# On Families of Objects and Their Semantics

Rafael Bidarra and Willem F. Bronsvoort
*Faculty of Information Technology and Systems*
*Delft University of Technology*
*Zuidplantsoen 4, NL-2628 BZ Delft, The Netherlands*
*email: (Bidarra/Bronsvoort)@cs.tudelft.nl*

## Abstract

*An approach is presented to define a family of objects in parametric modelling. It avoids an inherent problem of boundary re-evaluation in current parametric modelling systems. Even more important, it offers an effective way to define the properties, or semantics, of a family of objects. A family of objects is defined by a prototype feature model, including several types of constraints to precisely specify its semantics. A family membership test is used to check whether an instance corresponding to a particular set of parameter values belongs to a family. The underlying geometric representation is a non-manifold cellular model.*

**Keywords***:* parametric modelling, families of objects, semantics, feature modelling, cellular model

## 1. Introduction

In a parametric modelling system, once an object has been modelled, a user can easily create variants of that object by changing its parameters. This approach has a number of obvious advantages: it can be helpful in fine tuning the shape of an object, several objects that differ only in some details can be modelled with much less effort than with traditional solid modelling techniques, and libraries of standard parts can be built (Anderl and Mendgen 1995).

The set of possible variants of an object is called a *class* or a *family of objects*. So a family of objects represents a set of objects with "similar" shape. The user can create an *instance* from such a family by specifying values for the parameters that characterise the family. Constraints play an important role in the definition of parametric objects; these specify certain properties for the objects, for example that some parameter value has to be within some prescribed range. Each instance has to satisfy all constraints specified for the family.

Most current parametric modelling systems use a dual representation for an instance of a family of objects: on the one hand a model history and the set of values for the parameters, on the other hand a boundary representation of the resulting shape.

Shapiro and Vossler (1995) posed an interesting question about parametric modelling: what is a good definition for a parametric family of solids? Such a definition should determine which objects are a member of a family, and which objects are not. It has been shown that, in current parametric modelling systems, such a definition is hampered by certain peculiarities of boundary representations (Raghothama and Shapiro 1998). Even more important, specification mechanisms currently available for parametric objects, in particular the available constraint types, are insufficient to adequately maintain the design intent of a family in all its instances.

Therefore, in this paper it is argued that it would be better to use an alternative definition and representation for a parametric family of objects, in which the desirable properties for its instances are more precisely specified, and which avoids the pitfalls of the boundary representation. A family of objects here is defined by a semantic feature model (Bidarra and Bronsvoort 2000). Such a model contains features and constraints that explicitly define the semantics of the family of objects. A *family membership test* can be used to determine whether a given set of parameter values specifies a valid instance of the family. This test greatly benefits from the underlying non-manifold geometric representation, a cellular model.

In Section 2, the problems with families of objects in current parametric modelling systems will be illustrated. In Section 3, the new mechanism for defining families of objects will be introduced. In Section 4, the family membership test will be described. In Section 5, some results and conclusions will be given.

## 2. Families of objects in current parametric modelling systems

In most current parametric modelling systems, a user can create a model of an object, sometimes called a *prototype*, and then create variants of that object by specifying different parameter values. The parameters may be dimensions of basic shapes that constitute the object, or user-defined parameters that specify other dimensions of the combined shape. If the modelling system is a parametric feature modelling system, the basic shapes are feature shapes. The set of possible variants is designated a *family of objects*.

In the prototype, several constraints can occur, for example to align two faces of two basic shapes of the object, or to restrict the range of some parameter. The sequence of modelling operations used to create the prototype, including all its basic shapes and constraints, is called the *model history* of the object. This is in fact a procedural representation, used to re-compute the boundary representation of the object whenever parameter values are modified. Basically, this comes down to a stepwise boundary re-evaluation of all basic shapes in the model with their new parameter values.

Several researchers, for example Chen and Hoffmann (1995), Lequette (1997) and Raghothama and Shapiro (1998), have observed that unexpected, and even improper, models can result from such a boundary re-evaluation process in commercial modelling systems. Shapiro and Vossler (1995) related this problem to the question on how to define families of objects in parametric modelling. The main issue in the latter is to determine which objects belong to a family and which do not. It may happen that, even if all constraints defined for the prototype are satisfied for some set of parameter values, the corresponding instance of the object does not belong to the family of objects as intended by the user.

An example of getting an unexpected model is given in Figure 1, inspired by examples in the references given above. The model consists of a block, a protrusion and a slot. The protrusion has been attached to the block so that their coplanar top faces are merged into one face, $f_1$, see Figure 1.a. The through slot, which intersects both the block and the protrusion, has been attached to face $f_1$, causing it to be split into two faces, $f_{2a}$ and $f_{2b}$, see Figure 1.b. By modifying a positioning parameter, the protrusion is displaced downwards. When the model history is re-executed, after this parameter modification, the user will probably expect the model of Figure 1.c, but, in at least one current commercial system, he obtains the model of Figure 1.d instead. Although this model is a valid model, it is apparently not according to the user intent. It is therefore questionable whether it should be a member of the family of objects represented by the prototype model in Figure 1.b.

