# Modeling Families of Objects: Review and Research Directions

H.A. van der Meiden[1] and W.F. Bronsvoort[2]
Delft University of Technology, The Netherlands
[1] H.A.vanderMeiden@tudelft.nl
[2] W.F.Bronsvoort@tudelft.nl

## ABSTRACT

We review different approaches used for modeling families of objects, and we propose directions for future research. First, we discuss parametric, history-based models, which are mostly used in practice, but are often not adequate for modeling families of objects. We identify the main problem, namely that the semantics of families of objects, in particular topological properties, cannot be adequately specified and maintained with these models. Next, we discuss declarative models for families of objects. In such models, constraints are used to specify the semantics of a family of objects, including topological properties, and members of the family are determined by solving the constraints. We find that the declarative approach is promising, but also that current implementations are too limited for practical use. Also, we look at what kind of modeling tools are needed to facilitate creating and using families of objects, in particular, tools for computing parameter ranges and critical values corresponding to topological changes. Finally, we conclude that for future CAD systems, and other geometric modeling applications, the declarative approach needs to be further developed, and we propose a declarative framework for geometric modeling.

**Keywords:** families of objects, semantics, declarative models, modeling tools

## 1. INTRODUCTION

Geometric models are used in many different applications, e.g. CAD, scene modeling, and medical and scientific applications. What is needed in many of these applications is not a static geometric model, but a generic representation that can capture possible variations of a shape, i.e. represent similar shapes in a generic way. Such sets of similar shapes are called families of objects. Typically, families of objects are characterized by parametric models, which represent all possible shape variations by a set of parameters. By varying the parameters, variants of the model are obtained, each representing a different but similar object (see Fig. 1). The (infinite) set of all possible objects that can thus be obtained, is the family of objects represented by the parametric model. The objects in this set are the members of the family.

However, there is not yet a good way to parameterize geometric shapes in such a way that desirable properties and characteristics of the family of objects can be a priori specified and maintained. This means that after editing a model, or instantiating a family member, it should be manually checked for undesirable artifacts, which is error-prone and expensive for large models. Thus, a better definition of families of objects is needed, such that models can be automatically validated.

In general, we can say that a family of objects is a set of similar objects. But similar in what respect? Depending on the exact definition, the objects in Fig. 1 may or may not belong to one and the same family. On the one hand, it can be argued that these objects are members of the same family because they can be constructed in the same way, i.e. by the same sequence of geometric operations. Such constructive, or history-based, definitions are mostly used in practice [9], but, as we shall see, are often not satisfactory. On the other hand, it can also be argued that objects (c) and (d) do not belong to the same family as object (a) and (b) because their topology is different, i.e. the last two objects cannot be mapped onto the first two objects by a continuous transformation [22]. Such formal definitions, based on mathematical topology, are not widely accepted, because they are too restrictive for many applications.

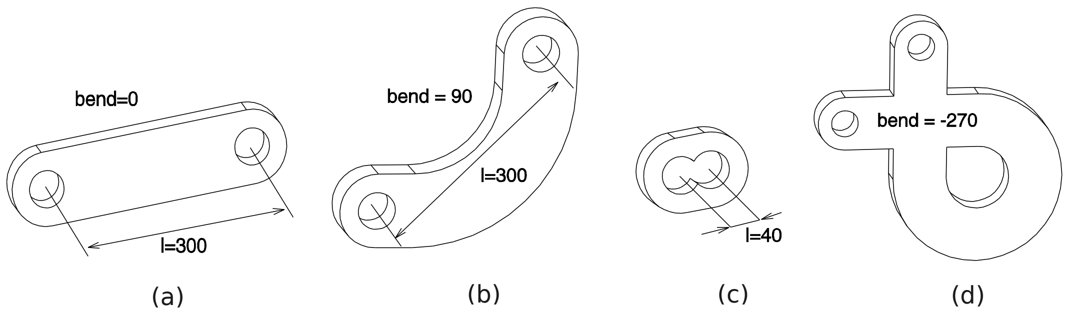


Fig. 1. Instances of a CAD model with two parameters: **l** and **bend.**

More importantly, the previous two definitions do not take into account that for different families of objects, different properties may be important, and it is thus desirable that when modeling families of objects, the user is able to precisely specify which objects are family members, and which are not, i.e. the semantics of a family. For example, the semantics of a family of objects may be that members must be able to function as a connecting rod between two round shafts, and thus each member must contain two separate, round holes. This would imply that objects (a), (b) and (d) from Fig. 1 might belong to the family, but object (c) does not. Such semantic, or declarative, definitions are promising, but currently not used in practice because they require a radical departure from the history-based model and workflow used in current modeling systems.
In Section 2, we review parametric, history-based models for families of objects used in practice, and extensions proposed in scientific literature. Next, in Section 3, we discuss declarative models for families of objects. In Section 4, we look at the kind of facilities needed for modeling families of objects. In particular, we look at current modeling facilities, and tools for computing parameter ranges and tracking topological changes. Finally, in Section 5, we suggest directions for future research, and, in particular, propose a general framework for declarative geometric modeling that can be used for a wide variety of applications.

## 2. PARAMETRIC MODELS

Families of objects are typically represented by parametric models. First, we discuss history-based models, which are created by current feature-based CAD systems, in Section 2.1. Next, in Section 2.2, we discuss more general dual-representation models, suggested by academics. Even more generally, the previous models can be considered as procedural or rule-based models. These models are discussed in Section 2.3, and we argue that these are not suitable for specifying semantics of families of objects.

### 2.1. History-based Models

Current commercial CAD systems (e.g. Pro/Engineer, CATIA, SolidWorks, NX, SolidEdge, Inventor) are parametric feature modeling systems. Such systems create models that essentially describe a history of modeling operations. These operations add features by operating on a boundary representation (b-rep). To remove a feature, the corresponding operation is removed from the history, and the history is re-evaluated to determine a new b-rep. We refer to these systems as history-based modeling systems, and to models produced by these systems as history-based models.
The procedure for modeling a family of objects with these systems, is as follows. Initially a single family member is modeled, which is referred to as the prototype object. Each modeling operation is instantiated with parameter values specified by the user. Other members of the family are instantiated by re-evaluating the modeling history, stored in the prototype, with a new set of parameter values. Each modeling operation is now executed with the new parameter values, to create a new b-rep, which represents the requested member of the family.
History-based models, even though they are the de-facto standard for commercial modeling systems, are not really suitable for representing families of objects. In [3], several major problems with the history-based approach are identified, of which the most relevant, in the context of families of objects, are the persistent naming problem, the feature ordering problem, and the inability to maintain feature semantics.
The persistent naming problem, essentially, is the problem of identifying corresponding entities in different members of a family. This identification is necessary because operations in the modeling history of a CAD model can contain references to entities that were created by previous operations in the modeling history, in particular, references to b-rep entities. Such a reference can be used, for example, for positioning a feature. However, when evaluating the history for different parameter values, a new b-rep is built, and references must now point to the corresponding

