Associativity Between Feature Models Across Domains

S. Subramani and B. Gurumoorthy

Department of Mechanical Engineering
Indian Institute of Science
Bangalore 560 012, INDIA

ssmani, bgm@mecheng.iisc.ernet.in

ABSTRACT

Associativity between feature models implies the automatic updating of different feature models of a part after changes are made in one of its feature models. This is an important requirement in a distributed and concurrent design environment, where integrity of part geometry has to be maintained through changes made in different task domains.

The proposed algorithm takes multiple feature models of a part as input and modifies other feature models to reflect the changes made to a feature in a feature model. The proposed algorithm updates feature volumes in a model that has not been edited and then classifies the updated volumes to obtain the updated feature model. The spatial arrangement of feature faces and adjacency relationship between features are used to isolate features in a view that are affected by the modification. Feature volumes are updated based on the classification of the feature volume of the modified feature with respect to feature volumes of the model being updated. The algorithm is capable of handling all types of feature modifications namely, feature deletion, feature creation, and changes to feature location and parameters. In contrast to current art in automatic updating of feature models, the proposed algorithm does not use an intermediate representation, does not re-interpret the feature model from a low level representation and handles interacting features. Results of implementation on typical cases are presented.

Categories and Subject Descriptors: J.6 [COMPUTER-AIDED ENGINEERING]: Computer-aided design (CAD)

General Terms: Algorithms, Design.

Keywords: Feature Editing, Feature Based Modeling, 3D Clipping, Concurrent Engineering.

1. INTRODUCTION

In a distributed and concurrent design environment, it is important to provide different *views* of a part to different task domains [1]. This requires multiple feature models of a part corresponding to different domains. For example the part model shown in figure 1 is viewed by the designer as a

block with one rib and by a process planner as a block with a corner slot. Associativity between feature models implies the automatic updating of different feature models of a part after changes are made in one of its feature models. This is an important requirement in a distributed and concurrent design environment, where integrity of part shape has to be maintained through changes made in different task domains. A distributed design environment could allow editing of the shape in one view only even if the change is triggered by considerations of another task domain. This is referred to as *one-way* editing [2, 3]. Alternately, shape can be changed in any view in environments that allow *multi-way* editing [2, 3]. In multi-way editing environments, updating shape information in all views can take two forms. In the first, feature models corresponding to each view are updated. In the second, a low level shape representation such as B-rep or cellular model [3] is updated. The problem of updating feature models is the focus of this paper. For example, if the width of the rib in figure 1 is changed by the designer, this change should be reflected in machining view as a change in the width of the slot. This mismatch between features in the two views has been cited as a reason for not attempting automatic update of feature models across views [2, 3]. Since editing of shape in each view is more likely to be feature based, updating feature models across views is a more direct approach than evaluating the feature model in the view being edited, transmitting the updated low level representation to other views and then re-extracting the updated feature model.

1.1 Related work

There have been several efforts to develop a feature based distributed design environment [4, 5, 6, 2, 7]. Among these, Han and Requicha [4], and De Martino *et al.* [5] support modification in only one view (the design view). Changes have to be made in the design view and the other application views are then obtained by feature conversion/extraction. Feature conversion for updating other view can be done incrementally (after each change) [8] or once after all changes have been made [4, 5]. This one-way architecture forces the user to interpret the changes desired in an application view in terms of the features available in the design view. In all these efforts the view designated for making changes is always the design view as the feature conversion algorithms used in these efforts can only work one way - from design (positive features) to machining (negative features). de Kraker [6], Hoffmann and Joan Arinyo [2] and Jha and Gurumoorthy [7] describe distributed systems using multi-way architectures. de Kraker *et al.* [6, 1] use an intermediate representation in between the feature models.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SM'03, June 16–20, 2003, Seattle, Washington, USA.

Copyright 2003 ACM 1-58113-706-0/03/0006 ...\$5.00.

