Patry G. Évaluation des dialogues structurés. Journées Francophones sur l'Ingénierie de l'Interaction Homme-Machine (IHM'99), Eds. J. Nanard & P. Girard, Pub. Cépaduès, Montpellier, 22-26 Novembre 1999, pp. 118-125.

Évaluation du Contexte de Travail par l'utilisateur

Guillaume PATRY

LISI/ENSMA
Téléport 2, BP 109
86960 Futuroscope cedex, France
patry@ensma.fr
Téléphone: (33/0) 5 49 49 80 62
Télécopie : (33/0) 5 49 49 80 64

RÉSUMÉ
Il est clairement démontré aujourd'hui qu'un écho proactif améliore sensiblement les qualités ergonomiques des applications interactives. Les différentes formes de dialogue employées dans le domaine de la conception technique n'intègrent pourtant que de façon partielle ces échos. Nous commençons par présenter ces différentes formes de dialogues utilisées dans le domaine, et les raisons pour lesquelles les échos n'y sont que peu ou pas intégrés. Par la suite nous analysons les différents problèmes soulevés par l'introduction d'échos proactifs complets en leur sein. Enfin, nous discutons des solutions possibles et présentons un modèle permettant d'obtenir le résultat recherché.

MOTS CLES : Conception Technique, Evaluation de la tâche, Dialogues Structurés, Echos Proactifs.

INTRODUCTION
Les travaux de Norman [11] ont défini un modèle largement accepté aujourd'hui de l'activité humaine face à l'ordinateur. Le raisonnement en buts/sous-buts permet de décomposer un problème complexe en sous-problèmes plus simples. L'application récursive de ce type de raisonnement permet d'arriver au niveau des tâches élémentaires, dont l'exécution est soumise au cycle perception/action. Au cours de ce dernier, le système permet à l'utilisateur de transformer ses objectifs en actions dont les effets lui sont rendus perceptibles afin de lui permettre d'évaluer le plus tôt possible leur adéquation avec le but visé.

La perception correcte et au plus tôt des effets des actions est l'un des points majeurs permettant à l'utilisateur d'éviter de se perdre dans la forêt de ses sous-buts [8, 13]. Cependant, la seule rétroaction ne permet pas d'éviter les erreurs de l'utilisateur, ce qui oblige le plus souvent l'incorporation de mécanismes « Défaire / Refaire » (UNDO/REDO en anglais) dans les systèmes complexes.

Afin d'améliorer la prévention des erreurs, des mécanismes d'échos proactifs (« proactive feedback » en anglais) peuvent être implémentés. Les menus grisés ou les fantômes de la manipulation directe en sont deux exemples qui répondent à deux grands besoins : la prévention automatique et la prévention informative. Les menus grisés sont la représentation visuelle de l'interdiction faite à l'utilisateur d'utiliser des actions non appropriées à un moment donné. Il s'agit d'une *prévention automatique* (et contraignante). À l'inverse, les fantômes de la manipulation directe se contentent de montrer, tout au long de l'action en cours (« Glisser / Déplacer » ou DRAG & DROP), *ce qui se passerait si* l'utilisateur terminait son action : le fichier serait déplacé *ici*, ou bien serait mis dans *la corbeille qui est actuellement mise en évidence*. Cet écho par anticipation joue un rôle d'information auprès de l'utilisateur en lui donnant une idée du résultat de l'action avant son exécution, jouant ainsi un rôle de *prévention informative*.

On remarquera que dans les deux cas, il ne s'agit que de conventions de dialogue, pas forcément en rapport avec l'accomplissement réel de la tâche en cours : ainsi, l'écho du passage d'une icône sur une fenêtre représentant un dossier ouvert est-il la mise en évidence de la fenêtre, et non pas l'insertion simulée du fichier manipulé (ou de son fantôme) à l'endroit où il serait effectivement mis. Cette remarque conduit à la conclusion que ces échos proactifs sont des techniques d'interaction propres au dialogue, et dont la partie sémantique est limitée et conventionnelle. Leur introduction dans un dialogue riche et complexe demande une profonde modification de ce dernier. Il est pourtant clairement établi que ces échos permettent d'améliorer sensiblement la qualité ergonomique des applications interactives [8].

L'objectif de ce travail est de proposer un moyen simple et systématique pour introduire un écho proactif au sein d'une application supportant un dialogue complexe. Notre domaine d'application est la Conception Technique, où l'application des théories de Norman se fait de manière naturelle. Dans un premier temps, nous présentons sommairement ce domaine, et montrons les intérêts et limites des différentes méthodes permettant à

un utilisateur d'atteindre ses objectifs. Puis, nous examinons les principales difficultés de l'insertion d'échos proactifs dans les dialogues structurés que nous utilisons. Enfin, nous discutons des solutions possibles et présentons un modèle permettant d'obtenir le résultat recherché.

