Introduire des techniques de Programmation sur Exemple dans une boîte à outils : une étude de besoin

Loé SANOU, Patrick GIRARD, Laurent GUITTET

LISI/ENSMA
Téléport 2-1, avenue Clément Ader
BP 40109, 86961 FUTUROSCOPE Cedex
{loe.sanou, girard, guittet}@ensma.fr

RESUME
Les systèmes de Programmation sur Exemple (Programming by Demonstration ou PbD) ont à ce jour démontré leur intérêt dans de nombreux cas. Cependant, l'introduction systématique des techniques de PbD dans les applications n'est pas aisée en raison d'un manque d'outils. Nous avons entrepris de combler cette lacune en définissant des outils susceptibles d'être incorporés aux boîtes à outils classiques. Cette contribution constitue une première étape, par l'identification des classes d'applications de PbD, et par la caractérisation des besoins essentiels en terme d'enregistrement et de rejeu.

MOTS CLES : Programmation sur Exemple, Boîtes à outils

CATEGORIES AND SUBJECT DESCRIPTORS: D.2.3 [Software Engineering]: Coding Tools and Techniques
GENERAL TERMS: Experimentation, Standardization

ABSTRACT
Programming by Demonstration (PbD) proved for a long time its interest in many cases. In the meantime, introducing such techniques into programs remains difficult, due to the lack of tools. We started to fill this gap by defining tools to be incorporated into classical toolboxes. This paper is a first step, with the classification of PbD applications, and the characterization of the essential needs for recording and replaying interactions.

KEYWORDS : Programming by Demonstration, Toolkits

INTRODUCTION

Les systèmes de Programmation sur Exemple (Programming by Demonstration ou PbD) sont assez nombreux dans la littérature, comme en témoignent les deux receuils qui y sont consacrés ([1] et [2]). Dans certains cas particuliers, l'intérêt de cette technique a été largement démontré. Cependant, la PbD requiert des techniques de programmation approfondies (différences entre variables et constantes, structures de contrôles, nomination des objets, niveau d'enregistrement des actions, …) et l'intégration des mécanismes de PbD nécessite le plus souvent une recomposition de l'application afin de pouvoir capturer les actions utilisateur à un niveau proche du noyau fonctionnel.

Jusqu'à ce jour, il n'existe pas de plate-forme robuste pour la construction de système de PbD et on ne dispose pas d'outils spécifiques permettant la réimplantation ou la réalisation de ces systèmes. La présente contribution se veut une première étape vers la réalisation d'une telle plate-forme, susceptible de mettre la technique de la PbD à la portée de tous les programmeurs. À partir d'une classification des systèmes de PbD, et en nous appuyant sur un exemple bien connu, nous définissons les principaux besoins d'une telle plate-forme en terme d'enregistrement et de rejeu, qui constituent les bases de la technique.

CLASSIFICATION DES SYSTEMES DE PBD

Le concept de la programmation sur exemple trouve son origine dans Pygmalion ([1]). Il a été approfondi à travers Smallstar ([1]). En 1980, Brad Myers le formalise dans ses travaux et lui donne le nom de Example-based programming ([3]). En 1993, A. Cypher fait un récapitulatif des systèmes existants et introduit la notion de Programming by Demonstration ([1]).

Objectifs des systèmes de PbD

Selon les auteurs de ces systèmes, l'objectif principal est de générer interactivement des programmes. Cela est explicite dans les systèmes Peridot, Mondrian, SmallStar, ou encore ToonTalk ([1]). Aussi, les systèmes de PbD cherchent-ils à fournir aux non programmeurs le pouvoir d'expression d'un environnement de programmation classique (éditeur, structuration de programmes, …). Le plus souvent, ils s'attachent également à enrichir et à personnaliser l'interface de dialogue du système interactif, en permettant à l'utilisateur d'associer l'exécution

d'un programme construit interactivement à une commande du système.

