Un Mécanisme Hiérarchique de Répétition et Prédiction de Tâches

Solange Karsenty-Aflalo et François Pachet
LAFORIA - Institut Blaise Pascal - Université Pierre et Marie Curie
4, Place Jussieu 75252 Paris Cedex FRANCE
Tel: 33 - 1 - 44 27 47 21 email:{sk,pachet}@laforia.ibp.fr

RÉSUMÉ

Dans cet article nous présentons un mécanisme général de *répétition* et *prédiction* basé sur une analyse des actions de l'utilisateur. Ce mécanisme requiert l'existence d'arbres de tâches qui décrivent les plans d'action typiques de façon hiérarchique, correspondant à divers niveaux d'abstraction d'une tâche donnée. Nous proposons un *mécanisme d'espionnage* qui permet d'observer les actions de l'utilisateur sans modifier le code source de l'application. Ce mécanisme peut se brancher sur n'importe quelle application Smalltalk. Un analyseur interprète les interactions espionnées afin de produire une interprétation des actions de l'utilisateur, sous la forme d'un *arbre de tâche partiellement instancié* (ATPI). Puis nous proposons d'utiliser ces ATPI pour suggérer à l'utilisateur des opérations de *répétition* et *prédiction* complexes et significatives. Ainsi, en interagissant à travers un ATPI, l'utilisateur peut interroger le système pour des conseils prédictifs, ou pour terminer une tâche en cours.

MOTS CLÉS : macro dynamique, analyse cognitive de tâche, répétition, interface prédictive, espion, arbre de tâches.

INTRODUCTION

Améliorer l'interaction homme-machine consiste entre autres à rendre plus efficaces les actions de l'utilisateur. Or, quelle que soit l'application utilisée, les actions de l'utilisateur présentent souvent un caractère répétitif qui engendrent une lassitude au fur et à mesure que l'utilisateur acquiert de l'expertise. C'est pourquoi plusieurs techniques ont été proposées pour la construction de macro-commandes, la prédiction ou la répétition [1,2]. Ces techniques ont pour but de factoriser les actions répétitives et construire ainsi des commandes composées à partir des interactions de base de l'application. L'utilisateur peut alors re-exécuter une séquence d'actions à l'aide d'une seule commande.

On constate que les systèmes proposant de telles techniques ont tendance à considérer un seul niveau d'abstraction correspondant aux interactions de base de l'application : ainsi la répétition a lieu au niveau des touches clavier dans [2]. Notre approche consiste à construire un mécanisme intelligent de répétition qui prenne en compte non seulement les interactions de base, mais aussi les niveaux plus abstraits de la tâche qui se décompose récursivement en sous tâches. L'utilisateur doit alors pouvoir re-exécuter une tâche abstraite.

Une seconde caractéristique des systèmes existants est qu'ils proposent de tel mécanismes pour des applications spécifiques. Notre approche consiste à présenter un mécanisme à usage général : celui-ci doit s'appliquer à des applications variées, et doit être implémenté de façon souple et solide. En particulier, nous visons à ajouter cette fonction pour des applications *existantes*, de manière a faciliter le prototypage.

Enfin, les mécanismes indépendants de la tâche proposent généralement un ensemble de *méta-commandes* telles que les touches *répétition* et *prédiction* [2]. Ils montrent rapidement des limitations dues au fait que le système ne peut proposer des opérations complexes de répétition et prédiction qui correspondent aux intentions de l'utilisateur. Nous affirmons que la répétition et prédiction d'action sophistiquée et intelligente nécessite (1) une technique d'analyse qui produit une interprétation de la tâche en cours, (2) une présentation explicite à l'utilisateur de la tâche en cours. En effet, c'est en donnant accès à l'utilisateur à la représentation de ses intentions par le système que nous lui permettons de choisir le niveau d'abstraction de la tâche pour les opérations de *répétition* et *prédiction* .

