1992

Erep An Editable High-Level Representation for Geometric Design and Analysis

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

Robert Juan

Report Number:
92-055

EREP AN EDITABLE HIGH-LEVEL REPRESENTATION FOR GEOMETRIC DESIGN AND ANALYSIS

Christoph M. Hoffmann
Robert Juan

CSD-TR-92-055
August 1992

Erep

An Editable High-Level Representation for Geometric Design and Analysis

Christoph M. Hoffmann*
Department of Computer Sciences
Purdue University

Robert Juan†
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya.

Working Paper, August 1992

Abstract

We propose a high-level, generative, textual representation for feature-based solid modeling, which we call Erep. We argue that such a representation should be independent of an underlying core solid modeler, and give some criteria it should satisfy. Such an Erep allows archiving geometric designs in a form that is both editable and translatable to any solid modeling system. Furthermore, the representation serves as a global schema by which to federate different modeling systems, and is extensible in a natural way to a representation from which to derive analysis representations and process plans. By federating with finite-element analysis packages, in particular, our approach offers closing the design-analysis feedback loop that previously required a manual link.

*Supported in part by ONR Contract N00014-90-J-1599, NSF Grant CCR 86-19817, and NSF Grant ECD 88-03017.

†While on leave in the Department of Computer Sciences, Purdue University. Partially supported by a NATO Scientific Programme fellowship and by a fellowship of the Spanish Ministerio de Educación y Ciencia

Contents

1 Introduction **3**
- 1.1 Geometry Representations . . . 3
- 1.2 Feature-Based Design . . . 4
- 1.3 Editable Representations . . . 4

2 General Requirements on the Representation **6**
- 2.1 Archivability . . . 6
- 2.2 Fidelity . . . 7
- 2.3 Constraints . . . 7
- 2.4 Definition . . . 9

3 Architecture of the Modeling System **9**

4 Language Concepts **11**
- 4.1 Geometric Language Constructs . . . 13
  - 4.1.1 Feature Construction . . . 14
  - 4.1.2 2-D Geometries . . . 15
- 4.2 Referential Constructs . . . 16
- 4.3 Constraints and Dimension Constructs . . . 16
- 4.4 Editing Issues . . . 17

5 Implementation Issues **18**

6 Analysis and Process Planning Representations **20**
- 6.1 Analysis Representations . . . 20
- 6.2 Process Planning Representations . . . 22

A Erep Example Language Specification **26**
- A.1 Assemblies . . . 26
- A.2 Parts and Features . . . 27
- A.3 Datums . . . 28
- A.4 Generated Features . . . 29
  - A.4.1 Extruded Features . . . 29
  - A.4.2 Revolved Features . . . 31
  - A.4.3 General Sweeps . . . 32
- A.5 Section Specifications . . . 34
- A.6 Patterns . . . 37
- A.7 Modifying Features . . . 37

B Orientation Rules for Axes and Planes **39**

1 Introduction

1.1 Geometry Representations

There is a rich variety of representation schemas for storing and manipulating geometric shapes and especially, solid models. Over the years, the primary family of schemas that has emerged as the most common one used in commercial and in research modeling systems is the *boundary representation* (Brep) in its many variants, for representing manifold and nonmanifold solids; e.g., [15]. In a Brep schema, the surface of a shape is represented as a collection of faces, edges and vertices, along with data structures that record adjacencies and incidences of them. The geometry of the faces, and possibly edges, is usually represented by parametric surfaces and curves, such as NURBS, Bézier patches, and so on; e.g., [2, 12]. Faces are then areas defined by a standard domain or by trimming edges. Other face geometries in use are implicit and procedural representations of the containing surfaces, [15, 22].

This style of representation originally competed with *Constructive Solid Geometry* (CSG); [20]. In CSG, solids are represented as algebraic expressions formed from a family of primitives, using the Boolean operations of (regularized) set union, intersection, and difference, as well as rigid body motions that specify the relative spatial positions of the operands. The classical set of primitives consisted of blocks, cylinders, spheres, cones, and sometimes tori. Specific instances of the primitives were instantiated by choosing values for edge length, radius, etc.

From the point of view of editing shapes, it is probably fair to say that the CSG representation exists at a higher level of abstraction in that it manipulates shapes represented in a purely symbolic form — without explicit reference to how to evaluate, for instance, whether a given point is on the surface of a primitive. Thus, in principle, one could choose to evaluate the CSG in any one of a variety of more explicit shape representations, converting it, for instance, to a Brep whose faces could be NURBS or even faceted. Nonetheless, Breps have gained wider acceptance whereas the use of CSG modelers has declined. The factors contributing to this probably include the ability of Breps to use splines in ways that allow local shape manipulations utilizing the full power and range of the methods developed by Computer-Aided Geometric Design (CAGD). Moreover, to-date there are no general algorithms for converting a Brep to an equivalent CSG representation, [10, 16, 24], and such algorithms would seem to require implicit algebraic surfaces.

Despite the ultimate lack of penetration, CSG has influenced deeply the way in which most modeling systems conceptualize and present to the user the various geometric operations and queries they implement. Thus, in a broader sense, we could say that CSG refers to building geometric shapes from a set of primitive solid shapes, however they be defined, using the Boolean operations

and rigid body motions. In this sense, the Bath modeler is a CSG modeler, [3], and CSG refers to high-level shape operations that are used widely.

1.2 Feature-Based Design

Neither Brep nor CSG representations satisfy by themselves the designer's needs well. The Brep describes solids at a very low level of abstraction. As a result, designers are forced to think from the very begining of the design process about characteristics that are relevant only in the last stages of design. Although CSG describes a solid at a higher level of abstraction than a Brep, it is still far from the designer's vocabulary; in many cases a CSG operation on the model does not correspond to a functional objective the designer has in mind.

From the designer's point of view, it seems more appropiate to construct a model in terms of functional elements, each of them having a particular significance to the design. This can be achieved by enabling the designer to express the design in terms of features like slots, holes, bosses, etc. The features may correspond to functional elements in the final product and can be categorized by type [30], and can be represented conveniently by a few parameters. Additional information for automatic process planning, NC programming and inspection can be either derived algorithmically or added to the features definition. This approach is known as *feature-based design*, and has received much attention during the last decade [21, 19, 27, 28, 31].

With the emergence of feature-based modelers, new concepts for defining and editing shapes and solids are coming into use. These new operations and manipulations are not as readily coded by a formal representation schema for reasons to be explained later. The importance of feature-based concepts is rooted in the convenience and efficiency of use, of geometric modeling systems, they make possible. These new advantages will surely result in broader use of geometric modeling systems in research and industry.

1.3 Editable Representations

Since the use of modeling systems in industry encourages electronically representing designs in a manner that is conducive to archiving and reuse, and since the design of new products and parts is often an evolutionary change of existing designs, it would seem necessary to devise geometric shape representations that permit more than archiving the completed detailed shape design and then interfacing that representation with analysis and production tools. Rather, one would like formal electronic representations that also allow one to modify an archived design and edit its features, or archive and reuse partial designs.

The subject of this paper is to analyze the nature of such editable geometry representations and to propose and justify basic criteria for judging their utility. An example design for an editable shape representation is also sketched, and

the manner in which it interacts with both the user interface and the underlying core modeling system is presented.

Because a feature-based design interface is conducive to many important advantages, including efficiency of use, it is tempting to compare the accompanying editable shape representation with a program written in a high-level programming language and the native shape representation of the underlying core modeling system as an equivalent machine-level program. This comparison seems especially appropriate in view of the fact that the editable representation should be parametric or even variational, so that it does not so much store the geometry itself but rather a set of rules that specify how to construct the geometry as a function of parameters and constraints. The agent that interprets the editable representation would then be a kind of geometry compiler, or interpreter, that translates the elements of the editable shape representation through a suitable sequence of geometric operations carried out by the underlying modeler, into a traditional Brep.

Although these parallels are very suggestive, they are not quite precise. In particular, the process of constructing and elaborating an editable representation depends on an interacting flow of information between the geometry interpreter, the graphical user gestures made through the interface itself, and the underlying geometric core modeler. It is this interaction that interferes with a strict segregation of the high-level and the low level representations and limits the analogy with programming in the traditional sense. However, we argue later that these limits are not overly confining.

The parallel between high-level and low-level programming languages is suggestive of another possibility afforded by our approach. Since the user does not really care about the representational details of the core modeling system — as long as the representation serves the purpose the user has in mind — one should contemplate translating the editable shape representation into an analysis representation, such as a suitable complex of tetrahedra or other mesh elements. Geometric research modeling systems exist today that use such an internal representation [14, 13], and suitably approached, our editable representation can be translated into tetrahedral meshes, whereupon they are easily interfaced with analysis codes. Doing so promises a convenient way of closing the design-optimization loop which currently requires an explicit translation from a geometric representation into an analysis representation, followed by the analysis, and subsequently, by human interpretation of the results to derive what consequences, if any, the analytical properties of the design have on the design details. Other possibilities include integrating process planning and federating different modeling systems. We also discuss these points in some detail.

2 General Requirements on the Representation

The natural mode of specifying shape is visual. Shape is efficiently perceived visually and is effectively communicated with a sketch or a more detailed rendering, often annotated with dimensions and constraints. Of course, such drawings can be represented, when completed, in the traditional way using a Brep. However, the resulting collection of faces, edges and vertices does not reflect the conceptual structural information implied by the method of construction. For example, if we create a generalized cylinder by sketching a cross section and revolving it, the final Brep will represent neither the rotational symmetry we specified by this construction, nor the axis of rotation in the sketch. Furthermore, the Brep would not represent a dimensioning scheme that we might have used to finalize the cross section and that could contain important dimensioning constraints and interdependencies.