Par exemple, Eager [4] automatise les tâches répétitives entre applications différentes. À l'inverse Tinker ([1]) a pour but de favoriser l'apprentissage de la programmation elle-même. Les programmes produits n'ont pas d'intérêt particulier. Pygmalion ([1]) se focalise sur la création interactive de programmes et StageCast Creator ([2]) génère des applications. Enfin, le système EBP ([2]) crée des outils de conception technique.

Classes des systèmes

Beaucoup de systèmes de PbD n'ont eu pour but que de démontrer la validité du concept même de programmation sur exemple. Afin de caractériser les besoins en terme de PbD, nous avons préféré classer les systèmes en fonction de l'utilisation de la PbD elle-même. De ce point de vue, on peut répartir les systèmes en trois catégories : **l'assistance, les outils de conception et l'apprentissage de la programmation**.

L'assistance est la catégorie regroupant tout les systèmes permettant d'automatiser des tâches utilisateurs. C'est le but de la majorité des systèmes de PbD. Ils offrent une aide contextuelle à l'utilisateur afin de faciliter ou de réduire l'exécution de certaines de ses tâches. Le but global du système est très éloigné de la PbD. Dans cette classe on peut citer Eager (voir plus haut), Smallstar ([1]), qui cherche à résoudre les problèmes d'expressivité des langages iconiques en permettant de créer des macros puissantes, ou encore Tels ([1]), qui est spécialisé vers les tâches d'édition de texte.

Les outils de conception rassemblent les applications dont le but est de produire un résultat qui découle directement de l'utilisation de la PbD. Ces outils peuvent être classés dans la rubrique générale de la conception assistée par ordinateur. Qu'il s'agisse de pièces mécaniques comme pour EBP ([8]), d'applications à destination des enfants pour StageCast Creator ([2]) ou encore d'un GUI-Builder amélioré pour PERIDOT([5]), ils ont en commun de s'appuyer sur la PbD pour atteindre leur objectif.

L'apprentissage de la programmation concerne les systèmes dont le but est l'apprentissage de la programmation. On peut citer Tinker ([1]) ou encore Toontalk ([1]) et MELBA le plus recent avec Guibert paru dans IHM'04.

DES SOLUTIONS EXISTANTES

Fournir des services pour utiliser la PbD au sein des applications est fortement dépendant de la classe de système. Les applications dont le but est l'apprentissage de la programmation ont des besoins extrêmement spécifiques ; la programmation sur exemple est le cœur même du système interactif. De même, les outils de conception demandent en règle générale une très forte imbrication du système de PbD dans le noyau fonctionnel même du système global. En revanche, les applications de la première catégorie n'utilisent la PbD que pour ajouter des fonctionnalités ou aider l'utilisateur dans sa tâche. Cependant, rares sont les travaux qui ont essayé d'apporter une solution générale permettant de développer à moindre coût des solutions incluant la PbD.

Deux solutions ont été développées selon des approches radicalement différentes : une approche interne, au travers du système AIDE ([6]), et une approche externe, avec PbDScript ([7]). Ces systèmes ne peuvent être classés dans les trois catégories présentées ci-dessus, leur objectif étant de construire des applications de PbD. Ils constituent une quatrième catégorie, que l'on peut qualifier d'**outils de PbD**.

L'utilisation de la voie interne consiste à intégrer la PbD au cœur de l'application, pendant sa construction. À l'inverse, l'approche externe consiste à espionner une application déjà développée, pour permettre l'application de la PbD. Elle peut faire le choix de l'établissement d'hypothèses préalables sur les noyaux fonctionnels (ce qui réduit le champs des applications) ou ne considérer que des interactions de bas niveau (de manière à étendre le choix des applications).

