Customizing by Demonstration
Generic Systems to Specific Tasks

P. Girard, G. Pierra and J.C. Potier

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

Abstract. Most of the computer applications are generic in nature. As a result, end-user must map their specific activities into the capabilities of the generic applications and their specific classes of objects onto the low-levels entities supported by their systems. Programming by Demonstration (PbD) is a technics that enables to abstract from a particular example of a process, the general program that describes a family of similar processes. We describe a 2D graphic editor which incorporates PbD capabilities. This system, named EBP, is intended to bring to draughtsmen a complete environment for programming without any textual interaction with programs. It follows technical draughtsmen habits, using a purely procedural apprroach without any inference, and it offers several original programming goodies: complete support of control structures, visual debugging, fully integrated PbD user interface.

1. INTRODUCTION

The motivation behind Programming by Demonstration (PbD) is simple and challenging : “if a user knows how to perform a task on the computer, that should be sufficient to create a program to perform the task”[Cypher 1993]. When a programming by demonstration manager is available within a user interface, the user may instruct the computer to “watch what he/she does”. Then the system is able to abstract from the particular example a general program. This abstraction mechanism enables to replace the values involved in the example by abstract variables. Then, this program may be triggered as a new command and provided with parameters values: the end user has customized its generic system toward its specific tasks without any explicit programming.

Many experimental systems have proved both the usability and the interest of this approach in various application area (see e.g., [Cypher 1993]). Nevertheless, no PbD system, to our knowledge, has reached the same expressive power as conventional programming in its application area.

The goal of this paper is to report our experience on the development of a programming by demonstration manager in a Computer Aided Design (CAD) environment. This system, called EBP for Example Based Programming, is intended to enable end users of a 2D graphic editor to generate every program that describes a family of cognate shapes through the interactive graphic design of one example of this collection.

2. PBD and CAD

In her book, *A small matter of programming, perspectives on end-user computing* [Nardi 1993], B. Nardi has identified CAD as a natural candidate for end-user programming, because these systems "allow end users to create useful applications with no more than a few hours of instruction". We will see in this section how this can be extended to complete PbD features.

2.1. From Visual programming to Programming by Demonstration

In the 1980s, several projects laid foundations for visual programming (reprinted in [Glinert 1990]). The main idea of this new approach is to replace words by pictures. While textual

programming offers no more help for understanding programs than meanings of words, visual programming uses pictures as a much more intuitive representation of this meaning. Visual programming has introduced, in different fields, pictorial representations for both variables and program functions. Nevertheless, visual programming mainly addresses static representations of programs. The remaining question is: What will actually occur when the program runs?

Using examples to design programs makes up the second step towards end-user programming environments. Instead of *selecting functions* and *choosing variables* to which each function shall apply, users (programmers) *do* functions on *values* which stand for program variables. This programming paradigm was born in Pygmalion [Smith 1975]. It has been largely analysed in Halbert's PhD Dissertation [Halbert 1984], and has been formalised in Myers' works [Myers 1990] under the name of Example-Based Programming. Cypher's compilation [Cypher 1993] introduced the term of Programming by Demonstration (PbD), which seems to be largely accepted today. The main idea is to avoid the abstraction level of variables by enabling the user to deal only with specific values of these variables. During example design, PbD systems analyse inputs, abstract variables from values, and build the program that generalize the example. Notice that the PbD paradigm is not about machine learning or artificial intelligence: "Rather , the focus is on ways to create the appropriate human-computer interaction so that end users can gain more control of their personal computers".

PbD opens the door to new generations of programming environments. In the fields where some visual appearance may be assigned to variable values, direct manipulation of these values allows implicit programs design. Despite an actual success, most systems seems to be mainly at a prototype stage. In the CAD area, PbD, under the name of "Parametrics", really found some commercial market.

2.2. CAD, a suitable area for PbD

Designing new products often consists in assembling pre-existing components intended to be used in different products. These components, named "standard parts" are gathered into families described by a part family model. According to [Shah and Mδntylδ 1995] "a part family model represents a collection of parts exhibiting some variation in dimensions, tolerances, and overall shape that nevertheless are considered similar from the viewpoint of a certain application". Because of the often large number of members of these collections, some unique part family model shall describe the whole collection of corresponding shapes.

In the first generation of CAD systems, part family models were described as parametric programs. In these conventional CAD systems, such programs were textually described, often in FORTRAN or in the C language. When triggered, they created geometric entities by means of API. These systems were used on end-user sites where draughtsmen were trained on CAD modelling. So, a strong requirement existed in the CAD area for end-user-oriented programming environments.

