Distributed Maintenance of Multiple Product Views

Christoph M. Hoffmann *
Department of Computer Sciences
Purdue University

Robert Joan-Arinyo †
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

November 11, 1999

Abstract

We present three mechanisms for maintaining consistent product views in a distributed product information data base. The mechanisms are used when one of the views makes a change to the product model and the other views must be updated to maintain consistency.

Keywords: CAD, Master Model, Distributed Data, Product View, Features, Constraints, Neutral Representations.

1 Introduction

Data bases are an important element in discrete product design and manufacture. A key component is the CAD model that primarily captures the shape design, but increasingly has been enriched with other design and analysis data. The integration of different product information domains has evolved into the concept of a *master model*, a single repository in which resides all relevant product data. The master model concept has been embraced by industry, but it raises significant technical problems that continue to be studied in research.

Different activities in product design and manufacture examine different subsets of the information in the master model. The presentation of such an information subset has been called a *view*, [10]. Maintaining views consistently

*Supported in part by ONR Contract N00014-96-1-0635 and by NSF Grants CDA 92-23502 and CCR 95-05745.

†While on leave in the Department of Computer Sciences, Purdue University. Partially supported by the CICYT Spanish Research Agency under Grant TIC98-0586-C03-03.

is a central problem in research on product design and manufacture and is the subject of this paper.

In [17] we have proposed a distributed approach to the creation and maintenance of a master model data base. We favor the distributed approach for two reasons:

1. The creation of a monolithic system from the ground up is, in our opinion, not only difficult, but also leads to a realization that would pose difficult software maintenance problems. Creating a master model repository by combining collaborating, but otherwise autonomous, subsystems offers more flexibility and a greater ability to adapt to new information domains that should be integrated in the future.

2. It appears unrealistic to expect that industry would rely on a single system and a single software provider. The typical situation is that a company is committed to a specific CAD vendor whose CAD data is partially inaccessible except through the CAD system, and that this partially opaque data has to be integrated with other data, possibly of corporate origin and often inaccessible to the CAD vendor for proprietary reasons.

So, a distributed federation approach to designing a master model repository appears to be a practical one. It also poses some interesting intellectual challenges.

In the earlier paper, we identified two mechanisms for maintaining consistency of views, an external information association mechanism, and a constraint reconciliation procedure. We expand on the details and applicability of those mechanisms and add a third mechanism as a complementary technique for maintaining consistent views under distributed updates. We distinguish several categories of master model update:

1. shape changes,

2. changes of parameters, dimensions, and constraints, and

3. changes of attributes.

Of these, shape changes are the most difficult ones to respond to. Not all changes can be dealt with fully automatically. Therefore, we impose restrictions on shape changes. When those restrictions are not met, human intervention would be required. Those restrictions will be discussed in detail and justified.

In this paper, we illustrate our techniques with the situation in which there are two views, the view of the designer who creates a solid shape by material additions and subtractions, and the view of an NC machining program engineer who would define the same shape purely by subtractions. Clearly, those views impose different feature structures. We wish to allow that both views make

alterations to the net shape. When one view makes a change, we would like the other view to adjust to that change automatically, and vice versa. This poses a fundamental challenge not easily met at this time.

Commercial CAD systems cannot accommodate such view differences. Instead, they implement the paradigm of a premier view, the design view, which must be used to make all net shape changes. From the design view all other views arise as derivative. This is not ideal in manufacturing, especially in certain applications. For example, in aircraft engine design different disciplines have very different views: The structural view considers the design of solid shapes, whereas the aero-thermal view must reason about the design of the passages between the solid shapes, leading to an entirely different conceptualization of the product shape. Research is needed to address these needs.

2 Prior Work

Recent efforts respond to the increasing need for tools that support concurrent design, manufacture and related activities in the product life cycle. It is widely accepted that these activities should be carried out in distributed and heterogeneous computing environments sharing a unique product model to guarantee consistent information, [1, 2].

Distributed and heterogeneous computing environments have been studied in several recent works. In [16], Han and Requicha report on the implementation of a distributed environment encompassing a simple feature-based design system, a geometric server, an automatic feature recognizer and a graphics renderer, all running as separate processes. The geometric server is a central server and the other components are its clients. The motivation of this work was to interface a unique feature recognizer with several solid modelers. The goal is achieved by building the geometric server as a set of solid modelers, each augmented with a software wrapper called *adaptor* that provides a uniform application programming interface. In this environment the geometric server stores both the net shape generated by the design system and the features extracted by the feature recognizer.

In [12], De Martino *et al.* present a distributed, object-oriented, feature-based system. An *intermediate modeler* is the server for a number of networked, distributed application clients. The intermediate modeler maintains a homogeneos, multiple-view, feature-based representation of the part. From this model, specific views can be derived. To maintain consistency between different views, only the designer client is allowed to modify the model. In this system, a single data structure stores the information of all views.

Hoffmann and Joan-Arinyo, [17], developed an architecture for a product master model that federates CAD systems with downstream application processes for different feature views that are part of the design process. The ar-

chitecture, based on a client-server model, addresses in particular the need to make persistent associations of design information with net shape elements. Moreover, the design respects the need of commercial CAD systems (and of downstream application clients) to maintain proprietary information that must not be disclosed in the master model.

To deal with the consistency and association problems systems are organized as either one-way or multi-way architectures. In *one-way* architectures, features in an application view are derived from the features that belong to a privileged view, usually the design view. The designer defines this view and conversion modules derive application-dependent feature models. If a modification is required by a downstream application, it must be entered in the privileged view first. Only thereafter can one derive new, application-dependent views; [8, 12, 16]. In the one-way approach, feature conversion is triggered by one of two different strategies. In one strategy, feature conversion is delayed until the design is considered completed, [12]. In the other strategy, also called *incremental feature recognition*, the conversion process is triggered immediately after the completion of each feature attachment operation in the design view, [14, 19, 21].