entities in this new b-rep. Therefore, a so-called persistent naming scheme is needed that can identify corresponding b-rep entities created for different parameter values.
Developing a persistent naming scheme is very difficult. One of the issues that a persistent naming scheme must deal with, is that entities may disappear or may be merged or split when parameter values are changed. It has been shown that current CAD systems use a flawed approach that can result in errors. Several schemes have been brought forward to alleviate the persistent naming problem, so that history re-evaluation can at least be consistently executed; for an overview see [12]. However, it should be noted that the persistent naming problem has not been solved completely, and that current naming schemes only work for most common situations.
The second problem with history-based models, the feature ordering problem, is caused by the fact that features add or remove material from the model in a fixed order. The order of modeling operations that seemed appropriate for a particular family member, namely the prototype object, may not yield the expected result when re-evaluating the history to create other family members. Consider, for example, Fig. 2. The prototype object consists of a base block, a protrusion feature and a blind hole feature, as shown in Fig. 2(a). When instantiating a variant object where the depth of the blind hole is increased beyond the height of the base block, two results are possible, depending on the order in which the features were created in the prototype. If the protrusion was created before the blind hole, the model shown in Fig. 2(b) emerges. If, however, those features were created in the reverse order, the model shown in Fig. 2(c) emerges.

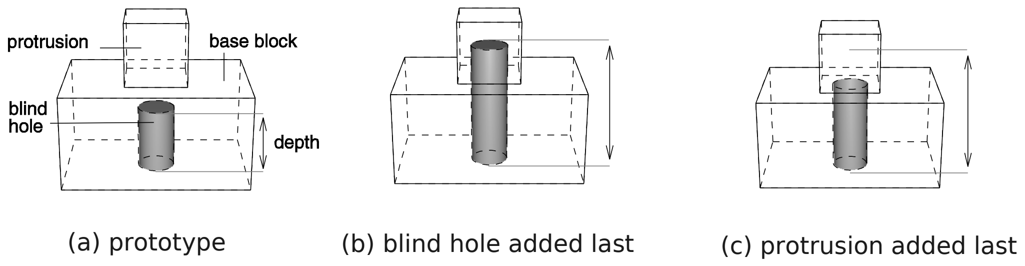
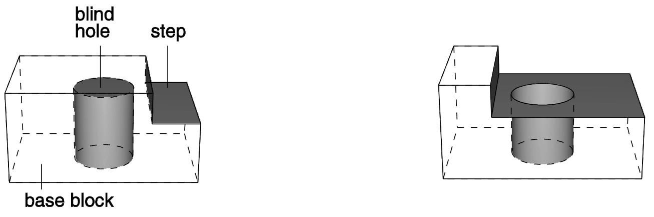


Fig. 2. The feature ordering problem.

The implication of the feature ordering problem for modeling families of objects, is that the order of operations must be taken into consideration, even though the effect may not be visible in the prototype. Although the order of features in the modeling history can usually be edited, this complicates design and editing of family models, in particular models with many interacting features.
The third problem with history-based models is maintaining feature semantics. Users of a CAD system expect features to have certain semantics, i.e. certain properties that are meaningful for the function or manufacturing of the product being modeled. In particular, topological properties are relevant for feature semantics. Due to interaction with other features, however, the topological properties of a feature may change. For example, Fig. 3(a) shows a prototype object consisting of a base block, a blind hole feature and a step feature. The semantics of a blind hole requires that the hole has a bottom, i.e. that the hole does not cut entirely through the object. When the step feature is changed as in Fig. 3(b), the blind hole feature does cut through the object, thus the semantics of the feature has changed, from the semantics of a blind hole to the semantics of a through hole.
By using a limited set of feature types and strict adherence to proven modeling practice, undesirable situations as described above can sometimes be avoided. However, this practice in fact obscures the problems with history-based models, which will still occur, in unpredictable ways.
To maintain feature semantics, topological properties must be verified, and if necessary, action must be taken to restore feature semantics, i.e. the user should be informed. Current CAD systems, however, can only check the topological properties of a feature during instantiation of the feature into the model. If, due to interaction with other features, the topological properties of a feature change at later stages in the evaluation of the modeling history, this cannot be detected. The reason is that the result of feature operations is stored in a b-rep, and the topology of the features cannot be stored in this representation. As a result, topological properties of the features cannot be adequately verified.



(a) (b) after modifying step feature

Fig 3. The problem of maintaining feature semantics.

## 2.2. Dual-representation Models

To understand the problems with modeling families of objects in a more general, theoretical context, the concept of dual-representation models has been introduced [22]. Such models consist of a parametric representation, e.g. a CSG representation, and a geometric representation, e.g. a cell-complex representation. History-based models are dual-representation models too, where the parametric representation is the modeling history, and the geometric representation is the b-rep. For a given set of parameter values, the parametric representation can be evaluated, resulting in the geometric representation of a family member.

This view has led to considering two types of families: the parameter-space family and the representation-space family. The parameter-space family is the set of all objects that can be obtained by varying the parameters. The representation-space family corresponds to the set of all objects that can be obtained by certain operations on the geometric representation.

A specific representation-space family is described by the concept of boundary representation deformation [19]. Basically, a family of objects is here defined by a prototype b-rep, and contains all objects that can be created by a continuous deformation of the prototype. The authors acknowledge that this definition of a family is too restrictive for practical modeling of families of objects, because the boundary representation deformation cannot account for splitting and merging of topological entities.

A more general framework for families of objects has been proposed in [20]. Here, the concept of part families is described using category theory, a branch of mathematics that deals with broad classes of mathematical objects, such as the category of sets and the category of topological spaces. A part family is defined as a sub-category of the category of cell-complexes, such that there is a mapping between the cell-complexes in the part family, with certain continuity preserving properties. This allows some topological variations, e.g. splitting and merging of topological entities.

To be able to determine whether an object is a member of such a representation-space family, geometric representations, generated for different parameter values, must be compared. In [21], a method is proposed to compare different geometric models, created from the same parametric model, by uniquely identifying entities. However, this basically requires that the persistent naming problem is solved, which is not generally possible, in particular when a model contains curved surfaces. In [18], a so-called constructive topological representation is presented that allows a mapping between different models to be established, without uniquely identifying all topological entities. From such a mapping, it can be determined whether the models are in the same representation-space family.

