

Solving topological constraints for declarative families of objects

Hilderick A. van der Meiden
Willem F. Bronsvort

Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, NL-2628 CD Delft, The Netherlands
email: H.A.vanderMeiden/W.F.Bronsvort@ewi.tudelft.nl

May 31, 2007

Abstract

Parametric and feature-based CAD models can be considered to represent families of similar objects. In current modelling systems, however, the semantics of such families are unclear and ambiguous.

We present the Declarative Family of Objects Model (DFOM), which enables to adequately specify and maintain family semantics. In this model, not only geometry, but also topology is specified declaratively, by means of constraints. A family of objects is modelled by a DFOM with multiple realisations. A member of the family is modelled by adding constraints, e.g. to set dimension variables, until a single realisation remains. The declarative approach guarantees that the realisation of a family member is also a realisation of the family.

The realisation of a family member is found by solving first the geometric constraints, and then the topological constraints. From the geometric solution, a cellular model is constructed. Topological constraints indirectly specify which combinations of cellular model entities are allowed in the realisation. The system of topological constraints is mapped to a Boolean constraint satisfaction problem. The realisation is found by solving this problem using a SAT solver.

Keywords: parametric and feature-based modelling, families of objects, declarative specification, topological constraints, SAT

1 Introduction

Parametric and feature-based modelling systems are used to create object models with a number of parameters. The set of objects that can be obtained by varying the parameters of a model, is often referred to as a family of objects, or family for short. Virtually all systems store a history of modelling

operations on a B-rep. We refer to these systems as history-based modellers and to their models as history-based models. Typically, the procedure for modelling a family of objects, is to initially model a single object, called the prototype or generic object. The history of modelling operations, used to construct the prototype object, is re-evaluated with different parameter values to generate other members of the family.

History-based models, even though they are the de-facto standard for commercial modelling systems, are not ideal for representing families of objects. Bidarra and Bronsvoort [1] identify six major problems with history-based models, 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 properly maintain feature semantics. The persistent naming problem basically is the problem of identifying topologically equivalent entities in different members of a family. This is a prerequisite for maintaining family semantics. Previous research on families of objects has focussed mainly on the persistent naming problem, see, for example, [2, 3, 4].

The feature ordering problem and maintenance of feature semantics have not received much attention. The order in which features are added to history-based models affects the resulting family of objects. This makes it difficult to design and edit family models. Also, the history-based modelling scheme does not have mechanisms for adequate specification and maintenance of semantics of features, and thus the resulting families have unclear semantics. In particular, topological properties cannot be adequately specified and maintained. These problems are discussed in more detail in Section 2.

We present a new model for families of objects, the Declarative Family of Objects Model (DFOM). In this model, both geometry and topology are specified declaratively. Declarative specifications state properties of objects, typically by means of constraints, but not how to construct those objects. Procedural specifications, on the other hand, specify how to construct objects, but there is no guarantee that any property generically holds for those objects. For modelling families of objects, however, the ability to specify generic properties is essential.

Declarative specification of geometry using constraints is common practice in current modelling systems, but topology is practically always specified in a procedural way [5]. Features in history-based systems correspond to set operations, or other operations that procedurally manipulate entities in a B-rep. These operations may change topological properties of the B-rep, even when this is not desired. In our new model, topological properties are declaratively specified by topological constraints that must hold for all members of a family. These constraints are imposed on topological entities that are either explicitly defined by features in the model, or implicitly by the interaction of features in the model. A topological constraint may specify, for example, that a face of a feature must be on the boundary of the model,

or that a feature may not be split into disjoint volumes. To find an explicit topology, the system of topological constraints is solved.

The complete system of constraints to be solved thus consist of geometric and topological constraints. Although geometry and topology are closely tied, in our approach they can actually be treated separately. Geometric constraint solving is a well-developed field and is not further addressed here. For an overview of recent work we refer to [6]. Solving topological constraints on feature models, to the best of our knowledge, has not been addressed before.

In Section 2 we discuss previous work on families of objects. Section 3 is dedicated to the Semantic Feature Model, which is the basis for the DFOM. The DFOM itself is presented in Section 4. Before solving, topological constraints are mapped to Boolean constraints, which is elaborated in Section 5. Section 6 discusses the technique used for solving Boolean constraints, and its performance. The DFOM has been implemented in a prototype feature modelling system; some aspects of this are presented in Section 7. Finally, we draw some conclusions in Section 8.

2 Previous work

Parametric and feature-based models can be thought of as dual-representation schemes [7], consisting of a parametric representation, e.g. a CSG representation, and a geometric representation, e.g. a B-rep. 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 parameter vectors that correspond to valid models. The representation-space family corresponds to the set of all objects that can be obtained by certain operations on the geometric representation. The two families are interrelated by procedures and constraints that relate parameters to geometry, but this relation is not well understood.

A specific representation-space family is described by the concept of boundary representation deformation [3]. 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, because the boundary representation deformation cannot account for splitting and merging of entities. A more general definition is proposed in [8]; a family may be considered equivalent to a category as defined by category theory (which allows to describe and compare the semantics of broad classes of mathematical objects, such as the category of sets and the category of topological spaces). Part families are defined as sub-categories of the category of cell-complexes, but an application of this theory to engineering practice is yet to be presented.

Suggested in [2] are two possible models for families of objects: the clas-

sifying set model and the parametric set model. Here, a model is defined as a representation of a family of objects that we wish to query and perform operations on. The classifying set model implements a set-membership query, which, given a family model and an object model, answers whether the object is a member of the family. The parametric set model requires that member instances can be identified by a parameter vector. The parametric access query, given a family model and a parameter vector, returns a member model. A parametric set model for representing families of decomposed point sets is the Generic Geometric Complex (GGC) [2]. The GGC represents families of Selective Geometric Complexes (SGCs). A SGC [9] represents an object by carriers, which are n-dimensional algebraic or parametric geometries, and by entities, which are disjunct point sets obtained from intersections of carriers. The GGC represents a family of SGCs with the same carriers and a common subset of entities, marked as essential. Unfortunately, the model does not allow complex patterns of entities to be included in or excluded from the family, so the class of families that can be modelled by the GGC is rather limited.

The models described above largely overcome the persistent naming problem, but other shortcomings of the history-based approach with respect to modelling families of objects, in particular the feature ordering problem and the problem of maintaining feature semantics, are not addressed.

