The EBP system: bringing programming to end-users

P. Girard, G. Pierra, J.C. 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,pierra,potier} @ensma.fr

ABSTRACT

Programming by Demonstration has proved to be an interesting research area. Nevertheless, despite many experimental systems, it did not find any field area where it should be used extensively. In this paper, we describe a Computer Aided mechanical Design system which incorporates Programming by Demonstration capabilities, named EBP, for Example-Based Programming in Parametrics. EBP is intended to bring to mechanical draughstmen a complete environment for programming by demonstration without any textual interaction. It follows technical draughstmen habits, for example by no use of inference, and offers many programming goodies (intelligent undo/redo, visual debugging, fully integrated PbD user interface, and so on.

KEYWORDS

Programming by Demonstration (PbD), Example-Based Programming, Computer Aided Design (CAD), adaptive interaction, user modelling.

Introduction

Over the last few years, lots of advances have been achieved to reduce the programming skills and the abstraction level which are required for computer using and programming. Visual programming [9] permits users to graphically select both the functions and the variables which constitute programs. Programming by Demonstration, or PbD [4], allows direct interaction with example values that represent the program variables instead of their abstract names, or iconic presentations. Many experimental systems have proved both the usability and the interest of the latter approach. 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 present a system, the EBP system (Example Based Programming in Parametrics), which is intended to enable Computer Aided Design (CAD) system users to generate every program that describes the geometric shapes of a collection of parts through the interactive graphic design of one example of this collection ("Variant Programming" [35]). Compared with CAD systems, the EBP system is able to describe every collection of shapes that might be described by conventional programs, even if they contain repetitive or alternative shape aspects. Compared with the existing PbD systems, (1) users may specify on the example every kind of conditional or recurrence-based loop structure without any textual manipulation, (2) the system does not use any inference mechanism but explicit dialogue conventions which are fully integrated within the usual dialogue of the CAD system, and (3) the system generates neutral forms of programs that may be run, later on, on every other CAD system that supports a standard Application Programming Interface (API).

The structure of this paper is as follows: section 1 briefly outlines the application of the PbD approach in modern CAD systems. Section 2 describes the requirements stated for EBP. Section 3 presents the EBP system itself. Last, section 4 relates PbD facilities of EBP to existing PbD systems.

Extended version of Girard P., Pierra G., & Potier J.-C. Customizing by Demonstration Generic Systems to Specific Tasks. *UI4ALL, 3rd ERCIM Workshop on User Interfaces for All, Ortrott, France,* 3-4 November 1997. p. 189-196.

1. PBD AND CAD

In her book, *A small matter of programming, perspectives on end-user computing* [25], 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. After a small description of PbD, we describe the application field of PbD in CAD, named *Parametrics programming*. Last, we describe two approaches; called *variational* and *parametric* approaches, and we relate them to the PbD terminology.

1.1. From Visual programming to Programming by Demonstration

In the 1980s, several projects laid foundations for visual programming (reprinted in [9]). 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. Commercial products like LabVIEW®[1], or experimental ones, like HI-VISUAL [11] proved the approach's validity. 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 [38]. It has been largely analysed in Halbert's PhD Dissertation [10], and has been formalised in Myers' works [23] under the name of Example-Based Programming. Then, various workshops took place, and have led to state of the art reports on end-user programming [22, 4]. Cypher's compilation [4] 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, and build the program able to generate the example, and some variants of this example. This analysis may be done by immediate action recording (**programming-with-example** in Myers terminology) or by inferencing mechanisms over the example value (**programming-by-example**).

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. So, Halbert's SmallStar [10] for iconic desktop programming, Myers' PERIDOT [20] and GARNET [24], Olsen's MACROS BY EXAMPLE [28] for UIMS programming, Cypher's KidSim [6] for simulation, Wilde's WYSIWYC Spreadsheet [42] or Geometer's Sketchpad [12] and Sassin's ProDeGE+ [36] in drawing systems, have proved in different fields the validity of this approach. Despite this 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.