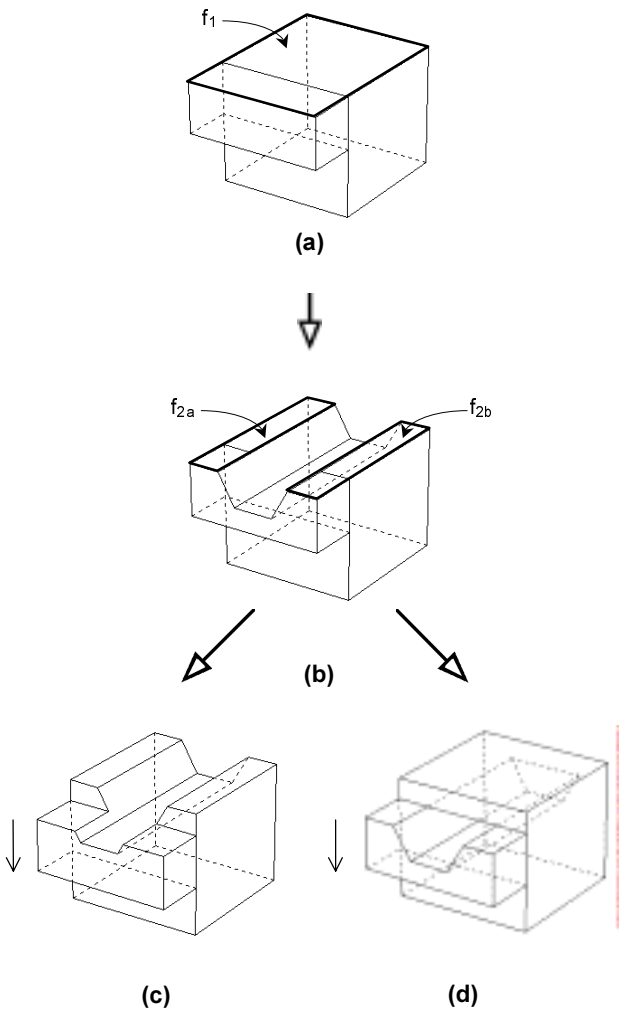



**Figure 1 – Obtaining an unexpected model after boundary re-evaluation**

However, boundary re-evaluation can even behave worse, as the resulting boundary model may become disconnected. An example of this, inspired by an example in Raghothama and Shapiro (1998), is given in Figure 2. The model in Figure 2.a has a cylindrical through slot, positioned at a distance *a* from the indicated edge, causing the top face of the block to be split into faces $f_1$ and $f_2$. A rib is then attached to face $f_2$, at a distance *b* of the slot edge indicated in Figure 2.b. The slot position is now modified, making the parameter value *a* equal to the radius of the slot, by which face $f_1$ in Figure 2.a collapses into an edge. According to the positioning constraints specified, the user would most likely expect the model in Figure 2.c as the result of this parameter modification. However, some current parametric modelling systems produce the model

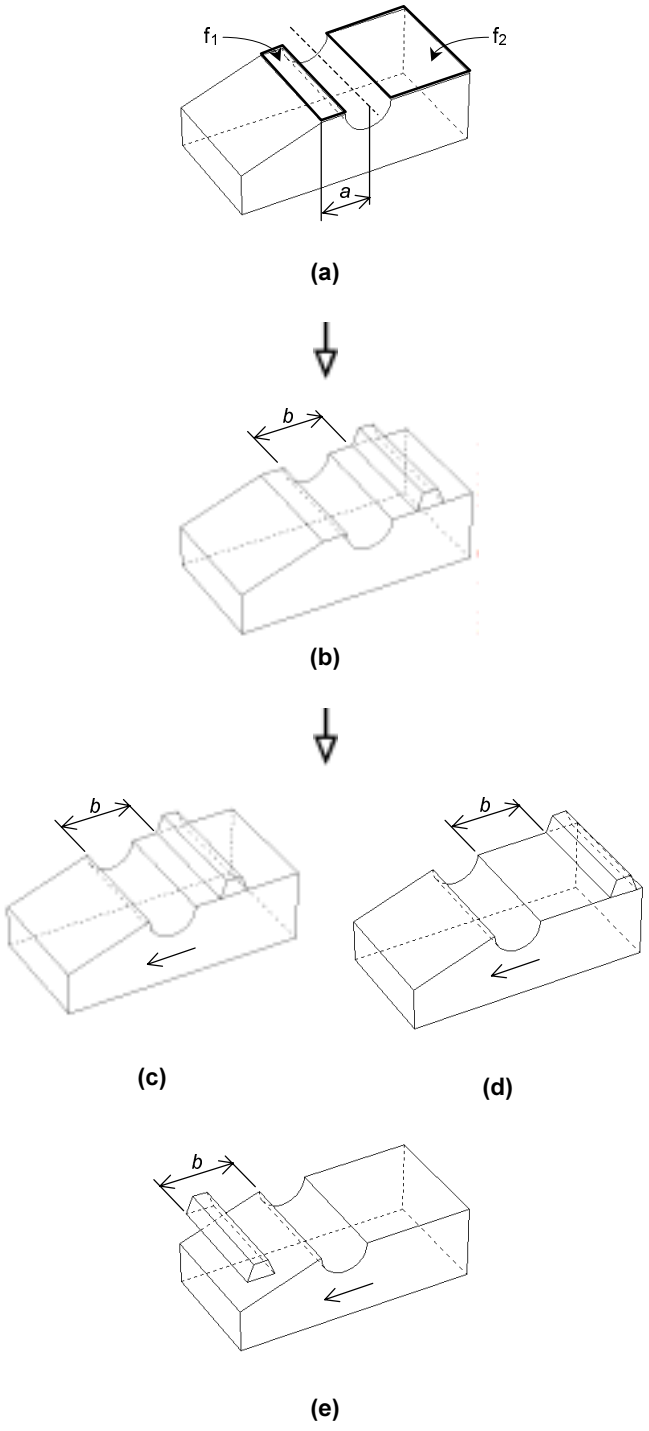



**Figure 2 – Obtaining an improper model after boundary re-evaluation**

in Figure 2.d, and other systems the model in Figure 2.e, where the rib is flipped across the slot and becomes disconnected from the object. In this last case, although the new set of parameters seems to define a "reasonable" variant of the original object, the modelling system produces an improper solid. So the only possible conclusion is that the variant should not be in the family of objects defined by the prototype, because it is definitely not in line with the user intent.

The underlying problem in both examples is the so-called *persistent naming problem*. At each modelling step in the model history, references can be made to topologic entities in the intermediate boundary representation, which is the result of all previous modelling steps. However, these entities may have been split, merged or deleted as a result of previous modelling operations, depending on the parameter values. As a result, references can be made to entities of intermediate boundary representations that are no longer there. In the example of Figure 1, face $f_1$ resulted from merging the coplanar block and protrusion top faces. When these are no longer coplanar, one of them is selected by the system as the reference for attaching the through slot. In the example of Figure 2, the slot edge used to position the rib (Figure 2.b) is merged with another edge, and the system selects the merged edge (Figure 2.c), another edge (Figure 2.d), or the merged edge but the other side of it (Figure 2.e), to position the rib.

