17

Multiple-way feature conversion - opening a view

Klaas Jan de Kraker, Maurice Dohmen and Willem F. Bronsvoort
Delft University of Technology
Faculty of Technical Mathematics and Informatics
P.O. Box 356, NL-2600 AJ Delft, The Netherlands
(DeKraker/Dohmen/Bronsvoort)@cs.tudelft.nl

Abstract

A new approach to feature modeling is presented. It supports multiple feature views of a product, each with its own feature model. Consistency between the views is maintained by performing multiple-way feature conversion. A product model has been defined that contains the different views and their feature models. The feature models of the different views are linked by the product geometry, which is represented by a cellular model.

For supporting multiple views, two types of operations are available that correspond with two types of feature conversions. Firstly, it is possible to *open* a view. The opened view is made geometrically consistent with the other views. Secondly, in line with the design by features approach, it is possible to *edit* a view. Changes made in one view are reflected in the other views. This paper focuses on opening a view. The proposed method provides a way to generically support different views.

A view is opened, i.e. a new feature model is built automatically, using the central cellular model, the feature classes available in the view, and the so-called *strategy* of a view. Each view has its own strategy that prescribes the feature model structure. It thus ensures that the feature model built is a good feature interpretation for the view.

A strategy prescribes feature classes in a certain order; instances of these classes are sought in the cellular model. Topological characteristics of a feature shape are used to generate the largest feature shape that satisfies the feature's validity conditions. The method can deal with feature intersections.

Keywords

Concurrent engineering, product modeling, feature modeling, feature conversion

1 INTRODUCTION

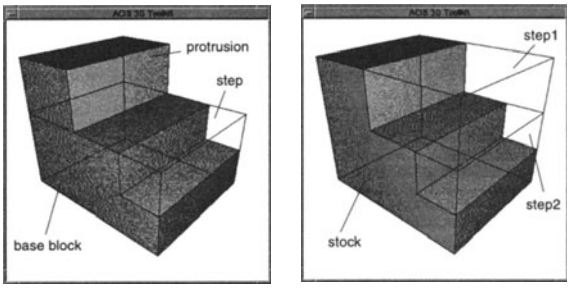
Concurrent engineering is a systematic approach to the integrated, concurrent design of products and their related processes, in particular production processes. It has been argued that concurrent engineering can lead to better product designs, e.g. in the sense that they are cheaper to produce and easier to assemble, and that it can considerably reduce product lead times (Parsaei and Sullivan 1993).

M. J. Pratt et al. (eds.), *Product Modeling for Computer Integrated Design and Manufacture*

Concurrent engineering requires a *product model* containing information for all product life cycle phases. Solid modeling only deals with information about the product geometry. In a concurrent engineering environment, also non-geometric information, e.g. functional information, is involved. This can be, for example, the function of some part of the product, or information about the way it is manufactured or assembled (Bronsvoort and Jansen 1993). *Feature modeling* does deal with such non-shape information in addition to shape information; both are represented in *features*.

Features can be used in several product life cycle activities. In the design of products, features can be used to model products with entities that are on a higher level, and closer to the way of thinking of a designer, than the entities used in geometric modeling; an example is a stepped hole. In process planning for manufacturing, features can identify areas in a product that can be manufactured in one operation with one type of equipment; an example is a slot that can be milled with a particular type of milling machine.

Each activity has its own *view* of a product, i.e. its own way of looking at it. Each view contains the features relevant to the specific activity. An example of two views of one product is given in Figure 1. In the design view, the product is represented by a base block with a protrusion and a step. In the manufacturing view, it is represented by a larger stock with two steps.



(a) (b)

Figure 1 Design and manufacturing view.

Concurrent engineering requires multiple views of one product to be supported simultaneously. Many of the activities in a concurrent engineering system will result in modifications of the design. For example, manufacturability analysis may show that dimensions of some part need to be modified. Modifications should preferably be made in the view in which the need for them arises, and all modifications in any view should be reflected in all other views. This requires *multiple-way feature conversion* from features in one view to features in the other views.