The feature ordering problem manifests itself when instantiating new members of a family. Features, as they are implemented in history-based modelling systems, are essentially higher-level modelling operations, which are part of the modelling history. When modelling a prototype object, the order of modelling operations that seemed appropriate for that particular family member, may not yield the expected result when re-evaluating the history to create other family members. Consider, for example, Figure 1. A prototype object is first modelled. It consists of a base block, a block protrusion feature and a blind hole feature, as shown in Figure 1a. 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 block protrusion was created before the blind hole, the model shown in Figure 1b emerges. If, however, those features were created in the reverse order, the model shown in Figure 1c emerges. The implication of the feature ordering problem is that when modelling a family of objects, the order of operations must be taken into consideration, even though the effect may not be visible in the prototype. This complicates design and editing of family models, in particular models with many interacting features.

Maintaining feature semantics means that features must satisfy certain predefined properties, in particular, specific topological properties, in the context of any model. Due to interaction with other features, however, the topological properties of a feature may change. For example, Figure 2(a)

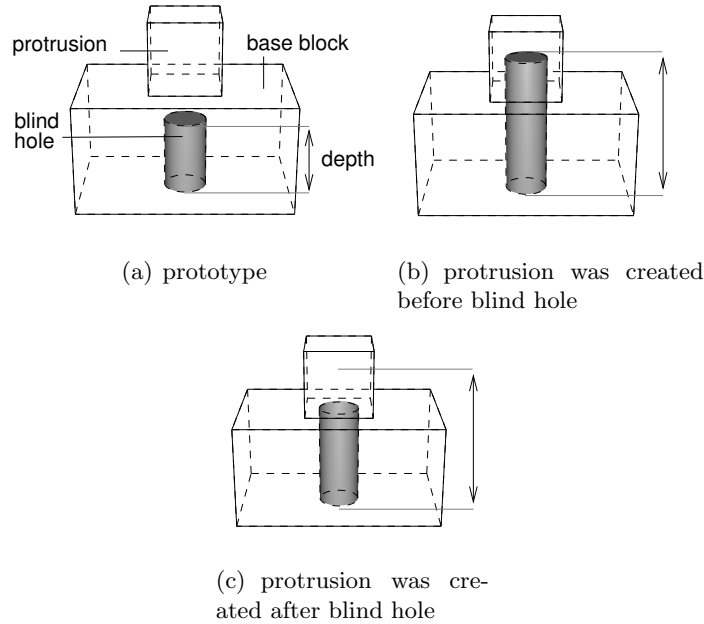


Figure 1: Example of the feature ordering problem. When the depth of the blind hole feature in the prototype (a) is increased, either model (b) or model (c) may emerge, depending on the modelling history.

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 Figure 2(b), the blind hole feature does cut through the object, thus the semantics of the feature has changed. If the semantics of one or more features changes, the model should be marked as invalid. History-based systems, however, only check the validity of a feature during instantiation of the feature in the model. If, due to interaction with other features, the semantics of a feature changes at later stages of the evaluation of the modelling history, this is not detected. In particular, topological properties of features cannot be adequately specified and maintained, because only the result of feature operations is stored in a B-rep, and insufficient feature information is kept. Maintaining feature semantics is essential for families of objects, because the semantics must be the same for all members of a family. If feature semantics is not properly maintained, the family may contain invalid models.

By using a limited set of feature classes and strict adherence to proven modelling practice, undesirable situations as described above can sometimes be avoided. However, this practice has also made problems with history-based models more obscure and unpredictable.

An alternative to history-based models for defining families of objects

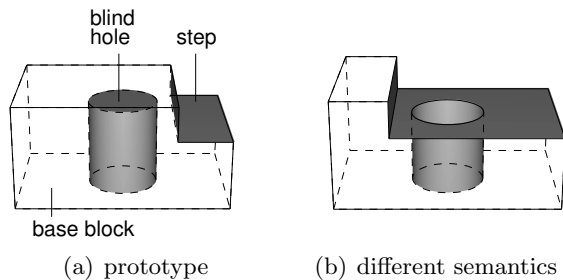


Figure 2: The semantics of a blind hole requires that the hole has a bottom face, as in the prototype (a). When the step feature is changed as in model (b), the blind hole is changed into a through hole.

is the Semantic Feature Model, which is defined only by a set of features and constraints, and keeps no record of modelling history. The work presented here is based on the Semantic Feature Modelling approach, which is elaborated in the following section.

3 The Semantic Feature Model

The Semantic Feature Model (SFM), introduced by Bidarra and Bronsvoort [1], is a declarative model, which allows feature semantics to be adequately specified and maintained. A SFM consists of a set of features and additional constraints between features. The shape and position of all features is determined by solving the constraints specified in the features (feature constraints), and the additional constraints between features (model constraints).

Each feature is instantiated from a feature class. A feature class consists of shape elements, positioning elements, topological elements, and a user interface, as shown in Figure 3. The user interface of a feature class contains parameters that control the shape and position of the feature. In particular, *shape parameters* determine the shape of the feature, via *shape constraints*, which are imposed on *shape entities*. *Positioning parameters* are parameters of *positioning constraints*, which are imposed on shape entities and on *attach faces* and *positioning faces*, i.e. faces of other features in the model or reference geometry, set by the user.

The topological elements of a feature class are its nature attribute, boundary constraints and interaction constraints. The *nature* attribute can be *additive* or *subtractive*, indicating whether the feature adds or removes material. A boundary constraint is associated with a feature face, and specifies that the face must be (partially) on the boundary of the model, or may not be on the boundary of the model. Interaction constraints are associated with a feature as a whole. Interactions with other features may create

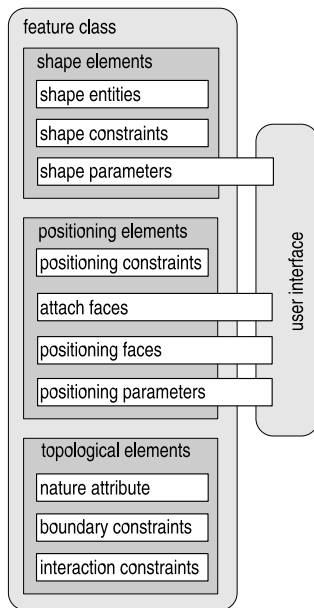


Figure 3: Elements of a feature class definition.

specific topological patterns, which can be disallowed by these constraints. Table 1 lists interactions commonly found in feature models [1].

The geometric representation is the *cellular model* (CM), a cell-complex representation that can be used to store semantic feature information [10]. The cellular model represents topological entities, i.e. vertices, edges, faces and cells, and all incidence 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 modelled object, or it represents an empty volume, i.e it is not part of the object.

