Simulating Feature-based Modelling in Programming-by-example Parametric CAD

Evgueni N. Loukipoudis and Jan A.A. Melkebeek
Department of Electrical Power Engineering
University of Gent
Sint-Pietersnieuwstraat 41, B-9000 Gent, Belgium

Abstract

As an alternative to the development of a full feature-based CAD system, we suggest a fairly simple extension to a system that possesses parametric capabilities so that feature-based modelling interface is simulated. We use programming-by-example in defining features by parametric programs following a purely procedural approach. The parametric programs constitute the feature model which is built up in a bottom-up manner. The structuring of the object in features allows modification to the parametric description in a graphical manner. Some problems related to the use of features in an engineering system for the purpose of simulation and analysis as well as the handling of duality and complementarity in a feature model are discussed. We have restricted our approach to 2D models and have implemented Electric Machinery specific features into a 2D CAD system.

Introduction

The notion of features, though not strictly defined, has proved appealing to engineers because of its application in capturing the design intent and semantics.

Apart from generating objects, features can represent complicated modification to the geometry, either as generalised Boolean operation with unambiguously defined semantics, or by operations on the boundary representation – surface semantics [1, 5, 9]. In both cases, for the sake of generality features can be described by procedural operations.

The procedural nature of feature modelling has an advantage over the classical design by constraint methods. It allows objects, comprising a variable number of items to be modelled using arrays of features (repetition in a matrix or circle) – the *pattern feature* as defined in the STEP standard.

Features can be considered as *parametric objects* in a broad sense, because they are driven by a number of numerical and graphical parameters and so is the modelling procedure in which they are embedded. The *feature model* is generally the procedure of attaching features to a basic object and imposing constraints that define feature dimensions and location with respect to each other.

This procedure is re-executed (following the history of feature assignment) any time a feature parameter changes, or a constraint on a feature parameter is modified. The execution generates the explicit geometry of instances which are further processed by standard boundary-based CAE methods.

Programming-by-example Parametric Systems

Feature modelling is a parametric design method, because an object with features is a generalised description of a similar (but not identical) topological structure with varying geometry, driven by the parameters of the features and the parameters that determine their position in the object [6].

Programming-by-example systems provide an easy manner of parametrising geometry based on the history of its construction [2, 4, 7]. The sequence of commands used for the construction of the example is the basis of the parametric program, the executions of which produce the different instances. The

variant we use is often referred to as *interactive programming* or *interactive parametrisation* because the parametric program is being generated on-line while the design session is active [3, 7].

Most of the statements of the parametric program are of the type:

```
%Object: Command(Parameter1,Parameter2,...)
```

where the Command is the modelling operation that the user invokes to construct the graphical Object after all necessary parameters (numerical, graphical, structural) have been interactively supplied.

The explicit model of such a system is the boundary representation of the geometry of the current instance – *the example.* Thus, in order to alter the model, new explicit geometry must be created by executing the parametric program with other values for its parameters. What is not easy to do in a programming-by-example system, is to modify the parametric object, i.e. the parametric program itself, rather than an instance that is produced by it.

This work presents a method to overcome this disadvantage by structuring the parametric programs in the form of features. Features are designed as parametric subprograms and are embedded into the model hierarchy through program calls. In this way, it is possible to identify and substitute them. This approach allows the modification of the feature attachments in the parametric program using graphical tools.

Parametric Operations and Features

A *parametric operation* is an operation that modifies existing parametric or explicit objects and is controlled by some additional *parameters.* Therefore, unlike the parametric object, a parametric operation can create a separate object, but can as well generate or modify a part of another object. A similar classification is often made for features grouping them either into *template* and *non-template* [8], or into generative, modification and datum ones [1].

Most CAD procedures and modelling operations are to be considered as parametric operations. That is why if a new parametric operation is designed, it can be added to the set of CAD tools available and treated as a standard CAD procedure. This has been used to develop *customised interfaces* preserving the same CAD kernel.

Feature Representation

We represent features as parametric operations, i.e. parametric programs, whose input parameters are not only numerical but graphical as well. The result of such a parametric program execution after a call made within a higher level program is equivalent to a feature attachment. This type of embedded parametric objects are called *occurrences.* The occurrence in this case is the feature which creates one or more objects and/or modifies some of the objects passed as input parameters.