Figure 1: Multiple Views

Feature model created in one view is converted to the intermediate representation and then other feature interpretations are extracted from the intermediate representation. They use a cell based model as the intermediate representation. Features are defined in terms of constraints between the feature's entities and the extraction process is based on identifying entities in the cell model satisfying these constraints. This procedure requires a strategy (that specifies the feature model structure for that view) to be prescribed for each view. For each feature class specified by the strategy, a matching procedure is used to identify an instance of that feature. The matching procedure matches the generic shape faces (defined for the feature class) with faces in Cellular Model faces. The extraction is therefore governed by the strategy for each view that prescribes feature classes to be searched for in a certain order. Multiple strategies have to be defined therefore, to obtain feature models where only the sequence of features is different.

Hoffmann and Joan-Arinyo [2] propose algorithms to edit *net shape* and rebuild feature models in different client views to reflect this change. Their updating scheme assumes that the shape changes do not alter the number of features or their interrelationships in other views. They also describe an algorithm for updating feature views if a feature is added or deleted in one view. However this algorithm assumes that the feature being added or deleted does not have any dependencies. In situations where stock/enclosing volume changes or if there are mismatches in the features in the two views, they indicate that feature extraction is required.

Jha and Gurumoorthy[7] have proposed an algorithm to propagate feature modification within and across domains automatically. Their work too is restricted to changes in one view that do not alter the number of features or interactions between them.

There are three types of modifications that have been identified in a distributed feature based environment [2]. These are: changes in dimensions and constraints, adding/deleting features and changes in relationship between net shape elements in one view due to changes in another view.

The first type of modification will also result in changes in a feature in the edit view. In this paper we describe a feature updating procedure that handles the first two types of modifications. The description assumes that there are two views and that modifications are made and updated incrementally(one at a time). The views used are generic with respect to application and are named as positive, negative and mixed based on the type of features used in that view. Note that the same algorithm will work for any pair of views from the above three types. The view in which editing/modification takes place is termed as the edit view and

the feature in that view that is created/deleted or modified is termed as the edit-feature. The view which has to be updated is referred to as the target view. It has been argued that updating feature models in views other than edit view will require evaluation and re-extracting [2, 3]. The novelty of our approach is that feature models are updated directly from the modified edit-view without the need for the modified B-rep of the part model. The input to the algorithm are feature models in each view. It is assumed that initially, the feature views are consistent, that is the evaluation of the feature model in each view will result in the same B-rep. Feature volume of the edit-feature is classified with respect to the feature volumes constituting the target view. This classification is used to update the feature volumes in the target view. The updated volumes are then re-classified as features that form the updated feature model of the target view. Classifying volumes into features is much simpler than extracting features from a low level representation such as the B-rep. The B-rep which constitutes the low level shape representation is updated in only the edit view.

The rest of the paper is organised as follows. The next section presents definitions of terms used in the algorithm. An overview of the algorithm to update feature models is described next followed by a detailed description of the steps in the algorithm. Results from an implementation of the algorithm on typical cases are presented next. The paper concludes with a discussion of these results.

2. DEFINITIONS

Definitions of the terms used in the description of the updating algorithm are presented in this section.

Feature definition and Face classifications

Form feature is defined as a set of faces with distinct topological and geometrical characteristics[9] and are created by the addition or subtraction of a sweep solid from an arbitrary solid. The sweep can be along a line or a curve as long as there are no self intersections. The sweep solid is referred to as the *feature volume*. Faces in the feature-solid (referred to as *feature faces*) are classified as *shell faces* (SF) (faces that constitute the shells of the swept solid) and *end faces* (EF), which close the ends of the shell. During feature

Figure 2: Face classification of a feature

attachment, some feature faces will overlap with faces of the existing feature set. These faces are called *shared faces* and the remaining feature faces are called *created faces*. Shared faces are further classified into shared shell faces (SSF) and shared end faces (SEF), based on their classification as shell and end faces respectively, of the feature-solid. Similarly created faces are further classified into created shell faces

(CSF) and created end faces (CEF). Figure 2 shows the feature volume corresponding to the feature created in a part model. The sweep solid and its construction are also shown. The different types of feature faces are labelled in the figure.