STRATÉGIES DE DIALOGUE EN CONCEPTION TECHNIQUE

La conception technique est un domaine où les règles de construction jouent un rôle fondamental. À l'inverse des systèmes de dessins grand public, où le positionnement des objets peut-être approximatif, les systèmes de conception technique fournissent à leurs utilisateurs des modes de construction d'objets par contraintes. Là où un logiciel de la première catégorie permettra de créer une ellipse par deux clics souris, puis de l'agrandir et de la déformer en cercle par manipulation directe, un système de conception technique fournira des primitives distinctes de création d'ellipses et de cercles, basées sur leurs caractéristiques géométriques (centre, rayons). Ce qui pourrait paraître frustre au premier abord (création d'un cercle par centre et rayon) s'avère en fait très puissant, tout en restant parfaitement défini *a priori*, par l'usage intensif des lois géométriques. Ainsi, peut-on créer, dans les systèmes de conception technique, un cercle par deux points (centre et rayon) ou passant par trois points, ce qui autorise toutes les combinaisons de tangences (tangent à une droite, un cercle, un arc, etc.).

L'utilisation de ces actions est la phase ultime du raisonnement en buts/sous-buts du concepteur technique. Prenons l'exemple de la construction suivante :

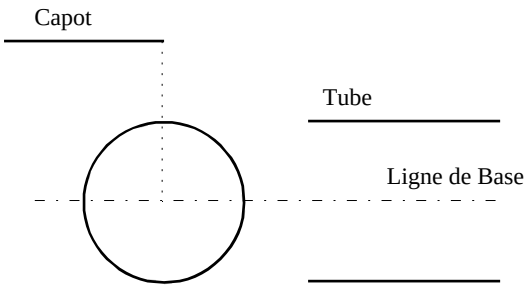

Figure 1 : Exemple de construction mécanique

Il s'agit d'une partie de mécanisme où le cercle (vue deux dimensions d'une poulie) doit pouvoir coulisser dans le tube de droite (symbolisé par les deux segments horizontaux) et est positionné au départ à l'aplomb d'un capot (l'élément supérieur). Le but du concepteur est donc le suivant : « *Création d'une poulie positionnée sur la ligne de base à l'aplomb de l'extrémité du capot, et dont le diamètre est égal au diamètre intérieur du tube* ». Trois grandes approches permettent dans les systèmes de conception technique de résoudre ce problème.

La méthode « table à dessin »

Le travail classique d'un concepteur sur une table à dessin consiste à bouleverser son arbre de buts/sous-buts. Il commence par identifier les entités supports de la construction désirée (droites, cercle, points…). Les entités constituant la construction finale (segments, arcs de cercle…) sont ensuite rajoutés en s'appuyant sur ces entités supports. La stratégie dite « table à dessin » consiste a plaquer cette méthode de travail sur le support informatique. Une application employant cette stratégie de dialogue dispose donc d'un très grand nombre de fonctions de création d'entités.

Dans notre exemple, il faut donc commencer par créer un point, centre du futur cercle, mesurer le diamètre du tube, créer le cercle centré sur le point, et enfin détruire (ou masquer) ce point.

Cette approche, très classique, est utilisée par la quasi-totalité des systèmes existants. Elle respecte les habitudes des utilisateurs (ceux qui sont passés par la table à dessin), mais nécessite la mise en place d'une stratégie très différente de la simple hiérarchie buts/sous-buts initiale. Elle nécessite l'ajout de tâches intermédiaires et de tâches finales (comme celle de nettoyage). L'écho sémantique est correctement effectué pour chacune des sous-tâches intermédiaires, mais aucun écho sémantique de la tâche globale n'est visible avant la fin de la construction. Toute erreur dans la réalisation d'une sous-tâche oblige la révision globale de la stratégie, et ce uniquement à la fin de la tâche globale.

De plus, l'utilisation par l'utilisateur d'une stratégie identique à celle de la table à dessin fait souvent abstraction des potentiels des systèmes informatique, et conduit à des stratégies de dessin sous-optimales [2]. Ces auteurs citent par exemple des surfaces fermées construites sous forme d'un assemblage de segments indépendants, ce qui interdit leur remplissage automatique par un motif, et oblige l'utilisateur à dessiner lui même celui-ci. Ce qui est en cause ici n'est pas la complexité du système de conception, généralement maîtrisé par l'utilisateur, mais bien la décomposition de la tâche par cet utilisateur.

La méthode à « post-contraintes »

L'augmentation de puissance des systèmes informatiques a permis de mettre en œuvre des techniques beaucoup plus séduisantes. L'idée générale consiste à dessiner approximativement, par exemple avec des techniques de manipulation directe, puis à contraindre *a posteriori* le schéma obtenu. Le système est ensuite en mesure d'ajuster le dessin aux contraintes exprimées. Dans notre exemple, les tâches se décomposent en un dessin approximatif du cercle, puis en l'expression des trois contraintes (alignement vertical par rapport à l'extrémité du capot, alignement l'axe du tube, et même rayon que ce dernier).