When supporting multiple views simultaneously, two types of operations are available that correspond to two types of feature conversions. Firstly, it is possible to *open* a view. Feature conversion to the opened view is then performed, making all views *consistent*. Views are consistent if they represent the same product geometry. In the view that is opened, a new feature model is built by instantiating features automatically, on the basis of the product model as already specified in all other views. Secondly, in line with the design by features approach, it is possible to *edit* a view. Either parameters can be changed, or feature and constraint instances can be created or deleted. Feature conversion from the edited view is then performed, propagating the changes to the other views. This paper focuses on opening a view. Information on how a view can be edited can be found in (Dohmen, de Kraker and Bronsvoort 1996).

Section 2 introduces SPIFF*, our multiple-view feature modeling system, outlines the problem of opening a view, and presents how feature classes are specified in SPIFF. Section 3 discusses current approaches to the problem of deriving a feature interpretation for a specific view. Sections 4 and 5 present a new method for generating feature interpretations for different views. Section 6 presents some results and conclusions of this method, and indicates directions for future research.

2 A MULTIPLE-VIEW FEATURE MODELING SYSTEM

SPIFF is our prototype system for multiple-view feature modeling. It is a feature modeling system that is not dedicated to specific views, but instead generically supports several views. Each view has its own library of feature classes, which embed constraints that specify feature validity conditions.

The *product model* contains the different views and their feature models that describe a product, see Figure 2. The feature models of the different views are linked by the product geometry.

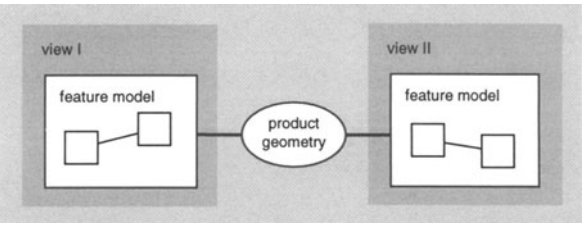


Figure 2 The product model.

In SPIFF, *feature validation* is performed, i.e. specifying and maintaining the meaning of features. Therefore, the representation of features consists of two different levels: *specification* and *maintenance*.

At the specification level, feature classes and their properties are specified, such as view-specific parameters and attributes and constraints, including relations with other features. Specifications are made in an object-oriented specification language in which three types of feature constituents are available: *nature*, *shape type*, and *validation constraints*. The nature of a feature is either additive or subtractive, specifying whether the feature adds or removes material to or from the product. Since only volumetric features are considered, the shape of a feature is represented by a parameterized solid. Several validation constraints are available that operate on feature elements, e.g. faces, and parameters. Attach constraints specify the attachment of a feature to a feature model. They specify a hierarchical relation between two feature elements, and they make one feature face coplanar to another. Semantic constraints specify topologic properties of feature elements. For a vertex, edge or face, a semantic constraint specifies the extent to which the element must lie on the product boundary. For a volume, a semantic constraint specifies the extent to which the volume is allowed to be intersected by features instantiated later. Dimension constraints specify an interval for the value of a feature parameter. For a complete description on the constraints that are used, see (Dohmen et al. 1996).

With our declarative way of feature specification using feature constituents, meaningful feature classes can be created. The validation constraint set may be extended, e.g. for specifying accessibility conditions for feature classes in a manufacturing process planning view.

*Named after Spaceman Spiff, interplanetary explorer *extraordinaire*.

At the maintenance level, feature validity is maintained. A Feature Manager controls a Constraint Manager and a Feature Geometry Manager. They store constraints and feature shapes, respectively, in their data structures, and they maintain these when the product model is edited. The Constraint Manager stores all constraint instances in a *constraint graph*. For how the Constraint Manager maintains all constraints, see again (Dohmen et al. 1996). The Feature Geometry Manager maintains a *cellular model*, which represents the product geometry and all feature intersections. Cells in the cellular model are volumetric; they can have overlapping boundaries, but they cannot have overlapping volumes. They reflect all feature intersections, and therefore can have any shape. With each cell, for each view, it is stored to which features the cell belongs. This information is stored with the cell volume, and all its faces, in an ownerlist for each view. An ownerlist is thus an ordered list of feature element names. The nature of a cell with respect to a view, is defined as the nature of the last feature in the cell's volume ownerlist. A cell is consistent if it has the same nature for each view. All views are consistent if all cells are consistent. For information on how the Cellular Model is created and maintained, see (de Kraker, Dohmen and Bronsvoort 1996).