Summarizing, the structural geometric information of the features and the interdependence of their design parameters is irretrievably lost in the Brep and this loss prevents convenient redesign. In consequence, a higher-level representation is needed. This representation should satisfy the following requirements:

1. *Archivability:*
   The representation should be suitable for archival and transmission, and should be independent of the underlying core modeling system.

2. *Fidelity:*
   The representation should record the sketches and conceptual construction steps by which the shape has been defined.

3. *Constraints:*
   The representation should support parametric and variational design, as well as the dimensioning scheme the designer has chosen. Both generic and specific design must be possible.

We argue the value of these requirements.

2.1 Archivability

A good design is a valuable asset and the potential basis of a family of future designs. It should be archived in a machine-readable form in a manner that interfaces conveniently with other electronic tools and programs. For example, in manufacturing we would submit the electronic design to analysis tools that assess manufacturability and functional performance. The current situation is, however, that editable representations exist for some modeling systems, but they are low-level, modeler-specific data records that cannot be exchanged with foreign modeling systems. In consequence, the archived design can only be processed with the modeling system that created it.

In some cases, newer releases of the same modeling system cannot process stored designs prepared by older system versions, be it because the stored representation refers to internal data structures that have become obsolete, or because changed tolerances in the newer system version no longer support geometric constructions whose interpretation succeeded before.[1] Moreover, the specificity of the proprietary design representation prohibits translating these representations by machine to ones understood by other tools performing, e.g., an analysis of physical properties of the design. In critical applications it is not uncommon that a design must be manually converted to another data format by a specialist.

2.2 Fidelity

The way a geometric design has been created defines a geometric structure that is often meaningful beyond the purpose of constructing the shape using the geometric operations available. For example, the designer may have chosen to create a feature in a certain way in order to facilitate a manufacturing process to be used for realizing the design. Moreover, the sequence of design operations, and their very nature, often has implications on the shape of variant designs that are obtained by changing certain relationships or dimensions. When editing a design, this structure also provides the basic handles for possible modifications and editing operations. Therefore, this editable representation should foremostly reflect the structure the designer has created. If additional structure can be inferred, it ought to be recorded as well. For instance, we may have created a cylinder by sweeping a circle (perpendicularly) along a line segment. In this case, we could infer the rotational symmetry of the shape and should record its axis which is the path of the circle's center.

2.3 Constraints

The power of a feature-based design is substantially increased by parametric and variational designs. In a parametric design, a sequence of constraints is defined and solved serially. For example, if the constraints concern dimensions d_k, the next constraint on d_{k+1} is an explicit function of the previous constraints, i.e.,

$$d_{k+1} = f(d_1, ..., d_k)$$

When solving parametric constraints, the constraints are simply evaluated in sequence.

In variational design, no explicit sequencing of constraints is provided, and

[1] This observation was made about Pro/Engineer by one of their experts. We have not systematically explored how wide-spread this problem might be in other commercial modeling systems.

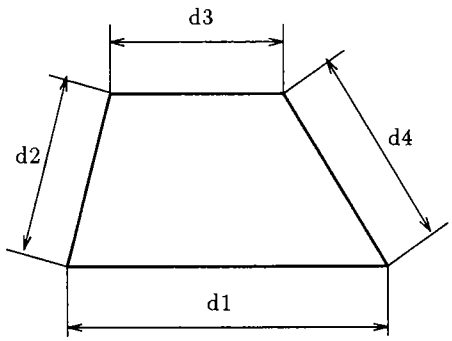

Figure 1: Variational constraints must be solved.

a system of simultaneous equations must be solved, i.e.,

$$\begin{aligned} f_1(d_1, ..., d_n) &= 0 \\ f_2(d_1, ..., d_n) &= 0 \\ &\vdots \\ f_r(d_1, ..., d_n) &= 0 \end{aligned}$$

Without addressing whether an interactive design environment can reasonably support a full variational constraint mechanism, we note that the editable representation has to account for both mechanisms. Consider for instance the trapezoid sketched in Figure 1. To determine the true shape, a variational system must be solved because of the chosen dimensioning scheme. A different dimensioning scheme, for example the one shown in Figure 2, may allow a parametric solution.

In addition to parametric and variational algebraic constraints, there are geometric constraints that express spatial interrelationship symbolically. For example, lines may be parallel or perpendicular, circles concentric, two geometric entities may be incident, and so on. Such constraints need to be recorded verbatim as their translation into equivalent algebraic form may lose information that is important in editing the design.

An important class of constraints can be used to place features with respect to each other in space in a manner that eliminates explicitly relating intrinsic coordinate systems to each other. Of course, there is no reason to eliminate explicit coordinate transformations from the editable representation. But the constraint-based concept is important in the user interface and can also represent structure needed in some editing operations. By the requirement of fidelity, this form of geometric constraint ought to be supported by the representation.

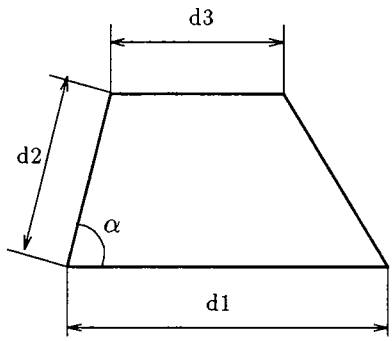

Figure 2: Parametric constraints can be solved.

2.4 Definition

We call a representation supporting the minimal requirements of archivability, fidelity, and constraints an *Editable representation* (Erep). We discuss in the following the general structure of Ereps and clarify our ideas with a specific example presented in the appendices. We will design a textual Erep, and insist on a strongly symbolic, textual form. However, we do not intend the Erep to be a programming language for design, and would not ask the designer to write Erep text. Doing so would be a mistake because of the loss of the visual element that is so important in shape specification. Rather, the Erep is accumulated under program control as the designer interacts with the graphical user interface (UI), and is edited under program control in response to shape changes or elaborations initatiated graphically from the UI.

The need for and potential benefits of so raising the level of abstraction in geometric design have been articulated before; e.g., [8, 9]. However, there seems to be no clear recognition that this higher level of abstraction ought to be supported by a formal representation, and that this representation should be independent of the subsystems to which it is ultimately translated.

3 Architecture of the Modeling System

Bacause of the strong interaction between the user interface, the Erep, and the core modeling system, we describe first a general architecture for a feature-based modeling system employing Ereps. Traditionally, the UI is a presentation of the functionalities and capabilities of the underlying core modeler. It is better, however, to target the user interface to a broad family of applications so as to achieve stability under technological advances in the underlying modeler. Typically, an application area has developed an established vocabulary in which it expresses

design, and only the emergence of new processes such as layered material deposition may affect this vocabulary profoundly. In contrast, the capabilities and functionalities of core modeling systems change rapidly as continuing research develops new algorithms and techniques, both in regards to topology and geometry.

The overall architecture of a feature-based modeler is shown in Figure 3. The

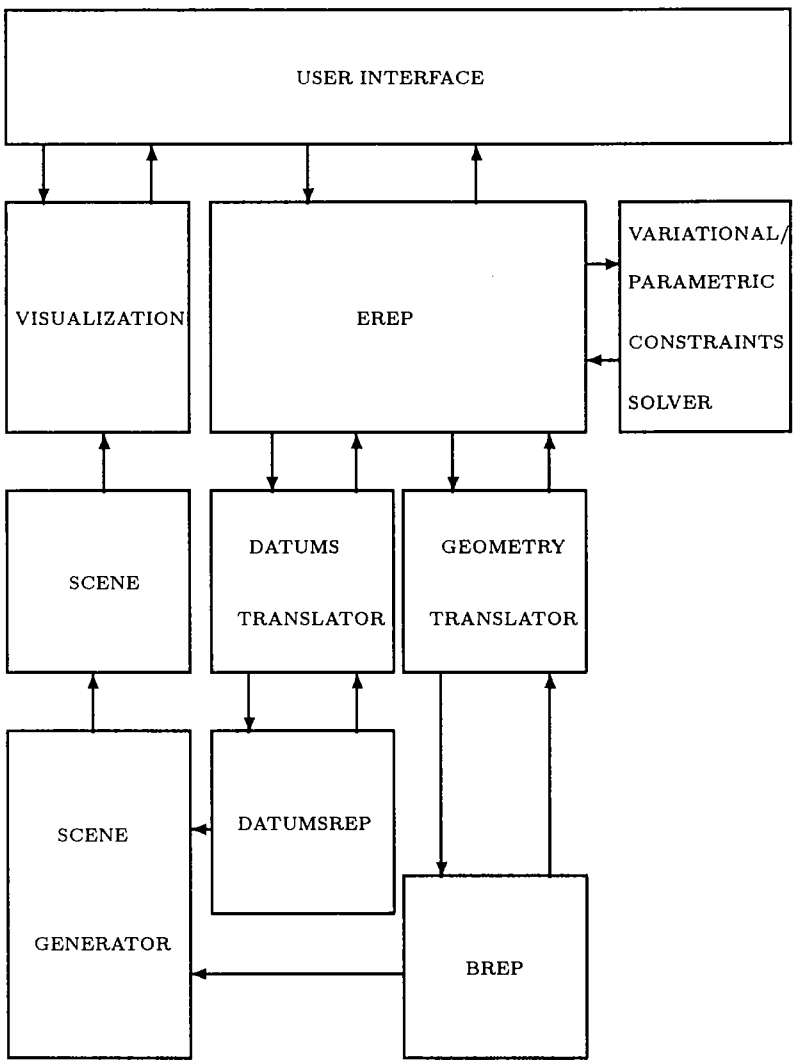

Figure 3: Architecture of the Modeling System

primary function of the geometry translator is to construct the detailed, specific geometry representation when needed. Note that the user interface must access

functions of the core system in order to present the user with the necessary views of the progressing shape design. Therefore, the information flow between the UI and the core modeling system must be standardized in order to achieve independence from the particulars of the core modeler. Since the core modeling system should not be expected to understand the style of interaction the user interface defines, we must translate the geometric features the user has identified by graphical selection, from the core representation, where they are determined, to the feature-based representation of the higher level that the designer works with. This is accomplished by means of a mapping that associates high-level features with low-level face groupings. This representation map is maintained by the geometry translator.