Cette méthode, qui peut sembler idéale, présente néanmoins quelques problèmes : elle n'est pas si simple à utiliser, car les contraintes exprimées peuvent être insuffisantes (problème sous-contraint, impossible à résoudre) ou trop nombreuses (problème surcontraint, avec possibilités de contradictions) ; de plus, elle est limitée à certains types de problèmes (2D par exemple). Elle est cependant présente dans de nombreux systèmes (I-Deas, Catia, Pro-Engineer...).

On remarquera cependant qu'elle nécessite une transformation complète de l'arbre des buts/sous-buts, et qu'elle cache la méthode de construction qui devient un peu « magique ». De plus, l'approximation de départ faite par l'utilisateur peut en fait s'avérer totalement remise en cause par la résolution du système de contraintes. Il n'est ainsi pas rare de s'apercevoir que, par un mauvais choix de contraintes, le dessin calculé n'a pas de rapport avec la solution recherchée. La notion d'écho sémantique est ici totalement absente.

La méthode par « dialogues structurés »

Le principe de base des dialogues structurés consiste à fournir à l'utilisateur le moyen de calquer au plus près son arbre des tâches sur son arbre de buts/sous-buts. Ceci se fait au moyen d'un dialogue riche et structuré, permettant d'exprimer les paramètres des tâches[1] au moyen d'autres tâches. Ainsi, dans notre exemple, l'arbre des tâches deviendra-t-il la création d'un cercle dont le centre est la projection sur la ligne de base de l'extrémité du segment représentant le capot, et dont le rayon est égal à la distance séparant le haut du tube de la ligne de base (rayon du tube).

Cette forme de dialogue peut être vue comme un ensemble de producteurs et de consommateurs : certaines tâches du système *consomment* des données *produites* par d'autres tâches. Par exemple, la tâche de création de cercle reçoit une partie de ses données de la tâche de calcul de la projection d'un point sur un objet. Ces tâches productrices consomment elles-mêmes des données, en provenance soit d'autres producteurs, soit de l'utilisateur. Celui-ci agit comme le producteur de données initial.

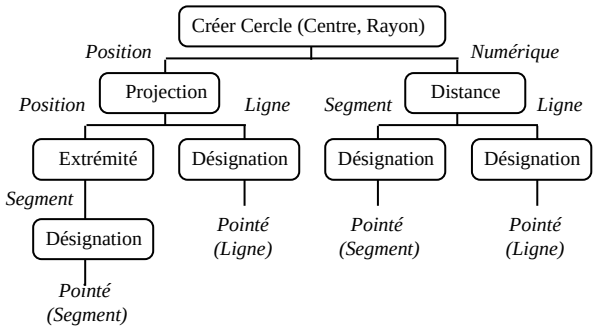

Figure 2 : Transmission de paramètres entre tâches. Les paramètres sont en italiques.

La mise en œuvre de cette méthode nécessite, pour éviter l'explosion combinatoire des agencements de sous-tâches, le développement d'une architecture d'application particulière comme celle décrite dans [6], [4, 5, 9], ou [14]. Elle est utilisée de manière ponctuelle dans quelques systèmes (I-Deas, Catia, Pro-Engineer, Autocad...).

Si cette méthode présente l'avantage de diminuer très fortement la distance sémantique (l'arbre des tâches est très proche de l'arbre des buts/sous-buts), les possibilités d'évaluation sont limitées. En effet, l'écho fourni par ces systèmes se limite à un écho articulatoire (mise en évidence du dernier élément sélectionné) pendant les sous-tâches, éventuellement par un écho de la sous-tâche en cours de plus bas niveau, et par un écho sémantique final de la tâche principale (construction du cercle dans notre exemple) bien tardif et non proactif. Ainsi, la conscience de la situation dans l'arbre des tâches n'est pas évidente, par manque d'écho sémantique, l'écho des sous-tâches est inexistant, et les erreurs ne sont découvertes qu'à la fin de la tâche globale.

La plupart des systèmes de CAO commerciaux n'emploient pas qu'une seule des méthodes présentées ci-dessus. Beaucoup conjuguent la méthode « table à dessin » avec l'une des deux autres. Cette stratégie est en effet la plus simple du point de vue implémentation logicielle, les fonctions n'étant pas liées les unes aux autres comme dans les dialogues structurés, ou aussi complexe que la résolution des équations associées aux dialogues à post-contraintes. Par exemple, Autocad© Version 12 emploie majoritairement la méthode « table à dessin », mais autorise pour certaines fonctions l'emploi de « fonctions d'accroche » pour fournir les points particuliers d'objets (se plaçant alors dans les logiciels utilisant les dialogues structurés). D'autres systèmes tels que Pro-engineer© utilisent « table à dessin » et « post-contraintes ».

Notre but principal étant de respecter la logique de l'utilisateur, nous approfondirons dans les prochaines sections la solution des « dialogues structurés », cette dernière étant la seule pouvant fournir un écho complet

[1] Nous employons tout au long de cet article le terme de « tâche » dans un sens correspondant à des tâches de relativement bas niveau, telles que « crer cercle » ou « calculer intersection »…

de la tâche en cours, et proposerons des solutions pour résoudre les différents problèmes exposés ci-dessus.

