1994

On Editability of Feature-Based Design

Xiangping Chen

Christoph M. Hoffmann
Purdue University, cmh@cs.purdue.edu

Report Number:
94-067

ON EDITABILITY OF FEATURE-BASED DESIGN

Xiangping Chen
Christoph M. Hoffmann

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD-TR-94-067
November 1994
(Revised 2/95)

On Editability of Feature-Based Design*

Xiangping Chen Christoph M. Hoffmann
Department of Computer Science, Purdue University
West Lafayette, IN 47907-1398

Report CSD-TR-94-067
November 1994
Revised, February 1995

Abstract

We present techniques needed for editing generative designs. When a feature is edited, all features attached later must be reevaluated to satisfy the required constraints and shape references in the initial design. We describe name matching techniques that support the design reevaluation procedure. The algorithms account for failed or multiple matches. The matching uses a naming schema based on historical, topological, and geometric design information that has been described in a companion paper.

1 Introduction

In feature-based design, parts are constructed from a sequence of form feature attachment operations [2]. This construction paradigm lends itself well to separating the design into two layers, one comprising an unevaluated, generic representation, the other comprising an evaluated, instance representation. (see Figure 1). Design intent is edited and stored in the unevaluated representation in an editable format (Erep as in [7]) which can be automatically compiled into an evaluated representation, as explained in [3].

To redesign a part, features are edited; their unevaluated descriptions are automatically modified and recorded by the design interface and are compiled serially by a design compiler[3]. To change the design requires changing only

*Supported in part by ONR contract N00014-90-J-1599, by NSF Grant CDA 92-23502, and by NSF Grant ECD 88-03017.

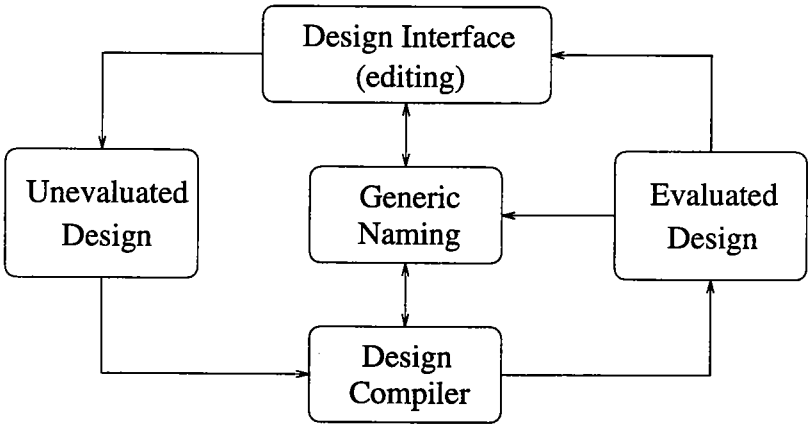

Figure 1: A design model is layered into an unevaluated representation and an evaluated representation.

the unevaluated representation, because the evaluated representation can be reconstructed by the design compiler without explicit user action. The purpose of this paper is to explain the basic mechanisms required for redesign in this design paradigm.

An important aspect of supporting design edits is a *generic naming* schema for topological entities[1]. This naming schema uniquely identifies a topological entity in a design and matches it with a topological entity or entities in the modified design. The major difficulty of the approach is that the evaluated design cannot be used explicitly and is discarded when the matching takes place. However, the referenced topological element may not occur explicitly in the unevaluated representation and therefore has to be deduced by suitable methods. The payoff of this approach is an enormous flexibility when editing designs, support of generic design, and substantial lowering of the functional barriers that exist in current CAD systems [7].

In feature-based design, editing operations can be classified into the following broad categories:

(E1) inserting or deleting an entire feature;

(E2) changing feature attributes, e.g. from a blind hole to a through hole;

(E3) modifying dimension values defining and/or placing a feature;

(E4) changing the dimensioning schema;

(E5) changing the feature shape definition, e.g. changing the cross section.

Here, we concentrate on editing operations (E1) through (E4), and on the name matching algorithms that are needed to support them. Operation (E5) requires certain mapping techniques in addition.

1.1 Editing Semantics

In constraint-based design systems, even changes to dimension values can entail substantial topological changes of the instance representation, and those changes must follow a predictable pattern that makes sense to the user. For example, consider Figure 2. By changing the position of the slot in Part A, the rounded edge ought to disappear in Part A': It seems unnatural to us to round the second, elongated edge, because we conceptualize the slot repositioning as a continuous motion of the feature from the position in Part A to the position in Part A'. In this motion, the length of the rounded edge in Part A diminishes to zero. Yet some commercial systems round the other edge, as shown in the figure.

On the other hand, when modifying the length of the slot in Figure 3, the round of Part B would sensibly be inherited by both segments of the divided edge in Part B', yet some commercial systems signal an error.

We learn from these two examples that the semantics of editing feature-based design deserves attention. We have adopted as guiding principle to consider the intended meaning to be derived from considering the editing operation as a continuous process in which specific events such as edge subdivision are interpreted using rules of positional inheritance. Of course, other interpretations are possible, and our interpretation does not claim to be universally compelling.

In a number of cases, different interpretations of the intended effect of the editing operation seem equally sensible. In those situations, we adhere to the principle that the effect of the modification should not depend on the prior design modification history. Because prior history is no longer is visually available, we feel that using it to decide between different interpretations would confuse the user. For example, consider Figure 4. Part C' was obtained from Part C by contracting the length of the slot, and the blend was spread to the larger edge of C' that "contains" the previously rounded edge. If we now lengthen the slot to divide again the rounded edge, by editing Part C', we would derive the shape of Part B' — not the shape of Part C — because reconstructing C depends on older design history. Note that none of these rules preclude adding an explicit history mechanism that checkpoints designs and allows backing up the design/editing process to earlier variants.

