CAD/Graphics'2001
August 22-24, Kunming
International Academic Publishers

ESSENTIAL DEVELOPMENTS IN FEATURE MODELLING

Willem F. Bronsvoort, Eelco van den Berg, Rafael Bidarra and Alex Noort
Faculty of Information Technology and Systems, Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands
W.F.Bronsvoort@its.tudelft.nl

ABSTRACT

Four essential developments in feature modelling are discussed that solve serious shortcomings in current feature modelling systems. First, in semantic feature modelling, it is possible to more adequately specify and maintain the meaning of features. Second, in enhanced multiple-view feature modelling, different views on a product are provided, also for conceptual and assembly design. Third, in collaborative feature modelling, several users can collaborate on the development of a product with full feature modelling functionality. Finally, in freeform feature modelling, features with freeform faces are made available. All developments are illustrated with results from research projects at Delft University of Technology, and also future developments are mentioned.

KEYWORDS feature modelling, validity maintenance, multiple views, collaborative modelling, freeform features.

INTRODUCTION

Feature modelling is nowadays the predominant way of product modelling. Although the functionality of feature modelling systems has been considerably improved during the last decade, there are still several shortcomings. This paper discusses a number of essential developments to solve the most important. All these developments take place in the context of research projects performed at Delft University of Technology, and are implemented in the prototype feature modelling system SPIFF developed at this university [1].

A number of concepts in feature modelling that are generally accepted now, and are used in the rest of the paper, are introduced in the next section. The four developments are introduced in subsequent sections.

The first shortcoming in current feature modelling systems is that the meaning, or semantics, of features is often not adequately maintained during modelling. The semantic feature modelling approach does handle this. How the semantics of all features is specified, and how it is maintained during all modelling operations, is discussed.

The second shortcoming is that product models with multiple feature views are not yet possible for all product development phases. Current multiple-view feature modelling systems only support form feature views, which can, for example, be used for part detail design and part manufacturing planning. An approach that also supports views for conceptual design and assembly design is discussed.

The third shortcoming is that collaboration of several users in developing a product with a feature modelling system is not yet adequately supported. A web-based collaborative system that offers interactive modelling facilities, but also all functionality of a real feature modelling system, is discussed.

The fourth shortcoming is that mostly only regular-shaped features can be used, whereas in practice products often contain freeform surfaces. Because of the many benefits of feature modelling, it is worth to develop techniques for freeform feature modelling as well. The main problems that occur here are discussed.

Some conclusions about the developments in feature modelling discussed in this paper are given in the final section.

FEATURE MODELLING

This section introduces the main concepts in feature modelling in general. There is now more or less consensus in the modelling community that a *form feature* (*feature* for short) is a representation of shape aspects of a product that are mappable to a generic shape and functionally significant for some product life-cycle phase. Functional information, e.g. on the use of the shape for the end-user or on the way the shape can be manufactured, can be associated with the shape information [2].

Several types of features can be distinguished; block protrusions, cylindrical holes and rectangular slots are typical examples of frequently used ones. Although

several attempts have been made, it has turned out to be very difficult to make a general classification of features. It is therefore important that new types of features can be easily introduced in a feature modelling system. All properties of a feature type are specified in the corresponding *feature class*, which defines a template for all its instances. This always includes the canonical shape of the feature and a number of parameters that characterise this shape, e.g. the width and depth for a rectangular slot. By determining values for the parameters, an *instance* of a feature class can be created in a model. The instance is normally *attached* to other features in the model, i.e. some of its faces are coupled to faces of other features. Fig. 1 shows a model that contains several features.

In advanced feature modelling systems, several properties that correspond to functional information can also be specified. In a feature class, *feature validity conditions* can be given that all instances of the class should satisfy. An example is that the radius of a blind hole should be between 5 and 15 mm. In addition to feature validity conditions, there can also be *model validity conditions*, which specify relations on or between instances in a feature model. Examples of model validity conditions are that two slots should be parallel, and that the diameter of a hole should be half of the width of the protrusion it is attached to. Model validity conditions are specified on the instances involved. All validity conditions can be specified with *constraints* on feature entities, in particular faces or dimensions of the features.

A feature model is usually *represented* by a graph and a geometric model of the resulting shape [2]. The graph contains all feature instances, with their shape and feature validity constraints, attach relations and model validity constraints. The geometric model can be a boundary representation, but also a more extended representation, such as a cellular model [3], that is more suitable for advanced facilities such as validity maintenance and feature conversion (see next sections).