OBSERVATION DES ACTIONS UTILISATEUR SANS MODIFICATION DU CODE SOURCE

La nécessité de pouvoir réutiliser des applications existantes est maintenant une chose vitale. Les mécanismes traditionnels de réutilisation de code se basent sur les techniques de programmation telles que l'héritage, la généricité et l'encapsulation. Celles-ci demandent de la part des concepteurs une compréhension profonde de la structure du programme réutilisé, et le forcent à suivre des schémas de conception préétablis. Ainsi la quantité de travail de programmation nécessaire pour ajouter aux applications de telles fonctionnalités dissuade le concepteur de construction d'extensions intelligentes. Un exemple frappant de cette difficulté à construire des systèmes d'analyses au dessus d'applications existantes est décrit dans [3], qui a nécessité la réécriture d'un émulateur de l'application hôte (WordPerfect).

Pour éviter d'avoir à modifier du code existant, nous avons développé une technologie permettant de brancher automatiquement un système d'observation et d'analyse sur

une application existante. Cette technique est basée sur la notion *d'espion*. Un espion est un objet capable d'intercepter des messages parvenant à un objet quelconque d'une application hôte. La technique d'espionnage exploite les propriétés de réflexivité du langage de programmation [4], et en particulier le mécanisme de capsules par redéfinition des erreurs [5], ainsi que *l'accès à la pile d'exécution* et les facilités d'échange *d'identités d'objets* [6].

Une première version opérationnelle a été réalisée en Smalltalk-80. L'extension de ce mécanisme vers d'autres langages à objets (C++, CLOS) est en cours d'étude, ainsi qu'un schéma de communication pour espionner des applications à distance, ceci à l'aide du protocole de communication RPC. Les espions peuvent filtrer les messages à intercepter selon un ensemble de messages prédéfinis qui sont d'intérêt pour le système de contrôle. Lorsqu'un espion intercepte un message, il construit un *objet d'interaction* comprenant l'émetteur, la date et les arguments du message, puis il le transmet au système d'analyse.

ANALYSE DES INFORMATIONS ESPIONNEES

Le mécanisme d'espionnage décrit ici permet de collecter les informations circulant entre les objets d'une application hôte (les messages), sans avoir à modifier son code. Ces informations brutes doivent être ensuite analysées afin d'inférer les intentions de l'utilisateur, par rapport à un ensemble de tâches prédéterminées.

Le problème de l'analyse de plans d'actions a été étudié depuis longtemps. Les approches à base de grammaire et d'analyse syntaxique [7], [8], [9] sont historiquement les premières à avoir été proposées, mais leur utilisation reste problématique (voir à ce sujet la comparaison de [3]). Du point de vue théorique, les travaux de [10], basée sur la circonscription résout le problème dans sa généralité, mais nécessite une connaissance préalable de la librairie complète des plans d'actions; et son algorithmique n'est pas complète [11].

Notre position est pragmatique, et consiste à proposer une architecture modulaire permettant de choisir le modèle d'analyse de plan d'actions le plus adapté au domaine utilisé. Nous avons proposé dans [12] un modèle général, baptisé architecture épiphyte par analogie avec la botanique. Dans ce modèle, les informations espionnées sont analysée par une procédure bottom-up, décrite dans la section suivante. C'est dans ce cadre que nous avons développé le système présenté ici.

LES ARBRES DE TÂCHES

Notre système nécessite l'existence d'arbres de tâches, tels que ceux que nous fournit l'analyse cognitive de tâches. Un arbre de tâches est une décomposition hiérarchique d'une tâche en sous-tâches pour une application donnée. La représentation sous forme d'arbre explicite la relation de composition entre sous-tâches. L'arbre est par ailleurs décoré : chaque tâche contient des spécifications de *contraintes de précédence* entre les tâches [13]. Par exemple, la tâche *new_frame* (Figure 1) doit être exécutée avant n'importe quelle autre sous-tâche de *make_dialogBox*. Enfin, les feuilles de l'arbre représentent les actions atomiques de l'application, soit le niveau d'abstraction le plus bas. Ces feuilles contiennent des spécifications des objets et messages correspondant aux actions qu'elles décrivent. Ces informations sont ensuite utilisées pour établir la connexion entre le système d'espionnage et le système d'analyse.