It is clear that the topological changes entailed by such simple editing opera-

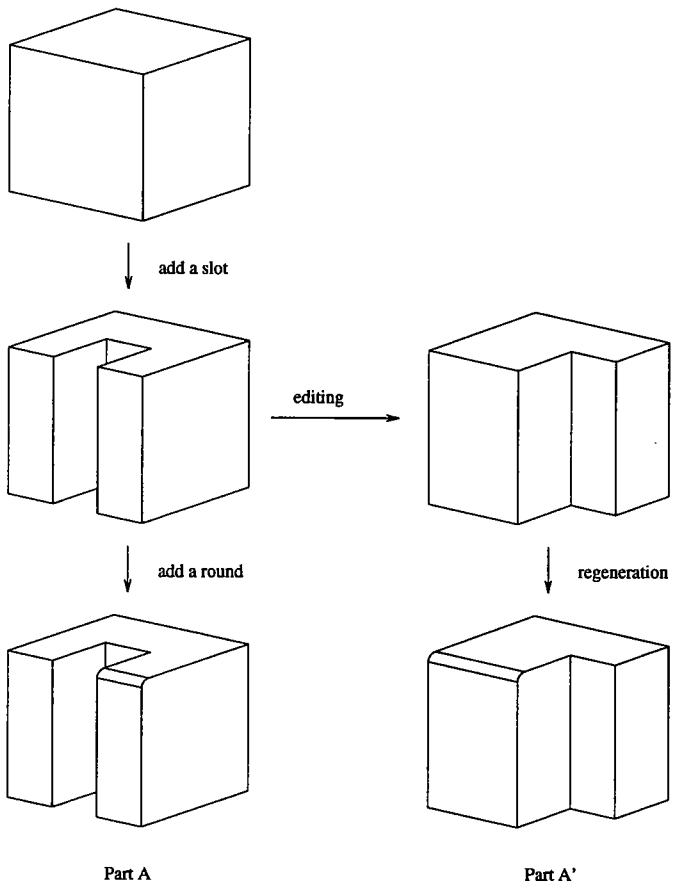

Figure 2: Editing example 1: When the slot (second feature of Part A) is moved to the right and becomes a step, rounding the edge in Part A' should be considered an error.

tions must be supported by a robust generic naming schema, along with sensible matching algorithms that give predictable, meaningful results. In this paper, we propose such matching algorithms so that the unevaluated design can be updated accordingly, and so that a design compiler[3] can instantiate the design consistently.

Matching does not always succeed. Even simple changes of dimension values can imply design variants in which elements referenced in the construction of subsequent features no longer exist. In some cases, plausible substitutes can be constructed, but in other cases matching, and with it reconstruction, can get stuck. In those situations different philosophies could be adopted. We could require some substitution to be made without exception, and accept substitutions that are wrong from the designer's point of view. This strategy makes

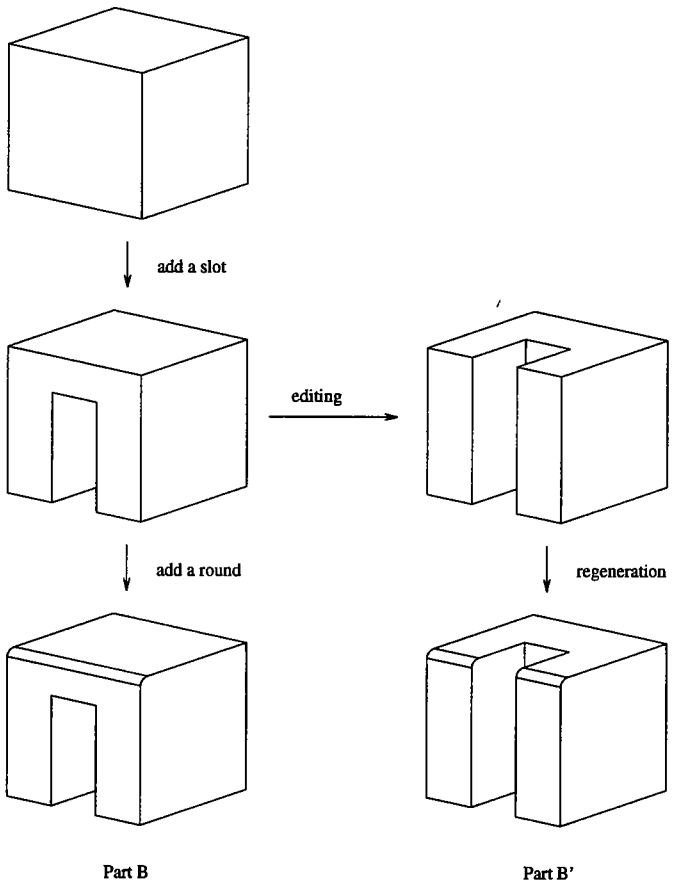

Figure 3: Editing example 2: When the slot (second feature of Part B) is moved to the right and becomes a step, rounding the edge in Part B' should be considered an error.

sense if the editing mechanism has been activated under program control, for instance in an automated design optimization loop. We could also require user intervention in such situations. That would be appropriate if a user initiates design editing. In this paper, we adopt the latter approach.

1.2 Prior Work

Editing operations such as insertion, deletion and modification in general have been addressed by Pratt[9] and alternative approaches are discussed in that paper. No naming mechanism has been suggested that would be appropriate in a constraint-based design environment. In [11], Rossignac has explained the difficulty of feature editing. He and his colleagues implemented feature editing