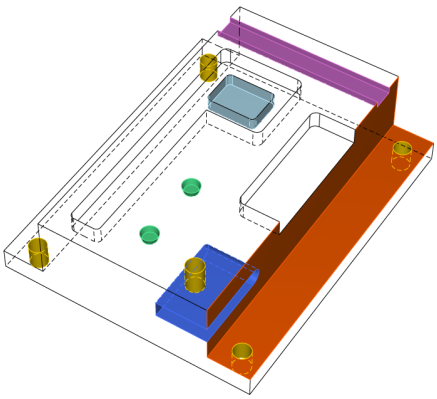

Fig. 1 Feature model.

There are many *applications* of features. In product design, shapes with some function for the end-user of the product can be considered as features. In design analysis, in particular stress analysis with the finite-element method, features may represent local stiffeners or other areas relevant for the analysis. In process planning for manufacturing, volumes in a product that can be manufactured with a single or a sequence of machining operations are considered as features. Each application has its own way of looking at a product, i.e. its own feature model of the product, with features relevant for that application. Such an application-specific feature model is called a *view* on the product [4].

Independently of their application, there are basically three ways to create a feature model: design by features, feature recognition and feature conversion.

In *design by features*, the designer specifies a feature model. He can create instances of feature classes stored in a library, by specifying values for the parameters, and add them to the model. In some systems, also new feature classes, so-called *user-defined features* [5] can be created by a user, after which instances from these can be added to the model. Feature instances can also be modified, by changing the value of their parameters, or be removed from the model. The features that are specified may well have a functional meaning for the end-user of the product, but can also be manufacturing features. In the latter case, the designer specifies a model that, more or less, corresponds to the way the product will be manufactured.

In *feature recognition*, features are recognised from a geometric model of a product. Historically, this was the first method to create a feature model, introduced in the context of manufacturing planning. Many methods for feature recognition exist, each with its own advantages and disadvantages [2]. The four most important categories of feature recognition methods are rule-based, graph-based, volume decomposition and geometric reasoning methods. Some successful feature recognition systems use combinations of these methods.

Once a feature model of a product has been created, either by design by features or by feature recognition, other feature models of that product, which correspond to other views, can be derived by *feature conversion*. For example, a manufacturing planning view can be derived from a design view. Feature conversion is a relatively new technique, and forms the basis for *multiple-view feature modelling* systems (see corresponding section).

FEATURE VALIDITY MAINTENANCE

In many current feature modelling systems, "features" only occur at the user interface level, whereas in the product model only the resulting geometry is stored. Such systems are in essence only geometric modelling systems. In other systems, see for example [6], information about features is stored in the product model, but it is not consistently checked that the meaning of all features is maintained during the whole modelling process. For example, a through hole can be turned into a blind hole by blocking one of the openings of the hole with a stiffener, without the system even notifying this change, see Fig. 2. Although geometrically this is correct, it is incorrect in the sense that the meaning, or *semantics*, of the feature is changed from a through hole into a blind hole.

Ideally, all validity conditions of the features, i.e. their semantics, should be checked by the system after each modelling operation. If some validity condition is no longer satisfied, e.g. the top face of a hole is no longer open, this should be notified by the system, and preferably the user should be assisted in overcoming this situation. An approach that supports these ideas is *semantic feature modelling* [7]. This approach guarantees that all design intent once captured in a model is maintained, bringing feature modelling to a really higher level than advanced geometric modelling. It involves specification of the semantics of features in their respective classes, and maintenance of this semantics during modelling.

Specification of *validity conditions* in a feature class can be classified into two categories: geometric and topologic. For both, constraints are used. These *feature constraints* are members of the feature class, and are therefore instantiated automatically with each new feature instance.

First of all, the geometry of a feature may be constrained by *geometric constraints*, e.g. requiring that some faces should be parallel or perpendicular. Another way of constraining the geometry of a feature, is by restricting the set of values allowed for a shape parameter with *dimension constraints*. For instance, the radius parameter of a through hole class could be limited to values between 1 and 10 mm. Finally, the geometry of a feature may be constrained by means of explicit relations among its parameters. These relations can be simple equalities between two parameters (e.g. between width and length of a square passage feature) or, in general, algebraic expressions involving two or more parameters and constants. For this, we use *algebraic constraints.*

The set of shape faces of a feature provides full coverage of the feature boundary. However, for most

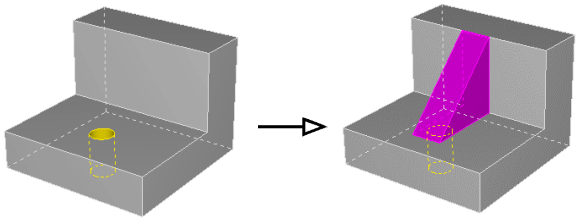

Fig. 2 Changing a through hole into a blind hole.

features, not all faces are meant to effectively contribute to the boundary of the modelled product. Some faces, instead, have a closure role, delimiting the feature volume without contributing to the boundary. The specification of such properties is called *topologic validity* specification. For this, we use two sorts of constraints: boundary constraints and interaction constraints.

