# La programmation sur exemple : principes, utilisation et utilité pour les applications interactives

*Loé SANOU – Patrick GIRARD – Laurent GUITTET*

Laboratoire d'Informatique Scientifique et Industrielle (LISI / ENSMA)
Téléport 2 -1, avenue Clément Ader BP 40109
86961 Futuroscope, France
[sanou, girard, guittet]@ensma.fr

**RESUME**
La programmation sur exemple, issue du besoin d'adaptation des applications interactives aux activités de l'utilisateur, recouvre toute une série de techniques, allant de l'enregistrement-rejeu à l'inférence de règles à partir des interactions de l'utilisateur. Dans ce travail, nous analysons les principes qui sous-tendent la programmation sur exemple, et montrons que, à travers une utilisation de plus en plus large, elle constitue une solution pertinente pour certaines classes de problèmes où elle était jusqu'à présent peu utilisée.

**MOTS CLES :** Programmation sur exemple, Adaptabilité, Adaptativité, Assistance, Tests d'interfaces.

**ABSTRACT**
Programming by Demonstration, which comes from the need to adapt interactive applications to users' activities, stands for some techniques, starting from recording/replaying to rule inference from user interactions. In this work, we analyse the underlying principles of Programming by Demonstration. Then, we demonstrate how, through an increasing usage, it constitutes a pertinent solution for some problem classes, where it has not been largely used yet.

**KEYWORDS :** Programming by demonstration, example-based programming, Adaptable, Adaptative, Assistance, User Interface Testing

**INTRODUCTION**
Depuis l'avènement des interfaces graphiques, les logiciels ont globalement évolué vers une augmentation des fonctionnalités. Les logiciels de bureautique, comme MS-Word© ou MS-Excel©, qui contiennent aujourd'hui plusieurs centaines de fonctions, en sont la parfaite illustration, au point que la majorité des utilisateurs ne connaissent pas, et donc n'utilisent pas, la plus grande part de leurs fonctions. Cette course en avant sur les fonctionnalités aboutit à imposer à l'utilisateur des interfaces de plus en plus lourdes, où les menus s'allongent, et où le nombre d'interactions purement articulatoires augmente de façon déraisonnable. L'émergence des techniques post-WIMP a engendré très récemment une salutaire remise en cause de cette tendance [2].

On peut pourtant considérer que la recherche de solutions à ce problème, dans les années 1980, constitue le point de départ des travaux sur le « End-User programming ». Ainsi, la première réponse a consisté à permettre à l'utilisateur d'adapter son application à ses besoins propres, au moyen de choix prédéfinis, les « Préférences », présentes aujourd'hui dans la majorité des logiciels. Tout naturellement, l'étape suivante a consisté en la possibilité d'adapter l'interface, par exemple en modifiant les items de menus mis à disposition de l'utilisateur. On appelle cette technique la « personnalisation ». Des solutions adaptatives ont ensuite été utilisées avec plus ou moins de bonheur. Les menus « adaptatifs » de certaines versions de MS-Word© ont ainsi été proposés, puis éliminés.

Toutes ces techniques présentent néanmoins l'inconvénient de manquer singulièrement de puissance d'expression. On ne peut en effet ni créer une nouvelle commande par assemblage de commandes existantes, ni même automatiser des enchaînements de commandes. Afin d'augmenter les possibilités d'adaptation, il était tentant de revenir aux principes simples et universels de la programmation. Des langages de scripts ont ainsi été conçus, permettant d'obtenir le degré de puissance souhaité. Malheureusement, cette technique, qui nécessite la maîtrise des principes de la programmation, s'avère inutilisable pour l'utilisateur « commun », le « end-user » des anglo-saxons. Des techniques d'enregistrement de macros ont alors vu le jour, tout particulièrement dans le domaine des systèmes iconiques [16, 17]. La programmation devenait accessible au « end-user ».

L'utilisation accrue de l'image a conduit à jeter les bases de la programmation dite visuelle (« Visual Programming »[12]). Cette dernière s'appuie sur le fait établi que, dans la majorité des cas, la symbolique de l'image apporte une information beaucoup plus importante qu'une simple représentation textuelle. Dans le domaine de la programmation, de nombreux systèmes ont ainsi été développés. Cependant, une compréhension des principes de la programmation est là encore nécessaire. Au-delà de la simple utilisation de l'image, c'est sur celle de l'exemple que se sont concentrés les travaux. L'idée générale est que tout raisonnement autour de l'exemple est

plus intuitif qu'un raisonnement abstrait. C'est ainsi qu'ont vu le jour les systèmes « avec exemple » (Programming with example), qui permettent l'exécution immédiate de l'exemple en cours de construction, et surtout les systèmes « sur exemple » (Programming by Example ou Programming by Demonstration, que nous appellerons PsE par la suite) [10].

La *figure 1* ci-dessous représente une illustration des différentes approches exposées précédemment, selon l'axe de l'engagement par rapport à l'exemple, et de la puissance d'expression de ces solutions.