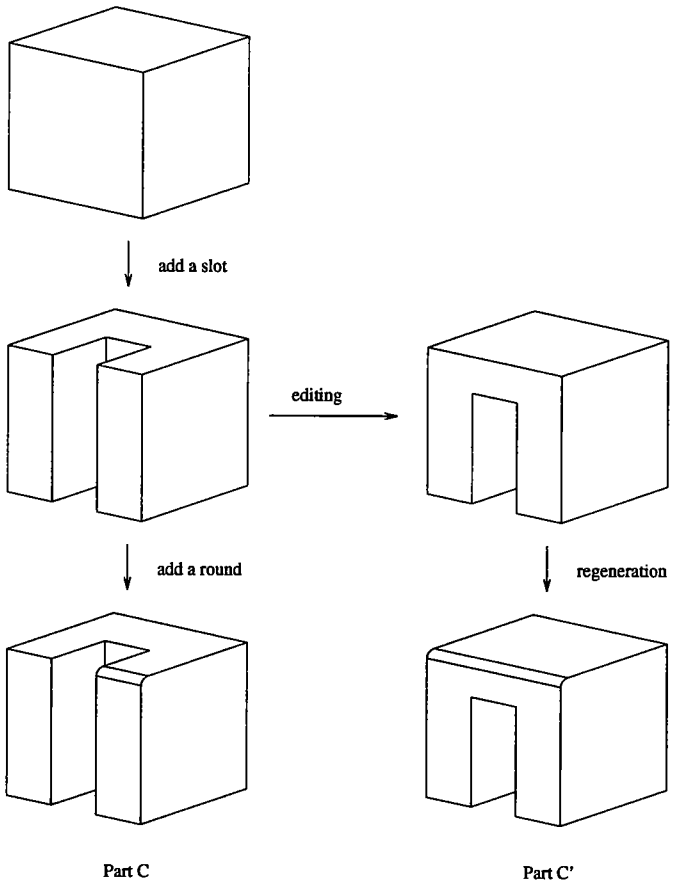

Figure 4: Editing Example 3: When the through slot (second feature of Part C) is changed to a blind slot, the entire edge of Part C' should be rounded. For subsequent editing Parts B and C' should be equivalent.

operations in a 2D system[12].

Current feature-based design systems support many of the editing operations (E1–E5). However, because of limitations their generic naming schemata have with disambiguating feature interactions, many restrictions are placed on the editing operations, and apparent errors are made. In [5, 6], some of these errors have been characterized using Pro/ENGINEER as example. However, an explanation of the precise methods implemented in Pro/ENGINEER cannot be given because they are guarded as proprietary information. Note that (E4) and (E5) fundamentally affect the way the design has been conceptualized. These two editing functions are not usually supported in current systems.

Prior work on generic naming is usually limited to naming faces [10, 14]. Such a schema cannot distinguish other topological entities and therefore is

inadequate in supporting the full range of editing operations we consider. In [1] we have explained a naming schema that addresses these problems, but have not discussed the algorithms needed to support editing. Kripac reports a naming schema in [8]. His work also describes several ways to match generic names, in support of changing dimension values, but he does not further elaborate in which situation each matching algorithm should be applied. The feature editing operations of many feature-based design systems, such as [4, 13, 14], are limited primarily because of the restricted naming ability. Good techniques for generic naming and matching are therefore a prerequisite to flexible design variation.

2 Feature Editing

In our feature-based part design implementation, features are attached serially to *prior geometry* (also see [2, 3]). The prior geometry is the part obtained from the evaluated preceding feature definitions. Following [7], we distinguish *generated*, *modifying* and *datum* features. The construction of generated features is briefly described as follows. Details are in [2]:

1. A cross section is sketched and then fixed by constraints and dimension values.
2. The cross section is oriented and positioned in the three dimensional space, based on constraints, with respect to the prior geometry.
3. A proto feature, defined as the extrusion or revolution of the cross section to a sufficient extent, is constructed based on a set of feature attributes, such as *from a plane* and *to offset*. The proto feature is then trimmed by observing its interference with the existing geometry, again based on feature attributes.
4. The trimmed proto feature is attached to the prior geometry.

Modifying features add chamfers and rounds to edges, or draft angles to a set of faces. They are constructed from feature attributes and an identification of geometric elements in the prior geometry. Datum features include datum points, axes and planes, auxiliary elements that conveniently help construct generated and modifying features. Datum features do not change the part geometry; rather, their main purpose is to effect coordinate manipulation in simple terms.

2.1 Structure of Editing Operations

From the editing operations specified in Section 1, we extract common procedural steps from which they can be composed. This elucidates in turn how the name matching algorithms should be structured and how exact a match is required.

(E1) Feature Deletion and Insertion

Deletion of a feature F requires the following steps:

1. The geometry is rolled back to the prior geometry as if we were creating the feature to be deleted.
2. The feature definition of F is deleted from the unevaluated representation.
3. The subsequent features are reevaluated from the unevaluated representation. Due to match mediation, the unevaluated description of the subsequent features may change.

Insertion of a feature is similar, except that a new feature definition is inserted in Step 2. Thus, editing operation (E1) requires only a naming schema and mediation when matching names. Furthermore, we must account for the possibility that a later feature references geometric elements that were created before by the now deleted feature.

(E2, E3, E4) Changing Attributes, Modifying Dimension Values, Changing the Dimension Schema

When editing dimension values, the dimension schema, or the attributes of a generated feature F, the following steps are executed:

1. The part geometry is rolled back to the state of the prior geometry that existed when F was defined.
2. The cross section on which F is based is reconstructed in the required projection.
3. Attribute changes, dimensional changes, and changes in the constraint schema are defined by the editing process. The cross section is then regenerated for a generated feature.

4. The proto feature is constructed, and the attachment of F proceeds as if F was defined for the first time.

5. The subsequent features are reevaluated from the unevaluated representation. Due to match mediation, the unevaluated description of the subsequent features may change.

If F is a modifying feature or a datum feature, the work is simplified in that no cross section has to be modified and reevaluated. However, in that case the match mediation has to be more rigorous, as we will detail in later sections.

(E5) Changing the Cross Section

Finally, if the cross section is redrawn in part or in whole, a remapping step is required in addition, in which the new names of geometric elements of the cross section are mapped, where possible, to names of the old cross section.

2.2 Editing Examples

We demonstrate the editing concepts with a sequence of editing examples done with our Erep system. In Figure 5, we create a part with four features: a block, a protrusion, a slot and a round, created in this sequence. The cross sections of the protrusion and the slot are shown to the right.

The way in which the part has been created implies some *feature dependency.* The definition of the slot in the example of Figure 5 makes reference to the protrusion, for placement purposes. On the other hand, the round does not make a reference to the protrusion or the slot. So, if we delete the protrusion feature, the round remains while the slot feature would be deleted.

If the slot feature should be kept, we would have to modify the dimension schema so that the slot is independent of the protrusion. This may be done by editing the slot first: The system brings up the cross section of the slot as shown in Figure 6(left). We may now delete the distance constraint that places the slot with respect to the protrusion edge, and add a new distance constraint that references equivalently the top edge of the block; Figure 6(right). Reevaluation with the new dimensioning schema will not alter the design form, but the slot's definition no longer references the protrusion and so the protrusion could be deleted next — without affecting the slot.