La figure 1 montre l'arbre de tâche d'une session typique dans VisualWorks, le générateur d'interface du langage ObjectWorks Smalltalk-80. Cet arbre est le fruit d'une analyse [14] qui contient une série de plans typiques d'action pour l'utilisation de l'environnement VisualWorks. **Il est construit interactivement grâce à éditeur d'arbres spé**cialisé (voir Figure 1 en annexe). Un tel arbre est ensuite utilisé pour identifier les actions de l'utilisateur. Bien entendu, il est nécessaire de construire un ensemble d'arbres de tâches qui couvre les actions les plus fréquentes. Plus cet ensemble est exhaustif et bien fondé, plus le système d'analyse sera pertinent. Les arbres sont ainsi regroupés dans une *librairie d'arbres de tâches*.

L'ANALYSE DES SÉQUENCES D'ACTIONS : LES ATPI

Chaque action de l'utilisateur se traduit un envoi de message dans l'application hôte. Les messages sont interceptés par les espions puis ils sont transmis à *l'analyseur d'actions*. Le but de cet analyseur est de produire une représentation des intentions de l'utilisateur. Cette représentation est modifiée chaque fois qu'une interaction est détectée par le mécanisme d'espionnage. Nous décrivons ici succinctement la procédure d'analyse.

Les séquences d'actions de l'utilisateur sont analysées par rapport à la librairie d'arbres de tâches. Le résultat de l'analyse est un ensemble (éventuellement vide) d'arbres de tâches partiellement instanciés (ATPI). Un ATPI est une instanciation partielle d'un arbre de tâches reconnu de la librairie. Chaque instanciation contient des informations comme les arguments et les dates auxquelles les actions ont été effectuées.

Notre procédure d'analyse est décrite en détails dans [15]. Elle tient de la même inspiration que [16], et repose sur un parcours récursif des arbres de la librairie à partir des feuilles. Le principe est le suivant:

- A chaque arbre de tâches est associé une structure isomorphe, appelée arbre des conseillers. Cet arbre est utilisé pour l'analyse des actions, et la mise à jour d'un modèle de l'activité en cours.
- A chaque fois qu'un espion intercepte un message jugé intéressant, il le transmet au conseiller terminal correspondant dans l'arbre des conseillers. Celui-ci effectue alors un processus d'analyse récursif, qui consiste à 1) mettre à jour son modèle de l'activité en cours (gestion des états courants de la tâche), et 2) transmet à son père dans l'arbre des conseiller un *signal* portant des informations utiles au conseiller père. Ce signal est alors interprété par le père par une procédure analogue (mise à jour du modèle local au niveau du père, et transmission au grand-père) et ainsi de suite, jusqu'à arriver au noeud racine de l'arbre.

Cette procédure a pour résultat de produire un - ou plusieurs - ATPI (voir Figure 2 pour un exemple d'ATPI). Ces ATPI sont alors utilisés comme éléments d'interface

principaux pour proposer les opérations de répétition et prédiction.

L'ATPI: UNE INTERFACE UTILISATEUR POUR LA RÉPÉTITION ET LA PRÉDICTION

Plutôt que de fournir à l'utilisateur des menus prédéfinis pour répéter certaines opérations, nous lui donnons une vue sur les ATPIs, à partir desquels il peut exécuter les actions de *répétition* et de *prédiction* tout en choisissant le niveau d'abstraction de l'action.

L'utilisateur a deux manières d'agir à partir de cette structure. Après avoir sélectionné un noeud de l'arbre choisi, il peut choisir l'une des deux commandes: *répéter* ou *prédire*. La *répétition* re-exécute la séquence d'actions désignée par le noeud. Si le noeud est terminal, seule l'action correspondante est re-exécutée avec les mêmes arguments que ceux enregistrés. Si le noeud est à un niveau intermédiaire, représentant une tâche de plus haut niveau, alors toutes les actions comprises sont re-exécutées (cf. par exemple la tâche *makeDialogBox*).