Several schemes have been proposed to solve the persistent naming problem (Kripac 1995, Capoyleas et al 1996, Lequette 1997), so that the boundary re-evaluation can at least be consistently executed, and in a deterministic way. However, the fundamental problem remains that the resulting model is not always as expected by the user: most users probably expect the model of Figure 1.c, but some users may nevertheless expect the model of Figure 1.d. Depending on how the persistent naming scheme works, the user will get one of them, but this one may not be according to his intent. One might also say that the semantics of the family of objects is not well defined.

Raghothama and Shapiro (1998) have introduced *boundary representation deformations* as a first step to come to a sounder definition of a parametric family of objects. Basically it is argued that as long as there is a continuous boundary deformation from the prototype model to the new instance possible, the new instance is considered to be a member of the family. However, as Raghothama and Shapiro also admit, continuous deformations seem to be too restrictive, and operations like splitting and merging of model entities have to be dealt with, in order to allow topological changes such as the elimination of holes, but this has not been elaborated. A major problem here seems to be that such operations can only be generically allowed or disallowed, e.g. for all holes in an object or for no holes at all. And, again, which objects belong to a family is determined, at least partly, by how the modelling system works, in particular which operations are available to allow discontinuous deformations.

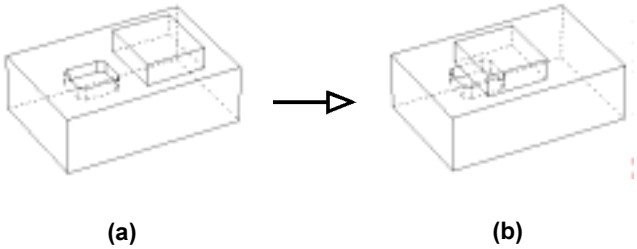



**Figure 3 - Changing semantics with a parameter modification**

An even more important shortcoming in current parametric modelling systems, which makes it difficult to come to a sound definition of a family of objects, is that currently available constraint types are not always expressive enough to adequately specify the design intent of a prototype object, and thus of the corresponding family of objects.

As an example, Figure 3.a shows a prototype model consisting of a block and, on its top face, a protrusion and a blind pocket. By modifying a positioning parameter for the protrusion, the variant given in Figure 3.b can be created. In this model, the pocket is partially closed by the protrusion, which might well be against the design intent of the prototype, in which case the variant should not be in the corresponding family of objects.

With currently available constraint types, the variant of Figure 3.b can only be excluded by setting distance constraints, for example, between the side faces of the pocket and of the protrusion. However, if there would be more than one protrusion on top of the block, separate distance constraints would have to be specified for each protrusion and the pocket to guarantee that the latter would never be closed for any variant.

For a somewhat complicated model, it is obviously unfeasible to specify the full design intent by such mutual constraints between pairs of faces. Another solution to accept or reject a variant would be to have the user inspect the resulting shape and decide. But that can be rather awkward, and might even be undesirable, in particular if the family of objects has been defined as a standard.

What is needed in the example of Figure 3 is a type of constraint that specifies that the pocket should remain open in all cases, i.e. that the top face of its shape should never be *on* the boundary of a variant. More in general, a set of constraint types is needed to explicitly specify which topological variations are allowed. Such a set of constraints, in combination with the more traditional geometric and algebraic constraints, can effectively specify the design intent, or *semantics*, of a prototype object, and therefore also of the corresponding family of objects. In addition, an automatic family membership test is needed that accepts a variant as a member of a family if and only if all its specified constraints are satisfied.

In this paper, the use of such a set of constraints is proposed to enhance the specification of a family of objects. All variants that satisfy the constraints specified for the prototype are said to belong to the corresponding family of objects. More specifically, a prototype object is defined by a *semantic feature model*, which consists of a set of features and a set of constraints; this is elaborated in Section 3. All objects are represented by a feature graph and a non-manifold cellular model, not only avoiding the problems of evaluation with a boundary representation, but also enabling a family membership test; this is elaborated in Section 4.

## 3. Defining semantics for a family of objects

This section presents the framework within which semantics is effectively incorporated in the definition of a family of objects. First, a variety of constraint types is defined, each of which can capture specific aspects of design intent. Next, these are integrated in the definition of a feature class. Finally, a formal definition for a semantic feature model is presented, providing the basis for a comprehensive definition of a family of objects.

### 3.1. Constraints

Constraints offer a convenient means to specify properties which the user requires in a feature model. Generally speaking, these properties may be classified as *geometric* or *topologic*.

**Geometric properties**
Two types of constraints can be used to specify geometric properties in and among features of a model: *geometric constraints* and *algebraic constraints*.

**Geometric constraints**: specify geometric relations (e.g. parallelism, perpendicularity or distance) between feature entities;

**Algebraic constraints**: specify expressions (equalities or inequalities) among feature parameters.

Geometric constraints can be used to specify geometric properties both within a feature instance or between entities of different feature instances. An example of the former is the requirement that the sides faces of a slot should be parallel, and an example of the latter is that the axes of two holes should be at a specified distance.

An essential property of geometric constraints is that they operate on entities of features in the model, e.g. their faces or axes, rather than on topologic entities of its geometric representation. This is crucial because, as described

**Table 1 – Classification of interactions**

| Interaction type | Description |
|---|---|
| **Splitting** | Splits the boundary of a feature into two (or more) disconnected subsets. |
| **Disconnection** | Causes the volume of an additive feature (or part of it) to become disconnected from the model. |
| **Boundary clearance** | Causes (partial) obstruction of a closure face of a subtractive feature. |
| **Volume clearance** | Causes partial obstruction of the volume of a subtractive feature. |
| **Closure** | Causes some subtractive feature volume(s) to become a closed void inside the model. |
| **Absorption** | Causes a feature to cease completely its contribution to the model shape. |
| **Geometric** | Causes a mismatch between a nominal parameter value and the actual feature geometry. |
| **Transmutation** | Causes a feature instance to exhibit the shape imprint characteristic of another feature class. |
| **Topologic** | Corresponds to the violation of a boundary constraint in a given feature. |

in the previous section, topologic entities can have an ephemeral existence in the geometric model, and are therefore unsuitable for being referenced in a persistent manner. References to feature entities, instead, remain valid as long as the respective feature instances remain in the model. By using them, the persistent naming problem outlined in the previous section is avoided.