The type of feature represented by the feature volume is decided by the number and arrangement of the four types of feature faces [9]. In this paper, the term feature is used to refer to the feature volume with the feature type identified.

Feature Relationship Graph

Feature relationship graph (FRG) is a graph whose nodes are features. An edge between nodes denotes adjacency / dependency relation between two nodes/features. The attribute associated with the edge indicates the nature of dependency, namely parent or child.

The dependency between features are implicitly available in the spatial arrangement of the feature faces with respect to each other. A feature x is said to be dependent on feature y , if and only if atleast one face of x is completely *overlapped* by any face belonging to y . In the dependency relationship defined above, feature x and y are referred to as child and parent respectively.

Nascent Feature

A feature is said to be nascent, if it has no child feature. The leaf nodes in the FRG are therefore nascent features.

Bounding Face of a solid/features set

A face is said to be bounding face of a solid/features set, if the entire solid/features set lies in only one half space corresponding to the face.

Enclosing Volume of a feature set

The enclosing volume of a feature set is a convex solid (not necessarily a box/cuboid), that contains all features in the set. The enclosing volume is obtained by clipping a universal solid by the bounding faces of the feature set.

3. OVERVIEW OF FEATURE UPDATING

There are three types of modifications possible in the edit-view. An existing feature may be deleted or modified and a new feature may be created. The updating process considers the modification of the feature model in the edit view as taking place in two steps - removing an existing feature (*old-edit-feature*) and replacing it with a new feature (*new-edit-feature*). Clearly the first step is not required when a new feature is created and the second step is not required when an existing feature is deleted. The proposed algorithm for updating feature models takes as input the old-edit-feature, new-edit-feature and the target view. Similar to the modification process, updating of the target feature model also involves two steps. In the first the removal of the old-edit-feature in the edit-view is updated in the target-view and in the second the addition of new-edit-feature in the edit-view is updated in the target-view.

During updating of the target feature model, interactions between feature volumes in the target view (target-feature) and the edit-feature are determined. If there is no interaction, the edit-feature is added to the target view list as it is or with face orientations flipped depending on whether the edit feature has been removed or added in the edit view. There are two types of interactions of interest. The target feature is completely inside the edit feature or it is partially

outside the edit feature. These two types of interactions are referred to as ABSORPTION and INTRUSION respectively following the classifications proposed by Bidarra *et al.* [10]. In the case of ABSORPTION type of interaction, the target feature is removed from the feature list. If the interaction is of type INTRUSION, the volume complementary to the edit-feature in the target-feature is determined. This volume is obtained as a set of volumes that are then added to the list of features in the target view.

Finally, contiguous volumes in the list are merged and the volumes in the updated set are classified to identify the features.

4. ALGORITHM DETAILS

The procedure UpdateFeature takes as input, target feature view, new edit-feature and old feature as input. If the change in the edit view was deletion, new feature will be empty and if the change was feature addition, old feature would be empty. Features in a view are represented by the faces forming the closed volume corresponding to the feature (these faces are referred to henceforth as *feature faces*). The face normals are assumed to be oriented away from the material enclosed in the part. The representation of feature also contains the type of the feature (positive or negative).

4.1 Construction of FCT

This task takes a list of faces as input. The faces in the input list are classified with respect to each other based on their relative configuration. Nine types of relative arrangements between faces have been identified and these are shown in the figure 3. The results of the classification are stored in the form of a matrix of size $n_f \times n_f$, where n_f is the total number of faces in the feature set. The element in the i^{th} row and the j^{th} column is the index (in figure 3) of the type of arrangement between the faces f_i and f_j . This matrix is referred to as the Face Classification Table (FCT).

