Paper published in: Proc. of the 27th International Symposium on Advanced Transportation Applications, ISATA'94, Aachen, Germany, 31st October - 4 November 1994 , pp. 397-404

Design and Exchange of Parametric Models for Parts Library[1]

Guy PIERRA, Jean-Claude POTIER, Patrick GIRARD

Laboratoire d'Informatique Scientifique et Industrielle - ENSMA
Site du Futuroscope
86960 FUTUROSCOPE - FRANCE
Tel.: (33) 49-49-80-60 - Fax: (33) 49-49-60-84
e-mail pierra@ensma.univ-poitiers.fr

Introduction

In recent years, a new generation of CAD systems appears on the market. These systems, which use different terminology to characterize their capabilities, provide for easy modification of a designed shape.

The common property of these systems is that their internal data structure is twofold. On the one hand, they record the explicit shape of the current designed product. On the other hand, they record the identifiers of the entities that constitute the shape, the constraints between these entities and the input values involved in these constraints, thus recording a program. Providing new input values to this program enables to create a variant of the designed shape. The underlying principles of these systems are all but new. The very innovative feature is the user friendly interface which enables to record and to trigger such a program.

The major drawbacks of these systems, denoted below as either parametric or variational, is the lack of capability to exchange their internal data structure. The definition of an exchange format is confronted with the union/intersection problem: parametric and variational approaches are evolving technologies, hence, each system uses different entities and, above all, records different relationships between these entities. An exchange format consisting of the union of all the possible relationships will hardly be processable by any one of these systems. An exchange format defined on the basis of the common relationships will not be powerful enough to record any one of these models.

The goal of this paper is to present how this problem may be solved in the context of standard parts libraries, where the exchange of parametric models is crucial. In this

[1] The research described in this paper was funded partially by EU under project ESPRIT III # 8984 (PLUS), and partially by the French Ministry of Industry under grant 93.4.930080.

context, the parametrized shapes are simple enough to enable to restrict the exchange format to the intersection of the various sets of relationships supported by the different parametric or variational systems. The choice of a standard procedural programming language (C, FORTRAN, ADA, ...) provides for expression of the ordering of the relationships to be computed. Like the parametric and the variational approaches, this approach is all but new. The innovative features, presented in this paper, are both the capability to generate this exchange format through a user-friendly interface, by recording graphical user interactions, and the capability to restore the parametric model of the exchanged shape on the receiving system, if this system supports such facilities.

1 - What is parametrics?

The concept of a class that constitutes the template of a set of (possible) instances is now well known in software engineering. The kernel concepts of a class are twofold. (1) A class describes the template structure of all its instances. (2) A class gathers the structural description of the instances ("attributes") and the behavioural description of the instances ("methods", "features", "operations"). Inheritance and message passing are not concepts intrinsic to a class: they are only additional features provided by the well-known object-oriented approach.

The concept of parametrics is directly related to this concept of a class. But parametrics has two specificities. First, it addresses only those classes whose instances are completely characterized by (i.e., that may be computed from) a set of numeric-valued or Boolean-valued attributes (called "parameters"). Second, the class level (i.e., both the structural description and the behavioural description) are always described together with one specific instance, that we call the *current instance* [PIERRA 1994b]. The structure of the current instance defines the structural template of the class. This very specific feature explains the user-friendliness of parametric systems: when building a class, the user interacts only with the instance level. It is the responsibility of the system to implicitly create the corresponding class. The user may then "modify" the current instance: in fact it creates a new instance of the implicitly recorded class. This approach, not so usual in software engineering, corresponds precisely to the concept of example-based programming [MYERS 1990]: from an example designed by a user, the system may infer a general program.

Even if it is mainly used for geometric shape representation, the concept of parametrics is in fact much more general: it applies to any kind of representations (whose content may be computed from "parameter") or even to product class definition (whose properties may be derived from a subset of attribute values, called in the Parts library emerging Standard ISO CDC 13584, "identification_attributes").

Two categories of systems provide for parametrics. The variational systems are based upon the concepts of declarative programming. The current instance is designed in the same way as on the non parametric CAD systems. But a second step is added. It is possible to state constraints between the model entities, by use of spatial relationships, or other ones [ROLLER 1990]. After constraint resolution, the class level is able to generate a family of varying instances. The limits of this approach result from its advantages. The designer does not specify the way to be followed to realize the different instances of the family. He only "constraints" the family. The system is the sole

responsible for the way the objects of the family are generated, solving a system of equations that is, in general, neither linear nor convex [PIERRA 1994b]. Hence, construction results are often unpredictable [VERROUST 1990]. Moreover, modifications are global. Because of the complete separation between definition and evaluation, the “debugging” and “maintenance” of variational models are rather difficult.

