Multiple-view feature modelling and conversion

Willem F. Bronsvoort, Rafael Bidarra, Maurice Dohmen
Winfried van Holland and Klaas Jan de Kraker

Faculty of Technical Mathematics and Informatics, Delft University of Technology
Zuidplantsoen 4, NL-2628 BZ Delft, The Netherlands
email: `bronsvoort@twi.tudelft.nl`

Abstract. A product model containing information for all product life cycle activities is central to concurrent engineering. Preferably each activity has its own view on the product model, with information relevant to that activity. In this paper, a feature modelling approach is outlined in which each view consists of a specific feature model, containing features relevant to the corresponding activity. Attention is paid to the product model, feature validation, feature conversion and specific assembly features. Feature validation is the basis for maintaining the meaning of features. Feature conversion is used to convert features from one view to other views; multiple-way conversions are possible. As an example of view-specific features, it is shown which assembly-specific features can be included in a feature model for the assembly planning view.

1 INTRODUCTION

Concurrent engineering is a systematic approach to the integrated, concurrent design of products and their related processes, in particular production processes [10]. There are two main aspects to concurrent engineering.

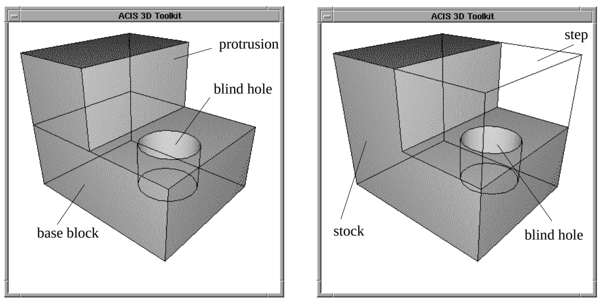
The first aspect is that already during the design phase of a product, criteria from downstream product life cycle phases, such as manufacturing and assembly, are taken into account by considering product properties and available resources. This is called *Design for X (DFX)*, where the X can stand for any product life cycle phase. Application of DFX can lead to better product designs, for example in the sense that they are cheaper to produce and easier to assemble, and to a reduction in product lead times by eliminating redesign iterations.

The second aspect is that certain activities for different product life cycle phases can be executed simultaneously. For example, part of process planning for manufacturing and assembly can already be executed when the detailed design has not yet been completely finished. Such simultaneous engineering can further reduce product lead times.

Both aspects require a product model containing information for all product life cycle phases. Solid modelling only deals with information about the geometry of a product. In a concurrent engineering environment this is a shortcoming, because here also non-shape information, eg functional information, is involved. This can, for example, be the function of some part of the product for the user, or information about the way some part of the product is manufactured or assembled [2]. *Feature modelling* 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 modelling; an example is a stepped hole, consisting of two concentric holes. In process planning for manufacturing, features can identify areas in a product that can be manufactured in one machining operation with one type of equipment; an example is a slot that can be milled with a particular milling machine. In assembly process planning, features can identify connections between parts.

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



(a) (b)

Fig. 1. Design and manufacturing view.

In this paper, an outline is given of such a feature modelling approach, with in each view a feature model specific for the corresponding activity. In Section 2, the product model is described. In Section 3, feature validation, the basis for maintaining the meaning of features, is discussed. In Section 4, a method for feature conversion from one view to other views is presented. In Section 5, a feature model specific for the assembly planning view is described. In Section 6, conclusions and future developments are discussed.

2 PRODUCT MODEL

The basic entity in our product model for concurrent engineering is a *feature*, defined as the representation of shape aspects of a physical product that are mappable to a generic shape and are functionally significant. A feature is characterized by a number of parameters, such as shape parameters. A parameterized description of a feature is called the *generic definition* of the feature. A generic feature can be instantiated multiple times by specifying values for its parameters. The topologic entities of a feature, such as vertices, edges, faces and volume, are called *feature elements*.

A feature has a certain *meaning*. An example is that a cylindrical blind hole has a circular top face, a cylindrical side face, and a circular bottom face, and, in addition, that the first face is not on the boundary of the modelled object, whereas the two other faces are.