In *multi-way* architectures, modifications required by an application are introduced in the view in which the need for them arises, and each modification, in any view, is propagated automatically to every other view; [5, 10, 17, 4]. Work based on this assumption may neglect to explain precisely how a feature view, other than the design view, can change the net shape of the design, and there is a paucity of techniques to formalize such changes. The work by Bronsvoort *et al.* is a notable proposal in that respect, [5, 9, 11, 13, 4]. This work models the net shape by a cell complex where the cells are refined such that every feature of an application view is composed of entire cells. That allows one to edit shape mechanically in any feature view and to achieve consistency across all views using constraint techniques. If an inconsistency between different views is found, the approach rebuilds the view that generated the inconsistency. The view is rebuilt incrementally by first removing some features and then adding new features.

In this paper, we continue to explore the architecture proposed by us in [17]. We examine first net shape changes that can be accommodated by other clients purely by constraint schema reconciliation ([17]). Next, we consider the effect of a net shape edit in which the feature structures of different client views need to be rebuilt. Here, we propose an algorithm that restructures the feature view of a client that did not edit the net shape, to make that structure consistent with the new net shape. Only if the algorithms fails do we require human intervention. Finally, we revisit the question of updating nongeometric information and relationships that are associated with the net shape elements.

3 Master Model Server and Clients

We assume the master model (MM) scenario described in [17]. There is an object-oriented MM server that records all information to be shared explicitly among the participating subsystems. The clients that connect to the MM server are assumed to be autonomous but collaborating. That is, they may be in possession of undisclosed proprietary information that is important to their role in product design and analysis, but they pledge to follow the protocols and conventions of the MM server and, in particular, disclose what shared information in the MM server repository is of interest to them.

The MM server receives notifications from a client that wishes to edit the product model, according to its own view. The client then proposes the edit and transmits the changes to the MM server. The MM server processes the changes and notifies the other clients that are affected by those changes in accordance with the protocols explained in [17]. If every interested client can successfully update, then the MM repository commits to the change and the edit proposed by the client is successful. Otherwise, the edit must be rolled back. We omit a description of the routine mechanisms needed to implement distributed, object-oriented data bases and the various mechanisms for locking and committing to such transactions.

The CAD system is one of the clients of the MM server. It is assumed that the CAD client publishes at least the net shape for deposition in the MM server. Feature information may, but need not, be published, as well as dimensional and constraint information. Note that this information can be published *explicitly* as a neutral data structure, or *implicitly* as a set of interrogating methods that produce information in response to queries. In an object-oriented MM server this distinction can be made transparent, perhaps with differentials in performance.

Accounting for shape changes requires a persistent naming mechanism and a protocol for expressing change. Persistent naming is, in essence, a mapping from shape elements (vertices, edges and faces) of the old net shape to those of the edited net shape. It is supported to by most CAD systems, in the following way:

> An application program accesses the shape generated by the CAD system and asks for a persistent name J of a particular face. J is generated by the CAD system. After the CAD system has made some edits, the new shape is likely a different data structure. Now the application program can ask the CAD system "tell me the face that has the persistent name J." The CAD program may respond with an identification of a face in the new net shape, or else that the face is no longer present and that the persistent name has become invalid.

Note that this persistence mechanism can be used to associate attributes or relations with shape elements, such as surface finish or tolerances. Such *ex-*

ternal associations can be updated automatically after edits by the associating application *without* the explicit involvement of the CAD system, at least for many updates. It is therefore a basis for distributed maintenance of product information.

As explained in [17], more information is desirable to manage external associations. Specifically, the *change protocol* is a neutral, qualitative description of the shape change that allows a greater degree of automatic external re-association. Moreover, the change protocol does not disclose proprietary methods the CAD system might use internally to manage persistence, and is therefore a realistic candidate for practical use.

4 Synopsis of the Results

We assume a MM server as described before. We wish to coordinate two separate views, by two clients. Each view has access to the net shape in the MM server, but may have a different feature decomposition and design history of the net shape. Each view maintains features and constraints. In addition, there are certain attributes associated with elements of the net shape in each view. We consider three types of edits and how they must be made consistent in the views:

1. In one view, a dimensional constraint is changed and, with it, the net shape. We want to update the other view consistent with the new net shape.

2. In one view, a feature is added or deleted, deriving a new net shape. We want to update the feature structure of the other view so it is consistent with the new net shape.

3. In one view, a shape change has been made that affects relationships among net shape elements in the other view. Update the relationships of the other view consistent with the change.

For the sake of specificity, we illustrate these edit operations by using a CAD client and a machining process planning (MPP) client, and consider geometric dimensioning and tolerancing (GD&T) relationships.

We restrict to changes in a product view that affect the net shape. Mechanical artifacts perform their function primarily through the interaction of shapes. Therefore, this assumption is not a strong restriction. Since geometric shape can be structured in many different ways, the main obstacle to coordinating views is that one view does not necessarily understand how another view structures the same shape. Therefore, it would be meaningless to announce to the MM server that feature X has been deleted from view Y, or that a particular dimension has been changed.