The representation map varies with the core modeler's representation. If the core modeler uses a Brep, the mapping is typically one that associates groups of faces in the Brep with certain points or lines of the Erep, or even with regions in sketching planes. If the core modeler uses an analysis representation, the mapping would additionally associate tetrahedra with logical feature volumes known in the Erep.

In the diagram, datums are handled by a separate translation. Typically, the datum modeling support is done by a surface modeler, whereas the geometry modeling support is done by a solids modeler.

The system capabilities are substantially leveraged when federating different modeling systems, each supporting different groups of functions of the UI. Federating different, separately developed systems requires a uniform global view of the user interface, and the Erep is ideally suited to provide such a global view. The separate information path for datums is a simple example of federating a surface and a solids modeler. But much more is possible. As shown in Figure 4, support for engineering analysis and for process planning can be incorporated by federating it in the same way. This allows in particular resolving geometry detail differently in the analysis and the geometry model. In section 6, we discuss these ideas in more detail.

4 Language Concepts

A detailed design of an example Erep is given in Appendix A. In this section we restrict the discussion to the rationale behind several language concepts.

It is important to recall that the Erep must completely and unambiguously specify the geometry without unduely referencing any particular method of implementing the geometry in this or that core modeling system. At the same time, the Erep is not constructed by the designer. Instead, it is accumulated as the result of graphical modeling and editing operations that the user executes in the user-interface. Since some of the gestures require visualization which, in turn, must be provided based on the explicit geometry representation of the

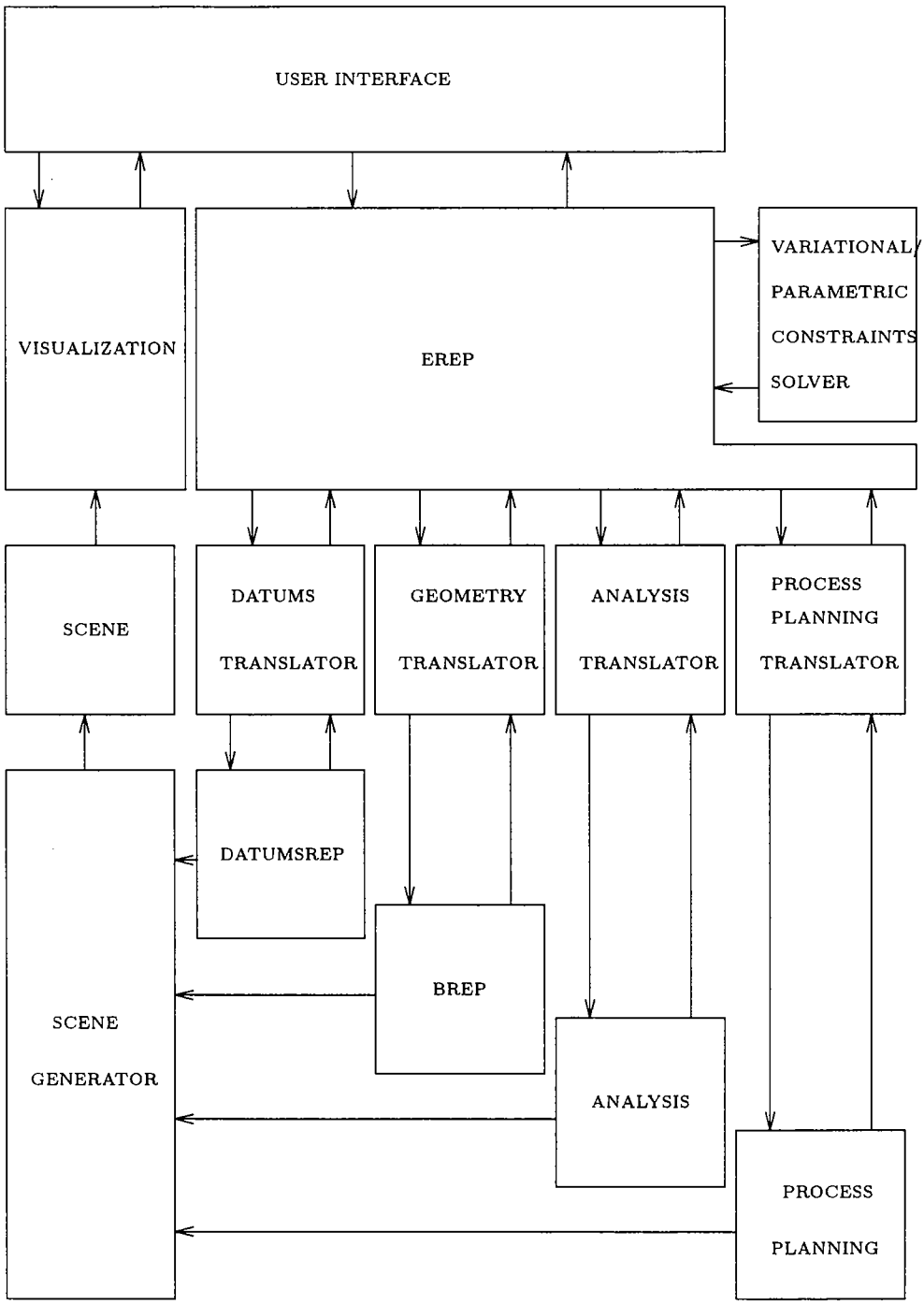

Figure 4: Federated Modeling Architecture

core modeler, the information flow between the UI, the geometry translator, and the modeling system must be standardized, and an independent naming scheme introduced at the Erep level, so that every "picked" geometry item can be recorded textually without reference to the graphics transforms and core data structures from which it was identified. This is accomplished by introducing an internal naming scheme that names faces, edges and vertices of the features. In particular, this leads to the concept of *logical* and *physical* face. For instance, the cylindrical surface of an extruded circle is a single logical face, independent of the diameter of the circle, but may well be mapped to several physical faces constructed by the core modeler in the Brep.

The Erep can be structured into several linguistic groupings. Broadly speaking, there is a *geometric part* that deals with the structured creation of geometric shapes, a *referential part* that deals with coordinate systems and their interrelationship, and a *constraint part* that concerns the expression of explicit and implicit constraints and dimensions. The various language elements interact.

The global process of shape creation is divided into individual feature creation and modification steps. Features have dependencies on *prior* features, that is, on other, previously created features, that are referenced. Such a reference can be geometric, for features that modify prior features such as a blend or a cut, or the reference can be positional, when aligning elements of the new feature with elements of prior features, or when referencing elements of prior features for the purpose of defining dimensions or constraints. These dependencies are logically an acyclic graph: Each graph node is a feature, and the features directly depending on it descend from it by a directed arc. This type of dependency is referred to hereafter as *explicit feature dependency.* Explicit feature dependency is a dependency on the generative Erep level in that it does not change with most editing operations.

In addition to explicit feature dependency, features interfere with each other geometrically in the instantiated model. For example, a cut may extend across several protrusions. This has been called *feature collision* by some authors; e.g., [27]. Such dependencies are recorded on the Erep translation level as they may change even when explicit feature dependencies do not. Logically, featue collision is recorded by an undirected graph, called the *feature interference graph*, whose nodes are the features. There is an edge between two nodes if the corresponding features interfere. Note that the graph is edited implicitly when altering the design.

4.1 Geometric Language Constructs

The purpose of the geometric constructs is to express basic shape operations. These include basic shape elements for sketching, such as lines, conic sections, and splines. Furthermore, basic solid shapes are created from 2-dimensional geometry using extrusion, revolution, and sweeping. These basic shapes are

combined as features using protrusions, cuts, and restrictions; i.e., regularized union, difference, and intersections. Shapes are modified by chamfering, blending, and positionally editing reference points and curves. There can be other shape operations, including the creation of ribs and shells, as well as feature changes due to editing dimensions or constraints.

Features can also be grouped, patterned, and mirrored. In a patterned replication, some parameters or dimensions could be varied. Mirroring operations require precise rules for the interpretation of mirrored attributes, and for permitted editing operations, if any.