The CM is constructed by combining all the features shapes in the model, and can be updated efficiently when the feature model is changed [11]. If features intersect, they 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 *dependency analysis*. The positioning faces and attach faces specified in the user interface determine dependency relations among features. If feature F_1 refers

to one or more faces of a feature F_2 , then F_1 is said to be *directly dependent* on F_2 . These relations are represented by a *dependency graph*, which is a directed graph, where every direct dependency of a feature F_1 on a feature F_2 is represented by an edge (F_1, F_2) . In general, a feature F_x is said to be dependent on a feature F_y if there is a path from F_x to F_y in the dependency graph. Feature precedence is a partial ordering derived from the feature dependency graph, as follows: if a feature F_x depends on a feature F_y , then F_x precedes F_y . For each cell in the CM, a precedence order is determined for the features in the owner list of the cell. The nature of the feature with the highest precedence determines whether the cell contains material. If the nature of that feature is additive, the cell contains material. If the nature of that feature is subtractive, the cell does not contain material.

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

A model for families of objects based on the SFM, is the Semantic Model Family [12]. A family consists of all models with the same features and constraints, but different parameter values. Boundary and interaction constraints guarantee that every member of the family has valid feature semantics. However, families may be ambiguous, meaning that for some parameter values the resulting member is not well defined. In most cases, a feature's contribution to the model is correctly determined by dependency analysis. Problems occur when there are features in the owner list of a cell that are independent and have conflicting natures.

For example, Figure 4 shows the feature dependencies of the model in Figure 1. The block protrusion and blind hole features are both dependent on the base block, because they refer to it for positioning, but there are no dependencies between the two features. Thus no feature precedence can be determined for these two features, and again either Figure 1b or Figure 1c emerges, dependent on the order of feature creation, just like in history-based systems. Interestingly, the Semantic Feature Modelling approach can detect that the semantics of the blind hole feature in Figure 1c is incorrect, because the bottom of the blind hole in the cellular model does not correspond to the bottom of the blind hole in the feature definition (the boundary constraint on this feature face is not satisfied). However, this information is not used to determine the correct feature precedence order.

The main shortcoming, more in general, 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. More importantly, the analysis is not directly related to 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

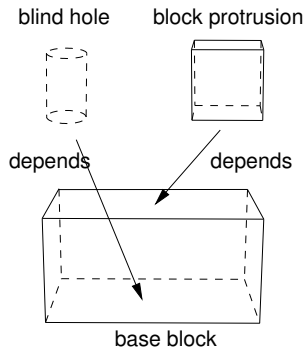


Figure 4: Feature dependencies for the model in Figure 1. The blind hole and the block protrusion depend on the base block, but are independent of each other.

result, a model that satisfies the topological constraints is not always found, even though such a model exists. This problem is resolved with the new model for families of objects that is presented in the next section.

4 The Declarative Family of Objects Model

The Declarative Family of Objects Model (DFOM) is a generalisation of the Semantic Model Family. The DFOM is similarly defined by a set of features, including feature constraints, and model constraints. The geometric representation is also the cellular model. However, whether the cells of the CM contain material is not determined by feature precedence, but by solving topological constraints. Another difference with the Semantic Model Family is that the DFOM can represent both families of objects and single objects. Subfamilies and family members are modelled by adding constraints to a family model, e.g. a constraint to set the value of a dimension variable. To determine whether a given DFOM is a subfamily or a member of the family defined by another DFOM, the features and constraints of the DFOMs are compared. Altogether, the DFOM can represent families of objects, in an unambiguous way.

A feature class defines a canonical shape, constraints, and a user interface. The canonical shape of a feature is represented by a canonical cellular model, referred to as the *feature CM*. The geometry of the entities in the feature CM is not completely determined. Only the type of the geometry is given, and the topological relations between the entities. For the cells, it is not yet specified whether they contain material. Geometric constraints are imposed on feature entities to determine their geometry and relative position and orientation. Topological constraints are imposed on a feature, to specify topological properties of the feature that must be maintained.

The user interface of a feature is the subset of the parameters of the

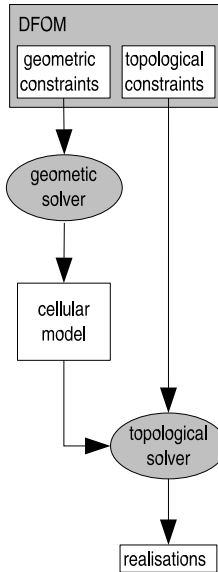


Figure 5: The interpretation of a DFOM consists of a geometric solving step, which yields a cellular model, and a topological solving step, which may yield one or more realisations.

feature constraints that is made available to the user. Interface parameters may be numeric parameters, e.g. distances and angles, or geometric parameters, e.g. curves and surfaces. The latter can be used, in combination with geometric constraints, to position and orient the feature relative to other features in the model.

Feature classes no longer specify a nature attribute, as was the case for the SFM. Instead, nature is implemented in the DFOM as a topological constraint that may be imposed on features. A constraint $\text{NATURE}(F, \text{ADDITIVE})$ or $\text{NATURE}(F, \text{SUBTRACTIVE})$ specifies that the nature of feature F is additive or subtractive, respectively. To further define feature semantics, other topological constraints can be imposed, in particular boundary constraints and interaction constraints.

A DFOM does not explicitly represent the geometry and topology of the model. Instead, one or more *realisations* may be derived from a DFOM by a process called *interpretation*. A realisation is a cellular model with a value assigned to each cell, specifying whether the cell contains material. The interpretation of a DFOM involves two solving steps. First, the geometric constraints in the DFOM are solved. From the geometric solution, a *combined cellular model* is constructed, containing all geometry of all features. Then, the system of topological constraints is solved. Each solution of the system is a realisation of the DFOM. This is illustrated in Figure 5.

Typically, if a DFOM represents a family of objects, the system of geo-

metric constraints will be underconstrained, i.e. it has one or more degrees of freedom. This occurs when, for example, some dimensions of some features have not been specified. A system that is underconstrained, has an infinite number of solutions, and these cannot be represented explicitly. Implicitly, the model represents all the realisations that would have been obtained if we could generate all the geometric solutions and interpret them further. If the geometric system has a finite number of solutions, then all the solutions can be generated explicitly and interpreted further. However, the number of solutions can be very large, exponential to the number of constraints in the system. It is desirable that the number of geometric solutions is kept low. Therefore, feature classes should be defined in such a way that if all parameter values are known, there is only one geometric solution [13].

When the geometric constraint system has been solved, the geometry of all feature entities has been determined. The feature entities are then merged into the combined CM. The combined CM can be generated efficiently by adding feature entities one at a time [11]. If the added entity intersects with other entities already in the combined CM, the intersecting entities are split into non-intersecting entities. For each entity in the combined CM, a list of references is kept to the original feature entities from which it was derived, referred to as the entity’s *owner list*. This allows us to map topological constraints on features to Boolean constraints on cells of the combined CM. For each cell in the combined cellular model, a Boolean variable represents whether the cell contains material or not (see Section 5 for details).