Consider now the part of Figure 5 with the original dimensioning schema shown in Figure 7(left). To enlarge the radius of the protrusion, we edit by modifying the dimension values of the cross-section; Figure 7(top middle). After

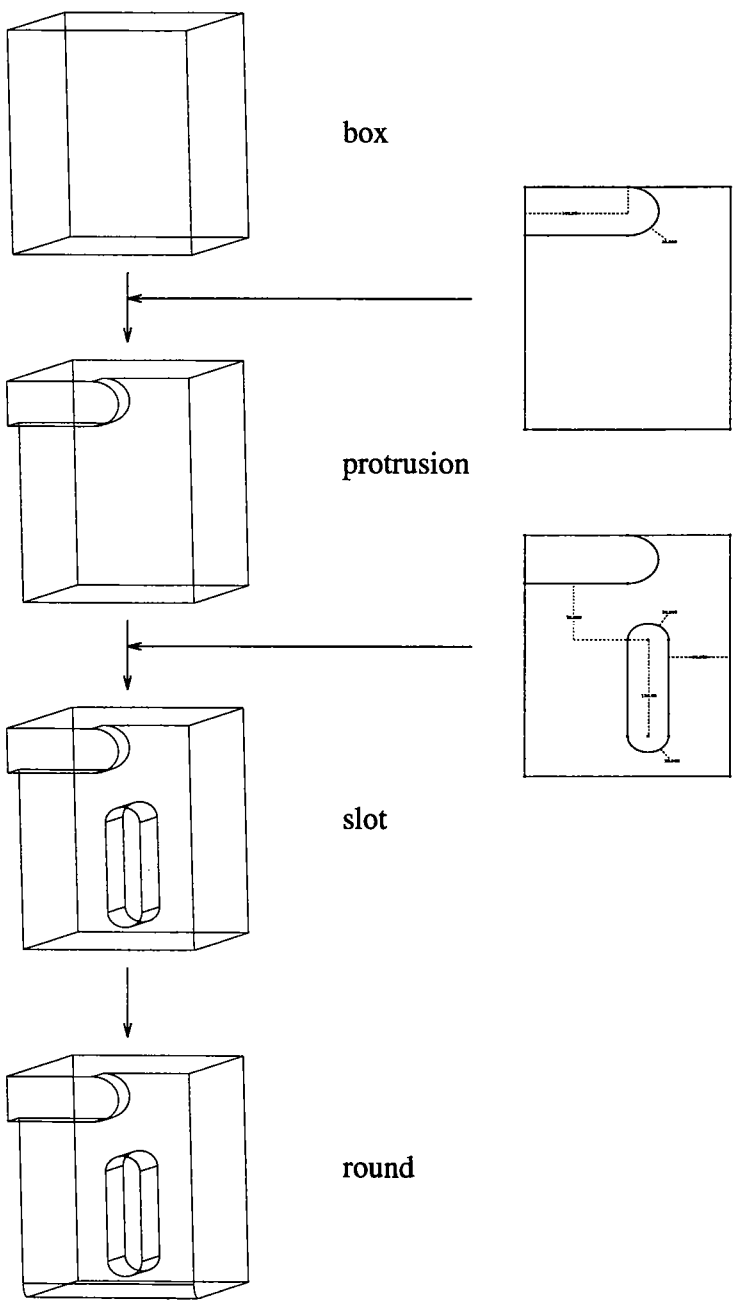

Figure 5: Erep feature construction: A part is created with a block, a protrusion, a slot and a round. The features were added sequentially. Cross sections are shown to the right.

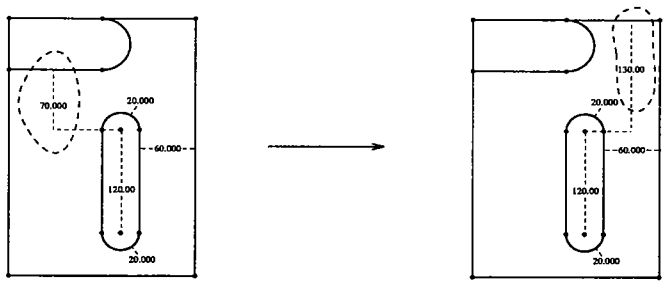

Figure 6: The slot dimensioning schema is edited. The new schema shown to the right no longer references the protrusion, so the slot position is now independent of the protrusion feature and its variations.

reevaluation, the part variant of Figure 7(top right) is produced, maintaining the constraints that specified the slot and its position relative to the protrusion. Note that the slot has moved. This would not be the case with the dimensioning schema of Figure 6(right).

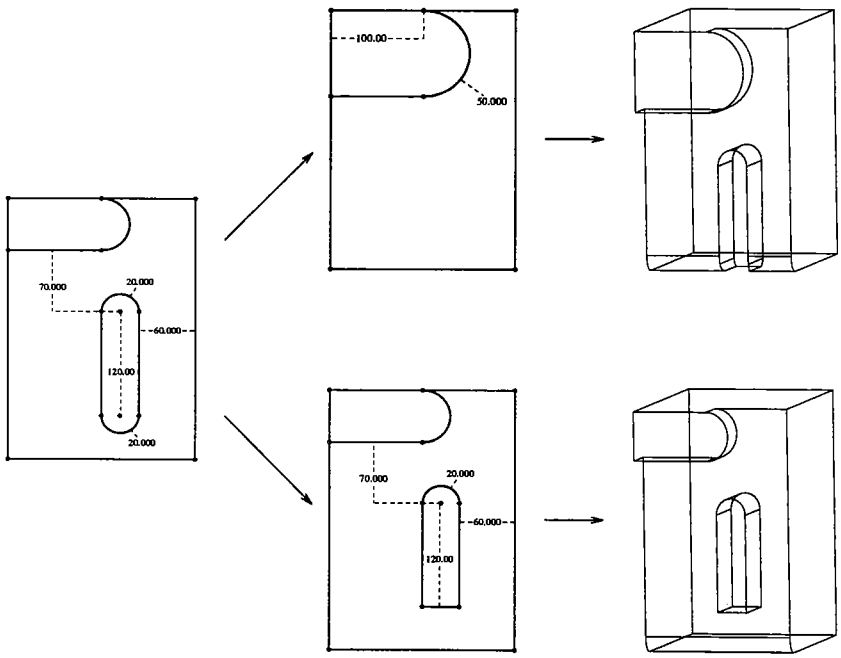

Figure 7: Top: Using the dimensioning schema left, editing the protrusion radius will affect the position of the slot feature.
Bottom: The cross section of the slot feature has been edited replacing one of the round ends with a square end.

