Girard P., Patry G., Pierra G., & Potier J.-C. *Deux exemples d'utilisation de la Programmation par Démonstration en Conception Assistée par Ordinateur*. Revue Internationale de CFAO et d'informatique graphique. 1997. vol. 12,n° 1-2, p. 169-188.

Deux exemples d'utilisation de la Programmation par Démonstration en Conception Assistée par Ordinateur

Patrick Girard, Guillaume Patry, Guy Pierra, Jean-Claude Potier

Laboratoire d'Informatique Scientifique et Industrielle
Ecole Nationale Supérieure de Mécanique et d'Aérotechnique
Site du Futuroscope - B.P. 109 - 86960 FUTUROSCOPE Cedex - France
e-mail : {girard,patry,pierra,potier}@ensma.univ-poitiers.fr

RÉSUMÉ. La programmation par démonstration (« Programming by Demonstration » ou PbD) est un concept récent qui a émergé grâce aux progrès réalisés dans le domaine de l'architecture des interfaces homme-machine. Son but est de permettre à un utilisateur final, peu ou même pas du tout versé dans « l'art de la programmation », de réaliser de réels programmes. Nécessitant des domaines d'application fortement graphiques, la PbD a trouvé dans la CAO un champ d'expérimentation naturel. Dans cet article, nous montrons comment l'application de ces concepts peut permettre de résoudre des problèmes tels que la conception et l'échange entre systèmes différents de familles de composants standard, ainsi que l'extension d'un système existant par ajout de nouveaux types d'objets.

ABSTRACT. Programming by Demonstration (PbD) is a new concept which has emerged with the advances in human computer interaction. It's goal is to allow end-users, with or without programming knowledge, to build real programms. CAD is a natural candidate for PbD. In this paper, we show how these concepts can help building and exchanging standard parts, and how existing systems can be extended by adding new kinds of objects.

MOTS-CLEÉS : Programmation par Démonstration ou Programming by Demonstration (PbD), Programmation basée sur Exemples, Conception Assistée par Ordinateur (CAO), Systèmes paramétriques, Composants standard.

KEY WORDS : Programming by Demonstration (PbD), Example-Based Programming, Computer Aided Design (CAD), Parametric Systems, Standard Parts.

Introduction

Au cours des dernières années, de nombreux travaux ont été entrepris dans le but de rendre plus abordable l'utilisation et la programmation des ordinateurs. La Programmation Visuelle [GLI 90] a ainsi permis aux utilisateurs de manipuler graphiquement les programmes ; mais c'est la Programmation par Démonstration [CYP 93] qui a permis d'aller le plus loin, en autorisant la manipulation directe des valeurs d'un exemple en lieu et place d'objets abstraits (les variables). De nombreux systèmes expérimentaux ont démontré la fécondité et l'intérêt d'une telle approche. Cependant, aucun système actuel n'a atteint le pouvoir d'expression d'un environnement de programmation classique, et les systèmes commerciaux tardent à émerger. L'objectif de cet article est de présenter deux applications des principes de ce type de programmation qui permettent de résoudre deux problèmes différents qui interviennent dans le domaine de la CAO : (1) la création et l'échange de familles de composants standard, et (2) l'adaptation et l'extension d'un système existant par adjonction de nouvelles fonctions et de nouveaux types d'objets.

Dans la première section, nous verrons que la PbD est déjà présente dans le domaine de la CAO, à travers les systèmes dits paramétriques. Puis, nous détaillerons le système EBP dans la deuxième section. Enfin, la troisième section nous permettra d'aborder le problème de l'adaptation d'un système CAO.

1. PbD et CAO

Dans son livre, *A small matter of programming, perspectives on end-user computing* [NAR 93], B. Nardi identifie la CAO comme un candidat naturel à l'application des techniques de programmation par démonstration. Nous verrons dans cette section que les systèmes dits paramétriques ont déjà permis de franchir un premier pas en ce sens.

Dans les années 1980, le développement des interfaces graphiques a conduit, à partir d'une idée simple, le remplacement des mots par des images, à la notion de « Visual Programming » [GLI 90]. Des systèmes divers, tant expérimentaux (par exemple, HI-VISUAL [HIR 90]) que commerciaux (comme LabVIEW®[1]) ont ainsi pu être développés. Ils permettent de choisir graphiquement fonctions et variables et de les agencer dans un espace en deux dimensions pour décrire le contrôle du programme. Utiliser un exemple pour concevoir le programme constitue l'étape suivante permettant d'aller vers des environnements de programmation pour utilisateurs finaux. Au lieu de manipuler fonctions abstraites et variables, il s'agit d'*exécuter* des fonctions *sur des valeurs,* tout en enregistrant le programme abstrait. Ce concept de programmation est né avec Pygmalion [SMI 75], et fut formalisé par Myers [MYE 90a] sous le nom de Programmation basée sur Exemple (Example-

---

1 National Instruments, USA

Based Programming). Plus récemment, le terme de Programmation par Démonstration (Programming by Demonstration) a été proposé dans le recueil de Cypher [CYP 93]. L'idée principale consiste à permettre à l'utilisateur de n'interagir qu'avec les valeurs des objets. Pendant la conception de l'exemple, le système de PbD analyse les entrées de l'utilisateur qui réalise l'exemple, et en abstrait le programme capable de regénérer l'exemple, ainsi que toute une famille de variantes. Cette analyse peut être faite par enregistrement des actions effectuées (comme les enregistreurs de Macros), ou par inférence sur l'exemple entier. Myers distingue ainsi les notions de « **programming-with-example** » (premier cas) et de « **programming-by-example** » (deuxième cas).