Algebraic constraints can be used to specify parameter relations within a feature instance, e.g. an equality between the width and the length of a square-section passage feature, or among parameters of different features, e.g. requiring the depth of a pocket to be twice the diameter of a hole. Another use of algebraic constraints is for the specification of the allowable range of values for a feature parameter, by means of inequality relations.

**Topologic properties**

Topologic properties can be specified on features in a model by using two types of constraints: *boundary constraints* and *interaction constraints*. Both types of constraints, generally designated as *topologic constraints*, specify which topologic variants of a feature instance are allowed:

**Boundary constraints**: specify the extent to which its feature faces should be *on* the model boundary;

**Interaction constraints**: specify that a particular feature interaction type is not allowed for the feature instance.

Boundary constraints are of two types: *onBoundary*, which means the feature face should be present on the model boundary, and *notOnBoundary*, which means the feature face should not be present on the model boundary. Furthermore, both types of boundary constraints are parameterised, stating whether the presence or absence on the model boundary is *completely* or only *partly* required. An example of this is a blind hole class for which the top face has a *notOnBoundary(completely)* constraint, the side face has an *onBoundary(partly)* constraint, and the bottom face has an *onBoundary(completely)* constraint.

Several functional aspects of feature semantics, however, cannot be fully specified by boundary constraints alone. Those are better described in terms of the topology of the feature volume and/or of the feature boundary as a whole, and therefore require a higher-level specification, not directly based on feature entities. Such functional requirements can be violated by *feature interactions*, which are modifications of the shape aspects represented by a feature that affect its functional meaning. Examples of this are a *transmutation* interaction of a blind hole into a through hole, or an *absorption* interaction that causes a feature to cease completely its contribution to the model shape. A non-exhaustive classification of feature interactions can be found in (Bidarra 1999). For completeness, it is briefly summarised in Table 1. Topologic interaction, corresponding to the violation of a boundary constraint, is by definition always disallowed.

## 3.2. Definition of a feature class

*Features* can be defined as "representations of shape aspects of a product that are mappable to a generic shape and are functionally significant for some product life-cycle phase". In other words, each feature has a well-defined meaning, expressed through its geometric and topologic properties. A *feature class* is a structured description of all such properties, defining a template for all instances of a given feature type. These properties include the *validity conditions* that all feature instances of that type should

satisfy. These conditions, as well as the feature shape and its parameters, are specified using the constraint types presented in the previous subsection.

The basis of a feature class is a parameterised shape. For a simple feature, this is a *basic shape*, e.g. a cylinder for a hole. A basic shape encapsulates a set of geometric constraints that relate its parameters to the corresponding shape entities. For a compound feature, the shape is a combination of several, possibly overlapping, basic shapes, e.g. two cylinders for a stepped hole.

The geometry of a feature, designated the feature's *shape extent*, accounts for the bounded region of space comprised by its volumetric shape. Moreover, its boundary is decomposed into functionally meaningful subsets, the *shape faces*, each one labelled with its own generic name, to be used in modelling operations. For example, a cylinder shape has a *top*, a *bottom* and a *side* face.

A feature class comprises also the notion of *feature nature*, indicating whether its feature instances represent material added to or removed from the model (respectively *additive* and *subtractive* natures).

Creation of an instance of a given feature class typically requires assigning numeric values to a set of scalar parameters $P_F = \{p_1, p_2, \ldots, p_f\}$ –so-called *feature parameters*– for determining its shape, and specifying some references to entities of other features for positioning it.

### 3.3. Definition of a semantic feature model

The two previous subsections described the specification of feature semantics by means of constraints. The use of such constraints should now be distinguished, according to whether they are specified *within* a feature class, thus holding for all its feature instances, or they are specified by the designer *among* (or *on* a) feature instance(s). The former are called *feature constraints*, as they are members of a feature class, whereas the latter are called *model constraints*. A model constraint instance, just like a feature instance, may require assigning numeric values to its scalar *constraint parameters* $P_C = \{p_1, p_2, \ldots, p_c\}$, as well as specification of any feature entities/parameters which it refers to.

Considering *n* features in a model, together with its *k* model constraints, we can define the set of *model parameters*, $P_M$, as

$$P_M = \{p_1, p_2, \ldots, p_m\} = \bigcup_{i=1}^{n} P_{F_i} \cup \bigcup_{i=1}^{k} P_{C_i}$$

Each sequence, $\boldsymbol{p}$, of *m* numeric values assigned to the model parameters $p_j$, j=1,…,m, defines a point in the parameter space $\mathbb{R}^m$. Such a sequence fully determines the shape and position of all features, and thus also the set of points of $E^3$ that is represented by the solid model.

A *semantic feature model* can then be defined according to the following:

*Definition: a semantic feature model is a triple*

$$\mathcal{M} = \langle F, C, \boldsymbol{p} \rangle$$

*where*

*F is a set of interrelated feature instances, each one with its own set of feature constraints;*

*C is a set of model constraints applied on features of F; and*

$\boldsymbol{p}$ *is a sequence of numeric values for the model parameters that fully determines the geometry of all feature instances of F.*

Typically, not all points in $\mathbb{R}^m$ correspond to a meaningful object representation; for example, some parameters in $P_M$, in particular feature dimensions, will be limited to a non-negative range of values. In general, the semantic feature model is said to be *valid* if and only if the validity conditions of all features in the model and all model constraints are satisfied; otherwise, it is an *invalid* model. In other words, a valid semantic feature model satisfies all its constraints or, formally, its sequence $\boldsymbol{p}$ of parameter values is such that the feature constraints of all features $f_j \in F$, and all model constraints in C, are satisfied.

