PERSISTENT NAMING FOR PARAMETRIC MODELS

Dago AGBODAN, David MARCHEIX and Guy PIERRA

Laboratory of Applied Computer Science (LISI)
National School of Engineers in Mechanics and Aeronautics (ENSMA)
Téléport 2 — 1 avenue Clément Ader — BP 40109
86961 Futuroscope Chasseneuil cedex
FRANCE
agbodan, marcheix, pierra@ensma.fr http://www.lisi.ensma.fr/

ABSTRACT

Nowadays, many commercial CAD systems support history-based, constraint-based and feature-based modelling. The use of these new capabilities raises the issue of persistent naming which refers to the problem of identifying entities in an initial parametric model and matching them in the re-evaluated model. The goal of this paper in to propose a naming mechanism and an hierarchical structure enabling to identify topological entities and to apprehend the "design intent".

Keywords : CAD/CAM, geometric modelling, parametrics, feature taxonomy.

1. INTRODUCTION

Static solid modelling systems (B-rep, CSG, etc.) largely used in the Computer Aided Design (CAD) area are more and more replaced by dynamic modelling systems (known as history-based, constraint-based and feature-based modellers) which allow both to express and to record conceptual designs and "design intents". These dynamic modelling systems are often gathered under the term of parametric modelling systems. A parametric model is composed of a representation of an object, of a set of parameters (characterising the object) and of a list of constraints (equations or functions) applied to the object. By extension, a parametric modeller is a system for geometric design which preserves not only the explicit geometry of the designed object (called "*parametric object*" or *"current instance"*), but also the set of constructive gestures used to design it (called *"design process"* or *"parametric specification"*).

This two-fold data structure enables rapid modifying by re-evaluation. But when re-evaluation leads to topological modifications, references (between entities) used in the constructive gestures are difficult to match in the new context, giving results different from those expected. A persistent naming system, robust regarding some topological modification, proves useful to preserve, from a re-evaluation to another, references between topological entities. It is the problem known as *"persistent naming"* or *"topological naming"* [Kripac95], [Capoy96]. The persistent naming mechanism should enable, on the one hand, unambiguous identification of geometric and topological entities of the parametric model (1^{st} issue) in order "to find" them in the re-evaluated model (2^{nd} issue) and on the other hand should enable to represent the "design intent" or rather to represent different semantics that might be expressed by the designer (3^{rd} issue).

This paper mainly focuses on the 1^{st} and the 3^{rd} issue. It is structured as follows. In section two, we give a detailed account of the major issues about parametric modelling. The third section discusses some pre-existing works, essentially two of the main works about topological naming. Each of these works only partially addresses 1^{st} and 2^{nd} issue. None addresses the 3^{rd} one. We introduce, in section four, an alternative approach. In section five, we propose a new feature taxonomy, suited to our approach. Finally, in section six and seven we precisely describe our naming mechanism as well as the structure enabling to handle such a mechanism.

2. MAJOR ISSUES

The main problem for parametric re-evaluation is to characterise geometric and topological entities of a parametric model. Characterising entities consists in giving them a name at design time and "finding them" again at re-evaluation time (i.e. matching entities of the initial model and entities of the re-evaluated model.) Let us take the example of Figure 1 to illustrate this problem.

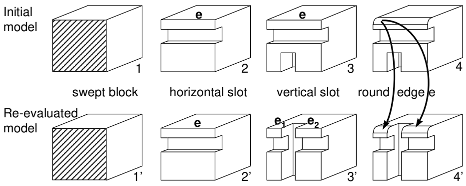

Figure 1 : Naming and matching problems.

In the above example the initial model is designed by means of a parametric specification containing four successive constructive gestures. The fourth one consists of rounding edge "e". If the initial model is saved after this fourth step, the current instance no longer contains edge "e" : it was removed by the rounding function. Thus the rounding function which has edge "e" as input parameter cannot any longer be represented in the parametric specification part of the model. Therefore "names" are needed to represent the entities referenced in the parametric specification whether or not they exist in the current instance.

