Vers un outil pour la réalisation de systèmes de PbD

Loé SANOU – Patrick GIRARD

LISI – ENSMA (Laboratoire d'Informatique Scientifique et Industrielle)
Téléport 2 – 1, avenue Clément Ader BP : 40109
86961, Futuroscope, France
{sanou, girard, guittet}@ensma.fr

RESUME
Cet article décrit une technique de réalisation d'une boîte à outils destinée à la réalisation d'applications utilisant la programmation sur exemple. Malgré la grande diversité de ces applications, il n'existe aujourd'hui aucune plate forme robuste permettant leur réalisation. Cette contribution vise à combler cette lacune. Pour une bonne orientation des travaux, une classification des systèmes dites de PbD est d'abord proposée. Les fonctionnalités de base d'un tel outil sont aensuite définie à partir de l'étude d'un système exemple (Eager). Enfin, un prototype réalisé à partir de la bibliothèque Swing de Java est présenté.

MOTS CLES : Programmation sur exemple, boîte à outils, interaction homme machine, événement swing.

ABSTRACT
This article describes a technique of a toolkit realisation-intended for the realization of applications that use programming by demonstration. In spite of the great diversity of these applications, there is not any robust platform today to build such applications. This contribution tries to fill this gap. A classification of known PbD systems is proposed. The tool functionalities are expressed based on an exemplary well-known application (Eager). Last, a prototype, which has been constructed starting based on the Java Swing library is presented.

CATEGORIES AND SUBJECT DESCRIPTORS: D.2.3 [Software Engineering]: Coding Tools and Techniques.

GENERAL TERMS: Experimentation, Standardization

KEYWORDS: Programming by Demonstration, toolkit, human computer interaction, Swing event.

INTRODUCTION

« Si un utilisateur est capable d'accomplir une tâche sur un ordinateur, cela doit suffire au système pour créer un programme permettant de généraliser cette tâche » [1]. C'est en ces termes que A. Cypher définit le principe de « Programming by Demonstration » (PbD). La traduction française la mieux appropriée de ces termes est « Programmation sur Exemple » [2]. C'est une approche permettant de créer des programmes en s'appuyant sur les actions réalisées par l'utilisateur final lors de la construction d'exemples. Des développements du concept à travers de multiples systèmes ont été réalisés en exploitant différentes voies de réalisation. Cependant, malgré un intérêt évident comme en témoignent [1] et [3], cette technique a rarement dépassé le cadre des laboratoires de recherche. L'une des raisons de cette absence peut être recherchée dans sa difficulté d'implémentation. En effet, implanter un système de PbD est une tâche complexe ; elle met souvent en jeu des techniques d'intelligence artificielle (apprentissage symbolique, etc.), de conception (choix d'une syntaxe et d'un vocabualire appropriés) ou d'implémentation (interpréteur ou compilateur intégré) spécifiques. Les techniques d'interaction homme machine (visualisation, notification, prévention d'erreur, annulation, etc.) [4] sont également concernées.

Au vu des efforts à fournir pour construire de telles application, il s'avère nécessaire de disposer d'une plate forme robuste et souple pour faciliter l'implantation de systèmes de PbD. De nos jours, il n'en existe aucune. Notre contribution est un premier pas pour combler cette lacune. À travers ce papier, nous présentons la possibilté de mise en place d'un outil intégré à la bibliothèque Swing de Java permettant de faciliter le travail des développeurs. Auparavant, nous présentons une classification de ces systèmes. Une implémentation des fonctionnalités au niveau de la boîte à outils est proposée à partir d'une identification des besoins des applications de PbD.