The following parametric program defines a simple rotor slot as a feature. It becomes a new CAD command, and when invoked, attaches the feature to the basic object (Fig.1).

```
Parametric RoundRotorSlot(%Obj,(Xc,Yc),Opening,Depth,Radius)
    %Circle1: Circle_by_Cent_Rad((Xc,Yc),Radius)
    Shift_by_Vector(%Circle1,(Depth-Radius)*Normal(%Obj))
    %Line1: Line_by_2Points((Xc,Yc),%Circle.Centre)
    Shift_Parallel(%Line1,+Opening/2)
    %Line2: Shift_Parallel(%Line1,-Opening)
    Trim(%Line1,%Line1.Start,Intersection_of(%Line1,%Obj))
    Trim(%Line2,%Line2.Start,Intersection_of(%Line2,%Obj))
    Trim(%Line1,%Line1.End,Intersection_of(%Line1,%Circle1))
    Trim(%Line2,%Line2.End,Intersection_of(%Line2,%Circle1))
    Delete_Part(%Circle1,%Line1.End,%Line2.End,(Xc,Yc))
    Delete_Part(%Obj,%Line1.Start,%Line2.Start,(Xc,Yc))
    : Join_In_Polyline(%Obj,%Line1,%Circle1,%Line2)
end
```

This basic object – the rotor – is in turn comprised of a variable number of slots repeatedly attached in a circular manner. This repetitive attachment is another parametric operation for which the feature `RoundRotorSlot` is a variable item (an input parameter). Some slot parameters are typical for this particular type, and so they are designed as global ones. This allows the use of varying parameter sets without changing the intermediate levels in the hierarchy.

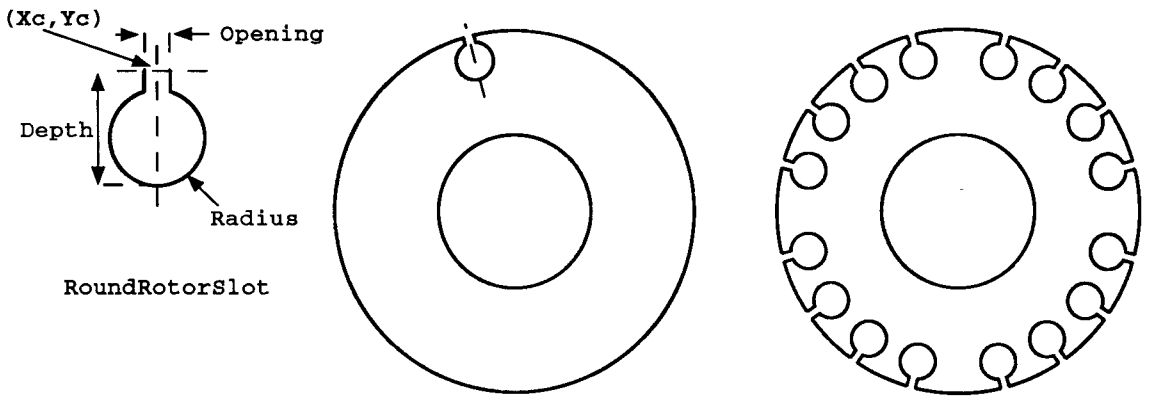

Figure 1: A Rotor with an attached feature

Feature attachment is structured by supporting *generic operations* - operations that are performed on arbitrary feature (or at least on a wide class of features) with the feature name given as one of its parameters. Such operations are suitable for representing *non-template* features. The bottom-up approach is used in this step-wise definition. Only a single feature-example is needed when generating the next level in the hierarchy.

The following fragment presents the simple case in which a rotor is comprised of `Nslots` equally distributed slots along its outer boundary.

```
Parametric RepeatCircular((Xc,Yc),Nslots,%Obj,EmbedFeature)
    %Pnt: Point(Xc,Yc+%Obj.Radius)
    repeat Nslots times
       %Obj: EmbedFeature(%Obj,%Pnt)
       Rotate(%Pnt,(Xc,Yc),360/Nslots)
    enddo
end

Parametric Rotor(OuterRadius,ShaftRadius,Nslots,Opening,Depth,Radius)
    %C1: Circle_by_Cent_Rad((0,0),OuterRadius)
    %C2: Circle_by_Cent_Rad((0,0),ShaftRadius)
    export (Opening,Depth,Radius)
    RepeatCircular((0,0),Nslots,%C1,RoundRotorSlot)
    : Material_Region(%C1,%C2,IRON)
end
```

A more complicated example is the one presented in Fig.1, where not all slots are necessarily equidistant. Instead, a `N1` number of slots must be grouped together in a feature (a separate operation), and this feature must be repeatedly attached to the circumference of the object a `N2` number of times.

