Apprends ce que je fais

Fabrice Depaulis , Laurent Guittet, Christophe Martin

LISI / ENSMA
Teleport 2 – 1, avenue C. Ader
BP 40109 – 86961 Futuroscope Cedex
{depaulis, guittet, martin}@ensma.fr

RESUME
Malgré la pertinence de l'approche proposée par la Programmation sur Exemple, consistant à donner la possibilité à un utilisateur non programmeur d'automatiser une tâche répétitive en exécutant interactivement un exemple de cette tâche, celle-ci n'a toujours pas trouvé sa place dans les logiciels grand public. Une raison de cet échec relatif vient de la difficulté, pour un concepteur d'application, d'intégrer en natif un système d'enregistrement, de généralisation et de rejeu des actions de l'utilisateur. La solution consistant à mettre en place un système « externe », adaptable à une application quelconque en s'appuyant sur son interface, a également été abandonnée devant la difficulté de généraliser des actions déconnectées de toute fonctionnalité du noyau fonctionnel. Dans cet article, nous montrons comment un utilisateur peut introduire le lien sémantique nécessaire entre l'interface et le noyau fonctionnel de l'application, dans le but d'enregistrer des programmes sur l'exemple.

MOTS CLES : Programmation sur Exemple

ABSTRACT
From twenty years, many works have dealt with programming by demonstration. They have tried to make a end-user interactively program repetitive task, spying his/her actions while he/she is realizing it. Though the approach seems relevant, no commercial software uses it. The main reason is that it is very difficult for a software designer to integrate Programming by Example concepts: recording, generalizing and playing again the user interactions is far from being easy. However, an other approach consists in using an external system, which works on the user graphical interactions, and which is able to spy any application. Unfortunately, it is very difficult for such a system to succeed in generalizing the user actions, because it doesn't understand their meaning, relatively to the functional core. This paper deals with a new method that consists in making the user fill the previous semantic gap, describing his/her interaction and the interface state meanings.

KEYWORDS : Programming by Demonstration

Réserver cet espace pour la notice de copyright

INTRODUCTION
Depuis le milieu des années 80 et la thèse de Dan Halbert [3], de nombreux travaux ont tenté d'amener un « utilisateur final » vers la programmation au moyen de ce que l'on appelle la Programmation sur Exemple (PsE ou PbD pour Programming by Demonstration). Cette approche consiste à permettre à un non programmeur de concevoir un véritable programme informatique sans avoir à utiliser un langage à la syntaxe complexe et rébarbative. Ainsi, un système utilisant la PsE est capable d'enregistrer une série d'actions interactives, puis de les généraliser afin de générer le programme correspondant.

Vingt ans après SmallStar [4] (permettant d'automatiser des tâches répétitives), et malgré quelques réussites commerciales dans le domaine du jeu (StageCast Creator™ [2]), ou de la CAO (EBP [5]), la PsE reste absente de tous les logiciels grand public. Une explication de cet échec relatif se trouve dans la complexité à mettre en place des techniques de PsE. D'une part, la PsE requiert des concepts qui ne sont pas faciles à appréhender par un non spécialiste (niveau d'enregistrement des actions, différenciation entre variables et constantes, introduction de structures de contrôle, nomination des objets). D'autre part, l'intégration des mécanismes de PsE nécessite la refonte de l'application : pour saisir la sémantique des actions de l'utilisateur, il faut en effet les capturer à un niveau proche du noyau fonctionnel.

Intégrer la PsE de manière native à l'intérieur d'un logiciel (PsE interne) nécessite donc un investissement lourd de développement. Pour combler cet écueil, une solution (PsE externe) consiste à utiliser une application externe (AE). Celle-ci est capable de créer un programme sur un exemple exécuté par l'utilisateur dans tout type d'application cible (AC). Au lieu de reposer sur une connaissance des fonctionnalités de l'AC, l'AE espionne et enregistre les interactions de l'utilisateur sur l'interface.