In the case of the NATURE constraint, the mapping is as follows. For each cell c , the set \mathcal{F} is determined, which is the set of those features in the owner list of c on which a NATURE constraint has been imposed. The dependencies between these features are analysed. This yields the set \mathcal{D} of *dominant features*, which are the features on which no other features in \mathcal{F} are dependent. The NATURE constraints of these features are dominant over the NATURE constraints of other features. Let \mathcal{N} be the set of NATURE constraint values (ADDITIVE or SUBTRACTIVE) associated with the features in \mathcal{D} . If \mathcal{N} contains only ADDITIVE, then the cell should contain material. This is mapped to a primitive constraint $c = \text{TRUE}$. If \mathcal{N} contains only SUBTRACTIVE, then the cell should not contain material. This is mapped to a constraint $c = \text{FALSE}$. If \mathcal{N} contains no value, or both values, then no constraint is imposed on c at this stage.

The model shown in Figure 6 represents a situation similar to Figure 1 in Section 2; a blind hole feature cuts through a base block, into a protrusion feature. The semantics of a blind hole requires that the bottom face of the hole is on the boundary of the model, expressed by a COMPLETELY-ONBOUNDARY constraint. Figure 6(b) shows that for one cell, the NATURE constraints cannot determine whether it should contain material. This cell corresponds to the intersection of the hole and the protrusion, which are independent of each other and have a different NATURE constraint. There-

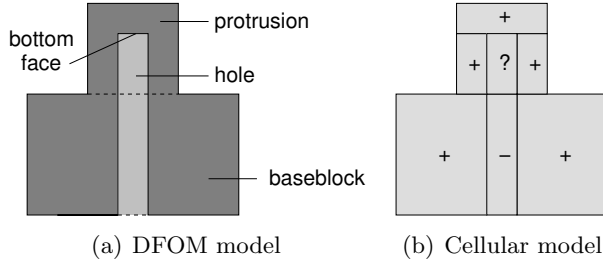


Figure 6: A 2D model illustrating the feature ordering problem, corresponding to Figure 1. In the cellular model, the value of cells marked with a sign are determined by NATURE constraints. Cells marked with '+' contain material, cells marked with '-' do not contain material. The value of the cell marked with '?' cannot be determined by NATURE constraints, and must therefore be determined by solving other topological constraints.

fore, the value of the cell can only be determined by solving other topological constraints. In this case, the cell should not contain material, because the bottom face of the hole must be on the boundary of the model, due to the COMPLETELYONBOUNDARY constraint.

In this example, and also in general, most cell values are determined by NATURE constraints. It therefore makes sense to first try to determine the values for cells carrying NATURE constraints. This can be done per cell, which is more efficient than considering all topological constraints and all cells at once. Usually, only a relatively small number of cells is involved in feature interactions, such that their value cannot be determined by NATURE constraints. For those cells, the other topological constraints in the model are solved (see Section 6).

The interpretation of the DFOM, as described above, guarantees that realisations satisfy all constraints, and therefore have the semantics specified by the model. A DFOM with more than one realisation thus represents a well-defined family of objects. However, because the set of realisations can be infinite, we cannot, in general, verify family membership by generating and comparing realisations. Therefore, family membership is defined differently, as follows.

A DFOM M represents a member of a family, represented by a DFOM F , if

- M contains the same set of features as F ,
- M contains a superset of the constraints of F , and
- M has exactly one realisation.

In other words, members of a family are instantiated by adding more constraints, until the number of realisations is just one. Subfamilies are defined

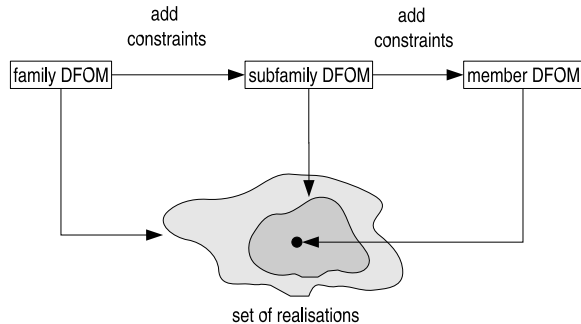


Figure 7: Specialisation and instantiation. By adding constraints to a DFOM, subfamilies and members are obtained. The realisations of a subfamily, and the realisation of a member, are in the set of realisations of the original family.

in a similar way: they are also represented by models with extra constraints, but can have more than one realisation. Thus, adding constraints to a DFOM is equivalent to specialisation of a family. Instantiation is essentially the same as specialisation; members are subfamilies with just one realisation. Models with no realisation at all are invalid.

The relation between DFOMs of the same family, and their realisations, is illustrated in Figure 7. To better understand this relation, consider that a subfamily, defined by a superset of constraints, always has a smaller (or equal) number of realisations than the original family, because more constraints have to be satisfied. Also, the set of realisations is a subset of the set of realisations of the original family, because all the constraints of the original family DFOM still have to be satisfied. A family member has only one realisation, which is thus guaranteed to be an element of the set of realisations of the family. On the other hand, we cannot guarantee that every realisation of a family DFOM has a corresponding member DFOM, because there may not be any combination of constraints that allows this realisation to be instantiated. Also, there may be a DFOM D_1 with one realisation, which is in the set of realisations of a given family DFOM D_f , and yet D_1 is not a member of D_f , because it contains different features or constraints. It should thus be kept in mind that family membership in terms of DFOMs is not equivalent to set membership for realisations.

To verify DFOM family membership, we need to be able to compare sets of features and constraints, and to determine whether a model has zero, one, or more realisations. Most geometric constraint solvers can identify overconstrained and underconstrained situations, corresponding to zero realisations and an infinite number of realisations, respectively. The topological constraint solver we use can generate the, always finite, set of all solutions, and thus we can determine the number of solutions. Sets of features and

constraints can easily be compared if features and constraints are uniquely identified by a name. This is only the case for models that were directly derived from a given family. In this case, verifying family membership is not interesting, because such models are by definition members of the family. To verify membership for models from another source, we can only consider the types of features and constraints, and the associations between them. In that case, the membership test is equivalent to graph matching.

Altogether, the DFOM described in this section allows families of objects to be specified with clear semantics. This semantics is declaratively specified by the user, thus families can be specified which include all, and only, the desired members. Family membership is well defined, and can be verified procedurally.

5 Mapping topological constraints

A realisation of a DFOM is a set of assignments, specifying for each cell in the combined CM whether the cell contains material, such that all topological constraints are satisfied. To find the realisations of a model, topological constraints are mapped to a system of Boolean constraints on the cells in the CM, and this system is then solved. The solutions of the system of Boolean constraints yield the realisations of the model.

