ELSEVIER

PII: S0010-4485(97)00070-5

Computer-Aided Design, Vol. 30, No. 4, pp. 301–313, 1998
© 1998 Elsevier Science Ltd. All rights reserved
Printed in Great Britain
0010-4485/98/$19.00+0.00

Representation and management of feature information in a cellular model

Rafael Bidarra†, Klaas Jan de Kraker and Willem F. Bronsvoort*

Many limitations in current feature modelling systems are inherited from the geometric representation they use for the product model. Both a very rigid and a very extensive representation are unsuitable for feature applications, at least if no convenient support is provided to manage the data. This paper describes a cellular representation for feature models that contains all the relevant information to effectively solve a variety of current problems in feature modelling. Much benefit is gained from a coherent integration between shapes of a feature model and cells in the cellular model. Every feature shape has an explicit volumetric representation in terms of cells. Specific subsets of its boundary are also distinguished in terms of cell faces and edges. Feature interactions are maintained in attributes of cells, cell faces and cell edges. Methods for modifying and querying the cellular model are presented, and their application is illustrated for feature validity maintenance, feature interaction management, feature conversion between multiple views, and feature visualization. © 1998 Elsevier Science Ltd. All rights reserved

Keywords: feature representation, cellular model, feature validity maintenance, feature interactions, feature conversion, feature visualization

1. INTRODUCTION

Features are representations of shape aspects of a physical product that are mappable to a generic shape and are functionally significant. Features can be used in several product life cycle activities, e.g. design, manufacturing and cost evaluation. Each activity has its own way of looking at a product, i.e. its own *view* of it, and therefore each view contains the features relevant to that specific activity.

The above considerations pose strong demands on the geometric representation of features in feature models:

(1) availability of an explicit representation for the whole boundary of volumetric features;
(2) ability to distinguish the specific roles of different subsets of that boundary;
(3) mapping of feature parameters onto entities of the geometry;
(4) recording of which features (possibly from different views) each topological entity relates to; and
(5) representation of interaction extents between overlapping or attached features.

The main goal of this paper is to show how a cellular representation of feature models can be effectively used for a variety of feature applications. This representation is shown to provide a significant contribution to solve several current research problems in feature-based modelling, e.g. validity maintenance, enhanced user assistance during model edition, feature conversion between views, and feature visualization.

We describe the structure and functionality of the Cellular Model of the SPIFF† modelling system, a prototype multiple-view feature-based modeller developed at Delft University of Technology. For an overview of the SPIFF system see Bronsvoort *et al.*[1].

This paper is organized as follows. Firstly, we survey recent advances in geometric representations for feature models. Secondly, we introduce the structure and functionality of the product model and the cellular model of SPIFF, with emphasis on the data contained in the cellular representation. The use of the Cellular Model classes interface is then illustrated by means of query and modification operations. Subsequent sections show how this data can be used, allowing for high-level information on feature semantics to be derived, analysed and managed. In particular, issues on feature validity maintenance, feature interaction detection, feature conversion and feature visualization are dealt with, emphasizing how their implementation strongly benefits from the cellular representation.

2. PREVIOUS RESEARCH

Although it did not take long until it was recognized that conventional boundary representations were not able to meet the requirements described above, such representations were sufficient for primitive feature-based modelling prototypes. In the meantime, alternative schemes better capable of supporting the geometric representation of both

Faculty of Information Technology and Systems, Delft University of Technology, Zuidplantsoen 4, NL-2628 BZ Delft, The Netherlands
*To whom correspondence should be addressed. E-mail: Bronsvoort@cs.tudelft.nl

† Named after Spaceman Spiff, interplanetary explorer *extraordinaire*

features and part models were sought for. Eventually, the need for more powerful and flexible feature model manipulation and, in particular, for managing feature interactions, led to the use of *cellular representation schemes*, in which feature information can be recorded as attributes of cell entities.

The first explicit proposal to use such a topological structure for the integrated representation of both features and the part model was presented by Pratt[2], as a direct consequence of choosing an explicit volumetric representation of features, with all its advantages. Such representation consists of sub-volumes lying either internal or external to the part volume itself, thus yielding an irregular structure in the representation of the part model. Although an extension layer on top of the radial-edge data structure[3] was suggested for that purpose, so far there are no implementations known that actually follow this approach.

Gomes *et al.*[4] proposed a somewhat more elaborate hybrid scheme, that recorded the model history while keeping its evaluated representation. It is based on hierarchically structuring all evolution stages of decomposition of sub-volumes. Eventually the intensive use of mating relations required between cells, together with the intended use of a conventional boundary representation for them, made the implementation of this proposal complex and inefficient.

A different proposal suitable for feature model representation was done by Rossignac[5]: the Selective Geometric Complex (SGC). This is a cellular scheme that permits the representation of objects of mixed dimensionality, with an internal decomposition structure. Conceptually, SGCs rely on the connectivity between n-dimensional cells (or simply n-cells), which is captured on an incidence graph indicating 'is-a-boundary-of' relations between cells of different dimensions. The use of an attribute *status* in each cell is proposed to distinguish which cells are contributing or not to the actual point set of the part, thus also allowing for models with open boundaries.

The above characteristics of SGCs made it an attractive choice for representing potentially overlapping features. But it was not until 1995 that its first implementation was developed, within the GNOMES geometric engine[6]. The main purpose of this system is to support co-operative applications, by providing a unified representation scheme for wireframe, surface and solid models that could overcome the disparity in requirements presented by various CAx systems used in different product life cycle activities (e.g. finite element analysis, manufacturability analysis, cost evaluation, etc.). Although GNOMES is a partial implementation of SGCs, with cells of dimensionality lower or equal to 3 only, it may indeed provide a rather convenient scheme for handling some of the advanced research problems previously mentioned, in particular for recording feature interactions. One limitation, however, that arises from its generality, is the inability to attach different attributes to 'each side' of a 2-cell (face); in this way, when such a cell forms an interior boundary, it is hard to keep track of the 'double' role it plays in bounding both neighbouring 3-cells.

Masuda[7] introduced the so-called complex-based non-manifold geometric models as a pragmatic alternative to SGCs, arguing that the limitation for his cell complexes to be always closed does not restrict in practice the flexibility of geometric models for CAD applications. Boolean operations developed under this approach are based on cancel, merge and extract operations specifically developed for the data structure, reputedly with considerable improvement in performance. This work focuses on representational issues of cellular models and on their suitability for feature models. For this purpose, original primitive shapes of features are explicitly maintained in parallel with the merged cellular model. However, the application of these shapes to interaction management and to feature validity issues, although promising, is not dealt with in the paper.

It can be concluded that a cautious balance should be achieved between the complexity inherent to a given geometric representation scheme, and the suitability for various applications within the scope of feature modelling systems. A cellular representation that contains exactly the data required, without excessive generality, appears to be more effective than other schemes, possibly with a potential broader scope, but whose implementation and use might cause development of feature applications more problematic and difficult. Current research problems, such as maintenance of feature validity and management of feature interactions, show an intrinsic complexity that encourages this pragmatic approach.

3. THE PRODUCT MODEL

In SPIFF, a product model consists of several *views* of a product, each one with its own *feature model*. A feature model contains a set of features instantiated from a feature library specific to that particular view. The geometry of all feature instances of a product is represented in a central geometric model—the *Cellular Model*—that is shared by all views and provides, thus, a common representation for the product geometry, see *Figure 1*.