ANALYSE DES SYSTEMES EXISTANTS

La PbD a été utilisée dans un grand nombre de systèmes expérimentaux [1]. Il convient néanmoins de noter qu'elle n'est pas utilisée à grande échelle. Le but principal de la PbD est de générer interactivement des programmes, i.e. généraliser les suites d'actions effectuées par un utilisateur lors de l'utilisation d'un programme. La PbD se place à un niveau d'abstraction beaucoup plus élevé que les enregistreurs de macros [5]. Elle manipule directement les objets et la sémantique associée permet de sortir de la simple réexécution des actions. Les systèmes de PbD présentent des objectifs très divers. Certains permettent l'apprentissage de la programmation tandis

que d'autres favorisent l'automatisation de tâches rébarbatives [2]. Ils couvrent une large variété d'applications. Pour mieux caractériser les besoins en terme de PbD, une classification s'impose.

Classification

Les premières classifications ont porté sur le domaine de la « Visual Programming » [6]. La taxonomie de Myers [5] classe toutes les approches de la programmation selon trois critères principaux orthogonaux. Elle apparaît insuffisante dans la mesure où elle ne porte que sur la phase d'exécution des programmes et sur leur représentation visuelle. Pour permettre une bonne comparaison des systèmes entre eux, une grille d'évaluation (dirigée par des critères de comparaison autour de quatre points) a été fournie dans [1]. Ce sont : domaine et utilisateurs, moyens d'interactions, l'inférence, et les connaissances du domaine. Cette grille permet de montrer la validité de certains outils. La taxonomie de Myers a été améliorée dans [2] par une classification suivant trois critères principaux et un critère secondaire : compilé ou interprété, programmation graphique ou non, programmation sur exemple ou non, et déclaratif ou impératif.

Si ces études sont intéressantes pour classifier les systèmes par rapport à l'aspect visuel ou par rapport à l'activité de programmation, elles ne permettent guère de les situer en fonction des utilisateurs. Ces classifications ne caractérisent pas non plus les systèmes en fonction de l'utilisation même de la PbD. En nous plaçant du point de vue du système de PbD lui même, nous proposons de classer les systèmes selon quatre catégories : l'assistance, l'apprentissage de la programmation, les outils de conception et les outils de PbD. *Le tableau 1* résume le positionnement de quelques-uns des principaux systèmes utilisant la PbD au regard de cette classification.

L'assistance offre une aide contextuelle à l'utilisateur afin de faciliter ou de réduire l'exécution de certaines de ses tâches. Elle concerne ainsi les systèmes permettant d'automatiser des tâches utilisateurs. On peut citer comme exemples Eager [1], SmallStar [1], Tels [1]. *L'apprentissage de programmation* donne à l'utilisateur la possibilité d'apprendre à programmer en s'appuyant sur des exemples, au lieu d'utiliser les classiques méthodes de programmation très abstraites. On peut citer dans cette catégorie Tinker [3] et MELBA [7]. *Les outils de la conception* sont des applications dont le but est de produire un résultat qui découle directement de l'utilisation de la PbD. La PbD est utilisée dans le cœur même du processus de conception. On peut citer ici EBP [2], StageCast Creator [3], Peridot [1]. Enfin, *Les outils de PbD* sont les systèmes dont le but est d'intégrer des techniques de PbD dans d'autres applications, i.e. développer à moindre coût des solutions incluant la PbD. Seuls les systèmes AIDE [8] et PbDScript [9] relèvent de cette catégorie.

Systèmes \ Catégorie	Ass	A.P	O.C	O.P
Aide Project	×			×
Eager	×			
EBP			×	
Gamut			×	
MELBA		×		
Metamouse			×	
Mondrian			×	
PbDScript				×
Peridot			×	
Pygmalion			×	
SmallStar	×		×	
StageCast Creator		×	×	
Tels	×			
Tinker		×		
ToonTalk		×	×	

Tableau 1 : Classification de quelques systèmes. Ass = Assistance – A.P = Apprentissage de programmation – O.C = Outils de conception – O.P = Outils de PbD.

Notre objectif est de fournir une boîte à outils favorisant la construction d'applications incluant la PbD. En ce sens, notre travail ne porte que sur la classe des outils de PbD. Il faut donc fournir une couche logicielle sur laquelle les développeurs peuvent bâtir, avec un minimum d'effort, des applications mettant en œuvre des techniques de la programmation sur exemple.