Moreover each constructive gesture creates a number of entities which have to be distinguished and therefore named, to be referenced by further constructive gestures, even if the same number of entities exist in all possible re-evaluation (no topological change). The problem is even more complex for parametric models, of which the entities and the number of entities change from one evaluation to another. Let us return to the above example, but this time in the re-evaluated model. We notice that, at step 3', the edge "e" has been split into edges "e_1" and "e_2". Thus at step 4' the problem is to determine which edge(s) has(ve) to be rounded. The problem is to identify, i.e. to match, edge "e" with edges "e_1" and "e_2" despite topology changes. Thus, when re-evaluation leads to topology changes a new issue is to match two different structures.

It is thus necessary to have, in addition to the naming mechanism (1st issue), a robust matching mechanism (2nd issue) regarding re-evaluation.

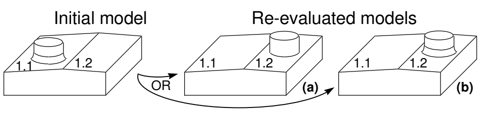

Figure 2 : Different semantics.

The third identified problem is to be able to express different semantics that capture the "design intent" by using high level abstractions (aggregates of geometric/topological entities). For example, the designer might want to express that a feature is applied on the global shell rather than on a particular face. Figure 2 illustrates this problem. The object is designed in three steps : creation of the block by extruding a polygonal contour, creation of the cylinder on the block, then rounding the edge between the block and the cylinder. According to whether the rounding function was expressed between the cylinder and face 1.1 (case 1) or between the cylinder and the upper shell composed of faces 1.1 and 1.2 (case 2), the re-evaluated model is different because the "design intent" is different. In the first case one obtains model (a) where the round disappeared since it cannot be made any more between the cylinder and face 1.1. In the second case one obtains model (b) where the round always exists since it could be made between the cylinder and the upper shell. To support these two different semantics, the naming mechanism should provide for naming and matching high level entities such as shells.

3. RELATED WORK

Following the pioneer work of Hoffmann and Juan [Hoffm93], over the last few years several authors have analysed the internal structure of parametric data models, proposing some editable representations [Hoffm93], [Pierr94], [Solan94], [Pierr96], [Laakk96], discussing their underlying mathematical structures [Pierr94], [Ragho98], describing the problems, either of the semantic of modelling operations [Hoffm93], [Chen94], [Agbod99] or of constraint management [Bouma93]. Most of them discussed parametric modelling in terms of creation (but not re-evaluation). Several naming scheme and persistent naming mechanisms have also been proposed. In particular Kripac [Kripa94] and Chen [Chen95] proposed solutions to address some of the problems mentioned in section 2. The first essentially developed a matching algorithm whereas the second focused rather on the unambiguous entity naming.

3.1 Kripac

Kripac [Kripa94] focuses on the name matching. He proposes an API (Application Programming Interface) encapsulating its topological identification system and guaranteeing the persistence of the names using a table of correspondence between an entity of the initial model and one or more entities of the re-evaluated model. He proposes an interesting structure for identification of any topological entities based on face history (creations, splits, merges and deletions of faces) and a complex name matching algorithm. Kripac's Topological ID System consists of 3 parts. First a face graph allowing both a naming of any faces and a name matching during re-evaluation. Second a table recording names for the only entities that are referenced in the parametric specification (edges and vertices are named in terms of their adjacent faces). This table also contains pointers to the geometry (current instance) more some information for the matching algorithm. Third the geometry of the designed object. During each re-evaluation all the faces, as well as every referenced entity in the parametric specification, are matched with the new ones.

Split faces are named in terms of all adjacent faces, but Kripac does not explain how the initial faces are named. Moreover, during re-evaluation, the proposed matching algorithm establishes some priority rules, independent of the designer's action, making the result of the re-evaluation unpredictable. Another limitation is that in its approach, Kripac preserves a copy of the geometric models at each step of the construction process. This speeds up the re-evaluation but it would require a memory space which is not compatible with the size of the real models used in CAD. Beside the limitation concerning the persistent naming problem, Kripac's model does not deal with the semantic representation problem stated as 3rd issue in section 2.

3.2 Chen

Chen proposes a model [Chen95] which is composed of two representations. For the first one, he uses an editable representation, called Erep [Hoffm93], which is an unevaluated, high-level, generative, textual representation, independent of any underlying core modeller. It abstracts the design operations, contains the parametric specification and stores all entities by name. The second representation, evaluated and modeller dependent, contains the geometry (the current instance). The link between these two representations is obtained by a name schema which establishes the link between the geometric entities of the geometric model and the generic names (persistent) of the unevaluated model.