The models for families of objects discussed above are mostly concerned with preserving continuity in the geometric representation, and therefore, families are defined in terms of continuous transformations or mappings. However, this definition of a family of objects is rather limited; in practice, it may be desirable for a family of objects to have members with more variation in topology. For example, it may be desirable to specify a family of objects with two hole features, such that in some members the two holes intersect, whereas in other members, the two holes do not intersect. In another family, intersection of the holes may not be desirable. Thus, it should be possible to specify in detail, when modeling a family, what is and what is not desirable in its members, i.e. the semantics of the family.

## 2.3. Procedural and Rule-Based models

The models for families of objects considered above are all procedural models. Procedural models (or constructive or imperative models) specify how to construct objects using a procedure, i.e. a sequence of parametric operations. The history-based model is essentially a procedural model, where parametric operations are performed on a b-rep.

But, how suitable is the procedural model for specifying the semantics of families of objects? To specify the semantics of a family of objects, we should be able to specify invariant properties, i.e. properties that must hold for all objects in the family. In particular, because we are mostly

concerned with the shape of objects, we should be able to specify invariant geometric and topological properties, e.g. the diameter of a hole and that it must be a blind hole.
Creating a procedural model for a family of objects, such that a set of particular geometric and topological properties holds for all possible resulting objects, can be very difficult, because the geometry and topology of an object may depend on the parameter values of all the operations, and the order in which these are executed. The problem is aggravated because some important topological properties cannot be specified and verified in boundary representations (see Section 2.1).
In general, creating a procedure such that some invariant property holds is a non-trivial problem; the more complex a procedure becomes, the more difficult it will be to guarantee specific invariant properties. Basically, this is the art of programming, which is not something we want to burden users of a CAD system with. Thus, procedural models are not particularly suitable for creating families of objects.
Families of objects may also be created using rule-based models, which are commonly used in knowledge-based engineering systems and automated design synthesis systems. In such systems, requirements are specified by a set of rules. The rules can be executed by the system in any order, to construct a set of objects, representing possible design solutions. The set of all possible objects that can be created from a set of rules can be considered a family of objects. Most rule-based models are based on shape grammars [23]. A shape grammar defines rules that perform elementary replacements in geometric representations.
A rule-based model can be considered to define a set of procedural models; i.e. rules are equivalent to parametric operations, only the order in which rules are executed is not fixed. Therefore, rule-based models have the same limitation as procedural models, namely that it is hard to guarantee invariant properties after executing a sequence of parametric operations. More generally, in [1] it is argued that all models based on parametric modeling operations are unsuitable for specifying both geometric and topological properties. Thus, rule-based models, like procedural models, are not suitable for specifying the semantics of families of objects.

## 3. DECLARATIVE MODELS

Declarative models, in contrast with procedural and rule-based models, explicitly state invariant properties of objects, but not how to construct those objects. A declarative model consists of variables, i.e. elements that exist in all objects in the family, but whose properties can vary, and constraints, which state invariant properties by imposing relations between variables. The model does not specify how to satisfy those constraints, but rather, a constraint solver is used to determine values for the variables such that all constraints are satisfied, i.e. the solutions or realizations of the model. In general, there can be many solutions to a system of constraints, and thus a declarative model can naturally describe a family of objects.
Constraints have been used in several declarative scene modeling systems; for an overview see [5] and [7]. Typically, the model in such a system consist of several objects that are placed in 3D space to satisfy positional constraints, specifying, for example, that object A must be to the left of object B. The modeling system determines various configurations of the objects in the scene, and presents these to the user. However, for modeling families of objects with invariant geometric and topological properties, these constraints cannot be used. Instead, geometric and topological constraints are needed.
Geometric constraints are used in current CAD systems to specify geometric relationships in sketches [8]. Such constraints are imposed on 2D geometric primitives such as lines and circles, to constrain their dimensions (lengths, radii), their relative position and orientation (distances, angles), and other relations (adjacency, tangency).
Topological constraints state invariant topological properties that must be satisfied by all members of a family. Useful topological constraints state requirements on the connectivity of specific point sets in a model that are meaningful to the user, i.e. features or faces of features. For example, a topological constraint may state that the bottom face of a blind hole feature must be on the boundary of the model, so that the hole is actually blind. In this way, topological constraints determine the possible topological variations of a model, independently of the geometric constraints. Thus, whereas geometric constraints are used to parameterize the shape of the model, topological constraints are used to limit the range of topological variations of the shape.
Such topological constraints cannot, in general, be specified in current CAD systems, because these systems create history-based models and use a b-rep for representing the geometry. The topology of the b-rep is determined by evaluating the modeling history, independently of any topological constraints. And although some topological aspects may be implicitly checked by such systems, in general, topological constraints cannot be verified, because the b-rep does not contain all topological information needed for this.
Therefore, we have developed declarative models for CAD, not based on modeling history and the b-rep, which are discussed next.

## 3.1. The Semantic Feature Model

The Semantic Feature Model (SFM) is a declarative model that allows feature semantics to be specified and maintained using constraints [3].

A SFM consists of a set of features, constraints specified in the features (feature constraints), and any additional constraints between features (model constraints). Each feature is instantiated from a feature class, which defines a parameterized shape with topological properties, in particular, a nature attribute, boundary constraints and interaction constraints.

The value of the nature attribute is either “additive” or “subtractive”, indicating whether the feature adds material to the model or removes material from the model.

A boundary constraint is imposed on a feature face, and specifies that the face must be (partially or completely) on the boundary of the model, or may not be (partially or completely) on the boundary of the model. A boundary constraint can be used to specify, for example, that the bottom face of a blind hole feature must be on the boundary of the model, so that the hole is always blind.

Interaction constraints are imposed on the feature as a whole. Interactions with other features may create specific topological patterns, which can be disallowed by these constraints. An interaction constraint can be used to specify, for example, that a feature may not be split into disjoint parts by other features in the model. Table 1 lists interactions commonly found in feature models that can be constrained in the SFM.

| Interaction type | Description |
|---|---|
| Splitting | Causes the boundary of a feature to be split into two or more disconnected subsets |
| Disconnection | Causes the volume of an additive feature (or part of it) to become disconnected from the model |
| Obstruction | Causes (partial) obstruction of the volume of a subtractive feature |
| Closure | Causes a subtractive feature volume to become (part of) a closed void inside the model |
| Absorption | Causes a feature to cease completely its contribution to the model boundary |

Table 1: A list of interactions in feature models. Adapted from [3].