In SPIFF, one engineer, e.g. a designer, can create an initial feature model, thus specifying the product geometry. If some other view is opened, e.g. to perform manufacturability analysis, a consistent feature model for this view must be created, which is done by automatically instantiating features. This feature model must be a meaningful feature interpretation that reflects the view's function. For finding such a feature model, the view's feature library and *strategy* are used. A view's strategy represents the feature model structure, i.e. the hierarchical organization, see Section 4. First, however, current approaches to generating a feature interpretation are discussed.

3 CURRENT APPROACHES

Opening a view comes down to generating a feature interpretation, given the product model specified by one or more other views, and the feature class library of the view. Most research in this area has focused on generating a feature interpretation for the manufacturing process planning view from a geometric model only, but sometimes design feature information is used.

There are several aspects relevant to the problem of generating a feature interpretation. One aspect is that solutions to this problem have been sought in two directions. The first is to generate all possible interpretations, and select the best one according to some criterion, e.g. cost (Tseng and Joshi 1994). The second is to generate one good interpretation only. Unlike in a manufacturing process planning view with subtractive features only, we have to deal with both additive and subtractive features. These features may intersect, which means that in a certain view, first material may be added, which later may be partly removed again, and vice versa. Due to explosion of the number of possible interpretations in the first solution, the second one is more feasible.

A second aspect is the *structure* of the generated feature model. The structure is the hierarchical organization of a feature model. It includes the feature types, in particular the feature natures. All current methods can only generate feature models with a certain fixed structure. In a manufacturing process planning view, all features are subtractive. A feature model generated by the volume decomposition method of Woo (1982) has an alternating structure; additive and subtractive features are recognized alternatingly, although the feature model may be rewritten into a model with only either additive or subtractive features. Such models thus also have a fixed structure, making the method suitable for specific views only. To make a method usable for different views, it should be generic and have control over the structure of the generated feature model. Currently there are no such methods.

A third aspect is that there are several methods that generate feature interpretations without using feature class specifications. The volume decomposition method of Woo (1982) and the cellular composition method of Dave and Sakurai (1995) are such methods. The problem of not having a feature validity specification is that valid features cannot be guaranteed. Thus a generated feature interpretation is not always meaningful.

An approach that searches for one good feature interpretation only, generically supports different feature model structures, and uses feature validity specifications, could generate meaningful feature interpretations for different views. However, there are two main problems that have to be solved for this.

Firstly, for finding one interpretation, it must be decided which feature classes must be selected to look for instances, and in what order. For this, several rules have been used (Vandenbrande and Requicha 1994). An example of a rule on the generic level is: some features are more likely to come first than others. An example of rule on the instance level is: a certain configuration of features in another view, or of topologic entities in the view that is being opened, is a hint for the existence of a certain feature.

Secondly, for finding one feature instance, its geometry must be identified. Well-known techniques for recognizing feature geometry from boundary representations are rule-based methods (Henderson 1984) and graph-based methods (Joshi and Chang 1988). Rule-based methods have the problem that they are slow. Graph-based methods have the problem that a graph is nonunique to a feature, making it difficult to decide to which class a recognized feature actually belongs. But most importantly, both techniques are based on topological adjacency relations that are easily disturbed by feature intersections, making it very difficult to recognize intersecting features.

A more robust technique, which can deal with feature intersections, is feature completion (Vandenbrande and Requicha 1994). Given characteristic traces of a feature, i.e. parts of feature faces on the product boundary, feature completion creates a feature shape. A valid feature has been found if it does not intrude the part, i.e. does not remove too much material, and satisfies dimensional and accessibility conditions.

Our goal is to develop a generic method for finding feature interpretations for several views. Since additive and subtractive features may intersect, consistency is reached nonmonotonically. Our method must be capable of controling the feature model structure so that consistency is reached.