MECANISMES ET BESOINS

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 classe « assistance », Eager [1], qui, au-delà de représenter l'un des exemples les plus connus de la PbD, permet d'exprimer des besoins très divers.

Expression des principes de la PbD

Eager s'adresse à des utiilisateurs n'ayant aucune connaissance préalable de la programmation. Il s'appuie sur une application composée d'un mailer et d'un éditeur de texte (*Figure 1*). Le principe consiste à automatiser les actions de l'utilisateur, même si elles concernent globalement ces deux applications. Considérons une tâche automatisable qui consiste à recopier les en-têtes de mails à partir du mailer dans l'éditeur de texte, en les numérotant. Un programme permettant de réaliser cette tâche s'exprimera sous la forme suivante, en pseudocode :

Répéter pour tous les mails
    Copier le titre du mail
    Inscrire un numéro (compteur incrémenté) sur la première ligne vide dans l'éditeur
    Coller le titre du mail
Fin répéter

Le paradigme de programmation sur exemple consiste à s'appuyer sur la réalisation interactive de la tâche par l'utilisateur pour construire le programme. La tâche consiste donc à sélectionner le titre du mail, le copier, activer l'éditeur, numéroter la ligne, puis coller le contenu du presse-papier. 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. Eager observe en permanence les interactions de l'utilisateur sans le perturber. Il se manifeste lorsqu'il détecte un pattern d'actions, i.e. une dernière suite de séquences est égale à la précédente. Dans l'exemple précédent, on ne remarque la présence d'Eager que lorque l'utilisateur finit de coller le deuxième sujet dans l'éditeur.

Afin de réaliser un programme de PbD tel Eager, il faut recenser et minimiser les actions de programmation explicites nécessaires pour construire les fonctionnalités de la PbD. Les boîtes à outils possèdent aujourd'hui des widgets évolués qui permettent de s'affranchir de beaucoup de programmation. Les applications de type WIMP (Wimdows, Icons, Menus and Pointers) [10], qui représentent la grande mojorité des applications actuelles sont organisées à base de conteneurs rassemblant des widgets actifs, comme des boutons ou des champs de texte.

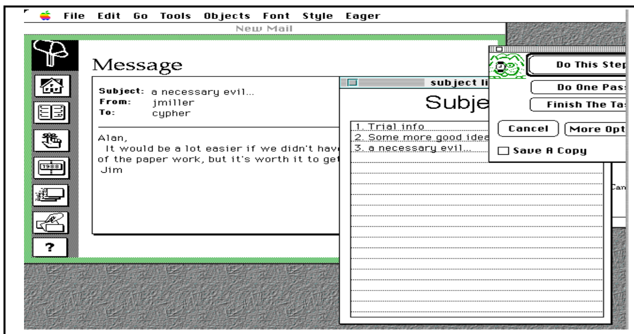

Figure 1 : Le système Eager (en demande de finition de tâche).

Ainsi, les systèmes de PbD analysent les actions de l'utilisateur lors de la réalisation de l'exemple. Ils utilisent deux types d'analyses : la première consiste à enregistrer les actions ou séquences effectuées, et la deuxième procède par inférence (mécanisme d'abstraction) sur l'exemple entier. Il en résulte une action d'espionnage sur le système hôte.

Les trois services essentiels à fournir

Une application de PbD doit permettre un enregistrement des interactions de l'utilisateur. Ces interactions enregistrées doivent être ré-exécutées : c'est la fonction de rejeu. Ce rejeu est en général la continuation d'une suite logique d'actions. À ceux-ci s'ajoutent les capacités de généralisation. *La figure 2* présente l'interaction entre ces fonctionnalités.