The geometric constructs have modalities that affect exactly how to interpret the operation. For instance, when creating a profiled cut, one may specify either a specific depth, or else that the cut should lie between two selected surfaces. In Figure 5, a hole extends between two specified surfaces that have been selected visually. When one of the surfaces is moved, the hole changes accordingly. When

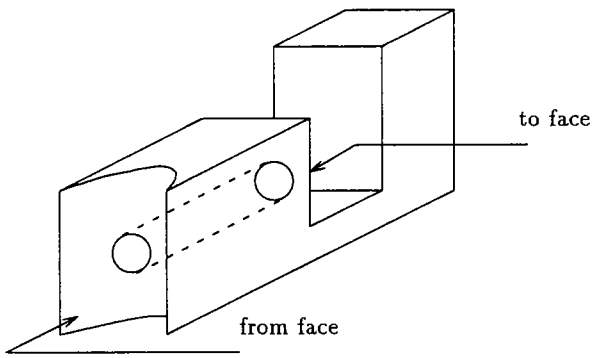

Figure 5: A hole positioned between two referenced surface areas.

one or both surfaces are deleted, reasonable rules and user-interactions have to be employed to reattach the orphaned feature. Those rules for reattaching or deleting orphaned features are part of the user-interface design. We do not propose any here, and only stress the need to have them.

4.1.1 Feature Construction

Three types of features are distinguished: generated features, datum features, and modifying features. Generated features create geometry from sketches and modalities interpreting the sketches. Datum features are used to establish a geometric reference scheme that does not rely on numerical coordinate transformations. Minimally, we have datum planes, lines and points. Datum surfaces and curves can also be included. Modifying features are operations that change

prior features. Examples include chamfers and rounds, shell operations, and so on. Dimensioning and constraints become part of the Erep description of each feature.

4.1.2 2-D Geometries

Two-dimensional sketches are constructed for the purpose of creating a three-dimensional feature, such as an extrusion or a sweep, or as separately stored sketches for future use in one or more designs. Topologically, the elements of the sketch must not intersect except at explicitly constructed points. Thus, the quadrilateral in Figure 6 is allowed only if the intersection p is made explicit by intersecting and subdividing the two line segments (a, c) and (b, d).

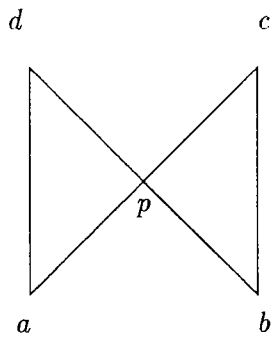

Figure 6: Sketched elements intersect only at explicitily constructed points.

2-D geometries are built from drawn primitives that include line segments, circular arcs, conic sections, and splines. In each case, there are reference points that are explicit, such as the endpoints of lines and arcs, or implicit, such as the centers of circle arcs.

It is possible to specify splines by control points. However, it might be more natural to specify the spline type symbolically and constrain the spline to interpolate a set of reference points. Just as partial differential equations systems allow the user to specify which solver should be used, there could be a set of construction algorithms from which the designer can specify which interpolation method should be applied to derive the spline.

The geometric primitives can be modified and refined using operations including intersection and filleting. In the case of filleting there is again a choice as to the method to be applied.

In addition to concrete geometric structures built in this way, datum points and lines can be specified. A datum line could be used as axis of symmetry for a mirroring operation, or as an aid in constructing a specific shape, or as axis of rotation when constructing a revolved feature from the drawing.

Dimensioning (both linear and angular) and geometric constraints such as perpendicularity, tangency or higher continuity are added so as to fully specify the sketch and compute its actual shape.

4.2 Referential Constructs

When creating a feature, say by sketching a cross section and extruding it, it is necessary to specify a sketching plane. The sketching plane serves two purposes at the same time: It provides the interface to the sketching operations and, when properly oriented and positioned, relates the new feature to the already existing features. Such planes are often datum planes.

Elements of prior features include (logical) edges and vertices, as well as silhouettes subject to some restrictions. All prior features are projected orthographically onto the sketching plane and can be used for alignment and other geometric constraints, as well as for dimensioning. Existing dimensions may also be referenced.

In addition to datum planes and the projection of prior features, other referential structures can be created such as datum axes and points, as well as datum surfaces and curves. The datum creation parallels the ordinary geometry creation, except that we are building surfaces and curves instead of solid geometry. It can be implemented as a parallel data structure in the core modeler.

We consider two types of datums: *feature datums* and *local datums*. A feature datum is a feature by itself and is explicitly created as a feature of a part. It can be referred to in any other new feature that is added to the part. Local datums are created as part of the process of defining another feature. Datums of this type can be referred to only inside the feature in which they were created. The distinction between the two types is made in support of hiding information: Feature datums are always displayed, whereas local datums would be displayed only when creating or editing the feature of which they are part. Feature datums and local datums both are created and positioned using geometric and dimensional constraints.

4.3 Constraints and Dimension Constructs

Constraints can be algebraic or geometric. An *algebraic constraint* relates variables that have numerical values, such as linear and angular dimensions. It can be equational or relational. Such a constraint is *parametric* if it has the form $x = f(\mathbf{y})$ where x is a variable and $\mathbf{y}$ is a set of variables not including x. The expression f is formed from the arithmetic operations and a selected set of functions.

A *set* of parametric constraints is parametric iff the variables can be ordered

such that the resulting set can be written sequentially in the form

$$\begin{aligned} x_1 &= f_1(\mathbf{y}_1) \\ x_2 &= f_2(\mathbf{y}_2) \\ &\vdots \\ x_n &= f_n(\mathbf{y}_n) \end{aligned}$$

where every $\mathbf{y}_i$ does not contain any of the variables $x_i, x_{i+1}, ..., x_n$.

A *variational* (equational) constraint has the form $f(\mathbf{y}) = 0$. It is well known that a set of variational equational constraints can be rewritten in parametric form under the assumption that the expressions f_i are formed only using the operations $+$, $-$ and $\times$; [5, 6, 15, 32]. These techniques have been developed in symbolic computation. In some cases these algorithms also apply to more general expressions. Since the symbolic computation that derives equivalent parametric constraints would be a preprocessing step, demands on their efficiency could be relaxed.

A *geometric constraint* relates two geometric elements and specifies, for example, that two lines are parallel, coincident, or perpendicular, that two circles are concentric, that a number of points are collinear or coplanar, and so on; e.g., [4]. In most cases, geometric constraints can be expressed equivalently by algebraic constraints, and geometric theorem proving has developed systematic tools to do so; e.g., [6]. The resulting constraints are almost always variational.

Because of fidelity, the geometric constraint formulation should be recorded as given, even when the constraint solver requires only algebraic constraints. Thus, between the Erep record of constraints and the constraint solver there could be an interface that translates constraints into the native representation of the solver. In this case, care has to be taken that the solver is able to give error messages for constraints that cannot be solved in terms that the user understands.

4.4 Editing Issues

A completed design or a design in progress can be edited by changing any feature with respect to its modality, shape, or its dimensioning and constraint scheme. This poses problems for reinterpreting dependent features. For example, consider a block with a cylindrical cut shown in Figure 7. Assume we have dimensioned the block and the cut as shown. Clearly, there is little problem editing the dimensions. Moreover, many changes to the shape of block can be done without affecting the rules for placing the cut.

Now suppose that the edge from which the axis of the cut has been dimensioned with d1 and to which it is parallel is deleted. Depending on how it is deleted, the placement of the cut becomes uncertain. If the edge is rounded,

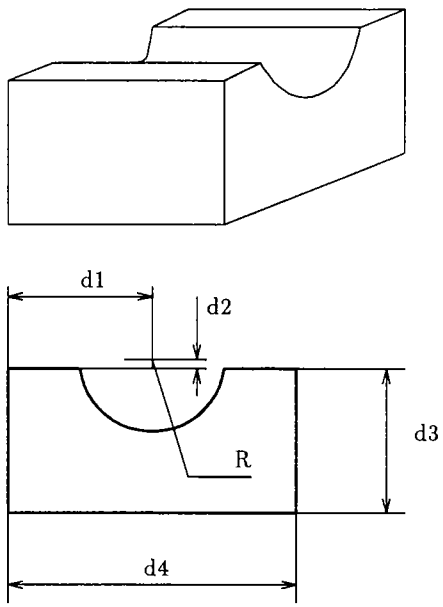

Figure 7: Editing a block with a cut.

we may assume by default that the unrounded edge is to be used because the round was introduced later. But if the block was created by extruding a rectangular cross section and we edit this cross section changing it into a triangle by eliminating the vertex that became, in the extrusion, the edge referenced by the cut, then no simple rule can be formulated. In such situations a dialogue with the user must establish what to do with the dependent feature.

5 Implementation Issues

The implementation of an Erep-based system poses some problems that deviate from ordinary Brep-based implementations because of the need to mediate between different levels of abstraction. This mediation, furthermore, needs to be done in a manner that does not commit to a particular underlying modeling system, for we want to retain the freedom of exchanging core modeling systems when technological advances make this advisable. When exchanging the core, we want to preserve our investment in archived designs as well as in the other system components and subsystems. To accomplish this requires standardizing the information flow between the various subsystems and maintaining association tables that establish inverse maps between, for example, Brep data structures and corresponding Erep entities. Since the initial information flow is always

from the UI to the core modeler, this is not really an inverse problem. Rather, it is like maintaining symbol tables in programming language translation for the purpose of supporting symbolic debuggers.

As an example consider how to support a graphical *pick* operation to identify geometric entities, say for generating a feature. It is clear that the result of a pick must be recorded in the Erep in a way that permits automatic regeneration of the geometry and editing it. This is done by naming every entity that can be picked and using this name. But the pick depends on a view and cursor position that are incidental. We could support the pick operation directly from the Erep, but this would require duplicating the work performed by the core modeler, and so it should be supported instead by a sequence of operations that involve computations at the core modeler level, i.e., operations on the Brep. But now we must be careful not to jeopardize the independence from the core modeler. We do it as follows:

1. Compute the line of sight through the mouse position clicked. This line is constructed from the viewing transform and the screen/mouse handler.