Chen defines a precise structure for naming invariant entities[1], particularly, for sweep operations. Every entity in a sweep is named by reference to the corresponding source entity of the swept 2D contour and the constructive gesture. He also proposes an identification technique for contingent entities[2] based on topological adjacencies and feature orientation. In the Chen approach every contingent entity is named. Unambiguous names are generated by composition of topological adjacencies. Unfortunately, Chen has mainly studied the naming problem at the construction stage of the parametric model, and has almost not studied the re-evaluation stage. The matching mechanism is not fully defined.

4. OVERVIEW OF OUR APPROACH

Our approach is similar to Kripac's one. The main differences are that we use a shell graph as basis of the persistent naming and that we use an hierarchical aggregation structure to capture, at various levels, the "design intent". Our approach deals with the three issues stated in section 2. We propose :

- A naming mechanism that consists of two parts. First an oriented shell graph (see section 6). Like the Kripac's face graph, our shell graph provides for tracing shell evolution. Following [Agbod99] we distinguish invariant and contingent shells and we propose to identify, unambiguously and uniquely, both the invariant faces and shells (see section 7.1), then the contingent faces and shells (see section 7.2). Second, a table where edges, paths and vertices referenced in the parametric specification are named in terms of their adjacent faces or shells (1st issue)
- A matching mechanism, outlined in section 7.2, where nodes of the shell graph are matched from the initial design shell graph onto the current re-evaluation shell graph. Other topological entities are matched by reference to the shell graph.
- A definition, for each kind of feature, of the various semantics that might be expressed by a designer. This is done through a taxonomy (see section 5) where each kind of feature is associated with a pre-defined hierarchical aggregation structure (3rd issue).

5. A FEATURE TAXONOMY

In order, to identify invariant shells in any form feature, we need a feature taxonomy. Indeed, these invariant shells do not change, whatever be the instanciation or whatever be the re-evaluation of the feature. They will constitute entry nodes of the shell graph. Most of current feature taxonomies propose to interpret form features from a machining point of view or from a specific application area point of view. These taxonomies do not focus on structural invariants and are thus not suited to our problem. We propose a feature taxonomy based on their invariant structure. By invariant structure, we mean both the initial intrinsic topological structure of feature (before interaction with the object's geometry) and the topological structure resulting from the interaction with this existing geometry.

For example, we consider as invariant the fact that systematically the final face of a *through_hole* is deleted when the feature (the *through_hole*) interact with the geometry of the object (i.e. when the hole is

[1] - An invariant entity is a geometric or topological entity which can be, completely and unambiguously, characterised by the structure of a constructive gesture and its input parameters, independently of involved values [Agbod99].

[2] - A contingent entity a geometric or topological entity that results from an interaction between the pre-existing geometric model and invariant entities resulting from a particular constructive gesture [Agbod99].

made in the object). The invariant structure depends on how the feature is designed. For example, for extrusion and revolution features invariants are the initial shell (ci), the lateral shell (cl) and the final shell (cf). The lateral shell (depending on the particular topology of the extruded contour) can be structured in sub-shells (cl_1 and cl_2) which themselves can be structured. For example, in right, left and bottom shells (cl_d, cl_g, cl_{fd}) for a *crossing_flat_slot*. The *crossing_flat_slot* example represented below in Figure 3, illustrates well the invariant structure of a feature. Note that some entities (ci, cl_1, cf) disappear systematically.

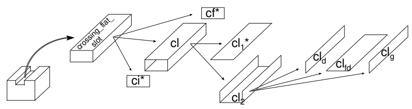

Figure 3 : Invariant feature structure example : *crossing_flat_slot*.

Our feature taxonomy (see Figure 4) is based on the feature taxonomy proposed by Shah and Mäntylä [Shah95] and the classification proposed in the STEP standard [Iso94]. We classified these features according to their structural invariants which we have extract for each one of them.