The second reason why PbD has been successful in the CAD area is the kind of dialogue language CAD systems support. CAD models, or technical drawings, are very different from pictures or artistic drawings: they shall conform to strong rules depending on the application area (mechanical design, architectural area, and so on). When designing such drawings, draughtsmen perfectly know the relationships that must exist between the entities of their model, and they want to have the capability of expressing these constraints in their design process. Since the early beginning of CAD, every CAD system provides commands which enable the expression of such constraints. Therefore, CAD system interfaces enable users to

explicitly specify every constraint that shall hold between objects, and CAD users are accustomed to specifying such constraints. Just recording these constraints builds the basis of sequential imperative program recording.

2.3. Parametrics

While every modern CAD system supports this kind of constraint-based definition of entities, constraints recording appeared much more recently. Beside the MEDUSA system [Newell, et al. 1983], that provided for constraint-recording capabilities in 1983, the generalization of this feature appeared in the late 80's. At this time, a new generation of systems appeared on the market; they were able to record these constraints, to change the numerical values involved in these constraints, and to compute the new model resulting from these values. These systems, often called *Dimension-driven* systems [Roller 1990] are so attractive that, at the present time, every competitive CAD system must provide some kind of parametric capabilities. This large diffusion proves the practical interest of the approach. It also proves that draughtsmen, end-users, are able to generate parametrized shapes, i.e. real visual programs, without programming knowledge.

2.4. Requirements for EBP

The requirements for EBP are the fllowing. (1) The generation process should be deterministic and fully controlled by the designer: one and only one shape should be generated for every allowed set of values of the input parameters, and this shape should precisely correspond to the part. (2) Every kind of shape family which might be described using some conventional way of programming should be able to be designed using this system. (3) The system should be able to generate an external representation of its internal data structure in the format of a FORTRAN program conforming to the standard API.

Compared to this threefold requirements the current status of the parametric technology is as follows. (1) Existing parametric CAD systems are based on implicit heuristics for ambiguity removal. Recording the same example on different CAD systems might give different results when the dimensions are changed [Bouma, et al. 1995]. (2) Only very simple case of repetitive structure, called linear or circular « patterns », are supported. No system, to our knowledge, support general recurrence-based loop structure. (3) One major problem in the CAD application area is the lack of capability to exchange parametric data models between heterogeneous CAD systems. Every parametric data structure is recorded in some proprietary format and is not or hardly mappable on any other parametric data structure.

3. From example to program: the EBP dialogue

In this section, we describe the user dialogue with the PbD manager, and the abstraction mechanisms used to build the program. Similar features should take place in most PbD managers.

3.1. Switching from example-dependent information to context-free information

Most of the geometric constructs are ambiguous: in particular, every constraint that involves a circle or a distance corresponds, in general, to two different solutions.

For example, building a line which starts on a given point, and which ends tangential to a circle, leads to two possible solutions, as pointed out in Figure 1.

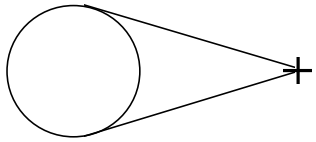

Figure 1: The two possible solutions for a line tangential to a circle

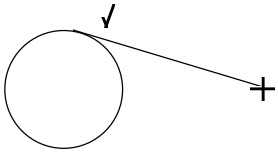

Figure 2: pointing solving

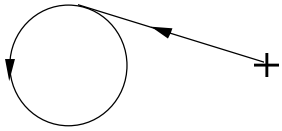

Figure 3 : Consistent orientation

This problem is perfectly solved in interactive geometric design. Most graphic editors or CAD systems use the mouse-click position of each input object to discriminate the possible constructs. They assume that the designer approximately knows the expected solution. For example, in the case of figure 1, the pointing click position results in the choice of the upper line solution (figure 2).

Unfortunately, this solution is not available with parametric programs. So, EBP ensures the translation from the context-sensitive information captured at the user interface level (the position of the mouse click) into a context-free information recorded in the program. In EBP, every entity is oriented according to the way it was constructed. Lines are oriented from their origin to their extremity, circles are oriented counter clockwise and so on. During program recording, the system translates the proximity disambiguity mechanism into the orientation mechanism as follows:

- With the proximity mechanism, the system calculates the right construction,
- Then, the system checks for the circle orientation.
- If this orientation is consistent whit the solution (as in figure 3), the system records the drawing without modification.
- If not, the system records the following sequence: change circle orientation, draw the line, and change the circle orientation again.

This mechanism, which remains unknown from users, is perfectly determinist. The same kind of translation should prove usefull whenever some example-values-dependant information is to be recorded in the PbD defined program.