La PbD ouvre la porte à de nouveaux environnements de programmation. Ainsi, SmallStar [HAL 84] pour la macro-programmation de systèmes icôniques, PERIDOT [MYE 88], GARNET [MYE 90b], MACROS BY EXAMPLE [OLS 88] pour les interfaces utilisateur, KidSim [CYP 95] pour la simulation, ou encore Geometer's Sketchpad [JAC 93] et ProDeGE+ [SAS 94] pour les outils graphiques, ont-ils prouvé dans différents domaines l'intérêt de cette approche. Malgré ces succès, la majorité des systèmes de PbD n'ont pas dépassé aujourd'hui le stade de prototypes. Dans la CAO cependant, la PbD a trouvé, avec les « systèmes paramétriques », un réel marché.

1.1. Les raisons du succès

Deux raisons majeures du succès de la PbD dans le domaine de la CAO peuvent être avancées. La première tient à la nature de l'activité de conception technique, la seconde au marché des composants standard.

L'activité de conception technique (qu'elle soit mécanique, architecturale ou autre) suppose l'application de règles strictes dépendant du domaine. En cela, elle n'a rien à voir avec l'activité beaucoup libre de conception « artistique ». Ainsi, au cours du processus de conception, le dessinateur technique connait-il parfaitement les relations qui doivent exister entre les différents éléments de son modèle en cours d'élaboration. De ce fait, les systèmes CAO lui permettent d'exprimer explicitement ces relations, généralement sous la forme de contraintes ou d'opérateurs géométriques, et les utilisateurs sont habitués à exprimer ces relations. Le simple fait d'enregistrer ces contraintes fournit une base pour la construction automatique de programmes.

La seconde raison se déduit de l'observation suivante : concevoir un nouveau produit consiste souvent à assembler des composants pré-existants. Ces composants, appelés « composants standard » ou « standard parts », sont regroupés en familles décrites par un modèle unique, qui exprime des variations possibles en dimensions, tolérances ou forme [SHA 95]. Ces composants peuvent être décrits dans des normes (par exemple, la famille de vis hexagonales ISO 1014), ou bien être propres à un fournisseur ou une entreprise utilisatrice. Dans la première génération de

systèmes CAO, ces familles étaient décrites sous la forme de programmes paramétrés, en langages généralistes (FORTRAN) ou spécialisés. L'inconvénient majeur de cette approche réside dans le fait qu'une double compétence (dessin technique / informatique) est alors requise. Écrire de véritables programmes n'est pas à la portée de tous les utilisateurs, même dessinateurs. La PbD offre alors une solution séduisante pour permettre la production de ces programmes par des dessinateurs

1.2. Les systèmes variationnels et paramétriques

Alors que tous les systèmes modernes de CAO offrent la possibilité d'exprimer les contraintes entre objets lors de la création de ces derniers, l'enregistrement de ces contraintes dans le modèle n'est apparu en fait que très récemment. Bien que le système MEDUSA [NEW 83] puisse être considéré comme le premier système ayant fourni la possibilité d'enregistrer des contraintes, dès 1983, la généralisation de cette possibilité ne s'est faite qu'à la fin des années 80. Une nouvelle génération de systèmes est alors apparue, capable de regénérer un nouveau modèle après modification d'une ou plusieurs contraintes enregistrées dans un modèle donné. En fait, ces systèmes, appelés « Dimension Driven Systems » (DDS) [ROL 90], ont une structure interne et un comportement doubles. Tout d'abord, ce sont des systèmes classiques, qui permettent à l'utilisateur de créer un modèle, de le visualiser, et de le modifier. Mais de plus, l'utilisateur peut généraliser son modèle (et ainsi créer un programme) en demandant la visualisation des contraintes et des valeurs qui sont impliquées dans le modèle ; la modification des valeurs permet au système de générer automatiquement un nouveau modèle, correspondant à l'application des mêmes contraintes sur ces nouvelles valeurs. En fait, ces systèmes peuvent être divisés en deux catégories, qui procèdent de deux approchent radicalement différentes, les approches déclarative et impérative [PIE 94b].

Les **systèmes Variationnels** cachent un **programme déclaratif** . L'utilisateur construit un exemple, et spécifie les contraintes, implicitement ou explicitement. Le système enregistre ces contraintes sous forme d'un ensemble d'équations, dont un solveur peut tirer une solution. La méthode de création de l'objet peut demeurer inconnue du dessinateur. De nombreuses méthodes ont été utilisées pour résoudre ces systèmes de contraintes, les plus efficaces semblant être aujourd'hui basées sur la réduction de graphes [BOU 95][OWE 91]. Malgré de réels progrès, cette approche souffre de certaines limitations : (1) L'ensemble d'équations a généralement plus d'une solution, et seules des heuristiques ont pu être définies pour capturer « l'intention » de l'utilisateur. Des exemples simples ont été publiés [BOU 95] qui ont démontré le côté inattendu des solutions parfois trouvées. (2) À cause de la nature "heuristique" de ces solutions, deux systèmes différents peuvent ne pas trouver la même solution. (3) L'approche variationnelle pure n'est pas applicable à beaucoup d'opérations, principalement en 3D. C'est pourquoi tous les systèmes dits

variationnels sont en fait hybrides, et utilisent une approche variationnelle en 2D et l'approche suivante en 3D.

Les **systèmes fonctionnels**, souvent appelés **systèmes Paramétriques**, cherchent à résoudre un problème très différent : étant donnée une classe de composants dont le processus de conception est bien connu, on veut que chaque instance de la classe puisse être générée automatiquement à partir de la valeur des paramètres qui la caractérise. Ces systèmes cachent une composition de fonctions analogue à un **programme impératif**. Ce programme est souvent capturé au cours de l'exécution du système interactif : le système CAO « espionne » le dessinateur au fur et à mesure que ce dernier construit son exemple. Par la suite, le système CAO est à même de rejouer la logique de construction, avec éventuellement de nouvelles valeurs. La représentation interne des programmes peut être textuelle, mais elle est plus généralement basée sur des structures de données [PIERRA 94][SOL 94] telles que les graphes orientés acycliques [CUG 88]. Le système Pro-Engineer®[2] est l'exemple le plus populaire de cette approche. Notons que rien n'interdit d'éditer les fonctions de construction, voire de les modifier. Les systèmes de type fonctionnel ont donc un domaine d'application beaucoup plus large que la simple réexécution d'un historique.