- **Primitive feature**s. Such feature are predefined features. They are characterised by their global shell and their faces.
- **Transition feature**s. Such features are features joining several shells such as *chamfer* or *rounding*. They are characterised by their global shell.
- **Basic volume feature**s. These features result from an extrusion or a revolution. They are characterised by their initial, lateral and final shells. They are classified according to the result of their interaction with the geometry. For example a *blind_hole*, obtained by extruding a circle, is characterised by these three shells (ci, cl, cf) and by the fact that the initial shell disappears from the geometry.
- **Basic surfaces feature**s. They are all the surfaces obtained by sweeping a loop. They are characterised by their initial and final loop and their lateral shell.

To these four basic classes one may add two other classes. **Container feature**s gathers some particular feature aggregations : specific aggregates (*blind_counter_sunk_hole*, ...), each one associated with a particular invariant shells structure, repetitive aggregates (*pattern*) and unstructured aggregates (*assembly*) that consist only of basic faces without invariant shells structure. **Texture feature**s (e.g. *knurl*, *threading*, ...) do not introduce new topological entities and are not relevant for our purpose.

The terminology used in this taxonomy, in particular names of the features, has been borrowed both from Shah and Mäntylä [Shah95] and from the STEP standard [Iso94].

6. SHELL GRAPH

Shells are defined as shell aggregates. On the lowest level of the hierarchy each shell represents a face. The shells are :

- **Hierarchical**. They are structured in shells, sub-shells, etc., in order to be able to apprehend the geometry at various levels of granularity, and thus to express different semantics. This structure is defined for each class of feature (a global shell and invariant sub-shells resulting from our feature taxonomy.
- **Connected**. Each shell is composed of jointed sub-shells. The objective of the shell graph is to represent the history (split and deletion, merge is currently not allowed) of the invariant initial structure (itself connected). Thus, a new graph node represents a new connected part which appears during a split and enables to identify it and to trace it.
- **Overlapping**. A sub-shell can be part of several shells. The overlapping hierarchical structure enables, during construction, the creation of any (connected) shell aggregate. This presents both a flexible and an extremely powerful designation mechanism. Especially for referencing aggregates stemming from the union of a feature with some parts of the object (see section 6.2 for details and see shell C3 in

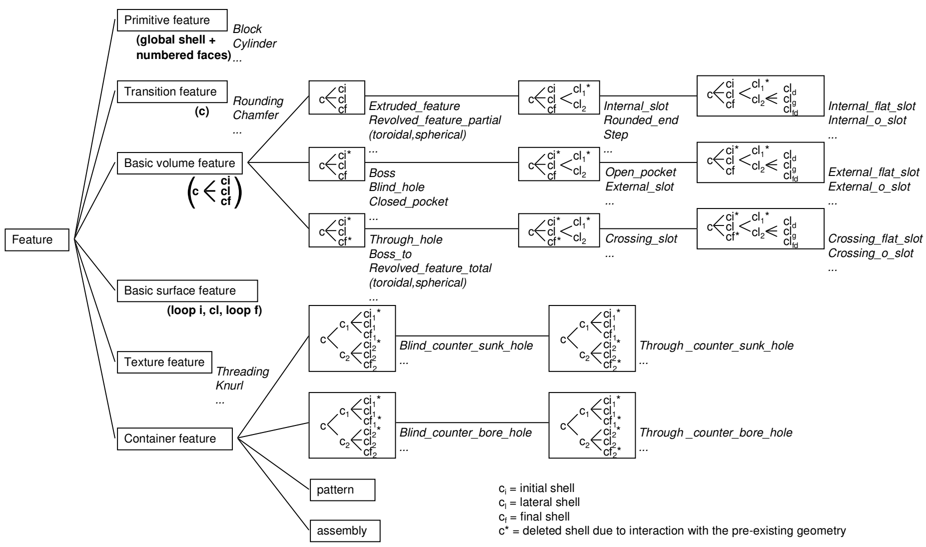

Figure 4 : Feature taxonomy (according to their invariant structure).

Figure 5 for an example).

6.1 Interest of the shell graph

There are two interests in using shells and a shell graph : entity aggregation and entity tracability (with the aim of matching).

