écho de ce que sera l'action standard... Pour les actions d'expression, l'action fugitive est la même que l'action standard.

Le regroupement de fonctions en interacteurs par niveau de tâche permet d'étendre le concept d'expression aux interacteurs entiers. Les interacteurs d'expression, dans leur ensemble, ne modifient pas le modèle. Le mot-clef LSI 'expression' permet de spécifier ce comportement. Dans ce cas, l'absence d'une action fugitive sur une transition indique que l'on doit utiliser l'action standard quel que soit le caractère fugitif du jeton. Ainsi, les deux portions d'automates ci-dessous sont-elles strictement équivalentes :

```
#Interacteur
   Information = Expression
...
#Transitions
   Initial > Centre : 'Centre'
   Centre  > Initial : Rectangle :
      Centre_Rectangle (Rectangle) Position;

#Interacteur
   Information
...
#Transitions
   Initial > Centre : 'Centre'
   Centre  > Initial : Rectangle :
      Centre_Rectangle (Rectangle) Position
    / Centre_Rectangle (Rectangle) Position;
```

Figure 12 : Équivalence de notations

L'existence de ce mot clef permet de factoriser le caractère 'transparent' d'un l'interacteur vis à vis du modèle géométrique. Il permet de spécifier en un mot le comportement de toutes les actions d'un interacteur, évitant ainsi d'avoir à réécrire celui ci pour modifier les transitions pour y ajouter une action fugitive.

5. CONCLUSION

Le système tel qu'il a été réalisé comporte plusieurs avantages. En premier lieu, et conformément au but qui lui était assigné, il permet d'assurer un exploration d'un dialogue structuré, et ce quelle que soit la complexité de ce dialogue. Celle-ci n'est limitée que par la décomposition en niveaux de tâches (laquelle est indépendante du concept d'exploration) et la mémoire à court terme de l'utilisateur (selon le modèle du processeur humain [8]).

En second lieu, le système permet une évolution facilitée d'une application de conception technique vers une application comportant l'exploration. Seules sont à ajouter des actions prévisualisant le résultat des actions pouvant modifier le modèle. Les actions d'expression sont directement réutilisables, sans aucune modification du source, et avec un minimum de modification aux description d'interacteurs existantes.

Les règles d'exploration et le système tels qu'ils ont été décrits dans cet article ont été validés sur une application prototype de conception technique simplifiée. Le portage d'une application industrielle sur ce noyau devrait est en cours. Par la suite, l'adjonction d'un noyau de manipulation directe au système permettra de suppléer à certaines lourdeurs de dialogue, tout en autorisant les tâches structurées lorsque nécessaire. Ces travaux s'intègrent dans un projet plus vaste, visant à définir un générateur d'applications interactives polyvalent.

BIBLIOGRAPHIE

1. *Coutaz, J. Interface Homme-Machine : un regard critique. In Proceedings of Journées d'Études AFCET : Interfaces Homme-Machine (21 Octobre, Paris), AFCET, 1992, pp. 1-24.*
2. *Foley, Dam, v., Feiner et Hughes. Computer Graphics, Principles and Practice. Addison-Wesley Publishing Comp., 1990.*
3. *Gardan, Y. La CFAO introduction, techniques et mise en oeuvre. Hermes, Paris, 1991.*
4. *Girard, P., Potier, J.-C. et Pierra, G. EBP : Example-Based programming in parametrics. Casstte Video IHM'96, Grenoble, 1996.*
5. *Guittet, L. Contribution à l'Ingéniérie des Interfaces Homme-Machine - Théorie des Interacteurs et Architecture H4 dans le système NODAOO. PhD : Poitiers, 1995.*

6. *Guittet, L. et Pierra, G. Conception modulaire d'une application graphique interactive de conception technique : la notion d'interacteur. In Proceedings of Interfaces Homme-Machine Lyon), 1993, pp. 151-156.*
7. *Meinadier. L'interface Utilisateur : pour une informatique plus conviviale. Dunod Informatique, Paris, 1991.*
8. *Miller, J.-G. The magic number Seven : somes limits in our capacity for processing information. 1986.*
9. *Norman, D. User Centered System Design. Lawrence Erlbaum Associates, 1986.*
10. *Pierra, G. Towards a taxonomy for interactive graphics systems. In Proceedings of Eurographics Workshop on Design, Specification, Verification of Interactive Systems (June 7-9, 1995, Bonas), Springer-Verlag, 1995, pp. 362-370.*
11. *Woods, W. Transition Network Grammars for Natural Language Analysis. 1970.*