### 3.4. Definition of a family of objects

Considering a semantic feature model $\mathcal{M}$ as the prototype model for a family of objects, $\mathcal{F}$, we propose the following definition:

*Definition: the family of objects defined by the semantic feature model* $\mathcal{M} = \langle F, C, \boldsymbol{p} \rangle$ *is the set of models*

$$\mathcal{F} = \{ \mathcal{M}' : \mathcal{M}' = \langle F, C, \boldsymbol{p}' \rangle \text{ is a valid model, } \boldsymbol{p}' \in \mathbb{R}^m \}$$

All instances of a family of objects have the same features and the same model constraints as the prototype. Essential in the definition is that *all* constraints must be satisfied for every member of the family.

The dimension of the parameter space of a family of objects does not necessarily have to be the same as that of the prototype. For example, a family parameter, say *throughHoleDiameter*, might be mapped to the *diameter* parameters of two through holes in the prototype. More generally, the specification of the family of objects may include a system of equations to map *n family parameters* to the *m* model parameters necessary to fully determine all features in an instance of the family. Such a system can be represented in the family specification by means of alge-

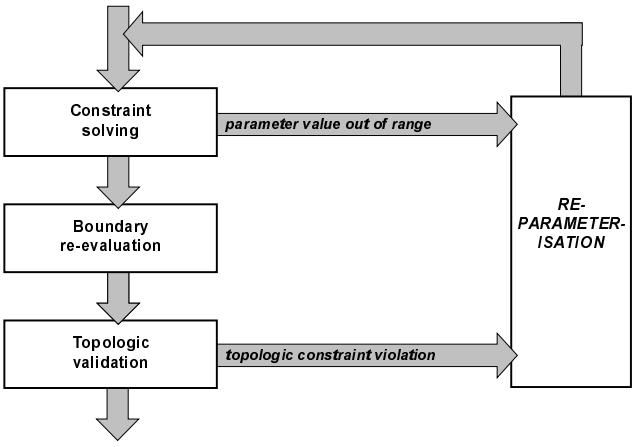



**Figure 4 - Phases of the membership test**

braic constraints.

Family parameters provide a convenient interface through which variants may easily be instantiated. In the remainder of the paper, family parameters will be designated simply as parameters, without further reference to any possible mapping between parameter spaces.

# 4. Family membership test

As mentioned in Section 2, a family membership test is required to determine whether a given sequence of parameter values $\boldsymbol{p'}$ specifies a valid instance of the family. According to the definitions in the previous section, checking membership comes down to assessing the validity of the resulting feature model.

A model validity maintenance process within the context of building a semantic feature model was described in detail by Bidarra and Bronsvoort (1999a). It has been implemented in the SPIFF system, a prototype multiple-view feature modelling system, developed at Delft University of Technology (Bronsvoort et al. 1997). In this system, a semantic feature model has a two-level structure that clearly distinguishes *modelling entities*, i.e. the entities on which all modelling operations are performed, from *entities in the evaluated geometric model*. The former are kept in the first level of the model –the so-called *Feature Dependency Graph*–, which contains all feature and (model) constraint instances, interrelated by *dependency relations*. The second level contains the evaluated geometric representation of the product in the so-called *Cellular Model*. Its entities are kept internal, being only required to "reflect" the geometry that results from the modelling operations performed on the first level.

We now concentrate on validity checking in the family membership test. This can be split into three phases, see Figure 4: (i) *constraint solving*, (ii) *boundary re-evaluation* and (iii) *topologic validation*, which will be subsequently described. In the end, we briefly explain how the user is assisted in re-parameterisation of an invalid instance to make it a member of the family.

## 4.1. Constraint solving

The first phase of the membership test is the internal geometric and algebraic constraint solving process. Its goal is to update the model parameters, i.e. dimensions, position and orientation of all features in the model, according to the new parameter values $\boldsymbol{p'}$ specified.

This task is performed by a Constraint Manager, which invokes two dedicated constraint solvers: a geometric constraint solver based on extended 3D degrees of freedom analysis (Kramer 1992), and a SkyBlue algebraic constraint solver (Sanella 1992). The iterative co-operation of these solvers, under the control of the Constraint Manager, is described by Dohmen (1997).

In this phase, the family membership test fails whenever some model parameter gets assigned a value out of its range of allowable values (specified with algebraic constraints, see Subsection 3.1). Such algebraic constraint violations are passed to the re-parameterisation mechanism for further processing, as depicted in Figure 4.

## 4.2. Boundary re-evaluation

When this second phase is reached, each feature in the model has all its parameters successfully updated. In particular, all feature shape extents have their dimensions, position and orientation fully determined. The Cellular Model may thus be updated, so that the effects of the operation are also reflected in the evaluated geometric model.

In this subsection, the main characteristics of the Cellular Model are first summarised, and then the two stages of the boundary re-evaluation process are shortly described: *re-evaluation of the Cellular Model* and its *history-independent interpretation*.

**The Cellular Model**

The Cellular Model is a *non-manifold* geometric representation of the semantic feature model, integrating the contributions from all its features. It has been described in detail in (Bidarra et al. 1998).

The Cellular Model represents an object's geometry as a connected set of volumetric quasi-disjoint *cells* of arbitrary shape, in such a way that each cell lies either entirely *inside* a shape extent or entirely *outside* it. The cells represent the point sets of the shape extents of all features in the model. Each shape extent is, thus, represented in the Cellular Model by a connected subset of cells.

Furthermore, the cellular decomposition is interaction-driven, i.e. for any two overlapping shape extents, some of