Dans le domaine de la CAO, l'approche DDS est si séduisante qu'aujourd'hui, tous les systèmes de CAO se doivent de posséder de telles possibilités. Cette diffusion très large démontre l'intérêt pratique de l'approche. Elle démontre également que les dessinateurs, utilisateurs finaux, sont capables de générer des formes paramétrées, autrement dit de réels programmes visuels, sans notion de programmation. Cela ne signifie cependant pas absence de modification du processus de conception. En effet, dessiner un modèle est sensiblement différent de dessiner une famille de modèles. Cependant, cette activité ne se situe pas au niveau d'abstraction de la programmation classique, et les systèmes paramétriques réduisent considérablement l'effort nécessaire pour la conception de familles de composants.

2. PbD et composants standard : le système EBP

Le premier exemple d'application que nous avons développé consiste à utiliser les techniques de PbD pour permettre la création de familles de composants standard portables par des utilisateurs de systèmes CAO. Nous commencerons par établir ci-dessous le cahier des charges du système, puis nous en décrirons les caractéristiques.

---

2 Parametric Technology Inc. USA.

2.1. Le cahier des charges d'EBP

La portabilité des catalogues de composants entre systèmes CAO différents est un point d'importance économique majeure pour les utilisateurs de systèmes CAO, les fournisseurs de composants, ainsi que les constructeurs de systèmes CAO. Elle devrait augmenter sensiblement le nombre de familles de composants disponibles sur différents systèmes, et par voie de conséquence, augmenter la qualité et la productivité générales. L'approche CAD-LIB a ainsi été developpée [PIE 94a], et constitue la base commune de travaux européens et internationaux de normalisation (CEN/TC310-pr ENV 40004 et ISO/TC184/SC4-ISO 13584 P-LIB).

En 1993, un projet dénommé PLUS (Parts Library Support and Supply) était lancé. Ce projet, soutenu par l'Union Européenne au sein du programme de Recherche et Développement ESPRIT, avait pour objectifs :

- de développer, sur la base des concepts de CAD-LIB, une spécification complète du format d'échange pour publication en tant que norme ISO 13584 (P-LIB),
- de valider cette spécification à travers le développement d'un ensemble complet d'outils pré-industriels capables de générer ou d'utiliser ce format d'échange.

Au-delà d'un modèle de données orienté-objet pour l'échange des données des bibliothèques de composants, le projet devait développer une approche permettant l'échange de familles de modèles géométriques. Au début du projet, les technologies variationnelle et paramétrique n'apparaissaient pas suffisamment mûres pour permettre le développement d'une norme de format d'échange pour les modèles paramétriques. Aussi, l'approche choisie fut-elle résolument « conservatrice ». Elle a consisté à développer une Interface de Programmation d'Application (API) standard (maintenant disponible sous le nom de ISO DIS 13584-31) associé à un couplage FORTRAN. Tous les systèmes CAO supportant une implémentation de cette API seraient ainsi capables d'exécuter un programme FORTRAN se référant à cette API.

Cette approche était cependant moins conservatrice qu'elle n'en avait l'air au premier coup d'oeil : le projet incluait également le développement d'un système PbD destiné à générer ces programmes au moyen d'interactions purement graphiques.

Ce contexte définit le cahier des charges de la conception du système EBP. (1) Le processus de génération doit être déterministe, et contrôlé pleinement par l'utilisateur : une seule forme doit être générée pour un ensemble quelconque de valeurs pour les paramètres d'entrée, et cette forme doit effectivement correspondre à un composant. (2) Toutes les familles descriptibles à l'aide d'une manière conventionnelle de programmer doivent pouvoir être construites avec le système. (3) Le système doit pouvoir générer une représentation externe de sa structure de données interne dans le format défini par l'API normalisée (i.e. un programme FORTRAN).

On peut noter que, parmi les systèmes paramétriques actuels (le premier point ci-dessus exclut le recours à toute forme d'inférence ou d'heuristique, donc exclut l'approche variationnelle), aucun d'entre eux ne remplit en totalité les conditions émises en (2) et en (3).

2.2. Le système EBP

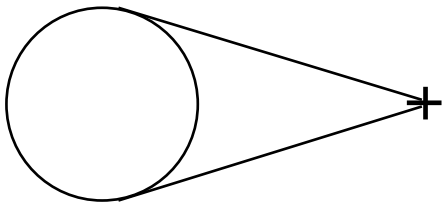
Le système EBP (Example Based Programming) [POT 95] est un système de CAO 2D. Il manipule des entités géométriques simples (points, droites infinies, segments, cercles, arcs, etc.) et des entités structurées (courbes, surfaces planes et ensembles structurés). La plupart des contraintes résultant des règles du dessin technique sont supportées. EBP fournit de plus une calculatrice puissante qui permet de combiner nombres et entrées graphiques dans des expressions grapho-numériques. Enfin, EBP permet la construction du modèle à travers une interface basée sur la norme X-MOTIF, et tourne actuellement sur stations Sun-Solaris[3] et DEC-Alpha[4]. Il est actuellement en cours de portage sur PC-WINDOWS NT.