In many of the current systems that are called feature modelling systems, very little support is given to maintain the meaning, or *validity*, of features when a model is edited. If, for example, a slot feature is inserted into a model with a cylindrical blind hole feature in such a way that the bottom face of the hole is no longer on the boundary of the object, then the feature is no longer a cylindrical blind hole, but has become a cylindrical through hole instead, and its functional meaning has thus changed (see Figure 2). Modelling systems that fail to notify this, are in essence only geometric modelling systems, but no feature modelling systems, because they do not maintain the meaning of features. In our feature models, each feature does have a well-defined meaning, and this meaning is maintained.

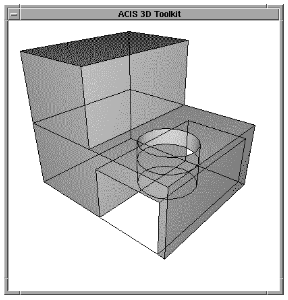


Fig. 2. Invalid blind hole.

The meaning of a generic feature is defined by its *feature validation constraints* (see Section 3). These include geometric relations within the feature, relations with other features, and validity conditions for feature elements. After feature instantiation, it is possible to specify additional relations between feature elements, possibly of different features, ie *model validation constraints*. A collection of feature instances and their relations is called a *feature model*.

Concurrent engineering requires multiple views of one product to be supported simultaneously, each view containing features relevant to its life cycle activity. Many of the activities in a concurrent engineering system will result in modifications of the product model. 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 made in any view should be reflected in all other views. This requires *feature conversion* from features in one view to features in other views (see Section 4). Such feature conversion involves management of the product's shape, which is represented by a different feature model in each view, and management of the constraints, which specify validity conditions of the features.

We are currently working on a prototype implementation of a system, called Spiff, to test our approach to feature modelling for concurrent engineering. In Spiff, a product model contains the different views and their feature models, see Figure 3. The feature models of the different views are linked by the product geometry.

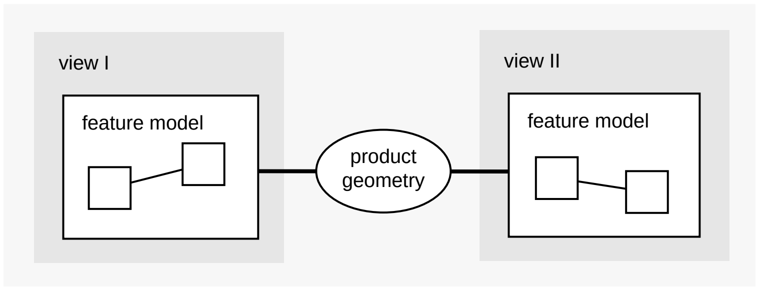


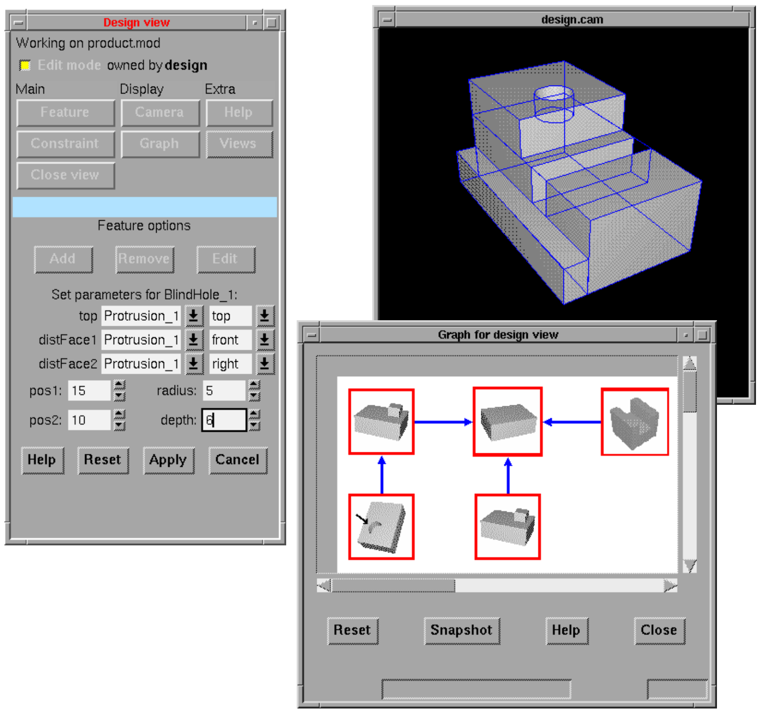
Fig. 3. The product model.