The geometric representation of the SFM is the cellular model (CM), a cell-complex representation that can be used to store semantic feature information [2]. The cellular model consists of topological entities, i.e. vertices, edges, faces and cells, and all topological relations between these entities. Note that in literature on cell-complex representations, usually, all topological entities are called cells, whereas here we use the word cell only for those entities representing volumes. All cells are quasi-disjoint, meaning that cells may touch (share a face, edge or vertex), but they cannot intersect. Each cell represents either a volume filled with material, i.e. it is part of the modeled object, or it represents an empty volume, i.e. it is not part of the object.

The CM is constructed by combining all the feature shapes in the model, and can be updated efficiently when the feature model is changed [4]. If features intersect, their entities are split into non-intersecting new entities, which are then added to the CM. The CM thus contains the geometry of all the features, including the geometry that is not on the boundary of the model. In contrast, the b-rep of a history-based model loses feature geometry with each set operation. For each cell, the CM stores a list of features that overlap with the cell, referred to as the owner list of the cell. For each cell, it is determined whether it contains material, by analyzing the dependencies between features in the owner list and determining a feature precedence order, i.e. which feature determines the nature of the cell.

After it has been determined for each cell in the CM whether it contains material, the validity of all features is checked by verifying that all boundary and interaction constraints are satisfied. If any constraint is not satisfied, the model is invalid, and the user is guided through a recovery process, until validity has been restored.

The main shortcoming of the SFM as a basis for defining families of objects, is that feature dependency analysis cannot always unambiguously decide which cells should contain material, in particular when there are features in the owner list of a cell that are independent and have conflicting natures. For example, in Fig. 2., the protrusion and the blind hole features are both dependent on the base block, because they refer to it for positioning, but there are no dependencies between these two features. Therefore, either Fig. 2(b) or Fig. 2(c) can emerge, depending on the order of feature creation, just like in history-based systems. In general, feature dependency analysis does not respect the semantics of features as specified by topological constraints. Topological constraints are only checked after a model has been created, instead of being used to create a valid model. As a result, a family of objects defined by a SFM is not complete, i.e. sometimes no object is found that satisfies the topological constraints, even though such an object exists.

**3.2. The Declarative Families of Objects Model**

In [13] we presented a new model for families of objects, the Declarative Family of Objects Model (DFOM), based on, and extending the concepts of the SFM. In the DFOM, unlike in the SFM, the geometry and topology of a model do not have to be fully specified. Thus, a DFOM generally represents a family of objects. Also, to determine members, topological constraints are solved, instead of using feature dependency analysis and constraint checking, as is the case in the SFM. This allows to properly specify families of objects.

A DFOM consists of geometric variables, called carriers, and topological variables, called constructs. Carriers define surfaces that partition space, e.g. a planar carrier defines a planar surface and two sides of the surface. Constructs basically represent point sets, i.e. volumes, surfaces, curves and individual points, constructed by intersections of subspaces defined by carriers. Carriers and constructs are related via so-called subspace constraints. Geometric properties of a family can be specified by geometric constraints on carriers, topological properties by topological constraints on constructs.

Carriers are used in various representations for families of objects, e.g. in [21] and [18]. Although carriers can be defined in any dimension, here we consider a carrier to be a function that partitions 3D Euclidean space into three subspaces, labeled *IN*, *OUT* and *ON*. The subspaces that correspond to *IN* and *OUT* are each connected, *3D* point sets. The subspace labeled *ON* is a surface, separating the *IN* and *OUT* subspaces.

A construct is a variable that represents a point set, corresponding to a subspace of $^3$. The actual point set represented by a construct, i.e. its value in a specific realization, is determined by the subspace constraints and the topological constraints imposed on it. Subspace constraints specify that a construct is a subset of the *IN*, *ON* or *OUT* subspace of a carrier. A construct that is not constrained *ON* any carrier represents a volume, which can be bounded by constraining the construct *IN* and/or *OUT* with respect to several carriers. For example, a construct constrained *IN* a planar carrier and *IN* a spherical carrier generally represents a half-sphere volume. A construct that is constrained *ON* a single carrier generally represents a surface. For example, a construct constrained *ON* a planar carrier, and *IN* a spherical carrier, generally represents a disk. A construct that is constrained *ON* two carriers generally represents one or more curves, and finally, a construct that is constrained *ON* three carriers generally represents a finite set of points. Some examples of systems of constructs and carriers are shown in Fig. 4 and Table 2.

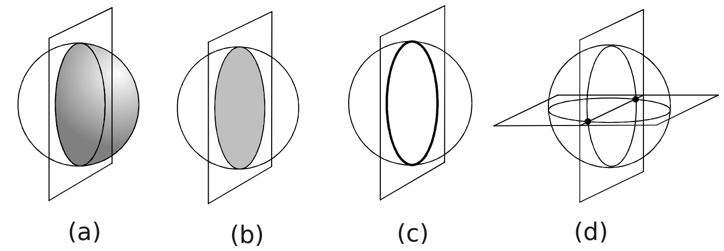


Fig. 4. Constructs build from a spherical carrier and one or two planar carriers

| Fig. 4 | construct | sphere | plane 1 | plane 2 |
|---|---|---|---|---|
| (a) | half-sphere | *IN* | *IN* | |
| (b) | disk | *IN* | *ON* | |
| (c) | circle | *ON* | *ON* | |
| (d) | two points | *ON* | *ON* | *ON* |

Table 2. Constructs and carriers in Fig. 4.

Implicitly, a DFOM defines a set of realizations, i.e. all object models that satisfy the constraints in the DFOM, thus representing all possible family members. The geometric representation of realizations is the cellular model (CM), the same representation that is used in the SFM.

Realizations may be derived from a DFOM by a three-step process (see Fig. 5). The process starts with solving the geometric constraints on the carriers in the DFOM [17]. After the geometric constraint system has been solved, the geometry of all carriers has been determined. The geometry of all constructs is then determined, by evaluating the subspace constraints, i.e. by intersecting carrier geometry. From this, a CM is constructed that contains a cell for every volume construct or intersection of volume constructs. To determine which cells of the CM contain material, we solve the system of topological constraints [15]. Basically, the material value of each cell is a Boolean variable, and topological constraints similar to those used in the SFM, e.g. boundary and interaction constraints, and the concept of feature nature, are implemented as Boolean constraints. A Boolean constraint solver, based on SAT technology, is used to determine possible solutions.