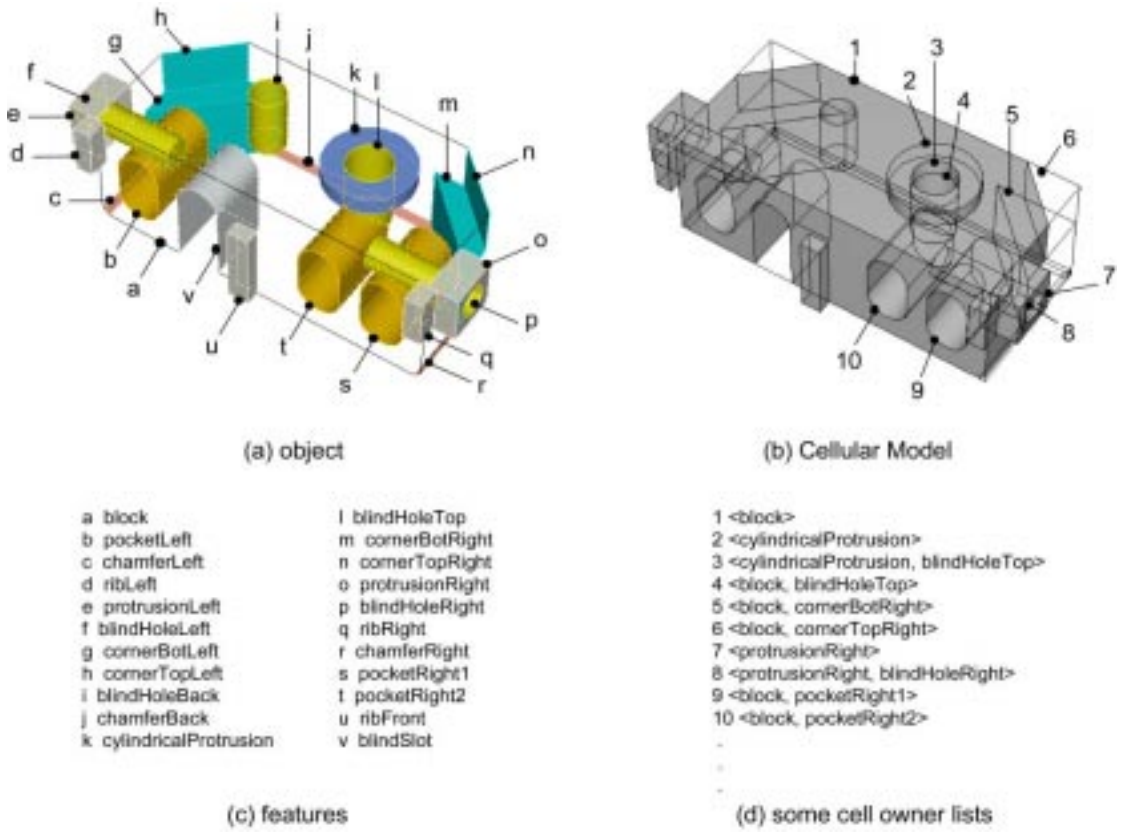



**Figure 5 – Cell owner lists in the Cellular Model**

their cells lie in both shape extents (and are called *interaction cells*), whereas the remaining cells lie in either of them. As a consequence of this, two cells can never volumetrically overlap. They may, however, be adjacent, in which case there is an interior *cell face* separating them.

As described in Subsection 3.2, the boundary of a feature's shape extent is decomposed into functionally meaningful subsets, the shape faces, each one labelled with its own generic name. Each shape face is represented by a connected set of cell faces. In order to be able to search and analyse features and their faces in the Cellular Model, each cell has an attribute –called *owner list*– indicating which shape extents it belongs to, see Figure 5. Similarly, each cell face has also an owner list, indicating which shape faces it belongs to.

Just like for features, the *nature of a cell* expresses whether its volume represents "material" of the object or not. Similarly, the *nature of a cell face* expresses whether it lies on the boundary of the object or not. Cell and cell face natures are determined after Cellular Model re-evaluation, in the interpretation stage.

The Cellular Model, including its attribute mechanism to maintain and propagate the owner lists of cells and cell faces, was implemented using the Cellular Topology husk of the Acis Geometric Modeller (Spatial 1999).

**Re-evaluation of the Cellular Model**

In contrast with most parametric systems, which use two non-associative set operations (union and difference) to evaluate the geometric model, in our approach *only one set operation* is used to evaluate the Cellular Model: it is computed by performing the *non-regular cellular union* of the shape extents of all features. Because it is a union operation, the order in which the shape extents are processed is now irrelevant for the final Cellular Model obtained. By these non-regular cellular operations, the cellular decomposition described above is computed. Essential in this process is the correct propagation of the owner lists of each cell and cell face when these are further decomposed.

Typically, when some model parameters are modified, only a few feature shapes are actually geometrically affected (i.e., in their dimensions, position and/or orientation). Consequently, *only* the shape imprint of those features needs to be updated. For this, their shapes are *removed* from the Cellular Model and then *re-added* with their new parameter values.

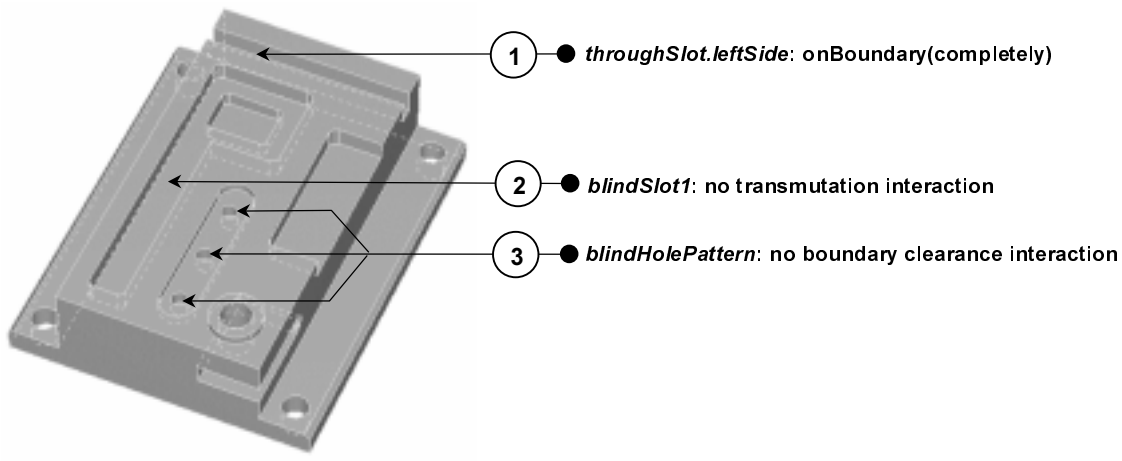



**Figure 6 – Prototype model and three of its topologic constraints**

**History-independent interpretation**
Interpretation of the re-evaluated Cellular Model consists of determining whether the point set represented by each cell does belong to (or represent "material" of) the object, i.e. determining the nature of that cell. Interpretation of the Cellular Model requires deciding *which* of the features in its owner list "prevails", either as additive or as subtractive.