The classification of a pair of faces is computed based on the classification of vertices in one face with respect to the other face. When a face is non-planar, its edges are discretised and the points so obtained are used for classification.

Figure 3: Types of classification of face F_2 with respect to face F_1

4.2 Construction of FRG

The FRG is constructed as a matrix whose ij^{th} element is either '1' or '0' depending on whether the i^{th} feature is the child/parent of the j^{th} feature. A feature is classified to be a child of another feature if one of its faces has been

classified to be of type overlap (figure 3) with respect to a face in the other feature. This classification is available in the FCT.

4.3 Procedure FeatureUpdate

This is the main algorithm for updating feature models. There are two tasks. The first task is to update the target view to account for the removal of the old feature in the edit view. The second task is to update the target view to account for the addition of the new feature in the edit view. Both the tasks are required only when a feature has been modified in the edit view.

The updating of the target view in both cases takes place as follows. Interaction between the feature in the edit view and the features in the target view are first identified. This is done by classifying the target-feature volume with respect to the edit-feature volume. If the interaction between the two is of type ABSORPTION (no part of target-feature is outside the edit-feature) the target-feature is removed from the feature list. If the interaction is of type INTRUSION, then the interacting target-feature is sent to the Procedure ClippingProcess to obtain new volumes which are then classified and entered in the target view feature list. If there are no interacting features, the feature in the edit view is inserted in the target view. The only deviations are that when adding the feature from the edit view in the target view, the feature type is flipped when handling the old feature. Moreover, in the check for interactions with the target view features, the old feature is only checked against features of the same type while the new feature is checked against features of the opposite type.

When the new feature is a positive feature then it is checked if the enclosing volume/stock has changed. If the enclosing volume/stock has increased then the extra stock is processed (by Procedure GrowStock) only if the target view consists of all negative features. This is because if the target view consists of both types of features, the new feature can get added as a positive feature. Procedure GrowStock replaces the enclosing volume of the target view by the new enclosing volume of the features in the target view and the new feature. The new enclosing volume is classified with respect to the old enclosing volume. The portion of the new enclosing volume that is outside the old enclosing volume is added to the list of features in the target view as a negative feature.

The final steps in the procedure merge features where possible. The FCT and FRG for the updated list of features are constructed. The FCT is used to identify pairs of features that each have a face with classification type *overlap* with respect each other. Such features are merged by removing these features from the feature list and combining (boolean union) the feature volumes. The combined volume is then classified to obtain the feature. These features are inserted into feature view.

4.4 Procedure ClippingProcess

Procedure ClippingProcess is used to resolve interactions between feature volumes. It takes two interacting features and decomposes the first feature to return a set of volumes that form the complement of the second feature. Decomposition is achieved by repeated application of Procedure Clip to the volume of the first feature. Procedure Clip splits a volume about a face and returns the portion of the vol-

Procedure 1 FeatureUpdate

Require: Target View, New Edit-Feature(NEF), Old Edit-Feature(OEF)

```

1: if OEF  $\neq$  NULL then
2:   Find features in target view (that are same type as OEF) that interact with OEF
3:   if there are interacting features then
4:     for all each feature (target-Feature) in the target view that interacts with OEF do
5:       if Type of interaction is ABSORPTION then
6:         Remove target-feature from the feature list of target view
7:       else
8:         New-Feature-Volumes=ClippingProcess(Target-Feature, OEF)
9:         Insert New-Feature-Volumes in Target View Feature List
10:      end if
11:    end for
12:   else
13:     Obtain New-Feature-Volume by flipping feature type of OEF
14:     Insert New-Feature-Volumes in Target View Feature List
15:   end if
16: end if
17: if NEF  $\neq$  NULL then
18:   if NEF is positive and Target View is Negative AND NEF is not completely enclosed in the enclosing volume of features in target view then
19:     GrowStock(NEF, Target View)
20:   end if
21:   Find features in target view (that are of opposite type as NEF) that interact with NEF
22:   if there are interacting features then
23:     for all each feature (Target-Feature) in the target view that interacts with NEF do
24:       if Type of interaction is ABSORPTION then
25:         Remove target-feature from the feature list of target view
26:       else
27:         New-Feature-Volumes=ClippingProcess(Target-Feature, NEF)
28:         Insert New-Feature-Volumes in Target View Feature List
29:       end if
30:     end for
31:   else
32:     Insert NEF in Target View Feature List
33:   end if
34: end if
35: Update FRG and FCT of Target View Feature List
36: Merge contiguous volumes in Target View Feature List and update list
37: Classify each volume in Target View Feature List to obtain corresponding New-Feature
38: Add New-Feature to Target View Feature List