3.2. Abstracting from values to variables

In PbD systems the user interacts with values (e.g., 2.14, *this* particular point on the screen), the recorded program reference variables (e.g., reel_1, point_3). A mechanism is required for abstracting values into variables (in recording mode) and for interpreting variables by their current values (in running mode). Unlike some parametrics systems where "programs" are directly related to example values (e.g., the function line_2_points directly refers to the example point), in EBP, programs (which are named "instances") are separated from examples. Relationships between example values and program variables are given through the dynamic context of the program that consists of a set of couples (value, variable).

3.3. Full control structure support

EBP includes full control structure support. More precisely, conditionals, iterations and subroutines are fully supported, even with recurrence. The program context consistency is managed by the system [Girard and Pierra 1995], and allows a consistent use of these structures.

Conditionals and iterations require Boolean expression definition. This is done using the numerical and logical calculators. The two alternate branches of conditionals may be defined either in a consistant way (running again the instance with alternate parameter values) or in some unconsistant way (drawing the two solutions with the same parameter values).

4. EBP and PBD systems

In this section, we use the summary sheet which has been proposed in [Cypher, et al. 1993] for PbD system characterization. We explore Uses and users (4.1), User interaction (4.2), Inference (4.3.), and Program constructs coverage (4.4)..

4.1 Uses and Users

EBP addresses mechanical 2D CAD. It is designed for producing standard programs to be run over different CAD systems, but explicit programming or textual modification of programs is never required. Unlike many systems, EBP produced standard API codes and provides computationaly completeness of generated programs.

EBP addresses trained end-users, without programming knowledge. This is completely different for example from KidSim [Cypher and Smith 1995] where intended users are kids, or from the Geometer's Sketchpad [Jackiw and Finzer 1993] where users are geometer's students. EBP users are expert CAD users, who perfectly know how to build CAD models. EBP is built on this background to provide them implicit programming capabilities.

4.2. User interaction

EBP is a macro-like system. It does not interleave program creation with program execution. Data description is always explicit. However, users are not required to give them *a posteriori*, as in Smallstar: constraint-based constructs available on the CAD system allow *a priori* definitions. EBP has special commands for program constructs definition, but users are never required to modify their textual program. Loops, conditionals and subroutines definition and usage are built with only interactive programming-with-example techniques.

Debugging facilities in EBP may be related to Zstep 94 [Lieberman and Fry 1995]. Lots of fonctionalities are similar: EBP's Visit menu looks like the ZStep 94's "video recorder", and ZStep 94's graphical step is very close from EBP's Visual Debugging. However, main objectives differ in the fact that EBP does not require users to read the FORTRAN program, while ZStep 94 is intended to help the programmer understand the correspondence between static program code and dynamic program execution.

EBP provides for textual visualization of programs. Nevertheless, it does not require nor allow any interaction over it. While Chimera [Kurlander 1993] and Pursuit [Modugno and Myers 1994] allow graphical visualization of programs, and direct interaction with it, modifying and debugging is achieved in EBP on the example itself. When modifications are made by the user, EBP maintains the consistency of the program by asking the user for no longer valid entity reference. This is to be compared with Pygmalion [Smith 1975] which requires that the remainder of a program be re-demonstrated, and Smallstar [Halbert 1984] or Chimera [Kurlander 1993] which do not check for validity.

4.3. Inference and Domain Knowledge

As already underlined, and unlike most PbD systems, EBP does not use any inference. It provides explicit commands for control structure definition, and implicit built-in program constructs. It does not require any supplementary information from the normal interactive use

of the system. It only uses predefined orientation rules to translate proximity disambiguity mechanism.

4.4. Program constructs coverage

Program constructs, and more precisely subroutines, conditionals and iterations, is the most important challenge for PbD systems. In fact, they are largely restricted in the existing systems. In 1990, this fact was considered as a major drawback for PbD systems [Myers 1990]. Cypher's 'Watch what I do' has a good summary on these features in existing PbD systems, that we will summarise below. In EBP, the whole set of geometric entities which is generated by a subroutine may be referenced as a unique set in the embedding context. It may therefore be used, for example in symmetry actions.

Conditionals are frequently supported, but they are often restricted to exception handling (object existence test, geometrical constraint satisfaction, ...). Last, iterations are mainly *set iterations* [Halbert 1984] which consist of repeating actions on object lists.

Finally, the systems which may be considered to be the most complete drop out the major characteristic of PbD, that is directly interacting with the example: they ask users for textual modifications when introducing control structures (Tinker [Lieberman 1993], Smallstar [Halbert 1984], Geonode [Van Emmerik 1990]).