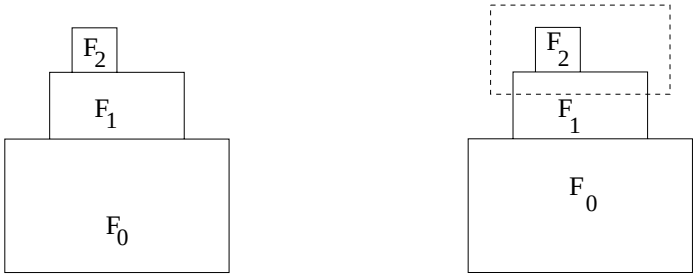

Figure 1: Left: The base feature F_0 is a block and is augmented by two extrusion features F_1 and F_2. Right: a cut of the dashed shape would eliminate F_2 altogether. Since F_2 does not contribute to the net shape, it ought to be dropped from the feature structure.

4.1 Change of a Dimension

A feature may be modified by changing a dimensional constraint. If the change does not alter the topology, this type of change is a good candidate for constraint reconciliation proposed in [17]. When the topology does change, the modification should be treated as a feature change. The change of a geometric constraint, for instance dropping a perpendicularity requirement or changing it to a specific angle, is approached in the same way.

4.2 Feature Change

We restrict to features that are extrusions and revolutions only. We assume that each feature makes a direct contribution to some of the faces of the net shape. Situations illustrated in Figure 1 are not considered. When a feature is added, deleted, or modified, the new net shape is communicated to the MM server, and the change protocol details how the new net shape relates to the old net shape.

A different view V updates by an algorithm that adjusts the feature structure of its own view to replicate the new net shape. The algorithm may backtrack and could fail. The view V may have a different feature structure. We do not change this structure automatically. To increase the flexibility of updating with a different feature structure, however, we consider reinterpretation of the way in which the feature has been generated, and in this sense we allow changing a feature.

4.3 Implied Attribute and Relationship Changes

In some cases, the change of net shape requires a change, in a different view, of an attribute of a net shape element, or of a relationship among several net shape elements. We propose a rule-based update procedure that accounts for the specific nature of the attribute or relationship. The rules are individual to

the nature of the attribute, but the way in which they are evaluated is general. The algorithm, first sketched in [17], applies the rules by a propagation through the change protocol.

5 Feature Views and Changes

We assume that the detailed feature structure maintained by a view is not disclosed to the MM server and remains private information of each client. We want to maintain consistent but different views.

Client 1 updates the net shape with some feature editing operation and derives a new net shape. Client 2 must update its own feature structure and obtain a consistent reinterpretation of the new net shape. In general, this is a rather broad problem statement, and a fully automated solution would include many techniques from feature recognition. See, for example, [3, 15, 21, 22, 23] and references therein.

We restrict the problem and the geometric operations, and allow the reinterpretation process to fail. If it fails, human intervention is required. This is justified because some adjustments that might be performed automatically should be reported to the user nevertheless, since they might violate design requirements the other view has been unaware of. Such notification is important in practice, but is not discussed here.

Recall that we assume that the features are cuts and protrusions that subtract from or add to the netshape. The generation method is assumed to be by extrusion or revolution. We exclude operations such as shelling, drafting, and blending. Note that the latter two could be derived by detailing steps from an appropriate attributation of the shape design. Finally, we consider the following attachment operations: *from–to* with explicitly designated targets, *from–prev*, *through–next*, *from all*, *through all*, and *blind* with a numerical extent; see also [7].

5.1 Feature Reinterpretation

We will allow feature update operations that redefine the feature generation method: Recall from [7] the notion of a *proto feature*. In the case of an extruded feature, the proto feature is a blind extrusion, to a particular extent, of the cross section, so that the chosen attachment attributes can be conceptualized as Boolean operations on the proto feature and the prior geometric shape. In the case of a revolved feature, the proto feature is a blind revolution, again of sufficient extent.

An *elementary reinterpretation* is a change in the generation method and cross section of a proto feature such that the same surfaces are generated but in a different way. The following cases are considered elementary reinterpretations:

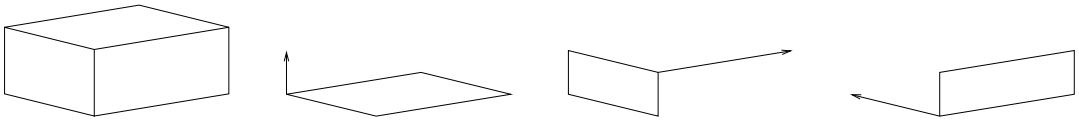

Figure 2: Three ways to create a parallelepiped

Parallelepiped extrusion: The proto feature is shown in Figure 2 and can be obtained from three different cross sections by extrusions in three directions. Note that the extrusion directions may be at an angle to the cross section plane.

Cylindrical revolution: The proto feature is shown in Figure 3 and can be obtained by a revolution or by an extrusion.

When necessary, we allow switching from one interpretation to another one.

For example, assume that the cylinder of Figure 3 was viewed by client 1 as a revolution and by client 2 as an extrusion, then a change in the cross section by client 1 may destroy the interpretation of client 2, and we may have to change the feature definition of client 2 from an extrusion to a revolution.

A *general reinterpretation* is the reinterpretation of a feature as a group of features, by a decomposition. For example, assume that client 1 creates a cylinder feature by revolution that has been interpreted as an extrusion by client 2. Client 1 edits the cross section, obtaining the new feature shown in Figure 4. The modification invalidates the extrusion interpretation of client 2. However, by splitting the extrusion into two separate extrusions, client 2 can maintain the interpretation.

Now consider any net shape obtained by the extrusion and revolution operations coupled with the attachment rules from before. Every surface is either a cylindrical or a revolute surface. Planar surfaces and surfaces on a right-circular cylinder are considered a special case of both the cylindrical and revolute surface type. Every cylindrical surfaces can be obtained by an extrusion, and every revolute surface by a revolution, from a suitable cross section element. Moreover, surfaces that are both revolute and cylindrical may allow elementary reinterpretation, depending on how they are delimited.