In the bottom of Figure 7 we show a cross section change of the slot. The

round end of the slot has been replaced with a square end; Figure 7(bottom middle). The result is shown in Figure 7(bottom right).

The editing semantics that we have illustrated is intuitive for feature-based design. However, both research and commercial systems are limited in their support of the feature editability, due to the technical demands constraint-based design imposes.

3 Feature Reevaluation

Since features are created sequentially, later feature can depend only on earlier features. When a feature has been edited, all later features may be affected. Therefore, reevaluation has to account for the possibility that features that have not been edited directly may have to change nevertheless. The changes could be deleting a *marginalized* feature, or modifying the names referenced by the feature.

A feature becomes *marginalized* when it no longer contributes to the geometric shape of the solid at the moment of attachment; for example, when a hole is dimensioned such that it no longer subtracts volume from prior geometry. Such features are deleted. A feature becomes *orphaned* when the names used in defining the feature and attaching it refer to geometric elements that no longer exist. For instance, if the position of a hole depends on having a given distance from a boss and the boss is deleted, the hole can no longer be positioned. Orphaned feature can be reattached as described later.

A generated feature refers to elements of the existing geometry, using a generic naming schema as explained in [1]. In some cases, the editing change implies that the name of the referenced entity changes. In such cases, the referenced names must be adjusted by *feature correction.* The possibilities are summarized as follows:

Let

$$S = (F_1, F_2, ..., F_{i-1}, F_i, F_{i+1}, ..., F_n),$$

be the unevaluated representation, and let F_i' be the description of the modified feature for F_i. Then

$$S' = (F_1, F_2, ..., F_{i-1}, F_i', F_{i+1}', ..., F_m')$$

where $m \leq n$ and F_j', $i \leq j \leq m$, are the regenerated features following F_i.

The rewriting of the later features $F_{i+1}, F_{i+2}, ...$, and deletion of the marginalized features, limits the dependency of editing on prior design history.

A marginalized feature is easily detected: When reevaluating the feature, the feature attachment procedure detects that the prior geometry remains unchanged. In this case, the feature description is deleted. An orphaned feature arises when, in the course of reevaluation, the cross section is not fully determined or cannot be placed. Both situations can be due to unresolved names for dimensioning, but also to loss of the sketching plane for generated features, or loss of an entity for modifying features. Orphans can be reattached if we initiate feature editing and let the user supply new targets for the unresolved names.

In all cases, the crucial aspect is a matching algorithm for generic names that has to mediate newly arising ambiguities and has to recognize failure to match. These algorithms differ depending on the use of the name. For dimensioning, ambiguities are acceptable as long as the target entities project to the same point, line or plane. For sketching planes, multiple coplanar faces are acceptable. For modifying features, ambiguous references must be of the same type and are considered collectively. In all other cases, mediation and error recovery is needed.

4 Matching a Vertex

Vertices are matched to define constraints, to construct datum features, to be used by modifying features, or to be used as a subexpression in another name (also see [1]). In the dimensioning case, we need the vertex in projection, otherwise we need to identify the vertex in the three-dimensional space. Moreover, only in the case of modifying features can we accept multiple matched vertices with the same name.

A vertex can be lost when editing because of changes in a feature collision. An example is shown in Figure 8. In some cases, lost vertices can be uniquely reconstructed on basis of their names. However, we limit reconstruction so as to limit dependency on the design history. Moreover, the vertices used in modifying features should not be recovered from the design history because a modifying feature operates on the geometry after attaching its previous feature. Therefore, if the vertex cannot be found or if reconstruction is not desirable, we need to interact with the user to find a suitable alternate.

A vertex name certifies whether the vertex is a vertex of a proto feature. If so, the vertex can always be recovered from the proto feature, because its name, constructed by [1], identifies the vertex uniquely and the proto feature can be constructed explicitly. Thus, V in Figure 8 can be localized even after it has been obliterated in the design variant, or is lost as result of merging with a prior

vertex. In the former case, the name of the vertex is not changed in the feature correction, but in the latter case the name of the coincident, merged vertex is substituted.

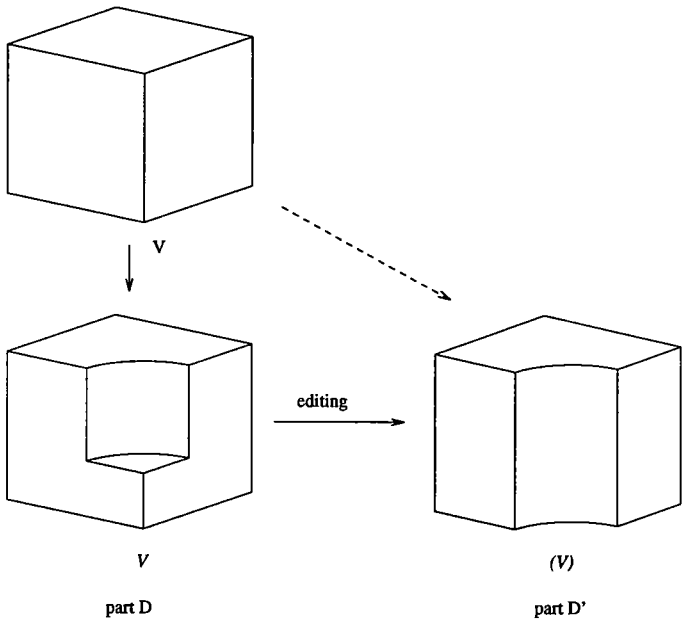

Figure 8: The vertex V of part D is not found in part D'. As a proto feature vertex, it can be reconstructed from the first feature of part D'.

In the case of an intersection vertex, there will be three or more incident faces that define the vertex. If no vertex can be found that matches the description, we construct an approximate match as follows by parsing the name expression generated in [1].