All the cells in the combined CM are mapped to Boolean variables. If a variable is TRUE, then the corresponding cell contains material; if it is FALSE, the corresponding cell does not contain material. A Boolean constraint is a predicate on a set of Boolean variables, which may be expressed by a Boolean function $p(v_1, v_2, \dots, v_k)$, where $v_1 \dots v_k$ are Boolean variables. If p evaluates to TRUE, then the constraint is satisfied. If p evaluates to FALSE, then the constraint is not satisfied.

In this section we present the mappings for the boundary and interaction constraints introduced in Section 3; the interaction constraints were listed in Table 1. The mapping of the NATURE constraint has already been discussed in Section 4.

A COMPLETELYONBOUNDARY(f) constraint specifies that the feature face f must be completely on the boundary of the model. Because realisations specify only whether cells contain material, constraints on lower dimensional entities, in particular faces, must be expressed in terms of cells. We determine first the set of cell faces $G = \{g_1, \dots, g_k\}$ in the cellular model that overlap with f . Each cell face g_i has two adjacent cells, c_i^a and c_i^b . A cell face g_i is on the boundary if and only if exactly one adjacent cell contains material. To satisfy the constraint, all cell faces must be on the boundary. Thus, the constraint is expressed as follows:

COMPLETELYONBOUNDARY(f) =

$$\bigcap_{i=1..k} ((c_i^a \cup c_i^b) \cap \neg(c_i^a \cap c_i^b))$$

The mappings of the other boundary constraints, i.e. PARTIALLYONBOUNDARY, PARTIALLYNOTONBOUNDARY and COMPLETELYNOTONBOUNDARY, are similar to the mapping above.

Interaction constraints are imposed on features as a whole. In some cases, the mapping to Boolean constraints involves only constraints on the cells owned by the feature. This is the case for the obstruction interaction and absorption interaction constraints.

An obstruction interaction is defined as causing partial obstruction of the volume of a subtractive feature. An obstruction interaction constraint disallows this interaction. In other words: to satisfy an obstruction interaction constraint on a feature F , all cells c_1, \dots, c_k that have F in their owner list, may not contain material:

$$\text{NOOBSTRUCTION}(F) = \bigcap_{i=1..k} (\neg c_i)$$

An absorption interaction is defined as causing a feature to cease completely its contribution to the model. In that case, the feature does not contribute to the boundary of the model. Thus, an absorption constraint on a feature F is satisfied if any of the feature's faces $f_1 \dots f_k$ is at least partially on the boundary of the model:

NOABSORPTION(F) =

$$\bigcup_{i=1..k} \text{PARTIALLYONBOUNDARY}(f_i)$$

The splitting, disconnection and closure interaction constraints require that the connectedness of the cellular model can be expressed as a Boolean constraint. We formulate an expression TRUECONNECTED(G) of a graph $G = (V, E)$, where $V = \{v_1, \dots, v_n\}$ is a set of Boolean variables and E is a set of edges, representing adjacency of variables. This expression evaluates to TRUE if and only if all variables with the value TRUE are connected via edges of the graph.

The TRUECONNECTED expression is based on a simple propagation algorithm: first, a single variable that is TRUE is marked, and the mark is propagated to the adjacent variables that are TRUE. When no more marks can be propagated, either all TRUE variables have been marked, in which case the system is TRUECONNECTED, or some TRUE variables have not been marked, in which case the system is not TRUECONNECTED (see Figure 8).

Representing the propagation of marks in an expression requires that we model each mark in each iteration as a variable. For each $v_i \in V$, we define

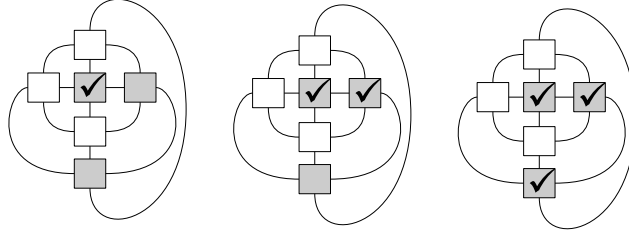


Figure 8: In each of the figures above, variables are represented by squares and adjacency information is represented by edges. Variables that are TRUE are indicated by gray squares. From left to right, marks are propagated to adjacent variables that are also TRUE.

mark-variables $m_{i,t}$, where $t \in \mathcal{N}$ is the iteration number. If and only if $m_{i,t}$ is TRUE, then variable v_i has been marked for iteration t . We know that the maximum number of iterations for spreading marks is equal to the number of variables in the adjacency graph, thus we need $|V|^2$ variables to represent the state of the propagation algorithm. For each node, and for each iteration, we must specify an expression representing the propagation of marks from the previous iteration:

$$m_{i,t} = v_i \cap \bigcup \{m_{j,t-1} | v_j \in Adj(v_i)\}$$

To begin propagation, a single node that evaluates to TRUE must be marked. However, when the expression is constructed, we cannot know which variables will evaluate to TRUE. We therefore order the variables and add constraints such that only the first node in the ordering that evaluates to TRUE is marked:

$$m_{i,0} = \bigcap \{\neg v_j | j < i\}$$

The TRUECONNECTED constraint is formulated as an expression that tests whether all variables v_i that are TRUE have been marked after the last iteration:

$$\text{TRUECONNECTED}(G) =$$

$$\bigcap_{i=1\dots n} \{v_i = m_{i,T}\} \quad T = |V|$$

Because this formulation of connectedness requires quadratic memory space, it should not be used to formulate connectedness for the whole model. It is currently only used for parts of the model determined by features with a splitting interaction or disconnection interaction constraint.