Cette méthode, déjà expérimentée dans quelques systèmes par le passé, n'a jamais été approfondie. La raison principale vient du fait que l'absence de connaissance du noyau fonctionnel des AC restreint énormément les possibilités de généralisation. Nous proposons d'améliorer cette technique en mettant à contribution l'utilisateur, et en lui donnant la possibilité de faire très simplement le lien entre l'interface et les fonctionnalités de l'AC. En enseignant lui-même au système ce qu'il sait d'une telle application, il permet de passer du simple enregistrement d'interactions de bas niveau à un véritable outil de PsE. Cette méthode permet de tendre vers un système de PsE universel, capable de fonctionner sur toute application, tout en dépassant le simple stade de l'éditeur de macros.

Dans cet article, nous étudions d'abord différents systèmes permettant à un utilisateur d'automatiser des tâches répétitives en créant interactivement un programme dans une AC. Nous décrivons ensuite notre propre méthode en montrant comment l'utilisateur est amené à introduire de la sémantique au niveau des interactions de base, et nous l'illustrons en présentant notre outil, PbdScript. Nous tirons enfin un bilan de cette approche et nous traçons les lignes de nos futures recherches.

SYSTEMES DE PSE « EXTERNES »

Dans cette partie, nous détaillons deux systèmes en expliquant en quoi ils présentent certaines limites que notre approche permet de dépasser.

Eager

Eager [1] est un système capable de repérer une tâche répétitive à l'intérieur d'une AC reposant sur HyperCard™ et de l'automatiser.

Une des principales faiblesses d'Eager vient de l'utilisation d'une connaissance « a priori » des AC. Par exemple, la notion de « collection d'objets » est définie au niveau d'HyperCard™. Les motifs répétitifs qu'il automatise sont donc composés d'actions spécifiques, communes à toutes les applications. Pour pouvoir être pris en compte par le système de PsE, chaque objet de la présentation est en bijection avec un élément d'HyperCard™. Ceci permet à Eager de connaître la sémantique de l'application.

La solution d'Eager consiste à utiliser une connaissance a priori de la sémantique du noyau fonctionnel de l'AC. Nous proposons de résoudre cette limitation en permettant à l'utilisateur de définir lui-même cette sémantique. Cette approche possède un principe analogue au contrôle non destructif utilisé en mécanique et consistant à propager des ondes dans d'un bloc pour en connaître la structure interne. Notre solution rejoint l'idée d'exploiter une boîte noire, en l'occurrence le noyau fonctionnel, à partir de sa surface, ici l'interface.

Automate™

Automate™ propose à un utilisateur de programmer explicitement des tâches de manière interactive : ou bien à partir des fonctionnalités connues de MS Windows™ (manipulation de fichiers, opérations réseau, etc.), ou bien à partir des interactions de base lui permettant d'utiliser l'interface d'une AC.

Le fait de raisonner sur des interactions d'aussi bas niveau (boîte à outils de MS Windows™) permet à Automate™ d'être suffisamment général pour pouvoir prendre en considération toutes les applications. Cependant il n'exploite pas pleinement les propriétés des composants graphiques, qui sont pourtant une représentation de l'état du noyau fonctionnel. Prenons l'exemple d'un client de messagerie instantanée. La fenêtre de l'application affiche des contacts, répartis dans des listes. Les noms des contacts actuellement connectés apparaissent en gras.

Considérons un script permettant de sélectionner tous les gens connectés d'une liste et de leur envoyer le message « A table ! ». Réaliser une telle tâche dans Automate™ n'est possible qu'en définissant au préalable une liste textuelle composée d'un certain nombre de contacts et d'effectuer une boucle sur cette liste. Cependant, la liste définie ne correspond pas constamment aux contacts connectés. Automate™ ne sait pas que l'attribut « en gras » d'un « item » graphique représente une partie de l'état du noyau fonctionnel, et correspond à « un contact est connecté ». Les seuls moyens d'établir ce lien sont de connaître préalablement l'application, ou de donner la possibilité à l'utilisateur de le faire lui-même.