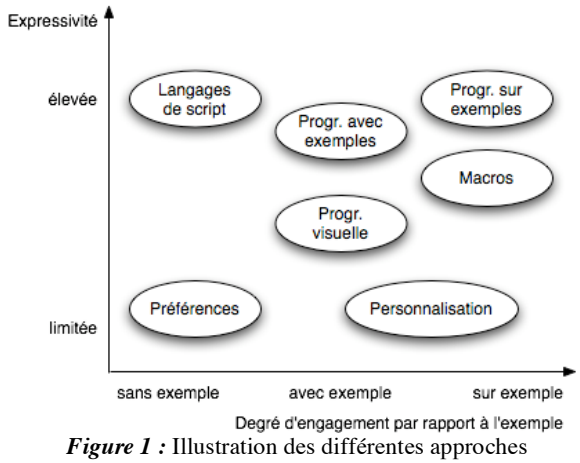


***Figure 1 :*** Illustration des différentes approches

Au-delà de cette trame historique, la programmation sur exemple (PsE) a été expérimentée dans des domaines très variés. Elle apparaît par exemple dans des applications grand public, comme MS-Excel© par exemple. Certains champs d'application ont exploité ses fondements avec beaucoup de réussite, comme la conception technique [11]. Enfin, la « End-User Programming » apparaît dans de nombreuses conférences et workshops.

Le présent papier a pour but de présenter une synthèse des principes, de l'utilisation et de l'utilité des systèmes de programmation sur exemple, qui nous conduira à explorer de nouvelles pistes où sa généralisation nous semble souhaitable. Notre exposé sera découpé en trois points. Dans un premier temps, nous décrivons quelques systèmes de PsE afin d'illustrer de façon concrète notre propos. Dans une deuxième partie, nous analysons les principes sous-jacents. Enfin, nous proposons des axes où il nous semble que la généralisation de son utilisation apporte des solutions originales et efficaces.

## QUELQUES EXEMPLES

Les systèmes de PsE couvrent une grande variété d'applications. Nous décrivons ici quelques systèmes représentatifs des différentes notions que nous détaillerons dans la section suivante.

### Naissance d'un concept

Pygmalion [6] est considéré comme l'un des tous premiers systèmes relevant de la PsE. Son objectif est de permettre la construction d'un programme en s'appuyant sur un exemple. L'environnement du système, semi graphique, est basé sur la métaphore du « tableau noir » : la zone de travail est vue comme un tableau où le programmeur peut « dessiner » ses idées. La *figure 2* présente l'environnement Pygmalion lors de la réalisation d'un exemple : la fonction « factorielle ». Pygmalion est à l'origine du concept d'icône disposant à la fois d'une image, d'un contenu (du texte pour une icône du document), et d'un comportement (déplacement d'une icône par « Drag and Drop »). Pygmalion fournit un menu permettant d'accéder aux icônes et aux opérateurs portant sur ces icônes. Lorsque Pygmalion est en mode enregistrement, une zone d'historique (« remembered ») affiche les deux dernières actions exécutées. Le système dispose aussi d'une zone où le programmeur peut taper et évaluer des expressions écrites en Smalltalk.

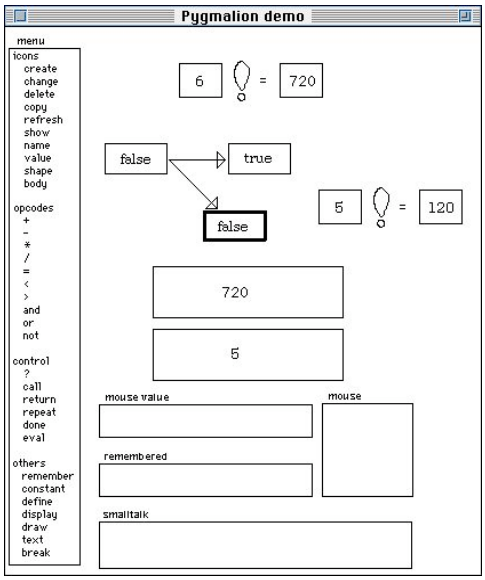


***Figure 2 :*** Pygmalion : Réalisation de la fonction « factorielle ».

Pygmalion fournit des représentations graphiques standard d'opérateurs arithmétiques et booléens. Pour réaliser la fonction « factorielle » par exemple, on commence par allouer une icône. Ensuite l'utilisateur lui associe deux sous icônes pour représenter l'argument et la valeur de la fonction, et il dessine un symbole pour la représenter (*figure 2*). On rend la bordure invisible et on associe un nom avec l'icône par une commande « define ». Le système effectue alors une capture d'écran : la zone de travail sera restaurée dans cet état chaque fois que la fonction sera appelée. L'appel de la fonction « define » fait basculer le système en mode enregistrement. On peut alors spécifier le comportement de la fonction, à l'aide d'un exemple.

### Des macros iconiques

Le domaine des macros est, comme nous l'avons dit en introduction, un domaine privilégié de la programmation par l'utilisateur final. Des systèmes plus ou moins graphiques et interactifs ont été développés, et des révisions modernes sont proposées, comme Automator dans le système Mac OS X, permettant d'automatiser des tâches répétitives comme le renommage de fichiers. SmallStar [16] est une des premières tentatives pour adapter une technique graphique de PsE à cette problématique. SmallStar est une simulation de l'interface graphique du système d'exploitation Star de Xerox [28] à laquelle des capacités de PsE ont été rajoutées. C'est le premier système de PsE destiné au grand public (i.e. des utilisateurs n'ayant pas de connaissance préalable de la programmation) dans le but d'automatiser les tâches répétitives souvent rencontrées lorsqu'ils utilisent Star.