The parametric systems, which provides a similar end-user interface, are based upon radically different concepts. The constructive process is recorded into the class level, to be re-evaluated after any modification [CUGINI 1988], [VAN EMMERIK 1990]. Hence, the whole class which is represented by the recorded process is directly induced from the design process which was used for the example of the family. This approach is theoretically less universal than the variational one. But, on the other hand, it allows much more control over the generation process.

A strict control over the generation process being crucial for standard parts libraries, we will mainly address, in this paper, the second approach that is usually referred to as "parametrics".

2 - Contents of a parametric model

A parametric instance consists of a set of parameters (usually numeric, sometimes Boolean) and of an ordered list of representation items (in geometry: points, curves, surfaces, solid bodies, more topology). The (implicitly recorded) class description contains, for each representation item, the function that enables to deduce it from (1) the parameter values and (2) the previous representation items of the representation item list [PIERRA 1994b].

For parametric shapes, these functions, called the *parametric function*s, are based on four constructs:

– constraint-based definitions, that enable to define a representation item through constraints with other representation items (e.g., to define a point as the intersection of two lines),

– numeric-valued and Boolean-valued expressions (e.g., x coordinate of a point defined as the half of some parameter),

– grapho-numeric expressions, that enable to involve in a numeric expression a numeric-valued function of other representation items (e.g., *distance_of* (<Point_1, Point_2>)),

– geometric expressions that enable to involve in a constraint-based definition virtual representation items (e.g., *projection_of* <point_1> *onto* <line_2>).

Parametrics systems are different from each other according to (1) the representation items that may be used within the instance and (2) the parametric functions that may be used to specify each type of representation item.

3 - Parametric model representation

The usual way for representing parametric models in parametric systems is to gather in the same data structure both the instance level (the current instance) and the class level (the composition of functions). This structure is always private. It appears to

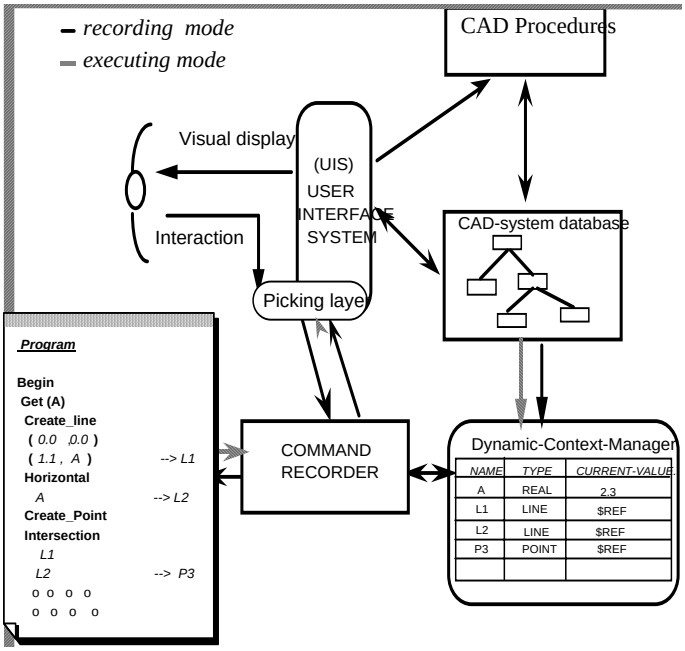

Fig 1 : *Architecture of the LIKE system*

be specific of each system, and even if some proposal for exchanging such a structure are emerging [PIERRA 1994a], there exists, presently, non standardized format that enables such an exchange.

There also exists another way, much more traditional in the CAD area, to represent a parametric model. It consists (1) in specifying the allowed parametric functions as an application programming interface (API), and (2) in representing the class level of the parametric model as a program which refers to this interface.

This second approach is one of the approaches provided in the emerging ISO Standard Parts Library (ISO CDC 13584). It is much less old fashioned as it may appear at first glance. First, thanks to the control structure provided by the support programming language, its expressive power is greater that the (existing) parametric models. Second, as outlined in the next section, this program may be generated out of an example-based programming environment that provides to the end-user (the "programmer") an interface as friendly as the existing parametric CAD systems.