- **Aggregation** (shell interest). The goal is, for the designer, to be able to reference geometry at various levels of granularity. Faces are low level entities, having often few meaning for the designer. Shells make it possible reference meaningful high level abstraction. For example, in an extrusion there is a single lateral shell which is composed of several lateral faces. The designer may reference this shell for example for matching a fillet.
- **Tracability** (shell graph interest). The goal is to be able to follow the shell evolution in order to be able, during model design, to identify the involved shells, then, during re-evaluation, to identify the effective shells (in the current instance) corresponding to the referenced shell. In addition to the possibility of referencing each shell represented in the graph, the shell graph last allows another level of aggregation. The aggregation of all the shells having had a common ancestor. Thus, for example, the abstraction of lateral shell 2.l in Figure 5 will exist even in case of split of this lateral shell since at each level of aggregation the history is preserved.

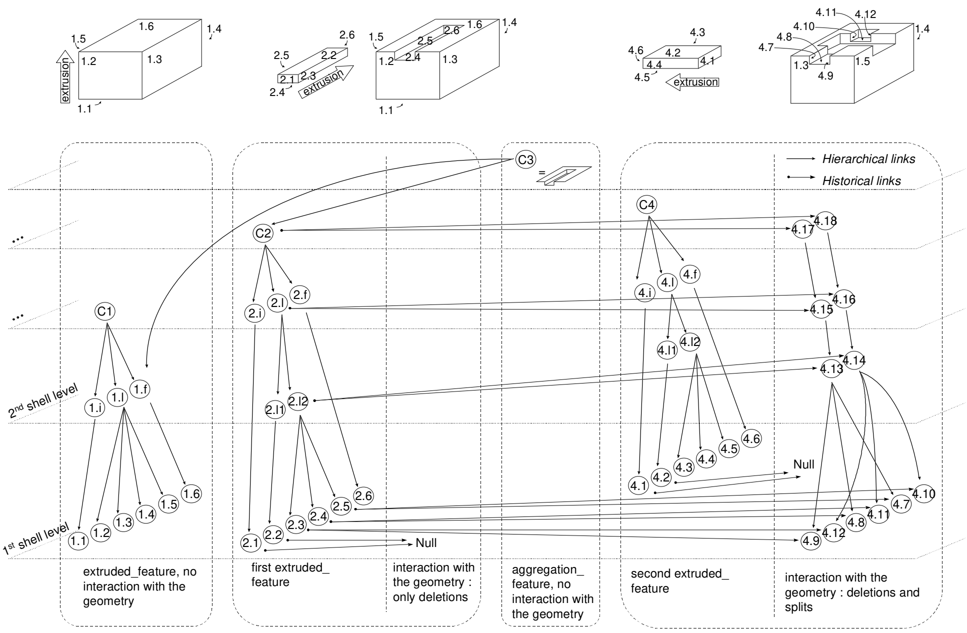

Figure 5 : Shell graph example.

6.2 Graph structure

The graph stores the feature structure (aggregation) and the shell history (tracability). Each constructive gesture can be broken up into two steps. The first step is the specification of the isolated feature. It corresponds to the invariant structure (defined in our feature taxonomy). This invariant feature structure is represented by hierarchical links between shells. It represents the entry nodes of the shell graph. The second step is the interaction with the pre-existing geometry which produces contingent entities. Those entities result from the changes in the initial hierarchical structure and the pre-existing geometry. The hierarchical structure is always represented by hierarchical links. The shell evolutions are described by historical links. Each node of the graph is identified by two elements. The first one is the construction step number and the second one is an identifier described in section 7.

6.2.1 Hierarchical links

The feature structure (described in section 5) is represented in the graph by the hierarchical links. Beside the representation of the feature's invariant structure, this hierarchical links enables to represent new aggregates that might appear during the design process. Figure 5 gives an example. An *internal_slot* is applied on the final shell (1.f) of an extruded bloc (*extruded_feature*). A shell C3 can be generated (for example on explicit request of the designer which wishes to handle this aggregate) in order to be able to name and thus to address the aggregate corresponding to the union of the slot (C2) and the shell carrying this slot (1.f). C3 thus has two hierarchical links to 1.f and to C2. Moreover, we can notice that this example illustrates the case of shell overlapping presented in section 6 since the shell 1.f belongs now to two distinct shells C1 and C3.

The hierarchical structure evolves with the modifications of the shells. Thus, in the graph example, illustrated in Figure 5, the remaining lateral shell (2.l2) which is composed of the shells 2.3, 2.4 and 2.5 is cut into two shells 4.13 and 4.14. These last, are in turn, composed of the result of the split of the shells 2.3, 2.4 and 2.5, so that, at each step, there exist a (modified) hierarchical structure.