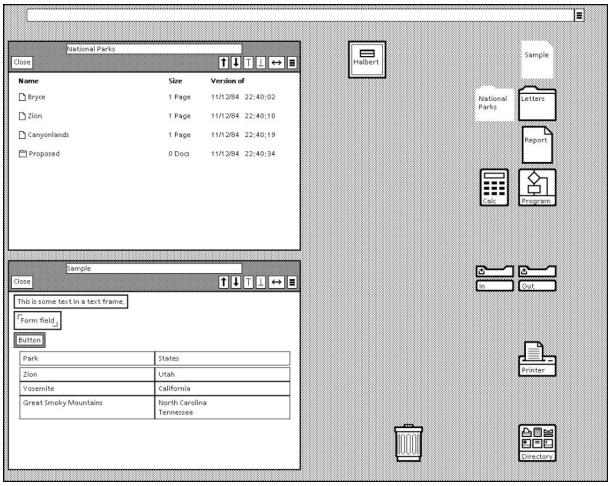


***Figure 3:*** Bureau SmallStar.

Star utilise la métaphore du bureau avec les icônes représentant les différents objets du système (programmes, imprimantes, documents, formulaires, boîtes aux lettres, tableaux, etc.). Dans SmallStar, pour créer un programme, l'utilisateur commence par ouvrir une icône programme puis presse le bouton « Start Recording » pour démarrer l'enregistrement. Il effectue ensuite les étapes nécessaires puis presse le bouton « Stop Recording » pour arrêter l'enregistrement. Une fois la phase d'enregistrement du programme terminée, l'utilisateur doit effectuer les généralisations nécessaires. Pour chaque type d'objet présent dans le système, y compris les commandes enregistrées, SmallStar propose un ensemble de descripteurs de données présenté à l'utilisateur dans une boîte de dialogue. L'utilisateur spécifie explicitement, au moyen d'interfaces textuelles dirigées, les structures de contrôle et les descripteurs de données. Une commande sous SmallStar possède un nom et un ensemble d'arguments ; elle peut impliquer plusieurs actions de la part de l'utilisateur.

### Le domaine de la conception technique

Le domaine de la conception technique (incluant en particulier la CAO, conception assistée par ordinateur) a pleinement assimilé les techniques de la PsE, au point qu'aujourd'hui, les systèmes dits paramétriques, les plus utilisés au plan industriel, fondent leur modélisation sur des principes de PsE.

*Example Based Programming* [25], (EBP) est un exemple d'environnement de développement de programmes paramétrés, susceptibles de générer des comportements standard. Il permet en particulier de mettre au point par manipulation directe et visuelle les programmes générés. EBP utilise des techniques de programmation sur exemple permettant la création de familles de composants standard portables par des utilisateurs de systèmes de conception assistée par ordinateur.

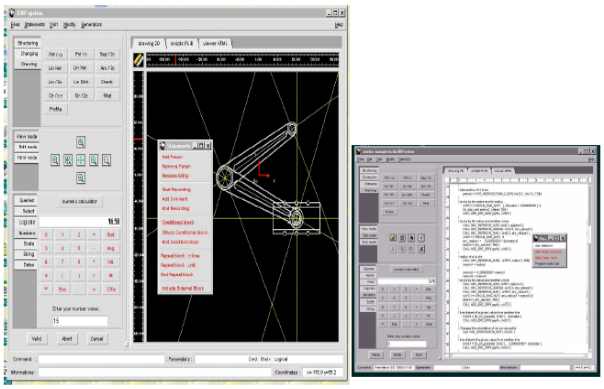

***Figure 4:*** Environnement EBP.

Le système EBP manipule des entités géométriques simples (points, droites infinies, segments, cercles, arcs, etc.) et des entités structurées (courbes, surfaces planes et ensembles structurés). La plupart des contraintes résultant des règles de dessin technique sont supportées. EBP permet de spécifier interactivement, de façon graphique un programme contenant des structures de contrôle usuelles et offre la possibilité de générer du code. La programmation sur exemple est réalisée dans EBP via un mode enregistrement des appels de fonctions du système. Les programmes au sein d'EBP (où ils sont nommés « instances ») sont séparés des exemples. EBP espionne l'utilisateur et construit une instance après l'activation du mode enregistrement. Le passage en mode utilisation permet à l'utilisateur d'exploiter cette instance. Lors de l'exécution de l'instance en mode enregistrement, la définition de la valeur du paramètre effectif est constituée d'une expression quelconque mettant en jeu des entités ou des paramètres du programme appelant. EBP comporte des structures de contrôle complètes. Les sous-programmes, les alternatives et les répétitions sont totalement disponibles.

### Enseignement, éducation, simulation

Fournir une assistance, par l'exemple, à la compréhension des mécanismes de programmation est un thème important dans les systèmes de PsE.

Un environnement d'initiation à la programmation et basé sur l'exemple est MELBA [13] (*Metaphors-based Environment to Learn the Basics of Algorithmic*). C'est un système d'apprentissage composé de trois zones : l'espace du programme qui permet de représenter et d'éditer celui-ci, une zone sémantique (représentant l'exécutant) et une zone pragmatique (représentant graphiquement le sujet du programme). Chaque zone est interactive et la cohérence de l'ensemble est assurée par des mécanismes de programmation sur ou avec exemple. MELBA est un environnement très souple, qui ne préjuge pas de la méthode utilisée pour appréhender les principes de la programmation. Ainsi, l'utilisateur peut-il partir de son savoir-faire sur la tâche et induire un programme avec l'aide du mécanisme de programmation sur exemple ou au contaire chercher à comprendre un programme grâce à l'animation de la zone de la tâche [15].

