

AN INTELLIGENT HYBRID APPROACH FOR DESIGN-BY-FEATURES

Lian Ding and Yong Yue

Department of Computing and Information Systems
University of Luton

Park Square, Luton LU1 3JU, United Kingdom
Email: lian.ding@luton.ac.uk, yong.yue@luton.ac.uk
Website: www.luton.ac.uk

ABSTRACT

This paper presents a new methodology for design-by-features. After a brief background study of design-by-features, a new architecture with a feature library, feature-based model, feature library management and feature-based model management is introduced. A standard feature class is defined for the feature library. Automatic feature-based model management covering interactive functions, identification of feature interaction and maintenance of the model validity is described. Implementation of the work and testing with sample components are presented. Finally, conclusions are drawn and further research summarised.

Keywords: design-by-features, feature-based model, feature interaction, feature library, and validity constraints.

1. INTRODUCTION

Design-by-features, which is one of the two main methods in feature technology, offers the user a library of features to represent components. This method makes computer aided design (CAD) more meaningful and easier to integrate with computer aided process planning (CAPP). Although it does not eliminate the need for feature recognition, the amount of work for recognising features is reduced remarkably [Gindy *et al* 1998]. A major problem with design by features is feature interactions. Feature interactions which occur inevitably when a design model is manipulated, have an important effect on the features involved and consequently on the validity of the feature-based model. The model validity, in turn is very important for process planning applications. Although there has been a considerable amount of research effort, the problem has not been well resolved due to the complexity and lack of sophisticated algorithms.

This paper presents an intelligent hybrid methodology for design-by-features taking into account feature interactions and validity constraints. The next section provides a brief survey of previous

work on design-by-features. Section 3 proposes a new architecture of a design-by-features system. Section 4 describes feature library and feature-based model. Section 5 discusses the feature validity maintenance with the feature-based model management. Section 6 presents the implementation and test results. The last section summarises the work and draws conclusions.

2. PREVIOUS WORK

Design-by-features can be broadly classified into two categories [Lee and Kim 1998]: Destruction by machining features and synthesis by design features. With the destruction by machining features method, a design model is built by subtracting depression features from a raw stock and the machining features are derived simultaneously. The synthesis by design features method generates a design model by adding protrusion features and subtracting depression features.

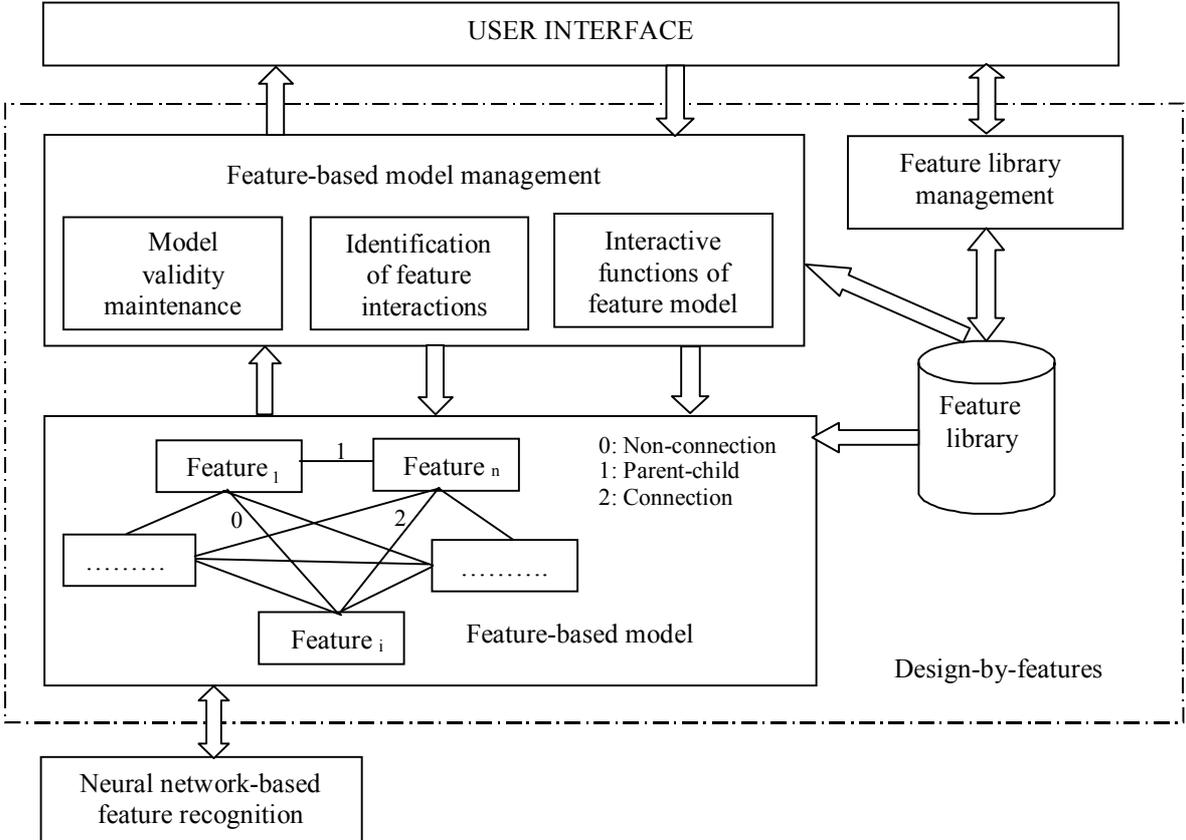
Various design-by-features systems have been implemented since the mid-1980s. Shah and Rogers [1988] developed an expert form feature modelling shell which supports user definition of