The representation of features is handled at two different levels: *specification* and *maintenance*. This separation provides a clean way of feature specification.

At the specification level, features are specified with all their properties, such as view-specific parameters and attributes and constraints. Specifications are made in an object-oriented specification language, via a graphical user interface, see Figure 4.

At the maintenance level, feature validity is maintained. A Constraint Manager and a Feature Geometry Manager store validation constraints and feature shapes, respectively, in their data structures, and they maintain these when the product model is edited, see Figure 5.

The Constraint Manager stores all constraint instances in a *constraint graph*, in which two types of nodes represent constraints and variables, and edges con-



Working on product.mod

Edit mode owned by **design**

Set parameters for BlindHole_1:

top	Protrusion_1	top
distFace1	Protrusion_1	front
distFace2	Protrusion_1	right

pos1:	15	radius:	5
pos2:	10	depth:	6

Fig. 4. The SPIFF graphical user interface.

nect constraints to associated variables. The variables are feature elements and feature parameters (see Section 3).

The Feature Geometry Manager maintains a *cellular model*, which represents the product geometry. 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 an arbitrary shape.

Features as described until now, which are often called *form features*, occur in all views on the product model, although different types may occur in different views. In addition, also other, view-specific features may be useful in each view. These features add activity-specific functional information using the form features. This has been implemented for the assembly planning view, in which, for example, connection features represent connection information between form features (see Section 5). In other views, other features may be useful besides the form features. For example, in the design view these might be conceptual features at a higher abstraction level than the form features. Such a model, with form features and other, view-specific features, is called an *enhanced feature model*.

A product model with an enhanced feature model for each life cycle activity, is ideal for a concurrent engineering system, because it contains all information

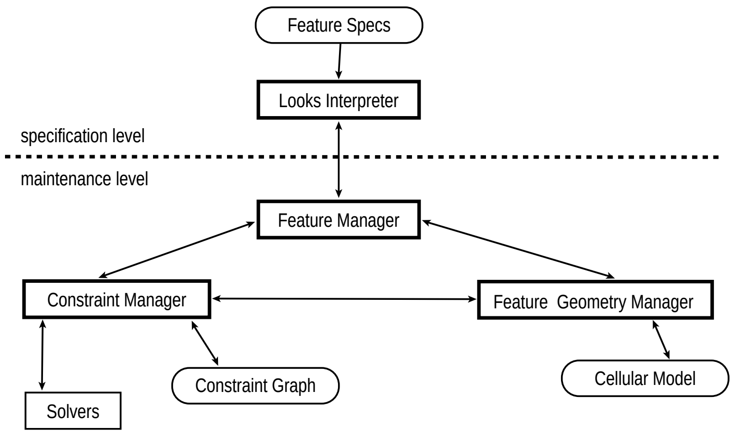


Fig. 5. Feature specification and maintenance.

required in the whole product life cycle. In the sequel, several aspects of our feature modelling approach that supports such a product model are discussed.

3 FEATURE VALIDATION

The basis to maintain the meaning of features is feature validation. Feature validation is performed by maintaining validation constraints. These are part of the definition of features and specify, for example, the kind of properties mentioned in Section 2 for the cylindrical blind hole. Both at the creation of a feature and at subsequent modelling steps, these constraints are checked to see whether the feature is still valid.

The following constraint types are used to specify feature and model validity. Shape, attach and semantic constraints are used only for specifying feature validity; the other constraints can be used both for specifying feature validity and for specifying model validity.

Shape constraints correspond to the type of feature shape, eg a block for a slot. They are used to control the relations between feature parameters and feature geometry.

Attach constraints specify the attachment of a feature to a feature model. A hierarchical relation is specified between two feature elements, eg faces, including the remaining degrees of freedom of the attached elements (see Figure 1a in which the top face of the cylinder is attached to the top face of the base block). These attachments are used instead of the parent-child relations between features used in many other systems. The advantage is that a feature instance does not necessarily have only one parent feature; it can be attached to feature elements of several features.