Our approach is to have an algorithm that repeatedly identifies features of certain classes, until the view is consistent. The feature classes are prescribed by a view-specific strategy that thus determines the feature model structure. Each feature is identified using the constituents in its class specification. The feature shape is created using its shape type and validation constraints. The first of the following two sections focuses on generating feature interpretations using a strategy, the second presents how a feature, including its geometry, is identified.

4 GENERATING A FEATURE INTERPRETATION

When an engineer opens a view, a meaningful feature model must be created that is consistent with the product geometry. For this purpose, each view has a *strategy*, which specifies the feature model structure. The strategy prescribes a feature class, of which instances are to be identified in the product geometry. This is repeated until the view is consistent. Identification is successful if an instance is found.

A strategy prescribes feature classes in a certain order. This implies specification of the structure of the feature model that will be found. Strategies of different views that have the same feature classes, may prescribe these classes in different orders. By first identifying some additive features, then some subtractive features, and maybe some additive features again, a meaningful feature model for a design view is created. By first identifying a stock, and then only subtractive features, a meaningful feature model for a manufacturing view is created.

After successful identification of a feature, the same feature class is tried again. This ensures that no occurrences are missed. Currently, a strategy uses generic rules of its view that are reflected in the order in which the feature classes are prescribed. To tune a strategy, view-specific rules can be added, e.g. rules on the instance level such as Vandenbrande and Requicha (1994) use. Rules on the generic level can use the outcome of previous identifications, e.g. to skip certain feature classes.

When a feature class has been prescribed by the strategy, an instance of it is sought, see Section 5. If an instance has been found, its shape is inserted into the Cellular Model, which is used to check view consistency by comparing all cell natures for the different views.

Opening a view has been implemented by creating an abstract view class from which all realizable view classes, such as design view and manufacturing view, are derived. A member function iterates over the feature classes that its strategy prescribes. It invokes the Feature Manager with a feature class to try to identify a feature instance of this class. This is repeated until the opened view is consistent. The realizable view classes implement the strategy in a member function using a feature class identifier. If the identification of a feature was successful, another instance of this class is tried, otherwise the next feature class is tried.

The presented method does not guarantee that a feature interpretation is always found. A feature interpretation cannot be found if one does not exist. However, if feature interpretations do exist, the presented method will find a meaningful feature interpretation.

5 IDENTIFYING A FEATURE

A feature's class specification declares its validity conditions, including its shape type. We use these validity conditions for identifying an instance in the Cellular Model.

When a feature instance must be identified, in fact a shape instance that satisfies its validation constraints must be identified. Often, more than one such shape can be found. We choose the largest feature shape that satisfies the feature's validation constraints. This minimizes the number of features, resulting in a meaningful feature interpretation.

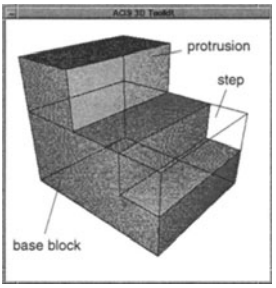
We use the attach and semantic constraints on generic feature faces for identifying a shape instance. A feature is attached with one or more faces, and its other faces have semantic constraints declaring they must at least be partly on the product boundary. These generic shape faces, which are known since the shape type is known, are matched with selected Cellular Model faces.

When a feature is instantiated by a user, it is attached to the boundary of the current feature model. Therefore, attach face instances are sought on the boundary of the currently opened view's feature model, which is not yet consistent. Since the other faces have the mentioned semantic constraints, they are sought on the product boundary. For both face types, a list of candidates is generated by querying the Cellular Model. Since inconsistent cells indicate areas where features must still be identified, candidates are selected from faces of such inconsistent cells. These candidates are matched with the generic shape faces.

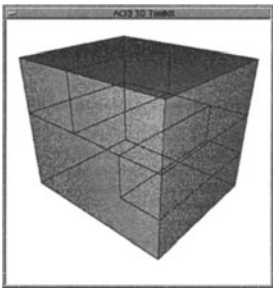
If a match is successful, the shape parameter values are derived, and a shape instance with those parameters is created. This shape is added to a list of candidate shapes. The remaining feature validation constraints of the candidate shapes, such as dimension and semantic volume constraints, are checked. The largest candidate shape that satisfies these constraints, is selected as the feature's shape. If no candidate shape satisfies the validation constraints, identification of the feature fails, and the strategy prescribes the next feature class.