Conclusion

Ces exemples montrent la difficulté que pose la mise en place d'un système de PsE agissant sur une AC de manière externe. Les outils détaillés ici font le choix entre deux solutions. La première consiste à établir des hypothèses préalables sur les noyaux fonctionnels et à réduire du même coup l'éventail des AC. La seconde consiste à ne considérer que des interactions de bas niveau, de manière à étendre le choix des AC. On s'aperçoit de la nécessité de faire appel à un apport de l'utilisateur afin de combler le vide sémantique établi par l'indépendance avec le noyau fonctionnel.

Pour établir un système de PsE capable à la fois de travailler sur une AC quelconque et de généraliser les programmes créés, il faut établir un lien entre les interactions de l'utilisateur et le noyau fonctionnel de l'AC. Sauf à prédéfinir ce lien, il n'est réalisable que par l'utilisateur lui-même. Dans la partie suivante, nous montrons comment un tel mécanisme peut être mis en œuvre.

PBD SCRIPT

Nous avons mis au point une application indépendante, PbDScript, qui permet à un utilisateur final de créer un

programme basé sur un exemple réalisé dans une AC quelconque. Le principe peut être résumé ainsi : l'utilisateur réalise des actions, PbDScript les enregistre, pour ensuite les reproduire (rejeu). Il s'agit d'un système de programmation impérative dans lequel l'utilisateur fixe le début et la fin de l'enregistrement. L'originalité de ce système vient de la phase d'apprentissage préalable que doit réaliser l'utilisateur, pour permettre à PbDScript de comprendre le sens des interactions réalisées sur l'AC.

Pour mettre en place ce mécanisme, il faut pouvoir connaître l'ensemble des composants graphiques (widgets) de l'AC. A partir de cette hiérarchie, il faut pouvoir donner la possibilité à l'utilisateur de nommer les interactions de base. Ces noms sont ceux des tâches utilisateur et établissent le lien avec les objets, attributs et actions du noyau fonctionnel. Enfin, il faut permettre l'introduction des paramètres et structures de contrôle permettant la généralisation.

Identification des widgets

Pour que notre application puisse observer (enregistrement) et « remplacer » (rejeu) l'utilisateur, elle doit pouvoir récupérer les événements survenant sur les widgets.

Une AC est toujours représentée par des fenêtres (widgets conteneurs) qui peuvent elles-mêmes contenir d'autres widgets. PbDScript récupère l'arbre des widgets de présentation de l'AC par un mécanisme d'introspection, puis l'affiche.

A l'heure actuelle, PbDScript est capable de récupérer la hiérarchie des widgets de toute application Java reposant sur les boîtes à outils Swing et Awt. Une fois l'AC lancée et la hiérarchie récupérée, le système est en mesure de faire un lien dynamique entre la représentation des widgets dans l'AC, et leur représentation dans PbDScript. Ainsi, lorsque l'utilisateur agit sur un widget dans l'AC, celui-ci apparaît en surbrillance dans PbDScript. Inversement, lorsqu'un élément de la hiérarchie est désignée dans PbDScript, le widget correspondant est mis en évidence dans l'AC. Cela permet à un utilisateur non spécialiste de faire facilement le lien entre les différentes représentations proposées.

L'arborescence des widgets, structure statique de l'interface, est la source de production des événements. La capture de cette dynamique constitue l'objet du chapitre suivant.

Apport de sémantique

Une fois que l'on a récupéré l'ensemble des composants graphiques d'une AC, il faut faire correspondre à un widget une action du noyau fonctionnel. En effet, on ne peut généraliser des actions que si l'on connaît le noyau fonctionnel de l'AC.

L'analyse et la conception de l'AC reposent sur trois principes essentiels.