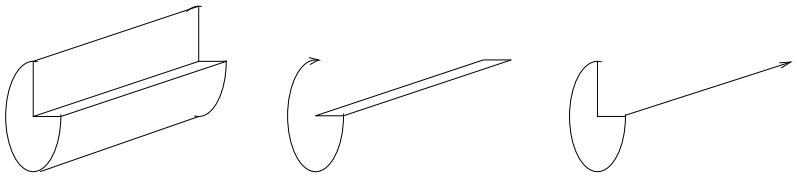

Figure 3: Two ways to create a cylinder

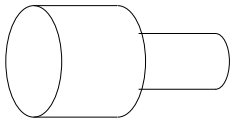

Figure 4: One revolved feature or two extruded features

When generating a feature, additional surfaces may be obtained by the attachment rules that are not part of the extrusion or the revolution. Therefore, except for such surfaces, different feature structures of the same net shape arise only as follows:

1. A parallel extrusion has an overlapping cross section;
2. A concentric revolution has an overlapping cross section;
3. An elementary reinterpretation of a part of the feature.

Note that the attachment rules blur recognition of these cases because, by feature collision, they may create delimiting feature surfaces that appear to exclude elementary reinterpretation. By considering first the proto feature shape this difficulty is ameliorated.

6 Change of a Dimension

When the editing client alters only constraints, under the assumption of topological invariance, the net shape change is such that the number of features and their interdependence in the updating client remains the same. Updating client structures can be made consistent with the new net shape either by reconciliation or by history-based adjustment.

6.1 Adjustment by Reconciliation

When the editing client alters only constraints, there is the possibility that the change impacts a single cross section only. Since the net shape mapping is communicated, the MM server can be told that the new net shape came about by constraint changes as well as on which geometric elements the constraint is defined. If those elements are generated by a single cross section and the prior geometry, in the updating client, constraint reconciliation can be used as described in [17].

At this time, the majority of CAD designs are history-driven, and the major CAD systems do not use variational spatial constraints to define a shape design. Because of this limitation, constraint reconciliation is restricted to those constraints that map, in the editing client, to a single feature. For other constraint changes we advocate the techniques explained later on.

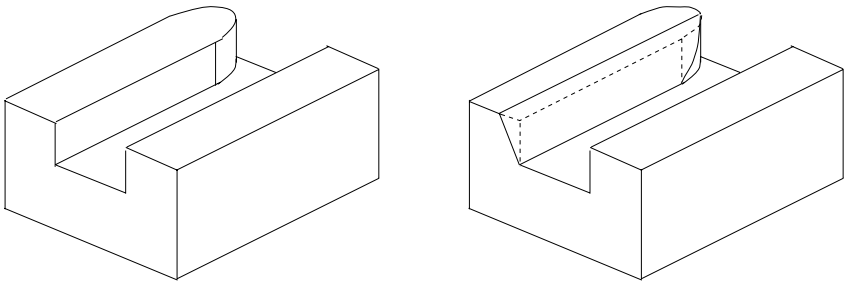

Figure 5: Left: Initial net shape. Right: Edited net shape.

Since the change protocol has identified all net shape alterations, including face relocations, we can identify the cross section impacted by a constraint change and recognize the fact that reconciliation is applicable. In the following we assume that a single constraint has changed. The case of multiple changes is a straightforward extension.

In general, the changed constraint does not appear explicitly in the affected cross section. It is possible, in principle, to express the functional dependence of the changed constraint on the other constraints in the cross section, and to derive from that dependence how the constraint values of the updating client should be changed. However, those dependencies can become intractable symbolic expressions, and it is much simpler to take a procedural approach.

Since the new constraint has been used to derive the new net shape, a cross section recomputation, from the new net shape, yields the appropriate cross section. From this cross section, the constraint values for the updating client can be computed by measurement. Thereafter, we must verify that the new cross section correctly generates the new net shape.

The recomputation of the cross section involves both geometric changes to the cross section and changes to the constraint schema. Constraint changes may be *trivial*, such as the change of an angle or distance constraint, or they may be *exceptional*, such as the change of a parallel constraint to a nonzero angle, the change of an incidence constraint to a nonzero distance, and so on. If the exceptional changes concern explicit, user-defined constraints, then the user should be notified of the adjustment performed by the algorithm. This is a measure that accounts for the possibility that the ensuing shape change may violate a design choice that the editing client may have been unaware of.

An example is given in Figure 5. If the features have been generated as shown in Figure 6, then constraint reconciliation is applicable. The designer's view, shown on the left of Figure 6, consists of two protrusions, labeled B and D, placed on top of block A. Protrusion B is generated by extruding a rectangular cross section perpendicular to the top face of the block A. The cross section has width d and height h. One of the ends of protrusion B has been rounded with

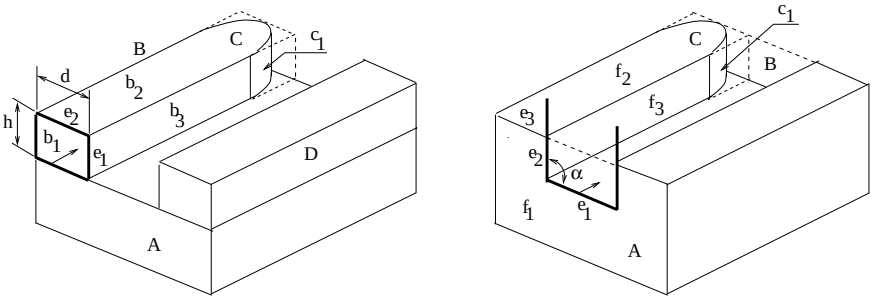

Figure 6: Design view left; MPP view right. Constraint reconciliation is possible.

a circular cut, C.