A property of this graph, which significantly simplifies its management is that the hierarchical links need to be described only at the level of the historical leafs of the graph. To prove this property we study for two successive steps and for two successive hierarchical levels (here lowest level of shells –faces– and first level of aggregation, but can be extended to any successive hierarchical levels) all the possible combinations of entity histories. The history is reduced to split. Indeed merge is not allowed and entity deletion (in fact, pointers to NULL) does not change the historical links of the graph.

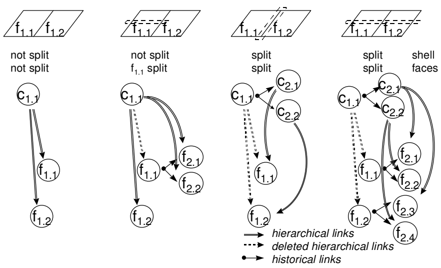

Figure 6 : Hierarchical links evolution.

It is possible to deduce from these four cases that the knowledge of the hierarchical structure at step *i* is sufficient to know the structure at step *i -1*. Indeed, it is sufficient to compare the construction step numbers (creation order) to know when a shell was created. For example, in the second graph of Figure 6, the face $f_{2.1}$ is a sub-shell of $c_{1.1}$. The face $f_{2.1}$ which was created after $c_{1.1}$ thus results from a shell ($f_{1.1}$) which was sub-shell of $c_{1.1}$. This reasoning, applied to each of the four cases, and extended to any step *i* and *i -1*, makes it possible to generalise the property. For example, in the (partial) graph represented in the table below, if the shell B is the Parent in the Hierarchy of shell D (PH_i (D) = B) at step *i*, it is possible to deduce from the creation order of D and B (n°(D) and n°(B)), which was the Parent in the Hierarchy of D (or C as the case may be) at step *i -1*.

Step i		Step i -1		
$PH_i(D)=B$	if	n°(D)<n°(B) and n°(B)=i	then	$PH_{i-1}(D)=A$
	if	n°(D)=n°(B) and n°(B)=i	then	$PH_{i-1}(C)=A$
	if	n°(D)>n°(B) and n°(D)=i	then	$PH_{i-1}(C)=B$

It is possible to find the hierarchical structure of step *i -1*, knowing the one of step *i*. A recurrence allows to conclude that it is possible to preserve the hierarchical links only at the level of the historical leafs. Note that capability to compute the state of the old graph, at any step, is essential to make the matching with the graph resulting from the re-evaluation.

6.2.2 Historical links

For each graph's node and in particular for each leafs of the hierarchical structure, we shall trace its evolution. There are 3 possibilities for this modification (merge is not allowed) :

- **Deletion**. The interaction between the geometry of feature and the geometry of the object on which it is applied, deletes a certain number of shells (of the feature and/or the object). These shells remain present in the graph, but without any counter part in the geometry. The historical links point to NULL.
- **Modification**. By modification, we mean any evolution which preserves the shells connections. We consider that such a modified shell is equal to the shell before modification. Indeed, if at step *i* a shell is perfectly identified, and if at the step *i +1* this shell is modified (except split), it remains perfectly identified in the graph. Thus, shell modifications are not represented in the graph
- **Split**. There is a split when a shell is cut into several disconnected shells. This split is represented in the graph by the historical links. Thus, a cut shell points on the resulting sub-shells and inversely.

Historical links enable to trace the topological entities evolution (split). The history is preserved at all levels of granularity (faces, shells, aggregates of shells, ...). This gives to the model an expression power higher than the one of Kripac's model. Indeed, when a shell is cut into several pieces during a constructive gesture, it is possible to have access not only to the shell (represented in the hierarchy) but also to each shell piece (represented in the history). Those historical links are mainly used for graph traversal during name matching.

6.3 Node structure

Each node represents a shell which exists or has existed in the model. All the shells without outgoing historical links exist in the geometry. Each node is composed of :

- A name composed of the construction step number and another identifier which characterises uniquely the shells (see section 7 below).
- Pointers to the predecessor and successor nodes representing the historical links.
- Pointers to the parent and the child nodes representing the hierarchical links.
- A list of adjacent faces for split shells.