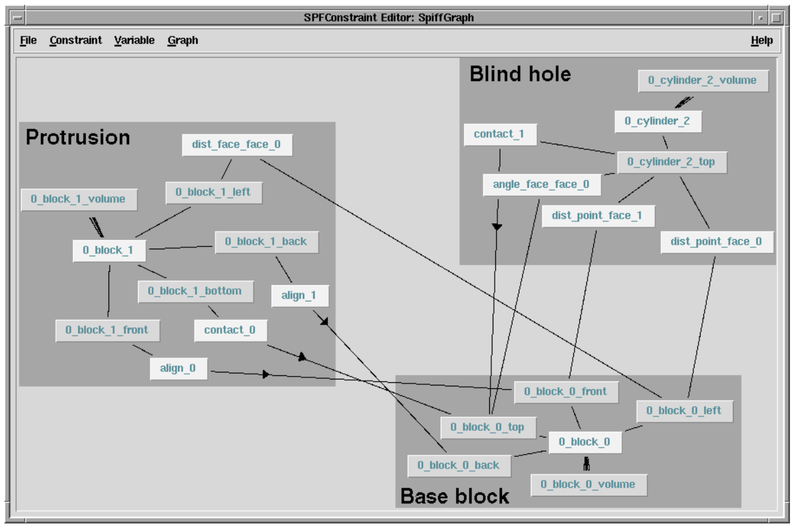


Fig. 6. Constraint graph.

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 intersect other feature volumes. The cylindrical blind hole of Figure 1a is represented by a cylinder that is subtracted. Its bottom face must be completely on the product boundary. The side face, on the other hand, must be at least partly on the boundary, and the top face may not be on the boundary. This may be the semantic specification of a blind hole in a design view. If a blind hole is part of a pen-hole connection feature in an assembly view (see Section 5), a semantic constraint on its volume may declare that no additive feature instantiated later may intersect the hole.

Geometric constraints specify geometric relations, such as parallelism and distance, between feature elements. When used as a model validation constraint, geometric constraints can dimension a feature model. An example is a geometric constraint specifying a distance between two parallel slots.

Dimension constraints specify an interval for the value of a feature parameter. An example of a dimension constraint in a manufacturing view, is a constraint on the width of a slot, declaring it to be within some specified range, because there is no milling equipment available to mill slots with smaller or larger widths.

Algebraic constraints specify equations containing feature parameters. An example is an expression for the length of the protrusion in Figure 1a to be equal to half the length of its base block.

Figure 6 shows part of the constraint graph for the feature model shown in Figure 1a. Light grey icons depict constraints, dark grey icons depict constraint variables, which are either feature elements or feature parameters. The features each have a shape constraint that connects the various feature elements.

After changing a feature model, all constraints in the graph are maintained by the Constraint Manager (see Figure 5), which handles the various constraint types by calling dedicated solvers.

For example, when a feature is instantiated, the Constraint Manager first determines the unspecified feature parameters by solving its attach, shape and geometric constraints. The dimensions of the feature are then checked against the dimension and algebraic feature validation constraints. After this, the feature shape is inserted into the geometry of the product model, ie the cellular model. Finally, the semantic feature validation constraints of the feature and its intersecting features are checked. If one of the constraints is not valid, the feature is not added to the model.

Dimension, algebraic and semantic constraints are currently only checked. Dimension constraints can be checked by testing whether the feature parameter involved has a value within the specified range. For algebraic constraints, a dedicated equation solver will be used. After the model has been edited, the semantic constraints are checked of all features that have been repositioned or redimensioned, and of the features that intersect any of these before or after the change.

To maintain the feature's attach, shape and geometric validation constraints, these constraints are mapped onto a constraint graph containing *primitive* geometric constraints that restrict translational and rotational degrees of freedom [4]. Examples of primitive constraints are a parallel-z-axes constraint and a coincident-points constraint. The primitive-constraint graph is solved using degrees of freedom analysis [9].
With the feature validation approach outlined, the validity of features can be maintained after each modelling operation. If it is detected that a feature is no longer valid, the modelling operation simply can be forbidden, and the user can then take an appropriate action, for example, change the feature parameters, or even change the feature type. For an ideal feature modelling system, however, this method is too rigid. The least that is desirable is that the user gets a good explanation on what has caused a feature to be no longer valid. A further improvement would be that he gets hints on how to avoid or overcome the problem. Ideally, the system automatically adapts the model to get a valid model again in cases where this is possible and desirable, although this should probably not be done without consulting the user. In the example in Figure 2, in which the bottom of the cylindrical blind hole was removed by the insertion of a slot into the model, the blind hole might be automatically converted into a through hole, if the user permits this.