L'opération de *prédiction* est en quelque sorte une opération de *prédiction* généralisée [2] qui est particulièrement utile pour les tâches non terminées. Cette opération suggère à l'utilisateur une séquence d'actions nécessaire pour terminer la tâche sélectionnée, selon les sous-tâches déjà effectuées. Bien sûr, la pertinence de la séquence d'actions proposée par le système dépend fortement des résultats obtenus par l'analyseur d'actions.

EXEMPLE

Nous avons expérimenté notre méthode de construction sur le constructeur d'interfaces du système VisualWorks. Grâce à notre système d'espionnage, la construction d'un mécanisme intelligent de répétition n'a requis aucune modification du code de VisualWorks. A titre d'exemple, l'arbre de tâche Figure 1 décompose la tâche racine "build_interface" en plusieurs sous-tâches, elles-mêmes décomposées en actions terminales telles que "Move object", "Create button" ou "Edit label". Les espions sont automatiquement installés sur les objets de VisualWorks, interceptant les messages adéquats et permettant l'analyse par le système d'analyse d'actions.

Une des sous-tâches fréquemment effectuées dans VisualWorks consiste à définir une boîte de dialogue typique comprenant les boutons "OK" et "CANCEL", ainsi qu'un message d'alerte. Supposons que l'utilisateur a successivement effectué les actions suivantes: "new frame", "create button", "button label" avec le paramètre "OK". L'analyseur construira alors un ATPI tel que celui Figure 1, puis il reconnaîtra que la sous-tâche (abstraite) "Button(OK)" a été réalisée. Le sous-arbre correspondant est alors mis en évidence dans l'interface de présentation. L'utilisateur peut alors cliquer sur le noeud "fill dialog box" puis sur le bouton "prédire", et le système exécutera les opérations manquantes, avec des paramètres par défaut pour le message et le bouton ("CANCEL").

Ainsi l'utilisateur peut choisir de répéter soit des séquences de commandes de base correspondantes aux feuilles de l'arbre, ou bien des commandes plus abstraites correspondantes aux *noeuds intermédiaires* d'un ATPI.

Bien entendu cet exemple illustre une situation idéale : l'intention de l'utilisateur correspond exactement à un et un seul arbre prédéfini (ici la fabrication d'une boîte de dialogue). Néanmoins nous affirmons que l'on peut, grâce à ce système, construire incrémentalement un ensemble d'arbres de tâches prédéfinis, qui couvre les plans d'actions les plus typiques, les résultats d'analyse cognitive, ou bien en analysant certains "cook-books" appropriés tels que [14].

CONCLUSION

Nous avons présenté une technique qui peut être appliquée à des applications existantes sans modifier leur code source, et qui fournit à l'utilisateur une vue sur l'interprétation par le système de ses actions, selon un ensemble d'arbres de tâches typiques prédéfinis.

Etant donné que l'implémentation est aisée, cela facilite le prototypage en utilisant des arbres incomplets qui peuvent être enrichis au fur et à mesure de l'utilisation du système. En donnant à l'utilisateur un accès explicite à la représentation de la tâche en cours, nous lui permettons d'effectuer des opérations sophistiquées de *répétition* et *prédiction*. Cet accès réduit la distance entre l'utilisateur et le système et améliore l'interaction: l'utilisateur est mieux conscient de l'état du système et le modèle conceptuel qu'il se fait du système est plus proche de la réalité.

Notre système est encore en cours de développement et s'est montré utile pour un nombre limité d'arbres de tâches prédéfinis. Cependant, parce que notre mécanisme d'espionnage nous permet d'étendre incrémentalement la librairie d'arbres prédéfinis sans modifier le code source de l'application, l'extension des fonctions de *répétition* et *prédiction* reste légère, et contribue à améliorer les performances de l'utilisateur sans affecter l'application hôte. D'autre part, l'architecture proposée garantie l'indépendance entre le mécanisme d'espionnage et celui d'analyse. Un des travaux en cours consiste à étudier le couplage de notre mécanisme d'espionnage avec d'autres systèmes d'analyse, et en particulier le modèle des hieractors de [17].