4 - Interactive definition of a parametric program

The example-based approach for programming [MYERS 1990], [GIRARD 1993] consists in "spying" a end-user when he or she defines an example of the program thus inferring the program. Several systems are based on this approach [MYERS 1990] [VAN EMMERIK 1990]. The LIKE system [GIRARD 1992] [GIRARD 1993] which is devoted to CAD, enables to record the user commands, to manage the program variables and parameters, and even to introduce the usual control structures of structured programming [GIRARD 1994]. The basis of the object management is an (implicit) dynamic program context. Each representation item created in the CAD database implicitly "declares" a new variable. Each representation item picked up by user (in the example, which stands for the *current instance* of parametric systems), is replaced by the corresponding variable (in the program).

Unfortunately, standing at the user command level, the program recorded by the LIKE system may only be processed by the same system which enables to record it. To clarify the problems which are to be solved to generate a neutral program, we present them on a (very) simplified version of the new Example-Based Programming environment, called EBP, we have developpedto generate neutral programs.

• **Entities:** EBP provides Direct Manipulation facilities on *Points, Lines* and *Circles*. Moreover, it can use atomic values (*numerical, Boolean*) and *locators* (generally "clicks" on pointing device).

• **Commands for entity creation:** *Create-Point, Create-Line* and *Create-Circle.* According to the provided operands, these three commands trigger the following seven CAD system procedures: *Create-Point-By-Cartesian-Values*, *Create-Point-By-Virtual-Point* (Click or result of expression), *Create-Line-By-Two-Points*, *Create-Line-By-Point-And-Circle* (resulting line is tangential to the circle), and the three procedures constructing a circle based upon its centre: *Create-Circle-By-Centre-And-Radius, Create-Circle-By-Centre-And-Line* (tangential to the line), *Create-Circle-By-Centre-And-Circle* (tangential to the circle).

• **Expressions:** Some commands allow calculation of virtual entities which may be used by other procedures: *Extremity* (which calculates the extremity point of a line), *Centre* (which calculates the centre point of a circle) and *Intersection* (which calculates the intersection point of two entities, Line or Circle).

Fig 2: *Simplified description of the EBP system*:

4.1 - Syntactical differences between the command language and the program

A CAD system allows model construction by means of system procedures involving parameters. In interactive mode, procedures and parameters are implicitly controlled by a user which manipulates commands and operands. To record a program requires the explicit description of the procedures involved and their parameters. The correspondence between operands and parameters may be insured, as in the LIKE system, by managing the dynamic context of the implicit program. The main difficulties consist in identifying the procedures triggered and the parameters of these procedures. These difficulties stand first at the syntactical level.

The first idea to erase the difference between interaction and programming level is to identify commands as procedures, and operands as parameters. That ideal case is unfortunately very uncommon in normal use of CAD systems. Four major cases may be found. For example, the following sequence of interactions (italics) may be directly translated to the next sequence of procedures (bold, in pseudo-Pascal style):

Interactions:
Create-Point 0.0 0.0 Create-Point 1.0 0.0 Create-Line Point-1 Point-2
Procedures:
Point-1 := Create-Point-By-Cartesian-Value (0,0 , 0,0)
Point-2 := Create-Point-By-Cartesian-Value (1,0 , 0,0)
Line-1 := Create-Line-By-Two-Points (Point-1 , Point-2)

4.1.1. Natural fault tolerance

A program must be faultless: in other terms, syntactically correct. User Interface System (UIS) have to be fault-tolerant: it is only natural to abort a construction, whatever the reason may be. For example, the following interactive sequence must be interpreted without any problem:

Interactions:
Create-Circle Point-3 Create-Line Point-1 Point-2
Procedures:
Line-1 := Create-Line-By-Two-Points (Point-1 , Point-2)

The two interactions "Creation-Circle" and "Point-3" are not used by the UIS. Nevertheless, no error shall occur. So, irrelevant interactions must be omitted in the translation process.

4.1.2. Persistence of commands

One of the most common feature of UISs is called command persistence or remanence. It allows the use of two procedures driven by the same command while avoiding to repeat the command:

```
Interactions:
Create-Line   Point-1   Point-2   Point-3   Point-4
Procedures:
Line-1      := Create-Line-By-Two-Points ( Point-1 , Point-2 )
Line-2      := Create-Line-By-Two-Points ( Point-3 , Point-4 )
```

Direct translation will try to do an action with four parameters, or will ignore the second construction.

4.1.3. Expressions vs. explicit operands

In a classic programming language, the use of expressions is widespread. In an UIS, it is achieved by the use of procedures which produce (like functions) a result which is immediately used by the UIS, without any user intervention, to trigger another procedure. For example, the next sequence will construct a line whose one extremity is exactly the centre of a circle:

```
Interactions:
Create-Line   Point-1   Centre   Circle-1
Procedures:
Line-1      := Create-Line-By-Two-Points ( Point-1 , Centre ( Circle-1 ) )
```

That dependency between the result of the function Centre and the action Create-Line-By-Two-Points must be known to effectively translate the sequence.