L'annexe 1 montre une copie d'écran du système EBP « classique ». Cette figure montre en outre un élément moins commun dans les systèmes CAO : la calculatrice grapho-numérique. Cette dernière, qui a quelques points communs avec le système Pygmalion [SMI 75], permet aux dessinateurs un usage semblable à celui des calculatrices numériques, mais intégrant les objets graphiques (dit = distance de deux entités, ang = angle d'un arc, ...). Une deuxième calculatrice, logique cette fois, permet de spécifier graphiquement le prédicat de contrôle des structures alternatives et répétitives rencontrées dans les familles de pièces (voir annexe 2).

2.2.1. Un système totalement déterministe

L'ambiguïté des constructions géométriques n'est pas spécifique des systèmes variationnels. Elle est en fait intrinsèque aux problèmes géométriques non linéaires, par exemple lorsqu'ils font intervenir des contraintes sur des cercles.

Ainsi, construire une ligne à partir d'un point et devant être tangente à un cercle conduit à deux solutions possibles, comme le montre la figure 1 :



3 Sun Inc., USA

4 Digital, USA

Figure 1 : *Les deux solutions possibles pour une ligne tangente à un cercle*

Ce problème est résolu à la perfection dans un système interactif... La plupart des systèmes CAO utilisent la position de la souris au moment de la désignation de chaque objet pour distinguer les solutions possibles. Ils supposent alors que l'utilisateur connaît approximativement la solution qu'il désire. Par exemple, dans le cas de la figure 1, le clic souris permet de choisir la solution supérieure (figure 2).

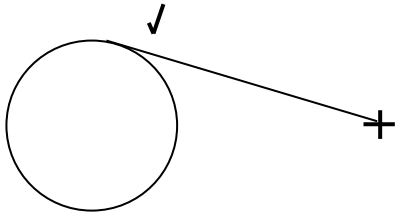


Figure 2 : *Résolution par pointé*

Si cette convention de dialogue peut être utilisée au niveau interactif, et donc par exemple lors de la conception de l'exemple dans un système paramétrique, elle ne peut être conservée en l'état dans le programme où, pour des valeurs différentes des paramètres, la position peut correspondre à des solutions différentes. Ce problème n'est pas spécifique de la PbD, mais se retrouve dans de nombreux cas de programmation géométrique [ROL 90].

Dans ces derniers cas, des éléments de discrimination de solutions hors-contextes ont pu être définis. Ainsi, dans l'API cible, la résolution des ambiguïtés est-elle assurée à l'aide d'informations topologiques supplémentaires : l'orientation des entités géométriques et la notion d'intérieur/extérieur pour les cercles. Par exemple, la figure 3 montre la solution unique de la fonction de création d'un segment passant par un point et un cercle de la figure 2 :

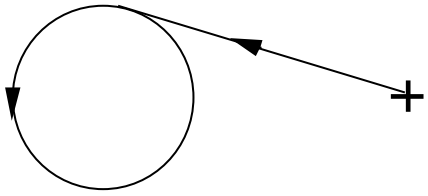


Figure 3 : *Résolution d'ambiguïtés par orientation*

EBP assure de cette façon la transformation des informations contextuelles capturées lors de l'utilisation interactive (la position de la souris lors de la désignation) en informations hors-contexte enregistrables dans un programme général.

Dans le système EBP, toutes les entités sont orientées par leur construction. Les segments sont orientés depuis leur origine jusqu'à leur extrémité, les cercles sont orientés en sens inverse des aiguilles d'une montre, etc. Durant l'enregistrement du programme, le système transforme l'information contextuelle « par proximité » en mécanisme d'orientation de la façon suivante :

- Le système calcule la bonne construction par proximité,
- Ensuite, le système calcule la bonne orientation.
- Si cette orientation est consistante avec la solution (comme dans la figure 3), le système enregistre l'action sans modification.
- Sinon, le système enregistre la séquence suivante : changer l'orientation du cercle, dessiner le segment, puis changer à nouveau l'orientation.

Ce mécanisme, qui reste caché à l'utilisateur final, est totalement déterministe.

2.2.2. EBP, système de Programmation par Démonstration

La programmation visuelle par démonstration est réalisée dans EBP via un mode d'enregistrement des appels de fonctions systèmes. Ceci signifie que, contrairement à certains systèmes paramétriques où les programmes sont directements basés sur leur valeur d'exemple (i.e. la fonction **line_2_points** fait directement référence au point exemple), les programmes au sein d'EBP (où ils sont nommés « instances ») sont séparés des exemples. Les relations entre valeurs d'exemple et variables du programme sont fournies au travers **du contexte dynamique** du programme. Ce mécanisme, courant dans les langages visuels, permet une référence indirecte entre la variable et sa valeur courante (au sein de l'exemple). Il renforce aussi l'indépendance entre le gestionnaire Pbd, qui manipule les variables, et le système de CAO où les valeurs d'exemple sont des références à la base de données CAO. Lorsque le programme est réexécuté (i.e. lors de modifications), les variables EBP ne sont pas modifiées, mais leurs contenus (adresses en base de donnée stockées au sein du contexte) sont mis à jour.

Après l'activation du **mode enregistrement**, le système EBP « espionne » l'utilisateur et construit une instance. Le passage en **mode utilisation** permet à l'utilisateur d'exploiter cette instance.

Les seules commandes additionnelles vis à vis d'un système de CAO traditionnel sont destinées à charger, nommer, sauver et exécuter des instances (commandes SAVE, NAME, LOAD, APPLY), ainsi qu'à définir, lire, écrire et saisir les paramètres (commandes DEFINE, READ, WRITE, ENTER). L'adjonction de structures de contrôle nécessite d'autres commandes, décrites dans la section suivante.