Most validity violations result from modelling operations that cause feature interactions. Feature interactions may have a very wide range of effects on a feature model. Although these may often be intended, in many situations they may affect the semantics of a feature, ranging from slight changes in parameter values to the complete suppression of its contribution to the model shape. To get more insight in this, a taxonomy for feature interactions has been developed that takes into account relevant design and technological criteria [1].

An immediate goal of our research on feature interaction management is the detection of each interaction class, as a result of monitoring each modelling operation. This is based on the analysis of the interaction extent and feature natures, the evaluation of semantic constraints, and a variety of topologic and geometric queries to the cellular model.

Our next research goals in this area include providing the user of SPIFF with a choice of reactions and suggestions, whenever one of the above interaction situations is detected (eg readjust some feature parameters, suggest a change in attachments, or perform a change in feature type).

4 FEATURE CONVERSION

In this section, our method to convert a feature model from one view to feature models in other views is presented.

Conversions proposed until now are one-way: a designer has to input a primary view, and conversion modules are available to generate other feature models for secondary views, in particular for a manufacturing process planning view [11, 3]. If a modification in the product model is required on the basis of the outcome of an activity, this modification has to be entered in the primary view, after which new secondary views can be generated when needed.

To support concurrent engineering with multiple feature views, the solution with a primary view and a number of secondary views is far from ideal. If an engineer finds in some secondary view that, for example, a dimension of the designed product has to be adjusted, he has to switch from his secondary view to the primary view to make the adjustment. In this view, however, the feature model is different from the one in his own view, and it may be difficult to determine the right adjustments of the feature parameters. We have therefore developed an approach that supports all feature views simultaneously, and performs multiple-way feature conversion between these views [7].

When supporting multiple views simultaneously, two types of operations can be identified that correspond to two types of feature conversion. 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 automatically built, 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. These changes are reflected in the other views by propagating the changes from the edited view to the other views.

After the product model has initially been specified in one view, another view can be opened. A generic method for opening a view has been developed that uses the cellular model and view-specific information [8]. A new feature model is automatically built by instantiating features one by one, until the opened view is consistent with the other views.

Given a set of generic features of a view and the cellular model, many interpretations in terms of features exist. One approach is to generate all possible interpretations and choose one, but, since there can be very many interpretations, this is not a satisfactory solution. Therefore, we try to find a good interpretation for each view directly.

For finding a good interpretation, we assign to each view a *strategy* for identifying feature instances in the cellular model. Such a strategy uses the view's generic features, and it reflects the view's function. Of the latter, important aspects are the structure of the feature model, and the order in which the features can occur. For example, the structure of a design feature model may be mainly constructive, but with some subtractive features. A manufacturing feature model, however, may be completely destructive. In both cases the feature order is important.

To open a view, its strategy provides a sequence of feature classes, and it is tried to identify instances of these classes in that order. This is repeated until the view is consistent. To identify an instance of a feature class, an instance of its shape must be identified. We propose to choose the largest feature shape that satisfies the feature's validation constraints. To identify a feature shape, first its attach faces are identified, and then these are used to perform a directed search to identify the other shape faces. Attach faces are sought on the boundary of the view's not yet consistent feature model. The other shape faces are sought on the product boundary. Both face types are matched with the generic shape faces using topologic and geometric relations. With the faces found, a shape is constructed. After this, values for the feature's parameters are derived. If all its validation constraints are satisfied, a valid feature instance has been created. An important advantage of this approach is that it can deal with any feature intersection.

We illustrate the opening of a view with an example. Assume that the product in Figure 7a has been created in a design view, by starting with a base block, and adding a protrusion and a blind hole (this is the same view as in Figure 1a). Now a feature model in the manufacturing view is built. First a stock is identified by adding the bounding box of the designed object, see Figure 7b.

Assume that the strategy prescribes to identify a step, which is represented by a block shape. The four faces indicated in the manufacturing model in Figure 7c are selected as attach faces. The other two shape faces are sought in the model of Figure 7a. This results in the creation of the step's block shape, which is subtracted in Figure 7d.