L'enregistrement s'appuie sur la définition a priori de commandes. En réalité, il s'agit d'espionner les événements du type click sur un bouton, « Copier Coller », ou de touches clavier. Les événements de type click sur un bouton doivent être explicitement programmés par le concepteur de l'application. Mais dans les autres cas, on doit tenir compte de deux possibilités d'interaction : l'utilisateur peut utiliser les menus, ou les raccourcis claviers qui sont généralement implantés par défaut dans les widgets, sans intervention explicite du concepteur. *La généralisation* est le mécanisme le plus complexe dans la mise en œuvre. En effet, non seulement la phase d'enregistrement dans actions est capitale, mais aussi la technique et le contexte de rejeu doivent être pris en compte. La généralisation doit permettre le paramétrage des actions et les adapter à de nouveaux contextes. *Le rejeu* consiste en la récupération des séquences et leur réexécution. Si l'on souhaite faire valider l'analyse d'une tâche exemple, on doit pouvoir réactiver les interactions prochaines de l'utilisateur en les mettant en évidence. Il n'est pas forcement nécessaire de rejouer exactement les interactions de l'utilisateur, comme par exemple les déplacements de souris.

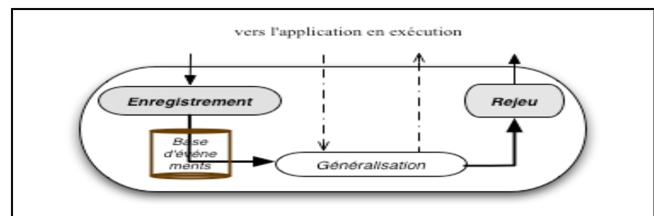

Figure 2 : Les besoins (interaction entre les fonctionnalités).

UNE PISTE DE SOLUTION

Plusieurs systèmes ont tenté de mettre à la disposition de l'utilisateur final les moyens nécessaires pour personnaliser ses logiciels. Seulement deux d'entre eux se sont interésssés aux développement d'outils de PbD : le projet Aide [8] (par une approche interne) et PbDScript [9] (par une approche externe). Ces deux systèmes présentent des intérêts certains, mais ne répondent pas exactement aux besoins ci-dessus cités. En particulier, ils ne permettent pas d'automatiser les applications sans apport sémantique de la part de l'utilisateur.

L'outil que nous ambitionnons de réaliser doit faciliter la réalisation d'application intégrant la PbD. Il doit être indépendant du domaine d'application et fournir des fonctionnalités générales réutilisables. Il ne s'agit pas pour nous d'inventer un système propriétaire, mais plutôt de s'appuyer sur l'extensibilité des boîtes à outils pour fournir un système généralisable. Nous nous basons sur les deux opérations de base que sont l'enregistrement et le rejeu. Nous élaborons le système à partir de la bibliothèque Swing de Java. À travers cette bibliothèque on définit un modèle d'architecture sous forme de classes Java, permettant de rejouer un dialogue d'application interactive par la voie interne.

Abstractions élémentaires

Les composants Swing sont entièrement dessinés en Java. Ils ont une indépendance totale vis-à-vis du système. Grâce à cette caractéristique, il est facile de fournir des fonctions génériques et réutilisables sans oublier l'extensibilité de façon à ce que le développeur puisse l'adapter à sa situation particulière.

L'espionnage consiste à observer en permanence des actions de l'utilisateur, à repérer des séquences dans la suite d'actions et à renseigner une liste. Le rejeu récupère les actions dans cette liste et les renvoie à l'application. Toute action est une commande, traduite par un événement. En considérant que les interactions de l'utilisateur sont des suites d'événements, il apparaît que l'enregistrement des actions est la mémorisation des événements. En Java, il est possible de gérer les événements par la file des événements système EventQueue de java.awt.Toolkit et de créer une liste indépendante qui servira de mémoire de sauvegarde aux événements. L'information relative à un événement est encapsulée dans l'objet événement dérivant de la classe java.util.EventObject. De là, on détermine la source de l'événement, l'écouteur ou la destination ainsi que ses propriétés. Les événements sont aussi facile à écouter à l'aide des « Listener » sous Java. Les « Listener » contiennent chacun des méthodes à implémenter par le développeur. Ces méthodes sont appelées lors de la réception d'un événement. En positionnant des « Listener » appropriés sur tous les widgets du système, on écoute les événements lancés par chaque composant qu'il suffit d'archiver. Le rejeu d'une opération s'effectuera par la création d'une liste de séquences à partir de l'archive (historique) et par l'exécution des événements de cette séquence.