In all cases the attached features form a rigid hierarchical structure, which is built up according to the sequence of design actions. This is useful for feature identification and modification, but makes model restructuring extremely difficult.

Feature Attachment

The representation of a feature is closely related to the explicit geometry representation used in the system and its final goal. Using *volumetric features* is mostly justifiable for systems that possess CSG

models and/or are aiming at producing machinable (manufacturing) output [6].

The main reason to use boundary representation is in the CAE and simulation tools such as: mesh generation and boundary conditions definition, that are related and deal with boundaries. Another reason is the need to maintain a unique model with features such as cuts, fillets, blending surfaces, etc. that are easier to implement as face operations.

When parametrisation is done using explicit boundary representation, it is possible that after a feature attachment, this feature might not be directly identified as it becomes an integral part of a boundary object. This is illustrated in the example given in Fig.1. In that example, the object `%C1` which is the outer boundary of the rotor will be modified while attaching each slot.

That is why, in order to be able to perform feature identification when the feature is attached to an object, each feature is comprised of two parts:

- The parametric procedure that instantiates the feature (by modifying the existing boundary of the objects) and
- An auxiliary element of a special type used for feature identification which for example can be one of:
  - the part of the object being created or modified: e.g. fillet arc, chamfer segment;
  - the bounding box of the objects created by the program.

The auxiliary element is used to link the instance of the feature in the explicit geometry database with the parametric program that instantiates it. It serves as an input parameter of the feature modification and deletion operation as this is described further.

Apart from being used to identify features, this auxiliary element can be visualised separately from the actual boundaries when a feature sketch of the object is requested by the user. This is also useful for program visualisation in programing-by-example parametrisation.

Operations on the Feature Model

There are three main operations that are related to features: insertion, deletion and modification. These operations are performed on the generated parametric program - the feature model. The first one is typical for standard programming-by-example systems. The only difference is the creation of the auxiliary element to be used later in the other two. The deletion and modification are done by changing the generated parametric program using graphical identification on the example. If performed during the example creation, these operations themselves are not stored as a part of the program as they concern its modification.

- **The insertion of a feature**: This is a basic operation that is realised by embedding a parametric operation in the parametric program (which is actually the feature model). Due to the procedural manner of generating the feature model, the feature occurrences are appended at the end of the parametrised description. This is quite natural as a feature attachment may use as referential data all that geometric elements that the example already contains.

- **The deletion of a feature**: This is equivalent to omitting an invocation of a parametric operation. It is only possible if the structure of the model can be preserved in case the attached feature is missing. This is another reason why we start attaching features to an object with a complete boundary representation according to the simulation requirements. If an unique attribute of the feature is referred to, the feature may not be deleted.

- **The modification of a feature**: The invocation of the parametric program is changed by entering other parameter values, or even changing the feature itself – the name of the parametric operation that is performed. The example given above presents one of the problems in feature modification. It is possible to have a variable feature (i.e. a feature being the input parameter of another feature) whose modification must concern the actual parameter as in the call to `RepeatCircular`.

- **The expansion of a pattern feature**: This is a specific modification concerning only pattern features - ones that attach an arbitrary number of the same feature at equispaced locations. From a program manipulation point of view, this substitutes a `repeat` loop with a fixed number of occurrences, so that each can be accessed and modified separately.

The Extension of the Parametric System

The library of hierarchically structured typical Electrical Machinery features has been created using a bottom-up example-based approach. Some of the most complicated modifications, as well as the ones that handle complementary features have been written manually as parametric programs. The power of the programming-by-example system is mainly used for the incremental structuring of the features in a parametric representation (a feature model) in a graphical way.

Two extra facilities have been implemented in the parametric system to ensure the possibility to modify on-line the parametric representation of an object.

- **The feature regenerator**: This module interprets the parametric program, which is invoked by the system not only for feature instantiation, but also when changes to the construction procedure are made. This is normally the case when a feature is deleted or modified. After the program is re-run, the current example represents the last modification being done. It is responsible for the corrections made to the feature model by: a/deleting an occurrence; b/changing the parameters of an occurrence.

- **The feature identifier**: This module is closely related to the object manager of the CAD system and is needed to identify a feature, extract its name (which is the name of the parametric operation that creates it) and its parameters. It uses the auxiliary element, created by each feature instantiation by searching only through the elements of that specific type.