AIDE vise à faciliter la création de systèmes en fournissant une couche logicielle sur laquelle les développeurs peuvent bâtir des applications mettant en œuvre des techniques démonstrationelles. L'utilisateur définit des macros opérant sur une collection ordonnée d'arguments. Afin d'enseigner une macro, l'utilisateur sélectionne les objets formant ses arguments et démontre les étapes de la macro en effectuant les commandes appropriées. Les objets sélectionnés à la fin de la macro constituent la valeur retournée par la macro. L'architecture d'AIDE s'articule autour des commandes de haut niveau et d'un gestionnaire de commandes gérant l'historique et la création de macros. Le processus de généralisation s'effectue par la détection de boucles non imbriquées, la généralisation des arguments et la détection de séquences (numérique, chaîne de caractères, points et classes). AIDE présente aux développeurs une structuration d'application mettant en œuvre la PbD mais n'est pas un outil permettant d'ajouter facilement des capacités de PbD à une application. Il suppose de respecter une architecture particulière, et impose le langage Smalltalk pour le développement.

PbDScript est un système qui permet d'enregistrer de façon totalement externe les interactions d'une application. PbDScript récupère l'arbre des widgets de présen-

tation d'une application Java par un mécanisme d'introspection, puis l'affiche. Il demande à l'utilisateur d'ajouter de l'information aux widgets et aux événements provenant de ces widgets ; la sémantique de l'application est ainsi décrite par l'utilisateur. PbDScript est un système très souple, qui permet de créer très simplement des macros sur une application quelconque. Cependant, ces fonctionnalités de PbD ne peuvent être inclues directement dans le programme. D'autre part, la tâche d'apprentissage de l'application est relativement longue car elle nécessite d'activer tous les écrans de l'application afin que l'introspection puisse se faire correctement.

Bien que présentant des intérêts certains, ces deux systèmes ne répondent pas au besoin que nous avons exprimé : PbDScript ne permet pas à un concepteur d'introduire des fonctionnalités de PbD dans l'application qu'il réalise. Quant à AIDE, il est limité de par ses choix d'implémentation (Smalltalk) et par sa technique d'espionnage, point sur lequel nous reviendrons plus loin.

L'EXEMPLE EAGER

Afin de rendre plus concrète l'expression des besoins en terme de PbD, nous avons choisi de nous appuyer sur une application typique de la catégorie « assistance », Eager.

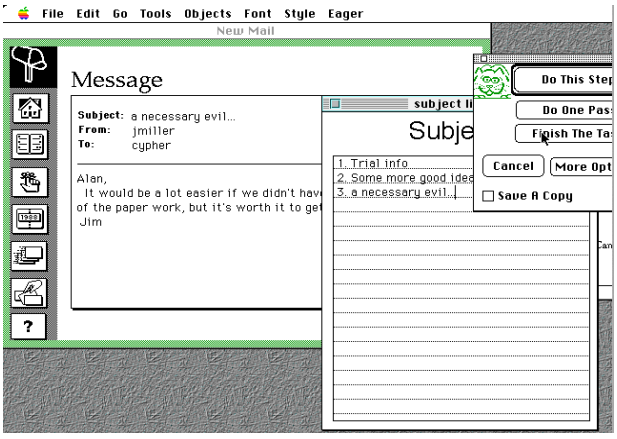

Figure 1 : Eager

Eager permet d'aider l'utilisateur dans ses tâches répétitives. Il prend en compte le fait que l'utilisateur ne réalise pas toujours immédiatement qu'il peut créer un programme pour accomplir sa tâche. Le système enregistre en permanence les actions de l'utilisateur, à la recherche de potentielles répétitions. Lorsqu'il détecte l'une d'entre elle, il demande à l'utilisateur de valider sa « déduction » en anticipant la commande suivante de l'utilisateur sans l'exécuter. Si l'inférence est incorrecte, la commande est considérée comme un contre exemple. L'utilisateur confirme la bonne inférence par clic sur une icône particulière de menu afin que Eager finalise la tâche. Il a le choix entre l'exécution d'un seul coup (un seuil de confiance égal au moins à trois itérations), ou de faire du pas à pas. Eager n'infère pas de conditions, ni de boucles imbriquées, ni de répétitions temporelles.