7. ENTITY NAMING

The entity (vertices, edges, paths and shells) identification is done by reference to faces (lowest level shells, i.e. leafs of the hierarchical structure). It is thus necessary to be able to name these faces in a unique and deterministic way. Generally, the identification of an entity is based on unchanging elements which characterise it in a unique way. In a parametric model, what never changes is the construction process[3]. Therefore, face naming is done by means of the construction step number (creation order) and by means of another identifier which characterises each face in a unique way. The problem is to define this identifier which characterises them in a unique way within each construction step.

For each construction step, we consider that there are two phases. Firstly, the creation of the feature where all the feature's entities, i.e. its invariants, and in particular the lowest level shells must be named. Secondly, the feature positioning within the existing geometry. This interaction with the existing geometry leads to modification and deletion of existing shells and to creation of new (contingent) shells. These contingent shells must also be named. Therefore there are two types of naming to implement : one for invariant shells and another for contingent shells.

7.1 Invariant shells

For the invariant shells, two cases are to be distinguished. First, the lowest level shells that are faces (leafs of the hierarchical structure), and second higher level shells.

7.1.1 Lowest level invariant shells

Identification of the lowest level invariant shells (invariant faces), is done by an integer number whose calculus is based on topology. This naming is robust regarding feature re-evaluation. We do not allow modification of the topology of features. In particular, for swept and revolved feature, changes of the 2D contours topology are not allowed. Thus, this "topological" naming is robust regarding re-evaluation.

Our topological naming is as follows. We begin from a starting coedge[4] and we make a radial traversal around the edges, then a normal traversal following the edges of each face contour. The traversal which is unique (compared to the starting coedge) and which cover the whole object makes it possible to assign a number to each face (see Figure 7). The traversal is done as follows.

- Do a normal traversal of the path containing the coedge of the current face.
- For each coedge do a radial traversal of all adjacent faces.
- If these faces are not numbered, assign a number and enqueue them.
- At the end of the path traversal take the first enqueued face. Repeat the previous traversal, starting from the coedge shared by this face and the face with the smallest number.

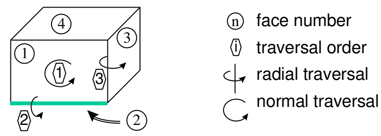

Figure 7 : Topological traversal.

The name of the lowest level invariant shells is thus composed of the construction step number and the face number resulting from the traversal.

7.1.2 Upper level invariant shells

Invariant shells, others than the lowest level ones, are aggregations of lower level shells. The invariant shells can be named by means of the list of shells which compose them, and by transitivity, by means of the list of the lowest level shells which compose them. These last are perfectly identified and the shell list is unique ; the shells are thus identified in a unique way by the list of the lowest level shells.

The name of upper level invariant shells is thus composed of the construction step number and the list of the names of the lowest level shells which compose the shell.

7.2 Contingent shells

The name of the contingent shells is composed of the construction step number and an iterative number. This number (arbitrary, but unique for each construction step) will be assigned again after re-evaluation during the matching (graph matching[5]). This number, insufficient to allow matching, is associated with the list of the adjacent faces to the shell. This list enables to distinguish two sub-shells resulting from the same shell. The list is composed of the lowest level shells (leafs of the hierarchical structure –faces–) which are adjacent to the split shell. This information will be used, during re-evaluation, for contingent shell matching.

Although the complete matching mechanism goes beyond the scope of this paper, we shall present its outlines. Its principle consists of :

♦ keeping the initial shell graph. The whole re-evaluation process is supported by this graph.

---

[3] We consider the modification of the construction process as a model edition and not as a model re-evaluation.

[4] The features are defined, in addition to their parameters, by a starting coedge. For example, in an extruded block, the first edge of the 2D contour defines the starting coedge.

[5] There are two stages during the matching :
(i) the graph matching, i.e. reconstructing the graph and matching the nodes.
(ii) the entity matching, i.e. determining the new entities involved in the parametric specification.

♦ Constructing during re-evaluation, a second shell graph, parallel to the initial shell graph. At each construction step, all the entries of the second graph are compared to the nodes (shells) of the initial graph.