1.2. 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 [37] "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". The context of some part family corresponds either to some product standard

[1] National Instruments, USA

(e.g., the family of ISO 1014 hexagonal screws), either to some supplier's environment family, or to some family of firm-specific components, which are described by end-users for internal use. Because of the often large number of members of these collections, some unique part family 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 create geometric entities by means of API. A lot of these systems have been developped on end-user sites where draughtsmen were trained on CAD modelling. So, a strong requirement exists in the CAD area for end-user-oriented programming paradigm.

The second reason why the PbD approach may be easily implemented 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. Geometric constraints are so specified by means of geometric operators (e.g., *middle_of, starting_point, projection_of ... onto...*). Numerical constraints are specified by *display calculators* which provide both algebraic operators (e.g., +, -, *, /) and geometric functions. These functions (e.g. *distance_of, angle_between, radius_of,* ...) take references to model entities as parameters, and return numerical values which, in turn, may be involved in numerical expressions. 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.

1.3. 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 [26], 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 [35] have a twofold data structure and a twofold behaviour. On the one hand, users may build (or may change, or may compute) the displayed model. On the other hand, users may ask for visualization of the constraints and the numeric values which are involved in the example design. This information, which stands for the program in the PbD terminology, is displayed in some conventional symbolic way, for instance through dimensionning. Then, users select the values they want to change, enter new values, and the system automatically computes the new model which corresponds to the same constructive process, or to the new solution of the same set of constraints.

In fact, dimension-driven systems proceed from two different approaches (declarative approach and imperative approach) [33]. **Variational systems** hide the **declarative program**. Users build the example, specify the constraints, either explicitly or implicily. These constraints are recorded as a set of equations, and some solver derives the solution. Variational geometry lies in geometric problem solving. The solution may be unknown from the user. Once constraints are stated, with some approximate geometric description, the solver tries to compute a solution. This approach corresponds to the popular sketchers which are available on most recent CAD systems: users draw some free-hand sketch of a

2D model, and the solver computes the exact model after constraints are stated. The user interface is very friendly. Lots of methods have been used to solve these constraints. Most efficient methods are based on graph reduction [29, 1]. Despite an actual progress, this approach suffers from three intrinsic weaknesses. (1) The set of equations has generally many solutions (exponential number), and only system specific heuristics have been defined so far to guess the "user intent". Very simple examples have been published [1] where the computed solutions were obviously not intended to be the right ones. (2) Because of the heuristic-driven nature of solving processes, different systems should provide different solution for the same model. (3) Pure variational systems are unable to capture the purely procedural constructs, such as Boolean regularized operations or sweeping. Therefore, every so-called "variational system" is in fact an hybrid system which is variational in 2D and mainly procedural in 3D.

Procedural systems, often called **Parametrics**, address a very different problem: "given a class of shapes whose design process is well known and may be supported by the interface of some CAD systems, we want every instance, characterized by its parameter values to be generated automatically in a deterministic way". **Parametric systems** hide an **imperative program**. This program is often captured using the example design process: the CAD system “spies on” draughtsmen while they are designing their example. As long as the example model grows, the constructive logic of draughtsmen is captured. Afterwards, the CAD system is able to replay the constructive logic, possibly with new input values. The internal representation of programs may be textual, but it is more generally based on data structures [33, 39] such as directed acyclic graphs [2]. The Pro-Engineer® system[2] is the most popular example of this approach.

In the CAD area, the dimension-driven approach is so attractive that, at the present time, every competitive CAD system must provide such 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. That is not to say without any modification of their working process. Effectively, drawing **shapes** is slightly different from drawing **families of shapes**. Nevertheless, this activity does not stand at the abstract level of conventional programming activity. Dimension-driven systems, or parametrics for short, largely facilitate the design of part family models. For collection or single shapes, just designing one shape provides for generating every family's shape.

2. REQUIREMENTS FOR EBP

Most choices which have been made for the EBP system are governed by the goals of the PLUS project and the habits of the users, i.e. technical draughtsmen. In this section, we describe the Project context of our work, and we enhance the main choices for EBP’s design.