Implémenation

Le traitement des événements en Swing peut s'implémenter soit en se basant directement sur la file des événements soit en créant une file (vecteur) propre à l'application. Dans le prémier cas, on crée une file d'événements identique à la file des événements système avec en plus la possibilité de conserver les événements. Puisse la file d'événements utilise la méthode « *dispatch* » pour l'exécution, on effectue une copie ou un clonage (avec la méthode *clone()*) qui sera à son tour exécuté par la méthode. Sinon, on envoie tout simplement l'événement par une notification à sa source qui se chargera de refaire l'opération ayant engendré cet événement.

En utilisant la méthode de la file des événements, on récupère l'ensemble des événements du système. Mais, cela n'est pas très judicieux car la majorité des actions utilisateur est constituée par des tâches articulatoires. Les plus rencontrées sont les déplacements de la souris (mouseMoved, mouseEntered, mouseExited, etc.). Il est donc nécessaire d'installer un filtre pour ne prendre en compte que les événements dont l'éxécution donnerait le résultat final escompté de l'opération tel que le click de la souris (mouse_Clicked). En utilisant les « Listener », on fixe dès le départ les événements à considérer. Le rejeu d'un événement de ce type est réalisé par la méthode *doClick()* ou par la méthode *fireActionEvent*.

CONCLUSION

La réalisation d'un outil pour faciliter l'intégration des fonctionnalités de la programmation sur exemple dans les applications se résume en la généralisation des différents widgets en Swing. Chaque composant est spécialisé de sorte à pouvoir prendre en compte la technique d'envoi des événements pour la construction de l'historique et la possibilté d'extensibilité.

Nous n'avons réalisé qu'un prototype portant sur une partie de cette bibliothèque. La généralisation n'ayant pas été prise en compte dans cet exemple, il est donc évident qu'après la spécialisation des widgets, la notion de contexte et d'état du système lors de l'enregistrement sont les priorités de nos travaux futurs.

BIBLIOGRAPHIE

1. Cypher, A., ed. *Watch What I Do: Programming by Demonstration*. 1993, The MIT Press: Cambridge, Massachusetts. 604.

2. Girard, P., *Ingénierie des systèmes interactifs : vers des méthodes formelles intégrant l'utilisateur*, in *LISI/ENSMA*. 2000, Université de Poitiers p. 92.

3. Lieberman, H., *Your Wish is my command*. 2001: Morgan Kaufmann. 416.

4. Fekete, J.-D. *Les trois services du noyau sémantique indispensables à l'IHM*. in *Journées Francophones sur l'Ingénierie de l'IHM*. 1996. Grenoble: Cépaduès.

5. Myers, B.A., *Taxonomies of Visual Programming and Program Visualization*. Journal of Visual Languages and Computing, 1990. 1(1): p. 97-123.

6. Glinert, E.P., ed. *Visual programming environments: paradigms and systems*. Vol. 1. 1990, IEEE Computer Society Press: Los Alamitos, California. 660.

7. Nicolas Guibert, L.G., Patrick Girard. *Apprendre la programmation par l'exemple : méthode et système*. in *TICE*. 2004. UTC Compiègnes- France.

8. Piernot, P.P. and M.P. Yvon, *The AIDE Project: An Application-Independent Demonstrational Environment*, in *[1]*. p. 383-402.

9. Depaulis, F., L. Guittet, and C. Martin. *Apprends ce que je fais*. in *15ème Conférence Francophone sur l'IHM*. 2003. Caen: ACM Press.

10. Dix, A., et al., *Human-Computer Interaction*. Second Edition ed. 1998: Prentice Hall. 638.