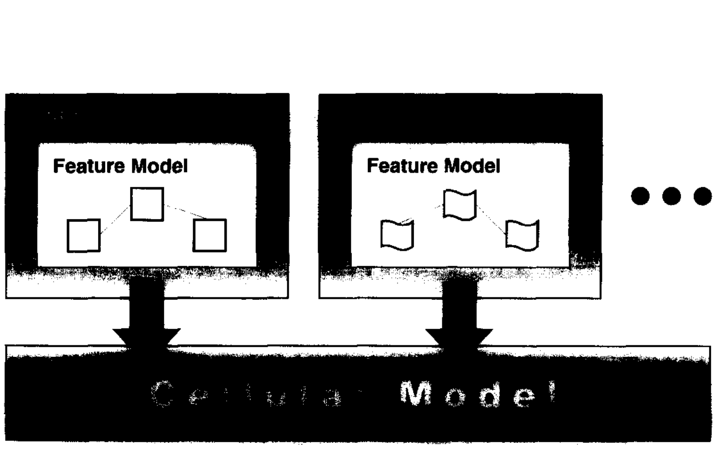


Figure 1 The product model

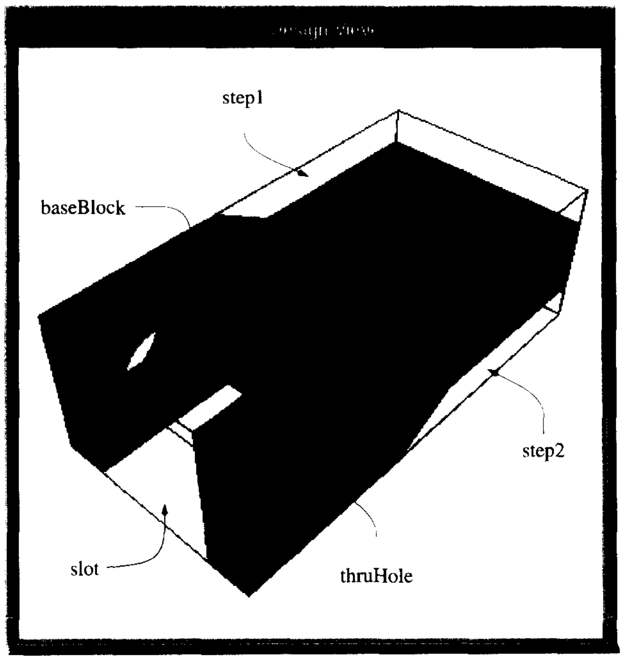


Figure 2 Example part with its feature shapes

Each feature is associated with one (or more) *shape* class(es). A through hole, for example, is associated with a *cylinder* shape. Shapes are parameterized, and their parameter values may be either set by the user, or derived automatically from those of other features already in the model: for a through hole, for example, its depth—the cylinder height—may be determined from a stock's dimension parameter. A shape instance, with all its parameter values set (dimensions, position and orientation), occupies a particular region of space—the *shape extent*. Features also assign to their shapes the *nature* attribute, specifying whether this volumetric extent is *additive*, i.e. adding material to the volume of the represented part, or *subtractive*, i.e. removing material from it. Each shape contains a set of *shape elements*, comprising functionally meaningful subsets of its boundary: *shape faces* and *shape edges*. For example, the cylinder shape has the following shape elements: the *top*, *bottom* and *side faces*, and the *top* and *bottom* circular *edges*. *Figure 2* depicts the feature shapes of the example part that will be used in the following sections.

Table 1 presents a group of methods that provide access to the information just described, pertinent to shapes and their elements.

Table 1 Basic access methods to entities in a feature model

Feature model (fm)	
fm.shapes	returns the set of all feature shapes of the feature model
Shape (s)	
s.nature	returns the nature specified for shape s
s.faces	returns the set of shape face elements of shape s
s.edges	returns the set of shape edge elements of shape s
s.view	returns the view from which shape s originates
Shape element (e)	
e.shape	returns the shape to which the element e (a shape face or a shape edge) belongs

4. THE CELLULAR MODEL

The Cellular Model represents a part as a connected set of volumetric quasi-disjoint *cells*, in such a way that each one either lies entirely *inside* a shape extent or entirely *outside* it. Cells in the Cellular Model represent the point sets of all shape extents of features from all views of the model. Each shape extent is, thus, represented in the Cellular Model by a connected subset of those cells.

Furthermore, this cellular decomposition of space is interaction-driven, i.e. for any two overlapping shapes, some of their cells lie in both shape extents (and are called *interaction cells*), whereas the remaining ones lie in either of them. As a consequence of this, two cells can never volumetrically overlap. They may, however, be adjacent, in which case there is an interior topological face of the Cellular Model separating them. Such a face can be regarded as having two 'sides', designated as partner *cell faces*. Naturally, a topological face that lies on the boundary of the Cellular Model has only one cell face (one 'side'), that of the only cell it bounds. In either case, a cell face always bounds one and only one cell. Each shape element is represented by a connected set of cell faces or cell edges (or simply *cell elements*), which account for the corresponding shape boundary point set—*the shape element extent*.

It can be concluded that every feature shape has a direct, explicit representation at the cellular level. Each cell in the Cellular Model stores, in an *owner list*, which shapes it belongs to (see *Figure 3a*); analogously, cell elements store, in an owner list, which shape elements they belong to (see *Figure 3b*). Because the SPIFF Cellular Model is aimed at representing feature shapes from different views of the product, each Cellular Model entity maintains one such owner list for each view. Provided cells can have a different nature for different views, the *nature* of a cell with respect to a view is defined as the nature of the shape most recently added to the cell's owner list for that view. A cell's nature expresses whether its volume belongs to the representation of the part in that view.

Table 2 presents a group of methods that provide access to the data explicitly stored in the Cellular Model. This information is basically of two kinds: topological, i.e. related to structural adjacency of Cellular Model entities (e.g. cells and cell faces), and attribute-based, referring to the data attached to various entities (e.g. owner lists in cells).

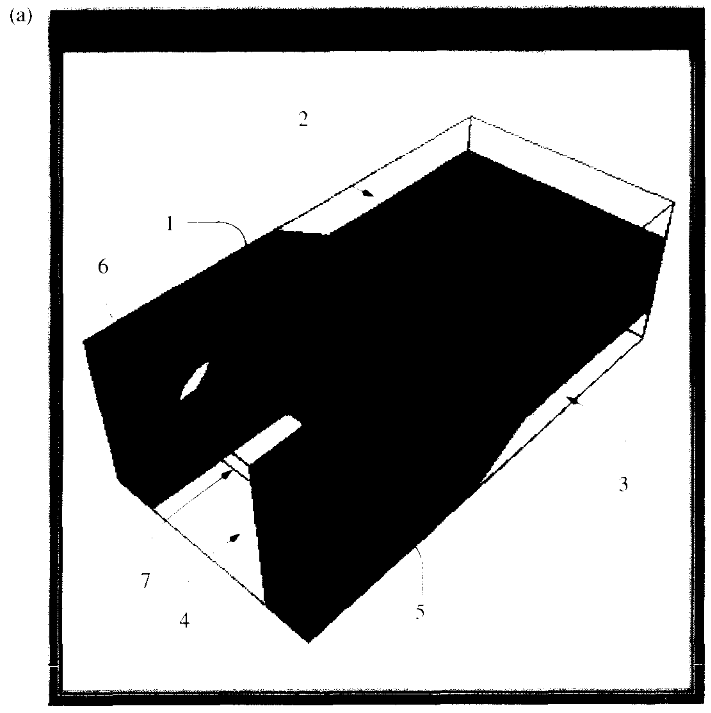
The Cellular Model was implemented using the cellular topology husk of the geometric modelling toolkit ACIS, as well as its attribute mechanism to maintain and propagate the owner lists in cells and cell elements.