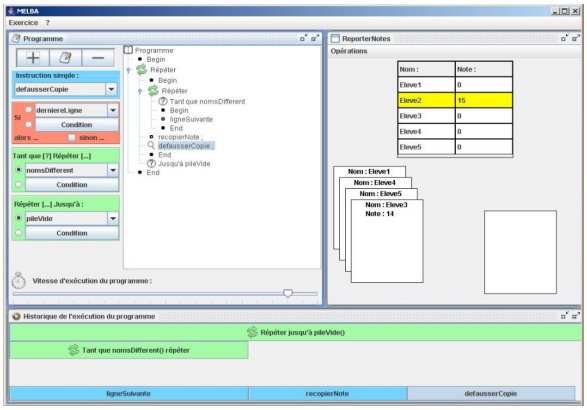

***Figure 5:*** Environnement MELBA
après chargement des données d'un exercice.

### Assistance à l'utilisateur

Eager [5] est un système emblématique de la PsE. Il permet d'automatiser les tâches répétitives, comme par exemple constituer automatiquement une liste numérotée de titres de documents. Il prend en compte le fait que l'utilisateur ne réalise pas toujours immédiatement qu'il peut créer un programme pour accomplir la tâche qui l'intéresse. Eager observe en permanence le comportement de l'utilisateur pour essayer de prédire ses prochaines commandes. Il anticipe la commande suivante de l'utilisateur sans l'exécuter lorsqu'il détecte une boucle. Pour cela, il propose à l'utilisateur la prochaine action à effectuer, à charge pour lui de le confirmer. La *figure 6* illustre l'étape où Eager (le petit chat) propose la prochaine action à l'utilisateur. Si celui-ci confirme l'inférence d'Eager, ce dernier est en mesure de terminer automatiquement la tâche.

Eager fait partie des systèmes qui cherchent à déduire des actions de l'utilisateur ses intentions, pour lui proposer son aide lorsque certains patterns d'interaction, comme une répétition, sont reconnus. De nombreux systèmes ont suivi cette voie, permettant dans certains cas de créer de nouvelles règles à partir des exemples.

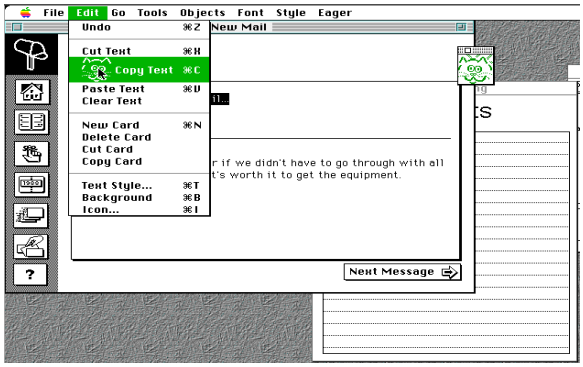


***Figure 6:*** Anticipation de commande dans le menu Eager.

Après ces quelques exemples, nous analysons, dans la section suivante, les principes qui fondent la PsE.

## PRINCIPES DES SYSTEMES DE PSE

La différence essentielle entre la programmation avec exemple et la programmation sur exemple réside dans le fait que cette dernière s'appuie sur une manipulation concrète de l'exemple. De fait, la technique de base consiste à implanter dans un système interactif un système d'enregistrement-rejeu, cher aux systèmes de macros sur exemple [19, 23].

### L'enregistrement-rejeu

Dans l'utilisation de la PsE, il convient de distinguer deux modes qui s'ajoutent ou se superposent à l'utilisation interactive classique du système. Le premier mode est le mode « enregistrement », pendant lequel le système enregistre les actions de l'utilisateur pour construire une trace. Cet enregistrement peut être explicite, comme dans les systèmes de macro, ou implicite comme dans Eager [4]. Dans le premier cas, l'application dispose d'un outil de type « panneau de contrôle de système enregistreur », avec les fonctions « RECORD », « PLAY », « STOP », ou encore « PAUSE » (*figure 7*).

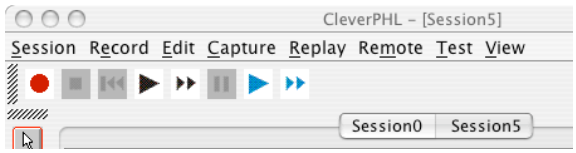


***Figure 7 :*** Un panneau de contrôle d'enregistrement-rejeu, ici dans l'application Jacareto[1]

Ce principe est poussé à son paroxysme dans les systèmes dits paramétriques, qui modélisent leurs données

[1] http://jacareto.sourceforge.net/

comme un historique d'actions constructives, enregistré au cours de la construction interactive de l'objet.

Le niveau d'enregistrement peut être très différent selon les systèmes. Les travaux menés sur les systèmes de conception technique [9, 11, 26, 31] ont permis de caractériser cinq niveaux d'enregistrement-rejeu.