2. Intersect this line with the instantiated model in the core modeler, and sort the intersected entities by depth.

3. Highlight each entity in turn, asking the user to confirm or reject the choice.

4. Translate the selected physical entity to the corresponding logical entity and record it by name in the Erep.

The problem is step (4) which requires maintaining an association map between a physical Brep entity and the corresponding logical Erep entity. Note that an Erep entity could correspond to several Brep entities. and the types of the Brep entity and the corresponding Erep entity may well differ. Since the Brep is constructed from the Erep, it is clear that such a map can be constructed straightforwardly. Other operations requiring the cooperation of several subsystems, possibly at different levels of abstraction, are implemented analogously. The auxiliary data structures needed to build the necessary associations are then fairly obvious.

A substantial aspect of the system is the constraint solver. There are a number of research constraint solvers, as well as commercial constraint solvers. The basic issues are how they negotiate speed, scope, and recoverability. Typically, the constraint solver requires translating some of the Erep constraint statements into a uniform, internal representation. To recover from under- or overconstrained situations, it is again necessary to build, when translating the constraints, an association table, so that failing constraints can be presented to the user in the terms he has used to express them. This applies to both the explicit constraints the user has expressed, as well as to the inferred constraints

that the UI deduced from the sketch. For example, we can adopt inference rules such as "if two lines have been sketched as approximate perpendiculars, then they are in fact perpendicular." In some cases, applying such rules automatically could be wrong, and the user should be given a mechanism for selectively rejecting such implied constraints. Moreover, when editing a design, some inferred constraints may have to be removed. The Erep supports this requirement by recording all constraints, whether they have been inferred or were given explicitly.

6 Analysis and Process Planning Representations

While geometric modeling capabilities in CAD systems have improved considerably, comparatively less attention has been paid in the past to integrating engineering analysis, process planning, and other important aspects of the product modelling process. In recent years, several research efforts have focused on integrating geometric design, manufacturing information, and design methodologies; [1, 11, 26, 29]. Such work makes progress towards a product modeling science.

Because the relevant engineering information for the product modeling process varies with the different aspects it encompasses, the representation used must also vary to properly meet the requirements of each aspect. These different representations can be derived from a common model — provided that the particular engineering data for each model is either already contained in the common model or else is derivable from it algorithmically.

We discuss how a number of issues can be addressed using the Erep approach in order to derive different representations for engineering analysis and process planning.

6.1 Analysis Representations

An engineering analysis of a geometric design requires additional information. This additional information has been called *analysis attribute data* [26] and is all the information beyond the geometric domain definition needed to fully define the physical problem to be solved [17, 25]. Analysis attribute data includes material properties, loads, boundary conditions, initial conditions, mesh gradation information, rules for idealizing the geometry, etc; [26, 29].

Analysis attributes specification is a geometry-based task, and the structure that supports analysis data will be strongly tied to the geometry representation. However, it is desirable to keep attribute specification as independent as possible from the details of the geometry representation, [26]. Hence, to associate attributes with a feature or a logical part of a feature in the Erep representation is a good and convenient choice, and would be superior to an association with the low-level Brep of traditional core modeling systems.

It seems natural to restrict the association of analysis attributes to existing entities in the geometric model only. However, attributes are not always defined in a one-to-one relation with geometric entities; e.g., [23]. Hence, it must be possible to associate an attribute either with several geometric entities or with only a part of a geometric modeling entity. In the latter case, the geometry representation needed to perform the analysis may be different from the geometric model designed. So, it is necessary to provide the capability to generate a geometry representation that is best-suited to carrying out the analysis without altering the original design. In the Erep approach, there is no need for edge or face breaks common in Breps; [25]. Instead, datums can be defined with which attributes are associated and then the association can be properly handled by the analysis translator.

The analysis of most engineering parts and assemblies ignores a number of geometric details of the geometric model. In a geometry-based integrated approach like the one sketched here, a feature in the Erep representation could be tagged by the user as either essential or inessential for analysis porposes. Furthermore, the Erep feature representation could support a set of rules defining under what conditions the feature should be removed in the analysis. Such conditions would depend on either predefined values or on values that are determined as part of an adaptive analysis process [25].

When dealing with complex structures, and in order to produce more computationally efficient models, an analysis makes use of *reduced element types.* These are elements whose dimensionality is less than that of the model. Representing these elements in the geometric model requires nonmanifold topologies. While this might be difficult to automate based on a Brep, it would be simple in the Erep, in contrast: It suffices to associate with a feature its *simplification rules.* For instance, a hole with a diameter-to-length ratio less than a certain value can be reduced to its axis. Extrusions or sweeps representing beams or plates are similarly simplified.

Mechanical design optimization iterates a cycle consisting of a design step, followed by one or more analysis steps. The result of the analysis steps are then used to alter the design, improving functional characteristics of the part or assembly that is designed. This optimization loop is implemented traditionally as follows: Interface the Brep geometry representation with a mesh generation program deriving an analysis representation that becomes the input to the analysis steps along with the analysis attribute data. But the output of the analysis is not really in a form that could be used to alter the geometry automatically, chiefly because the association of elements with the geometry is at best on the Brep-level that does not support feature-based editing. By deriving the analysis representation from the Erep instead, the association between feature and elements is direct. Thus it becomes possible to process the analysis results to recommend feature modifications and additions. For example, if an area of high

stress is along a concave edge that borders two features, we could deduce automatically that the addition of a fillet feature is advisable. Moreover, if the stress area is across a thin-wall protrusion we could deduce that the thickness of the walls ought to be increased. These deductions can be made algorithmic, because the Erep association delivers high-level geometry information.

6.2 Process Planning Representations

A process planning system is a procedure that converts the design information into process documentation; [7]. The goal of process planning is to choose a set of manufacturing operations from a predefined repertoire and define on them a sequence that meets the design objectives at minimal cost; [19]. To accomplish this task, a process planning system analyzes the product design, transforming design features into manufacturing features.

Much research has been done to derive features needed for machining parts from the Brep; e.g., [1, 7, 19]. This work seeks to apply sophisticated geometric reasoning to derive, from the lower-level geometry representation such as the Brep, groupings of geometric elements that constitute a feature. This task is complicated by feature interference that may obliterate tell-tale geometric entities. By raising the level of abstraction in the product design model, however, feature extraction and transformation becomes much simpler. We now sketch some ways in which an Erep can be used in process planning.

There are two process planning approaches: the *variant* approach and the *generative* approach. The variant approach sets up a process plan as a modification of a plan previously defined for a design that is, in some sense, similar to the design under consideration. The generative approach generates a plan from scratch considering the part features and making no prior assumptions; [1, 19].

A generative process planner carries out several tasks. For example, consider the system presented in [1] which performs the following six tasks: feature refinement, process selection, tool selection, process sequencing, fixture planning, and numerically controlled cutter path generation.

With each type of feature there is associated a limited set of possible manufacturing methods; [1, 19, 30]. The advantage that the Erep approach offers in this case is the fact that it is already a feature-based representation. In order to make these advantages more explicit, we relate the Erep approach to some of the process planning tasks mentioned before.

Some tasks may use geometric reasoning, for example generating approach and feed directions in feature refinement. This requires that the feature representation be cross-linked to an auxiliar boundary representation; [1]. Using Ereps, a cross-link is provided by the geometry translator without additional work, and the generative attributes of the feature, extruded, revolved, etc., assist in determining approach and feed directions.

Determining both process sequences and the processes themselves would

depend on the feature relationships. Among all possible feature relationships, feature nesting and feature intersection appear to be most important to process planning [1]. The feature collision graph maintained by the Erep translation, and the explicit feature dependency graph of the Erep, give directly both the intersecting features and the nested features. Sometimes, the process or the process sequence or both depend on the tolerances imposed by the designer [19, 18]. But an Erep can be naturally extended to associate tolerance information with the features through the dimensioning and constraints definition scheme.

The variational approach benefits from the fact that the Erep approach encourages archiving and editing previous designs. A new design that has been derived from an old design by editing is likely to require a process plan similar to that of the old design. Thus, if the design editor archives not only the design but also the design history and associated process plans, it would be simple to retrieve process plans associated with the prior designs and modify them as needed.

References

[1] D.C. Anderson and T.C. Chang. Geometric reasoniong in feature-based design and process planning. *Computer and Graphics*, 14:225–235, 1990.

[2] R. Bartels, J. Beatty, and B. Barsky. *Splines for Use in Computer Graphics and Geometric Modeling.* Morgan Kaufmann, Los Altos, Cal., 1987.

[3] A. Bowyer, J. H. Davenport, D. A. Lavender, A geometric algebra system. In D. Kapur, editor, *Integration of Symbolic and Numeric Methods.* MIT Press, 1991.

[4] B. Bruderlin. Symbolic computer geometry for computer-aided geometric design. In *Advances in Design and Manufacturing Systems*, NSF Conference, Tempe, AZ, 1990.

[5] B. Buchberger. Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory. In N. K. Bose, editor, *Multidimensional Systems Theory*, pages 184–232. D. Reidel Publishing Co., 1985.

[6] B. Buchberger, G. Collins, and B. Kutzler. Algebraic methods for geometric reasoning. *Annual Reviews in Computer Science*, 3:85–120, 1988.