5. CONCLUSION

In this paper, we have shown how PbD managers might enable to customize generic system toward user-specific tasks or classes of objects. We have presented the EBP system, which constitutes a complete PbD environment for parametric geometry. This system:

- does not use any inference mechanism to ensure full user control onto the (implicit) program;
- supports every control structure of imperative programming without any direct interaction with the textual program;
- is able to generate conventional programs that may be used on different systems or compiled to reduce their running time.

Acknowledgements

The research described in this paper was funded partially by the European Commission under Project ESPRIT III #8984 (PLUS), and partially be the French Ministry of Industry under grant 93.4.930080. An enhanced version of this paper, with lots of EBP's snapshots may be found at http://www.lisi.univ-poitiers.fr/members/girard.html

References

[Bouma, et al. 1995] W. Bouma, et al., "Geometric Constraint Solver", *Computer Aided Design* , 27 (6), 1995, pp. 487-501

[Cypher 1993] A. Cypher, "Watch What I Do: Programming by Demonstration", Cambridge, Massachusetts, 1993, 604p.

[Cypher, et al. 1993] A. Cypher, D. S. Kosbie and D. Maulsby, "Characterizing PBD Systems", in Watch What I Do: Programming by Demonstration, Cambridge, Massachusetts, The MIT Press, 1993, pp. 467-484

[Cypher and Smith 1995] A. Cypher and D. C. Smith, "KidSim: End User Programming of Simulations", in *CHI'95*, 1995, Denver, Colorado, ACM/SIGCHI, pp. 27-36

[Girard and Pierra 1995] P. Girard and G. Pierra, "Structures de contrτle gınurales en Programmation par Dımonstration", in *Septiθmes Journıes sur l'Ingınierie de l'Interaction Homme-Machine*, 1995, Toulouse, Cıpaduθs, Toulouse, pp. 61-68

[Glinert 1990] E. Glinert, "Visual Programming Environments", ACM Press, 1990, 600 p.

[Halbert 1984] D. Halbert, "Programming by Example", Programming by Example, University of California, Berkeley, 121 p.

[Jackiw and Finzer 1993] R. N. Jackiw and W. F. Finzer, "The Geometer's Sketchpad: Programming by Geometry", in Watch What I Do: Programming by Demonstration, Cambridge, Massachusetts, The MIT Press, 1993, pp. 293-308

[Kurlander 1993] D. Kurlander, "Chimera: Example-Based Graphical Editing", in Watch What I Do: Programming by Demonstration, Cambridge, Massachusetts, The MIT Press, 1993, pp. 271-292

[Lieberman 1993] H. Lieberman, "Tinker: A Programming by Demonstration System for Beginning Programmers", in Watch What I Do: Programming by Demonstration, Cambridge, Massachusetts, The MIT Press, 1993, pp. 49-66

[Lieberman and Fry 1995] H. Lieberman and C. Fry, "Bridging the Gulf Between Code and Behavior in Programming", in *CHI'95*, 1995, Denver, ACM/SIGCHI, pp. 480-486

[Modugno and Myers 1994] F. Modugno and B. A. Myers, "A State-Based Visual Language for a Demonstrational Visual Shell", in *IEEE Symp. on Visual Languages*, 1994, Saint-Louis, Missouri, pp. 304-311

[Myers 1990] B. A. Myers, "Taxonomies of Visual Programming and Program Visualization", *Journal of Visual Languages and Computing* , 1 (1), 1990, pp. 97-123

[Nardi 1993] B. A. Nardi, *A Small Matter of Programming, Perspectives on End User Computing*, Cambridge, Massachusetts, The MIT Press, 1993, 157 p.

[Newell, et al. 1983] R. Newell, G. Parden and P. Parden, "Parametric Design in MEDUSA System", in *CAPE'83*, 1983, Amsterdam

[Roller 1990] D. Roller, "Dimension-Driven Geometry in CAD: a Survey", in Theory and Practice of Geometric Modeling, Springer-Verlag, 1990, pp. 509-523

[Shah and Mδntylδ 1995] J. J. Shah and M. Mδntylδ, *Parametric and Feature-based CAD/CAM*, New York, John Wiley & Sons, 1995, 619 p.

[Smith 1975] D. C. Smith, *A Computer Program to Model and Stimulate Creative Thought*, Basel, Birkhauser, 1975, 187p.

[Van Emmerik 1990] M. Van Emmerik, "Interactive Design of Parametrized 3D models by direct manipulation", PhD Thesis, Delft, 1990, 141 p.