For this, precedence relations among features are determined, based on dependencies between features and on possible overlap between independent features. Once all precedence relations have been established, a *global ordering* of the set $F$ of all features in the model can be easily performed by a classical topological sorting algorithm. The features in the resulting sorted sequence are then assigned unique, increasing precedence numbers. Eventually, every cell owner list (a subset of $F$) is sorted according to these precedence numbers, and each cell's nature becomes therefore automatically determined: it is the nature of the last feature in its owner list, i.e. the feature with the highest precedence number. This scheme always yields an interpretation of the Cellular Model that is unambiguously determined, independent of the model history. For details on the precedence criteria and sorting algorithms involved in Cellular Model interpretation, as well as for a discussion on the advantages of history-independent boundary evaluation, the reader is referred to (Bidarra and Bronsvoort 1999b; Bidarra and Bronsvoort 2000).

### 4.3. Topologic validation

Once the model boundary has been re-evaluated, the shape imprint of all modified features exhibits their updated geometry and topology, and detection of violations in topologic constraints can take place. In this third phase, the model is considered invalid, and thus the family membership test fails, if any boundary or interaction constraint is violated for some feature.

For checking each boundary constraint and each interaction constraint, see Table 1, topologic entities of the Cellular Model are queried and analysed, using the feature information maintained in their owner lists. Details on the interaction detection algorithms can be found in (Bidarra 1999). Eventually, the set of constraint violations, if any, is analysed, and their causes are identified and passed for further processing to the re-parameterisation mechanism, see Figure 4.

The re-parameterisation process includes reporting to the user the constraint violations detected, documenting their scope and causes, and, whenever possible, providing corrective hints. In case of a parameter that has been assigned a value out of its specified range, this is straightforward. In case of a topologic constraint violation, re-parameterisation hints are presented to the user in terms of the relevant parameter values causing the violation.

## 5. Results and conclusions

We have introduced a new approach to define a family of objects, and to determine whether a particular instance belongs to the family. A family of objects is defined by creating a prototype semantic feature model (Bidarra and Bronsvoort 2000), in which the semantics of the family can be fully specified by several types of constraints, in a relatively easy way. Given a set of parameter values, the family membership test can be used to determine whether the corresponding instance belongs to the family.

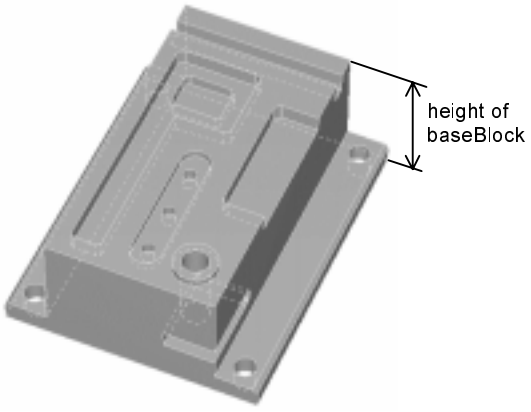
In Figures 6-8, we show an example of its application

in the SPIFF modelling system. In Figure 6 the geometry of a prototype model is shown, together with three of its topologic constraints relevant for the example. In Figure 7 three instances are shown that do belong to the family defined by the prototype, and in Figure 8 three instances that do not belong to the family.

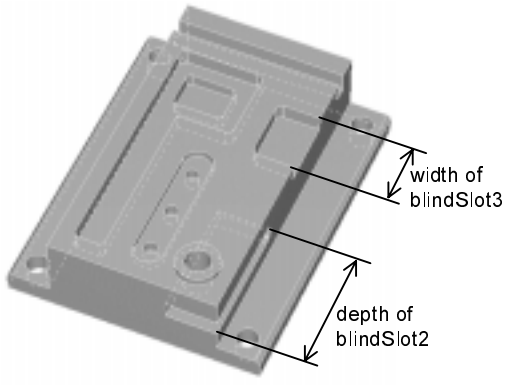
The first advantage of the approach is that the instance corresponding to a set of parameter values always has a shape that may be expected on the basis of the prototype and the parameter values. Unexpected results, which regularly occur in current parametric modelling systems, in particular because of the persistent naming problem in boundary re-evaluation, do not occur. The reason for this is that all references during modelling are made to persistent feature entities instead of non-persistent entities in a boundary representation.

The second advantage of the approach is illustrated by the example given above. If one considers the shape of the prototype in Figure 6 and of the instances in Figures 7 and 8 only, it is not obvious that the instances in Figure 7 should indeed belong to the family defined by the prototype, and the instances in Figure 8 not. In our approach, the membership of these instances is determined by the constraints in the definition of the prototype of the family. More generally, the design intent, or semantics, of a family of objects can be specified much more precisely than in current parametric modelling systems, in particular with the topologic constraints.

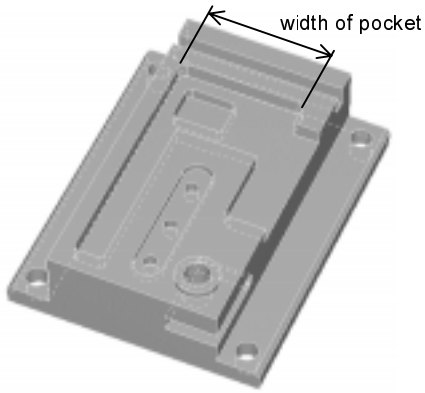
Regarding the implementation, it has been shown that a cellular model can be very valuable to represent objects in the context of defining families of objects, as has also been suggested by Shapiro and Vossler (1995) and Raghothama and Shapiro (1998). The cellular model is in particular exploited by the family membership test; for example, the model offers even the possibility to give, in case an instance is not a member of the family, hints to the user to re-parameterise the instance in a way to make it a member of the family. Bidarra et al. (1998) already showed that a cellular model can be very useful in several other contexts too, e.g. multiple-view feature modelling. So although cellular models are not used in current modelling systems, they might well become more popular in the future.