A *boundary constraint* states the extent to which a feature face should be on the model boundary. An example of this is a blind hole class for which the entrance face has a *notOnBoundary* constraint, whereas the bottom face has an *onBoundary* constraint.

Boundary constraints are insufficient to fully describe several other functional aspects that can be inherent to a feature class as well. These are better described in terms of the feature volume or feature boundary as a whole. An example of this is the requirement that every feature instance of some class should somehow contribute to the shape of the part model. Such functional requirements can be violated by feature interactions caused during incremental editing of the model.

Feature interactions are modifications of shape aspects of a feature that affect its functional meaning. An example of this is the *transmutation interaction* of the through hole into a blind hole in Fig. 2. We use *interaction constraints* in a feature class to indicate that a particular interaction type is not allowed for its instances.

Embedding validity conditions in each feature class and checking them when an instance is created, enhances the modelling process, as it guarantees that at the creation of a feature instance its semantics effectively matches the requirements of its class. In fact, one of the basic ideas of feature modelling is that functional information can be associated to shape information in a feature model. This association, however, becomes useless when the shape imprint of a feature, once added to the model with a specific intent, is significantly modified later by a modelling operation.

Feature model validity maintenance is the process of

monitoring each modelling operation in order to ensure that all feature instances still conform to the validity criteria specified for them.

Validity maintenance can be split into two types of tasks: (i) *validity checking*, performed at key stages of each modelling operation; and (ii) *validity recovery*, performed when a validity checking task detected a violation of some validity criterion.

The basic idea of model validity maintenance is that a modelling operation, to be considered valid, should entirely preserve the design intent specified with features as well as model constraints. In other words, after a valid modelling operation, the feature model conforms to all its constraints.

If a constraint is no longer valid because of a modelling operation, e.g. a boundary constraint on the entrance face of a blind hole is violated by blocking the face with another feature, the model enters an invalid state, and a valid model should be achieved again. This is straightforward if the operation is cancelled: all that is needed is to backtrack to the valid model state just before executing it, by "reversing" the invalid operation.

However, to always have to recover from an invalid operation by undoing it is too rigid. It is often much more effective to constructively assist the user in overcoming the constraint violations, in order to restore model validity. In most cases, if the user receives appropriate feedback on the causes of an invalid situation, it is likely that other corrective actions might preferably be chosen. We call this process *validity recovery*, and it includes reporting to the user constraint violations, documenting their scope and causes, and, whenever possible, providing context-sensitive corrective hints.

The user can specify several modelling operations in a batch and execute them, in order to overcome the invalid model situation. Execution of these *reaction operations* follows the same scheme as normal operations, which means that their outcome is checked for validity. At any stage when the model is invalid, the user may give up attempting to fix it and backtrack to the last valid stage.

The specification of reaction operations is supported by automatically generated hints, which document each constraint violation detected and suggest solutions. These vary with the type of constraint involved. In the example of the blocked entrance face of a blind hole, the feature that blocks the face is reported to the user, who can then reposition it to solve the problem.

In all cases, the scope of the reaction choices made available is restricted to those features and model constraints that are somehow involved in the invalid situation. This assists the user in concentrating validity recovery efforts on effective and meaningful reactions.

The semantic feature modelling approach has been fully implemented in the SPIFF modelling system. It uses a feature dependency graph and a cellular model as main model representations [7,3]. Gao et al. [8] have presented an alternative approach to semantic feature modelling, closer to current commercial feature modelling systems.

MULTIPLE-VIEW FEATURE MODELLING

Multiple-view form feature modelling is a product development approach that combines concurrent engineering and feature modelling. *Concurrent engineering* aims at designing better products in less time, by using Design for X (where X stands for any product life cycle phase) [9] and by enabling simultaneous activities in several product development phases [10].

Multiple-view form feature modelling supports applications from various phases of product development, by providing interpretations of, or *views* on, a product for each of these applications. Each view contains a form feature model specific for the application. Since the feature models of all views represent the same product, they have to be kept *consistent*.