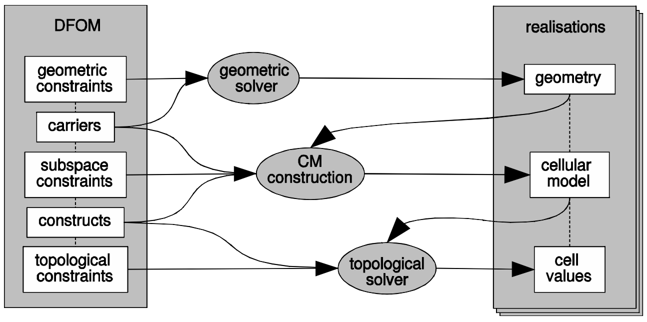


Fig. 5. The three-step process for deriving realizations from a DFOM.

A DFOM can have zero, one, a finite or a (countable or uncountable) infinite number of realizations. These realizations correspond to the members of the family. However, family membership can better be defined in terms of DFOMs, as follows.
A DFOM *M* represents a member of the family represented by a DFOM *F*, if and only if

- *M* has the same set of variables as *F*,
- *M* has a superset of the constraints of *F*, and
- *M* has exactly one realization.

Similarly, we can define a subfamily of a given family as a DFOM with a superset of the original DFOM's constraints. Thus, family membership is a well-defined relationship, which can be tested by comparing DFOMs. This is much easier than comparing geometric representations of realizations, which is difficult due to the persistent naming problem.
In Fig. 6, two realizations of a DFOM created in SPIFF, a prototype modeling system developed at Delft University of Technology, is shown. The user can manually choose one of the realizations, or add constraints to reduce the number of realizations. If a constraint is added to the blind hole that specifies that the hole may not be obstructed, then only realization (a) will be found, because in model (b) the volume of the blind hole is partially obstructed.

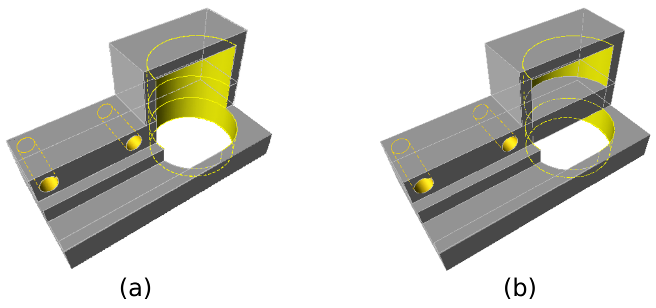


Fig 6. Two realizations of a single DFOM in SPIFF.

## 4. FACILITIES TO SUPPORT MODELING FAMILIES OF OBJECTS

Current modeling systems do not provide adequate tools for modeling and using families of objects. A family, consisting of a potentially infinite number of objects, is an abstract concept that is difficult to visualize and interact with in a natural way. We investigate these problems in Section 4.1. To relieve these problems, new modeling tools are needed. A useful tool when instantiating members of a family, is one that computes the range of allowable values for a parameter. It can also be useful to know the parameter values for which topological changes occur in the model, i.e. the critical parameter values. Tools for computing parameter ranges and critical values are discussed in Section 4.2.

### 4.1 Interactive Modeling of Families of Objects

For modeling families of objects, interactive modeling tools are needed to define the family, and interactive exploration tools are needed to inspect the set of family members.
In current CAD systems, a family of objects is created by first modeling a prototype object. The prototype object can be interactively edited by selecting entities in the b-rep, by adding and removing features in the modeling history and by changing parameter values, e.g. dimensions.

Other family members are instantiated, e.g. in an assembly model, by re-evaluating the modeling history with new parameter values.
One problem with this approach is that, because the prototype must always be a completely determined object, the designer is forced to make choices, such as in which order to add features to the modeling history, and which values to set for parameters. However, as requirements are often refined or changed during the design process, it may be necessary to undo previous choices, e.g. by changing dimension values or by removing features, to satisfy the new requirements. As we have seen in Section 2.1, changes to the modeling history are not intuitive and can result in errors. Also, alternative design solutions may be missed because of choices made early in the modeling process.
For declarative models, the design process can be thought of as a gradual narrowing down of a broad family of objects to a smaller family of objects, by adding or changing requirements, specified in the model using constraints. No arbitrary design choices have to be made to create a single object, until the last moment, e.g. just before analysis or manufacturing. Thus, no potential solutions are discarded during the modeling process. However, in declarative models, geometry and topology may not be fully determined, and there is no obvious and meaningful way that these aspects can be visualized and interacted with. Thus, in practice, first a member model must be instantiated (by specifying additional constraints or parameter values), which can then be visualized and interacted with. Fortunately, in declarative systems, features and constraints can be removed without the limitations imposed by a fixed modeling history, and thus changes in requirements can be incorporated at any time, by mapping operations on an arbitrary member model to operations on the family model.
Thus, in both history-based and declarative systems, a family model is edited by interacting with a single family member (i.e. the prototype or an arbitrary one, respectively). The effects of the edit on other family members, however, may be not immediately obvious to the user. A system for modeling families of objects should therefore provide tools that allow the user to inspect the set of objects in a family. Tools from declarative scene modeling systems ([5,7]) for exploring families of objects are useful for this, e.g. the possibility to visualize several members at the same time. However, to explore a family in this way, members of the family have to be instantiated. This can be problematic, because it is not a priori known which parameter values will result in valid family members. This is, in particular, difficult to determine for complex models, and models that have been created by a third party. Also, because the set of family members is often infinite, not all members can be inspected, and it will be difficult to get an overview of the modeled family. In particular, members with undesirable topological properties may exist that are hard to find by manually instantiating members.
For instantiating members of a family, it is therefore useful to know the range of allowable parameter values, i.e. the parameter range. Also, it is useful to know the parameter values for which topological changes occur, i.e. the critical values, because this allows the user to explore the topological variations of a family. Methods to compute parameter ranges and critical values are discussed next.