The view of the MPP client, shown on the right side of Figure 6, is different: It consists of a block A into which a slot has been cut. Since the angle α between edges $f_1.e_1$ and $f_1.e_2$ has a meaning to MPP, it has been explicitly established as an angular dimension. Similarly in the designer's view, the rounding of one side of the slot has been generated with a circular cut, C.

Figure 7 shows how the MPP client can edit by giving a new value α' to angle α. The design client, to update its own view, first computes a cross section to generate the new shape of the extrusion B by replacing edges $b_1.e_1$ and $b_1.e_2$ with edges $f_1'.e_2$ and $f_1'.e_3$, respectively. Then, new values for the dimensional constraints are computed. The old dimension value d becomes d' and the 90° constant angle between edges $b_1.e_1$ and $b_1.e_2$ becomes the varying angle dimension with value β.

6.2 History Adjustments

Suppose we have changed a constraint in the view V_0. In that view, the constraint is on prior geometry and on elements of a single feature only. However, that does not mean necessarily that we can account for the change in the other view in a single cross section: Consider the design view shown in Figure 8. If

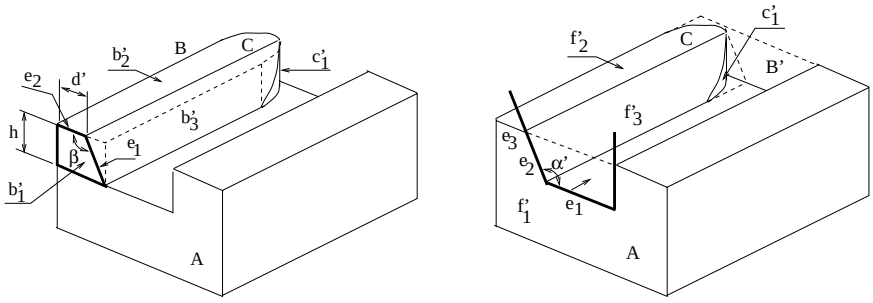

Figure 7: Left: Updated design view. Right: New MPP view.

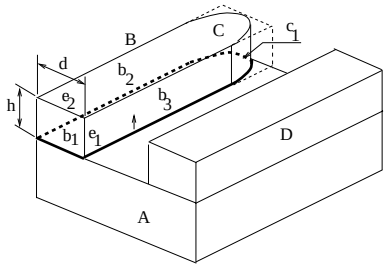

Figure 8: A different design view of the original net shape. Constraint reconciliation is not possible, and reinterpretation would be required.

the edit shown in Figure 7 has been carried out by the MPP client, the CAD client cannot update by constraint reconciliation alone. Instead, the CAD client has to use more general techniques. Let us show how a history-based approach can handle this situation.

We distinguish two types of proto feature faces, *perimeter* faces and *extent* faces. Perimeter faces are the faces swept by the geometric elements of the proto feature cross section. Extent faces include the front and back faces that bound the proto feature extent. As we shall see, extent faces become involved in updating operations explicitly only when reinterpretations are done.

The change client 1 performs is communicated to client 2 by the following information: The new net shape, and the change protocol that establishes a mapping between the shape elements of the old and the new net shapes. Client 2 examines the cross sections in an order that is compatible with its feature history. Recall the assumption that the net shape change is such that the number of features and their interdependence remains the same in the updating client, a feature is adjusted in the following situations:

1. Its cross section contributes to the old net shape a boundary element that has been changed in the new net shape;

2. The feature attachment method refers to a shape element that has been changed.

In each case, the cross section is recomputed for the new net shape and the attachment method validated. If the cross section cannot be adjusted consistent with the new feature shape, a question that is answered generatively, we attempt elementary reinterpretation. If that fails as well, then this attempt at feature adjustment fails. We can then backtrack and try a different feature sequence.

Note that feature attachment induces a partial order on features. If feature A uses geometric elements of feature B, to define the cross section or the attachment method, then the creation of B must precede the creation of A. We

consider linear orderings that are consistent with the partial order. Since linear orderings are not unique in general, backtracking is possible upon failure.

Assume that the preceding features have been already updated. The next feature, in sequence, that contributes to changed elements of the new net shape, is then processed using Algorithm 1

Algorithm 1

1. Identify the underlying proto feature.
2. From the change protocol, extract the subset of faces that are part of some perimeter face in the proto feature being updated.
3. Replace in the proto feature those perimeter faces that have changed with the new ones.
4. Compute cross sections C_1 and C_2 at each end of the proto feature.
5. If $C_1 \neq C_2$ or $C_1 = C_2$ but they are not valid cross sections go to step 6. Otherwise $C_1 = C_2$ is the new proto feature cross section. If necessary, adjust it by reconciliation. If required, adjust proto feature extent. Stop.
6. Try elementary proto feature reinterpretation.

Changed perimeter faces of the proto feature are replaced with new ones using Algorithm 2.

Algorithm 2

For each face in the subset of changed perimeter faces do

1. If the supporting half space of the old and new faces are the same do nothing.
2. If the supporting half space of the new face is a parallel translation of the old face then
   a) Translate the old face to the half space that supports the new face.
   b) If needed, adjust faces in the proto feature incident to the old face.
3. Otherwise
   a) Replace the old face with the new one.
   b) If needed, adjust faces in the proto feature incident to the old face.