- Une série d'interactions correspond à une tâche utilisateur ;
- Une tâche utilisateur non articulatoire correspond à une action du noyau fonctionnel ;
- Le noyau fonctionnel est réfléchi dans les widgets.

Partant de ces principes, notre solution est de demander à l'utilisateur de permettre à l'utilisateur d'ajouter de l'information aux widgets (partie statique)et aux événements provenant de ces widgets (partie dynamique).

La solution la plus « naturelle » consiste à proposer d'associer une phrase à un couple [widget, événement]. L'utilisateur crée donc en quelque sorte un « dictionnaire » de commandes associé à son application. Reprenons l'exemple « A table ! ». Une zone de liste affiche la liste des contacts. Cliquer sur un élément de cette liste signifie pour l'utilisateur « Sélectionner un contact ». Au couple [zone de liste, clic], l'utilisateur va donc associer la commande « Sélectionner un contact » (Figure 1). De plus, dans cette zone de liste, l'attribut de présentation « gras » d'un élément permet d'indiquer à l'utilisateur que cet élément représente un contact actuellement connecté. L'utilisateur a une connaissance de la correspondance entre l'attribut de présentation et l'état du noyau fonctionnel. Pour cela, nous avons étendu la possibilité d'association « [widget, événement] / phrase » au couple « [widget, valeur d'attribut] / phrase ». Dans l'exemple, cela consiste à associer le couple [Élément de la zone de liste, En gras] à la fonction « Est connecté ». Dans la suite, le terme action du couple désigne aussi bien un événement provenant d'un widget qu'une fonction permettant de récupérer l'état de ce widget.

L'autre possibilité d'apport de sémantique consiste à utiliser la structure (1) ou la classe (2) de certains widgets.

- (1) Dans le cas d'un widget de type « liste », l'utilisateur sait qu'il peut s'agir d'une collection provenant du noyau fonctionnel. L'itération sur cette collection est le support de la généralisation de l'action répétée. En effet, cette action est paramétrée par une valeur du même type que celui de l'indice de boucle. Dans l'exemple « A table ! », on itère l'action « Envoyer un message à une personne » en prenant comme paramètre de cette action tous les items de la liste.
- (2) La connaissance qu'a l'utilisateur de l'application lui permet de regrouper des widgets représentant des concepts généralisables. En effet, dans l'interface graphique d'une application, plusieurs widgets peuvent être apparentés aux paramètres d'une même action du noyau fonctionnel. Par exemple, les boutons 0 à 9 d'un logiciel de type cal-

culatrice activent la même action du noyau fonctionnel : « Saisir nombre (digit) ». Ce regroupement dans une commande va permettre dans notre outil un paramétrage des scripts.

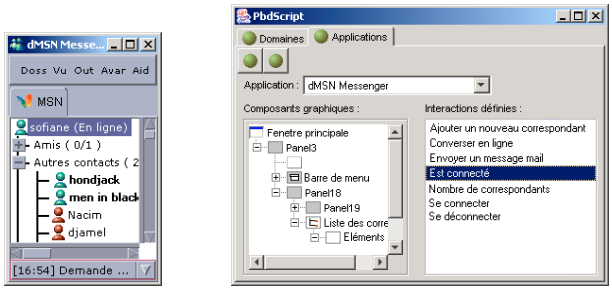

Figure 1. Phase d'association de l'exemple « A table ! ». A gauche, le client de messagerie, à droite PbDScript.

Cette nomination faite par l'utilisateur lui permet d'écrire le script en manipulant l'AC, et de le visualiser dans ses propres termes. Le formalisme utilisé repose sur un langage d'interaction plus que sur un véritable langage de programmation. Dans cette représentation, les paramètres sont implicites. Dans l'action « Envoyer un message à un contact », le paramètre « contact » n'apparaît pas.