PRINCIPES DES DIALOGUES STRUCTURÉS

La mise en œuvre des dialogues structurés nécessite de distinguer au sein du contrôleur de dialogue deux types de tâches : les tâches terminales et les sous-tâches d'expression. Les premières constituent les actions principales de chaque phrase du dialogue ; elles modifient généralement le modèle du noyau fonctionnel (*Créer point, Créer cercle*) mais peuvent également toucher la présentation de l'application (*zoom, changement de point de vue, etc.*). À l'inverse, les sous-tâches de d'expression ont pour but de calculer des expressions afin de mettre le résultat à la disposition d'autres (sous-)tâches. Ces expressions peuvent être de simples accès à des valeurs caractéristiques (Centre de, Extrémité de, etc.), ou bien constituer des expressions géométriques complexes (*Projection de, distance, etc.*). Ces sous-tâches de production effectuent une transformation de leur paramètres d'entrée en paramètres de sortie qui sont eux-mêmes utilisés comme paramètres d'entrée dans la tâche en cours.

Ce principe a été mis en œuvre dans les travaux de L. Guittet ([6, 7]) dont l'intérêt principal réside dans la possibilité de créer des dialogues structurés très riches en évitant l'explosion combinatoire des états du dialogue.

ÉCHO SÉMANTIQUE ET DIALOGUES STRUCTURÉS

Le premier moyen d'obtenir un écho sémantique est d'incorporer de façon systématique dans le dialogue structuré la technique dite de Rubber Banding [3], que l'on peut traduire par « Écho élastique » en français. 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 de son action courante. 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.

Dans le cadre des tâches structurées, ceci n'est pas suffisant pour fournir un écho de la tâche complète. En effet, la position du second point du segment peut n'avoir qu'un lointain rapport avec la position de la souris, et être déterminée par une expression complexe mettant en jeu des sous-tâches de production comme la projection ou l'intersection. Pour obtenir un bon écho sémantique de la tâche globale, il conviendra de prendre en compte cette partie de l'interaction et de présenter un écho de chacune des (sous-)tâches en jeu. L'ensemble de ces échos forme ce que l'on nomme l'exploration de la tâche finale.

But

Si l'on prend comme exemple la tâche permettant de réaliser la figure 1, on constate que l'introduction d'un écho proactif sémantique de la tâche globale se traduit par l'écho des sous-tâches permettant d'aboutir à la définition du centre du cercle final et par l'ajout de l'exploration du but final, la construction du cercle.

Le premier point signifie la matérialisation de la projection du segment sur la ligne de base en un point, centre du futur cercle. Une ligne pointillée peut représenter la projection, tandis que le centre du cercle peut être visualisé comme un point particulier, par exemple sous la forme d'une croix cerclée. Ceci peut être étendu à la deuxième sous-tâche, par la visualisation par une ligne pointillée de la distance dès que possible. Ajouter l'exploration consiste à visualiser le cercle final potentiel durant la dernière interaction élémentaire, c'est-à-dire entre le clic bas et le clic haut de la sélection de la ligne de base.

Notons que l'obtention d'une telle visualisation permet au concepteur d'éviter des erreurs telles que la confusion entre diamètre et rayon par mauvaise sélection du dernier segment. L'écho sémantique lui montre son erreur avant qu'il ne termine sa désignation.

De par la complexité des modèles des applications de conception, il est impossible de déléguer complètement à la couche de présentation du modèle ARCH [1] la visualisation du modèle géométrique technique [12]. Les objets du domaine sont en effet fortement liés, et la présentation d'un objet dépend souvent étroitement des autres objets du modèle. Seul le noyau fonctionnel est en mesure d'assurer convenablement cette visualisation [6]. De ce fait, l'obtention d'un réel écho sémantique, et non d'un simple fantôme de présentation conventionnel, impose que cet écho soit réalisé sous le contrôle direct du noyau fonctionnel (ou de l'adaptateur associé), et non déporté au sein de la présentation comme c'est souvent le cas.

Problèmes

Incorporer un écho sémantique au sein d'une application introduit un certain nombre de problèmes, que cette application utilise un dialogue sous forme simple ou structurée. Parmi ceux ci on trouve celui de la constance du modèle, ainsi que celui de la gestion de l'affichage. En outre, d'autres particularités spécifiques aux dialogues structurés apparaissent lors de cette introduction, comme le problème de la cohérence du dialogue ou celui du contrôle de la fin de tâche.

Constance du Modèle : L'incorporation d'un écho dans une fonctionnalité est globalement réalisé de la même manière que cette fonctionnalité appartienne à un dialogue structuré ou non. Dans un premier temps la fonctionnalité acquiert, de manière cyclique, des

informations temporaires (« fugitives »). La fonctionnalité fournit, à partir de ces valeurs, un écho du résultat de l'action si ces valeurs étaient employées (« écho fugitif »). Dans cette phase, le modèle géométrique n'est pas modifié. Dans une seconde phase, la valeur définitive est acquise. C'est à ce moment que le modèle est modifié pour y incorporer le nouvel objet.