1. Le premier niveau correspond à l'enregistrement des actions les plus élémentaires de l'utilisateur. Les premiers systèmes d'enregistrement de macro fonctionnaient selon ce principe, avec comme corollaire que la situation initiale jouait un rôle prépondérant sur le bon déroulement de la phase de rejeu : les positions graphiques étant enregistrées en absolu, toute différence dans la configuration initiale de l'écran aboutissait à une exécution erronée de la macro. Dans Jacareto, l'enregistrement se passe au niveau de la boîte à outils d'interaction, Swing en l'occurrence ; ce sont ainsi les événements de bas-niveau (MouseEvent, MouseMoveEvent, KeyEvent) qui sont enregistrés et rejoués, permettant un rejeu à l'identique de toutes les actions de l'utilisateur.
2. Le deuxième niveau concerne l'aspect lexical de l'interaction. Les éléments sont enregistrés au niveau de l'action élémentaire, du point de vue de l'utilisateur, et non plus de la boîte à outils. Eager [4] en est la parfaite illustration. Les actions telles que le changement de focus sur une application, ou la sélection d'un item de menu, sont ainsi rejouées pour permettre à l'utilisateur de confirmer les propositions du système.
3. Le troisième niveau correspond à l'aspect syntaxique de l'interaction. Le système Like [9] en est un bon exemple. Dans ce système de conception technique, le dialogue entre l'utilisateur et le système est décrit par le moyen d'automates évolués (des Augmented Transition Networks, ou ATN [32]), manipulant des jetons de dialogue (commandes, paramètres). L'enregistrement de ces jetons constitue la base de l'enregistrement au niveau syntaxique. Nous verrons que ce niveau d'enregistrement présente un intérêt certain pour l'interprétation des actions de l'utilisateur. Notons que, lorsque le style de dialogue est la manipulation directe, comme c'est le cas par exemple pour Peridot [22], il est très difficile de distinguer le niveau lexical du niveau syntaxique : chaque interaction élémentaire est à la fois un élément lexical et un élément syntaxique, puisque la réaction du système est supposée continue.
4. Le quatrième niveau est tout naturellement le niveau sémantique, en référence à la théorie de la compilation. Il s'agit là d'enregistrer la fonction du système invoquée par l'utilisateur, pour être à même de la rejouer directement en phase d'exécution. Dans les systèmes de conception technique, cette approche est représentée par EBP [25], dont le but premier consiste à créer de véritables programmes paramétrés, échangeables d'un système de CAO à un autre. Grammex [21] et StageCast Creator [3] sont deux autres exemples de ce niveau d'enregistrement.
5. Le dernier niveau concerne le niveau pragmatique. Ici, on ne considère que les actions du système qui ont un rôle dans la résolution du but final de l'utilisateur. Les actions articulatoires, ainsi que les opérations non constructives ne sont pas enregistrées. Les systèmes paramétriques en conception technique en sont la parfaite illustration. Comme il est expliqué dans [31], ce sont les données mêmes de l'application qui enregistrent leur historique de construction, ce qui permet de les rejouer après d'éventuelles modifications de paramètres.

Le choix du niveau de l'enregistrement est très important en fonction du but recherché par l'utilisation de la PsE. En effet, plus l'enregistrement s'effectue à un niveau élémentaire dans le dialogue entre l'utilisateur et l'application, et plus il sera dépendant des changements intervenant dans l'interface de l'application. Si l'objectif de la PsE consiste à construire des programmes réutilisables (comme des programmes paramétrés en conception technique, par exemple), tout changement dans l'interface invalidera ces programmes si l'enregistrement se fait à un niveau inférieur au niveau sémantique.

#### La généralisation

Le simple enregistrement-rejeu ne permet pas de faire autre chose que de la répétition à l'identique d'une séquence, qu'elle se situe au niveau de l'interaction ou du système lui-même. Ce qui distingue une simple trace d'exécution d'un vrai programme relève de la généralisation.

Le premier niveau de généralisation consiste à considérer les objets manipulés. En effet, un programme est censé pouvoir s'exécuter sur des « variables », c'est-à-dire avec des valeurs différentes d'une exécution à l'autre. Lorsque l'on travaille sur un exemple, on dispose de valeurs concrètes ; la généralisation consiste à distinguer les valeurs qui vont changer d'une exécution à l'autre de celles qui demeureront constantes. SmallStar [16, 17] est le premier système à se focaliser sur les caractéristiques des objets manipulés par les systèmes de PsE. Il définit une notion de descripteur, qui permet de transformer la référence à un objet unique en une définition sémantique. La notion de variable, bien connue en programmation classique, est évidemment définie. Elle est étendue par des notions plus spécifiques, comme par exemple la notion de collection : une variable peut être généralisée de manière à répondre à une définition générique, ce qui entraîne implicitement une itération de collection. Le système appartient à la catégorie des systèmes iconiques, et la généralisation se traduit par exemple par l'utilisation de joker, comme « *.txt » pour représenter des noms comportant obligatoirement un suffixe particulier. Notons que, dans la majorité des cas, la sémantique de la variable n'a pas besoin d'être connue

par le système d'espionnage, qui se contente d'être capable de référencer l'objet pendant la phase de rejeu. Ceci n'est plus vrai dès lors que l'on ajoute des structures de contrôle au programme.