form features. Chang [1989] proposed a feature-based design and process planning system using a solid modeller. Chan and Nhieu [1993] proposed a framework for implementing a feature-based application with a CAD system, in which a user-defined external feature database was built, based on a hierarchical structure containing all feature information for the downstream application. De Martino *et al* [1994] introduced a method to recognise and update features after each feature-based design procedure by separating the interacting features. Lee and Kim [1998] employed an incremental approach for extracting machining features from a feature-based design. Tseng [1999] presented a modular modelling approach by strengthening the technical support provided to the designer. Hounsell and Case [1999] applied a method to identifying structured geometric spatial feature interactions based on a broad multilevel classification. Bidarra and Bronsvort [2000] proposed a semantic feature modelling approach to defining and maintaining the semantics of feature during all the modelling operations. However, the problems of conventional design-by-features method, such as feature interactions have not been fully resolved.

3. ARCHITECTURE OF THE DESIGN-BY-FEATURES SYSTEM

The proposed design-by-features system has been implemented in four parts: feature library, feature-based model, feature library management and feature model management. The feature library is composed of a set of standard feature classes used for designing the feature-based model. The feature-based model is used to design and store a component directly with features. The feature library management maintains the feature library functions, such as adding, modifying and deleting feature classes. The feature-based model management provides an interface between the user and the system applications, which consists of interactive functions to create and manipulate the feature-based model, specifying feature class, maintaining and recovering model validity, identifying feature interactions and so on.

The architecture of system is shown in Figure 1.



Architecture of the design-by-features system
Figure 1

4. FEATURE LIBRARY

In the design-by-features system, component models are created by specifying parameter values for the standard feature classes in the library. That is, the feature library is used to set up the context for the design and the system.

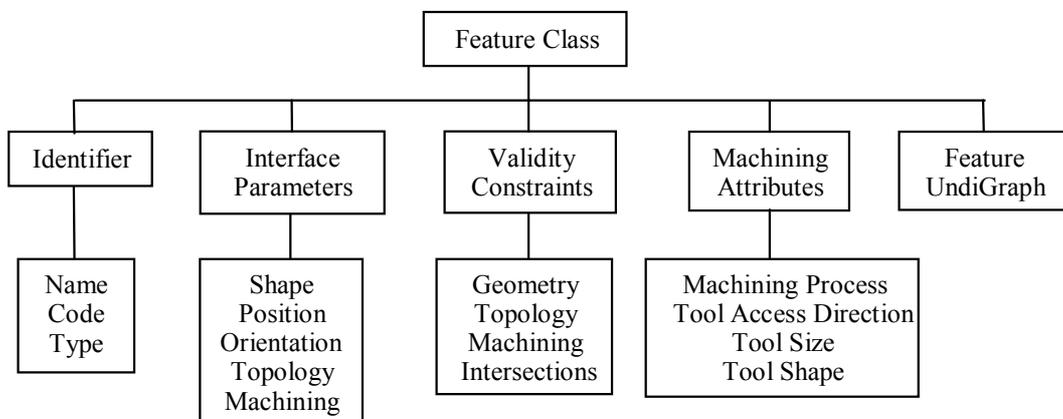
4.1 Feature Class Definition

Feature classes in the system are broadly divided into two categories: predefined features and user defined features.

