On User-Defined Features

Christoph M. Hoffmann
Department of Computer Sciences
Purdue University

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

August 20, 1997

Abstract

Feature-based design is becoming one of the fundamental design paradigms of CAD systems. In this paradigm the basic unit is a feature and parts are constructed by a sequence of feature attachment operations. The type and number of possible features involved depend upon product type, the application reasoning process and the level of abstraction. Therefore to provide CAD systems with a basic mechanism to define features that fit the end-user needs seems more appropriate than trying to provide a large repertoire of features covering every possible application.

A procedural mechanism is proposed for generating and deploying user-defined features in a feature-based design paradigm. The usefulness of the mechanism relies on two functional capabilities. First the shape and size of the user defined features are instantiated according to parameter values given by the end-user. Second the end-user positions and orients the feature in the part being designed by means of geometric gestures on geometric references.

Keywords: Computer aided design, feature based design, user defined features, parametric design.

1 Introduction

Feature-based CAD systems such as Parametric Technology's Pro/Engineer deploy a design paradigm in which the designer may use a set of predefined features, such as slots, ribs, and holes, and operations for defining *sketched features* where geometry is created by sweeping a planar cross section or lofting between two planar cross sections. These operations are adequate to create complex shape designs, but they do not necessarily make a good connection between this basic repertoire and application needs for mechanical parts which use complex shape elements that reflect sophisticated functional requirements.

Some industry studies have attempted to define a comprehensive set of design features from mechanical parts. Here, a common experience has been that each new part gives rise to several new features, in addition to the ones previously conceptualized. Thus, seeking to devise a universal set of features would lead to a potentially unmanageable number of features that a CAD system might be asked to provide. For this reason, research has begun to investigate generic mechanisms for defining features using basic definitional mechanisms, provided by a core feature system, and giving the user the option of building custom feature libraries that might satisfy specific application needs without having to reimplement the CAD system or the system's interfaces.

While the feature mechanisms provided by commercial systems provide some of the needed mechanisms, they fall short in part because of the absence of sophisticated ways to build and use compound features. Feature definition languages such as the ASU test bed, [32], are an attempt to provide a more sophisticated definitional environment. The feature work by Bronsvoort *et al.*, [1], is another such attempt that focuses on the delicate problems of feature validity and conversion.

Commercial design practice is deeply influenced by the parametric constraint-based paradigm. The paradigm offers the possibility of designing variational classes of parts, that is, part families whose members are derived by changing a few parameters. The way CAD systems implement the paradigm is based primarily on a design feature history that can be re-evaluated upon changing parameters, thereby deriving the class members automatically. Here, the freedom to structure the design history any way the designer sees fit has turned out to be detrimental in many cases [6] because the chosen design history and consequential feature dependencies have profound implications on the ability to change the design.

CAD users combat the problem of undisciplined design history by devising design styles that are intended to minimize the interdependency of design features in the design history of a part. User-defined compound features have the potential of simplifying such work-arounds: Instead of urging the CAD user to adhere to a particular style, user-defined features can be compounded to become larger functionally meaningful design units. The advantage here is that design changes can be prestructured in the user-defined feature, and thus the compound features can be made to have greater functional independence from each other. Thus, the labor of bringing the design history into harmony with the functional requirements of the part family is done up-front once, when defining a user-defined feature, instead of repeatedly and indirectly, using a company-specific design style.

In addition to providing mechanisms for compounding feature elements into complex, user-defined features, tools are needed to ensure such user-defined features are used as intended. Validation rules should be associated with the feature definition whose successful evaluation assures us that the use of the feature is correct in a technical sense, thereby providing a measure of assistance to the designer to manage complexity of the design. For example, if a feature is to be used as a through hole, subsequent material additions should not convert it into a blind hole or an internal void because of feature collision.

In this paper, we describe some basic mechanisms for realizing user-defined features that permit compounding shapes, encapsulating attributes and parameters, and associating topological validity rules. Following earlier work, we have chosen to express the compounding mechanisms using a generative paradigm that derives from our Erep work and is responsive to constraint-based, feature-based design paradigms commonly in use in commercial practice. As in the earlier work, [15], we advocate neutral mechanisms for implementing user-defined features.

Because this work focuses on generative mechanisms for creating and using user-defined features, it does not address the problems of feature conversion and multiple feature views that are the subject of other work on feature-based design. A full exploration of the way in which a generative, history-based design can be combined with multiple feature views and feature conversions is beyond the scope of this paper. We hope to address it elsewhere.

2 Previous Work

A thorough review of the very extensive literature on feature-based design is presented in [27]. Among the different aspects that research in feature based design technology explores, we are interested in methods and tools that facilitate the design of parameterized features and parts.

There are numerous papers reporting on languages to represent features, including [10, 11, 12, 13, 15, 17, 18, 19, 25, 38, 34, 36]. The motivations for this approach fall roughly into the following categories: Evolving solid modeling, facilitating geometric reasoning, improving manufacturability, enabling CAD/CAM integration, supporting consistency verification, and unifying solid modeling tools.

Although the important role that feature libraries can play in CAD/CAM systems was recognized years ago, [29], not many works have addressed the problem of defining and using feature libraries explicitly. Papers explaining a rationale in support of the concepts proposed include the following.

Luby *et al.* present in [21] a system that uses two types of features: *macro*-features and *co*-features. Macro-features are classes of geometric forms such as boxes, U-channels, L-brackets, and slabs. Co-features are attachments or details which can be added to macro-features, such as holes, bosses and ribs.

The authors state that modules are available that let the designer add, create, modify and delete both macro- and co-features. How such modules might work, however, is not explained.