BIBLIOGRAPHIE

1. Kurlander, D. Feiner, S. A History-Based Macro by Example system. Proceedings of ACM Symposium on User Interface Software Technology (UIST'92), pp. 99-106, (1992).
2. Masui, T. Nakayama, K. Repeat and Predict - Two Keys to Efficient Text Editing. Human Factors in Computing Systems, Proceedings of ACM CHI'94, pp. 118-123, (1994).
3. Desmarais, M. C. Giroux, L. Larochelle, S. An Advice-Giving Interface Based on Plan-Recognition and User-Knowledge Assessment. *Int. Journal of Man-Machine Studies* (1993) 39, pp. 901-924. (1993).
4. Foote, B. Johnson, R.-E. Reflective facilities in Smalltalk-80. *Proceedings of OOPSLA'89*, pp. 327-336, New Orleans, Louisiana, (1989).
5. Pascoe, G. Encapsulators: A New Software Paradigm in

Smalltalk-80. *Proceedings of OOPSLA'86*, pp. 341-346, Portland, Oregon, (1986).
6. Pachet, F. Wolinski, F. Giroux, S. Spying as an Object-Oriented Programming Paradigm. Proceedings of TOOLS Europe '95, Versailles, France, Prentic-Hall, pp. 109-118, (1995).
7. Reisner, P. Formal Grammar and Human Factors Design of an Interactive Graphic System. IEEE Transactions on Software Engineering, Vol. SE-7/2, pp. 229-240.
8. Payne, S. J. Green, T.R.G. Task-Action grammars - A model of the mental representation of task languages. Human-Computer Interaction, Vol. 2, pp. 93-133.
9. Hoppe, H.U. Task-Oriented Parsing - A Diagnosis Method to be used by Adaptative Systems. Proceedings of CHI '88, Washington, D.C., ACM Eds, pp. 241-247 (1988).
10. Kautz, H.A. Allen, J. F. Generalized Plan Recognition. *Proceedings of AAAI '86*, Philadelphia, Pa. pp, 32-37, (1986).
11. Kautz, H.A. A circonscriptive theory of plan recognition, in *Intentions in Communications*, P. Cohen, J. Morgan and M. Pollack, eds., MIT Press, (1990).
12. Pachet, F. Giroux, S. Building plan recognition systems on arbitrary applications: the spying technique. Working notes of IJCAI 95 worklshop on "new generation of plan recognition systems" .
13. Pachet, F. Djamen, J.-Y. Frasson, C. Kaltenbach, M. Production de conseils pertinents exploitant les relations de composition et de précédence dans un arbre de tâches. *Submitted to Technique et Sciences Educatives* (1995b).
14. Visual Works Cookbook, version 2.0. ParcPlace Systems, (1994).
15. Paquette, G. Pachet, F. Giroux, S. EpiTalk, a Generic Tool for the Development of Advisor Systems. Artificial Intelligence in Education, (1995). To appear.
16. Quast, Laus-Jürgen. Plan Recognition for Context-Sensitive Help. International Workshop on Intelligent User Interface '93. Orlando, Florida, pp. 89-96, January, (1993).
17. Kosbie, D. Myers, B. Extending Programming By Demonstration With Hierarchical Event Histories. *Proceedings of East-West Human Computer Interaction Conference 94,* pp. 128-139, Springer-Verlag, Lecture Notes in Computer Science, n. 876 (1994).