La deuxième phase de généralisation consiste à ajouter une sémantique de contrôle au programme. Très naturellement, les techniques peuvent relever d'approches différentes. Lorsque l'on s'appuie sur une sémantique de programmation structurée, il convient d'ajouter ou de repérer des alternatives et des répétitions. C'est le cas de SmallStar, par exemple. Les approches fonctionnelles (Pygmalion [29]) ou logiques (StageCast Creator [3]) ont également été explorées. Dans tous les cas, afin de contrôler les structures utilisées, il devient nécessaire de définir parmi les objets une sémantique booléenne. En effet, lors du rejeu, c'est une valeur booléenne qui permettra de choisir quelle alternative exécuter, ou qui dirigera la règle à appliquer. Des expressions booléennes plus complexes pourront s'avérer nécessaires (EBP [25]). Enfin, pour permettre à l'utilisateur de comprendre plus facilement cette sémantique, des expressions numériques seront également introduites.

#### Inférence, règles

La mise en œuvre de la généralisation peut s'effectuer de manières très diverses. Nombreux sont les systèmes qui demandent explicitement à l'utilisateur de faire cette généralisation. SmallStar oblige l'édition du programme enregistré, pour effectuer manuellement la généralisation des objets, et impose l'introduction textuelle des structures de contrôle. Le caractère sur exemple devient moins marqué. D'autres systèmes (Pygmalion, EBP, StageCast Creator) exigent que les structures de contrôle soient prévues par le programmeur, et introduites par des commandes spécifiques.

Pour éviter ce côté explicite, analysé comme contraire à la logique de l'utilisateur non-programmeur, plusieurs auteurs ont choisi de proposer des méthodes utilisant l'exemple de façon plus marquée. Ainsi, PBE [1] se sert-il de deux exécutions différentes pour déduire quelles sont les variables et quelles sont les constantes. Eager [4] demande à l'utilisateur de confirmer les actions qu'il a détectées, et analyse les déviations pour construire le programme correct.

Allant plus loin sur la voie de l'inférence, c'est-à-dire l'induction de faits nouveaux à partir de faits connus, de nombreux auteurs ont couplé des moteurs d'inférence aux systèmes de PsE. Eager [4] ou Mondrian [20] reconnaissent ainsi des situations complexes. Des résultats probants ont ainsi été obtenus dans le domaine de la simulation. La définition de nouvelles règles par l'utilisateur est considérée comme l'étape suivante : Grammex [21] permet ainsi de définir explicitement, sur exemple, de nouvelles règles qui s'appliqueront ensuite comme des règles pré-existantes dans le système. Ces approches, quoique séduisantes, se trouvent le plus souvent limitées par leur pouvoir d'expression : elles sont spécifiques du domaine d'application, et peuvent difficilement être généralisées. Elles se heurtent à un deuxième obstacle : lorsque l'on construit de nouvelles règles, il est difficile d'être sûr qu'elles s'appliqueront sans erreur dans tous les cas de figure.

### DES CHAMPS D'APPLICATION EN EXPANSION

La PsE a donné lieu à de nombreux systèmes. Le reproche que l'on peut faire à ces travaux est le manque d'évaluations conduites à leur propos. En dehors de MELBA, pour lequel plusieurs expérimentations ont été réalisées, aucune démarche formelle d'évaluation n'a été engagée. On peut considérer néanmoins que la commercialisation de systèmes relevant de la PsE constitue une forme d'évaluation. La plus grande réussite est certainement celle des systèmes de conception technique dits paramétriques, qui constituent aujourd'hui la majorité des systèmes de cette catégorie. Un autre domaine notable est celui de la simulation, où des logiciels comme StageCast Creator[2] ou AgentSheets[3] ont été commercialisés.

Trois domaines se dégagent comme particulièrement pertinents du point de vue de la PsE.

#### Fonction d'assistance

Il s'agit ici d'un retour aux sources. La possibilité de personnaliser son application est un besoin considéré comme majeur par les utilisateurs. De nombreux éditeurs de logiciels ont ainsi développé des solutions s'appuyant peu ou prou sur les techniques de la PsE pour enrichir les possibilités d'adaptation. Les techniques de généralisation *a priori* sont utilisées dans l'outil Automator[4] qui permet d'automatiser des tâches dans Mac OS X. Les techniques d'inférence et d'espionnage trouvent leur utilité dans des agents d'aide, ou autres assistants. Dans MS-Excel©, par exemple, un espionnage systématique de l'utilisateur permet de détecter les situations s'apparentant à une gestion de liste (*Figure 8*), comme le fait Eager.

Les techniques de la PsE, utilisées de façon ponctuelle dans certaines situations d'interaction, sont de nature à apporter de nouvelles fonctions d'assistance à l'utilisateur. Selon les besoins, des techniques explicites de généralisation seront préférables à l'utilisation de l'inférence, principalement lorsqu'on souhaitera que l'utilisateur puisse contrôler finement son automatisation. Elles sont la base de l'adaptabilité des applications. L'usage de l'inférence permet en plus d'atteindre des objectifs d'adaptativité.