[7] T.C. Chang and R.A. Wysk. *An Introduction to Automated Process Planning Systems.* Prentice Hall, 1985.

[8] J. Chung, R. Cook, D. Patal, and M. Simmons. Feature-based geometry construction for geometric reasoning. In *Proc. ASME Conf. Comp. in Engr., Vol 1*, pages 497–504, San Francisco, 1988.

[9] J. Cunningham and J. Dixon. Designing with features: The origin of features. In *Proc. ASME Conf. Comp. in Engr., Vol 1*, pages 237–242, San Francisco, 1988.

[10] D. Dobkin, L. Guibas, J. Hersberger, and J. Snoeyink. An efficient algorithm for finding the CSG representation of a symple polygon. *ACM Computer Graphics*, Vol. 22, No. 4:31–40, August 1988.

[11] J.R. Dixon E.C. Libardi Jr. and M.K. Simmons. Computer environments for the design of mechanical assemblies: A research review. *Engineering with Computers*, 3:121–136, 1988.

[12] G. Farin. *Curves and Surfaces for Computer-Aided Geometric Design.* Academic Press, 1988.

[13] V. Ferrucci. *Simple*^n_X user manual and implementation notes. Technical Report RR 06-90, Università di Roma La Sapienza, Dept. of Informatics and Systems, 1990.

[14] V. Ferrucci and A. Paoluzzi. Sweeping and boundary evaluation for multidimensional polyhedra. Technical Report RR 02-90, Università di Roma La Sapienza, Dept. of Informatics and Systems, 1990.

[15] C. M. Hoffmann. *Geometric and Solid Modeling.* Morgan Kaufmann, San Mateo, Cal., 1989.

[16] R. Juan. On Boundary to CSG and Extended Octrees to CSG conversions. In W. Strasser, editor, *Theory and Practice of Geometric Modeling*, pages 349–367. Springer-Verlag, 1989.

[17] C.N. Tonias M.S. Shephard and T.J. Weider. Attribute specification for finite element models. *Computers and Graphics*, 6:83–91, 1982.

[18] M. J. Pratt. Synthesis of an optimal approach to form feature modeling. In *Proc. ASME Conf. Comp. in Engr., Vol 1*, pages 263–274, San Francisco, 1988.

[19] M. J. Pratt. Solid modeling and the interface between design and manufacture. *IEEE Computer Graphics and Applications*, 4, No. 7:52–59, July, 1984.

[20] A. Requicha. Mathematical models of rigid solids. Technical Report Memo 28, University of Rochester, Production Automation Project, 1977.

[21] J.R. Rossignac, P. Borrel, and L.R. Nackman. Interactive design with sequences of parameterized transformations. Technical Report RC 13740, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, New York, 1988.

[22] _____. SHAPES geometry library reference manual. Technical report, XoX Corporation, 1991.

[23] M. Sabin. Modelling operations in a cellular modeller. Manuscript, 1992.

[24] V. Shapiro and D.L. Vossler. Boundary-based separation for B-rep to CSG conversion. Technical report, Department of Computer Science, Cornell University, 1991.

[25] M.S. Shephard. Finite element modeling within an integrated geometric modeling environment: Part ii – attribute specification, domain differences, and indirect element types. *Engineering with Computers*, 2:72–85, 1985.

[26] M.S. Shephard. The specification of physical attribute information for engineering analysis. *Engineering with Computers*, 4:145–155, 1988.

[27] R.M. Summer. Feature and face editing in a hybrid solid modeler. In *ACM/SIGGRAPH Symposium on Solid Modeling Foundations and CAD/CAM Applications*, June 1991.

[28] T.L. De Fazio, *et al.* A prototype of feature-based design for assembly. Technical Report CSDL-P-2917, The Charles Stark Draper Laboratory, Inc., Cambridge, MA, 1990.

[29] V. Unruh and D.C. Anderson. Feature-based modeling for automatic mesh generation. *Engineering with Computers*, 8:1–12, 1992.

[30] P. Wilson and M. Pratt. A taxonomy of features for solid modeling. In *Geometric Modeling for CAD Applications*, pages 125–136, J. Wozny, H.W. McLaughlin, J.L. Encarnaçao, editors. Nort-Holland, IFIP conference, 1988.

[31] P. R. Wilson. Unsolicited proposal to CAM-I Inc. from Lucas Research Centre on form features. Technical Report UP-82-GM-01.1, CAM-I, Arlington, Texas, 1982.

[32] Wu Wen-Tsün. Basic principles of mechanical theorem proving in geometries. *J. of Systems Sciences and Mathematical Sciences*, 4:207–235, 1986.

A Erep Example Language Specification

We give a syntactic definition of an example Erep. The example has been kept simple and is not meant to be a comprehensive design. Rather, this Erep shows how the geometry can be specified without a commitment to an underlying core modeler to whose functionalities and representations the Erep translation would map. The Erep is easily extended. In particular, the additions sketched in Section 6 are easily formalized and incorporated.

A.1 Assemblies

An assembly is a collection of parts that have been assembled, one at a time, according to a set of matching rules. A set of matching rules specifies the constraints that locate a part relative to the parts already assembled.

```
<assembly>     ::=  ASSEMBLY <name> <stamp>
                      <global_info>
                      <parts_list>
                    END_ASSEMBLY

<global_info>  ::=  GLOBAL
                      UNITS <unit>

<parts_list>   ::=  PARTS <part>
                 |   <parts_list>  ;  <part> <matching_rules>

<stamp>        ::= integer

<unit>         ::= mm | cm | m | in | ft
```

The constraints are analogous to the datum-placing constraints described later. Conceptually, they mate or align surfaces, curves and points, until the relative position of the part, with respect to the partial assembly, has been completely specified.

```
<matching_rules> ::= <rule>
                   |  <matching_rules> ; <rule>

<rule>           ::= <mating>  |  <aligning>

<mating>         ::= MATE  <geo_pair>
                   |  MATE  OFFSET <exp> <geo_pair>
                   |  ALIGN <geo_pair>
                   |  ALIGN OFFSET <exp> <geo_pair>
```

```
<geo_pair>       ::= <point>   <point>
                   | <axis>    <axis>
                   | <surface> <surface>
```

To ALIGN two surfaces means to have them coincide in the same orientation. To MATE them means to have them coincide in opposite orientation. In analogy, curves are aligned or mated depending on whether their orientation agrees or is opposite. The number and type of rules used to position each part in the assembly should be both necessary and sufficient.

A.2 Parts and Features

A part consists of several features that are *datum*, *generated*, or *modifying* features. Since a modifying feature alters prior features, it cannot be a first feature.

```
<part>      ::= PART <name> <stamp>
                  <global_info>
                  <features_list>
                END_PART

<features_list> ::= <d_feature>
                  | <g_feature>
                  | <features_list> <feature>

<feature> ::= <d_feature> | <g_feature> | <m_feature>
```

A <d_feature> is a datum feature. Datums are points, lines, planes, or coordinate systems. The manner in which they are specified will be explained below.

```
<g_feature> ::= <e_feature>
              | <r_feature>
              | <s_feature>
```

Generated features (<g_feature>) are constructed from two-dimensional sections using one of several operations that depend on chosen attributes. Logically, we have restricted generated features to sweeps, where extruded (<e_feature>) and revolved features (<r_feature>) are special cases of the general sweep operation (<s_feature>).

```
<m_feature> ::= <c_feature>
              | <o_feature>
              | <f_feature>
```

Modifying features (<m_feature>) operate on three-dimensional geometry and change edges and vertices by chamfering (<c_feature>), rounding (<o_feature>), or filleting (<f_feature>). Other modifying features could be defined such as drafting faces.

A.3 Datums

There are two types of datums, the *feature datum* and the *local datum.* A feature datum is a feature, whereas a local datum is part of another feature, such as a generated feature. The feature datum is specified as

```
<d_feature> ::= FEATURE <name> <stamp>
                  <feat_depends> ;
                  <datum_type> <orient_modification> ;
                  <constraint list>
                END_FEATURE

<orient_modification> ::=  <empty>  |  ORIENT_OPPOSITE

<feat_depends>        ::=  <empty>
                         |  <feat_depends> , <name>
```

and the local datum is specified as

```
<l_datum>    ::= DATUM <datum_id> <datum_type> <orient_modification> ;
                   <constraint list>
                 END_DATUM

<datum_type> ::= DATUM_POINT | DATUM_AXIS | DATUM_PLANE | DATUM_CS
```

The two types are equivalent, but giving them a separate syntactic form supports information hiding in the user interface.

We consider linear datum types, namely points, axes and planes, as well as coordinate systems which are a group of three oriented datum axes that are pairwise orthogonal and have been named x, y and z. They are defined with geometric constraints that are expressed as follows.

```
<constraint_list> ::= <geo_constraint>
                    |  <constraint_list> ; <geo_constraint>

<geo_constraint>  ::= <const_verb> <const_type> <name>

<const_verb>      ::= ON
```

```
                  | PARALLEL  | OFFSET  <exp>
                  | NORMAL    | ANGLE   <exp>

<const_type>     ::= <empty>
                  | EDGE    | FACE
                  | CSA_X   | CSA_Y   | CSA_Z
                  | CSP_X   | CSP_Y   | CSP_Z
```

Datum constraints are composed by placing the datum geometrically with respect to another geometric entity. For example, when defining a point by the intersection of a `line` and a `plane`, this is expressed as `ON plane; ON line`.