In this way, a generic scheme is provided for identifying any type of feature with a certain shape. An important advantage of this method is that it can deal with any feature intersection. This is achieved by using parameter values derived from faces in the Cellular Model, rather than using topological adjacency relations, and creating a feature shape with these parameters.

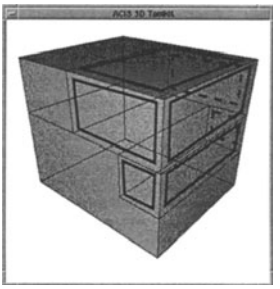
An example illustrates the presented scheme. Assume that the product geometry in Figure 3a has been created in the design view, and a manufacturing view is going to be opened. The strategy first prescribes that the stock must be identified, which is done by taking the bounding block, see Figure 3b. Then, assume that the strategy prescribes that a step must be identified.



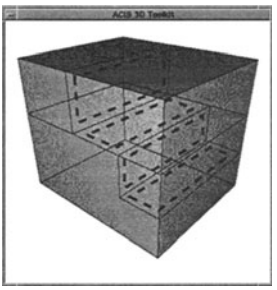
(a) Designed object



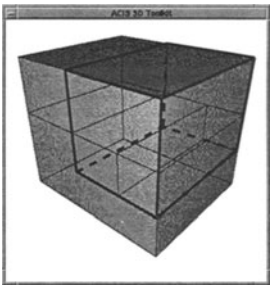
(b) Stock identified



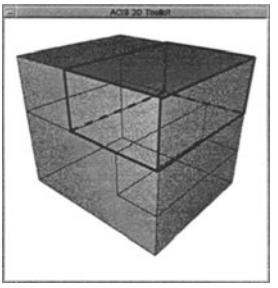
(c) Candidate attach faces



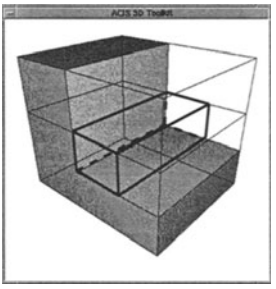
(d) Candidate other faces



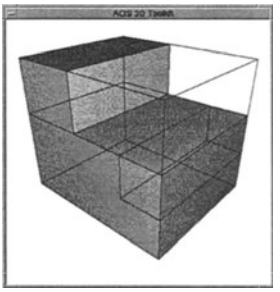
(e) Candidate shape 1



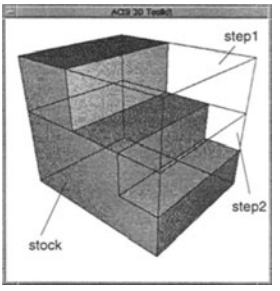
(f) Candidate shape 2



(g) Candidate shape 1 inserted



(h) Candidate shape 2 inserted



(i) Second step inserted

Figure 3 Opening manufacturing view.

The step has four attach faces, and since it is represented by a block shape, it has two other faces. Candidates for the attach faces are depicted in Figure 3c, candidate faces for the other two faces are depicted in Figure 3d. The candidate attach faces are matched with the attach faces of the feature class. The other candidate faces are matched with the other generic feature faces. Matching is performed by assigning candidate faces to the faces of the feature class. It is performed efficiently using geometric tests, including parallellism, perpendicularity, and distance. If these tests are passed, the feature shape parameter values are derived, and a shape instance is created. This results in the creation of four candidate blocks of which the largest two are depicted in Figures 3e,f. They differ in their bottom face. For the shape in Figure 3e, the lower horizontal face from Figure 3d was taken. For the shape in Figure 3f, the higher horizontal face from Figure 3d was taken. The shapes are checked against the feature's validation constraints, starting with the largest. Assume that all dimension constraints are satisfied. However, it can be seen in Figure 3g that this cannot lead to a feature interpretation: the cell indicated would become subtractive, and never can become additive again, because no additive features are available in the manufacturing view. This is detected using a semantic constraint on the volumes of the subtractive manufacturing features. Therefore the largest shape is rejected, and the second largest is tried, see Figure 3h. All its validation constraints are satistied, and thus a valid step instance has been found.