Note that for planar faces a supporting half space is a parallel translation of another if, and only if, the normals to the supporting planes are parallel. For cylindrical faces, the axes of both cylindrical half spaces must be parallel.

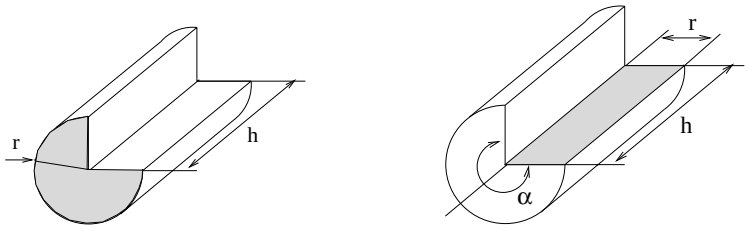

Figure 9: Cylindrical proto feature with two possible interpretations. Left: extruded, client 1. Right: Revolved, client 2.

Parallelepiped extrusions have four perimeter faces and, as we have already shown, there are three different possible ways to generate them. If F designates the union set of perimeter and extent faces of the proto feature, reinterpretation is attempted as follows by Algorithm 3:

Algorithm 3

1. Group the four perimeter faces of the updated proto feature into two sets, say $F_1 = \{f_{11}, f_{12}\}$ and $F_2 = \{f_{21}, f_{22}\}$, each with two parallel faces.
2. For $i = 1, 2$ do Steps 3 to 5.
3. $F' = F - F_i$ is the subset of proto feature faces which are candidates to be perimeter faces in the feature reinterpretation.
4. Build the cross section C_1 as the intersection of the faces in F' with the plane supporting the face f_{i1}, and build the cross section C_2 as the intersection of the faces F' with the plane supporting face f_{i2}.
5. If $C_1 = C_2$ then we have found a valid cross section for a sweep of C_1 from f_{i1} to f_{i2}. If needed, adjust the proto feature extent. Stop.
6. If there were no matching cross sections, the reinterpretation fails.

A revolved feature that is either a right cylinder or a right cylinder with a missing wedge allows reinterpretation as a revolved or an extruded feature; Figure 9. Assume that client 1 interprets such a feature as extruded (Figure 9 left) while client 2 interprets the same feature as revolved (Figure 9 right). It is clear that editing a parameter of client 1 can be directly updated in the view of client 2.

However, when client 2 edits the cross section, client 1 can update only if the changes keep the top and bottom faces perpendicular to the axis of revolution and maintain a straight line parallel to the axis of revolution; Figure 10. Otherwise, client 1 must reinterpret the feature as revolved. Therefore, for revolved features, elementary reinterpretation is as follows:

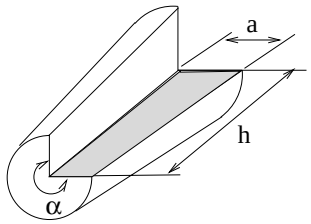

Figure 10: Updated feature that can be interpreted only as revolved.

Algorithm 4

1. If the feature has two planar perimeter faces (revolution is smaller than 360°) take one of them as the new cross section C. Go to step 4.

2. Otherwise build a plane π perpendicular to the old cross section sketching plane and through the center of the cylindrical perimeter face.

3. Build the new cross section C by intersecting plane π with the updated proto feature.

4. Reinterpret the proto feature as a revolution whose cross section is C and whose axis is a line parallel to the old sweep direction through the center of either the bottom or the top proto feature face.

5. Adjust proto feature extent, if needed.

7 Feature Addition and Deletion

Algorithm 1 assumes net shape changes under which the number of features, and their interdependence, in client 2, remains the same. This assumption will often be violated when allowing addition or deletion of a feature. We consider such edits now and assume, furthermore, that the feature added or deleted does not have any other feature depending on it. Such a feature would be a leaf feature in the dependency graph. If nonleaf features are to be deleted, then we would either delete the dependent features first, or request that the editing client restructure its design such that the dependencies are eliminated. The latter operation is familiar from commercial CAD systems.

There are two fundamental issues that arise when adding or deleting a feature. First, the feature vocabulary of the editing client may be richer than that of the updating client. For instance, the CAD client can add protrusions, but the MPP client may be restricted to cuts from a stock shape only. Second, the feature structure of the editing client may be substantially different from the structure of the updating client. For example, the CAD client may be adding a

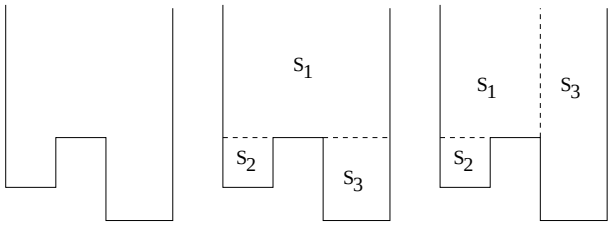

Figure 11: Compound cut. Left: Design view. Middle: MPP view. Right: A different MPP view.

single profiled cut, but the MPP client may have to decompose the profile into a compound structure of simpler profiles.

7.1 Adding a Protrusion or a Cut

The single feature added is, by definition, a leaf feature. If the updating client has a compatible feature type, then the change requires constructing a new cross section and generating the appropriate feature. Difficulties arise for incompatible feature vocabularies.

When the CAD client adds a profiled cut, the difficulty for the MPP client may be that the cut must be decomposed into several cuts accounting for available machining processes. The problem is solved by *tiling* the CAD profile with elementary profiles that cover it. For instance, consider the cut profile of Figure 11. The profile on the left can be decomposed into three rectangular cuts, S_1, S_2, S_3. Note that different decompositions are possible. Tiling algorithms are easy to devise. More sophisticated tilings would consider cost, machining characteristics, etc. See for example [18, 20].