```

Procedure 2 ClippingProcess

Require: ftr1, ftr2

```
1: Construct faceList consisting of faces in ftr1 and ftr2
2: Construct FCT of faceList
3: for all face in ftr2 that is not a bounding face of faceList
  do
4:   if ftr2 is NEGATIVE type then
5:     Flip orientation of face normal
6:   end if
7:   volume = clip(ftr1,face)
8:   Insert volume in Volume-List
9: end for
10: return Volume-List
```

ume that lies along the normal of the face. The faces of the second feature that are not bounding faces of the set of faces in both features are used for clipping. These faces are identified from the FCT of the faces in both the features.

4.5 Classification of feature volume

The feature type corresponding to the feature volume obtained by the ClippingProcess is identified in this step. This involves reasoning on the spatial arrangement of the feature faces and their interactions. The face classifications illustrated in figure 2 are used to classify the feature volume as a feature [9].

5. RESULTS AND DISCUSSIONS

The proposed algorithm to update feature modifications has been implemented in C++/Unix environment with the help of Shapes geometric kernel [11]. The kernel is used to perform the geometric computations such as volume classification and splitting of volumes. The algorithm has been tested with typical parts including some parts from the design repository [12].

Figure 4: Example Part 1 - Updating after creation of fillet in one view

Figure 4 shows the creation of a fillet feature on example part 1. The created feature in this instance has curved faces. In the clipping process, the curved feature is hidden using its bounding box and the bounding box used to clip the interacting feature. Finally the bounding box is clipped with respect to obtain the correct feature volume (figure 4(iv)).

Figure 5 shows the creation of feature which results in a larger stock/enclosing volume in the machining (only negative) domain. The updated machining view with the enlarged stock is shown in the figure 5(iv). Figures 6 and 7

Figure 5: Example Part 2 - Feature creation resulting in change of stock

Figure 6: Example Part 3 - Feature modification

Figure 7: Example Part 4 -Modification of a feature with dependencies

show examples of updating after feature modification in a view. Figure 8 shows the updating of feature model when the feature modification interacts with other features in the edit view.

Figure 8: Example Part 2 - Feature Creation resulting in Interacting Features

5.1 Discussion

The correctness of the updating algorithm is argued as follows. The input feature models are consistent (they correspond to the same B-rep) to begin with. Once a feature is edited in a view, the feature models are updated and it remains to be shown that the updated feature models correspond to the updated B-rep. The features in the target view that do not interact with the edited feature (feature that is created/deleted/modified) are consistent with the modified B-rep. There are only two possibilities of interaction between the edited feature and a feature in the target view. In the ABSORPTION type interaction, the absorbed volumes of target features are removed in the updated model giving the correct result. In the INTRUSION type of interaction between the edited feature and the target feature, the feature model is updated by removing the target feature and adding features obtained by clipping the target feature by the faces of the edited feature. This will result in a set of feature volumes of the same type (positive or negative) that when combined yield the volume of the original feature minus the portion of the edited feature inside this volume. Again the updated feature model is consistent with the modified B-rep. In situations where feature addition results in a change of enclosing volume, the additional volume is treated as another feature volume in the target domain to get the correct result.