5. OPERATIONS ON THE CELLULAR MODEL

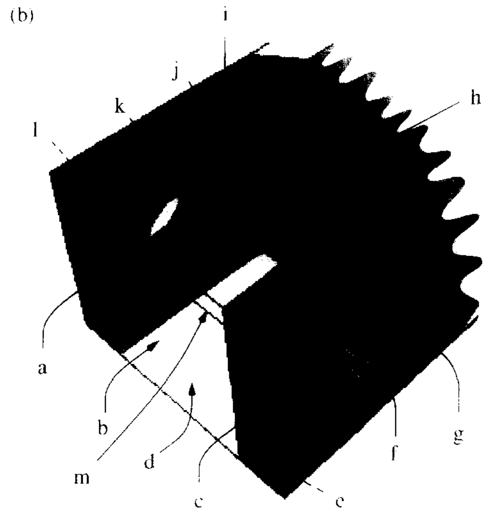
In this section the basic functionality of the Cellular Model presented so far is extended with operations that provide it with modifying and querying capabilities.

5.1. Modification operators

The two basic operations that modify the Cellular Model are the insertion and the removal of a feature shape. These operations have a twofold effect on the Cellular Model: (i) they change its topology, and (ii) they update owner lists of Cellular Model entities accordingly. Both mechanisms are described and illustrated.



1 - baseBlock
2 - baseBlock, step1
3 - baseBlock, step2
4 - baseBlock, slot
5 - baseBlock, thruHole
6 - baseBlock, thruHole
7 - baseBlock, thruHole, slot



a - baseBlock.front
b - baseBlock.front, slot.top
c - baseBlock.front
d - baseBlock.bottom, slot.end1
e - baseBlock.bottom
f - baseBlock.right, thruHole.end1
g - baseBlock.right
h - baseBlock.top, slot.end2
i - slot.bottom
j - slot.left
k - slot.left
l - thruHole.side
m - thruHole.side

Figure 3 (a) Cellular model with cell owner lists. (b) Some cell face owner lists

Table 2 Basic access methods to entities in the Cellular Model

Cellular Model (cm)	
cm.cells	returns the set of all cells of the cellular model
cm.cellFaces	returns the set of all cell faces of the cellular model
cm.cellEdges	returns the set of all cell edges of the cellular model
Cell (c)	
c.boundary	returns the set of cell faces that bound the volume of cell c
c.ownerlist(view)	returns the list of shapes of specified view that own cell c
c.nature(view)	returns the nature of cell c in the specified view
Cell face (cf)	
cf.cell	returns the cell bounded by cell face cf
cf.partner	returns the partner cell face of cf (if it exists)
cf.boundary	returns the set of cell edges that bound cell face cf
cf.ownerlist(view)	returns the list of shape elements of shapes from specified view that own cell face cf
Cell edge (ce)	
ce.ownerlist	returns the list of shape elements of shapes from specified view that own cell edge ce

Feature shape insertion

First, the Cellular Model uses the fully specified set of shape parameter values to instantiate one single cell for the new shape and to position it in space. The owner list of this cell has just one element: the shape itself. Also, each cell element has only one element in its owner list: the corresponding shape element. The example in *Figure 4* illustrates this stage for the insertion of a rectangular slot in the model, showing the owner lists of the shape cell faces.

Second, a non-regular Boolean union is performed between the Cellular Model and this cell. In this process, all cells (C_m) in the model are collected that somehow intersect the shape cell (C_s). Mutual cellular decomposition is then carried out between C_s (or any cell arising from its decomposition) and each C_m. This may occur in two ways:

(1) the two cells intersect only over their boundaries, in which case there are no new cells created; instead, their overlapping cell elements are decomposed, yielding cell elements that lie either in the intersection or outside it; or
(2) the two cells overlap volumetrically, in which case the decomposition results in cells that lie either inside the intersection or outside it. Mostly, a subset of the boundary of these cells is also decomposed, yielding cell elements that lie either on the intersection or outside it (this is not the case only when one of the cells lies entirely inside the other).

Whenever two cells undergo this mutual decomposition, the owner lists of all new entities are systematically updated as follows:

(1) (i) a new cell that lies in the intersection of C_s and C_m merges their owner lists into its owner list; (ii) the remaining cells resulting from the decomposition receive as owner list that of the respective cell from which they originate (either C_s or C_m);
(2) analogously, (i) a new cell element lying on the boundary intersection of C_s and C_m merges the owner lists of the overlapping cell elements into its owner list; (ii) new cell elements lying on the boundary of either C_s or C_m receive as owner list that of the respective element from which they originate; and lastly, (iii) any remaining cell elements arising from the decomposition, for example an edge arising from a face–face intersection, receive an empty owner list.

The final situation of the Cellular Model after decomposition and owner list updating is shown in *Figure 3.*

Feature shape removal

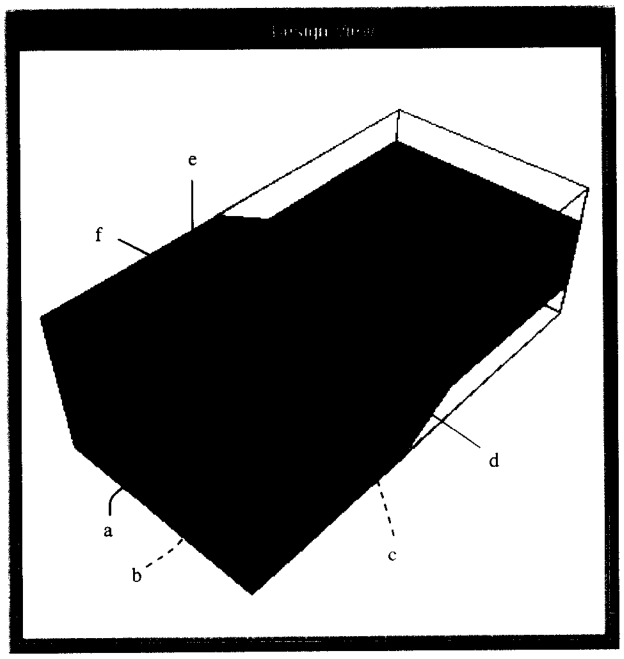
The operation of removal of a shape from the Cellular Model is carried out in two steps. First, all references to the shape or to its elements are removed from the owner lists of all entities in the Cellular Model (cells and their cell elements). This results in the occurrence of either (i) adjacent entities with exactly the same owner lists or (ii) cells with an empty owner list; these last cells are immediately removed. The second phase is then performed by merging and removing unnecessary entities according to the following scheme:

(1) while two adjacent cells with the same owner lists can be found, merge them, by removing all cell faces that separate them
(2) while two adjacent cell faces with the same owner lists can be found, merge them, by removing all cell edges that separate them
(3) while two adjacent cell edges with he same owner lists can be found, merge them into just one cell edge

It can easily be proved that when cellular condensation is carried out in this way, the model attained in the final stage has the minimal number of entities required for the representation of all feature shapes of all views in the model.