The portability of parts libraries between different CAD systems is a major economic concern for CAD system users, for component manufacturers, and for CAD systems vendors. This portability would drastically increase the number of available part families on the different CAD systems, and therefore would increase the quality and the productivity of the design process for assembly modelling. To allow such a portability, a whole set of concerns, known as the CAD-LIB approach, has been developed [31]. They constitute the agreed basis of European and International standardisation works (CEN/TC310-pr ENV 40004 and ISO/TC184/SC4-ISO 13584 P-LIB).

The goals of the PLUS project, which is funded by the European Union (EU) as part of the ESPRIT R&D program, are:

---

2 Parametric Technology Inc. USA.

- to develop, on the basis of these concepts, a complete specification of the exchange format intended to be published as ISO 13584 (P-LIB),
- to validate this specification through the development of a complete set of pre-industrial tools which either generate or reuse this exchange format.

Beside object oriented data model for the exchange of parts library data, the project has to develop an approach for the exchange of part family geometric models. When the project started (in 1993), the parametric technology did not appear mature enough to be able to commit to the development of a standard exchange format for parametric data models. Therefore, the selected approach has been rather conservative. It consisted in developing a standard API (now available as ISO DIS 13584-31) associated with a FORTRAN binding. Every CAD system which supports some implementation of this standard API would be able to execute FORTRAN programs referring to this API.

However, this approach was in fact less conservative at it might appear at the first glance: the project also included the development of a PbD system that was intended to be able to generate these variant programs through pure graphical interactions.

This context defines the requirements that governed the EBP system design. (1) The generation process should be deterministic and fully controlled by the designer: one and only one shape should be generated for every allowed value of the input parameters, and precisely the shape which corresponds to the part. (2) Every kind of shape family which might be described using some conventional way of programming should be able to design 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.

Note that if the first requirement enforced to follow a procedural approach without any implicit inference or heuristic mechanism, none of the two last requirements were fulfilled neither by the existing parametric systems nor by the existing prototype of PbD systems. While several systems support pre-defined repetitive pattern structures, none of them, as far as we know, supports general purpose program/subprogram structure with graphical parameter passing mechanisms and recurrence-based iterations where each loop is defined through explicit recurrence relationships within the previous branch.

3. THE EBP SYSTEM

In this section, we describe the EBP system, from a standard CAD point of view to a more specific EBP viewpoint. Snapshots from the actual system are provided in Annexes.

3.1. EBP: a classic 2D CAD system

The EBP system [34] is a 2D CAD system. It manipulates simple geometric entities (points, unbounded lines, trimmed lines, circles, curves, and so on) and structured entities (composite curves, planar surfaces, and structured sets). Most constraints that result from technical drawing rules are supported. EBP provides a powerful display calculator that enables graphical inputs of both numerical and graphical expressions. Last, EBP allows model definitions through the use of menus and graphical interactions, over the X-MOTIF interface, and runs on Sun-Solaris[3] and DEC-Alpha[4] platforms.

Annex 1 shows a snapshot from the "classic" CAD system EBP. But this snapshot also shows a less usual feature on a CAD system: the logical display calculator. This feature, that shares some commonalities with some mechanism presented in the Smith's Pygmalion system [38] (bottom, left), enables already CAD users, trained in using some numerical

[3] Sun Inc., USA
[4] Digital, USA

display calculator, to graphically specify the control predicate of their alternative or repetitive shape aspects. For instance, a fillet may be defined as dependent onto the constraint :

val (line_1) > 2 x dist (point_1, point_2)

where val means length (of a line), and where line_1, point_1 and point_2 are graphically selected on the example.

3.2. Ambiguity removal and system determinism

The ambiguity of geometric constructs is not specific to variational systems: it is in fact intrinsic to geometry where 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.

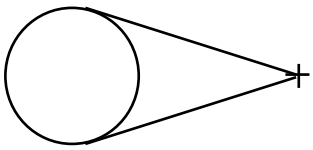

Figure 1: The two possible solutions for a line tangential to a circle

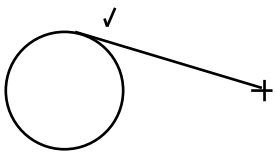

Figure 2: pointing solving

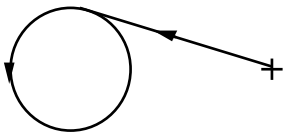

Figure 3 : Consistent orientation

This problem is perfectly solved in interactive geometric design. Most 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).