As can be seen from the results, the algorithm can handle changes that result in change in stock/base solid for negative features, interacting features and even features such as fillets and blends. Unlike the approach of Hoffmann and Arinyo [2] the features being deleted/modified can have dependencies (see figure 7). It must be mentioned here that the interactions between the modified feature and the other features in the edit view are resolved interactively by the user. The FRG in each view can be used to alert the user of all dependencies that are likely to be affected by the change and the procedure used to obtain the updated feature model in another view can also be used to update the interacting features in the edit view. However since the final arbiter is the user in that domain the interactions are only flagged. For instance, in figure 7, the user may decide to keep the

length of the smaller rib the same and translate it to retain its attachment to the larger rib whose thickness has been reduced as opposed to extending its length. Alternately, this can be achieved by a constraint manager as is done by Hoffmann and Arinyo [2] and de Kraker *et al.* [6].

The algorithms presented here will work for feature views with volumetric features. The feature definitions and the algorithms support arbitrary swept features. However the implementation has focussed only on extrusion type features. Updating of models with non-volumetric (surface) features needs to be explored.

6. CONCLUSIONS

Algorithms to update feature models directly from a modified feature model has been described. Three types of feature views - only positive features, only negative features and mixed features are supported. Modifications can be made in any view and the remaining views are automatically updated. The updating algorithm uses clipping of a volume by a face to identify volumes corresponding to the updated model. The algorithm does not use any intermediate representation and updates the feature volumes. The algorithm handles various kinds of feature modifications like feature deletion, feature creation, transformation and parameter changes. Changes to features with dependencies and enclosing volume are also handled.

7. REFERENCES

- [1] de Kraker KJ, Dohmen M, and Bronsvort WF. Multiple-ways feature conversion to support concurrent engineering. *Proceedings Solid Modeling '95, Third Symposium on Solid Modeling and Applications*, pages 105–114, 1995.
- [2] Hoffmann CM and Joan-Arinyo R. Distributed maintenance of multiple product views. *Computer Aided Design*, 32(7):421–431, 2000.
- [3] Wu D and Sarma R. Dynamic segmentation and incremental editing of boundary representations in a collaborative design environment. *Journal of Computing and Information Science in Engineering*, 1:1–10, 2001.
- [4] Han J and Requicha AAG. Modeler-independent feature recognition in a distributed environment. *Computer Aided Design*, 30(6):453–463, 1998.
- [5] De Martino T, Falcidieno B, and Hazinger S. Design and engineering process through a multiple view intermediate modeller in a distributed object-oriented system environment. *Computer Aided Design*, 30(6):437–452, 1998.
- [6] de Kraker KJ, Dohmen M, and Bronsvort WF. Maintaining multiple views in feature modeling. *Proceedings Solid Modeling '97, Fourth Symposium on Solid Modeling and Applications*, pages 123–130, 14-16 May 1997.
- [7] Jha K and Gurumoorthy B. Automatic propagation of feature modification across domains. *Computer Aided Design*, 32(12):691–706, 2000.
- [8] Laakko T and Mantyla M. Feature modeling by incremental feature recognition. *Computer Aided Design*, 25(8):479–492, 1993.
- [9] Nalluri Rao SRP. *Form feature generating model for feature technology*. PhD thesis, Department of Mechanical Engineering, Indian Institute of Science, Bangalore, India, 1994. <http://www.cad.mecheng.iisc.ernet.in/thesis/Nalluri-thesis>.
- [10] Bidarra R, Dohmen M, and Bronsvort WF. Automatic detection of interactions in feature models. *Proceedings of 1997 ASME Design Technical Conferences*, 1997. paper number DETC97/CIE-4275.
- [11] XOX Corporation (1450 Energy Park Drive, Suite 120, St. Paul, MN 55108, USA.), *Shapes Kernel - A System for Computing with Geometric Objects*, 2.1.5 edition, May 1996.
- [12] National design repository. <http://repos.mcs.drexel.edu> (As on March 2003).