## 4.2 Parameter Ranges and Critical Values

In [23], the parameter range problem is considered for a 2D polygon with only horizontal and vertical line segments and distance constraints between them. A method is presented there to determine the range for a single distance parameter such that the topology of the polygon does not change. The considered constraint system is very simple, but the authors do make some useful observations concerning the problem in general. In particular, they observe that a system of geometric constraints in general has a large number of solutions, exponential with respect to the number of constraints in the problem, and that a different parameter range may be found for each solution. Also, they suggest that only one parameter at a time should be considered, because the combined parameter range for *n* parameters is a subset of *n*-dimensional space that will be very difficult to determine and to present to the user.
The parameter range problem is considered for systems of geometric constraints in [24]. This approach can be used to find an interval for a parameter of a system of constraints such that a solution is feasible. However, it cannot deal with parameter ranges that consist of several disjoint intervals, and, because it is based on sampling, it cannot determine the exact bounds.
In [14], we presented a method to determine the exact range of any single parameter of a system of geometric constraints. The method computes the range for a single parameter, referred to as the variant parameter. The range to be computed is a set of intervals such that for any value in these intervals, a solution of the system exists.
The considered constraint problems are systems of distance and angle constraints on points in 2D or 3D that are well-constrained. It is assumed that a geometric constraint solver is available that can find a decomposition of the system into subproblems, and the solutions for each subproblem and the system as a whole.
Basically, the method first determines all "geometrically" critical values of the variant parameter. A critical value of a parameter is here defined as any value $c$ for which there is an arbitrarily small value $\varepsilon$, with $|\varepsilon| > 0$, such that for $c$ and $c + \varepsilon$ the system has a different number of solutions. The idea behind this approach is that the solvability of the system (i.e. whether the system can be solved,

using a particular constraint solver) can only change at critical values, and not between two subsequent critical values.

Critical parameter values are related to degenerate subproblems. A subproblem is degenerate, for some parameter value, if for some arbitrarily close value it has a different number of solutions. For each subproblem that is dependent on the variant parameter, several degenerate cases may exist, i.e. there can be several ways in which the subproblem degenerates. The degenerate cases of the simplest possible subproblem, a triangle *ABC* with two three distances, *AB*, *AC* and *BC*, are illustrated in Fig. 7. The triangle *ABC* exists only if the triangle inequality is satisfied, i.e. if $AC + BC \quad AB \quad |AC - BC|$. If the distance *AB* is the variant parameter, and $AC =$ ۴ and $BC =$ ۳, then there are two critical values: $AB =$ ۱ and $AB =$ ۷. For these values, *ABC* degenerates to a configuration of three points on a line, as shown in Fig. 7(b) and Fig. 7(c).

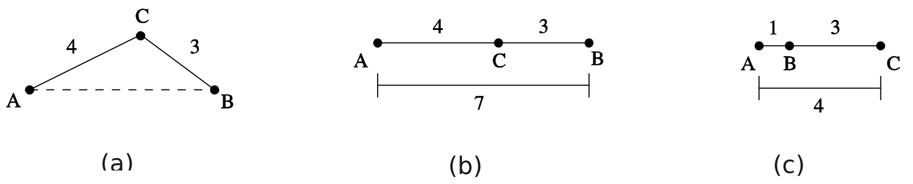


Fig. 7. Degenerate cases for a triangular subproblem *ABC*, with variant parameter *AB*.

In general, finding critical values is achieved by first removing the constraint corresponding to the variant parameter from the constraint system. Then, for each dependent subproblem, constraints are added such that the subproblem degenerates, resulting in a new system of constraints for each case. Each such system is solved, and the critical values are determined by measuring the value of the variant parameter in the solutions of each system.

To determine the parameter range from the critical values, the method determines the solvability of the system for each interval between two subsequent critical values, by picking a value in each interval and solving the system with that parameter value. The parameter range is the set of intervals for which solutions can be found for the system.

In [16], we presented a method to track topological changes in a parametric model, such that the “topologically” critical parameter values and parameter ranges can be determined.

A critical value of the variant parameter is here defined as any value $c$ for which there is an arbitrarily small value $\varepsilon$, with $|\varepsilon| >$ ۰, such that for $c$ and $c + \varepsilon$ the realizations of the model have different topologies. Critical values thus correspond to changes in the topology of the realizations of a model, when a parameter is varied continuously.

Fig. 8. shows the realizations of a model for different critical parameter values. Critical values often correspond to features that are tangent, as indicated by the arrows in the figure.

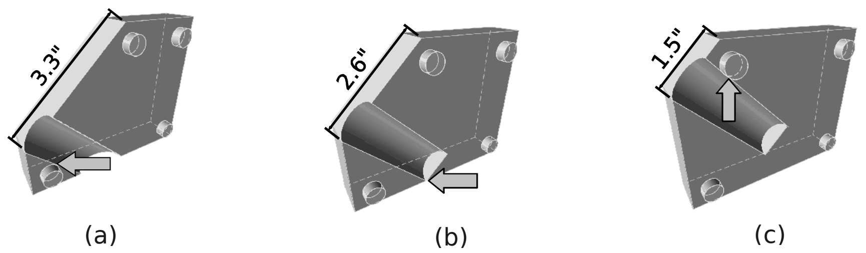


Fig. 8. Critical values for an example model. Arrows show where features are tangent.

Critical values are related to degenerate entities, i.e. entities that should not occur in the geometric representation of a model. Examples of degenerate entities are edges of zero length, faces with zero area, and cells with zero volume. Such entities should be represented by lower-dimensional entities instead. Entities that represent disjoint point sets, or point sets that can be decomposed into point sets of different dimension, should be split into several entities, and are therefore degenerate too.

Although degenerate entities should never occur in a geometric representation, we can impose constraints on the carriers that determine an entity such that it degenerates if the constraints are satisfied. For each entity in the geometric representation, one or more degenerate cases can be formulated in terms of geometric constraints. For example, a vertex, determined by the intersection of two lines, degenerates when the intersection of the two lines no longer exists. This corresponds to a constraint specifying that the lines should be parallel.

The basic approach for computing critical values is as follows. First we remove the constraint corresponding to the variant parameter, i.e. it is considered to be a variable without a fixed value. Effectively, a constraint will be removed from the geometric constraint system. We then determine which entities were dependent on the variant parameter and may therefore degenerate. For each degenerate case of each entity, we add specific constraints to the system to enforce the

degeneration. By solving the modified system, we obtain values of the variant parameter for which an entity degenerates, and thus the topology of the model must change, i.e. critical parameter values. By repeating this process for every entity that is dependent on the variant parameter, we can obtain all critical values.
The critical values of a variant parameter can be used to determine the parameter range. To do this, we first determine the geometric parameter range, including the “geometrically” critical values, using the methods from [14]. Then we determine the critical parameter values corresponding to topological changes, as discussed above. When all critical values have thus been determined, then for each interval between two subsequent critical values, we pick a parameter value in that interval and we regenerate the CM for that value. For this CM, we test whether the topological constraints in the model are satisfied. If so, then the whole interval is part of the parameter range. Each interval between two subsequent critical values can thus be marked as part of the parameter range, or not part of the parameter range. For the critical values, we can also determine whether the topological constraints in the model are satisfied. Thus, we have exactly determined for which values and intervals the geometric and the topological constraints are satisfied.