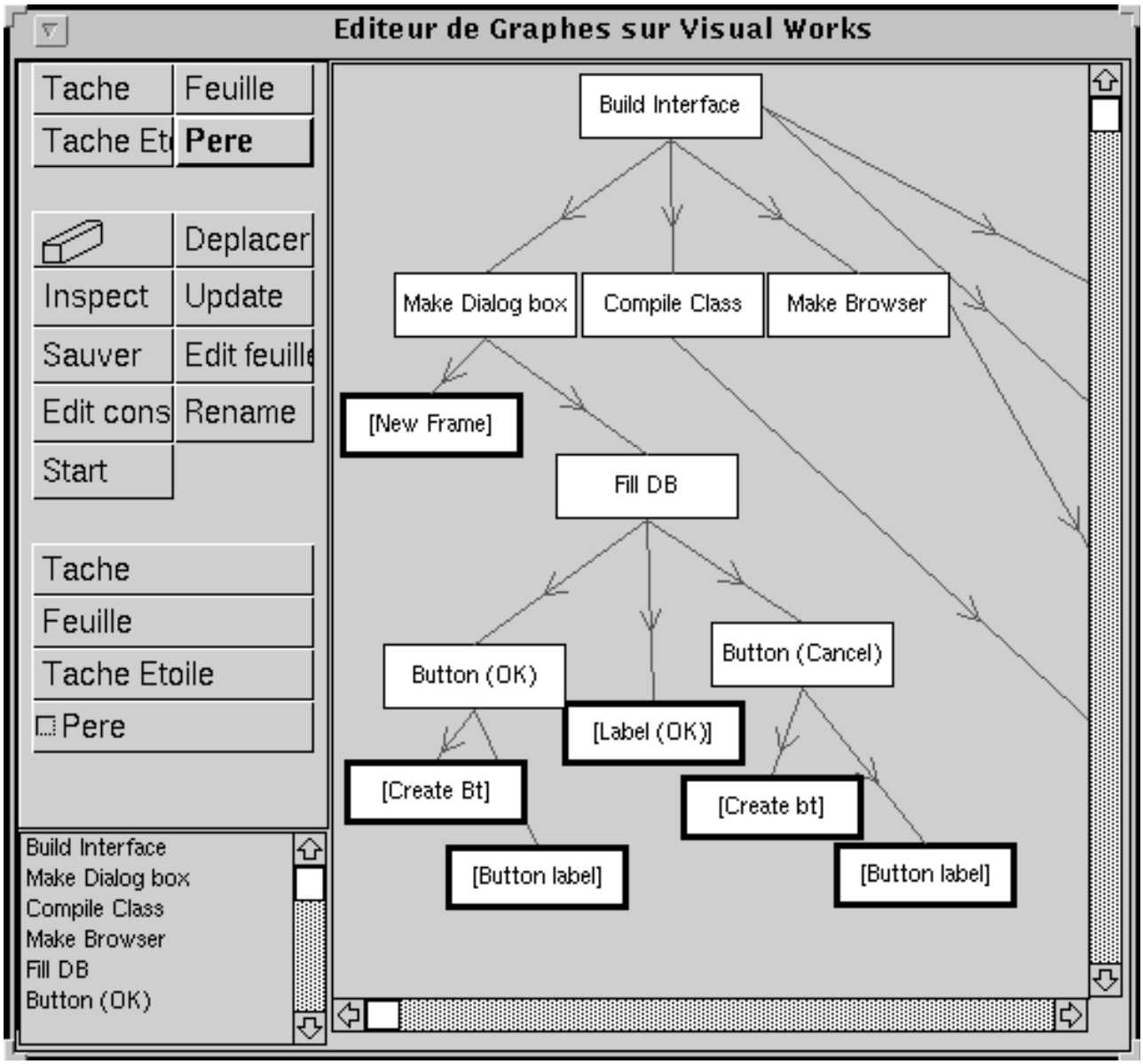

Figure 1: Un arbre de tâches pour VisualWorks.

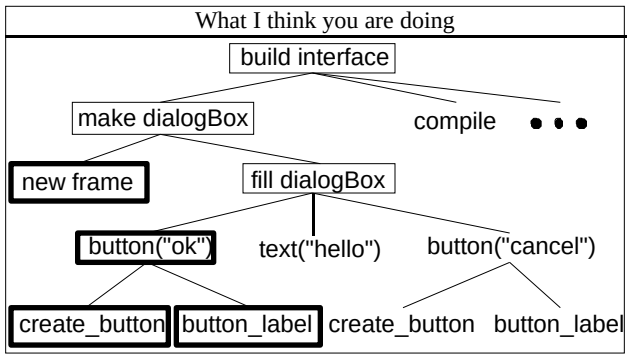

Figure 2: Un ATPI pour VisualWorks. Les rectangles indiquent les tâches en cours. Les rectangles en gras indiquent les tâches ou actions achevées, selon l'interprétation de l'analyseur.