5.2. Query operators

Operations in *Table 2* provide the appropriate vocabulary on top of which a second layer of operations has been devised.



a -	slot.top
b -	slot.end1
c -	slot.right
d -	slot.end2
e -	slot.bottom
f -	slot.left

Figure 4 Owner lists of slot cell faces at the insertion stage

Table 3 Collector methods

Collector (c)	
c.contains(element)	returns TRUE if given element belongs to collector c
c.remove(element)	removes given element from collector c
c.add(element)	adds given element to collector c
c.union(collector)	adds all elements of given collector c (if not already there)
Ordered list (l)	
l.after(element$_1$, element$_2$)	returns TRUE if element$_1$ occurs after element$_2$ in the ordered list l
l.first	returns the first element of the ordered list l
l.last	returns the last element of the ordered list l

Only those methods that are used directly in the application problems discussed in this paper are presented. Therefore, most geometric queries, and other methods used in constraint solving aspects of SPIFF, are here omitted. Most of the operations return a *collector* object; therefore we use in their algorithms several methods commonly provided by collector classes (sets, ordered lists, etc.), as shown in *Table 3*. The operations in *Table 4* give basic functionality for selecting Cellular Model entities, possibly based on some filter criteria, as well as for deriving some of their useful properties or relations not explicitly stored.

These operations can be used from the perspective of feature model entities (shapes and shape elements), as shown in *Table 5*. In this way, their geometric representation can be selectively accessed, allowing for higher-level methods to assist in drawing conclusions about actual feature semantics in the model, as will be shown in the following sections.

6. SEMANTIC CONSTRAINTS

Many of the current systems that are called feature modelling systems are in essence only high-level geometric modelling systems, because they provide only very little assistance in maintaining the meaning, or validity, of features during a modelling session. One approach to maintain feature validity is by means of validation constraints, which are part of each feature class specification—the *generic feature definition.* Several types of validation constraints are used in SPIFF; among them, geometric and algebraic constraints, handled by dedicated solvers, play an important role in determining each feature's geometry[8]. All validation constraints must be solved or checked to verify whether a feature is valid, both at its creation step and in subsequent modelling steps[9].

An important aspect of feature validity is specified in generic feature definitions by means of so-called *semantic constraints*[10]. Semantic constraints specify how each

Table 4 Examples of Cellular Model query methods

Cell face (cf)		

cf.onBoundary(view)	returns TRUE if the cell face lies on the model boundary for the specified view, and FALSE otherwise	if cf.cell.nature(view) = cf.partner.cell.nature(view)
		    return FALSE
		else
		    return TRUE

cf.adjacent(cellFace)	returns TRUE if cellFace is adjacent to cell face cf, and FALSE otherwise	for each cell edge ce$_1$ in cf.boundary
		    for each cell edge ce$_2$ in cellFace.boundary
		        if ce$_1$ = ce$_2$
		            return TRUE
		return FALSE

Cell (c)		

c.adjacent(cell)	returns TRUE if cell is adjacent to cell c, and FALSE otherwise	for each cell face cf in c.boundary
		    if cf.partner.cell = cell
		        return TRUE
		return FALSE

Cellular Model (cm)		

cm.cell(nature, view)	returns the set of cells with specified nature in the Cellular Model for specified view	for each cell c in cm.cells
		    if c.nature(view) = nature
		        cells.add(c)
		return cells

cm.bound ary(view)	Returns the set of cell faces that make up the boundary of the part for specified view	For each cell c in cm.cells(**additive**, view)
		    for each cell face cf in c.boundary
		        if cf.onBoundary(view)
		            boundary.add(cf)
		return boundary

Table 5 Example methods of feature model entities for accessing the cellular representation

Shape edge (se)		

se.extent	returns the set of cell edges that lie in the extent of se	for each cell edge ce in cm.cellEdges
		    if ce.ownerlist(se.shape.view).contains(se)
		        extent.add(ce)
		return extent

Shape face (sf)		

sf.extent	returns the set of cell faces that lie in the extent of sf	for each cell face cf in cm.cellFaces
		    if cf.ownerlist(sf.shape.view).contains(sf)
		        extent.add(cf)
		return extent

Shape (s)		

s.extent	returns the set of all cells that lie in the shape extent of s	for each cell c in cm.cells
		    if c.ownerlist(s.view).contains(s)
		        extent.add(c)
		return extent

s.extent(nature)	returns the set of cells with specified nature that lie in the shape extent of s	for each cell c in s.extent
		    if c. nature(s.view) = nature
		        extent.add(c)
		return extent

s.boundary	returns the set of cell faces that lie in the extent of shape face elements of s	for each shape face sf in s.faces
		    for each cell face cf in sf. extent
		        boundary.add(cf)
		return boundary

s.boundary(nature)	returns the set of cell faces that lie in the extent of shape elements of s, according to whether they lie also on the model boundary	for each cell face cf in s.boundary
		    if cf. onBoundary(s.view) xor
		            nature = **subtractive**
		        boundary.add(cf)
		return boundary

s.wire	returns the set of cell edges that lie in the extent of shape edge elements of s	for each shape edge se in s.edges
		    for each cell edge ce in se.extent
		        wire.add(ce)
		return wire

feature instance is allowed to deviate from its canonical behaviour. This can be stated in terms of the feature shape and its elements. A semantic constraint on a shape specifies that no other shapes (with a specified nature) are allowed to overlap with its extent. A semantic constraint on a shape element specifies the extent to which it should belong to the model boundary or not. An example is that for a cylindrical blind hole, its circular bottom face should be completely on the boundary of the modelled object, its side face should be at least partly on the boundary, whereas its circular top face should not be on the boundary. In SPIFF this can be achieved by including in a blind hole class specification the following semantic constraint declarations:

```
semanticBottom: OnBoundary(completely);
semanticSide:   OnBoundary(partly);
semanticTop:    NotOnBoundary(completely);
```

It should be remarked that this kind of feature validity specification has an important role in preserving the functional and technological meaning of features: in the blind hole example, the full presence of the bottom face on the boundary assures its 'non-through' functional character, whereas the absence of the top face on the boundary is a necessary (although not sufficient) clearance condition from the manufacturability point of view.

The maintenance of semantic constraints relies on the capabilities presented above for querying the Cellular Model in order to retrieve a shape (or shape element) extent, and to assess topological properties of those cells (or cell elements). *Table 6* shows several methods that are used in SPIFF as Boolean checkers for some semantic constraints. They are automatically applied, after any model modification operation, to all shapes on which semantic constraints were specified. In addition, their result is used to provide the user of SPIFF with more appropriate explanations and/or hints whenever some semantic constraint violation needs to be overcome.

7. FEATURE INTERACTION DETECTION

Feature interactions may have a very wide range of consequences and effects on a feature model. While these may often be intended, it is also true that most feature validity violations are caused, in one way or another, by feature interactions. Indeed, these may affect the semantics of a feature, ranging from slight changes in actual parameter values to the complete suppression of its contribution to the model shape. It is, thus, important for any feature-based modelling system to be able to detect such situations, so that they can be properly classified, reported to the user and overcome. For a formal definition of feature interactions and a taxonomy, see Bidarra and Bronsvoort[11].