If this user-friendly dialogue convention shall be maintained at the user interface level, i.e., to design the example/program, it cannot be stored in the parametric program where, for different values of the parameters, the position might correspond to a different solution. But, this problem of constraint-based geometric constructs is also well-known in variant programming [35] where programming languages were used for defining part family models.

In such parametrics programs, context-free ambiguity removers are defined. In the target API, ambiguity removal is defined by topological informations: geometry entity orientation, and in/out for circles. For example, figure 3 shows the unique solution of the function Line_by_Point_and_Circle from figure 2.

Therefore, 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 whith 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.

3.3. EBP: a fully integrated PbD system

Visual Programming by Demonstration is achieved by “command recording” mode. This means that, 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. This mechanism, usual in programming languages, ensures an indirect reference from the program variables to their current values (in the example). It provides more independance between the PbD manager that deals with variables and the CAD system where example values are CAD database pointers. When the program is re-run (e.g., during modification), the EBP variables are not changed, but their addresses, stored in the dynamic context, are up-dated.

After **recording mode** activation (*Record Instance*), the EBP system "spies on" the user and builds an instance. Switching to **running mode** allows the system to run this instance (*Apply Instance*).

The only additional commands from traditional CAD systems are designed to *RECORD / NAME / LOAD / APPLY* instances and to *DEFINE / READ / WRITE / ENTER* parameters. Adding control structures requires more specific commands that are described in the following section.

A typical session of EBP would be as follows: after piece analysis (What are the parameters? Where are the dependencies? ... Every task that draughtsmen are used to do even without EBP, every time they plan to build some CAD model) the user begins PbD recording. He/She defines the parameters, and then, he/she draws an example, using the parameters instead of "direct values". The DEFINE command opens a window, where a name is given (it is displayed every time the program is run). The ENTER command enables entering the values of the parameters for the example. These values are entered through the CAD system interface, and their types define the parameter's types. The WRITE/READ command enables recording/getting values on/from a file which will be linked to the program instance. This is used for recording the allowed sets of parameter values for part families. As soon as parameters are defined, they are displayed in a menu where the user can pick them up, for example when defining expressions using the display calculators.

When the example is complete, the user can save the resulting instance, change some parameters, and try a new run. Recording in files values for parameters is very easy; this allows rapid testing. Recorded instances are included into a pop-up menu, and are usable with minimal effort.

The LOAD command selects an instance and the APPLY command runs it. Note that these commands may be selected both outside and inside the recording mode. In the first case, a model will be created into the CAD system database. EBP appears as a macro-by-example facility. In the second case, the APPLY command is recorded as a **call routine** in the embedding instance, and EBP ensures parameter passing. In both cases, after the apply command has been selected, EBP displays each parameter name and waits for a value. This value is defined using the whole CAD system user interface. This means that, when applying the instance in recording mode, parameter value definition consists of every expression that involves entities or parameter values of the embedding instance. These expressions are stored in the embedding instance the actual parameter value for the embedding instance.

3.4. Full control structure support

EBP includes full control structure support. More precisely, conditionals, iterations and subroutines are fully supported. The program context consistency is managed by the system [7, 8], and allows a consistent use of these structures.

Conditionals and iterations require Boolean expression definition, which is made through both 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).

Several iteration features are provided: set iterations over rubber-band rectangle selections and multiple geometric transformations are straightforward in CAD systems, and are supported by EBP. As for ambiguity removal, context dependent information (the two corners of the rubber-band rectangle) are translated into context-free information (the set of entity names which were referenced by this shortcut). But much more general features, such as *Repeat n times*, *While loops* and *Repeat ... until loops* are also provided. They allow recurrence-based definitions, in a pure interactive way.