Une session typique d'EBP serait la suivante : aprés l'analyse de la pièce (quels sont les paramètres ? Où sont les dépendances ?... tâches courantes, que les dessinateurs sont habitués à réaliser à chaque fois qu'ils envisagent de réaliser un modèle CAO), l'utilisateur commence l'enregistrement PbD. Il définit les paramètres, puis dessine un exemple en utilisant les paramètres au lieu de valeurs directes. La commande DEFINE ouvre une fenêtre, où l'on fournit un nom (lequel est

affiché a chaque exécution du programme). La commande ENTER permet l'introduction de la valeur des paramètres pour l'exemple. Ces valeurs sont saisies via l'interface du système de CAO, et leur type définit le type des paramètres. Les commandes READ et WRITE permettent la lecture (resp. l'enregistrement) de valeurs à partir d'un fichier, lequel sera associé à l'instance. Cette fonctionnalité est utilisée pour enregistrer les valeurs autorisées de paramètres pour des familles de pièces. Dès que les paramètres sont définis, ils sont affichés au sein d'un menu où l'utilisateur peut les désigner, par exemple pour définir une expression à l'aide de la calculatrice grapho-numérique.

Lorsque l'exemple est complet, l'utilisateur peut sauver l'instance résultante, changer quelques paramètres et commander une nouvelle exécution. Enregistrer sur fichier des valeurs de paramètres est aisé, permettant des procédures de test rapides. Les instances enregistrées sont incluses dans un menu, et sont utilisables avec un minimum d'effort.

La commande LOAD sélectionne une instance et la commande APPLY l'exécute. Ces commandes peuvent être sélectionnées à la fois en mode utilisation et en mode enregistrement. Dans le premier cas, un modèle sera créé au sein de la base de données du système CAO. EBP correspond alors à un système de macro-par-exemple. Dans le second cas, la commande est enregistrée comme un appel de sous-programme dans l'instance en cours, et EBP assure le passage des paramètres. Dans les deux cas, après la sélection de la commande, EBP affiche chaque paramètre et attend une valeur. Cette valeur est définie à l'aide de l'interface du système CAO. Ceci signifie que, lors de l'exécution de l'instance en mode enregistrement, la définition de la valeur du paramètre est constituée d'une expression quelconque mettant en jeu des entités ou des paramètres du programme appelant.

2.2.3. Toutes les structures de contrôle

Le système EBP comporte des structures de contrôle complètes. Plus précisément, les sous-programmes, déjà évoqués ci-dessus, les alternatives et les répétitions sont totalement disponibles. La consistance du contexte des programmes est gérée entièrement par le système [GIR 90][GIR 95], et permet une utilisation correcte de ces structures.

Les alternatives et les répétitions supposent la définition d'expressions Booléennes, ce qui est fait par l'intermédiaire des calculatrices grapho-numérique et logique. Les deux branches des alternatives peuvent être définies soit dans un mode consistant avec les données (en exécutant l'instance avec de nouveaux paramètres) soit dans un mode inconsistant (en dessinant les deux solutions avec le même ensemble de paramètres)

Plusieurs schémas d'itération sont fournis : l'itération d'ensembles (« set iteration ») pour des objets sélectionnés à l'aide d'un rectangle élastique, ou les transformations géométriques multiples, qui sont communément disponibles dans les systèmes CAO, sont également proposées par le système EBP. Comme dans le

cas de la résolution des ambiguïtés, les informations dépendantes du contexte (les deux coins du rectangle élastique par exemple) sont traduites en informations indépendantes du contexte (l'ensemble des entités incluses dans le rectangle élastique). Mais, d'une façon plus générale, des structures telles que *Répéter_n_fois..., Tant que...*, ou *Répéter... Jusqu'à ...* sont aussi disponibles. Elles permettent la définition d'itérations de récurrence, et ce d'une manière totalement interactive.

Considérons par exemple la définition d'une boucle *Répéter... Jusqu'à ...* : supposons que l'on veuille dessiner la figure 4.

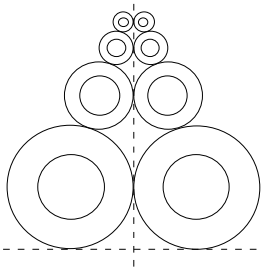


Figure 4 : *Un dessin (complexe)*

Elle est faite de cercles dont le rayon décroit de moitié à chaque itération, jusqu'à atteindre une valeur minimale. Le programme nécessaire peut être défini comme une répétition (pour obtenir une colonne de cercles) suivie d'une symétrie (pour obtenir l'autre colonne). Chaque tour (excepté le premier) peut être défini comme suit : *construire un cercle tangent au plus grand cercle du tour précédent et à l'axe vertical, dont le rayon est égal à la moitié du rayon du cercle du tour précédent ; puis, construire un second cercle dont le centre est le même que celui du dernier cercle, et dont le rayon est égal à la moitié du rayon de ce cercle.* Comme le montre la figure 4, le premier tour de boucle est quelque peu différent car le premier cercle se réfère à une autre entité (la ligne horizontale).

Trois points doivent être soulignés : (1) au cours d'un tour de boucle, les objets sont définis par rapport à ceux qui ont été créés au cours du dernier tour et/ou du tour courant ; (2) les actions réalisées durant le premier tour sont les mêmes que celles effectuées durant les suivants ; la seule différence réside dans les références aux objets du tour précédent qui doivent alors être recherchés dans le contexte englobant ; (3) enfin, une relation de récurrence doit être exprimée au cours du second tour de la répétition, en demandant à l'utilisateur quelle référence utiliser (cela peut être le même objet ou n'importe quel objet créé durant le premier tour).

Ces trois remarques constituent la base de l'interface utilisateur et des conventions de dialogue du système EBP. Deux commandes sont disponibles (REPEAT et UNTIL). L'utilisateur sélectionne la commande REPEAT et définit le

premier tour de boucle, en accédant pleinement à toutes les entités visibles sur son écran. Ensuite, il sélectionne la commande UNTIL.