1. Let E_v be the vertex name we seek to match (see [1]). Let F be the feature used in the feature orientation part of E_v. We consider a set V of vertices that are intersection vertices and part of the feature F.

2. Let $J = \{f_1, f_2, ..., f_m\}$ be the incident faces listed in the first part of E_v. We partition V into subsets $V_1, V_2, ..., V_m$, where V_k consists of the vertices in V that have k incident faces in J.

3. We examine V_k, for $k = 1...m$. Each vertex w in V_k is assigned a grade G_w as follows: Initially, $G_w = 3k$. If w is manifold, and E_v requires a manifold vertex, then we add 1 to G_w. We add 2 if the order in which the incident faces are around the vertex agrees with the order of J for manifold vertices (note that $k \geq 3$ in that case). If w is nonmanifold and we require

a nonmanifold vertex, then 1 is added to G_w. For every expression $[f, s]$ in the feature orientation, we add 1 to G_w if the corresponding computation for w would also produce $[f, s]$ as feature orientation element.

4. The set of vertices approximately matching E_v is the set of vertices w with the maximum grade G_w. If $G_w < T$, where T is a preset threshold, then the match is assumed to have failed.

Assume that a set of vertices has the same maximum grade G_w and passes the threshold criterion. For dimensioning use, such a multiple match is acceptable when all matched vertices project to the same point. If this is not the case, the name E_w of every matched vertex can be computed. This partitions the match set into classes of vertices that are equivalent in the new design variant. The user is then asked to identify which equivalence class of the matched vertices is meant. The set so identified becomes the accepted match.

5 Matching an Edge

Edges are matched for defining constraints, constructing datum features and for being used by rounds and chamfers. In the case of constraints and datum features, ambiguous edges can be tolerated if they are collinear line segments with consistent orientation. In the case of 2D constraint definitions, ambiguities are acceptable if the edges project onto the same line, again with consistent orientation.

Edges of a generated feature have unique names. As in the vertex case, an obliterated edge can be reconstructed from the design history. However, note that such an edge may have been subdivided, and only some of the resulting segments may be meant.

We structure the matching algorithm by parsing the naming expression of an edge in [1]. First, a preliminary set of candidate edges is identified, based on matching adjacent faces. Unless two or more adjacent faces can be matched, the edge cannot be matched and user intervention is required.

Next, the preliminary set of edges is narrowed using the incident vertex specification and feature orientation. If the vertices can be identified, then we can also determine which subdivision segments are to be matched. If no vertex information is given (for closed edges), a subdivision test may still be applicable. Note that this step is not needed for an edge used in constraint definitions. Edge names are further analyzed based on feature information, especially when the vertex match is unsuccessful or ambiguous.

Preliminary Edge Set

A preliminary edge set is found using immediate context. Let E_e be the edge name we are to match. If the edge is (a subdivision of) an edge of a proto feature, then the preliminary set consists of all edges that are adjacent to the faces of the proto feature. Such edges must lie on a common space curve that can always be reconstructed. Note that the set may be empty.

If we are matching intersection edges, the preliminary edge set consists of all edges that have a maximum number of adjacent faces that lie in the cycle of faces C_e identified by the first part of the edge expression E_e. At least two adjacent faces must be matched, otherwise the preliminary set is empty.

If the preliminary edge set is empty the match has failed. However, even if the preliminary edge set is a singleton, the match is not successful unless this edge has the equivalent description of adjacent vertices (for a modifying feature), and feature orientation information to E_e. For example, consider the part in Figure 9 left. Assume that we created the round slot by an extrusion from the front to the rear, and rounded the right edge of the slot. By changing dimension values, the slot can be repositioned so that it becomes a step. The middle variant correctly recognizes that the singleton preliminary edge set does not constitute a match on basis of the incident vertices or feature orientation information, whereas the right design variant is an error.

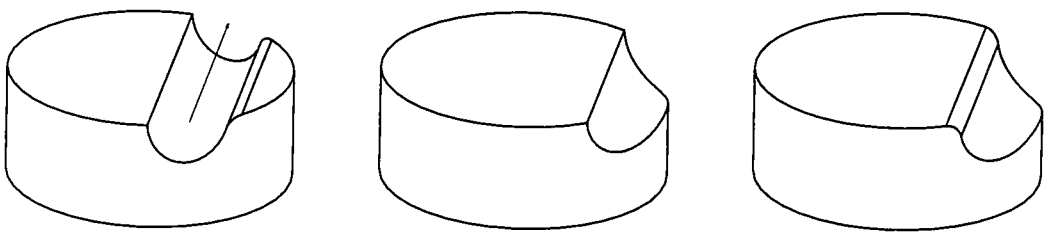

Figure 9: After repositioning the slot, the correct design variant is shown in the middle.

Incidence Based Narrowing

Preliminary edge identification does not account for the possibilities that an edge has been subdivided, or that two different edges have the same immediate context. Examples are shown in Figures 9 and 10. An effective distinction between the two types in mathematical terms is complicated, because the distinction should conform with user intuition and then must be based on dividing space curves into real components, a complicated undertaking. For instance,

the two near-circular edges of a cylindrical through hole in a sphere would be considered two distinct edges with the same immediate context, even when the axis of the hole does not contain the center of the sphere. However, classical algebraic geometry would consider both curves part of a single space curve connected in complex projective three-space. Therefore, we do not differentiate between the two cases.

The edge name E_e has the form $[C_e, L_e, F_e]$ (see [1]). The expression L_e identifies the incident vertices and is now matched. If $L_e = []$, the edge was closed when the name was constructed. In this case, no vertex matching is possible and we continue with feature orientation based narrowing.

If $L_e = [V, W, s]$, then we match the vertices described by V and by W. If the vertices can be matched uniquely, or can be uniquely reconstructed, or if one edge and a direction can be established on the basis of s and the immediate context specification, then the matching process continues with a subdivision identification. Otherwise, we continue with feature orientation based narrowing.

If we have uniquely identified or reconstructed two vertices v and w described in E_e, then we retain all edge segments that lie between the two vertices. See also Figure 10. From this reduced edge set, every candidate edge is deleted that has an inconsistent feature orientation.