A more difficult case is when the editing client has feature operations of a type not available to the updating client. For instance, assume that the CAD client adds a protrusion, but the MPP client can only add cut features to a base feature, the stock. Then the update for the MPP client is more difficult. An example is shown in Figure 12. Here, the adjustments of Algorithm 1 cannot succeed because the mapping between the old and the new net shape does not

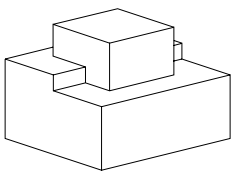

Figure 12: The CAD client adds the boss straddling the step. An MPP client would have to update by enlarging the stock and adding several cut features.

identify the top face of the boss as belonging to the face set created by the base feature, i.e., the stock.

The example illustrates the problem for shape edits that are made with a feature vocabulary richer than that of the updating client. Updates with a restricted vocabulary can succeed only if geometric reasoning techniques are employed. Such techniques have been developed in the context of feature recognition in, e.g., [6] and [15].

7.2 Deleting a Protrusion or a Cut

Since the deleted feature need not correspond to a single leaf feature in the updating client, several features may have to be adjusted using Algorithm 1. Moreover, deleting a protrusion in the CAD client may imply adding cut features in the MPP client. We also expect the possibility that features become redundant in the course of executing Algorithm 1, for instance when deleting a cut.

8 Updating Attributes and Relations

We consider changes that require updating secondary information associated with net shape elements. For example, consider the case where the CAD client does not maintain or process tolerancing information, material properties, surface finish, or engineering notes. We consider these information domains conceptually as relationships that are defined for net shape elements. In the case of unary relationships, such as surface finish, it is customary to speak of *attributes*. Whether the relationships are unary, binary or of higher degree,[1] the basic problem is simply to maintain the associations correctly. This is done by transferring the associations, suitably edited, from the old to the new net shape.

As analyzed in [17], the mapping that relates the old net shape to the new net shape can be represented as a graph whose nodes are net shape elements and whose edges are operations on them. The operations are *move*, *change*, *split*, *merge*, *delete*, and *new*. The graph describes the changes each shape element of the old net shape undergoes and the net shape elements, if any, of the new net shape that correspond or to which a contribution is made. Figure 13 illustrates these concepts. Circles with lower-case letters represent net shape geometric elements. Circles with capital letters represent attributes associated with net shape elements. The graph is directed and acyclic, and is therefore a good candidate for rule-driven algorithms to update associations. The rules include computations on the net shape elements involved.

[1] We could represent form features as n-ary relationships on a suitable set of faces, edges and vertices.

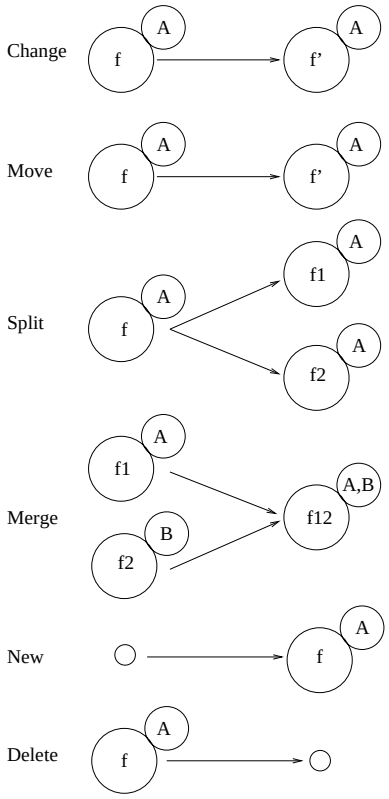

Figure 13: Graph description of changes undergone by net shape elements.

The update rules depend on the information domain. In [17], we illustrated this approach using notes attached to net shape elements. Here, we consider a GD&T attribute specifying a nominal dimension between two parallel faces with a parallel tolerance. The rules are applied at every net shape edge where there is an change of one or the other parallel face. Using the rules, we "evaluate" the change protocol graph in the order of the changes, from the graph roots (elements of the old net shape) to the leaves (elements of the new net shape).

change: The net shape element has changed in area, but not in position and orientation. No action is required.

move: The net shape element has changed in position and orientation. If not in an intermediate position, evaluate the nominal dimension from the new position and orientation. If no longer parallel within tolerance, or if no longer within the nominal dimension range, notify user.

split: Replicate the GD&T attribute and attach to all split descendants.

merge: Merge all attributes of the merged entities.

new: No action required.

delete: Notify user that the GD&T attribute has been orphaned.

We see that the update activity for attributes is well-suited to a rule-based approach. The key operational devices needed in general are

1. Call a user-defined evaluation routine with a Boolean result.
2. Issue a user-defined notification. Notifications are minimally: attribute orphaned, attribute violated, attribute replicated.
3. Replicate an attribute.
4. Merge an attribute set.

Moreover, those devices are general, independent of the attribute domain, except for a call-back mechanism which is domain-dependent.

9 Summary and Conclusions

We have explored the requirements for the distributed maintenance of consistent master model information in a federated architecture, in which different software clients connect to a server and collaborate by disclosing information required by other clients to construct and maintain consistent views of the design. In particular, we have focused on maintaining shape-related information without forcing

the CAD client to disclose proprietary design and editing information. In conjunction with earlier work, we have argued that it is possible to construct such a system that succeeds in automating a wide range of view updating operations. From an applications point of view, we find it especially important to account for the need to preserve privacy of proprietary information.