4.1.1 Predefined features

Predefined features refer to a set of standard feature classes, which are defined as a template in the feature library. According to the requirements for integrated CAD/CAM environments, predefined design features cover the majority of the primitive features which likely to be of interest for the application of machining process planning. In the system, six primitive internal feature classes are included: round hole, conical hole, general hole, slot, pocket and step. However, it is possible to add more primitive features, especially external features in the future. The feature classification proposed has been based on the ISO AP224 STEP standards [STEP]. This provides an opportunity to bring the work towards industrial applications.

As illustrated in Figure 2, the standard feature class is explicitly described by a hierarchical data structure, which consists of five parts: Identifier, Interface Parameters, Validity Constraints, Machining Attributes and Feature UndiGraph. The Feature UndiGraph is proposed in order to depict feature patterns.



Data structure of feature class

Figure 2

Definition:

$$\text{Feature UndiGraph} = (F, R)$$

where F is a finite non-NULL set of faces the feature consists of:

$$F = \{\text{face}_i \mid \text{face}_i \in \text{Feature}\};$$

four attributes are attached to each face: the number of adjacent Feature Faces (FF), the number of adjacent Virtual Faces (VF), the shape type and the normal.

$R = \{FR\}$, is a set of relationships between the faces.

FR is a relationship with no specific direction between two faces:

$$FR = \{\langle \text{face}_i, \text{face}_j \rangle \mid P(\text{face}_i, \text{face}_j) \wedge (\text{face}_i, \text{face}_j \in F)\}$$

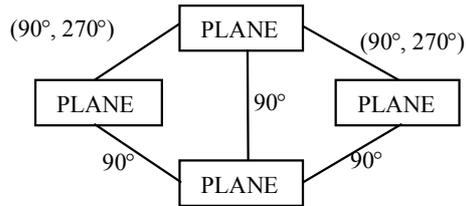
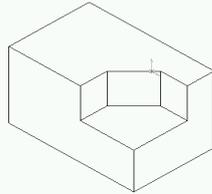
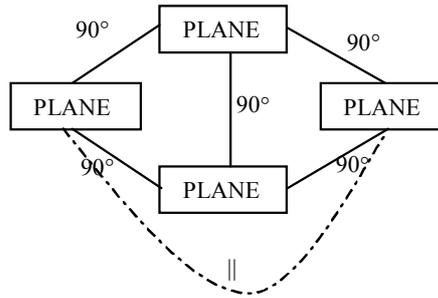
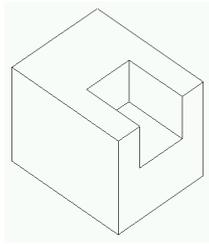
$P(\text{face}_i, \text{face}_j)$ is a path with no specific direction between face_i and face_j

Here, FR is symmetrical, i.e. $\langle \text{face}_x, \text{face}_y \rangle = \langle \text{face}_y, \text{face}_x \rangle$

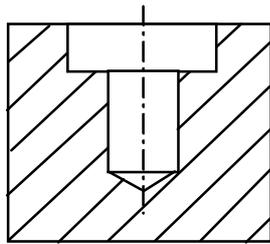
Figure 3 shows examples of Feature UndiGraph.

4.1.2 User-defined features

If predefined standard features are insufficient, the user can define his/her own features based on the standard design features stored in the library. An example of user-defined features is a compounded feature consisting of a blind hole and a through hole as shown in Figure 4.