---

[2] http://www.stagecast.com/
[3] http://www.agentsheets.com/
[4] http://www.apple.com/fr/macosx/features/automator/

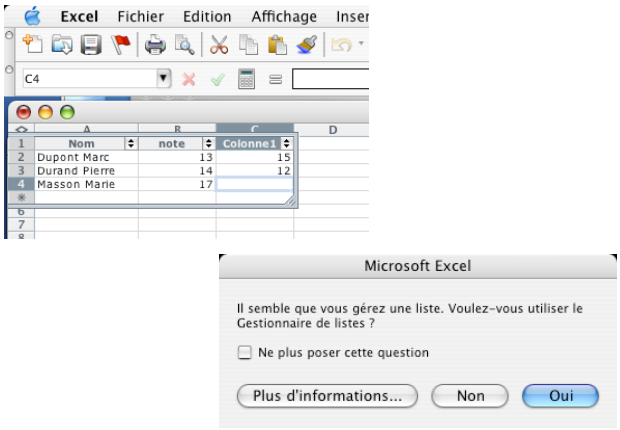


***Figure 8 :*** L'exemple d'utilisation de la PsE dans MS-Excel©

### Pédagogie

Les applications pédagogiques et de simulation constituent un champ d'investigation important pour la PsE. Les objectifs des systèmes qui l'utilisent sont très variés. Ils peuvent concerner l'apprentissage de la programmation (MELBA [14], Toontalk[5][18] ), ou plus généralement de la stratégie de résolution de problème (StageCast Creator [30], Agentsheets [27]). L'effort principal se porte dans ces systèmes sur l'aide à l'explicitation des principes de généralisation. De ce fait, un effort tout particulier est porté sur la visualisation des programmes, qui cherche à être la plus explicite possible. Les avancées dans le domaine des nouvelles techniques d'interaction trouvent également leur application dans la PsE. Ainsi, l'application des techniques des interfaces tangibles a donné lieu au Topobo[6], qui permet à de jeunes enfants à étudier la dynamique des actions (et donc la programmation) en s'appuyant sur la manipulation de d'objets de construction qui mémorisent leurs actions et les rejouent.

### Outils de conception et de validation

Le domaine des outils de conception est certainement celui où la PsE est intégrée le plus profondément aux applications. Les outils graphiques modernes utilisent abondamment l'analyse temps-réel des interactions pour fournir une assistance au placement des composants (alignement, contraintes, etc.). On trouve ces fonctionnalités dans des outils de dessin (OmniGraffle[7]), de présentation (Keynote[8]) ou encore les GUI-Builders (Matisse[9] dans NetBeans).

L'utilisation des techniques d'enregistrement-rejeu s'avère extrêmement prometteuse pour la vérification et la validation des applications interactives. Alors que les techniques de tests unitaires font de plus en plus d'émules[10], et que leurs outils sont de plus en plus largement utilisés, tester les interfaces s'avère encore un processus difficile. Les techniques d'analyse de code ont largement montré leurs limites, et la validation par les modèles n'est pas encore opérationnelle au niveau industriel. Aujourd'hui, ce sont les approches intégrant la PsE qui semblent émerger. Ainsi, Jacareto[11] permet-il d'enregistrer les interactions de bas-niveau et autorise-t-il l'automatisation de tests d'interface. Pour rendre ces outils plus puissants, un couplage avec les approches à base d'analyse de tâches, comme dans [8], s'avère très efficace.

## CONCLUSION

La programmation sur exemple recouvre une série de techniques, allant de l'enregistrement-rejeu à l'inférence de règles à partir des actions de l'utilisateur. La constante de ces techniques est l'utilisation intensive d'un exemple tiré de l'action de l'utilisateur sur un système interactif.

Les différentes facettes de la PsE ont été décrites et validées dans de nombreux systèmes expérimentaux. Alors que la PsE véhicule une image de technique à la portée limitée, on s'aperçoit qu'elle est de plus en plus souvent utilisée dans les applications interactives. À partir d'une présentation d'exemples significatifs, nous avons analysé les principes qui sous-tendent la PsE. Nous avons pu ainsi définir cinq niveaux possibles dans l'enregistrement des actions de l'utilisateur, qui s'appliquent à des problématiques différentes. Nous avons ensuite catégorisé les diverses formes de généralisation des programmes. Enfin, dans une dernière partie, nous avons exploré quelques grands secteurs où la PsE apporte des résultats significatifs, en terme d'augmentation des possibilités des systèmes.

La généralisation des techniques de PsE passe, à notre sens, par la définition d'outils permettant une implémentation généralisée de ces techniques. Cela est vrai au niveau des techniques d'enregistrement, ou des approches internes (Aide [24]) comme externes (PbDScript [7]) doivent être proposées. Cela est vrai également pour les techniques de généralisation, qui doivent être abordées de façon générique. Les champs d'investigation demeurent ici largement ouverts. Pour terminer, notons que le point faible des travaux sur la PsE réside dans les études d'utilisabilité, qui n'ont été que très rarement conduites jusqu'à présent. Il y a là également de grandes possibilités pour de futurs travaux.

## BIBLIOGRAPHIE

1. Bauer, M.A., Programming by Examples. Articial Intelligence, 1979. 12: pp. 1-21.

[5] http://www.toontalk.com/
[6] http://web.media.mit.edu/~hayes/topobo/
[7] http://www.omnigroup.com/applications/omnigraffle/
[8] http://www.apple.com/iwork/keynote/
[9] http://www.netbeans.org/kb/articles/matisse.html

[10] http://www.extremeprogramming.org/, http://www.junit.org/
[11] http://jacareto.sourceforge.net/