Splitting interaction is defined as an interaction that causes the boundary of a feature to be split into two or more disconnected subsets. A splitting interaction constraint disallows this interaction. To satisfy the constraint for a feature F , all cell faces that correspond to a face of F and that are on the

boundary of the model, must be connected. For each cell face $g_1 \dots g_k$ of each cell that has F in its owner list, we define a variable $v_i = \text{ONBOUNDARY}(g_i)$, $i = 1 \dots k$. A graph $G_{ob} = (V, E)$ is constructed, where $V = \{v_1, \dots, v_k\}$ and the edges E represent the adjacency between the corresponding cell faces. Then the spitting interaction constraint is simply:

$$\text{NOSPLITTING}(F) = \text{TRUECONNECTED}(G_{ob})$$

Disconnection interaction is defined as an interaction that causes the volume of an additive feature, or part of it, to become disconnected from the model. A disconnection interaction constraint disallows this interaction. In other words: all cells of a feature F that contain material, must be connected to the rest of the model. We construct a graph G_{adj} containing variables V_F for all cells which have F in their owner list, plus variables V_{adj} for the cells that are adjacent to V_F . There is an edge in the graph for each pair of adjacent cells, i.e. cells that share a cell face. The rest of the model is represented by a single variable v_{model} , and there is an edge in the graph between v_{model} and each $v \in V_{adj}$. To test for disconnection interaction, we impose a `TRUECONNECTED` constraint on this graph:

$$\text{NODISCONNECT}(F) = \text{TRUECONNECTED}(G_{adj})$$

A closure interaction is defined as causing some subtractive feature to become a closed void in the model. Combinations of subtractive features may form closed voids, whereas the features separately are not. A closed void is a set of empty cells which is surrounded by non-empty cells. A `NOCLOSURE` constraint is satisfied if every empty cell, owned by the feature, is connected to the space outside the model. Since we do not know before solving which cells contain material, the `NOCLOSURE` constraint potentially affects all cells of the cellular model. It is possible to map this constraint to a Boolean expression, but the size of the expression is quadratic with respect to the size of the model. Instead, we might not map this constraint, but verify it procedurally for each realisation.

In general, if mapping a constraint is very expensive, it may be cheaper to verify it procedurally for each realisation, even though the number of realisations may be higher without mapping the constraint. As a rule, constraints involving global properties of the whole model, e.g. connectedness of the whole model, are not mapped. The number of realisations for which such constraints must be verified will be small, because most cells in the cellular model are determined by solving other Boolean constraints.

6 Boolean constraint solving

The Boolean constraint problem to be solved is the following: given a set of n Boolean variables $V = \{v_1, v_2, \dots, v_n\}$ and a set of m topological constraints

$C = \{c_1, c_2, \dots, c_m\}$, find an assignment $v_i := x_i$, where $x_i \in \{True, False\}$ for every $v_i \in V$, such that every constraint $c_j \in C$ is satisfied. This problem is known as the *Boolean satisfiability problem*, or SAT problem, which is an NP-hard problem [14]. The SAT problem has been well studied, and search algorithms exist that can find solutions efficiently for many instances. State-of-the-art solvers learn from earlier 'mistakes' and avoid search paths that are unlikely to lead to a solution. One such solving technique is *conflict-driven learning* [15, 16], which has been implemented in the MINISAT solver [17] that was used in our implementation (see Section 7). This solver, like most SAT solvers, is specialised to problems formulated as *conjunctive normal form* (CNF) clauses. A CNF clause is a set of literals, where a literal is a variable $v_i \in V$ or the negation of a variable, \bar{v}_i . A clause (l_1, l_2, \dots, l_j) is interpreted as $l_1 \cup l_2 \cup \dots \cup l_j$. A set of clauses $\{c_1, c_2, \dots, c_k\}$ is interpreted as $c_1 \cap c_2 \cap \dots \cap c_k$.

Some topological constraints can perhaps be formulated in CNF manually, but this is cumbersome for all but the most trivial constraints. It is, however, feasible to formulate topological constraints as expressions using the common Boolean operators, AND, OR, and NOT, as done in the previous section. Such expressions are represented in our system as trees. Each node of the tree represents either an operator or a variable. Operator nodes point to other nodes, which are used as inputs. Since creating small expressions for generic constraints can be difficult, large expressions should be expected. We use some simple reduction strategies to reduce the size of expressions: if an expression contains several sub-expressions that are lexically equivalent, they are represented by a single node that is referenced multiple times. Also, during construction of an expression, some local simplification rules are applied to reduce the number of nodes, e.g. removing double negations and evaluating expressions containing constants.

The Boolean expressions are converted to CNF clauses, using a method called *Tseitin encoding* [18]. This method replaces each subexpression with a new variable. The meaning of the operators between subexpressions, is expressed by CNF clauses on the new variables. For example, the expression $a \cup (b \cap c)$ is expanded as follows. The subexpression $b \cap c$ is substituted by a new variable x . Now the expression becomes $a \cup x$, which is already a CNF expression. We also represent the expression $x = b \cap c$ in CNF, which becomes:

$$(b \cup c \cup \bar{x}) \cap (b \cup \bar{c} \cup \bar{x}) \cap (\bar{b} \cup c \cup \bar{x}) \cap (\bar{b} \cup \bar{c} \cup x)$$

The complete expression thus corresponds to the following set of clauses:

$$(a, x), (b, c, \bar{x}), (b, \bar{c}, \bar{x}), (\bar{b}, c, \bar{x}), (\bar{b}, \bar{c}, x)$$

The system of CNF clauses is solved with the MINISAT solver [17]. The standard implementation of this solver finds only one solution for a given

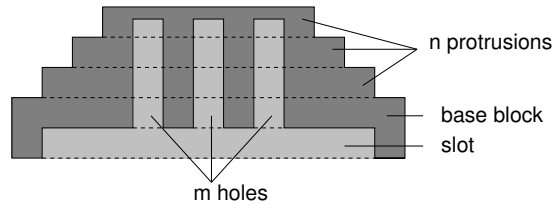


Figure 9: 2D model with $n = 3$ protrusion features and $m = 3$ hole features.

system. For our application we need to find all possible solutions. Therefore, we have implemented an algorithm that finds all solutions, by systematically adding extra constraints to the system, forcing the solver to find different solutions.

To gain insight in the feasibility of solving topological constraints, we have experimented with solving several idealised models. The models are 2D, and consist of orthogonal rectangular features. They do not represent realistic models, but are intended to generate large systems of topological constraint systems, with a complexity that can be expected from realistic models. The entities of the cellular model are also orthogonal rectangular cells.

The first model, shown in Figure 9, contains a base block with a slot. On the base block, n protrusions are stacked. To the slot, m blind holes are attached, such that the holes intersect with the protrusions. This model has been solved for different values of m and n , to determine the solving times for increasing numbers of feature interactions. The hole features and the protrusion features are all dependent on the base block, but are independent of each other. Thus, the values of the intersection cells of the holes and the protrusions cannot be determined by feature dependency analysis. However, each hole has a topological constraint that it may not be split. The model has exactly one solution for any given number of holes and protrusions.

Table 2 shows, for different values of $m + n$, the number of cells in the cellular model, the number of clauses in the constraint system, and the solving time in milliseconds. The number of cells in the model is roughly quadratic to the number of features in the model, because the model is designed to have a large number of feature interactions. The number of clauses is linear with respect to the number of cells in the model. Each solving time in the table is the sampled mean over 50 experiments. The standard deviation for the solving times is approximately 10% of the mean.