Let us illustrate an interactive REPEAT - UNTIL definition: assume we want to design the drawing shown in figure 4. It is made of circles decreasing by a rate of one half radius at each loop, to reach a given minimum. The program to be constructed might be defined as an iteration (to obtain one column of circles) and a symmetry (to obtain the other one). Any loop but the first might be defined as follows: *build a first circle, tangential to the corresponding circle in the previous loop, tangential to the central vertical axis, and with a radius half that of corresponding circle in the previous loop; then create a second circle whose centre is the same as the first circle, and whose radius is half the previous circle.* As shown in figure 4, the first loop has a slightly different specification because the constraints that define the first circle refer to entities of the embedded context (the horizontal line)

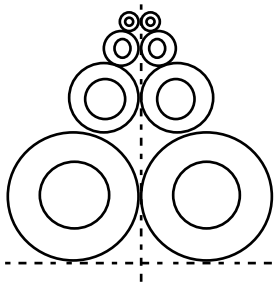

Figure 4: A (complex) figure

Three points are to be underlined: (1) in one loop, objects are defined from objects created during the previous loop and during the current loop; (2) actions performed during the first loop are the same as actions performed during the next ones; the only difference concerns the reference to the objects belonging to the previous loop, which must be found in the embedding context (in the example, the first circle is tangential to both existing lines); (3) recurrence relationships may be fully defined during the second execution of the first loop actions, just by asking the user for the actual object that shall be used for every referenced object in the first loop: it may be the same object (the vertical line in our example) or any object which has been created during the first loop (in our example, the horizontal line must be replaced by the first circle of the first loop).

These three remarks are the basis of the user interface and the dialogue conventions of the EBP system. Two commands are provided (REPEAT, and UNTIL). The user selects the

REPEAT command, and defines the first loop, with full access to the embedding context. Then, he/she selects the UNTIL command.

The system automatically switches to running mode, in order to perform the second loop. Every command performed during the first loop is run. While running each command, the system echoes the embedding context objects, asking the user for either picking the same (which defines a constant reference during the loop run) or picking another object which has been created during the previous loop (which defines a recurrence relationship, re-evaluated for any loop). Every other entity selection is refused by the system. The same mechanism is provided for any expression, allowing the definition of new expressions (in our example, the expression for the radius of the first circle).

Once each object reference has been confirmed or changed by some reference to entities from the first or the current loop, the system asks the user for the definition of the control expression. This solution implicity defines a recurrence-based relation which is consistent for the whole iteration. Using the display calculators, users can access every object from both branches of the iteration and from the embedding context. After that, the system runs the remaining loops until the controlling expression fires. Each recurrence relation is evaluated from any loops.

3.5. An actual programming environment

EBP is a complete programming environment, that provides every usual debugging facility, in a programming with example style. Every interaction with programs is done through example interaction. Generated programs are shown in some specific window which is only displayed on user request. A special menu, the *visit menu*, allows re-running the instance.

3.5.1. Intelligent UNDO/REDO and program modification

Both during program recording and debugging, every modification may be done into the program. Successive UNODOs enable returning to previous steps. Then, some additional constructs may be done, and some steps may be modified or deleted. EBP manages the program dynamic context to ensure that addition/deletion does not change references to variables. When REDOing some command which references some deleted entity, EBP asks the user for a new entity to replace the previous one.

3.5.2. Visual debugging

Because the textual representation of programs is never supposed to be displayed, debugging "virtual" programs might appear difficult. Fortunately, the example always give an input/output interface with the program.

Like every debugging environment, the EBP manager enables programs to be run step by step or until their end, or to be reinitalized. It also provide the possibility to visit the program **until one entity is drawn**. The user graphically selects this entity, and the program run until it is drawn. The user may then make every modification on the example/program, before REDOing the remainder of the program.

3.5.3. Program generation

EBP was designed to produce standard parts portable program libraries (ISO13584 compliantFORTRAN programs reference the ISO 13584-31 API.

4. EBP AND PBD SYSTEMS

In this section, we use the summary sheet which has been proposed in [5] for PbD system characterization. We explore Uses and users (4.1), User interaction (4.2), Inference (4.3.), Program constructs coverage (4.4), and we conclude by evolving consideration over the EBP project (4.5).