The detection of most types of feature interactions relies mainly on analysis of the topology of features in the model;

Table 6 Methods for checking semantic constraints

Shape (s)		

s.noOverlap(nature)	returns TRUE if no other shape, with specified nature, inserted in the model after s overlaps with it, and FALSE otherwise	for each cell c in s.extent
		  list ← c.ownerlist(s.view)
		for each shape s_i in list
		    if list.after(s_i,s) and s_i.nature = nature
		      return FALSE
		return TRUE

Shape face (sf)		

sf.onBoundary(mode)	returns TRUE if the extent of shape face sf is (completley or partly, according to specified mode) on the model boundary, and FALSE otherwise	if mode = **completely**
		  for each cell face cf in sf. extent
		    if not cf.onBoundary(sf.shape.view)
		      return FALSE
		  return TRUE
		else //mode = **partly**
		  for each cell face cf in sf.extent
		    if cf.onBoundary(sf.shape.view)
		      return TRUE
		  return FALSE

sf.notOnBoundary(mode)	returns TRUE if the extent of shape face sf is (completely or partly, according to specified mode) not on the model boundary, and FALSE otherwise	if mode = **completely**
		    for each cell face cf in sf.extent
		      if cf.onBoundary(sf.shape.view)
		        return FALSE
		    return true
		else //mode = **partly**
		    for each cell face cf in sf.extent
		      if not cf.onBoundary(sf.shape.view)
		        return TRUE
		    return FALSE

a few interaction classes require, in addition, some geometric queries. In both cases, however, it is essential that at the representation level of the feature model, sufficient information is stored and retrieved in an efficient way. The Cellular Model provides this functionality. Several methods were developed on top of it that identify each interaction situation independently of the complexity or the number of features involved. This identification also receives valuable hints from a detailed analysis of the outcome of semantic constraints checking, for instance in the case of feature type changes (transmutation interactions).

The first necessary condition for the occurrence of an interaction between two features is that their shapes have a non-empty intersection, which may take place either volumetrically or between their boundaries. For some cases, an intersection is also required between a subset of their shape boundaries (with only 'onBoundary cell faces', i.e. s.boundary(additive), in *Table 5*). These two methods, shown in *Table 7*, are used in SPIFF in a first phase, in order to delimit the scope of the set of features for the global interaction mechanism, whenever a feature shape is added to or removed from the model.

In a second phase, identification of possible classes of interaction taking place among features within this set is carried out. See Bidarra *et al.*[9] for an extensive description of interaction detection algorithms. Two examples of Boolean detectors are presented in *Table 8*. The first one performs the detection of *disconnection interactions*, which cause a part to have its volume split into several disconnected regions (see *Figure 5a* for an example). The second one identifies *absorption interactions* on a shape, characterized by the complete suppression of the contribution of a feature to the global model shape (see *Figure 5b* for an example).

Table 7 Methods used for retrieving feature shapes in interaction

Shape (s)		

s.overlappingSet	returns the set of shapes overlapping with shape s (either volumetrically or between their boundaries - cell faces and edges)	for each cell c in s.cells
		  overlappingSet.union(c.ownerlist(s.view))
		for each cell face cf in s.boundary
		  overlappingSet.union(cf.ownerlist(s.view))
		for each cell edge ce in s.wire
		  overlappingSet.union(ce.ownerlist(s.view))
		overlappingSet.remove(s)
		return overlappingSet

s.adjacentBoundaries(s_1)	returns TRUE if the "additive boundary" of shape s intersects that of shape s_1, and FALSE otherwise	for each cell face cf_1 in s_1.boundary(**additive**)
		  for each cell face cf in s.boundary(**additive**)
		    if cf.adjacent(cf_1)
		      return TRUE
		return FALSE

Table 8 Example methods for feature interaction detection

Cellular Model (cm)		
cm.disconnection(view)	returns TRUE if the set of all additive cells of the Cellular Model is not connected (i.e. at least one cell is not accessible from the others), and FALSE otherwise	cells ← cm.cells(**additive**, view) c_1 ← cells.first for each cell c_2 in cells if not cells.accessible(c_1, c_2)* return TRUE return FALSE
Shape (s)		
s.absorption	returns TRUE if the shape s has undergone an absorption interaction, and FALSE otherwise	for each cell c in s.extent if c.ownerlistls.view).last = 2 return FALSE return TRUE

*The `accessible` ($\mathbf{e}_1$, $\mathbf{e}_2$) method applied on a **`set`** of entities returns TRUE iff:

(a) the two specified elements, $\mathbf{e}_1$ and $\mathbf{e}_2$, verify either $\mathbf{e}_1 = \mathbf{e}_2$ or $\mathbf{e}_1$. `adjacent` ($\mathbf{e}_2$); or

(b) there is a third element $\mathbf{e}_3$ in the **`set`** such that $\mathbf{e}_1$.`adjacent`($\mathbf{e}_3$) and **`set`**.`accessible`($\mathbf{e}_3$,$\mathbf{e}_2$) .

and FALSE otherwise.

8. FEATURE CONVERSION

For maintaining multiple views of a product, *feature conversion* is used. Two forms of feature conversion can be distinguished[12]. The first is the derivation of a feature model for a new view given the product model already specified by one or more other views, which is called *opening a view*. The second form is propagating feature parameter changes from one view to the other views. Here, only opening a view is discussed.