Figure 10 shows a plot of the solving times against the number of cells in the model. The latter quantity is representative for the complexity of the model, because it depends on the number of feature interactions. The plot is quite jagged, due to the large standard deviation, but seems to suggest that solving time grows slightly faster than linear with respect to the number of cells in the model.

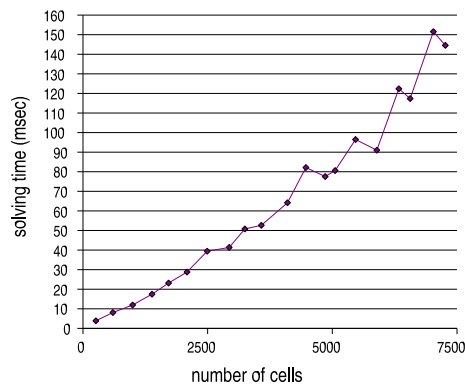


Figure 10: Plot of solving times against the number of cells for the model in Figure 9.

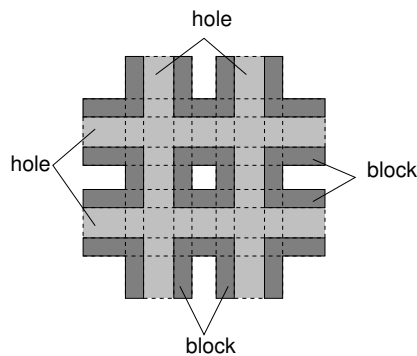


Figure 11: 2D model with $n + n$ intersecting blocks and $n + n$ through holes, shown for $n = 2$.

We have also run the experiment on a slightly more complex model, shown in Figure 11. A plot of the solving times against the number of cells for this model is shown in Figure 12. This plot also shows a slightly more than linear increase of the solving time with the number of cells.

It should be noted that, in these models, the number of cells grows quadratically with the number of features, and the number of clauses quickly gets very large. This is because the example models were intentionally designed to maximise the number of feature interactions. Yet solving times are relatively low, and do not seem to grow alarmingly with the number of cells. In realistic models, the number of cells per feature will likely be much smaller, and thus even better solving times can be expected.

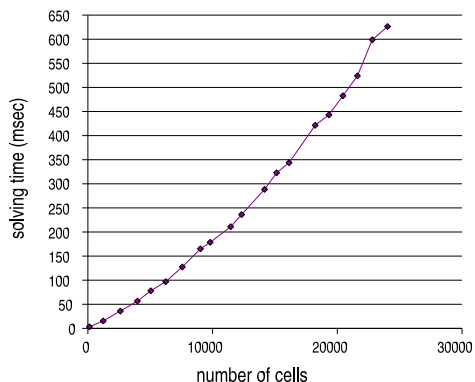


Figure 12: Plot of solving times against the number of cells for the model in Figure 11.

7 Implementation

Our implementation of the DFOM is based on SPIFF, a feature modelling system developed at Delft University of Technology. SPIFF originally implemented the semantic feature modelling approach presented in [1]. Because the DFOM is based on concepts of semantic feature modelling, many parts of the modelling system could be re-used. In particular, the DFOM implementation uses the same feature class definitions and canonical shape models as the semantic feature model.

The most important modification concerns the evaluation of the cellular model. Originally, from the partial feature ordering, determined by feature dependency analysis, a strict feature precedence order was derived, by giving precedence to features that have been more recently created or edited. For each cell in the cellular model, whether it contains material was determined by the feature with the highest precedence number in the owner list of that cell. Topological constraints were verified by the system after the cellular model had been evaluated.

In the new interpretation, whether a cell contains material is determined by feature dependency analysis only if the analysis yields a strict ordering for the features in the owner list of the cell. All other cells values are determined by solving the topological constraints on the model. Because of this other interpretation, a DFOM may have zero, one or more realisations, whereas for a semantic feature model, always a single realisation was found. The modelling system now allows the user to choose a realisation from the set of all realisations, found by the topological solver.

For example, in Figure 13, two realisations of a feature model are shown. The model contains a large blind hole that has a PARTIALLYONBOUNDARY constraint on its bottom face. The user can choose one of the realisations, or add constraints to reduce the number of realisations. If a NOOBSTRUCTION

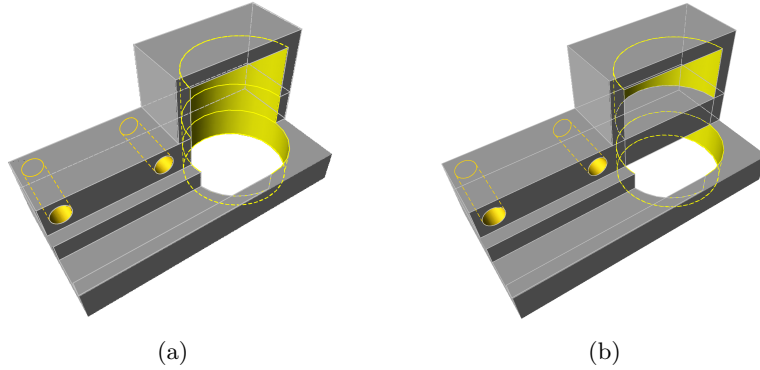


Figure 13: Two realisations of a model in SPIFF.

or NOSPLITTING constraint is added to the blind hole, only realisation (a) will be found, because in model (b) the boundary of the blind hole is split, and the volume of the hole is obstructed.

If a model is invalid, i.e. it has no realisations, then the user is informed which constraints cannot be satisfied, and the affected features are highlighted. Note that, in general, several disjunct sets of constraints may not be satisfiable at the same time. It is difficult to break down and present this information to the user. To be able to highlight features, some "nearly satisfied" realisation must be generated. Currently, such a realisation is determined using only feature precedence, i.e. the model is interpreted as a Semantic Feature Model. From this realisation it is determined which constraints are not satisfied and which features are involved. The user is presented with a dialog that gives options to restore model validity by changing or removing the constraints and/or features involved.

8 Conclusions

The Declarative Family of Objects Model (DFOM), presented here, allows families of objects to be specified using constraints on geometry and topology. It does not have the main problems associated with history-based models. In particular, it correctly maintains feature semantics, and is not affected by the feature ordering problem. Feature and constraint definitions of the Semantic Feature Model have been re-implemented in the DFOM. The ambiguity of feature dependency analysis is overcome by solving topological constraints. The model has been implemented in the prototype feature modelling system SPIFF.

Solving systems of topological constraints is a new, interesting development in CAD. It is equivalent to the satisfiability problem, which is NP-hard. Our experiments with the MINISAT solver, however, show that solving topo-

logical constraints is feasible, even for large models with many interacting features. The CNF problems derived from such models, although easy to solve for the MINISAT solver, are very large. Generating these problems, however, is expensive. A more efficient algorithm for mapping and solving topological constraints, would therefore be very useful. The MINISAT solver can be extended with new types of constraints, but the effectiveness of the solver's optimisation strategy for such constraints is unknown.