Dans le cas d'un dialogue structuré, nous avons vu qu'il convenait de distinguer les tâches terminales des tâches de production.

Tâches terminales
Les tâches terminales constituent les tâches principales de chaque phase du dialogue ; elles modifient généralement le modèle du noyau fonctionnel (*Créer point, Créer cercle*). Elles consomment des paramètres, mais n'en produisent aucun.

Les fonctions implémentant ces tâches doivent donc avoir un comportement différent selon que les données qu'elles reçoivent sont fugitives ou non. Elles ne doivent en aucun cas modifier le modèle à chaque réception d'un paramètre fugitif, mais uniquement lorsque l'ensemble de leurs paramètres est non fugitif. L'exécution d'une fonction fugitive doit résulter en l'affichage d'un écho fugitif sur la surface d'affichage, sans que le modèle géométrique soit modifié. Cet affichage doit correspondre à un écho de ce qui serait réalisé si le paramètre n'était pas fugitif. Par exemple, dans le cas où il s'agit d'une tâche de création, l'écho fugitif doit avoir l'apparence de l'objet qui serait créé si le paramètre n'était pas fugitif.

Tâches de production
À l'inverse, les tâches de production ont pour but de calculer des expressions afin de mettre le résultat à la disposition d'autres (sous-)tâches. Ces expressions peuvent être de simples accès à des valeurs caractéristiques (*Centre de, Extrémité de*, etc.), ou bien constituer des expressions géométriques complexes (*Projection de, distance, etc.*). Ces sous-tâches de production transforment les paramètres d'entrée qui leur sont fournis en paramètres de sortie qui sont eux-mêmes utilisés comme paramètres d'entrée dans la tâche en cours.

Ces tâches de production ne modifient jamais le modèle. Elles agissent de fait comme une transformation : la tâche consomme un ou plusieurs paramètres pour en produire un autre (par exemple une tâche d'intersection consomme deux objets pour fournir une position). Une telle transformation est indépendante du caractère fugitif ou non des paramètres : une même désignation fournit dans tous les cas le même objet ; le centre d'un objet reste le même que cet objet soit fugitif ou non... Une tâche de production se comporte donc de la même manière, que les valeurs transmises soient définitives ou non.

Dans les deux cas, une tâche ne pouvant s'exécuter que lorsque tous ses paramètres lui sont fournis, il en résulte que le comportement fugitif d'une tâche 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 tâche, le paramètre fugitif sera le dernier dans l'ordre chronologique.

Gestion de l'Affichage : La présence d'un écho dynamique au sein d'une fonctionnalité implique qu'à chaque acquisition d'une donnée fugitive, cette fonctionnalité doive, avant d'afficher l'écho du résultat courant, effacer celui associé à la valeur précédente, s'il existe.

Ceci peut être réalisé de plusieurs manières : la fonction peut, par exemple, disposer d'un état et d'une mémoire associée lui permettant de savoir si elle a déjà produit un écho, et les caractéristiques de cet écho. Cette solution, simple à mettre en œuvre, est relativement lourde en ce sens que toute fonctionnalité doit alors disposer d'un tel couple (état, mémoire). En outre, dans le cas d'un dialogue structuré, cette fonction ne peut être employée simultanément qu'à un seul niveau de l'arbre des tâches. Une autre solution consiste à fournir au développeur du système un composant spécialisé, auquel le contrôleur de dialogue a un accès privilégié, pour réaliser les échos fugitifs. C'est alors ce contrôleur de dialogue qui prendra à sa charge l'effacement des échos résultant des appels précédents à la fonction. Ces deux possibilités seront étudiées, dans la section « Solutions ».

Les deux points ci-dessus concernent tous les dialogues. Les points suivants sont particuliers aux dialogues structurés.

Cohérence du dialogue : L'exploration au sein d'un dialogue structuré peut être vue comme la pseudo-exécution de la fonctionnalité terminale : tous les paramètres sont fournis, mais seul l'écho de l'action est réalisé. Du point de vue du dialogue, cela suppose que ce dernier n'évolue pas. Ceci doit être étendu à toute la hiérarchie des sous-tâches en cours. Un écho fugitif comportant un nombre quelconque d'appels (jusqu'à ce que l'utilisateur relâche la souris par exemple), les paramètres non fugitifs doivent être conservés entre chaque appel de la fonctionnalité 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. L'appel d'une fonctionnalité avec des paramètres fugitifs ne doit pas modifier le dialogue. Ceci implique également qu'une fonctionnalité utilisant une donnée fugitive ne doit pas consommer (détruire) les paramètres qui lui sont

transmis. Ceux-ci doivent rester valides après appel de cette fonctionnalité.