Examples of Feature UndiGraph
Figure 3



A component feature
Figure 4

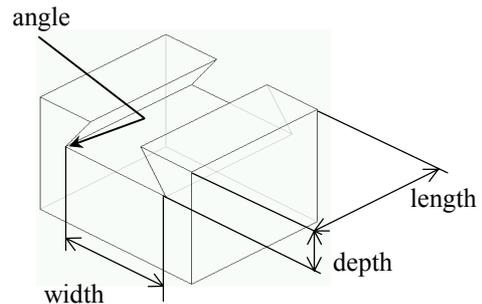
shape, called shape parameters. For example, the basic shape volume for a hole feature is a cylinder for which the main geometrical parameters include the radius and height. Another example is the slot shown in Figure 5, which is described as a block with parameters of length, width, depth and angle. The shape parameter set is sufficient to represent a feature and uses high level information (feature shape volume) instead of low level geometrical information (e.g. vertex and edge) designating the feature boundaries.

4.2 Specification of Feature Class

The instantiating of a feature class involves specifying the parameters of the feature (e.g. shape, position and orientation), detecting its validity, and interfacing to the feature-based model.

4.2.1 Parameters of feature class

Feature parameters serve as an interface for the user to specify the feature class. They are used for determining not only how a feature class is presented to the user but also how the user interacts with the system. A set of parameters of a feature class is defined based on the requirements for both design and machining purposes: shape, position and orientation, and validity constraints. The feature class in the system, which is either an additive feature or a subtractive feature, is represented by the feature volume or its boundary elements as a whole. The feature can also be either a primitive feature or a compounded feature. The basis of a feature class is its shape volume associated with a set of geometric parameters relating to the corresponding feature



A slot feature
Figure 5

The position and orientation parameters define the spatial relationships between the feature instance and the world co-ordinate system by fixing its degrees of freedom. Machining parameters such as access direction, tool geometry and tool size are related to the machining operations correspond to the feature.

4.2.2 Validity constraints

At the feature instantiation stage, the feature validity is checked automatically based on its validity constraints. As illustrate in Figure 2, four types of constraints are involved in defining the feature class: geometric, topological, machining and interacting constraints.

- Geometric constraints

Geometric constraints are indispensable for each feature, which have a standard range for specifying the value of each parameter for the shape, position and orientation.

- Topological constraints

The faces of the parameterised shape volume for the feature can be classified into two types: VF (Virtual Face) and FF (Feature Face). A VF is not the boundary of feature, but the boundary of the feature volume. An FF is the boundary of both the feature volume and the feature itself. A special topological relationship of FF and VF exists for each feature class. Constraints of such property are termed topological validity constraints. For instance, the depth of a blind hole must be restricted to be less than the size of the stock where the hole is to be added. Otherwise the blind hole would become a through-hole. The length limit of a blind or through-slot is another example. In general, the restricted value can be calculated by algebraic expressions with the parameters of the feature class. Table 1 presents an example of length limit of a blind slot when angle α is restricted between 0° and 90° .

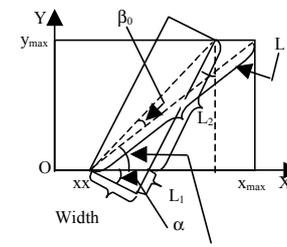
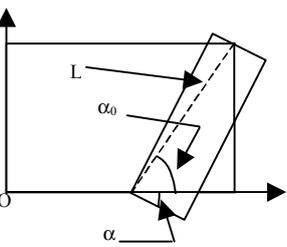
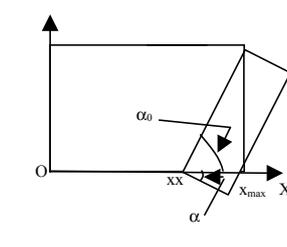
- Machining constraints

From the machining point of view, dimension and tolerance constraints should be applied to the feature definition. The values of machining constraints depend on the specific workshop environment (e.g. machine tools). For instance, the radius and height of a through-hole may be restricted to a range based on the manufacturing environment.

- Interacting constraints

Geometrical, topological and machining constraints, as described above, are insufficient to fully retain feature validity when feature interactions exist. As known, feature interactions can cause serious constraint violations of valid feature instances. Therefore, the constraints for feature interactions must be defined, such as the dependent properties between parent and child features.