The DFOM allows a large variety of new features and constraints to be defined, which results in the required flexibility to adequately define the semantics of a family of objects. Topological constraints are essential for the DFOM. Such constraints and our solving method might be applicable in other modelling applications as well.

Acknowledgements

H.A. van der Meiden's work is supported by the Netherlands Organisation for Scientific Research (NWO).

References

- [1] R. Bidarra, W. F. Bronsvort, Semantic feature modelling, *Computer-Aided Design* 32 (3) (2000) 201–225.
- [2] A. Rappoport, The Generic Geometric Complex (GGC): a modeling scheme for families of decomposed pointsets, in: C. M. Hoffmann, W. F. Bronsvort (Eds.), *Proceedings Solid Modeling '97, Fourth ACM symposium on Solid Modeling and Applications*, May 14-16, Atlanta, Georgia, USA, ACM Press, 1997, pp. 19–30.
- [3] S. Raghobhama, V. Shapiro, Boundary representation deformation in parametric solid modeling, *ACM Transactions on Graphics* 17 (4) (1998) 259–286.
- [4] R. Bidarra, P. J. Nyirenda, W. F. Bronsvort, A feature-based solution to the persistent naming problem, *Computer-Aided Design and Applications* 2 (1-4) (2005) 517–526.
- [5] B. Bettig, V. Bapat, B. Bharadwaj, Limitations of parametric operators for supporting systematic design, in: *CDROM Proceedings DECT-2005, ASME International Design Engineering Technical Conferences*, September 24-28, Long Beach, California, USA, ASME, 2005.
- [6] M. Sitharam, J.-J. Oung, Y. Zhou, A. Abree, Geometric constraints within feature hierarchies, *Computer-Aided Design* 38 (1) (2006) 22–38.

- [7] V. Shapiro, D. L. Vossler, What is a parametric family of solids?, in: C. M. Hoffmann, J. R. Rossignac (Eds.), Proceedings of the Third ACM/IEEE Symposium on Solid Modeling and Applications, May 17-19, Salt Lake City, Utah, USA, ACM Press, 1995, pp. 43–54.
- [8] S. Raghobhama, V. Shapiro, Topological framework for part families, *Journal of Computing and Information Science in Engineering* 2 (4) (2002) 246–255.
- [9] J. R. Rossignac, M. A. O’Connor, SGC: a dimension-independent model for pointsets with internal structures and incomplete boundaries, in: M. Wozny, J. Turner, K. Preiss (Eds.), Proceedings of the 1988 IFIP/NSF Workshop on Geometric Modeling, Rensselaerville, New York, USA, North Holland, 1988, pp. 145–180.
- [10] R. Bidarra, K. J. de Kraker, W. F. Bronsvoort, Representation and management of feature information in a cellular model, *Computer-Aided Design* 30 (4) (1998) 301–313.
- [11] R. Bidarra, J. Madeira, W. Neels, W. F. Bronsvoort, Efficiency of boundary evaluation for a cellular model, *Computer-Aided Design* 37 (12) (2005) 1266–1284.
- [12] R. Bidarra, W. F. Bronsvoort, On families of objects and their semantics, in: Proceedings of Geometric Modeling and Processing 2000, April 10-12, Hong Kong, China, IEEE Computer Society, 2000, pp. 101–111.
- [13] H. A. van der Meiden, W. F. Bronsvoort, An efficient method to determine the intended solution for a system of geometric constraints, *International Journal of Computational Geometry and Applications* 15 (3) (2005) 279–298.
- [14] C. H. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1995.
- [15] J. P. Silva, K. A. Sakalla, Grasp - a new search algorithm for satisfiability, in: Proceedings of ICCAD 1996, IEEE/ACM International Conference on Computer-Aided Design, November 10-14, San Jose, California, USA, IEEE Computer Society Press, 1996, pp. 220–227.
- [16] L. Zhang, C. F. Madigan, M. W. Moskewicz, S. Malic, Efficient conflict driven learning in a boolean satisfiability solver, in: Proceedings of ICCAD 2001, IEEE/ACM International Conference on Computer-Aided Design, November 4-8, San Jose, California, USA, IEEE Computer Society Press, 2001, pp. 279–285.

- [17] N. Een, N. Sörensson, An extensible SAT solver, in: E. Giunchiglia, A. Tacchella (Eds.), Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003, Santa Margherita Ligure, Italy, May 5-8, 2003, Selected Revised Papers, Vol. 2919 of Lecture Notes in Computer Science, Springer Verlag, 2004.
- [18] G. S. Tseitin, On the complexity of derivation in propositional calculus, in: Slisenko (Ed.), Studies in Constructive Mathematics and Mathematical Logic, 1970, pp. 115–125.

Vitae



Hilderick A. van der Meiden is PhD student at the Faculty of Electrical Engineering, Mathematics and Computer Science of Delft University of Technology, The Netherlands. He graduated in computer science at the same university in 2004. His research interests include feature modeling, constraint solving, and semantics of families of objects.



Willem F. Bronsvort is associate professor CAD/CAM at the Faculty of Electrical Engineering, Mathematics and Computer Science of Delft University of Technology, The Netherlands. He received his MSc degree in computer science from the University of Groningen in 1978, and his PhD degree from Delft University of Technology in 1990. His main research area is feature modeling, in particular semantic feature modeling, multiple-view feature modeling, freeform feature modeling, and mesh generation from feature models. He has published numerous papers in international journals, books and conference proceedings, is on the editorial board of several journals, and has served as co-chair and member of many program committees of conferences.

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 some subtractive feature volume to become a closed void inside the model
Absorption	Causes a feature to cease completely its contribution to the model boundary

Table 1: Feature interaction types.

$m + n$	# cells	# clauses	time (msec)
10+10	259	1457	3.9
16+16	601	3467	8.1
21+21	996	5802	11.9
25+25	1384	8102	17.4
28+28	1717	10079	23.1
31+31	2086	12272	28.7
34+34	2491	14681	39.4
37+37	2932	17306	41.3
39+39	3246	19176	50.7
41+41	3576	21142	52.6
44+44	4101	24271	64.1
46+46	4471	26477	82.1
48+48	4857	28779	77.5
49+49	5056	29966	80.6
51+51	5466	32412	96.4
53+53	5892	34954	91.0
55+55	6334	37592	122.0
56+56	6561	38947	117.3
58+58	7027	41729	151.4
59+59	7266	43156	144.0

Table 2: Solving statistics for the model in Figure 9.