4.1.4. Activity files

The last syntactic problem of the translation of command recording sequences results from a common feature of UISs called "threads of task accomplishment" [BASS 1991]. Authors frequently describe the ability, in a UIS, to stop an unachieved sequence of interactions to realize another one, and to terminate the first sequence after the end of the second one. As a result, the operands of one command are embedded within the operands of another one.

All the previous cases point out the syntactic difference between interactive command level and program action level. In command level (which is recorded by LIKE), interactions are heavily context sensitive. In action level (as in programming languages), procedures and parameters shall conform to simpler syntactic rules, suitable for parsing and compiling.

4.2. Semantic problems

Assuming that all the syntactic problems have been solved or bypassed, the interpretation of programs requires a semantic phase, which shall precede the code

generation. We will only develop in this section the problem of command overloading and ambiguous geometric constructs.

4.2.1. Command overloading

We can notice that only three commands may trigger seven different procedures. The parameter types allow discrimination of procedures. This is a very common case in CAD systems.

4.2.2. Ambiguous geometric construct

Constraint-based definition often involves construction ambiguities. The creation of a line starting from a point and tangential to a circle is a good example. Assuming we want to create only one of these lines, we shall do, with EBP:

```
Interactions:
Create-Line   Point-1   Circle-2
Procedures:
Line-2      := Create-Line-By-Point-And-Circle ( Point-1 , Circle-2 )
```

But how can the system choose between the two solutions? Most of interactive CAD systems (and it is the case in EBP) derive this information from the position of the click which designates the circle: the line which is closest from that position shall be chosen. This solution is very user-friendly: it does not require any additional interaction, and it is quite natural (the user approximately knows where his line must be created). Unfortunately, it is not usable in programming mode. In fact, during program execution, objects are known by their name in the program, and the designation location shall not remain available (they are meaningless). The only possible solution consists in automatically translating the graphical discriminating values into calculated ones.

In EBP this problem is solved as follows. In program recording mode, the CAD system procedure get the designation position and returns a computed ambiguity-remover (based on entity orientation, as in ISO CDC 13584-31) which is stored in the program. When the program is run (for instance for debugging) a different procedure is called that get the ambiguity-remover as input parameter.

4.3 - From command recording to procedure recording: The EBP system

All the previously discussed difficulties may be removed by changing the level were the user commands are to be captured. The UIS analyser exactly knows the missing information in the LIKE's recorded program. When triggering a system procedure, the analyzer knows the exact syntactic tree of the corresponding call statement, even if it involves expressions. For example, the analyser is able to construct the next complex syntactic tree when it analyses the following sequence:

```
Interactions:
Create-Line   Centre   Circle-1   Intersection   Line-1   Circle-1
Procedures:
Line-2  := Create-Line-By-Two-Points (    Centre ( Circle-1) ,    Intersection   ( Line-1 , Circle-1 ) )
```

Capturing the user commands only when the UIS triggers some procedure solves all the mentioned problems, and restricts the (main) exchanges between the system and the program recorder to the UIS of the first one. It shall be noted that this solution is similar to that retained by Yamaguchi [YAMAGUCHI 1987] to improve its system by historical recording. A major distinction must be made between those two works. The

Yamaguchi's system is based on artificial intelligence analysis, and it only uses the command tree to help this analysis. Our approach is purely algorithmic, and the tree is the basis of the constructed program.

On the other hand, the main difference with the LIKE system is that the recorded program is now a (structured) list of parametric functions, clearly associated with their parameters. To generate a parametric program that refers to an API which supports these parametric functions is therefore straightforward. The construction of the program is completely implicit for the user. Like in a parametric CAD system, the user only works at the current instance level.

Conclusion

Parametrics is a new technology whose main characteristic is to provide a user-friendly interface to implicitly designed parametric programs. The usual architecture of such systems is to link together, in the same data structure, both the instance level (*the current instance*) and the class level (*the parametric program*). At the present time, no Standard for exchanging such data structures exists.