De Kraker *et al.*[13–15] describe a generic method for

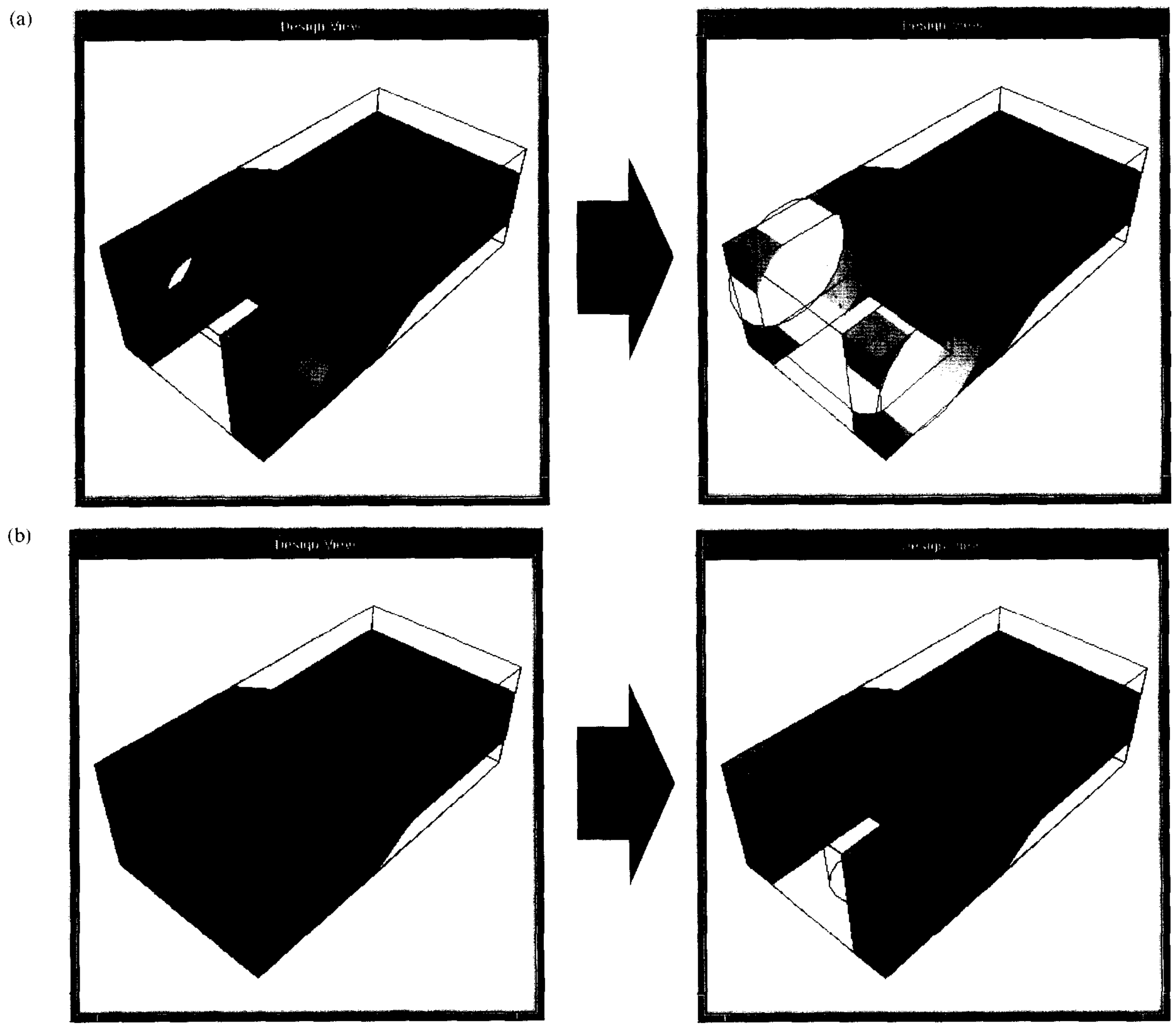


Figure 5 Examples of feature interactions: (a) disconnection interaction; (b) absorption interaction

Table 9 Methods of Cellular Model for feature conversion

Cell (c)		

c.consistent(view$_1$,view$_2$)	returns TRUE if the cell nature with respect to the given views is equal, and FALSE otherwise	if c.nature(view$_1$) = c.nature(view$_2$)
		    return TRUE
		else
		    return FALSE

Cellular model (cm)		

cm.consistent(view$_1$, view$_2$)	returns TRUE if the nature of each cell is equal with respect to the given views, and FALSE otherwise	if c.nature(view$_1$) = c.nature(view$_2$)
		    if not c.consistent(view$_1$,view$_2$)
		        return FALSE
		return TRUE

cm.inconsistentCells (view$_1$,view$_2$)	returns the set of inconsistent cells with respect to the given views	for each cell c in cm.cells
		    if not c.consistent (view$_1$, view$_2$)
		        inconsistentCells.add(c)
		return inconsistentCells

opening a view incrementally. In this method, instances of a feature class are recognized in the Cellular Model using geometric reasoning.

For recognizing an instance, geometric reasoning procedures span a search tree using topologic characteristics of the feature class, in which generic feature faces are matched with Cellular Model faces. This matching is performed efficiently by using geometric tests, such as parallelism and perpendicularity, on normals of planar cell faces. The leaves in the search tree generate candidate feature instances that satisfy all geometric feature validity conditions. From these candidates, the largest feature that also satisfies all other feature validity conditions is selected as the recognized instance. In this way, valid feature instances are recognized.

The functionality of the Cellular Model that is used, is based on the notion of *cell consistency*. A cell is called consistent if its nature is the same with respect to the views considered. Similarly, views are called consistent if they represent the same geometry, i.e. if all cells are consistent. See *Table 9* for some methods to query the Cellular Model on consistency.

Such consistency is illustrated with an example. *Figure 6* depicts the volume owner lists for two consistent views. Additive and subtractive cells are depicted in light grey and white, respectively. If a hole is inserted into view I, the two views become inconsistent because the hole cell becomes inconsistent, see *Figure 7a*. It is subtractive with respect to view I, because the last owner in its list is a hole that is subtractive. Yet, it is additive with respect to view II, because the last owner in its list is a stock that is additive. This inconsistency can be resolved by, for example, inserting a similar hole in view II, see *Figure 7b*.

View consistency is checked by traversing all cells and checking their consistency. For feature recognition, firstly inconsistent cells, including their faces, are obtained. Furthermore, the normals of faces are calculated, which are defined only for cell faces that are on the boundary