## 5. DIRECTIONS FOR FUTURE RESEARCH

We consider the declarative approach to modeling families of objects preferable over history-based parametric models. The declarative approach makes it possible to specify and maintain semantics for families of objects in a proper and intuitive way. Also, methods for computing parameter ranges and critical values exist only for declarative models. However, current declarative models are still too limited in the shape domain and types of constraints needed for practical modeling. In this section, we first discuss the limitations of, and possible extensions to declarative models, in particular the DFOM. Secondly, we propose a general declarative framework that can be used for a variety of geometric modeling applications, e.g. product synthesis, scene modeling and architectural design.

### 5.1 Current Limitations and Possible Extensions

Current declarative models are often limited in variety of shape that can be modeled, and in the types of constraints that can be imposed to parameterize a shape. For example, the DFOM currently only supports planar, spherical and cylindrical carriers. The implementation can be easily extended to support other regular geometric primitives, e.g. cones and tori. However, supporting freeform geometry, e.g. NURBS surfaces, is more problematic. In particular, no good methods are known for solving geometric constraints on NURBS and other parametric surfaces.
More advanced visualization and exploration tools are needed to create and use families of objects in an intuitive way. The prototype implementation of the DFOM allows users to cycle through a finite number of realizations. An improvement would be the possibility to visualize several members at the same time. However, for models with an infinite set of realizations, more advanced techniques are needed. For example, critical values corresponding to topological changes may be used to display a set of key objects that give a good overview of the variability in the family. Also, it may be possible to use parameter ranges and critical values to visualize the degrees of freedom of parts of a model.
Current methods for parameter range computation consider only one variant parameter. The parameter range corresponding to valid models when two parameters, or perhaps three parameters, are varied simultaneously, may also be useful for a designer, since such a range can still be presented to a user graphically. Considering even more parameters simultaneously may be useful for model optimization problems, since parameter ranges can be used to reduce the search space of such problems.
New types of semantics are also needed for CAD, for example, to specify patterns with a variable number of features and constraints. As an example, suppose that we wish to model a family of ball bearings. The balls are arranged in a circular pattern around the axle, inside the bearing (see Fig. 9). Given the diameter for the axle and the diameter for the bearing, we can determine the size of the balls and the maximum number of balls that fit in the bearing. Or, given the size of the axle and the number of balls, we can determine the maximum size of the balls and the diameter of the bearing. In current parametric systems, such relations can be manually programmed for some models, but the relations cannot be maintained when changing the parametric definition or when directly editing the model. A declarative way of specifying such relations, e.g. specifying a pattern of “a number of balls in a circular arrangement”, makes it possible to maintain these relations when the model is changed, i.e. when adding or removing constraints and/or objects. With such new semantic concepts, more complex models can be specified more easily than is possible in current CAD systems.

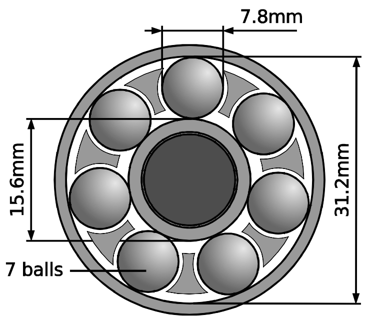


Fig. 9. Parameterization of a ball bearing.

## 5.2. A Declarative Framework for Geometric Modeling

So far, in this paper, we have mostly considered CAD applications. However, in basically all geometric modeling applications there is a need to specify and maintain semantics. We therefore envision a generic declarative framework for geometric modeling that allows semantics to be specified for a variety of applications. The framework consists of a constraint-based formalism for specifying semantics, and generic, extendable methods to analyze and solve systems of constraints. Application-specific ontologies can be defined to map semantic models in a particular application domain to the constraint-based formalism. Application-specific solving strategies can be defined to improve the efficiency of constraint solving for a particular application domain. Modeling systems and tools built on top of the framework allow the user to interact with a model of which the semantics is automatically maintained (see Fig. 10).

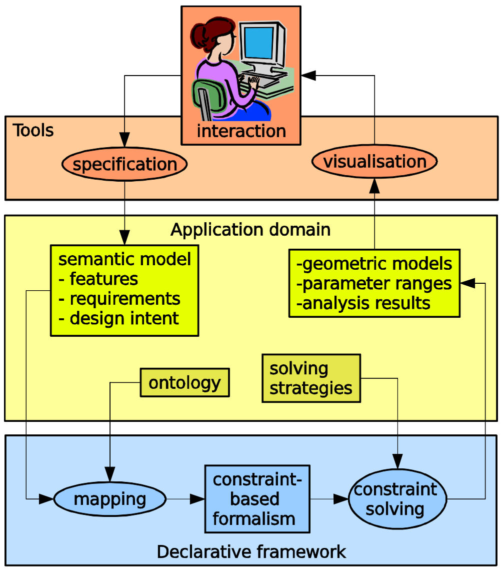


Fig. 10. Information flow in the declarative framework for geometric modeling.

The constraint-based formalism must define a compact but expressive vocabulary of variable types and constraint types for specifying semantics, including, but not limited to:

- logical, algebraic, geometric and topological variables and constraints
- patterns with an unknown number of variables and constraints
- ordering and symmetry definitions.

With this vocabulary, application-specific semantics, i.e. concepts such as features, functional requirements and design intent, can be defined in ontologies. For example, the concept of a through-hole feature may be mapped to constraints that specify that the top and bottom faces of the feature must not be present in the geometric model. High-level concepts such as “a set of features in a circular arrangement” can be defined using patterns of variables and constraints. Also, concepts that involve ordering or symmetry can be defined using this formalism. For example, a set of features may be ordered such that the area of the features is increasing, and one or more symmetry planes of a feature may be defined.
For solving the systems of constraints in this formalism, we need a solving approach that combines different domain-specific solvers, e.g. based on [6]. The solving method should include solvers for very general problem domains, e.g. symbolic algebraic, discrete and numerical solvers, as well as

more efficient solvers for specialized problem domains, in particular the geometric and topological domains. The envisioned constraint solving approach will be extendable with application-specific solving strategies that determine which solvers should be used, so that problems that occur frequently in a particular application can be solved efficiently.
Tools built on top of the framework provide 3D visualization and specification mechanisms, i.e. graphical user interaction. The tools can make use of the framework to specify semantic models and to determine geometric models that satisfy the semantics. The validity of models can be automatically maintained when interactively changing elements of the model, by letting the constraint solvers adjust other elements in the model to re-satisfy any unsatisfied constraints. By solving constraints, it is also possible to synthesize geometric models and optimize existing models from requirement specifications [24]. Systems of constraints can be analyzed using the constraint solvers to determine properties of the family of objects, e.g. parameter ranges and critical parameter values.
Finally, various tools built on top of the framework can exchange semantics, because semantics can be mapped to the shared constraint-based formalism. This makes it possible to work with a model in one tool while maintaining the semantics specified for that model in another tool. For instance, dimensions and functional requirements of a part can be specified in a CAD system, while an assembly planning tool may generate mating and fitting conditions for the part in a larger product, and ordering conditions that reflect the assembly sequence. Altogether, we believe that such a declarative approach to geometric modeling can solve many, long standing issues in CAD and other applications, and should be researched and developed further.