**(a) height of *baseBlock* increased**




**(b) depth of *blindSlot2* increased; width of *blindSlot3* decreased**




**(c) width of *pocket* increased**

**Figure 7 – Examples of members**

## References

Anderl, R. and Mendgen, R. (1995) Parametric design and its impact on solid modeling applications. In: *Proceedings Solid Modeling '95 – Third Symposium on Solid Modeling and Applications, 17–19 May, Salt Lake City, UT, USA,* Hoffmann, C.M. and Rossignac, J.R. (Eds.), ACM Press, New York, pp. 1–12

Bidarra, R. (1999) Validity maintenance in semantic feature modeling. PhD Thesis, Delft University of Technology, The Netherlands

Bidarra, R. and Bronsvoort, W.F. (1999a) Validity maintenance of semantic feature models. In: *Proceedings of Solid Modeling '99 – Fifth Symposium on Solid Modeling and Applications*, *9–11 June, Ann Arbor, MI, USA*, Bronsvoort, W.F. and Anderson, D.C. (Eds.), ACM Press, New York, pp. 85–96

Bidarra, R. and Bronsvoort, W.F. (1999b) History-independent boundary evaluation for feature modeling. In: *CD-ROM Proceedings of the 1999 ASME Design Engineering Technical Conferences, 12–15 September, Las Vegas, NV, USA,* ASME, New York

Bidarra, R. and Bronsvoort, W.F. (2000) Semantic feature modelling. To be published in: *Computer Aided-Design* **32**(3)

Bidarra, R., de Kraker, K.J. and Bronsvoort, W.F. (1998) Representation and management of feature information in a cellular model. *Computer-Aided Design* **30**(4): 301–313

Bronsvoort, W.F., Bidarra, R., Dohmen, M., van Holland, W. and de Kraker, K.J. (1997) Multiple-view feature modelling and conversion. In: *Geometric Modeling: Theory and Practice – The State of the Art,* Strasser, W., Klein, R. and Rau, R. (Eds.), Springer, Berlin, pp. 159–174

Capoyleas, V., Chen, X. and Hoffmann, C.M. (1996) Generic naming in generative, constraint-based design. *Computer-Aided Design* **28**(1): 17–26

Chen, X. and Hoffmann, C.M. (1995) On editability of feature based design. *Computer-Aided Design* **27**(12): 905–914

Dohmen, M. (1997) Constraint-based feature validation. PhD Thesis, Delft University of Technology, The Netherlands

Kramer, G.A. (1992) Solving geometric constraint systems: a case study in kinematics. The MIT Press, Cambridge, MA

Kripac, J. (1995) A mechanism for persistently naming topological entities in history-based parametric solid models. In: *Proceedings Solid Modeling '95 – Third Symposium on Solid Modeling and Applications, 17–19 May, Salt Lake City, UT, USA,* Hoffmann, C.M. and Rossignac, J.R. (Eds.), ACM Press, New York, pp. 21–30. Also in: *Computer-Aided Design* 1997; **29**(2): 113–122

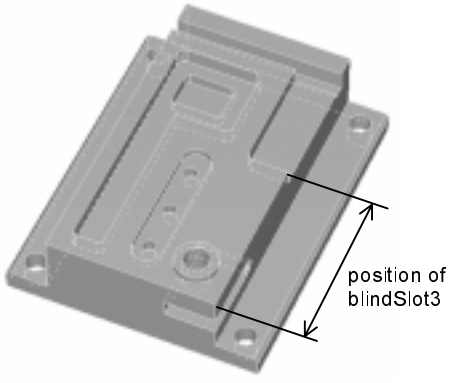
Lequette, R. (1997) Considerations on topological naming. In: *Product Modeling for Computer Integrated Design and Manufacturing – Proceedings TC5/WG5.2 International Workshop on Geometric Modeling in Computer Aided Design, 19–23 May 1996, Airlie, VA, USA*, Pratt, M., Sriram, R.D. and Wozny, M.J. (Eds.), Chapman & Hall, London, pp. 394–403

Raghothama, S. and Shapiro, V. (1998) Boundary representation deformation in parametric solid modeling. *ACM Transactions on Graphics* **17**(4): 259–286

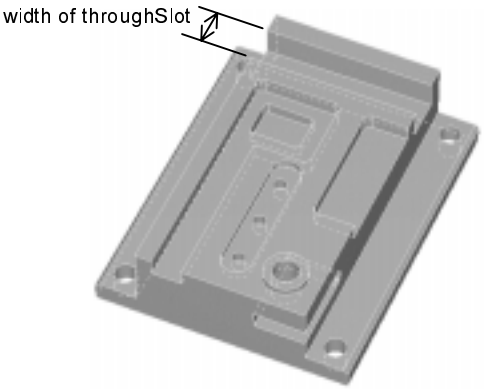
Sanella, M. (1992) The SkyBlue constraint solver. Technical Report 92-07-02, University of Washington, WA, USA

Shapiro, V. and Vossler, D.L. (1995) What is a parametric family of solids? In: *Proceedings of Solid Modeling '95 – Third Symposium on Solid Modeling and Applications*, *17–19 May, Salt Lake City, UT, USA,* Hoffmann, C.M. and Rossignac, J.R. (Eds.), ACM Press, New York, pp. 43–54

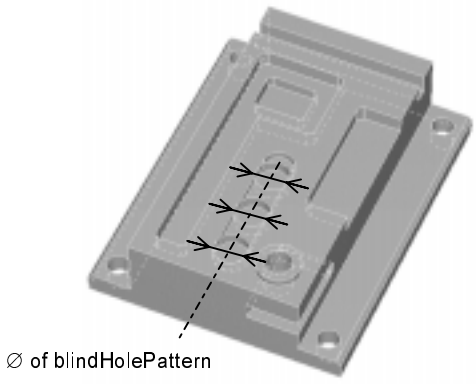
Spatial (1999) Acis 3D Modeling Kernel, Version 5.3, Spatial Technology Inc., Boulder, CO, USA




**(a) position of *blindSlot3* modified: constraint ① violated**




**(b) width of *throughSlot* increased: constraint ② violated**




**(c) diameter of *blindHolePattern* increased: constraint ③ violated**

**Figure 8 – Examples of non-members**