2. Beaudouin-Lafon, M. Designing interaction, not interfaces. in Advanced Visual Interfaces. 2004. Gallipoly, Italie, pp. 15-22.
3. Canfield Smith, D., Cypher, A., and Tesler, L., Novice Programming Comes of Age, in Your Wish is My Command, H. Lieberman, Editor. 2001. pp. 7-20.
4. Cypher, A. Eager: Programming Repetitive Tasks by Example. in Human Factors in Computing Systems (CHI'91). 1991. New Orleans, Louisiana: ACM/SIGCHI, pp. 33-39.
5. Cypher, A., Eager : Programming Repetitive Tasks by Demonstration, in Watch What I Do: Programming by Demonstration, A. Cypher, Editor. 1993, The MIT Press: Cambridge, Massachusetts. pp. 205-217.
6. Cypher, A., ed. Watch What I Do: Programming by Demonstration. 1993, The MIT Press: Cambridge, Massachusetts. 604p.
7. Depaulis, F., Guittet, L., and Martin, C. Apprends ce que je fais. in 15ème Conférence Francophone sur l'Interaction Homme-Machine. 2003. Caen: ACM Press, pp. 236-239.
8. Fabio Paternò, C.S., Preventing user errors by systematic analysis of deviations from the system task model. International Journal of Human-Computer Studies, 2002. 56(2): pp. 225-245.
9. Girard, P., Environnement de Programmation pour Non-Programmeur et Paramétrage en Conception Assistée par Ordinateur : le système LIKE. Thèse: Université de Poitiers. 195p.
10. Girard, P., Ingénierie des systèmes interactifs : vers des méthodes formelles intégrant l'utilisateur. HDR: Université de Poitiers. 92p.
11. Girard, P., Bringing Programming by Demonstration to CAD Users (Chapter 7), in Your wish is my command, H. Lieberman, Editor. 2001, Morgan Kaufmann. pp. 135-162.
12. Glinert, E.P., ed. Visual programming environments: paradigms and systems. Vol. 1. 1990, IEEE Computer Society Press: Los Alamitos, California. 660p.
13. Guibert, N. and Girard, P. Programmation sur Exemple et Enseignement assisté par ordinateur de l'algorithmique : le projet MELBA. in 15° Conférence Francophone sur l'Interaction Homme-Machine (IHM'2003). 2003. Caen: ACM Press, pp. 248-251.
14. Guibert, N., Girard, P., and Guittet, L. Example-based Programming: a pertinent visual approach for learning to program. in Advanced Visual Interfaces. 2004. Gallipoli, Italy: acm press, pp. 358-361.
15. Guibert, N., Guittet, L., and Girard, P. Apprendre la programmation par l'exemple : méthode et système. in Technologies de l'Information et de la Connaissance dans l'Enseignement Supérieur et l'Industrie (TICE). 2004. UTC Compiègnes- France.
16. Halbert, D., Programming by Example. PhD Thesis: University of California. 121p.
17. Halbert, D., SmallStar : Programming by Demonstration in the Desktop Metaphor, in Watch What I Do : Programming by Demonstration, A. Cypher, Editor. 1993, The MIT Press: Cambridge, Massachusuetts. pp. 102-123.
18. Kahn, K., How Any Program Can Be Created by Working with Examples, in Your Wish is My Command, H. Lieberman, Editor. 2001. pp. 21-44.
19. Kurlander, D. and Feiner, S. A History-Based Macro by Example System. in ACM Symposium on User Interface Software and Technology (UIST'92). 1992. Monterey, California: ACM/SIGCHI, pp. 99-115.
20. Lieberman, H., Mondrian: a Teachable Editor, in Watch What I Do: Programming by Demonstration, A. Cypher, Editor. 1993, The MIT Press: Cambridge, Massachusetts. pp. 341-360.
21. Lieberman, H., Nardi, B.A., and Wright, D. Grammex : Defining Grammar by Example. in Human Factors in Computing Systems (CHI'98). 1998. Los Angeles, California: ACM/SIGCHI, pp. 11-12.
22. Myers, B.A., PERIDOT: Creating User Interfaces by Demonstration, in Watch What I Do: Programming by Demonstration, A. Cypher, Editor. 1993, The MIT Press: Cambridge, Massachusetts. pp. 125-154.
23. Olsen, D.R. and Dance, J.R., Macros by Example in a Graphical UIMS. IEEE Computer Graphics and Applications, 1988. 12(1): pp. 68-78.
24. Piernot, P.P. and Yvon, M.P., 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. pp. 383-402.
25. Potier, J.-C., Conception sur exemple, mise au point et génération de programmes portables de géométrie paramétrée dans le système EBP. Thèse: Université de Poitiers. 140p.
26. Potier, J.-C., et al., Génération graphique interactive de programmes de géométrie paramétrée. Revue d'Automatique et de Productique Appliquée (RAPA), 1995. 8(2-3): pp. 229-234.
27. Repenning, A. and Perrone, C., Programming by Analogous Examples, in Your Wish is My Command, H. Lieberman, Editor. 2001. pp. 351-370.
28. Smith, D., et al., Designing the Star User Interface. Byte, 1982. 7(4): pp. 242-282.
29. Smith, D.C., A Computer Program to Model and Stimulate Creative Thought. 1977, Basel: Birkhauser. 187p.
30. Smith, D.C., Cypher, A., and Tesler, L., Novice Programming Comes of Age, in Your Wish is My Command, H. Lieberman, Editor. 2001. pp. 7-20.
31. Texier, G., Contribution à l'ingéniérie des systèmes interactifs : Un environnement de conception graphique d'applications spécialisées de conception. Thèse: Université de Poitiers. 230p.
32. Woods, W., Transition Network Grammars for Natural Language Analysis. Communications of the ACM, 1970. 13(10): pp. 591-606.