Semantically, the constraints must make sense and fully define the geometric placement without redundancy. The verb `ON` specifies incidence, of a point with another point, axis, plane or surface. The verbs `PARALLEL` and `OFFSET` specify a constraint of orientation in space, with `OFFSET` adding a distance constraint. The verbs `NORMAL` and `ANGLE` also specify orientation constraints. When a datum point is constrained by `OFFSET` only a distance constraint is expressed. For example, with a prescribed offset from a reference point, a datum point would be constrained to lie anywhere on a sphere centered at the reference point with radius equal to the offset distance. Other geometric constraints could be added, such as tangency or curvature continuity.

No constraint type is required when the `<name>` refers to a datum point, axis or plane. Coordinate-system-based constraints have been separated by the axis names and whether the axis or its associated zero plane is referred to. So, `CSA_X` refers to the x coordinate axis, whereas `CSP_X` refers to the plane $x = 0$. For instance, a point constrained by `ON CSA_X csys1` must lie on the x-axis of the coordinate system `csys1`, but the constraint `OFFSET 55.3 CSP_X csys1` requires the point to lie on the plane $x = 55.3$ in the coordinate system `csys1`.

Axes and planes have an orientation that is implied by the way in which they have been constrained. The orientation rules are explained in Appendix B. This default orientation can be explicitly reversed by `ORIENT_OPPOSITE`.

A.4 Generated Features

In the example language, generated features are sweeps. For linear and revolving sweeps, special syntactic forms have been chosen in view of their simplicity and frequent use. All other sweeps are considered to be general, as explained later.

A.4.1 Extruded Features

An extruded shape is a sweep of a plane cross section along a line segment. An extruded feature is either a *protrusion* that adds material, or else a *cut* that

removes material. If the extrusion trajectory is NORMAL, the extrusion proceeds in the vertical direction to the cross section plane. Otherwise the cross section is extruded along a datum axis.

```
<e_feature>     ::= FEATURE <name> <stamp> EXTRUDED;
                      <feat_depends> ;
                      <volumetric_type> ;
                      { <l_datum> }
                      <e_trajectory> ;
                      <e_extent> ;
                      <cross_section> ;
                      <pattern>
                    END_FEATURE

<volumetric_type> ::= PROTRUSION  <orientation>
                    |  CUT  <orientation>

<e_trajectory> ::= TRAJECTORY NORMAL
                 |  TRAJECTORY <datum_axis>

<e_extent>      ::= EXTENT <e_from_spec> <e_to_spec>

<e_from_spec>  ::= FROM offset  |   FROM ALL
                 |  FROM face    |   FROM <datum_plane>


<e_to_spec>     ::= TO   offset  |   TO   ALL
                 |  TO   face    |   TO   <datum_plane>
```

The extent of the sweep is determined by specifying both ends. With respect to the orientation of the cross section plane, the sweep proceeds from the <e_from_spec> to the <e_to_spec> in the direction of the plane normal. In case the cross section plane is a reference to an already existing geometric item, an orientation specification of the cross section states whether this normal or its opposite is to be used.

An offset states that the beginning or end of the extrusion is at a plane parallel to the cross section plane at distance offset. Positive offsets are in the direction of the normal, negative offsets are in the opposite direction. A zero offset specifies the cross section plane itself.

A sweep can also be bounded by a face or a datum plane. The face could be curved and the datum plane need not be parallel to the cross section plane. In those cases, the sides of the swept shape extend to the intersection with the bounding face or the datum plane.

ALL means that the sweep extends a variable distance. When ALL is the <e_from_spec>, the beginning of the sweep lies an arbitrary distance in the opposite direction of the normal, and as a <e_to_spec> it lies an arbitrary distance in the normal direction of the cross section plane. In case of a cut, ALL implicitly specifies the farthest intersected surface that faces away from the sweep, and in the case of a protrusion the farthest intersected surface that faces in the opposite direction.

The details of the cross section and pattern specification are explained later.

A.4.2 Revolved Features

```
<r_feature>     ::= FEATURE <name> <stamp> REVOLVED;
                      <feat_depends> ;
                      <volumetric_type> ;
                      { <l_datum> }
                      <r_trajectory> ;
                      <r_extent> ;
                      <cross_section> ;
                      <pattern>
                    END_FEATURE

<r_trajectory> ::= TRAJECTORY
                       AXIS  <datum_axis>
                    END_TRAJECTORY

<r_extent>      ::= EXTENT <r_from_spec> <r_to_spec>
                  | EXTENT FULL

<r_from_spec>   ::= FROM angle
                  | FROM face  | FROM <datum_plane>


<r_to_spec>     ::= TO angle
                  | TO face    | TO  <datum_plane>
```

Revolved features are similar to extruded features in that they sweep along a standard trajectory, bounded by extent specifications closely analogous to those of extrusions. The standard trajectory is a revolution around an axis, in the direction of a right-hand rule with the thumb pointing in the axis direction. The angle zero is the right halfplane of the cross section plane, as seen from the positive side, and illustrated in Figure 8. The orientation specification of the cross section may change that.

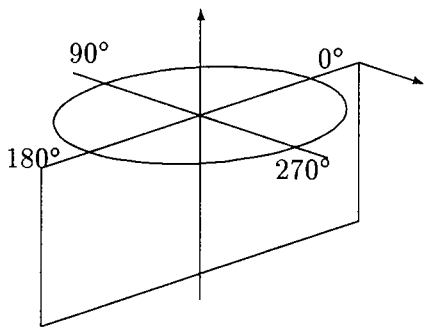

Figure 8: Revolving angle orientation convention, based on axis and on sketching plane orientation.

An `angle` specifies a half plane through the axis of rotation at the proper angle. A `FULL` extent is a full revolution by 360 degrees. The other specification possibilities are as for extruded features. Note that revolution extents may overlap, in which case the effect is equivalent to a full revolution.

A.4.3 General Sweeps

There are three differences between general sweeps and extruded or revolved shapes:

1. The trajectory now can be a general space curve.
2. The orientation of the cross section may vary along the trajectory.
3. The shape of the cross section may vary along the trajectory.

These new possibilities for varying shape require many additional parameters and attributes whose intuitive control raises research issues. We limit therefore the possibilities by restricting the cross section to be planar, and closed, and disallowing shape variation. The orientation at any point is perpendicular to the trajectory. Patterning has been disallowed arbitrarily.

```
<s_feature>     ::= FEATURE <name> <stamp> SWEPT;
                      <feat_depends> ;
                      <volumetric_type> ;
                      { <l_datum> }
                      <s_trajectory> ;
```

```
                    <s_extent> ;
                    <cross_section>
                  END_FEATURE

<s_trajectory> ::= <p_trajectory> | <g_trajectory>

<p_trajectory> ::= TRAJECTORY_P
                    PLANE <sketching_plane>
                    COMPONENTS
                      <component_list>
                    END_COMPONENTS
                    CONSTRAINTS
                      <constraint_list>
                    END_CONSTRAINTS
                    CONSTRUCTION
                      POS <point_id> <position>
                    END_CONSTRUCTION
                  END_TRAJECTORY

<g_trajectory> ::= TRAJECTORY_G
                    COMPONENTS
                      <comp_list>
                    END_COMPONENTS
                  END_TRAJECTORY

<s_extent>     ::= EXTENT <s_from_spec> <s_to_spec>
                 | EXTENT FULL

<s_from_spec>  ::= face | <datum_plane> | <datum_point>
                 | ALL

<s_to_spec>    ::= face | <datum_plane> | <datum_point>
                 | ALL

<comp_list>    ::= <edge_spec>  |  <spline_spec>
                 | <comp_list>  <edge_spec>
                 | <comp_list>  <spline_spec>

<edge_spec>    ::= edge | ( face , face )

<spline_spec>  ::= SPLINE <type> <point> <point> { <point> }
```

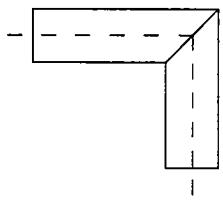

Figure 9: Corner rule for general sweeps

The trajectory is a curve that can be built from components. The curve must be continuous with nonsingular smooth segments, but may contain tangent discontinuities. At an angle in the trajectory, the sections are joined by extending the sweep from both sides along the limit tangent until its intersection with the bisecting plane of the two limit tangents. See also Figure 9.

The extent specification either terminates the sweep at specific faces or datum planes, or else ends the sweep at planes through a datum point that must lie on the trajectory. This plane through the datum point is perpendicular to the trajectory tangent at the point.

The trajectory may be open or closed. The extent specification `ALL` is as in the `<e_extent>` applying to open trajectories only. If the trajectory is too short, it is extended by the tangent at the trajectory end points. A `FULL` extent specification only applies to closed trajectories.

A planar trajectory is specified by the same syntax as cross sections, and is described later. A nonplanar trajectory is specified by a list of components where each component is an edge occurring in a prior feature or a spline. The spline is typed so that its construction from the list of points is understood. For example, type Bézier interprets the points as control points.

A.5 Section Specifications

Sections are two-dimensional drawings. The geometry is specified by describing the constituent curves and lines, primarily topologically, and giving a set of constraints that fully describe relative position and dimension of the geometric quantities. Apart from the component curves that comprise the section, there may be auxiliary geometric items such as lines of symmetry that are used to construct the section and/or solve the constraints.