To reach view consistency, another smaller step is identified in the same way. The result is depicted in Figure 3i.

6 RESULTS, CONCLUSIONS AND FUTURE WORK

Results of the presented approach to opening a view are shown using two examples. The examples show that meaningful feature interpretations can be generated for different views.

The first example shows a way to propagate a change made in the manufacturing view back to the design view. It reopens the design view on the model from the previous section, after it has been edited in the manufacturing view: a hole has been added, see Figure 4a. Figures 4b-e demonstrate how a new feature model for the design view is created. A base block, a protrusion, a hole and a step are identified as explained earlier, after which the change has been propagated from the manufacturing view to the design view.

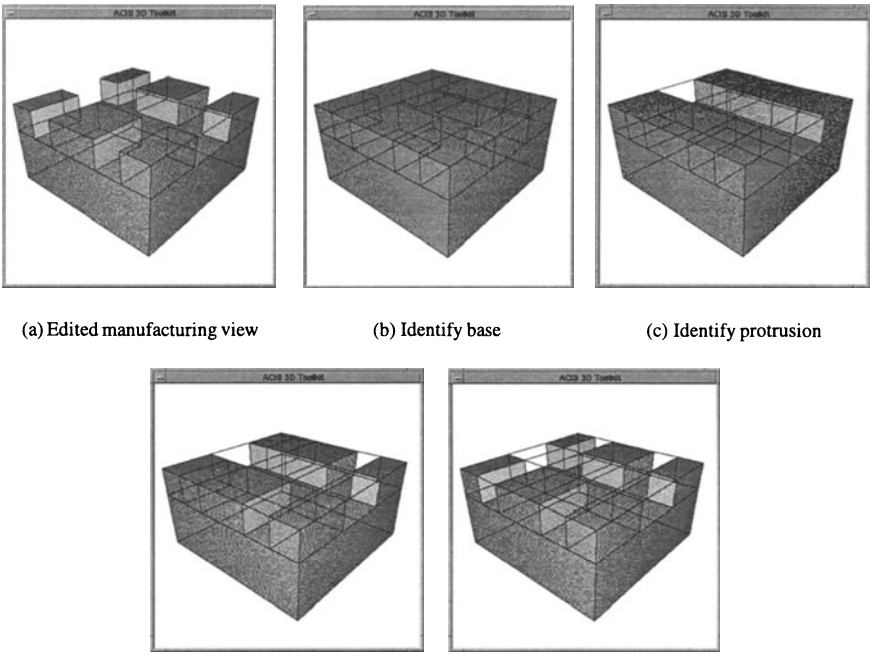
The second example demonstrates that intersecting features can be dealt with. The product depicted in Figure 5a has been created in the design view, and the manufacturing view is opened. After identifying a stock, only slots can be identified, see Figures 5b-e. First, a wide slot is identified. Then, two smaller slots are identified that intersect the first. After this, a consistent feature model for the manufacturing view has been obtained.

In conclusion, a method has been presented for generating a meaningful volumetric-feature interpretation for different views. Although the method has been tested on simple models only, the results are promising. This method generically uses view-specific information, such as the view's feature classes, and the view's strategy.

The strategy of a view controls the hierarchical organization of the feature model that is obtained. Currently, the strategies repeatedly prescribe all feature classes in a certain order. To optimize the strategies, view-specific rules can be added, e.g. to skip certain feature classes.

The feature classes of a view contain validity conditions, such as shape type and other validation constraints, which are used to identify instances. To identify an instance of a feature shape in the product geometry, an approach is used that creates a feature shape instance. In this approach, faces from the product geometry are used to derive feature parameters. A feature shape is created using these parameters. An important advantage of this approach is that it can deal with any feature intersection.