Quite a lot of research has been done on multiple-view form feature modelling during the last years. In the SPIFF modelling system, applications from several product development phases are integrated by providing a view with a constraint-based form feature model for each of them, and combining these feature models into one product model [1]. It uses *multiple-way form feature conversion* to allow changes in the feature model of any view to be propagated to the feature models of the other views. Other approaches to multiple-view form feature modelling have been presented by, among others, De Martino et al. [11] and Hoffmann and Joan-Arinyo [12]. All these approaches share a number of shortcomings.

First, all approaches focus on the later product development phases in which the geometry of the product has been fully specified. They use the geometry of the product as a basis, and, therefore, cannot be applied in the early product development phases, such as conceptual design.

Second, all approaches deal with a single part only. Real products, however, rarely consist of a single part. Dealing with products that consist of multiple parts, i.e. assemblies, does not only involve the separate parts, but also the relations between these parts.

To support the early product development phases and products with multiple parts, the *enhanced*

multiple-view feature modelling approach has been developed. Here, instead of form feature models, enhanced feature models are used. Such models still contain features, but these are here defined as aspects of the product that have some functionality. The features can be form features, but are in general features at a higher abstraction level. A prototype enhanced multiple-view feature modelling system is being developed based on the SPIFF modelling system. It supports four product development phases (see Fig. 3).

The first phase, in which the product architecture is determined by specifying components and their interfaces, is *conceptual design*. Components are built of a base shape, concepts, such as depressions and protrusions, and reference elements. Interfaces between components are specified by means of degrees of freedom between the components. The complete geometry of the components does not have to be specified. For example, for some concept only certain properties, such as its maximum volume, can be specified. An example of a conceptual design view for a bench vice, consisting of a base yaw, a moving yaw and a spindle, is given in Fig. 3a.

The second phase, in which the physical connections between the parts are determined, is *assembly design*. The connections are represented by *connection features*, such as dove-tail and pen-hole connection features; see also [13]. An example of an assembly design view of the bench vice of Fig. 3a, with the form features for the connections between the components, is given in Fig. 3b.

The third phase, in which the details of the geometry of parts are determined, is *part detail design*. Detail design features are form features; examples are a through hole and a rectangular protrusion. An example of a part detail design view for the base yaw part of the bench vice of Fig. 3a, is given in Fig. 3c.

The fourth phase, in which the way each part is to be manufactured is determined, is *part manufacturing planning*. Manufacturing planning features are again form features, such as slot and hole. An example of a manufacturing planning view for the base yaw part of the bench vice of Fig. 3a, is given in Fig. 3d.

The enhanced multiple-view feature modelling approach keeps the feature models of all views consistent. The views can be divided into a group of views that deal with all components of a product and the relations between them, i.e. the *aggregate-oriented views*, and a group of views that deal only with a single part, i.e. the *part-oriented views*. The conceptual design view and the assembly design view are the aggregate-oriented views, and the part detail design views and the part manufacturing planning views are the part-oriented views (see Fig. 4). The part-oriented views on a part should represent the same part. The aggregate-oriented views should represent the same product, and represent the parts from the part-oriented views as (sub)components that are related. If the views satisfy these requirements, they are said to be *consistent*. Feature conversion is used to achieve this. It involves linking features, mapping features and recognising features [14].

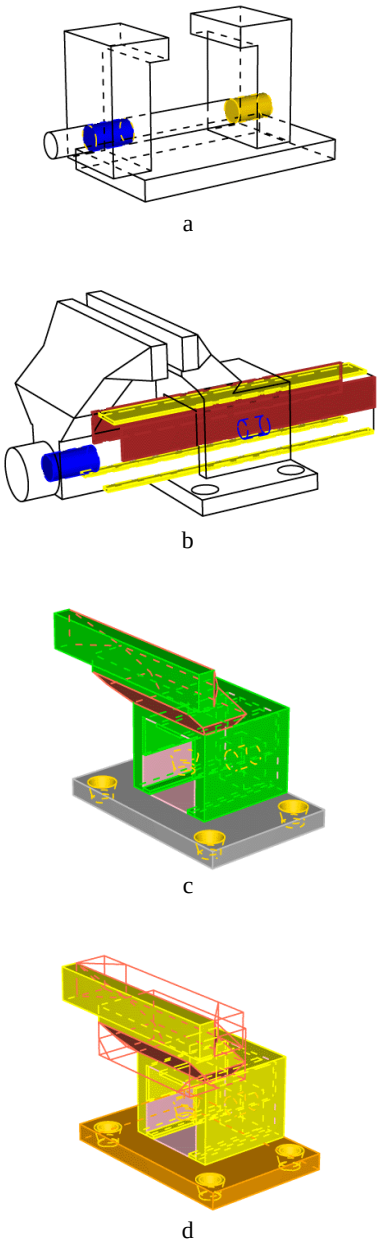

Fig. 3 A conceptual design view (a), an assembly design view (b), a part detail design view (c), and a part manufacturing planning view (d).