A section is associated with a sketching plane that is typically a datum plane. The geometry of prior features is available in projection onto the sketching plane and may be referenced for the purpose of constraining the cross section. The user draws a rough sketch of the cross section, and from this drawing initial coordinates are assigned to points. When generating the cross section to

constraint specification, the point coordinates change. The current set of point coordinates is recorded in the CONSTRUCTION section, and is edited each time the cross section is regenerated.

```
<cross_section> ::= CROSS_SECTION <name>;
                      PLANE <sketching_plane>
                      COMPONENTS
                        <component_list>
                      END_COMPONENTS
                      AUXILIARY
                        <components_list>
                      END_AUXILIARY
                      CONSTRAINTS
                        <constraint_list>
                      END_CONSTRAINTS
                      CONSTRUCTION
                        POS <point_id> <position>
                      END_CONSTRUCTION
                    END_CROSS_SECTION
```

Sketches are made on datum planes or face planes. The geometric components are lines, points, arcs and splines. Splines are typed so the point list following can be interpreted.

There are three types of arcs. ARC3 is a circular arc where three points on the perimeter are known. The first and last are the end points, the second point is in the interior. The arc is oriented by the order of the three points. In ARCC+ and ARCC- the second point is the center of the arc. The ARCC+ is oriented counter clockwise from the first to the third point. The ARCC- is oriented clockwise from the first to the third point. For ARCC+ and ARCC- the first and third point may coincide and the arc is then a full circle oriented as before.

```
<sketching_plane> ::= face    |  <datum_plane>

<components_list> ::= <component>
                    |  <components_list> ; <component>

<component>       ::= <line>  |  <arc>  |  <spline>

<line>            ::= LINE <line_id> <point_id> <point_id>

<arc>             ::= ARC3  <arc_id> <point_id> <point_id> <point_id>
                    |  ARCC+ <arc_id> <point_id> <point_id> <point_id>
                    |  ARCC- <arc_id> <point_id> <point_id> <point_id>
```

```
<spline>          ::= SPLINE <spline_id> <type>
                        <point_id> <point_id> {<point_id>}
```

Constraints are geometric or other, and in most cases relate two named geometric quantities. Geometric constraints express symbolically that two geometric items are incident, tangent, parallel, perpendicular, or concentric. Other constraints quantify a symbolic variable, <cs_oth_symb>, whose value will be a distance or angle between two geometric items. The symbolic variables can be related explicitly with a relation. Constraints relating three quantities are collinearity of three points, or naming the angle subtended by three points.

```
<constraint>      ::=  <sec_geo_constr>  |  <sec_oth_constr>

<sec_geo_constr>  ::=  <sec_geo2_verb> <name> <name>
                   |   <sec_geo3_verb> <name> <name> <name>

<sec_geo2_verb>   ::=  ALIGN  |  TANGEN  |  ON
                   |   PARAL  |  PERP    |  CONCEN

<sec_geo3_verb>   ::=  COLLIN

<sec_oth_constr>  ::=  <sec_oth2_verb> <sec_oth_symb> <name> <name>
                   |   <sec_oth3_verb> <sec_oth_symb> <name> <name> <name>
                   |   <relation>

<sec_oth2_verb>   ::=  ANGLE  |  DISTAN

<sec_oth3_verb>   ::=  ANGLE

<sec_oth_symb>    ::=  <name>
```

Relations assign or constrain the symbolic variables of distance or angle. The relation syntax is not explicitly given and could be adopted from FORTRAN or some other programming language.

```
<relation>        ::= <simple_rel>
                   |  <compound_rel>

<simple_rel>      ::= <assignment>
                   |  <relational>

<assignment>      ::= <sec_oth_symb> = <exp>

<exp>             ::= an arithmetic expression
```

```
<relational>      ::= a Boolean expression using relational and
                      Boolean operations

<compound_rel>    ::= IF <relational> THEN
                        <relation>
                      { ELSE
                        <relation> }
                      ENDIF

<position>        ::= ( <number> , <number> )
                    | INTERSECTION <component_id> <component_id>

<component_id>    ::= <line_id>
                    | <arc_id>
                    | <spline_id>
```

A.6 Patterns

A pattern is a set of features created from a given feature using a replication rule. The pattern is treated as a single feature. Replication can follow a linear or revolute pattern

```
<pattern>         ::= SINGLE
                    | MULTIPLE <p_attributes>

<p_attributes>    ::= LINEAR <direction>  <step> <instances>
                    | ARRAY  <direction>  <step> <instances>
                             <direction>  <step> <instances>
                    | CIRCUL <datum_axis> <step> <instances> <a_sense>

<step>            ::= real

<instances>       ::= integer

<a_sense>         ::= CW  |  CCW
```

A.7 Modifying Features

Modifying features alter three-dimensional geometry of prior features. We consider the edge modifying operations of beveling, rounding, and filleting. Other operations would be easy to add.

```
<c_feature> ::= FEATURE <name> <stamp> CHAMFER;
                  <feat_depends> ;
                  <edgelist> ;
                  <chamf_spec> ;
                  <end_cond>
                END_FEATURE

<o_feature> ::= FEATURE <name> <stamp> ROUND;
                  <feat_depends> ;
                  <edgelist> ;
                  RADIUS radius ;
                  <end_cond>
                END_FEATURE

<f_feature> ::= FEATURE <name> <stamp> FILLET;
                  <feat_depends> ;
                  <edgelist> ;
                  RADIUS radius ;
                  <end_cond>
                 END_FEATURE

<edgelist>  ::= <edge_spec> { <edge_spec> }

<edge_spec> ::= edge   |   ( face , face )

<chamf_spec>::= ANGLE  angle   WIDTH  number
             |  WIDTH1 number  WIDTH2 number

<end_cond>  ::= face  |  <datum_plane>
```

An edge can be specified either by referencing it or else by referencing a pair of adjacent faces whose intersection is the edge. The edges must be an open linear sequence or a simple, closed loop of edges.

In the case of chamfers, the cut can be specified by giving an angle and a width of the cut, or by giving two widths on either side of the edge. When an angle other than 45° is given, the edges must be specified as pairs of faces and the angle is measured against the first face of each pair which must be planar. If consecutive left faces in an edge list are not coplanar, the chamfer will have an edge that is determined by the rules for generating sweeps along trajectories that have tangent discontinuities, whith the chamfered edges serving as trajectory.

Open edge lists require an <end_cond> of the modifying feature. Here the modifying feature is considered a sweep of a standard cross section and the end

condition is interpreted as the <s_from_spec> and <s_to_spec> are in sweeps. For simplicity, we require a single <end_cond> is applied at both ends.

B Orientation Rules for Axes and Planes

The construction of protrusions and cuts, and the measurement of signed offsets, requires that datum planes and axes be oriented. Since datums are constructed from constraints, we need to define rules for unambiguously deriving a default orientation from the constraints and from the sequence in which they have been given. Which set of rules is used and whether the rules are intuitively understood by the user is not important, because the user interface will routinely issue a query asking whether the default orientation, shown graphically, is the intended one. If the user indicates that it is not, the ORIENT_OPPOSITE modifier is inserted into the Erep. Because orientation is used to place dependent features, an editing operation that alters the orientation of a datum requires in particular that the Erep definition of dependent geometric entities and features must be edited accordingly.

Three conceptual devices are used to derive an orientation from constraints. The first device defines the orientation from an oriented geometric entity that is used in the constraints. Below, this method accounts for cases (1)–(3). The second device considers two or three points in space, ordered in a way that is derived from the geometry of the constraints and the order in which the constraints have been specified. Two points so ordered orient an axis, and three points define an oriented triangle that orients a plane by a right-hand rule. Below, this method accounts for cases (4)–(6). Finally, an orientation may be implied by observing on which side of a plane that is to be placed a constraining geometric quantity lies. This method is used below in case (7).

We now give a set of rules for orienting planes. Let P be a plane defined by the constraints.

1. PARALLEL or OFFSET to a given plane Q. Then P and Q have the same orientation.

2. NORMAL or ANGLE to an axis. Then P is oriented so that the inner product of its normal with the axis orientation vector is positive.

3. One NORMAL PLANE Q constraint and either an ON AXIS or two ON POINT constraints are given. The axis or the two points define an oriented line that is projected onto the intersection of Q and the plane P and so orients the intersection ℓ. The orientation of P is obtained by rotating Q about ℓ counter-clockwise, as seen in the orientation of ℓ. An analogous construction is used for the ANGLE ... PLANE constraint.

4. `ON AXIS` and `ON POINT` q. Two points p_1 and p_2 are chosen on the `AXIS`, ordered by the axis orientation, and the point q is added. If the axis constraint precedes the point constraint, in the Erep specification, the order is (p_1, p_2, q), otherwise it is (q, p_1, p_2). Then the plane orientation is from the right-hand rule applied to the points.

5. Three `ON POINT` constraints define the plane. The right hand rule is applied to the three points in the order in which the constraints have been given.

6. Two `ON AXIS` constraints are given. The axes must intersect, and three points are chosen as follows. The first point p is the axis intersection. The second point lies on the first axis, following p in the orientation of the axis. The second point follows p on the second axis.

7. A `PARALLEL AXIS` or an `OFFSET AXIS` constraint. The plane is oriented so that the axis lies on its positive side.

Axis orientation is based on the analogous principles, inheriting an orientation from a given axis, or being oriented by two ordered points, where again the point ordering depends in some cases on the order in which the constraints have been written down. Since any unambiguous set of rules will do we do not give one here.