- For each matched shell, the same iterative number will be assigned.
- For the shells which are not matched, a new iterative number (which do not exist in the initial graph) will be assigned

♦ Matching the entities involved in the parametric specification. For each entity a matching algorithm, similar to the one proposed by Kripac, comparing the second graph and the initial graph, is applied. Each old entity, which do not match exactly a new one, is replaced by one or several new entities based on comparison of "ancestors".

8. CONCLUSION

With the development of parametric modelling systems, the persistent naming problem becomes more and more important to enable topology change during model re-evaluation. Therefore, robust naming mechanism and matching mechanism have to be developed. In this paper, we have proposed :

- A naming mechanism (1st issue) where we distinguish invariant entities and contingent entities.
- An hierarchical structure which enables to express different semantics (3rd issue). In order to identify feature invariants, a new taxonomy based on the intrinsic and semantic structure of each feature was introduced.
- A shell graph recording this hierarchical structure and the evolution of the shells. The graph will enable us to match contingent shells during re-evaluation (2nd issue).

This approach offers three principal advantages. Firstly it enables to unambiguously identify topological entities of an initial parametric model. Secondly it enables to match entities between the initial model and the re-evaluated model, in spite of the multiple topological variations. Finally, such a mechanism enables the capture of different semantics that might be expressed by the designer.

The work presented in this paper is still under development. We are actually implementing our model in the Cas.Cade development environment, testing various matching solutions in case of large topological changes.

ACKNOWLEDGEMENTS

The authors are grateful to Pascal Lienhardt and Laurent Fuchs for several useful workshops and a number of insightful discussions.

REFERENCES

[Agbod99] Agbodan,D, Marcheix,D, Pierra,G : A Data Model Architecture For Parametrics in *Journal for Geometry and Graphics*, Vol.3, N°.1, pp.17-38,1999.

[Capoy96] Capoyleas,V Chen,X Hoffmann,C.M. : Generic naming in generative, constraint-based design in *Computer Aided-Design*, Vol.28 N°1 pp.17-26, 1996.

[Chen95] Chen,X : Representation, Evaluation and Editing of Feature-Based and Constraint-Based design, *Ph.D. thesis*, Department of Computer Sciences, Purdue University, West Lafayette, Indiana, 1995.

[Hoffm93] Hoffmann,C.M., Juan,R : EREP : an editable high-level representation for geometric design and analysis in *Technical Report CER-92-24*, Department of Computer Sciences, Purdue University, West Lafayette, Indiana, 1993.

[Iso94] ISO FDIS 10303-224 :1999 : Industrial Automation Systems and Integration — Product Data Representation and Exchange — Part 224 : Application protocol : Mechanical product definition for process planing using machining features, ISO, Geneva, 1994.

[Kripa95] Kripac,J : A mechanism for persistently naming topological entities in history-based parametric solid models (Topological ID System) in *Proceedings of Solid Modeling '95*, Salt Lake City, Utha USA, pp.21-30, 1995.

[Laakk96] Laakko,T, Mäntylä,M : Incremental constraint modelling in a feature modelling system » in *Computer Graphics forum*, Vol.15, N°3, EUROGRAPHICS'96, Poitiers, France, pp.366-376, 1996.

[Pierr94] Pierra,G, Potier,J.C., Girard,P : The EBP system : Example Based Programming for parametric design, Workshop on Graphic and Modelling In *Science and Technology*, in : *Springer Verlag* Series, Coimbra, 27-28 June 1994.

[Pierr96] Pierra,G, Ait-Ameur,Y, Besnard,F, Girard,P, Potier,J.C. : A general framework for parametric product model within STEP and Part Library in *European Conference Product Data Technology*, London, 18-19 April 1996.

[Ragho98] Raghotama,S, Shapiro,V : Boundary Representation Deformation in Parametric Solid Modeling in *ACM Transactions on Graphics*, Vol.17, N°4, pp.259-286, October 1998.

[Schen94] Schenck,D, Wilson,P : Information Modelling The EXPRESS Way, Oxford University Press, 1994.

[Shah95] Shah,J.J., Mäntylä,M : Parametric and feature-based CAD/CAM : Concepts, Techniques, Applications, John Wiley and Sons Inc., July 1995.

[Solan94] Solano,L, Brunet,P : Constructive Constraint-based model for parametric CAD systems in *Computer-Aided Design*, Vol.26, N°8, pp.614-621, 1994.