## 6. ACKNOWLEGEMENTS

Hilderick A. van der Meiden's research has been supported by The Netherlands Organisation for Scientific Research (NWO).

## 7. REFERENCES

[1] Bettig, B.; Bapat, V.; Bharadwaj, B.: Limitations of parametric operators for supporting systematic design. In: CDROM Proceedings DETC-2005, ASME International Design Engineering Technical Conferences, September 24-28, Long Beach, California, USA. ASME, New York, NY, USA, 2005.
[2] Bidarra, R.; de Kraker, K. J.; Bronsvoort, W. F.: Representation and management of feature information in a cellular model, Computer-Aided Design, 30(4), 1998, 301–313.
[3] Bidarra, R.; Bronsvoort, W. F.: Semantic feature modeling, Computer-Aided Design, 32(3), 2000, 201–225.
[4] Bidarra, R.; Madeira, J.; Neels, W.; Bronsvoort, W. F.: Efficiency of boundary evaluation for a cellular model, Computer-Aided Design, 37(12), 2005, 1266–1284.
[5] Bonnefoi, P.-F.; Plemenos, D.; Ruchard, W.: Declarative modeling in computer graphics: current results and future issues. In: Bubak, M. (ed.): Proceedings ICCS 2004, International Conference on Computational Science, June 6–9, Krakow, Poland, Volume 3039 of Lecture Notes in Computer Science, pages 80–89, Springer, Berlin, Germany, 2004.
[6] Borning, A.; Freeman-Benson, B.: Ultraviolet: A constraint satisfaction algorithm for interactive graphics, Constraints 3(1), 1998, 9–32.
[7] Gaildrat, V.: Declarative modelling of virtual environments, overview, issues and applications. In: Plemenos D. and Miaoulis, G. (eds.): Proceedings 3IA'2007, International Conference on Computer Graphics and Artificial Intelligence, May 30–31, Athens, Greece, Volume 10, Pergamon Press, Elmsford, NY, USA, 2007.
[8] Hoffmann, C.M.: Constraint-based computer-aided design, Journal of Computing and Information Science, 5(3), 2005, 128-187.
[9] Hoffmann, C. M.; Joan-Arinyo, R.: Parametric modeling. In: Farin, G.; Hoschek, J. (eds.): Handbook of Computer-Aided Design, pages 519–541, Elsevier Science, The Netherlands, 2009.
[10] Hoffmann, C. M.; Kim, K.: Towards valid parametric CAD models. Computer-Aided Design, 33(1), 2001, 81–90.
[11] Joan-Arinyo, R.; Mata, N.: Applying constructive geometric constraint solvers to geometric problems with interval parameters. Non-linear Analysis – Theory, Methods & Applications, 47(1), 2001, 213–224.
[12] Marcheix, D.; Pierra, G.: A survey of the persistent naming problem. In: Lee, K.; Patrikalakis, N. (eds.): Proceedings Solid Modeling '02 - Seventh Symposium on Solid Modeling and Applications, 17–21 June, Saarbrücken, Germany, pages 13–22, ACM Press, New York, NY, USA, 2002.
[13] Meiden, H.A. van der: Semantics of Families of Objects, PhD Thesis, Delft University of Technology, 2008, http://graphics.tudelft.nl/~rick.
[14] Meiden, H. A. van der; Bronsvoort, W. F.: A constructive approach to calculate parameter ranges for systems of geometric constraints, Computer-Aided Design, 38(4), 2006, 275–283.
[15] Meiden, H. A van der; Bronsvoort, W. F.: Solving topological constraints for declarative families of objects, Computer-Aided Design, 39(8), 2007, 652–662.

[16] Meiden, H. A. van der; Bronsvoort, W. F.: Tracking topological changes in feature models. In: Lévy, B; Manocha, D. (eds.): Proceedings ACM Symposium on Solid and Physical Modelling, June 4–6, Beijing, China, pages 341–346, ACM Press, New York, NY, USA, 2007.

[17] Meiden, H. A. van der; Bronsvoort, W. F.: Solving systems of geometric constraints using non-rigid clusters. In: Chen, F.; Jüttler, B. (eds.): Advances in Geometric Modelling and Processing, Proceedings GMP Conference, April 23–25, Hangzhou, China, Volume 4975 of Lecture Notes in Computer Science, pages 423–436, Springer, Berlin, Germany, 2008 [an extended version has been accepted for publication in Computer-Aided Design, DOI: 10.1016/j.cad.2009.03.003].

[18] Raghothama, S.: Constructive topological representations. In: Kobbelt, L.; Wang, W. (eds.): Proceedings ACM Symposium on Solid and Physical Modeling, June 6–8, Cardiff, Wales, UK, pages 39–51, ACM Press, New York, NY, USA, 2006.

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

[18] Raghothama, S.; Shapiro, V.: Topological framework for part families, Journal of Computing and Information Science in Engineering, 2(4), 2002, 246–255.

[21] Rappoport, A.: The Generic Geometric Complex (GGC): a modeling scheme for families of decomposed pointsets. In: Hoffmann, C. M.; Bronsvoort, W. F. (eds.): Proceedings Solid Modeling ’97, Fourth ACM Symposium on Solid Modeling and Applications, May 14–16, Atlanta, Georgia, USA, pages 19–30, ACM Press, New York, NY, USA, 1997.

[22] Shapiro, V.; Vossler, D. L.: What is a parametric family of solids? In: Hoffmann, C.M.; Rossignac, J. R. (eds.): Proceedings of the Third ACM/IEEE Symposium on Solid Modeling and Applications, May 17–19, Salt Lake City, Utah, USA, pages 43–54, ACM Press, New York, NY, USA, 1995.

[23] Stiny, G.; Gips, J.: Shape grammars and the generative specification of painting and sculpture. In: Petrocelli, O. R. (ed.): The Best Computer Papers of 1971, pages 125–135, Auerbach, Philadelphia, 1972.

[24] Summers, J.D.; Bettig, B.; Shah, J.: The Design Exemplar: a new data structure for embodiment design automation, Journal of Mechanical Design, 126(5), 2004, 775-787.