Le lien établi par l'utilisateur entre les widgets de type « liste » et les collections du noyau fonctionnel constitue un premier pas vers les structures de contrôle. Dans ce cas, la génération des structures correspondant au parcours de tous les éléments est inférentielle. Cependant, dans certaines situations, l'utilisateur peut avoir à déclarer une boucle ou une condition de manière impérative. Ceci est réalisé explicitement, a posteriori, dans le langage d'interaction qu'il a lui-même définit. Dans l'exemple « A table ! », l'attribut « en gras » d'un élément de liste correspondant à l'information « un contact est connecté », peut être utilisé pour réaliser une condition.

CONCLUSION - OUVERTURES

Les systèmes de PsE existants proposent le choix entre deux solutions. La PsE interne consiste à intégrer la PsE au cœur de l'AC en utilisant un développement ad hoc, non réutilisable. La PsE externe prend le parti d'espionner les interactions exécutées sur n'importe quelle application, au détriment d'une certaine puissance de généralisation des programmes générés.

Nous avons mis au point un outil qui reprend la solution d'un système de PsE externe. L'originalité de notre approche consiste en la façon dont l'utilisateur peut décrire interactivement ce qu'il sait de l'AC, comblant le vide sémantique qui l'empêcherait, comme ces prédécesseurs, de mettre en place des mécanismes de généralisation.

A l'heure actuelle, PbDScript permet de définir la sémantique de toute application écrite en SWING ou AWT, à partir de la hiérarchie de ses widgets. L'extension à toute application Windows ou XWindow est en cours, et repose sur le même mécanisme d'introspection. L'utilisateur peut alors manipuler l'AC sur un exemple pour définir de véritables programmes. PbDScript permet d'introduire les structures de contrôles et les paramètres nécessaires à la généralisation.

Parmi les différentes perspectives qui s'ouvrent à nous, une des idées les plus intéressantes concerne la possibilité de définir interactivement des programmes sur une application donnée pour les réutiliser sur une deuxième application, possédant des fonctionnalités similaires. Par exemple, si l'on reprend l'exemple « A table ! », on imagine très bien que le programme défini dans MSN Messenger™ puisse être utilisé dans Yahoo ! Messenger™. Pour cela, il suffit que la sémantique définie au niveau de MSN Messenger™ soient redéfinies, avec le même nom, au niveau de Yahoo ! Messenger™. On s'aperçoit que la sémantique définie constitue une abstraction utilisable dans différents logiciels. Une autre ouverture concerne la possibilité de permettre aux programmes enregistrés de s'enrichir à chaque exécution (définition implicite de structures conditionnelles). Lorsque l'exécution se passe de manière différente à celle de la description (apparition d'une fenêtre non prévue), le système passe en mode enregistrement et rend la main à l'utilisateur pour qu'il décrive interactivement l'attitude à adopter.

REFERENCES

1. Cypher, A. Eager: Programming Repetitive Tasks by Example. In *Proceedings of Human Factors in Computing Systems (CHI'91)* (27 April - 2 May, New Orleans, Louisiana), ACM/SIGCHI, 1991, pp. 33-39.
2. Cypher, A. et Smith, D.C. KidSim: End User Programming of Simulations. In *Proceedings of Human Factors in Computing Systems (CHI'95)* (7-11 May, Denver, Colorado), ACM/SIGCHI, 1995, pp. 27-36.
3. Halbert, D. *Programming by Example*. PhD Thesis : University of California, 1984.
4. Halbert, D. SmallStar : Programming by Demonstration in the Desktop Metaphor. Cypher, A. (Ed.). In *Watch What I Do : Programming by Demonstration*, The MIT Press, Cambridge, Massachusuetts, 1993, pp. 102-123.
5. Potier, J.-C. Conception sur exemple, mise au point et génération de programmes portables de géométrie paramétrée dans le système EBP. Doctorat d'Université (PhD Thesis) : Université de Poitiers, 1995.