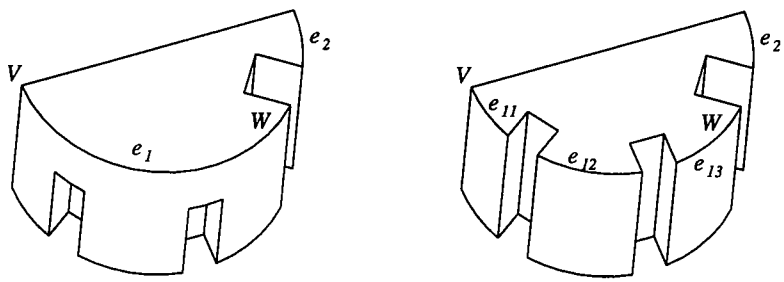

Figure 10: Left: Edge e_1 is named in the design. After editing, the edge is subdivided by colliding features. Segments e_{11}, e_{12}, and e_{13} match.

If we can match only one vertex, say V, from the name E_e, then all edges not incident to V are removed from further consideration. This implements the semantics discussed in conjunction with Figure 2. The remaining edges must have consistent feature orientation to constitute a match.

Feature Orientation Based Narrowing

In general, an edge E_w is considered an exact match if the edge is in the preliminary edge set and has two vertices that are uniquely matched from the

description in E_e. However, there are other cases requiring further analysis, for example edges of which one or no adjacent vertex has been matched and closed edges that do not have adjacent vertices. We narrow as follows.

Examine every candidate edge e'. Let $E_e = [C_e, L_e, F_e]$, and let F be the feature name occurring in $F_e = [F, [f_1, s_1], ..., [f_k, s_k]]$. We require that at least one of the faces $f_1, ..., f_k$ be adjacent to e'. Moreover, for every subfield $[f, s]$ where f is adjacent to e' we require that the orientation s agree with the orientation of e'. Only edges satisfying these two criteria remain in the match set.

Grading Multiple Matches

If the narrowing does not produce a unique match, we give each remaining candidate edge e' a grade based on how closely the vertices of e' match the vertex descriptions V and W in E_e. This is done by evaluating the vertices of e' as described in Section 4. The edges that have the highest grade are the approximate match of E_e.

Match Mediation

When the matching algorithm has failed, then the user is asked to reidentify the edge. A new edge name is computed, and regeneration continues.

When matching has produced several edges, additional information may be needed depending on the purpose of the match. In the case of rounds and chamfers, multiple matches are always accepted. All edges matched are used in the feature operation. The names of the matched edges are recomputed from the current design variant and are stored for future reference. In the case of 2D dimensioning, multiple matches are acceptable only when every matched edge projects to the same circle or line with identical orientation. Otherwise, the user is asked to differentiate which edge is to be referred to. In the case of datum plane definitions, multiple matches are accepted when the edges are collinear line segments. For datum axis definitions, the collinear line segments must be consistently oriented.

6 Matching a Face

Faces are matched to determine the limits of feature attachment operations, defining draft operations, identifying a sketching plane, or constraining a datum definition. The degree of exactness with which a face is to be matched varies

with the operation. For faces used in defining constraints and sketching planes, the match only requires matching the internal geometry of a named face, not a particular subdivision. As before, an obliterated face used in those cases can always be reconstructed from the design history, from the corresponding proto feature or modifying feature. In the other cases, however, a more specific match is required.

All faces of a solid must be subdivisions of faces created in proto features and modifying features, with the possible exception that two faces may be merged so that one of them loses its identity. Faces on the proto feature of a generated feature have unique names that are assigned when the feature is created. Faces of chamfers and rounds are named after the vertices and edges they chamfer or round.

When a face has been subdivided, the internal geometry of this face does not change, therefore all subfaces inherit the same name of their parent face. When faces are merged, every face in the merging list can be the match representative. However, because of feature editing, different face representatives may lead to different interpretations. In Figure 11, suppose the merged face of f_1 and f_2 are initially used for the sketching plane of the third feature. After the second feature is edited, using f_1 and f_2 as the representatives of the merged face leads to protrusions with different heights.

When merging faces, there is no clear preference of how to name the resulting face; either name could be used. In our implementation, the result has the name of the face that belongs to the earliest feature among all merging components.

Preliminary Set

As explained in [1], every face name starts with one of the following five expressions:

> (`start_f feature.f2D`), (`end_f feature.f2D`), (`side_f feature.f2D`), (`r`(l_e)) or (`c`(l_e))

Here, faces of type `c`(l_e) are from chamfers and faces of type `r`(l_e) are from rounds. These expressions are used to select the preliminary set of faces. If only the geometry of a face is required, the faces in this set will typically lie on the same surface, and so any face in the set can serve as a match. Moreover, a face can be reconstructed if the match set is empty. If faces in the preliminary match set have different underlying surfaces, for instance when the set contains chamfers of several edges, then further narrowing is needed.