Next the strategy prescribes to identify a blind hole. The circular face indicated in the manufacturing model in Figure 7e is selected as attach face. The side and bottom faces are sought in the model of Figure 7a. This results in the

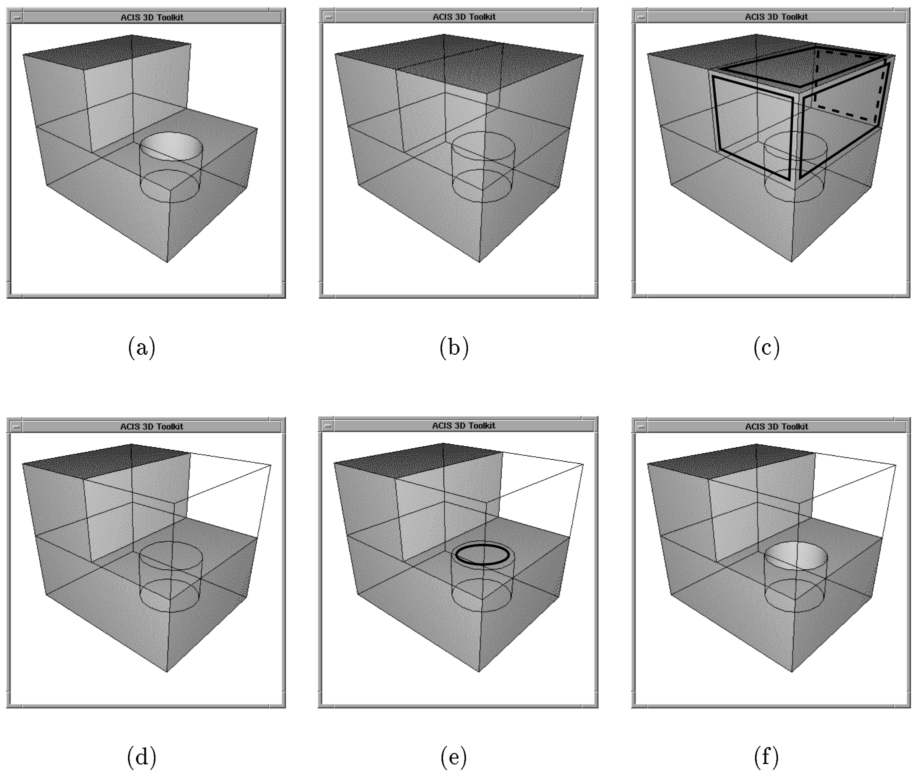


Fig. 7. Open view example.

creation of the blind hole's cylinder shape, which is subtracted in Figure 7f (this is now the same view as in Figure 1b).

Now, since the manufacturing view represents the same geometry as the design view, and thus the manufacturing view is consistent with the design view, the opening of the manufacturing view has been completed.

Editing a view requires maintaining the product model. This involves, among other things, simultaneously maintaining the validation constraints in all views.

After a view has been opened, and before a parameter can be changed, links with the other views implicitly stored in the cellular model are made explicit in geometric constraints between the views. Constraints for this are called *link constraints*, and they couple degrees of freedom of overlapping boundary feature elements of different views. These inter-view constraints are established automatically, and are considered in the constraint maintenance.

To deal with conflicting constraints, each view is assigned a priority that is used to decide whether the edit operation will be issued and propagated or not.

If the edit view conflicts with higher priority views, the change is issued only if the higher priority views can be successfully reopened. Reopening a view is efficiently done with an incremental version of the open view function. If the edit view conflicts with lower priority views, the change is issued, and it is tried to reopen the lower priority views.

5 AN ASSEMBLY FEATURE MODEL

As described in Section 2, for a specific view the form feature model can be enhanced with other, view-specific features. In this section, an enhanced feature model for the assembly planning view is discussed, and it is shown how this model can profitably be used in several assembly planning modules.

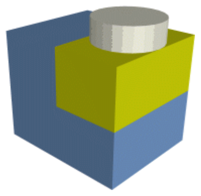
Most products need to be assembled from subassemblies and parts. These components are related to each other, often with specific connections. Characteristics of these connections are stored in so-called *assembly features.* We distinguish two types: *connection* and *handling* features, storing information on connections between components, respectively information specific for handling a component.