Some information is given here on how a designer might work with the various views.

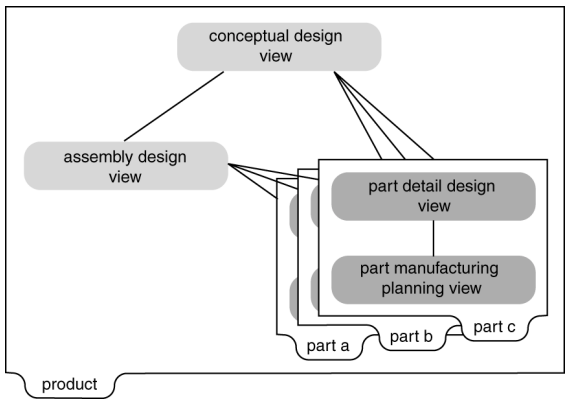

Fig. 4 Relations between the supported views: the conceptual and assembly design view deal with the whole product, whereas the part detail design and manufacturing planning views deal with the individual parts.

The *conceptual design view* allows the designer to specify the product architecture with components and interfaces.

The *assembly design view* allows the designer to refine the interfaces between the components in the conceptual design view. A connection feature needs to be created for each interface in the conceptual design view, and linked to that interface. The interface and the connection feature should reduce the same freedom. In order to accommodate the connection feature, form features may be created on the components in the assembly design view.

The *part detail design views* allow the designer to refine the parts that are represented by the components in the conceptual design view, and which may have been refined in the assembly design view to accommodate connection features. Form features may be created for concepts in the conceptual design view, and linked to those concepts. A form feature should satisfy the requirements specified for the corresponding concept, e.g. that the volume should be less than 80 cm^3. In addition, form features are automatically created to represent the regions of a part that correspond to the form features of the connection features on the related component.

The *manufacturing planning views* allow the designer to analyse the parts for manufacturability and to create a manufacturing plan for them. The feature model in a manufacturing planning view is linked to the feature model in the corresponding part detail design view.

In all cases, when the feature model of a view is changed, the feature models of the other views involved are made consistent, in order to check whether their requirements are still satisfied.

Implementation of enhanced multiple-view feature modelling in the SPIFF modelling system is near completion [14]. Experiences so far have shown that feature views that represent only those aspects of a product that are relevant in the associated product development phase, can significantly increase the insight of the designer into the product model.

COLLABORATIVE FEATURE MODELLING

Collaborative modelling systems are distributed multiple-user systems aimed at supporting engineering teams in co-ordinating their modelling activities.

So far, only a small number of tools have been developed that somehow support collaborative design activities. For example, tools for collaborative model visualisation via Internet are now becoming available, providing concepts such as shared cameras and telepointers. However, such tools are primarily focused on *inspection*, e.g. using simple polygon mesh models, and do not support real *modelling activities*. In other words, they are valuable assistants for teamwork, but no real CAD systems.

An interesting research challenge consists of developing a collaborative feature modelling system that offers all facilities of advanced feature modelling systems to its users, while at the same time providing them with the necessary co-ordination mechanisms that guarantee an effective collaboration. Typically, in a collaborative session, different participants would be provided with their own, application-specific views on the product model. The co-ordination mechanisms should solve the critical problems of concurrency and synchronisation that characterise collaborative design environments. *Concurrency* involves management of different processes trying to simultaneously access and manipulate the same data. *Synchronisation* involves propagating modified data among users of a distributed application, in order to keep their data consistent. This section discusses webSPIFF, a web-based collaborative feature modelling system that is a major step in this direction.

webSPIFF has a client-server architecture, consisting of several components (see Fig. 5). On the server side, two main components can be identified: the SPIFF modelling system, providing all feature modelling functionality; and the Session Manager, providing functionality to start, join, leave and close a modelling session, and to manage all communication between SPIFF and the clients.

The clients perform operations locally as much as

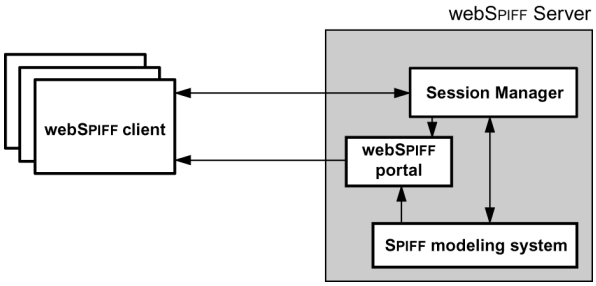

Fig. 5 Architecture of webSPIFF.

possible, in particular regarding visualisation of, and interaction with, the feature model. Only high-level semantic messages, e.g. specifying modelling operations, as well as a limited amount of model data necessary for updating the client data, are sent over the network.