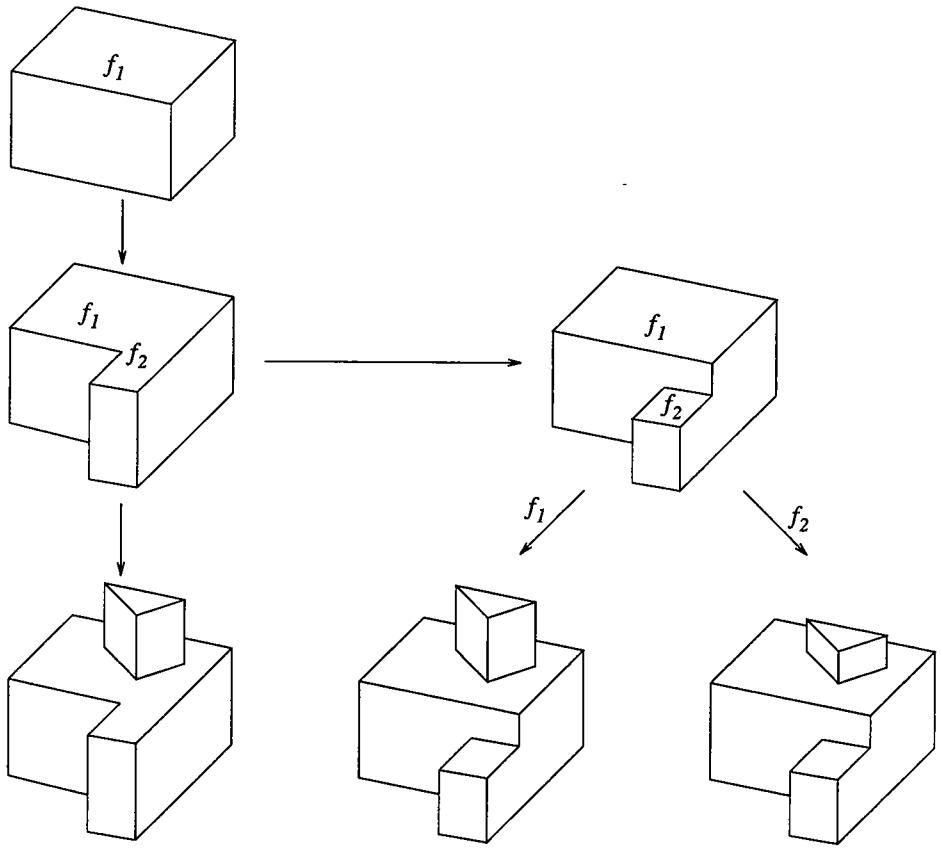

Figure 11: Different representatives of a merged face produce different results after feature editing.

Narrowing Multiple Matches

When a specific subdivision of a face is required, we process the adjacent edges and compare them to the required edges. Edge matching proceeds for each individual bounding edge as described before in Section 5. The grade for each candidate face in the preliminary matching set is the number of matched boundary edges. This number has to exceed a threshold which is set as a fraction of all adjacent edges. As an example, assume that a face is to have 5 adjacent edges with prescribed names. We match each edge name against the edges of a candidate face. With a threshold set at 3/4, the match succeeds if at least four out of the five edges can be matched, and fails if three or fewer edges are matched.

7 Conclusions

The matching mechanism is an inverse processes of generic naming. Naming and matching are prerequisites for flexible, constraint-based feature editing. Without the ability to do graded matches, however, only exact unambiguous matches would be allowed, and we would seriously restrict the flexibility of feature editing. Therefore, we have introduced rankings of partial matches by assigning numerical grades.

The support of feature editing is only one application of our naming and matching techniques. The fundamental significance of these techniques is that a separation is achieved between generic design and modeler-specific design. The naming and matching systems are key components of the design compiler that translates the generic design that is modeler-independent to a specific model instance that is constructed with the infrastructure of a particular geometric modeler[3].

Our naming and matching algorithms are a contribution to defining a modeler-independent semantics of feature-based editing, just as our earlier paper [2] gave a semantics for feature attachment. Where possible, our semantics depends on the conceptualization that the result of a changed dimension ought to be derived by considering it a "deformation" process governed by gradually varying the dimension from the initial to the final value.

In our implementation of the editing semantics, we have concentrated on this semantics aspect, experimenting with the mechanisms described here and in the companion paper [1]. Thus, no attempt has been made to devise efficient searching algorithms for locating, in the Brep instance, which boundary element is referred to by a particular name. Thus, finding the possible referenced entities takes linear time in the number of boundary elements. Significant heuristic and asymptotic improvements are easy to make. For instance, when a design is evaluated, searching structures can be devised that index subsets of boundary elements by the feature(s) they belong to. This immediately results in sublinear pe4rformance, and coupled with efficient searching techniques leads to very attractive speed-ups. The details are routine.

References

[1] V. Capoyleas, X. Chen, and C. M. Hoffmann. Generic naming in generative, constraint-based design. Technical Report 94-011, Purdue University, Computer Science, 1994.

[2] X. Chen and C. Hoffmann. Towards feature attachment. *Computer Aided Design*, page to appear, 1994.

[3] X. Chen and C. Hoffmann. Design compilation for feature-based and constraint-based CAD. In *Proc 3rd ACM Symp on Solid Modeling*, 1995.

[4] J. C. Chuang, D. R. Patel, and M. K. Cook, R. L. Simmons. Feature-based modeling for mechanical design. *Computers & Graphics*, 14(2):189–199, 1990.

[5] C. M. Hoffmann. On the semantics of generative geometry representations. In *Proc. 19th ASME Design Automation Conference*, pages 411–420, 1993. Vol. 2.

[6] C. M. Hoffmann. Semantic problems in generative, constraint-based design. In *Parametric and Variational Design*, pages 37–46. Teubner Verlag, 1994.

[7] C. M. Hoffmann and R. Juan. Erep, an editable, high-level representation for geometric design and analysis. In P. Wilson, M. Wozny, and M. Pratt, editors, *Geometric and Product Modeling*, pages 129–164. North Holland, 1993.

[8] J. Kripac. *Topological ID system - A Mechanism for Persistently Naming Topological Entities in History-based Parametric Solid Models.* PhD thesis, Czech Technical University, Prague, 1993.

[9] M. J. Pratt. Synthesis of an optimal approach to form feature modeling. In *ASME Computers in Engineering*, volume 1, pages 263–274, 1988.

[10] A. A. Requicha and S. C. Chan. Representation of geometric features, tolerances and attributes in solid modelers based on constructive geometry. *IEEE CG & A*, 2(3):156–166, 1986.

[11] J.R. Rossignac. Issues in feature-based editing and interrogation of solid models. *Computers and Graphics*, 14:149–172, 1990.

[12] J.R. Rossignac, P. Borrel, and L.R. Nackman. Interactive design with sequences of parameterized transformations. In V. Akman, P. ten Hagen, and P. Veerkamp, editors, *Intelligent CAD Systems 2: Implemental Issues*, pages 93–125. Springer Verlag, 1989.

[13] J. J. Shah, M. T. Rogers, P. C. Sreevalson, D. W. Hsiao, A. Mattew, A. Bhatnagar, B. B. Liou, and D. W. Miller. The A.S.U. features testbed: an overview. In *ASME Computers in Engineering*, volume 1, pages 233–241, 1990.

[14] G. P. Turner and D. C. Anderson. An object-oriented approach to interactive, feature-based design for quick turnaround manufacturing. *ASME Computers in Engineering*, 1:551–555, 1988.