Transmission du caractère fugitif : Dans les dialogues structurés, les paramètres sont transmis de tâche en tâche, et interprétés au sein de différentes fonctionnalités. Si une fonctionnalité recevant un paramètre fugitif retournait un paramètre non fugitif, la tâche de niveau supérieur en attente de ce dernier exécuterait sa fonctionnalité définitive, 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 données (fugitives), toutes les tâches actives doivent recevoir des paramètres fugitifs, quelle que soit leur position dans cette hiérarchie. Tout paramètre produit par une fonctionnalité à l'aide de paramètres fugitifs est dont lui même fugitif. Ceci peut être automatiquement assuré par le contrôleur de dialogue.

Persistance de l'écho et contrôle de la fin de tâche : Nous avons précédemment vu qu'au sein d'un dialogue standard, il n'existe qu'une seule fonctionnalité active à un instant donné, et donc qu'un seul écho. En outre cet écho n'apparaît que lorsque le dernier paramètre est en cours de saisie. Au sein d'un dialogue structuré au contraire, plusieurs fonctionnalités sont actives à un instant donné, et peuvent chacune fournir leur écho.

En outre, Ces échos sont disponibles bien avant le dernier paramètre de la tâche terminale, à savoir les échos des sous-tâches de production ayant fourni les premiers paramètres de cette tâche.

Ces échos doivent rester visualisés même après la terminaison de la sous tâche de production qui les a produits, et ce afin de fournir à l'utilisateur un rappel visuel des informations déjà transmises à la tâche finale. Ces échos doivent perdurer jusqu'à la fin de la cette dernière, moment où ils doivent disparaître. Il convient donc pour un retour d'information maximum vers l'utilisateur, de contrôler l'effacement des échos présents à l'écran lorsque la tâche terminale s'achève.

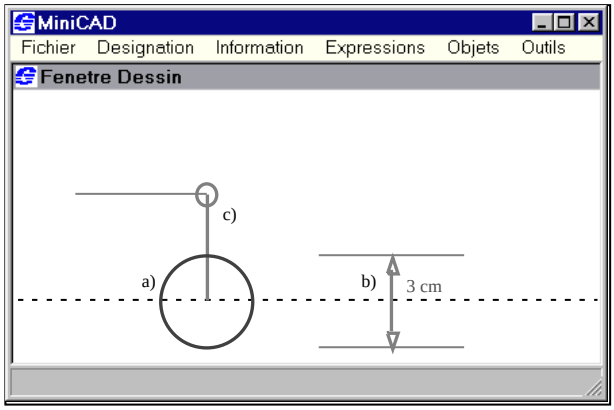

Figure 3 : échos d'une tâche terminal accompagné des échos des sous-tâches ayant produit les paramètres.

SOLUTIONS

L'introduction de l'exploration dans les dialogues structurés suppose le respect par les différents composants des règles présentées dans la section précédente. Ces règles s'appliquent principalement au niveau de la tâche, et montrent une forte différence entre les comportements des deux types de tâches présents dans les dialogues structurés. Nous étudierons donc dans un premier temps leur comportement dans le cadre de l'exploration. Dans un second temps, les différentes solutions pour la gestion de l'effacement des échos fantômes seront étudiées.

Analyse des Tâches

L'examen des règles générales pour l'introduction d'une exploration dans les dialogues structurés conduit à la mise en évidence d'une dichotomie très forte entre les deux types de tâches (production et terminale). Deux points particulièrement les différencient : d'une part la différence de réaction entre ce que fait la tâche lorsque l'un des paramètres est fugitif et lorsque aucun des paramètres ne l'est, et d'autre part le comportement lors de la validation des paramètres d'une tâche, c'est-à-dire lorsque l'on passe d'un paramètre fugitif au même paramètre non fugitif.

Tâches de Production : Pour les tâches de production, il n'existe aucune différence de comportement entre la fonction lorsqu'un paramètre est fugitif et lorsque aucun paramètre ne l'est. Dans les deux cas, comme on l'a vu il est possible d'extraire ou de calculer les informations demandées. L'application de la persistance d'écho vue plus haut implique aussi que même lorsqu'une telle tâche reçoit un paramètre final, elle réalise un écho. La réaction dans les deux cas est virtuellement identique.

Tâches Terminales : En ce qui concerne les tâches terminales, on peut voir que ce que l'utilisateur considère comme une seule tâche (« Créer Cercle ») est vu par le système comme un ensemble de deux fonctions : une fonction « Créer Cercle Temporaire », qui est appelée tant qu'au moins un des paramètres est de nature fugitive, et qui « fait semblant » de créer un cercle en affichant un écho du futur cercle, et une fonction « Créer Cercle Réel », appelée lorsque tous les paramètres sont validés, qui crée réellement le cercle dans le modèle géométrique, et l'affiche. Lors de la validation des paramètres de la fonction, la règle de persistance d'écho impose à la fonction le nettoyage des échos des sous-tâches, celles-ci s'étant terminées sans effacer les leurs. Lors de l'exécution d'une tâche avec des paramètres non fugitifs, la tâche commence donc par effacer tous les échos : ceux résultant d'une éventuelle exécution avec un paramètre fugitif, tout comme ceux résultant de l'exécution d'une (ou plusieurs) sous-tâche de production.