4.1 Uses and Users

Application domain
EBP addresses mechanical 2D CAD. While Metamouse [18], Chimera [13], the Geometer's Sketchpad [12] and Mondrian [14] relate to the graphical area, none of these systems specifically addresses CAD. The only experimental PbD CAD system is Geonode [41].

Tasks within the domain
EBP is designed for producing standard programs to be run over different CAD systems. It may be related to Pygmalion [38], Tinker [15] and Smallstar [10] for its programming goal, but explicit programming or textual modification of programs is never required. It should also be compared with some systems which produce programs, like Peridot [20] or Mondrian [14] (LISP code), LEMMING [27] (specific NIC code), and Geonode [41] (C code). Unlike these systems, EBP produced standard API codes and provides computationaly completeness of generated programs.

Intended users
EBP addresses trained end-users, without programming knowledge. This is completely different for example from KidSim [6] where intended users are kids, or from the Geometer's Sketchpad [12] 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. In contrast with Metamouse [18], Tels [43] or Tinker [15], 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 avalaible on the CAD system allow *a priori* definitions.

Interactive program constructs definition
Like Pygmalion [38], EBP has special commands for program constructs definition. Unlike Smallstar [10], 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.

Moreover some predefined control structures, which are implicit in CAD systems, are fully integrated in EBP (set iteration over rubber-band rectangle selection, multiple geometrical transformations).

Modifying and debugging
Debugging facilities in EBP may be related to Zstep 94 [16]. 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, like Smallstar [10] or GeoNode [41]. Nevertheless, it does not require nor allow any interaction over it. While Chimera [13] and Pursuit [19] 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 [38] which requires that the remainder of a program be re-demonstrated, and Smallstar [10] or Chimera [13] 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, as for example the Voice Input solution [40]. 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 [23], this fact was considered as a major drawback for PbD systems. Cypher's 'Watch what I do' has a good summary on these features in existing PbD systems, that we will summarise below. In current PbD systems, subroutines with parameters are proposed in Chimera [13], Geometer's Sketchpad [12], Macros-by-example [28] and AIDE [30]. Nevertheless, parameters are limited to input parameters; so doing, we should consider them as 'macros', instead of real subroutines. 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. Recursion may only be found in Geometer's Sketchpad [12], but the lack of control expressions in that system makes the recursive definition incomplete: users are required to give the recursion depth they want the system to apply at any run.

Conditionals are frequently supported (Peridot [20], Turvy [17], Metamouse [18], TELS [43], Pursuit [19]), but they are often restricted to exception handling (object existence test, geometrical constraint satisfaction, ...). Last, iterations are mainly *set iterations* [10] which consist of repeating actions on object lists (Peridot [20], Eager [3], Pursuit [19]) or in texts (TELS [43]). Some systems search for repetitive patterns in user actions, and infer repetitive tasks (Eager [3], Turvy [17], Metamouse [18], Peridot [20]), or allow predefined number of object creations (Peridot [20]). General recurrence-based iterations are not supported. The only system that introduces restricted forms of recurrence relations (chosen from predefined classes of recurrence patterns) is Peridot: '*the constraint in all items after the first is made to refer to the corresponding item in the previous cycle*' [21]. No existing system or prototype is able to generate the repetitive structure proposed in Figure 4.

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 [15], Smallstar [10], Geonode [41]).

4.5. Evolving considerations

EBP has been implemented in Ada language, for about 200,000 documented lines. It runs on X-MOTIF platforms, Sun-Solaris and Dec-Alpha.

Within the PLUS project, EBP is already used to generate the library of a bearing and linear system supplier. Some other exchange formats are planned to be generated, including AutoLISP and a STEP-compliant data model oriented parametric exchange format which was recently proposed [32]. Future work includes the development of an industrial product for ISO 13584-compliant file generator, and 3D extension.

CONCLUSION

In this paper, we have presented the EBP system, which constitutes a complete PbD environment for CAD parametric design. 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 program;
- is able to generate conventional programs that may be used on different CAD systems.

From the PbD point of view, the EBP system proves that, at least in some application area where system users have particular skills, complete PbD environments may be developped. Complete PbD environment means both computational-completeness of generated programs and real debugging with example facilities.