Notre objectif consiste à identifier les besoins directement liés à la PbD lors de la conception d'une application comme Eager. Eager permet d'automatiser des tâches effectuées dans des applications classiques. L'exemple d'Eager s'appuie sur deux applications simples que sont un mini-éditeur de texte et un mini-mailer. La tâche automatisable consiste à recopier les en-têtes de mails à partir du mailer dans l'éditeur de texte, en les numérotant.

D'un point de vue utilisateur, la tâche consiste à sélectionner le titre du mail, le copier, activer l'éditeur, numéroter la ligne, puis coller le contenu du presse-papiers. Le retour dans le mailer et l'activation du mail suivant constituent la fin de la séquence qui doit être répétée pour tous les mails.

BESOINS POUR UNE BOITE A OUTILS

Du point de vue du programmeur, il s'agit de recenser et de minimiser les actions de programmation explicite nécessaire pour construire ces fonctionnalités de PbD. Les boîtes à outils possèdent aujourd'hui des widgets évolués qui permettent de s'affranchir de beaucoup de programmation. Ainsi, le « Copier-Coller » est-il fonctionnel sans effort particulier dans des widgets de type texte. Les applications de type WIMP, qui représentent la grande majorité des applications actuelles, sont organisées à base de conteneurs rassemblant des widgets actifs, comme des boutons ou des champs de texte. Dans le cas de notre exemple, l'éditeur de texte possède les fonctions de couper-coller par l'intermédiaire de menus, et le mailer possède en plus des boutons « Suivant » et « Précédent » pour naviguer dans les mails.

Quels services fournir ?

Une application de PbD doit permettre l'enregistrement et le rejeu des interactions de l'utilisateur, auxquelles s'ajoutent des capacités de généralisation. Plusieurs travaux ont défini les besoins de couplage entre les applications hôtes et les systèmes de PbD, en se focalisant particulièrement sur la généralisation. On trouvera ainsi une étude comparée dans [8]. Mais dans l'optique de fournir des outils généraux et indépendants des applications, il faut s'intéresser beaucoup plus finement aux opérations de base que constituent l'enregistrement et le rejeu

Enregistrement

Dans AIDE, l'enregistrement s'appuie sur la définition a priori de commandes. Dans notre exemple, il s'agirait de commandes telles que « Copier », « Coller » ou « Mail Suivant ». Dans le dernier cas, il s'agit d'espionner un

événement du type du clic sur un bouton, qui doit être explicitement programmé par le concepteur de l'application. Mais dans les deux autres cas, on doit tenir compte de deux possibilités d'interaction : l'utilisateur peut utiliser les menus, ou bien les raccourcis clavier qui sont généralement implantés par défaut dans les widgets, sans intervention explicite du programmeur.

Se restreindre aux actions de type « commande » n'est cependant pas suffisante pour permettre d'espionner en totalité l'utilisateur. Dans le cas d'Eager, l'utilisateur effectue des actions de frappe au clavier à l'intérieur de l'éditeur de texte. Lorsque l'on utilise des widgets évolués, ce type d'interaction n'a pas besoin d'être programmé. Le widget le gère tout seul. On voit ici la limite du système AIDE qui impose la définition de commandes, qui seules peuvent être enregistrées. À l'inverse, n'espionner que les opérations de très bas niveau ne permet pas de s'abstraire des tâches articulatoires qui constituent une part importante du dialogue homme-machine.

Rejeu

Au niveau du rejeu, Eager est un bon exemple des différents besoins qui se font jour. Lorsque Eager souhaite faire valider son analyse de la tâche exemple, il doit réactiver les interactions de l'utilisateur en les mettant en évidence. Par exemple, lorsque l'utilisateur est invité à confirmer une commande activée par un item de menu, Eager modifie la présentation du menu pour ajouter à l'item choisi une icône spécifique (Figure 2)