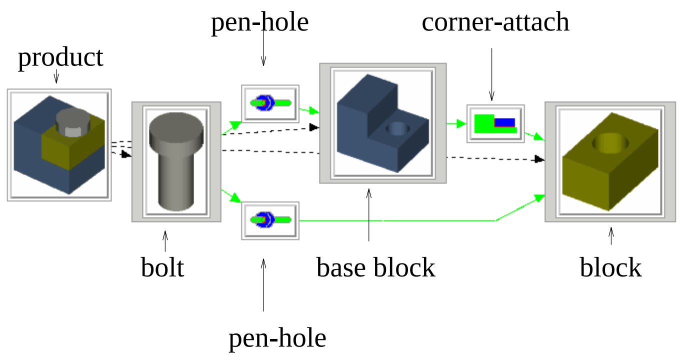
A handling feature stores for a component the areas where it can, or where it cannot, be grasped, the way it is transported to the assembly environment, the orientation on and the geometry in contact with the fixture, and the grippers that can be used to assemble the component.

With connection features, relations between components are modelled with relations between form features on the components. The information stored in these features is, among other things, used to verify whether the contacts can be made, and whether tolerances can be managed.

The idea of connection features is that characteristics of connections can be incorporated in these features, eg insertion point, insertion path, final position, tolerances, contact faces, internal freedom of motion, attachment agent, and geometric refinements such as chamfers and rounds to ease assembly operations. The final position, or goal position, is the position and orientation of the assembled component relative to the already assembled components, called the partial assembly, after the assembly operation has been completed. The insertion point is the position and orientation relative to the final position where there is not yet contact between the assembled component and the partial assembly, and where the insertion operation is started. The insertion path is the trajectory from the insertion point to the final position. Tolerances and contact faces between assembled component and partial assembly give clues for calculation of the internal freedom of motion, ie the set of motions that can separate the component and the partial assembly. An attachment agent is a component that is needed to enforce the specific connection, eg a bolt and a nut to fasten two plates are attachment agents in the connection of the plates.

The assembly feature information can be added to a model either by a designer or by an assembly planning expert. In figure 8, an example is given of an assembly model, consisting of three components (with grey border) and three


(a) product



(b) assembly feature model

Fig. 8. An assembly feature model for a product.

connection features.

It is now illustrated how this assembly feature information can be used in planning modules.

Connection features can be very useful for stability analysis, in which it has to be determined which components can move relative to each other. Each connection feature contains the internal freedom of motion between the mutually connected components. Combining the internal freedom of motions of all connection features of a component, gives information on the resulting internal freedom of motion for that component relative to other components. Because the internal freedom of motion is known for every connection feature, it is no longer necessary to calculate it from the geometry of the components, which makes stability analysis simpler and faster.

In motion planning, a path is determined to move a component from its feed position to its final position on the partial assembly. Gross motion and fine motion planning are distinguished.

During gross motion planning, a collision-free path for the assembled component from its feed position to the insertion point is searched. Because connection information is not used here, the assembly feature concept can hardly be exploited. Only the position and orientation where the gross motion must stop, and change to fine motion, can be gathered from the connection feature.

Fine motions, on the other hand, can hardly be planned without assembly features. In fine motion planning, the final stage of assembling a component onto a partial assembly is planned. Component and partial assembly are very close to each other, or even make contact. All kinds of subtle movements, including compliant movements, must be executed to make the connection. For example, for assembling a round pen-hole connection, other fine motion strategies are needed than for assembling a square pen-hole connection. Associating predefined fine-motion-strategy information with a connection feature, can result in better strategies and in a reduction of planning time.

Grasp planning is done to determine the areas on components where grippers can grasp the component. Here both handling features and connection features can be used. Handling features contain gripper information that is used to compute gripper-specific areas of a component where it either can or cannot be grasped. These areas are independent of the actual position and orientation of the component. Connection features can give additional information on areas where not to grasp, because these features specify areas that are involved in a connection. These areas are dependent on the components already assembled in the partial assembly. By combining information derived from handling features and connection features, areas can be determined where the component can be grasped. Handling features can give additional information on how to grasp the component, eg which gripper can be used and which forces the gripper should apply on the component [5].

The assembly features are also very useful for finding possible assembly sequences [6]. In assembly sequence planning, all information found in the previously presented modules is combined to determine possible assembly sequences. Finding the ideal assembly sequence, in terms of assembly time and used resources, out of all possible sequences can be very time consuming, and the assembly information stored in connection features can be used as heuristics to speed up this process. Some connection features already contain information about possible assembly sequences. A connection feature with agents for connecting two plates with a bolt and nut, for example, 'knows' that first the plates and thereafter the bolt and nut must be assembled, instead of first the bolt and nut, leaving no room for assembling the two plates.