Le système passe alors automatiquement en mode exécution, afin de réaliser le second tour. Chaque action enregistrée au cours du premier tour est alors exécutée, et le système met en évidence chaque objet manipulé. L'utilisateur doit alors préciser si, pour le second tour, l'objet désigné demeure le même (il s'agit alors d'une référence constante, comme celle de la ligne verticale dans l'exemple) ou bien s'il s'agit d'un autre objet, créé durant le tour précédent ou le tour courant (il s'agit alors d'une relation de récurrence). Tout autre choix est refusé par le système. La figure 5 illustre cette étape. Le même mécanisme est fourni pour toute expression, permettant la définition d'expressions nouvelles (dans notre exemple, l'expression pour le rayon du premier cercle).

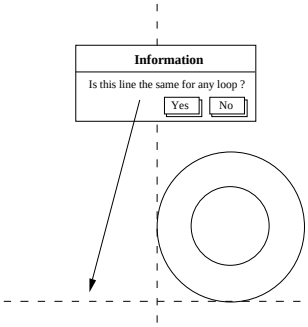


Figure 5 : *Question pendant le second tour*

Une fois chaque référence confirmée ou changée, le système demande à l'utilisateur de définir l'expression de contrôle. À l'aide de la calculatrice logique, ce dernier peut définir une expression utilisant des objets des trois contextes accessibles : l'englobant (interprêté comme une référence constante), le deuxième tour (interprêté comme le tour courant) ou le tour précédent (interprêté comme le tour n-1). Le système peut ensuite exécuter la totalité de l'itération.

L'annexe 2 montre le programme de la figure 4 en cours de création. On notera la fenêtre menu propre à l'outil de PbD (*Define statement*) ainsi que la fenêtre d'information correspondant à la figure 5. Enfin, les deux calculatrices sont visibles, quoique inactives ici.

2.2.4. EBP : un véritable environnement de programmation

Le système EBP est un environnement de programmation complet, qui fournit toutes les facilités de mise au point, dans un mode totalement basé sur exemple. Chaque interaction avec le programme se fait à travers l'exemple d'exécution. Les programmes générés sont montrés dans une fenêtre spécifique affichée uniquement

sur demande de l'utilisateur. Un menu spécial permet la réexécution des programmes.

· **Modification des programmes et UNDO/REDO intelligent**

Toutes les modifications de programme sont possibles, tant en mode mise au point qu'en création de programme. Des UNDOs successifs sont possibles, de manière illimitée, et l'ajout ou la suppression d'actions est toujours possible, et ce de manière consistante avec le reste du programme. Ainsi, quand la réexécution d'une partie de programme résultant d'une modification fait référence à des entités non-existantes du fait des modifications, le système EBP demande à l'utilisateur de remplacer les désignations d'objets.

· **Mise au point visuelle**

La représentation textuelle des programmes n'étant jamais supposée visible, la mise au point de programmes « virtuels » pourrait apparaître difficile. En fait, cette représentation n'est pas nécessaire, l'exemple constituant toujours une interface d'entrée/sortie pour le dialogue avec le programme.

L'annexe 3 montre certains éléments particuliers à l'environnement EBP : La fenêtre *Informations : Scripts* montre un programme FORTRAN en cours de modification, alors que la fenêtre *Macros Tear-off* montre les possibilités de mise au point du système. Les deux calculatrices sont cette fois actives.

Comme tout environnement de mise au point (« debugger »), le système EBP permet d'exécuter un programme pas à pas, ou jusqu'à sa fin. Il permet également de ré-exécuter le programme **jusqu'à ce qu'une entité soit créée**. L'utilisateur désigne graphiquement l'entité, et le programme s'exécute jusqu'à ce pas. Les modifications et l'exécution pas à pas sont alors possibles.

· **Génération de programmes portables**

Le système EBP a été conçu pour permettre la production de bibliothèques de composants standard, selon la norme ISO 13584-31. Ces programmes, qui peuvent demeurer cachés à l'utilisateur, ne peuvent être modifiés textuellement sous l'environnement EBP.

Après avoir été utilisé, dans le cadre du projet PLUS, par un utilisateur, fabricant de composants pour décrire les familles de composants de son catalogue, le système EBP est actuellement en phase de pré-industrialisation. Cet outil est destiné à être intégré à un éditeur HTML, développé au sein du projet PLUS par la société SPRING pour capturer les structures de données et les éléments documentaires constituant un catalogue de composants. L'intégration de ces deux outils permettra à un fabricant de composants de produire un catalogue informatisé intelligent (conforme à l'ISO 13584) de façon purement visuelle et graphique.

3. Pbd et Interface utilisateur

Un système de programmation par démonstration permet d'ajouter à une application CAO des fonctions spécialisées, par exemple pour créer des objets complexes au sein du modèle, comme c'est le cas pour les composants standard. Ces objets ne sont cependant que des ensembles d'entités de base, groupées ou non. L'utilisateur ne peut les manipuler directement et de manière naturelle. Le second travail que nous avons réalisé a consisté à étendre, par le biais de la PbD, les fonctionnalités de base d'un système CAO, de façon à permettre la manipulation des objets créés de la même manière que les objets natifs du système. Le but de cette étude, nommée GIPSE, est donc d'obtenir un véritable système de CAO disposant d'un système de « features » extensibles.

3.1. Le système GIPSE

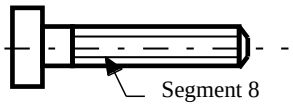
La Pbd permet, comme on l'a vu dans les sections précédentes, d'ajouter des fonctions de création au noyau d'un système de CAO. L'utilisation « intelligente » de ces objets nécessite cependant plus que leur création. Il faut aussi pouvoir, d'une part, récupérer ou calculer des informations, de manière sûre, sur ces objets et leurs composantes, et, d'autre part, les manipuler comme les objets natifs du système. La programmation par démonstration peut, moyennant certaines adaptations, être utilisée pour ajouter ces fonctionnalités au système.