From the CAD systems point of view, the EBP system proves that parametrics CAD systems, which are already very successful for sequential (or simple repetitive pattern-based) parametric design, may be extended to support the parametric design of every conditional or repetitive shape aspect.

From a user interface viewpoint, usual interactive systems are generally only sequential systems. The EBP system suggests extending the dialogue command language towards recurrence-based repetitive command constructs. It also proves that very powerful macro-with-example recorders may be developped.

ACKNOWLEDGMENTS

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.

REFERENCES

[1] Bouma, W., Fudos, I., Hoffmann, C., Cai, J., and Paige, R., "Geometric Constraint Solver," *Computer Aided Design*, vol. 27, pp. 487-501, 1995.

[2] Cugini, U., Folini, F., and Vicini, I., "A Procedural System for Definition and Storage of Technical drawings in Parametric Form," presented at EUROGRAPHICS'88, pp. 183-196, 1988.

[3] Cypher, A., "Eager: Programming Repetitive Tasks by Example," presented at CHI'91, New Orleans, Louisiana, pp. 33-39, 1991.

[4] Cypher, A., "Watch What I Do: Programming by Demonstration," . Cambridge, Massachusetts: The MIT Press, 1993, pp. 604.

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

[6] Cypher, A. and Smith, D. C., "KidSim: End User Programming of Simulations," presented at CHI'95, Denver, Colorado, pp. 27-36, 1995.

[7] Girard, P. and Pierra, G., "End User Programming Environments : Interactive Programming-On-Example in CAD Parametric Design," presented at EUROGRAPHICS'90, Montreux, pp. 261-274, 1990.

[8] Girard, P. and Pierra, G., "Structures de contrôle générales en Programmation par Démonstration," presented at Septièmes Journées sur l'Ingénierie de l'Interaction Homme-Machine, Toulouse, pp. 61-68, 1995.

[9] Glinert, E., "Visual Programming Environments," in *Press Tut.*: IEEE Computer, 1990, pp. 600.

[10] Halbert, D., "Programming by Example," in *Berkeley*, 1984.

[11] Hirakawa, M., Tanaka, M., and Ichikawa, T., "An iconic programming system: HI-VISUAL," *IEEE Transactions on Software Engineering*, vol. 16, pp. 1178-1184, 1990.