Gestion des Échos

L'un des points communs aux deux formes de tâches concerne la gestion des échos fugitifs. Dans les deux cas elles doivent, avant d'afficher l'écho du résultat courant, effacer celui associé à la valeur précédente, s'il existe. Ceci peut être réalisé de plusieurs manières.

Mémoire et État : La manière la plus simple consiste à associer à chaque fonction un état lui indiquant si elle est en cours d'écho dynamique ou non, et une mémoire référençant les échos déjà réalisés.

Lorsque la fonction reçoit un paramètre fugitif, elle vérifie son état et modifie son comportement en conséquence : Si elle n'est pas en cours d'écho dynamique, elle modifie cet état, effectue son calcul et affiche son écho. Les références de cet écho sont stockées dans la mémoire de la fonction. Si elle était déjà en cours d'écho dynamique, elle efface les échos stockés dans la mémoire associée, effectue son calcul, et réalise ses nouveaux échos, lesquels sont stockés à leur tour dans la mémoire.

Lorsque la fonction reçoit un paramètre non fugitif, son comportement varie selon qu'elle est une fonction associée à une tâche de production ou à une tâche terminale. Dans le premier cas, elle fait comme si le paramètre était fugitif, mais se replace à la fin en mode « hors écho dynamique ». Dans le second cas, la fonction efface tous les échos, et crée ou modifie l'objet concerné, puis le réaffiche.

Cette solution, si elle a le mérite d'être conceptuellement simple, présente cependant un certain nombre de désavantages. Elle oblige notamment à associer à chaque fonction deux variables externes (globales) puisque devant conserver leurs valeurs entre chaque appel. Une telle multiplication de variables globales va à l'encontre des préceptes du génie logiciel. En outre, certains logiciels de CAO contiennent plusieurs milliers de fonctions (AutoCAD par exemple en contient dans sa dernière version plus de 2000 [2]). Se pose alors la question d'une part de l'efficacité et d'autre part de la maintenance d'un tel code.

Groupes d'échos : Une autre possibilité consiste à placer la gestion de l'effacement des échos en dehors des fonctions. Le contrôleur de dialogue dispose en effet, au moment où il demande à une fonction de s'exécuter (que ce soit avec ou sans paramètre fugitif), des mêmes informations que cette fonction. Il sait notamment si le paramètre qu'il transmet est fugitif ou non (ceci est rendu obligatoire par la règle qui oblige l'état du dialogue à rester constant dans le cas d'un paramètre fugitif). De même il sait si la fonction qu'il appelle est terminale ou non (dans le second cas, elle est sensée retourner un paramètre).

Si ce dialogue disposait d'un moyen d'effacer les échos réalisés précédemment par les fonctions, il lui serait possible de prendre en compte cette portion de la gestion, et de simplifier d'autant l'écriture des fonctions. Celles-ci n'auraient plus alors qu'à effectuer leurs calculs et afficher leurs échos. La gestion des effacements des échos des éventuels appels précédents serait ainsi reportée dans le contrôleur de dialogue.

Pour ce faire, il faut que les échos soient réalisés à travers un composant spécialisé auquel ont accès aussi bien les fonctions (pour pouvoir afficher les échos), que le contrôleur de dialogue (pour pouvoir les effacer).

D'un point de vue informatique, l'utilisation de cette technique facilite l'implémentation des fonctions. Ces dernières n'ont pas à s'occuper d'effacer un écho résultant d'un éventuel appel précédent, et n'ont donc pas d'états associés. Une fonction destinée à produire le centre d'un objet, par exemple, se contente de calculer la position de ce centre et de réaliser un écho. On obtient ainsi une gestion aisée des échos, sans avoir à coder cette gestion dans les fonctions. En contrepartie de ces avantages, cette méthode force le modèle (les fonctions) à utiliser un module spécialisé pour réaliser les échos. En outre, cette gestion étant assurée au niveau du contrôleur de dialogue, elle est constante sur toute l'application, et ne nécessite aucun codage particulier dans les fonctions autres que l'utilisation du composant spécifique pour l'affichage des échos.

Implémentation

La structure décrite permet de réaliser un système où toute tâche dispose d'un retour d'information sémantique lors de sa réalisation, et ce quel que soit le nombre de sous-tâches entrant en jeu. En outre, cet écho étant géré au niveau de chaque (sous) tâche, on dispose d'un écho sémantique multi-niveau. Dans notre exemple de construction de poulie, l'utilisateur dispose, de manière dynamique lors de la saisie du segment, à la fois de l'écho de la sélection du segment, de l'écho de l'extrémité qui en découle, de l'écho de la projection, et finalement de l'écho du cercle qui serait créé si la souris était relâchée à cet endroit.