Trois objectifs ont guidé la conception du système GIPSE, tous trois centrés sur l'ergonomie : (1) une sécurité d'utilisation accrue (on doit limiter les actions de l'utilisateur à celles réellement réalisables, par exemple, une instance ne doit pas admettre de paramètres de nature non valide) ; (2) l'homogénéité de l'interface (les objets créés en PbD disposent automatiquement des opérations existantes pour les objets natifs, comme la désignation, la lecture des paramètres ou les transformations géométriques) ; (3) une invocation facile des instances (une icône permet d'instancier une feature particulière au même titre qu'un cercle ou une spline). Ces trois aspects sont développés dans les sections suivantes.

3.2. La sécurité par le typage

Prenons comme exemple la définition de la fonction d'interrogation de la hauteur du filetage d'une vis, elle même créée par un programme Pbd, `Création_Vis`. Cette vis correspond dans la CAO à un ensemble (ou groupe) d'entités de base (segments, cercles, rectangles...). La fonction désirée, `Hauteur_Filetage`, prend donc en paramètre d'entrée un ensemble. Le contexte de la fonction contiendra alors les composants de cet ensemble passé en paramètre, ici ceux de la vis. Cette fonction peut être générée en programmation par démonstration (figure 6). Le programme est alors fonctionnel, mais peut être employé sur n'importe quel ensemble d'entités, y compris sur ceux qui ne correspondent pas à

une vis. Dans le cas d'un ensemble où il n'existe pas d'entité S8, ou si la fonction taille ne peut manipuler cette entité, le programme appelé échoue, avec des conséquences variables selon sa fonction. Ce problème montre bien que l'instance doit être représentée comme une entité particulière (une vis) et non comme un ensemble d'entités élémentaires (ensemble de points, segments, etc.).



Début
Paramètre (*Ensemble*) -> S1...S14
retourner Taille (S8) -> N1
Fin programme

Figure 6 : *Le programme Hauteur Filetage*

La quasi-totalité des systèmes CAO possèdent la capacité de regrouper des entités élémentaires au sein d'un ensemble, et de nommer ces ensembles. La première étape consiste alors à ajouter au sein de `Création_Vis` une action donnant un nom particulier à l'ensemble créé, et de rajouter automatiquement à `Hauteur_Filetage` un test vis à vis du nom de l'ensemble passé en paramètre.

Mais cette première étape est cependant insuffisante car, si elle permet à l'opérateur de translater une vis (i.e., un ensemble), de désigner une vis (i.e., un ensemble), etc., elle lui permet aussi de modifier les entités qui constituent la vis (sans changer son nom) ou de changer le nom sans changer le contenu.

La deuxième étape consiste alors à définir, au sein du système CAO, un type particulier d'ensembles appelé « FEATURE ». Ce type partage avec les ensembles ordinaires toutes les opérations globales (désignation, transformations géométriques, ...). Il est en revanche protégé contre la modification interactive de son nom ou son contenu. Toute demande de modification entraîne l'affichage des paramètres de la feature aux fins de modification éventuelle. Le programme est alors réexécuté comme ce serait le cas pour un cercle ou une spline. Le type « feature » peut ensuite être dynamiquement sous-typé pour les différentes « features » créées en PbD. Lorsqu'un nouveau sous-type est ainsi défini, il est possible de lui associer de nouvelles opérations, telles que l'interrogation de sa longueur filetée qui retourne une valeur réelle pouvant elle-même être impliquée dans une expression grapho-numérique.

On peut ainsi définir la production de `Création_Vis` comme étant un objet de type 'Vis'. La fonction `Hauteur_Filetage` prend alors comme paramètre un objet 'Vis' à la place d'un paramètre 'Ensemble', empêchant par lˆ-même son utilisation abusive sur des objets auquel la fonction ne s'applique pas.

3.3. Une manipulation automatique

La définition de nouveaux types d'objets s'accompagne de la création automatique d'un certain nombre de fonctionnalités génériques associées à cette catégorie d'objets, telles que la désignation ou les transformations géométriques. Par exemple la translation d'une vis correspond à la translation de tous ses composants par un même vecteur. La création de fonctionnalités spécifiques à une catégorie est réalisée d'une part par la fonction de modification qui permet de la modifier en changeant la valeur de ses paramètres, et d'autre part par les éventuelles fonctions additionnelles définies par PbD.

Enfin nous avons également implémenté la fonction d'exploration dynamique usuelle dans les systèmes de programmation visuelle. À la fin de la construction d'un exemple, l'opérateur peut associer à chaque paramètre une entité caractéristique de l'objet (par exemple, dans la figure 6, la longueur de filetage au segment 8). Lorsque la commande 'MODIFY' est sélectionnée, et la vis désignée, les entités caractéristiques sont mises en évidence, et peuvent être modifiées par l'utilisateur (« Drag & Drop »). Le programme est réexécuté en permanence de façon à montrer la « feature » en cours de modification.

3.4. Spécialisation de l'interface

Les ajouts de fonctionnalités que permet la Pbd, aussi complets soient-ils, ne sont cependant qu'un ensemble de programmes qu'il reste à rendre facile d'accès. Lorsqu'une instance est définie, le système génère automatiquement une icône et celle-ci est ajoutée, par défaut, au menu 'FEATURE'. GIPSE permet alors, et sans sortir de la session en cours, de modifier la struture des menus, plaçant par exemple la nouvelle icône dans un sous-menu du menu 'FEATURE'. Cette méthode permet de spécialiser le logiciel, en étendant le modèle par des objets (ainsi que leurs méthodes) spécifiques au domaine souhaité. Une application vers les systèmes « orientés métiers » semble ici naturelle. De même, lorsqu'une « feature » est désignée, le système affiche automatiquement le menu des fonctions (commandes) associées à ce type de feature telles qu'elles ont été définies par Pbd.