From the above, it is clear that in many assembly planning modules it is advantageous to use the information stored in the assembly features. The presented enhanced feature model is thus useful for the assembly planning view.

6 CONCLUSIONS AND FUTURE DEVELOPMENTS

Several new concepts and methods have been presented to apply feature modelling in concurrent engineering. Each product life cycle activity can have its own view on the product model, each view consisting of a feature model with features relevant for that view.

Feature validation is an aspect of feature modelling that erroneously is neglected in most current feature modelling systems. It becomes even more important in a concurrent engineering environment, because in different views, features can have very specific meanings. Feature interaction management is a good basis for assisting the user in creating valid models.

Feature conversion as described here, for obtaining a feature model for any view, is a very promising approach to multiple-way feature conversion. Such conversion is a prerequisite in a concurrent engineering environment, because all feature views have to be simultaneously supported in such an environment.

An enhanced feature model for a view contains, besides the form features specific for that view, also other features specific for that view. This concept has been demonstrated for the assembly planning view, in which assembly features are added to the form features. It has been shown that assembly features can be used profitably in assembly planning. For other views, other enhanced feature models are foreseen.

The work described will continue. The main goal is the development of a multiple-view enhanced feature modelling system, with good facilities for feature validation, interaction management and conversion. Such a system can adequately support the kind of product model that is ideal for concurrent engineering.

ACKNOWLEDGMENTS

Rafael Bidarra's work is supported by the Portuguese Praxis XXI Program of JNICT.
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).
We thank Bart Vergouwe for implementing the SPIFF graphical user interface, and Kees Seebregts for his support in the software development.

References

1. R Bidarra and W F Bronsvoort. Towards classification and automatic detection of feature interactions. In D Roller, editor, *Proceedings 29th International Symposium on Automotive Technology & Automation*, pages 99–108, 1996.
2. W F Bronsvoort and F W Jansen. Feature modelling and conversion - Key concepts to concurrent engineering. *Computers in Industry*, 21(1):61–86, 1993.
3. P Dave and H Sakurai. Maximal volume decomposition and its application to feature recognition. In *Proceedings of the 15th ASME International Computers in Engineering Conference*, pages 553–568, 1995.
4. M Dohmen, K J de Kraker, and W F Bronsvoort. Feature validation in a multiple-view modeling system. In J M McCarthy, editor, *CD-ROM Proceedings of the ASME 1996 Design Engineering Technical Conferences and Computers in Engineering Conference*, 1996.
5. W van Holland and W F Bronsvoort. Extracting grip areas from feature information. In J M McCarthy, editor, *CD-ROM Proceedings of the ASME 1996 Design Engineering Technical Conferences and Computers in Engineering Conference*, 1996.
6. W van Holland and W F Bronsvoort. Assembly features and sequence planning. In M Pratt, R D Sriram, and M J Wozny, editors, *Product Modelling for Computer Integrated Design and Manufacture*. Chapman & Hall, 1997. To be published.

7. K J de Kraker, M Dohmen, and W F Bronsvoort. Multiple-way feature conversion to support concurrent engineering. In C Hoffmann and J Rossignac, editors, *Solid Modeling '95, Third Symposium on Solid Modeling and Applications*, pages 105–114. ACM Press, May 1995.
8. K J de Kraker, M Dohmen, and W F Bronsvoort. Multiple-way feature conversion - opening a view. In M Pratt, R D Sriram, and M J Wozny, editors, *Product Modelling for Computer Integrated Design and Manufacture.* Chapman & Hall, 1997. To be published.
9. G A Kramer. *Solving geometric constraint systems: a case study in kinematics.* The MIT Press, Cambridge, Mass., USA, 1992.
10. H R Parsaei and W G Sullivan. *Concurrent Engineering; Contemporary Issues and Modern Design Tools.* Chapman & Hall, London, 1993.
11. J H Vandenbrande and A A G Requicha. Geometric computation for the recognition of spatially interacting machining features. In J J Shah, M Mäntylä, and D S Nau, editors, *Advances in feature based manufacturing*, pages 83–106. Elsevier Science B.V., Amsterdam, 1994.