Table 1. Length limit of a slot

Primitive variables	$L = \sqrt{(x_{\max} - xx)^2 + (y_{\max} - oy)^2}$ $\beta_0 = \arcsin(\text{width} / L)$ $\alpha_0 = \text{arccotg}\left(\frac{ x_{\max} - xx }{ y_{\max} - oy }\right)$
Case 1	 <p> $L_{\text{limit}} = L_1 + L_2,$ $L_1 = \text{width} * \text{tg}\alpha$ $L_2 = (y_{\max} - oy) / \text{cos}\alpha$ </p>
Case 2	 <p> $(90^\circ - \alpha_0) > \alpha \geq (90^\circ - \alpha_0 - \beta_0)$ $L_{\text{limit}} = L * \text{cos}(90^\circ - \alpha - \alpha_0)$ </p>
Case 3	 <p> $(90^\circ - \alpha_0) > \alpha \geq (90^\circ - \alpha_0 - \beta_0)$ $L_{\text{limit}} = L * \text{cos}(90^\circ - \alpha - \alpha_0)$ </p>

5. FEATURE-BASED MODEL MANAGEMENT

Feature-based modelling allows the user to design a component directly with features predefined in the feature library. Unlike conventional geometrical models, the proposed feature-based model is

designed for high-level data, i.e. feature instances. Therefore all the modelling operations are feature-based. As shown in Figure 1, the highest level of the feature-based model can be presented as a graph, where the nodes correspond to feature instances and the arcs store interaction relationships between feature pairs. Three types of feature relationships are defined within the feature-based model: parent-child relationship, connection relationship and non-connection relationship [Ding *et al* 2000]. The proposed feature-based model makes it possible to trace the modelling process using a top-down approach based on the hierarchical structure. Another characteristic of the model is the consistence of the feature instance. The validity of the feature instance is kept effectively by the feature-based model management. The functions of feature-based model management can be generally grouped into three categories: interactive functions for creating and manipulating the feature model, identification functions for detecting feature interactions and maintenance functions for keeping model validity.

5.1 Interactive Functions of Feature-Based Model

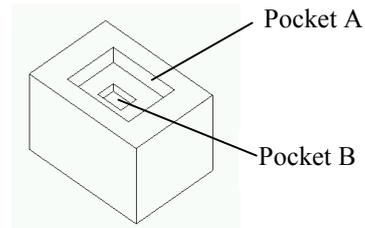
Three main operations are considered for the interactive functions: adding new feature instance, editing and deleting existing feature instance.

5.1.1 Adding a new feature instance to the model

With a full set of interface parameter values, a pre-defined feature class can be initialised as a new feature instance. Validity check for internal geometric and topological constraints for the user-supplied parameters will be operated at the same time. When the validity check process is successfully passed, the new feature instance is added to the feature-based model.

5.1.2 Editing a feature instance in the model

Existing feature instances can be modified by specifying new values for their parameters. The process is called editing features. The validity check and interaction identification are necessary for the feature being modified. If the feature to be modified is a parent feature, the validity of all its child features must be checked and necessary changes to their feature class or dimensions made. In the example shown in Figure 6, pocket A is the parent of pocket B. When the depth of pocket A is shortened, the depth of pocket B must be changed accordingly, i.e. increased by the amount by which pocket A is shortened. After all necessary operations, the data structure of the feature-based model is updated.



Parent-child relationship between two pockets
Figure 6

5.1.3 Deleting a feature instance from the model

When a feature instance is deleted from the feature-based model, all its parameters and interaction relationships with other features will be deleted completely. However, this is not a straightforward operation if the feature to be deleted has special interaction relationship, e.g. parent-child relationship. For the situation shown in Figure 6, if pocket A is to be deleted, pocket B will become an invalid pocket or its feature class and dimension must be changed accordingly.

5.2 Identification of Feature Interactions and Maintenance of Model Validity

Although a feature instance inherits all the validity constraints defined in the feature class, it must be checked / modified to maintain the feature validity when feature interactions are present. Feature interactions are still regarded as a major problem affecting feature-based models.