The server co-ordinates the collaborative session, maintains a central product model, and provides all functionality that cannot, or should not, be implemented on the client. In particular, as soon as real feature modelling computations are required, they are executed at the server, on the central product model, and their results are eventually exported back to the clients.

An important advantage of this architecture is that there is only one central product model in the system, thus avoiding inconsistency between multiple versions of the same model.

As a basis for the server, the SPIFF modelling system was taken, which offers several advanced modelling facilities. As already discussed in the previous sections, it offers feature validity maintenance functionality and multiple views on a product model. In addition, SPIFF offers *sophisticated feature model visualisation* techniques, which visualise much more specific feature information than most other systems do. For example, feature faces that are not on the boundary of the resulting object, such as closure faces of a through slot, can be visualised too. All these facilities are computationally expensive, and require an advanced product model, including a cellular model with information on all features in all views [3].

The Session Manager stores information about an ongoing session and its participants. It manages all information streams between webSPIFF clients and the SPIFF modelling system. Since several session participants can send modelling operations and queries to the webSPIFF server at the same time, concurrency must be handled at the Session Manager. It is also the task of the Session Manager to synchronise session participants, by sending them updated model data, after a modelling or camera operation has been processed. The Session Manager has been implemented using the Java programming language.

The clients of webSPIFF are using standard web browsers. When a new client connects to the webSPIFF portal, a Java applet is loaded, implementing a simple user interface, from which a connection with the Session Manager is set up.

Once connected to the server, the user can join an ongoing collaborative session, or start a new one, by specifying the product model he wants to work on. Also, the desired view on the model has to be specified. The current version of webSPIFF provides two views: one for design and another for manufacturing planning of parts. Information on the feature model of that view is retrieved from the server, and used to build the client's graphical user interface, through which the user can start active participation in the modelling session.

Obviously, clients should be able to specify modelling operations in terms of features and their entities; for example, a feature, to be added to a model, should be attachable to entities of features already in the model. After a feature modelling operation, with all its operands, has been fully specified, the user can confirm the operation. The operation is then sent to the server, where it is checked for validity and scheduled for execution. This can result in an update of the product model on the server, and thus also of the feature model in the view of each session participant.

In addition to the above functionality, several visualisation and interactive facilities of the SPIFF system have also been ported to the clients. webSPIFF clients provide two ways of visualising the product model, both making use of so-called *camera* windows, i.e. separate windows in which a graphical representation of the product model is shown. First, a *sophisticated feature model image* can be displayed. Second, a *visualisation model* can be rendered that supports interactive modification of camera viewing parameters, e.g. rotation and zoom operations. After the desired viewing parameters have been interactively set, they are sent to the server, where a sophisticated feature model image corresponding to the new parameter values is rendered and sent back to the client, where it is displayed. webSPIFF cameras also use a *selection model*, providing facilities for interactive specification of modelling operations, e.g. assisting the user in selecting features or feature entities by having them picked in a sophisticated image of the model.

To support all these facilities, the webSPIFF clients need to locally dispose of some model data. This data is derived by the server from its central model, but it does not make up a real feature model. webSPIFF clients need just enough model information in order to be able to autonomously interact with the feature model, i.e. without requesting feedback from the server. Client

model data is never modified directly by the clients themselves. Instead, updated model data is sent back to the client after a modelling operation has been executed at the server. In addition, if the central feature model has been changed, appropriate updated model data is sent to all other session participants as well. This consists of, possibly several, new model images, a new visualisation model, and an incremental update of the selection model.

A good distribution of the functionality between the server and the clients has resulted in a well-balanced system. On the one hand, the server provides participants in a collaborative modelling session with the advanced functionality of the SPIFF feature modelling system. On the other hand, all desirable interactive modelling functionality is offered by the clients, ranging from display of sophisticated feature model images to interactive model specification facilities.

All functionality has been implemented in the webSPIFF prototype system [15], of which a demo version is available on Internet for users to experiment with, at www.webSPIFF.org.

FREEFORM FEATURE MODELLING

In almost all current feature modelling systems, only regular-shaped (form) features, i.e. features having (rounded) prismatic and cylindrical shapes, can be used. However, products in practice often contain freeform surfaces. Therefore, a research project has recently been started on freeform feature modelling. Many concepts from modelling with regular-shaped features can be mapped to freeform feature modelling in a straightforward way. It will be indicated what has already been achieved here, and which issues are still open.