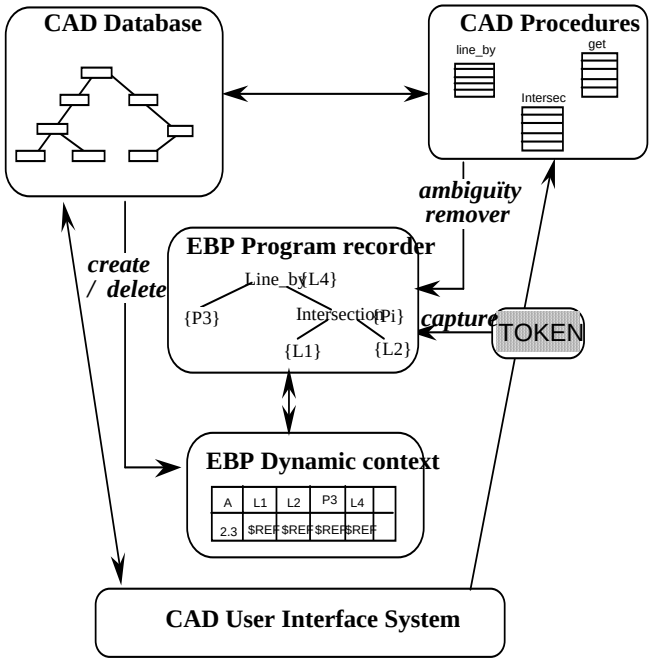

***Fig 3:** Architecture of the EBP system*

Exchanging parametric models by means of parametric programs based on a standardized API is a traditional approach. It provides all the expressive power of the usual programming language. Nevertheless this approach does not provide for end-user parametric design.

In this paper we have investigated the concept of Example Based Programming for parametrics. We have also presented the EBP system which enables to an end-user to design a neutral parametric program with the same friendliness of the interface as he or she may use on a parametric system.

Moreover when a parametric model, exchanged in the format of a program, contains only sequential constraint-based statements (and/or graphical-expressions), and when it refers to numeric values through numeric entities (i.e., through their entity names), the processing of such a model on a parametric system (supporting the same parametric functions and numeric expressions) may generate in the internal parametric data structure the storage of the existing functions and/or expressions relating to the entities of the model, thus generating the parametric model itself.

A parametric or a variational model is basically a program. Using a programming language for describing and exchanging such a program is the more straightforward approach. We have shown in this paper that this approach may be as user friendly as any other approach.

Acknowledgments

The research described in this paper was funded partially by EU under project ESPRIT III # 8984 (PLUS), and partially by the French Ministry of Industry under grant 93.4.930080.

References

[BASS 1991] BASS L., COUTAZ J. : Developing Software for the User Interface, *SEI Series in Software Engineering, Addison-Wesley , 1991, 251p.*

[CUGINI 1988] CUGINI U., FOLINI F., VICINI I. : A Procedural System for the Definition and Storage of Technical Drawings in Parametric Form, *Proc. of EUROGRAPHIC'88, 1988, pp. 183-196.*

[GIRARD 1992] GIRARD P. : Environnement de programmation pour non programmeurs et Paramétrage en Conception Assistée par Ordinateur : Le Système Like, *PhD Thesis, Poitiers, 1992, 195p.*

[GIRARD 1993] GIRARD P., PIERRA G. : Command Recording versus Parametric and Variational Systems, and old / new third way of parametrizing CAD models by End Users, Proc of *COMPEURO'93, Paris (Mai 1993), Ed. IEEE Comp. Society Press, pp. 194-200.*

[GIRARD 1994] GIRARD P., PIERRA G. : One more step towards end-user programming environments: introducing control structures in visual example-based programming, *Tech. Report LISI/94003, 1994, 15p.*

[MYERS 1990] MYERS B. : Taxonomies of Visual Programming and Program Visualization, *J. of Visual Lang. and Comp., 1, 1990, 97-123.*

[PIERRA 1994a] PIERRA G. : A general framework for parametric product modelling, ISO-STEP meeting, Davos, Mai 1994, ISO / TC 184 / SC4 / WG2 N183, 51 p.

[PIERRA 1994b] PIERRA G., POTIER J.C., GIRARD P.: The EBP system : Example Based Programming for parametric design, Workshop on Graphic and Modeling In Science ant Technology, Coimbra, 27-28 june 1994 (to appears in Springer Verlag Series)

[ROLLER 1990] ROLLER D., SCHONEK F., VERROUST A. : Dimension-driven geometry in CAD : a survey, *in Theory on practice of geometric Modeling, Springer Verlag, 1990, pp. 509-523.*

[VAN EMMERIK 1990] VAN EMMERIK M. : Interactive design of parametrized 3D models by direct manipulation, *PhD Thesis, Delft University, NETHERLAND, 1990, 141p.*

[VERROUST 1990] VERROUST A. : Construction d'objets géométriques définis par des contraintes, *BIGRE, 67, Jan. 1990, pp. 62-74.*

[YAMAGUCHI 1987] YAMAGUCHI Y., KIMURA F. : Interaction management in CAD systems with history mechanism, *Proc. of EUROGRAPHICS'87, 1987, pp. 543-557.*