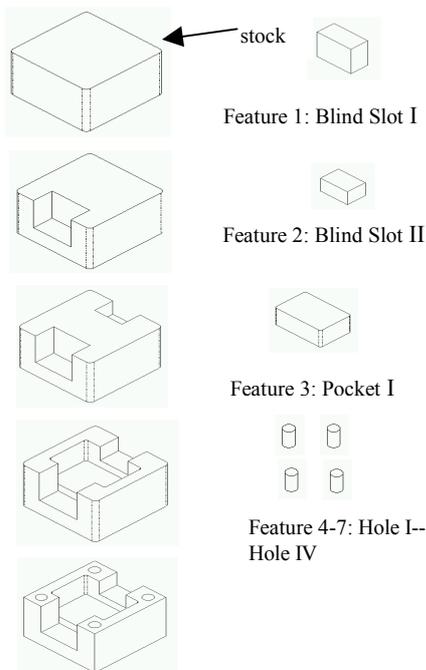
The approach proposed in this work identifies the feature relationships which are defined in an earlier section. The key mechanism of the proposed algorithm is to make use of the Interacting Entity (IE) between feature pairs. The process applies a search algorithm to traversing all the features in the hierarchical feature-based model. A Boolean intersection operation is performed between the features and an IE is produced as the result. The IE is analysed to determine the relationship between feature pairs, which can be parent-child relationship, connection relationship or non-connection relationship. With the algorithm, all relevant interaction situations between feature pairs can be detected, reported and handled in an appropriate way. An advantage of the interaction identification is that it focuses on analysing the interacting entity of feature pairs instead of the union volume between the interacting features. Therefore, it is efficient to tackle situations where an operation with feature interactions causes some constraint violation of the model validity.

6. IMPLEMENTATION AND TESTING

The work has been developed using ACIS Release 6.2, SolidWorks 2000 and Matlab 6.0 on a PIII 500 personal computer. The programs are written in C++ calling to the modellers' Application Programming Interface (API) functions.

SolidWorks APIs are called to create the feature-based model by specifying parameterised features defined in the feature library in a three-dimensional screen. The ACIS modeller is used to implement the Boolean operation-based algorithm for searching and manipulating features, especially checking interacting features. The interacting features are dealt with specifically by analysing the IE between the feature pair. For interacting features, there exist five conditions: merge, class change, divide, dimension change and connect. A feed-forward neural network available in the Matlab neural network toolbox is utilised to recognise features with any changes. The output of the system is a feature-based model, containing the information of all machining features defining the component, such as feature type, relationship between feature pairs, etc. This model can be used to analyse the component design.

The system developed in this work has been tested with a number of components. Two examples and their test results are shown in Figures 7, 8 and 9, and Figures 10, 11 and 12, respectively.



Example 1
Figure 7

Feature 1 \cap Feature 3  face: PF-PF

The relationship between Feature 1 and Feature 3 is Connection.

Feature 2 \cap Feature 3  face: CF-PF

The class of Feature 2 is changed from Blind Slot to Through Slot. There is a Parent-child relationship between Feature 2 (Child feature) and Feature 3 (Parent feature).

Note: Feature Face (FF) is presented as a special face that actually constitutes the basic shape of a feature.

PF-PF: the intersecting face is regarded as Partial FF (PFF) for pairs of features.

CF-PF: The intersecting face appears as a whole FF (CFF) of feature f_1 while a PFF of feature f_2 .

Analysis of interacting features of Example 1
Figure 8

Feature 1: Blind Slot

Feature 2: Non-connection	Feature 3: Connection
Feature 4: Non-connection	Feature 5: Non-connection
Feature 6: Non-connection	Feature 7: Non-connection

Feature 2: Through Slot

Feature 1: Non-connection	Feature 3: Child
Feature 4: Non-connection	Feature 5: Non-connection
Feature 6: Non-connection	Feature 7: Non-connection

Feature 3: Pocket

Feature 1: Connection	Feature 2: Parent
Feature 4: Non-connection	Feature 5: Non-connection
Feature 6: Non-connection	Feature 7: Non-connection

Feature 4: Hole

Feature 1: Non-connection	Feature 2: Non-connection
Feature 3: Non-connection	Feature 5: Non-connection
Feature 6: Non-connection	Feature 7: Non-connection

Feature 5: Hole

Feature 1: Non-connection	Feature 2: Non-connection
Feature 3: Non-connection	Feature 4: Non-connection
Feature 6: Non-connection	Feature 7: Non-connection