De fait l'utilisateur dispose d'une série d'échos qui l'informe en temps réel sur le quoi (le résultat potentiel de son action, tel qu'indiqué par l'écho de la tâche finale), mais aussi sur le contexte actuel (la suite d'interprétation par le système des entrées, formée par les échos des sous tâches employées). Le retour d'information fournit donc à l'utilisateur l'interprétation par le système de ses entrées.

CONCLUSION

Les différentes définitions présentées ont été validées par la réalisation d'un logiciel de conception technique. 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 une 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 [10]).

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 du dialogue.

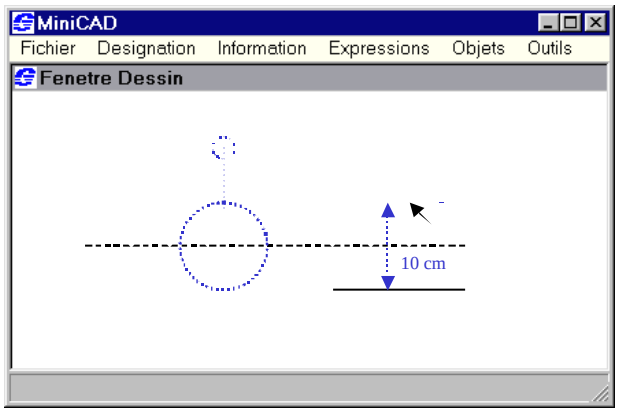

Figure 4 : Contexte d'exécution tel que perçu par l'utilisateur.

L'ensemble des problèmes exposés initialement n'est cependant pas complètement résolu. En effet, si l'utilisateur peut avoir une meilleur appréciation de la tâche en cours, il ne peut revenir en arrière que durant une même arborescence de sous-tâche. Il peut modifier l'entrée en cours, mais non celles qu'il a déjà réalisées. Dans notre exemple lorsque la définition du centre du cercle est effective, on ne peut plus que la réfuter totalement.

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. Le portage d'une application industrielle sur ce noyau est en cours. Par la suite, l'adjonction d'un noyau de manipulation directe au système permettra de suppléer 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. Bass, L., Faneuf, R., Little, R., Mayer, N., Pellegrino, B., Reed, S., Seacord, R., Sheppard, S. et Szczur, M.R. A Metamodel for the Runtime Architecture of an Interactive System. *SIGCHI Bulletin*. 24, 1 (1992), pp. 32-37.
2. Bhavnami, S.K. et John, B.E. Exploring the Unrealized Potential of Computer Aided Drafting. In *Proceedings of CHI'96* (13-18 April, Vancouver), Addison-Wesley Publishing Comp., 1996, pp. 232-239.
3. Foley, Dam, v., Feiner et Hughes. *Computer Graphics, Principles and Practice*. Addison-Wesley Publishing Comp., 1990.
4. Gardan, Y., Jung, J.-P., Kolopp, J.-N., Minich, C. et Totino, W. Une approche nouvelle de la convivialité dans un système de CAO : les principes de dialogue dans SACADO. In *Proceedings of MICAD* Parsi, 21-25 Mars), 1988, pp. 281-296.
5. Gardan, Y., Jung, J.-P. et Martin, B. An End-User oriented approach to design man-machine interface for CAD/CAM. In *Proceedings of IEEE Internationel Conference on Systems, Man and Cybernetics* Le Touquet, France, 17-20 Octobre 1993), 1993, pp. 525 à 530.
6. Guittet, L. *Contribution à l'Ingéniérie des Interfaces Homme-Machine - Théorie des Interacteurs et Architecture H4 dans le système NODAOO*. Thèse de Doctorat : Université de Poitiers, 1995.
7. 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.
8. Hix, D. et Hartson, H.R. *Developping user interfaces: Ensuring usability through product & process*. John Wiley & Sons, inc., Newyork, USA, 1993.
9. Martin, B. *Contribution pour une nouvelle Approche du dialogue Homme-Machine en CFAO*. Thèse de Doctorat : Université de Metz, 1995.
10. Miller, J.-G. *The magic number Seven : somes limits in our capacity for processing information.* 1956.
11. Norman, D. *User Centered System Design*. Lawrence Erlbaum Associates, 1986.
12. Pierra, G. Towards a taxonomy for interactive graphics systems. In *Proceedings of Eurographics Workshop on Design, Specification, Verification of Interactive Systems* (June 7-9, Bonas), Springer-Verlag, 1995, pp. 362-370.

13. Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S. et Carey, T. *Human-Computer Interaction*. Addison Wesley Publishing Company, Wokingham, England, 1994.

14. Qiang, L., Wei, L., Ke, X. et Jiaguang, S. An Event-Driven and Object Oriented FrameWork for Human Computer Interface of CAD System. In *Proceedings of CAD & Graphics'97* (2-5 Dec., Shenzen, China), International Academic Publishers, 1997, pp. 42-45.