Freeform features can be defined in the same way as regular-shaped features. They still correspond to generic shapes, the only difference being that there is more modelling freedom for the shape of the features; typically, their faces can be modelled with NURBS [16].

In freeform feature modelling, the general outline of a product is often created in the initial phase of the modelling process by defining a *primary feature*, which here can be a freeform volumetric shape. Later, *secondary features* can be attached to the primary feature in order to adjust the product model, while preserving its global outline. Secondary freeform features are also referred to as *detail features*.

Features like protrusions, holes and slots can still be used in this context, but now with freeform faces. In addition, many new types of useful features can be envisaged. The definition of a freeform feature class is more complicated. The canonical shape now has to be modelled with, for example, NURBS. In addition, a meaningful set of parameters has to be chosen that makes intuitive instantiation and modification of the feature possible, and a mapping between these parameters and the low-level definition entities of the NURBS has to be established. Instances can again be created in a model by determining values for the parameters, but their "attachment" to other features is more complicated than that of a regular-shaped feature. In Fig. 6, two freeform feature models are shown, with different parameter values for some features.

Au and Yuen [17] indicate that the most important relation between individual freeform features in an object is how they are connected, i.e. the order of continuity between adjacent faces of the features. Smooth transitions between adjacent faces are often realised by *blends*. They are mainly used to prevent sharp cuts, but can also be useful in closing gaps that occur when adjacent faces do not connect properly.

The idea that properties that correspond to functional information can be included in freeform feature classes, and on or between freeform feature instances, by *validity conditions*, has hardly been explored, but seems nevertheless very promising. One can easily imagine conditions that all instances of some freeform feature class have to satisfy, e.g. that the curvature of their side faces is limited, or conditions on interaction between feature instances, e.g. that the area modified by another feature is limited. Such validity conditions can again be specified with constraints, although these may become quite complex.

A freeform feature model can still be represented by a graph, with all feature instances, attach relations and model validity constraints, and a geometric model of the resulting shape, e.g. a boundary representation with NURBS patches. A more advanced geometric model than a boundary representation may be desirable here too, but this has to be further explored.

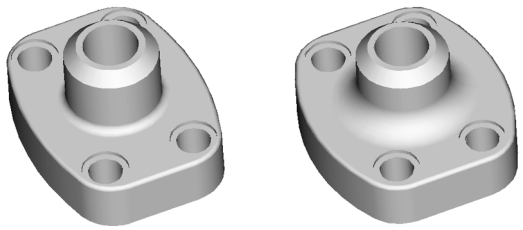

Fig. 6 Variations of a freeform feature model: (a) is standard model and (b) is model with different parameter values for some features.

The possible *applications* of freeform features, the corresponding *views* on a product, and the ways to create a feature model, are similar to those for regular-shaped features. This is why freeform feature modelling can be regarded as an extension of modelling with regular-shaped features. However, many difficult problems have to be solved to make freeform feature modelling mature. Some of these problems are mentioned here.

In *design by freeform features*, the creation of individual freeform features by specifying values for their parameters is again the basic idea. Because of the large shape domain of freeform features, a good mechanism to define new feature classes is indispensable. The canonical shape, the parameterisation and the validity conditions should be easy to specify for a user. In addition, ways have to be provided to indicate how individual freeform features in an object have to be "attached" to the model, including the order of continuity between adjacent faces of features.

Validity maintenance has hardly been explored for freeform features. As already remarked above, one can easily imagine useful validity conditions for freeform features. However, these become only really useful if they are maintained during the modelling process. How this can be achieved, will be a major issue of our future research in this area.

Although some methods have been proposed for *freeform feature recognition*, these are not yet mature. Considering the long history in, and large variety of currently available methods for, recognition of regular-shaped features, much more research is to be expected here.

Feature conversion and *multiple-view feature modelling* are other subjects that have not yet been tackled in the context of freeform features. Several topics mentioned above come together here, such as parameterisation of features, validity specification and maintenance, and feature recognition.

Research on freeform feature modelling is in its preliminary stage. A survey on the most important concepts has been produced [18], and some initial implementation work has been done.

CONCLUSIONS

Four essential developments in feature modelling were discussed in this paper. Here, some conclusions about these developments and some future work are discussed.