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

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

[14] Lieberman, H., "Mondrian: a Teachable Editor," in *Watch What I Do: Programming by Demonstration*, A. Cypher, Ed. Cambridge, Massachusetts: The MIT Press, 1993, pp. 341-360.
[15] Lieberman, H., "Tinker: A Programming by Demonstration System for Beginning Programmers," in *Watch What I Do: Programming by Demonstration*, A. Cypher, Ed. Cambridge, Massachusetts: The MIT Press, 1993, pp. 49-66.
[16] Lieberman, H. and Fry, C., "Bridging the Gulf Between Code and Behavior in Programming," presented at CHI'95, Denver, Colorado, pp. 480-486, 1995.
[17] Maulsby, D., "The Turvy Experience: Simulating an Instructible Interface," in *Watch What I Do: Programming by Demonstration*, A. Cypher, Ed. Cambridge, Massachusetts: The MIT Press, 1993, pp. 239-270.
[18] Maulsby, D., Witten, I., Kittlitz, K., and Fraceschun, V., "Inferring Graphical Procedures: the Compleat Metamouse," *Human-Computer Interaction*, vol. 7, pp. 47-89, 1992.
[19] Modugno, F. and Myers, B. A., "A State-Based Visual Language for a Demonstrational Visual Shell," presented at IEEE Symp. on Visual Languages, Saint-Louis, Missouri, pp. 304-311, 1994.
[20] Myers, B. A., *Creating User Interface by Demonstration*: Academic Press, 1988.
[21] Myers, B. A., "PERIDOT: Creating User Interfaces by Demonstration," in *Watch What I Do: Programming by Demonstration*, A. Cypher, Ed. Cambridge, Massachusetts: The MIT Press, 1993, pp. 125-154.
[22] Myers, B. A., "Report on the end-user programming working group," in *Languages for Developing User Interfaces*, B. A. Myers, Ed. Boston: Jones & Bartlett, 1992, pp. 343-366.
[23] Myers, B. A., "Taxonomies of Visual Programming and Program Visualization," *Journal of Visual Languages and Computing*, vol. 1, pp. 97-123, 1990.
[24] Myers, B. A., Giuse, D., Dannenberg, R., Vander Zanden, B., Kosbie, D., Pervin, E., Mickish, A., and Marchal, P., "GARNET: Comprehensive Support for Graphical, Highly Interactive User Interfaces," *IEEE Computer*, vol. 23, pp. 71-85, 1990.
[25] Nardi, B. A., *A Small Matter of Programming, Perspectives on End User Computing*. Cambridge, Massachusetts: The MIT Press, 1993.
[26] Newell, R., Parden, G., and Parden, P., "Parametric Design in MEDUSA System," presented at CAPE'83, Amsterdam, 1983.
[27] Olsen, D. R., Ahlstrom, B., and Kohlert, D., "Building Geometry-based Widgets by Example," presented at CHI'95, Denver, Colorado, pp. 35-42, 1995.
[28] Olsen, D. R. and Dance, J. R., "Macros by Example in a Graphical UIMS," *IEEE Computer Graphics and Applications*, vol. 12, pp. 68-78, 1988.
[29] Owen, J., "Algebraic Solution for Geometry from Dimensional Constraints," presented at ACM Symp. Found. Solid Modeling, Austin, Texas, pp. 397-407, 1991.
[30] 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, Ed. Cambridge, Massachusetts: The MIT Press, 1993, pp. 383-402.
[31] Pierra, G. and Aït Ameur, Y., "Logical Model for Parts Libraries," ISO-CD 13584-20 1994.
[32] Pierra, G., Aït Ameur, Y., Besnard, F., Girard, P., and Potier, J.-C., "A General Framework for Parametric Product Model within STEP and Parts Library," presented at European PDT Days, London, UK, pp. 69-104, 1996.
[33] Pierra, G., Potier, J.-C., and Girard, P., "Design and Exchange of Parametric Models for Parts Library," presented at 27th International Symposium on Advanced Transportation Applications, ISATA'94, Aachen, Germany, pp. 397-404, 1994.
[34] Potier, J.-C., "Conception sur exemple, mise au point et génération de programmes portables de géométrie paramétrée dans le système EBP," in *LISI*. Poitiers: ENSMA, 1995, pp. 140.
[35] Roller, D., "Dimension-Driven Geometry in CAD: a Survey," in *Theory and Practice of Geometric Modeling*: Springer-Verlag, 1990, pp. 509-523.
[36] Sassin, M., "Creating User Intended Programs with Programming by Demonstration," presented at IEEE Symp. on Visual Languages, Saint-Louis, Missouri, pp. 153-160, 1994.
[37] Shah, J. J. and Mäntylä, M., *Parametric and Feature-based CAD/CAM: Concepts, Techniques and Applications*. New York: John Wiley & Sons, 1995.
[38] Smith, D. C., *A Computer Program to Model and Stimulate Creative Thought*. Basel: Birkhauser, 1975.
[39] Solano, L. and Brunet, P., "Constructive Constraint-Based Model for Parametric CAD Systems," *Computer Aided Design*, vol. 26, pp. 614-621, 1994.
[40] Turransky, A., "Using Voice Input to Disambiguate Intent," in *Watch What I Do: Programming by Demonstration*, A. Cypher, Ed. Cambridge, Massachusetts: The MIT Press, 1993, pp. 457-464.

[41] Van Emmerik, M., “Interactive Design of Parametrized 3D models by direct manipulation,” . Delft, 1990, pp. 141.
[42] Wilde, N., “WYSIWIC (What You See Is What You Compute) Spreadsheet,” presented at IEEE Symposium on Visual Languages, Bergen, Norway, pp. 72-76, 1993.
[43] Witten, I. H. and Mo, D., “TELS: Learning Text Editing Tasks from Examples,” in *Watch What I Do: Programming by Demonstration*, A. Cypher, Ed. Cambridge, Massachusetts: The MIT Press, 1993, pp. 183-204.