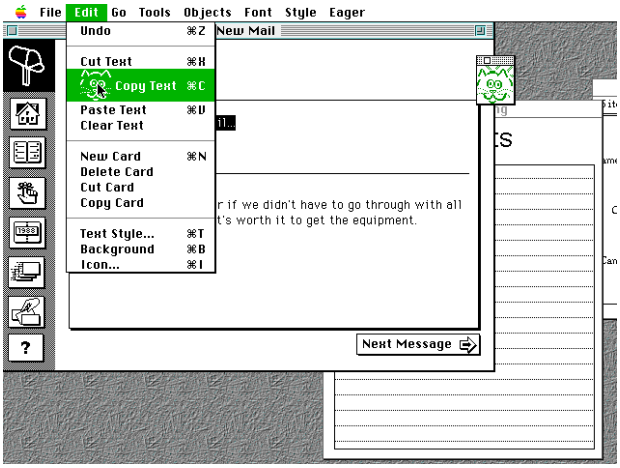

Figure 2 : Rendu spécifique à Eager

À l'inverse, lorsque la tâche automatisée doit être complétée, il n'est plus forcément nécessaire de rejouer très exactement les interactions de l'utilisateur. On constate donc que la boîte à outil doit permettre ces deux modes de fonctionnement selon le besoin de rejeu. Là encore, AIDE se révèle insuffisant dans son approche, car le rejeu des commandes est loin de satisfaire tous les besoins d'une application comme Eager.

Principes de réalisation

À partir de l'étude que nous avons réalisée, nous développons actuellement un prototype de boîte à outils sur la base de Swing. L'utilisation de l'abonnement, qui a servi de base à la réalisation de PbDScript, est faite pour satisfaire le besoin d'espionnage des commandes explicites. Dans une première approche, nous n'avons pas retenu la solution des hiérarchies d'action préconisée par AIDE, car elle impose une architecture d'application particulière, qui nous semble inappropriée à ce niveau.

Pour gérer les actions implicites, nous pensons qu'il est nécessaire de sous-classer les widgets de la boîte à outils support. Ceci permet d'apporter toute la puissance nécessaire tout en limitant le travail du programmeur d'application. Ce sous-classement permet également de contrôler finement le rendu des widgets pour ajouter les primitives nécessaires au rejeu.

CONCLUSION

La présente étude constitue un premier pas vers la réalisation d'une véritable boîte à outils susceptible de simplifier la mise en œuvre de la programmation sur exemple dans les applications interactives. La prochaine étape consiste à définir les solutions d'implémentation et à les expérimenter.

BIBLIOGRAPHIE

1. Cypher, A., ed. *Watch What I Do: Programming by Demonstration*. 1993, The MIT Press: Cambridge, Massachusetts. 604.
2. Lieberman, H., *Your Wish is my command*. 2001: Morgan Kaufmann. 416.
3. Myers, B.A., *Taxonomies of Visual Programming and Program Visualization*. Journal of Visual Languages and Computing, 1990. **1**(1): p. 97-123.
4. Cypher, A. *Eager: Programming Repetitive Tasks by Example*. in *Human Factors in Computing Systems (CHI'91)*. 1991. New Orleans, Louisiana: ACM/SIGCHI.
5. Myers, B.A., *Creating User Interfaces Using Programming by Example, Visual Programming, and Constraints*. ACM Transactions on Programming Languages and Systems, 1990. **12**(2): p. 143-177.
6. Piernot, P.P. and M.P. Yvon, *The AIDE Project: An Application-Independent Demonstrational Environment*, in *Watch What I Do: Programming by Demonstration*, A. Cypher, Editor. 1993, The MIT Press: Cambridge, Massachusetts. p. 383-402.
7. Depaulis, F., L. Guittet, and C. Martin. *Apprends ce que je fais.* in *15ème Conférence Francophone sur l'Interaction Homme-Machine*. 2003. Caen: ACM Press.
8. Girard, P., *Ingénierie des systèmes interactifs : vers des méthodes formelles intégrant l'utilisateur*, in *LISI/ENSMA*. 2000, Université de Poitiers: Poitiers. p. 92.