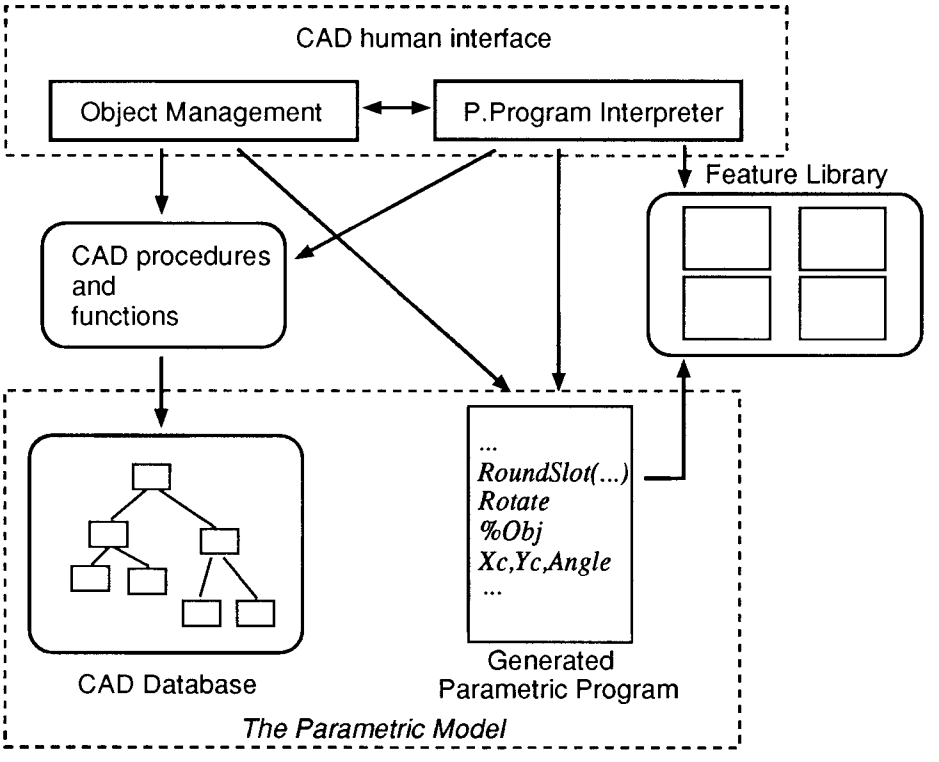

Figure 2: The Structure of the Extended Programming-by-example System

The identification has been made an integral part of the object management, which is isolated from the CAD engine and practically translates the user interaction with the CAD explicit model to variable objects, created by the program execution. This module distinguishes between variable and fixed data entries.

The interpreter is invoked each time a feature is attached, or regeneration is required – whenever the generated parametric program is interactively modified. Both facilities have been implemented as an integral part of the parametrisation tools (object management, program generation, program interpretation) and not as separate modules.

Features in a CAE system

The form features are not only a tool for the geometric modelling of an object. Information about the form is used for simulation such as the finite element method, because it is used in the mesh generation and material characterisation.

The use of parametric operations for representing features is also justified by the fact that a feature attachment might cause modification to the *interfaces of material regions*, and/or the outer boundary of the object where some *boundary conditions* are normally imposed. This is additional data used for simulation and analysis that a feature designer must be aware of.

Preserving Material Consistency

The example in Fig.1 shows a simple operation of attaching a feature to an object, that results in the modification of just one boundary contour. This is the outer boundary of the object `Rotor`, which is passed to the feature instantiation procedure as an input parameter.

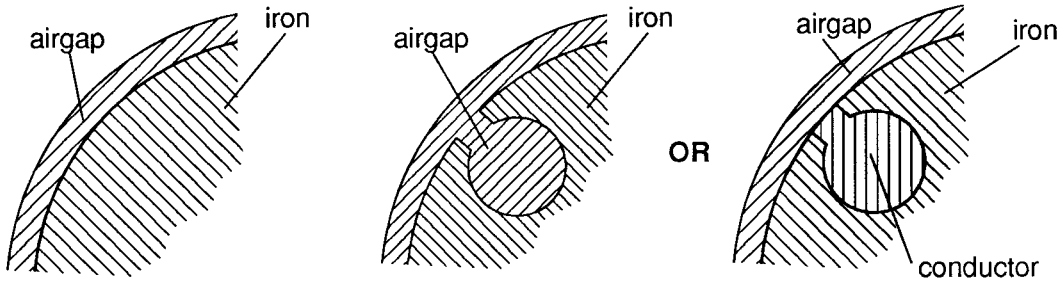

Figure 3: Possible material region modification by feature attachment

In a more realistic case, the rotor is embedded in a rotating machine and its outer boundary is the interface between two material regions: the iron of the rotor, and the air in the *airgap*. Thus, the placing of a slot feature must modify not only the rotor outer boundary, but the airgap inner one as well. In another case, the slot should create a new region *conductor*, which needs to be places inbetween the existing two.