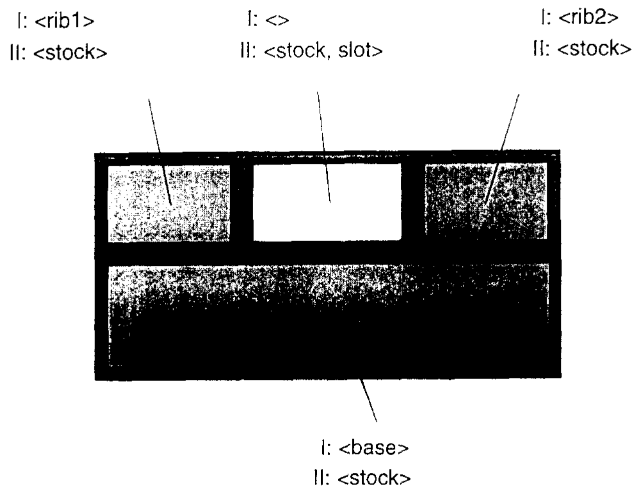


Figure 6 Examples of multiple view owner lists

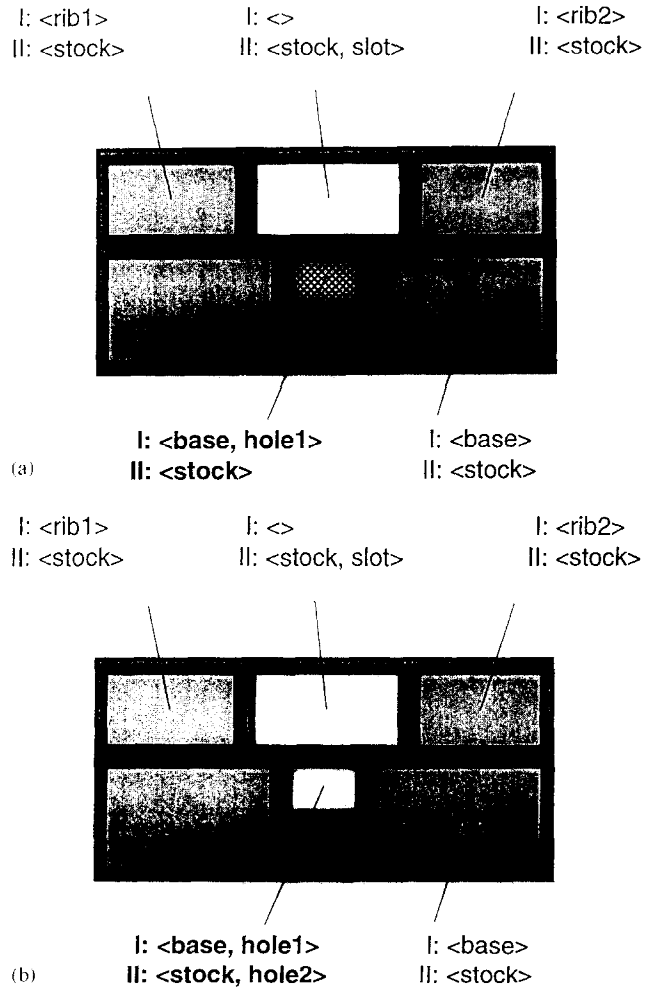


Figure 7 View consistency: (a) inconsistent views; (b) consistent views

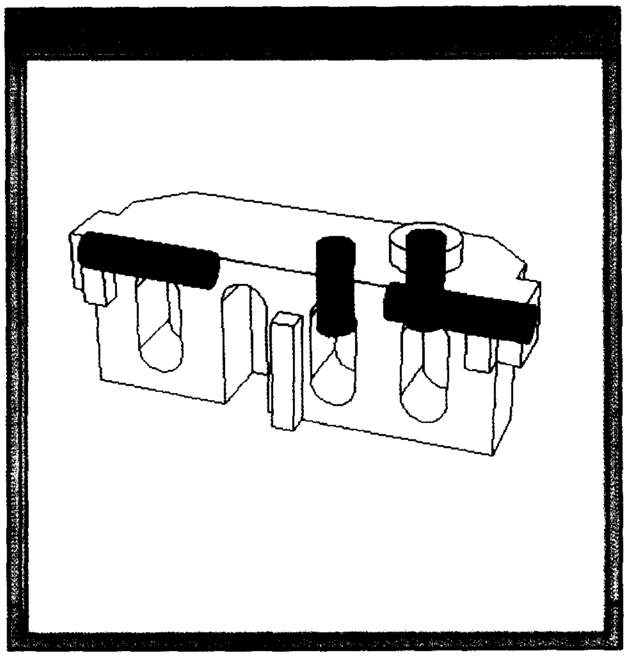


Figure 8 Distribution of holes

with respect to a view, and they point away from the additive cell they bound.

9. FEATURE VISUALIZATION

In most feature-based modelling systems, only the resulting geometry of a feature model is visualized, although features that incorporate functional product information are also important. This functional information can also be visualized to provide better insight in the feature model[16]. In SPIFF, a feature can be activated or deactivated for visualization. Activated features are visualized in a way different from the rest of the model. They can be visualized with a different display method, for example with shaded faces, whereas the rest of the model is visualized with lines, or with a different colour.

The following figures show examples of feature visualization, with different types of engineering information. In *Figure 8*, all holes are activated for visualization, showing their distribution. This could, for example, be used to evaluate manufacturability. *Figure 9* demonstrates removal of so-called occluded lines. To get a better image of an activated feature, here the green coloured step, the hidden lines of the rest of the model that are behind the visualized feature, which are called occluded lines, are removed in *Figure 9b*. In *Figure 10*, besides boundary feature elements, also non-boundary elements are visualized, showing the volume that is removed by the feature. Furthermore, the relation of parts of one feature can be shown, e.g. that the two hole parts actually belong to one hole. Lastly, *Figure 11* visualizes feature interactions by displaying the intersection region of the two slots in a different colour.

It may be clear that a conventional boundary representation does not provide the information required for generating the images depicted in *Figures 8–11*. In a boundary representation, topology entities of features may be split, missing, or merged with other entities. Furthermore, a boundary representation does not contain any feature information. The Cellular Model does provide this information in owner lists, and can therefore be used for feature visualization.

As the geometric model for visualization, a subset of the cells in the Cellular Model is used. This set consists of all additive cells, and all subtractive cells that belong to activated features. It contains exactly the information necessary for visualization: subtractive cells that would obstruct visibility are not there, but it does contain all required topological elements, with the required feature identification information in attached owner lists.

To generate an image, visualization parameters and the subset of cells in the Cellular Model are used. Examples of visualization parameters are the set of activated features and

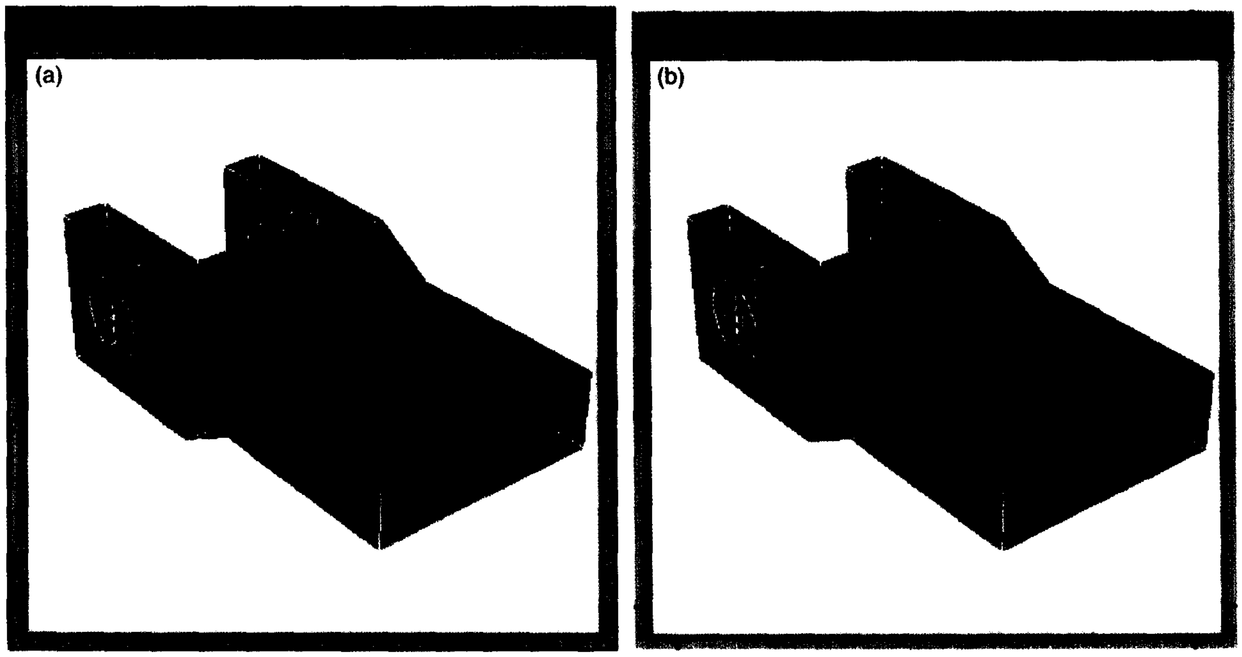


Figure 9 Occluded lines: (a) present; (b) removed

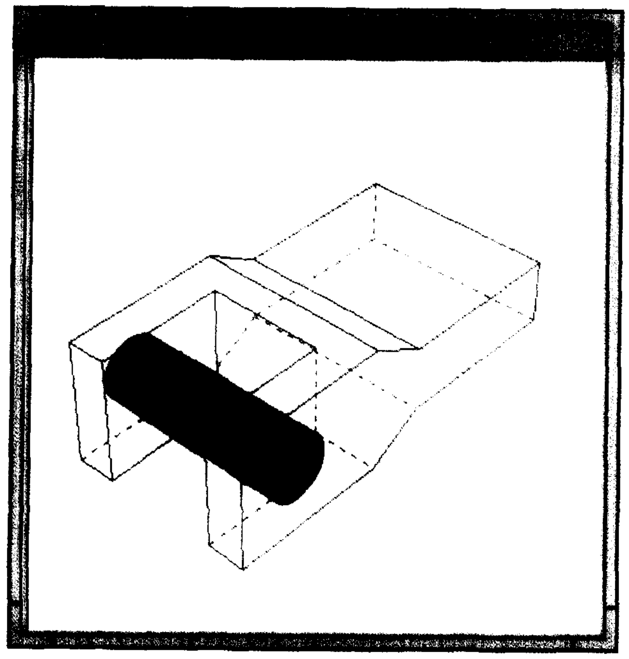


Figure 10 Non-boundary faces

their associated colours, and the display methods for boundary and non-boundary feature elements and for the rest of the model. Visibility information of the cell elements is calculated, including occluded line information. Subsequently, the cell elements are drawn in different ways, depending on the visualization parameters and on the information stored in the owner lists, e.g. whether or not an activated feature occurs in the owner list.