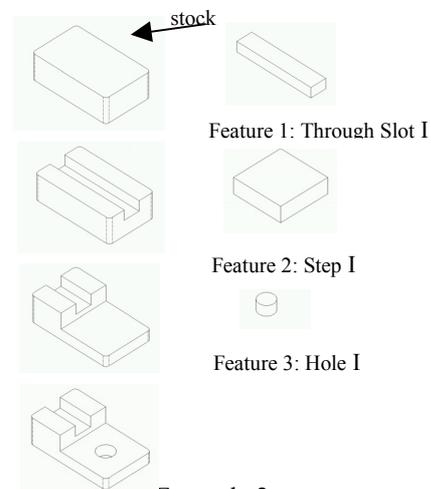
Feature 6: Hole

Feature 1: Non-connection	Feature 2: Non-connection
Feature 3: Non-connection	Feature 4: Non-connection
Feature 6: Non-connection	Feature 7: Non-connection

Feature 7: Hole

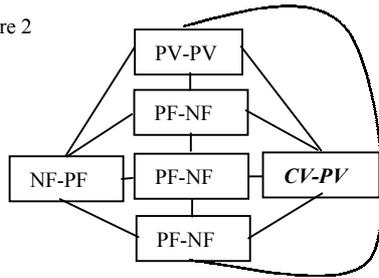
Feature 1: Non-connection	Feature 2: Non-connection
Feature 3: Non-connection	Feature 4: Non-connection
Feature 5: Non-connection	Feature 6: Non-connection

Result of Example 1
Figure 9



Example 2
Figure 10

Feature 1 \cap Feature 2



The dimension of Feature 1 is changed. The relationship between Feature 1 (Child feature) and Feature 2 (Parent feature) is Parent-child.

Feature 2 \cap Feature 3 \odot face: PF-CV

The relationship between Feature 2 (Parent) and Feature 3 (Child) is Parent-child.

Note:

Virtual Face (VF) means a special face that does not actually constitute the feature but represents as a virtual face for the bounding boundary of Spatial Virtual Entity.

CV-PV: The intersecting face represents a whole VF of feature ft_1 while a PFF of feature ft_2 .

Analysis interacting features of Example 2
Figure 11

Feature 1: Through Slot

Feature 2: Child Feature 3: Non-connection

Feature 2: Step

Feature 1: Parent Feature 3: Parent

Feature 3: Hole

Feature 1: Non-connection Feature 2: Child

Result of Example 2
Figure 12

7. CONCLUSIONS

This paper has presented an intelligent hybrid design-by-features methodology considering feature interactions. The method for resolving feature interactions for design-by-features has been successful with a range of prismatic components.

Further research will be to enhance the capability and integrate the work with CAPP applications.

REFERENCES

Bidarra R, Bronsvort WF, Semantic feature modelling, *Computer-Aided Design*, Vol 32, No 2, pp 201-225, 2000

Chan KC and Nhieu J, A Framework for feature-based applications, *Computers and Industrial Engineering*, Vol 24, No 2, pp 151-164, 1993

Chang TC, *Expert Process Planning for Manufacturing*, Addison-Wesley, New York, 1989

De Martino T, Falcidieno B, Giannini F, Hassinger S and Ovtcharova J, Feature-based modelling by integrating design and recognition approaches, *Computer-aided Design*, Vol 26, No 8, pp 646-653, 1994

Ding L, Yue Y and Ahemt K, An integrated approach to integrating CAD/CAM, *Proceedings of 6th Annual Conference of the Chinese Automation and computer Society in the UK*, Loughborough, 23-24 September 2000

Gindy NNZ, Yue Y and Zhu CF, Automated feature validation for creating / editing feature-based component data models, *International Journal of Production Research*, Vol 36, No 9, pp 2479-2495, 1998

Hounsell MS and Case K, Feature-based interaction: an identification and classification methodology, *Proceedings of the Institution of Mechanical Engineers, Part B*, Vol 213, pp 369-380, 1999

Lee JY and Kim K, A feature-based approach to extracting machining features, *Computer Aided Design*, Vol 30, No 13, pp 1019-1035, 1998

Shah JJ and Rogers MT, Expert form feature modelling shell, *Computer Aided Design*, Vol 20, pp 515-524, 1988

STEP, STandard for External representation of Product data, ISO 10303-224 (Industrial automation systems and integration. Product data representation and exchange: mechanical product definition for process plans using machining features)

Tseng YJ, A modular modelling approach by integrating feature recognition and feature-based design, *Computers in Industry*, Vol 39, pp 113-125, 1999