3.5. GIPSE : un environnement évolué de personnalisation

L'ensemble des fonctionnalités présentées précédemment : création de nouveaux types d'objets et de fonctions spécifiques à ces objets, intégration dans l'interface de ces nouveaux types et structuration des menus, rendent possible la modification d'un logiciel de CAO pour le spécialiser dans un domaine particulier, ou pour une tâche particulière.

4. Conclusion

La programmation par démonstration (« Programming by Demonstration » ou PbD) est un concept récent qui résulte des progrès réalisés dans le domaine des interfaces homme-machine. Nécessitant des domaines d'application fortement graphiques, la PbD a trouvé dans la CAO un champ d'expérimentation naturel. Dans cet article, nous avons décrit deux applications possibles de la PbD en CAO, qui dépassent le cadre des systèmes variationnels et paramétriques actuels.

Le système EBP permet à un projeteur de créer, avec un minimum d'effort d'abstraction, des familles de composants standard, dans un format normalisé permettant l'échange entre systèmes CAO différents. Il constitue un outil majeur pour la création et la gestion des bibliothèques de composants, tant pour les fournisseurs que pour les utilisateurs finaux qui définissent en interne leurs propres composants.

Le projet GIPSE, quant à lui, vise à permettre l'extension et la spécialisation « métier » d'un système CAO par un utilisateur non-programmeur. Il ouvre la voie à la programmation aisée de systèmes à « features » souples et extensibles.

Les deux cas présentés ici démontrent tout l'intérêt de l'approche PbD dans le cadre de la CAO, et ouvrent de nouvelles voies pour la programmation par l'utilisateur.

5. Références

[BOU 95] Bouma W., et al., « Geometric Constraint Solver », *Computer Aided Design*, 27, 1995, p. 487-501.

[CUG 88] Cugini U., et *al*., « A Procedural System for Definition and Storage of Technical drawings in Parametric Form », *Proceeding of the EUROGRAPHICS'88* (1988), p. 183-196.

[CYP 93] Cypher A., *Watch What I Do: Programming by Demonstration*, The MIT Press, 1993.

[CYP 95] Cypher A., et *al*., « KidSim: End User Programming of Simulations », *Proceeding of the CHI'95* (1995), p. 27-36.

[GIR 90] Girard P., et al., « End User Programming Environments : Interactive Programming-On-Example in CAD Parametric Design », *Proceeding of the EUROGRAPHICS'90* (1990), p. 261-274.

[GIR 95] Girard P., et al., « Structures de contrôle générales en Programmation par Démonstration », *Actes des Septièmes Journées sur l'Ingénierie de l'Interaction Homme-Machine* (1995), p. 61-68.

[GLI 90] Glinert E., *Visual Programming Environments*, IEEE Computer, 1990.

[HAL 84] Halbert D., *Programming by Example*, PhD, Berkeley, 1984.

[HIR 90] Hirakawa M., et al., « An iconic programming system: HI-VISUAL », *IEEE Transactions on Software Engineering*, *16*, 1990, p. 1178-1184.

[JAC 93] Jackiw R.N., et al., « The Geometer's Sketchpad: Programming by Geometry », in *Watch What I Do: Programming by Demonstration*, Cypher A., The MIT Press, 1993, p. 293-308.

[MYE 88] Myers B.A., *Creating User Interface by Demonstration*, Academic Press, 1988.

[MYE 90a] Myers B.A., « Taxonomies of Visual Programming and Program Visualization », *Journal of Visual Languages and Computing*, *1*, 1990, p. 97-123.

[MYE 90b] Myers B.A., et al., « GARNET: Comprehensive Support for Graphical, Highly Interactive User Interfaces », *IEEE Computer*, *23*, 1990, p. 71-85.

[NAR 93] Nardi B.A., *A Small Matter of Programming, Perspectives on End User Computing*, The MIT Press, 1993.

[NEW 83] Newell R., et al., « Parametric Design in MEDUSA System », *Proceeding of the CAPE'83* (1983), p. 124-154.

[OLS 88] Olsen D.R., et al., « Macros by Example in a Graphical UIMS », *IEEE Computer Graphics and Applications*, 12, 1998, p. 68-78.

[OWE 91] Owen J., « Algebraic Solution for Geometry from Dimensional Constraints », *Proceeding of the ACM Symp. Found. Solid Modeling* (1991), p. 397-407.

[PIE 94a] Pierra G., et al., « Logical Model for Parts Libraries », ISO-CD 13584-20, 1994.

[PIE 94b] Pierra G., et al., « Design and Exchange of Parametric Models for Parts Library », *Proceeding of the 27th International Symposium on Advanced Transportation Applications, ISATA'94* (1994), p. 397-404.

[POT 95] Potier J.-C., « Conception sur exemple, mise au point et génération de programmes portables de géométrie paramétrée dans le système EBP », thèse de doctorat, Poitiers, 1995.

[ROL 90] Roller D., « Dimension-Driven Geometry in CAD: a Survey », in *Theory and Practice of Geometric Modeling*, Springer-Verlag, 1990, p. 509-523.

[SAS 94] Sassin M., « Creating User Intended Programs with Programming by Demonstration », *Proceeding of the IEEE Symp. on Visual Languages* (1994), p. 153-160.

[SHA 95] Shah J.J., et al., *Parametric and Feature-based CAD/CAM: Concepts, Techniques and Applications*, John Wiley & Sons, 1995.

[SMI 75] Smith D.C., *A Computer Program to Model and Stimulate Creative Thought*, Birkhauser, 1975.

[SOL 94] Solano L., et al., « Constructive Constraint-Based Model for Parametric CAD Systems », *Computer Aided Design*, *26*, 1994, p. 614-621.