In this manner, the partitioning of the whole model in regions, ready to be processed by CAE tools (e.g. Mesh Generation) is maintained consistent at all stages of design. As such a partitioning is consistent before the attachment of the feature, its deletion will still leave a ready-to-be-processed engineering model.

A similar problem exists when the outermost boundary of the model is modified. There, the boundary conditions need to be modified. Thus, a slot on the outside of the stator is implemented as a separate feature from a rotor or an inner stator slot.

Describing Complementarity

A serious problem that has not been addressed extensively in feature-based modelling is the representation of different views on the same parametric model. Most solutions are limited to supporting one set of features and relying on feature recognition tools for generating a different view on the same geometry. This concerns primarily volumetric features used in CSG-trees which are generally not unique but rather complementary.

Electric Machinery features (as well as many others volumetric features) are inherently complementary and need to be handled simultaneously in the same design session. For example, a rotor

can be constructed by attaching slots, but these slots form teeth, which will later be referred to for assigning specific physical properties. For example, defining a tooth excitation must influence the material characteristic of the adjacent slots – the conductors. Conversely, a rotor, built up from teeth will need the slot description for machining purposes.

The procedural representation we use limits the dual definition to a strict hierarchical structure, that depends on the type of features the model is designed by. We use the following scheme to describe complementarity. The parametric program that generates the slot, generates an additional feature Tooth as the boundary between the centres of the current slot and the adjacent counterclockwise one. This feature is an object, that can be referred to in graphonumeric expressions and thus can be attached additional information to. The change of a slot parameter and the re-execution of the parametric program will generate also new instances of the teeth. However, it is not possible to remove or modify such a feature.

Discussion

The above limitation is difficult to overcome using purely procedural models. A dual structure can only be equally designed and modified if the model is relational rather than hierarchical. Though being useful in the manipulation of parametrised descriptions, the method described forces the user to be involved in how the structure of the object designed is represented.

Modification to features of the parametric object are limited, and presently only the identification is done graphically. It is possible to translate graphical manipulations on the identification object to changes of location parameters. Substituting a feature by changing the name of an attached feature might lead to unexpected problems, that need to be resolved by altering the feature definition.

Acknowledgement

This research was carried out in the frame of the Inter-university Attraction Poles for fundamental research funded by the Belgian State.

References

[1] CHEN, X., AND HOFFMANN, C. M. Towards feature attachment. Tech. Rep. CSD-TR-94-010, Department of Computer Science, Purdue University, 1994.

[2] GARDAN, Y. A system for the interactive description of parametrized elements. In *Proceedings of 5th IFIP/IFAC Conference* (1983).

[3] GIRARD, P., AND PIERRA, G. Command recording versus parametric and variational systems, and old/new third way of parametrizing CAD models by end users. In *Proceedings of COMPEURO'93 (IEEE-SEE)* (1993), pp. 194–200.

[4] GIRARD, P., PIERRA, G., AND GUITET, L. End user programming environments: Interactive programming-on-example in CAD parametric design. In *Proceedings of EUROGRAPHICS'90* (1990), Elsevier (North-Holland), pp. 261–274.

[5] HOFFMANN, C. M. On the semantics of generative geometry representations. In *Proceedings of 19th ASME Design Automation Conference* (1993).

[6] KLEEMAN, M. W. Feature based parametric solid modeling for improving mechanical engineering design. In *Proceedings of Autofact'89, Detroit, MI, USA* (1989).

[7] LOUKIPOUDIS, E. N. Interactive parametrization and its application in mechanical design. In *Proceedings of CAD&CG'89, Beijing, China* (1989), Pergamon Press, Oxford.

[8] REQUICHA, A. Research in feature recognition. In *Feature Based Modelling in the Product Development, Darmstadt, Germany* (1994).

[9] SHAH, J., AND ROGERS, M. Functional requirements and conceptual design of feature-based modeling system. *Computer-Aided Engineering Journal* (1988), 9–15.