We find that maintaining different feature views is complicated by the current, history-based CAD design style. In our algorithm to partially automate updates, we apply a core set of techniques familiar from the feature recognition literature when dealing with updating the feature history. In many situations an adjustment is possible purely by constraint reconciliation, a concept we introduced earlier. Were it not for the sequential design history implemented by CAD systems, constraint reconciliation would be more widely applicable. Finally, the maintenance of attributes can be completely automated, and, with it, the maintenance of many downstream views that can be derived from attributes and relations maintained on the net shape elements.

The limitations of our algorithms are less consequential in practice than might seem so at first glance. We note that major changes of the net shape, such as the ones indicated in some of the examples, probably require human review by different individuals, so that automating radical design changes is not appropriate. Only routine changes should be automated. In view of this situation, it may not be worthwhile looking for perfect algorithms, from an applications point of view. Of course, a perfect history adjustment algorithm that can handle all possible situations would be an interesting technical and intellectual accomplishment.

References

[1] *IEEE Computer. Special issue on Concurrent Engineering*, 26(1), 1993.

[2] *Computer-Aided Design. Special issue on Network-Centric CAD*, 30(6), 1998.

[3] V. Allada and S. Anand. Feautre-based modelling approaches for integrated manufacturing: state-of-the-art survey and future research directions. *International Journal for Computer Integrated Manufacturing*, 8(6):411–440, 1995.

[4] R. Bidarra. *Validity Maintenance in Semantic Feature Modeling.* PhD thesis, Inf Technology and Systems, Technical University of Delft, 1999.

[5] W.F. Bronsvoort and F.W. Jansen. Multi-view feature modelling for design and assembly. In J.J. Shah, M. Mäntylä, and D.S. Nau, editors, *Advances in Feature Based Manufacturing*, Manufacturing Research and Technology, 20, chapter 14, pages 315–330. Elsevier Science B.V., 1994.

[6] M.A. Chamberlain, A. Joneja, and T.C. Chang. Protrusion-features handling in design and manufacturing planning. *Computer Aided Design*, 25(1):19–28, 1993.

[7] X. Chen and C.M. Hoffmann. Towards feature attachment. *Computer Aided Design*, 27(9):695–702, 1995.

[8] J.J. Cunningham and J.R. Dixon. Designing with features. The origin of features. In V.A. Tipnis and E.M. Patton, editors, *Computers in Engineering Conference and Exhibition*, volume 1, pages 237–243, San Francisco, 1988. ASME.

[9] K.J. de Kraker, M. Dohem, and W.F. Bronsvoort. Multiple-way feature conversion. Opening a view. In M. Pratt, R.D. Siriram, and M.J. Wozny, editors, *Product Modeling for Computer Integrated Design and Manufacture*, pages 203–212, London, UK, 1997. Chapman and Hall.

[10] K.J. de Kraker, M. Dohmen, and W.F. Bronsvoort. Multiple-way feature conversion to support concurrent engineering. In C.M. Hoffmann and J. Rossignac, editors, *4th Symp. on Solid Modeling and Applic.*, pages 105–114, Salt Lake City, UT, 1995.

[11] K.J. de Kraker, M. Dohmen, and W.F. Bronsvoort. Feature validation and conversion. In D. Roller and P. Brunet, editors, *CAD Systems Development*, pages 121–142. Springer Verlag, Heidelberg, 1997.

[12] T. DeMartino, B. Falcidieno, and S. Hassinger. Design and engineering process integration through a multiple view intermediate modeller in a distributed object-oriented system environment. *Computer-Aided Design*, 30(6):437–452, May 1998.

[13] M. Dohmen, K.J. de Kraker, and W.F. Bronsvoort. Feature validation in a multiple-view modeling system. In *16th ASME International Computers in Engineering Conference*, Irvin, NY, USA, 19-22 August 1996. ASME.

[14] J. Han and A.A.G. Requicha. Incremental recognition of machining features. In *Proceedings of the ASME Computers in Engineering Conference*, pages 587–598, Minneapolis, MN, 1994.

[15] J.H. Han and A.A.G. Requicha. Integration of feature based design and feature recognition. *Computer-Aided Design*, 29(5):393–403, May 1997.

[16] J.H. Han and A.A.G. Requicha. Modeler-independet feature recognition in a distributed environment. *Computer-Aided Design*, 30(6):453–463, May 1998.

[17] C.M. Hoffmann and R. Joan-Arinyo. CAD and the product master model. *Computer-Aided Design*, 30:905–919, 1998.

[18] A. Kusiak. Optimal selection of machinable volumes. *Transactions of Institute of Industrial Engineering*, 22(2):151–160, 1990.

[19] T. Laakko and M. Mäntylä. Feature modelling by incremental feature recognition. *Computer-Aided Design*, 25(8):479–492, August 1993.

[20] D.S. Nau, G. Zhang, and S.K. Gupta. Generation and evaluation of alternative operation sequences. In A.R. Thangaraj, A. Bagchi, M. Ajanappa, and D.K. Anand, editors, *Quality Assurance Through Integration of Manufacturing Processes and Systems*. ASME, 1992. (Proc. 1992 ASME Winter Annual Meeting, Publication No. PED-Vol. 56, pp. 93 – 108).

[21] H. Suh and R.S. Ahluwalia. Feature generation in concurrent engineering environment. In J. Rossignac and J. Turner, editors, *Symposium on Solid Modelling Foundations and CAD/CAM Applications*, pages 493–502, Austin, TX, June 5-7 1991. ACM Press.

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

[23] X. Xu and S. Hinduja. Recognition of rough machining features in 2 1/2 D components. *Computer-Aided Design*, 30(7):503–516, 1998.