10. CONCLUSIONS

A cellular model is a very good basis for the geometric representation of feature models. Considerable advantages arise from integrating explicit representations for feature volumes, feature boundaries and their mutual interaction extents. The cellular model that has been presented contains all information required for maintaining high-level feature semantics. The use of attributes on cellular model entities proves to be very effective for this goal, if their propagation is consistently carried out as the model evolves. Operations that perform selective access to cellular model entities have been developed, and successfully applied to solve problems such as feature validation, feature interaction management, feature conversion between views, and advanced feature visualization.

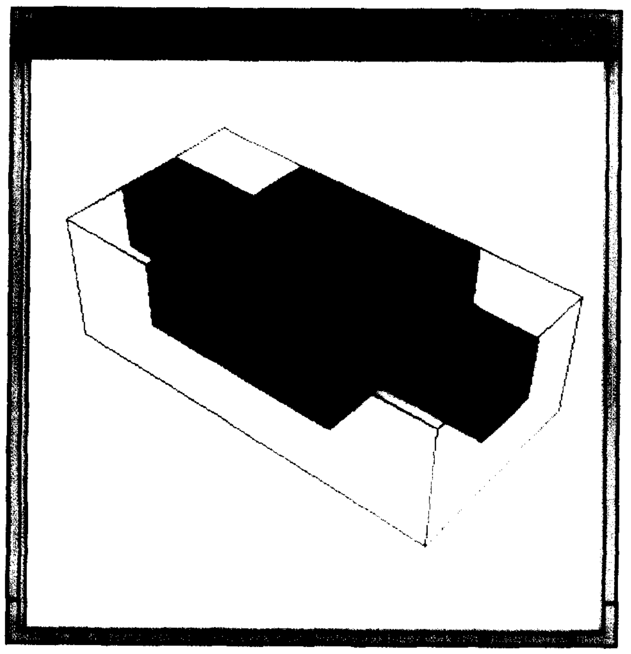


Figure 11 Feature interactions

ACKNOWLEDGEMENTS

Rafael Bidarra's work is supported by the Praxis XXI Program of the Portuguese Foundation for Scientific and Technological Research (FCT).

Klaas Jan de Kraker's work is supported by the Netherlands Computer Science Research Foundation (SION) with financial support from the Netherlands Organisation for Scientific Research (NWO).

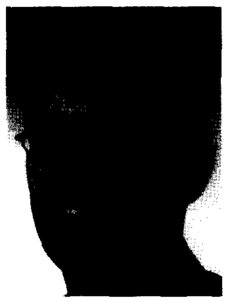
REFERENCES

1. Bronsvoort, W. F., Bidarra, R., Dohmen, M., van Holland, W. and de Kraker, K. J., Multiple-view feature modelling and conversion. In *Geometric Modelling: Theory and Practice—The State of the Art*, ed. W. Strasser, R. Klein and R. Rau. Springer–Verlag, Heidelberg, 1997, pp. 159–174.
2. Pratt, M. J., Synthesis of an optimal approach to form feature modelling. In *Proceedings of the ASME 1988 Computers in Engineering Conference*, Vol. 1, ASME, New York, 1988, pp. 263–274.
3. Weiler, K., Topological structures for geometric modelling. PhD Thesis, Rensselaer Polytechnic Institute, NY, 1986.
4. Gomes, A., Bidarra, R. and Teixeira, J. C., A cellular approach for feature-based modelling. In *Graphics Modelling and Visualization in Science and Technology*, ed. M. Göbel and J. C. Teixeira. Springer–Verlag, Heidelberg, 1993, pp. 128–143.
5. Rossignac, J. R., Issues on feature-based editing and interrogation of solid models. *Computers & Graphics*, 1990, **14**(2), 149–172.
6. Sriram, R. D., Wong, A. and He, L. X., GNOMES: an object-oriented nonmanifold geometric engine. *Computer-Aided Design*, 1995, **27**(11), 853–868.
7. Masuda, H., Topological operators and boolean operations for complex-based nonmanifold geometric models. *Computer-Aided Design*, 1993, **25**(2), 119–129.
8. Dohmen, M., de Kraker, K. J. and Bronsvoort, W. F., Feature validation in a multiple-view modeling system. In *CD-ROM Proceedings of the ASME 1996 Computers in Engineering Conference*, ed. J. M. McCarthy, ASME, New York, 1996.
9. Bidarra, R., Dohmen, M. and Bronsvoort, W. F., Automatic detection of interactions in feature models. In *CD-ROM Proceedings of ASME 1997 Computers in Engineering Conference*, ASME, New York, 1997.
10. Bidarra, R. and Teixeira, J. C., A semantic framework for flexible feature validity specification and assessment, In *Proceedings of the ASME 1994 Computers in Engineering Conference*, Vol. 1, ASME, New York, 1994, pp. 151–158.
11. Bidarra, R. and Bronsvoort, W. F., Towards classification and automatic detection of feature interactions. In *Proceedings of the 29th International Symposium on Automotive Technology and Automation*, ed. D. Roller, Automotive Automation Ltd, Croydon, 1996, pp. 99–108.
12. de Kraker, K. J., Dohmen, M. and Bronsvoort, W. F., Multiple-way feature conversion to support concurrent engineering. In *Proceedings of the Third Symposium on Solid Modeling and Applications*, ed. C. Hoffmann and J. Rossignac. ACM Press, New York, 1995, pp. 105–114.
13. de Kraker, K. J., Dohmen, M. and Bronsvoort, W. F., Multiple-way feature conversion-opening a view. In *Product Modeling for*

Computer Integrated Design and Manufacture, ed. M. Pratt, R. D. Sriram and M. J. Wozny. Chapman & Hall, London, 1996, pp. 203–212.

14. de Kraker, K. J., Dohmen, M. and Bronsvoort, W. F., Maintaining multiple views in feature modeling. In *Proceedings of the Fourth Symposium on Solid Modeling and Applications*, ed. C. Hoffmann and W. F. Bronsvoort, ACM Press, New York, 1997, pp. 123–130.
15. de Kraker, K. J., Feature conversion for concurrent engineering. PhD thesis, Delft University of Technology, The Netherlands, 1997.
16. Versluis, J. W., Bronsvoort, W. F., de Kraker, K. J. and Seebregts, K., Feature visualization. In *CD-ROM Proceedings of the ASME 1997 Computers in Engineering Conference*, ASME, New York, 1997.

Rafael Bidarra is a research assistant at Delft University of Technology since 1995. He graduated in electronics engineering at the University of Coimbra, Portugal, in 1987, and has specialized since then in computer science. His main research interests are feature modelling and geometric reasoning. He is working on a PhD project on feature validation and interaction management.



Klaas Jan de Kraker has been a research assistant at Delft University of Technology. He received his master's degree in computer science from this university in 1993. From 1993 to 1997 he worked on a PhD project on multiple-view feature modelling and conversion for concurrent engineering. Since 1997 he has worked at the research department of Baan Company in Ede, The Netherlands.

Willem F. Bronsvoort is an associate professor CAD/CAM at Delft University of Technology. He received his master's 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 interests are geometric modelling, including display and mesh generation algorithms, and feature modelling, including assembly features, feature validation and interaction, and multiple-view feature conversion. He has had many publications in international journals, books and conference proceedings. He is book review editor of Computer-Aided Design, and has served as programme co-chair of Solid Modelling '97 and as a member of several programme committees.