Currently, only coplanar feature attachments are used; features inserted at an angle are not yet supported. We believe that by extending the algorithm for selecting candidate faces, such cases can also be dealt with. Furthermore, the feature domain will be extended with new shape types, transition and pattern features and with view-specific features such as assembly features.



(a) Edited manufacturing view (b) Identify base (c) Identify protrusion

(d) Identify hole (e) Identify step

Figure 4 Opening design view.

ACKNOWLEDGMENTS

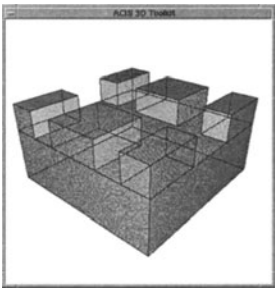
Klaas Jan de Kraker's work is supported by the Netherlands Computer Science Research Foundation (SION), with financial support from the Netherlands Organization for Scientific Research (NWO).

REFERENCES

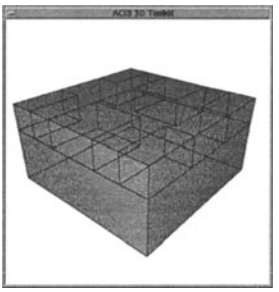
Bronsvoort, W. F. and Jansen, F. W. (1993). Feature modelling and conversion - Key concepts to concurrent engineering, *Computers in Industry* **21**(1): 61–86.

Dave, P. and Sakurai, H. (1995). Maximal volume decomposition and its application to feature recognition, *Proceedings of the 15th ASME International Computers in Engineering Conference*, pp. 553–568.

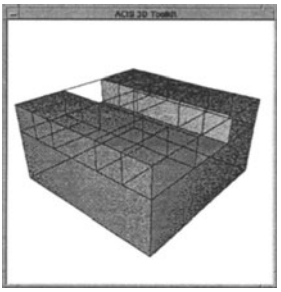
Dohmen, M., de Kraker, K. J. and Bronsvoort, W. F. (1996). Feature validation in a multiple-view modeling system, *Proceedings of the 16th ASME International Computers in Engineering Conference*. To be published.



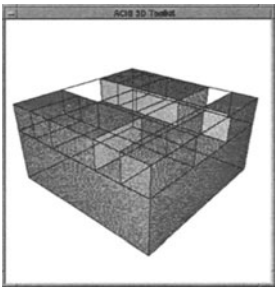
(a) Block with protrusions



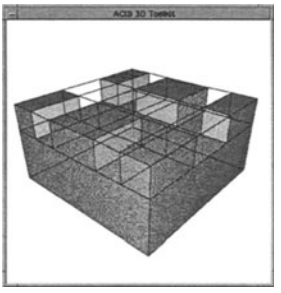
(b) Stock identified



(c) Slot 1 identified



(d) Slot 2 identified



(e) Slot 3 identified

Figure 5 Opening manufacturing view.

Henderson, M. R. (1984). *Extraction of feature information from three dimensional CAD data*, PhD thesis, Purdue University.

Joshi, S. and Chang, T. C. (1988). Graph-based heuristics for recognition of machined features from a 3D solid model, *Computer-Aided Design* **20**(2): 58–66.

de Kraker, K. J., Dohmen, M. and Bronsvoort, W. F. (1996). Feature validation and conversion, *in* D. Roller and P. Brunet (eds), *CAD Tools for Products*, Springer, Berlin.

Parsaei, H. R. and Sullivan, W. G. (1993). *Concurrent Engineering; Contemporary Issues and Modern Design Tools*, Chapman and Hall, London.

Tseng, Y.-J. and Joshi, S. B. (1994). Reconizing multiple interpretations of interacting machining features, *Computer-Aided Design* **26**(9): 667–688.

Vandenbrande, J. H. and Requicha, A. A. G. (1994). Geometric computation for the recognition of spatially interacting machining features, *in* J. J. Shah, M. Mäntylä and D. S. Nau (eds), *Advances in feature based manufacturing*, Elsevier Science B.V., Amsterdam, pp. 83–106.

Woo, T. C. (1982). Feature extraction by volume decomposition, *Proc. conf. on CAD/CAM technology in mechanical engineering*, MIT Press.