The semantic feature modelling approach has turned out to be a very useful and powerful way of fulfilling a major goal of feature modelling: storing and maintaining the design intent in a product model. However, more research should be done on defining the semantics of features. Although several types of constraints are now available to impose certain validity conditions on models, many other validity conditions should be made available in a generic way as well. As an example one can think of general accessibility conditions for a hole. Ways to specify such requirements, and to solve the corresponding constraints, should be found.

Multiple-view feature modelling is becoming even more attractive when also product development phases are supported in which the geometry does not have to be fully specified, i.e. conceptual design, and in which not only single parts but also assemblies are dealt with, i.e. assembly design. Although an initial implementation of a system that supports this has been described, many extensions can be thought of. In particular, more views can be introduced and the conversion between views can be further automated.

Collaborative feature modelling is a very interesting option for collaborative modelling in general, because it offers all advantages of multiple-view feature modelling to the participants in a collaborative session. A web-based prototype system has been presented for collaborative modelling of parts. Currently, we are working on extensions to include collaborative conceptual and assembly modelling, by including views for these product development phases in the system.

Freeform feature modelling is a very promising and useful extension of regular-shaped feature modelling. However, many intricate problems have to be solved, in particular feature attachment, feature validity maintenance and feature recognition and conversion.

Altogether, we believe that the developments discussed in this paper contribute to making feature modelling more mature, and thus even more attractive as the way of product modelling in the future.

REFERENCES

[1] Bronsvoort WF, Bidarra R, Dohmen M, van Holland W and de Kraker KJ. Multiple-view feature modelling and conversion. In: Geometric Modeling: Theory and Practice - The State of the Art. Strasser W, Klein R and Rau R (eds). Berlin, Springer, 1997, 159-174.

[2] Shah JJ and Mäntylä M. Parametric and Feature-based CAD/CAM; Concepts, Techniques and Applications. New York, John Wiley & Sons, 1995.

[3] Bidarra R, de Kraker KJ and Bronsvoort WF. Representation and management of feature information in a cellular model. Computer-Aided Design, 1998, 30 (4): 301-313.

[4] Bronsvoort WF and Jansen FW. Feature modelling and conversion – Key concepts to concurrent engineering. Computers in Industry, 1993, 21 (1): 61-86.
[5] Hoffmann CM and Joan-Arinyo R. On user-defined features. Computer-Aided Design, 1998, 30 (5): 321-332.
[6] Parametric. Pro/ENGINEER, version 2000i. Parametric Technology Corporation, 2000, Waltham, MA.
[7] Bidarra R and Bronsvoort WF. Semantic feature modelling. Computer-Aided Design, 2000, 32 (3): 201-225.
[8] Gao S, Chen Z and Peng Q. Feature validity maintaining based on local feature recognition. In: CD-ROM Proceedings of the 2000 ASME Design Engineering Technical Conferences and Computers and Information in Engineering Conference, September 10-13, Baltimore, Maryland, USA, New York, ASME.
[9] Ulrich KT and Eppinger SD. Product Design and Development (Second Edition). Boston, Irwin/McGraw-Hill, 2000.
[10] Bullinger HJ and Warschat J. Concurrent simultaneous engineering systems; the way to successful product development. Berlin, Springer, 1995.
[11] De Martino T, Falcidieno B and Hassinger S. Design and engineering process integration through a multiple view intermediate modeller in a distributed object-oriented system environment. Computer-Aided Design, 1998, 30 (6): 437-452.
[12] Hoffman CM and Joan-Arinyo R. CAD and the product master model. Computer-Aided Design, 1998, 30 (11): 905-918.
[13] van Holland W and Bronsvoort WF. Assembly features in modeling and planning. Robotics and Computer Integrated Manufacturing, 2000, 16 (4): 277-294.
[14] Bronsvoort WF, Noort A, van den Berg J and Hoek GFM. Product development with multiple-view feature modelling. In: CD-ROM Proceedings of FEATS 2001 - International IFIP Conference on Feature Modeling and Advanced Design-For-The-Life-Cycle Systems, June 12-14, 2001, Valenciennes, France.
[15] Bidarra R, van den Berg E and Bronsvoort WF. Collaborative modeling with features. In: CD-ROM Proceedings of the 2001 ASME Design Engineering Technical Conferences, Pittsburgh, Pennsylvania, USA, New York, ASME.
[16] Piegl LA and Tiller W. The NURBS Book (Second Edition). Berlin, Springer, 1995.
[17] Au CK and Yuen MMF. A semantic feature language for sculptured object modelling. Computer-Aided Design, 2000, 32 (1): 63-74.
[18] van den Berg E, Bronsvoort WF and Vergeest JSM. Freeform feature modelling: concepts and prospects. Submitted for publication, 2001.