In [30, 31, 32], Shah *et al.* present the ASU shell. It is a testbed for rapid prototyping of feature based applications. The library of generic features is organized in the form of a list of properties. Each feature has a feature type identifier, a name, a list of generic, compatible features, and a solid representation. The solid representation of a form feature is a CSG tree. More recently, the testbed has evolved towards a multi-layered programming interface; [28]. Roughly speaking, a neutral functional view of geometric modeling services is implemented. Using object-oriented programming, the functional view is interfaced with Parasolid and ACIS. The programming layers are extensible.

Nielsen *et al*, [24] report on an experimental design-with-features system for thin-walled components that enables the user to extend its usefulness by defining feature-form types.

Laakko and Mäntylä, [17, 18, 19], describe a language to represent features embedded in Common Lisp. The feature models are organized as a structure of Lisp frames. Frames model two different types of features: *features classes* which are templates that store generic information, and *feature instances* that store specific information belonging to individual features. Feature classes are organized by a taxonomy and use inheritance. A feature model is a list of several feature instances.

In [10], Duan *et al.* report on a solid modeling tool for a feature-based design and manufacture system. The authors claim that, in their system, designers are free to create any kind of feature. Furthermore, users can construct their own feature libraries dedicated to specific applications. Unfortunately, the paper does not give any details about how these goals are achieved.

De Kraker *et al.* in [8] and Dohmen *et al.* in [9] report on the specification of a feature language developed at Delft University of Technology. Features are specified using predefined types in the object-oriented, imperative programming language LOOKS. Therefore, the feature library is a library of LOOKS procedures and defining a new feature means to write new code for it. In [7], maintenance of different feature views is discussed. The geometry representations are supported by an underlying cellular model. Feature views are declared along with permitted editing methods and validity constraints. Roughly speaking, declaring a view is analogous to defining feature classes. Since multiple, overlapping views are permitted, features must be located by feature identification algorithms since editing, through different feature views, may change or delete a different feature view.

Recently, Middleditch and Reade described in [23] a formal definition for geometric features focusing just in geometric aspects. The authors claim that their definitions can be used as basis of a suite of functions to support feature-based modelling.

In contrast to a programming approach such as [9], we propose in this paper the use of a data representation for user defined features that extends the Erep approach of [15, 5]. We explain how the data is to be interpreted semantically, but without committing to specific algorithms. In particular, the representation is not an extension of a CAD system API, or a programming language.

Maintaining neutrality, we approach user-defined feature representations from this perspective because it facilitates defining feature data models that can be used to exchange feature data between different CAD systems. This approach necessitates paying careful attention to providing an unambiguous semantics of the data representation. The semantics is procedural, and a number of detail issues, such as how to solve constraint systems that combine spatial geometric and equational algebraic constraints, have been deferred. We do explain in some detail, however, the workings of topological attributes that are used to validate specific deployment of user-defined features.

3 The Erep Framework

The Erep work, begun with [15], provides a basis for expressing user-defined features. We briefly review some of the key aspects of the work.

Erep means "editable representation." It is a neutral format for expressing form features and constraints, and we have implemented it as a CAD system that has the feel of Pro/Engineer. By neutral we mean that the (textual) representation is not committed to a core solid modeling system. The implementation uses ACIS as engine that provides boundary evaluations of solids. It is driven by an Erep interpreter that, with help of a constraint solver, implements constraint-based generative design using a graphical user interface. The communication between the GUI, the constraint solver, and the Erep interpreter is textual Erep.

The representation of a part design, in Erep, is a generative feature description. Three primary types of features are distinguished: *Generated features*, *modifying features*, and *datum features*. We describe how these features work.

Briefly, a generated feature creates geometry from a cross section and a set of attributes and constraints. The cross section is described using variational geometric and dimensional constraints. It is placed with respect to the existing geometry in 3-space by a sketching plane that can be the supporting plane of a face or a datum plane that is positioned by spatial constraints. Projected elements of the prior geometry can be used in the constraint schema and serve as one device to position the cross section in the sketching plane. The cross section generates solid geometry by sweeping (extrusion, revolution, or general sweep). This solid geometry is used to modify the prior existing geometry by making a cut (material subtraction) or a protrusion (material addition). The extent of the cut or protrusion is governed by attributes that reference the prior geometry. By using "blind" dimensions, normal CSG operations can be obtained. By using "extent" attributes such as *from-to* or *through-next*, more complicated shape modifications become possible.

A modifying feature is a *chamfer* or a *round* that locally modifies a set of vertices and edges. It is described by identifying the affected edges and vertices and specifying a set of attributes such as the radius of the round.

A datum feature is a construction point, line or plane. Any number of these features can be placed using sequential spatial constraints. Their primary purpose is to provide a graphical, intuitive way of orienting and positioning generated features without difficult coordinate system computations.

Erep is, to a large part, a generative representation that describes how to construct the solid shape. This procedural orientation makes it suitable to express a procedural description of user-defined features using the same mechanisms already implemented in the basic Erep system. The simplicity of visually defining Erep designs is a further reason why we adopted the Erep approach to UDFs.

There are a number of key technical aspects that the Erep approach has posed and which were addressed in a number of papers. The *feature attachment problem* is the problem of devising an unambiguous semantics for attachment attributes such as *cut through to the next face* or *make a protrusion from this face to that face*. In ordinary design situations, the unambiguous interpretation of such attachment attributes is rather straightforward. However, as demonstrated in [5], a number of issues arise that leave room for debate, such as the relationship between what the end-user may call a face, for the purpose of attachment, and how it might be mapped to the faces of the underlying boundary representation.

The subject of variational constraints poses technical issues in implementing Ereps, and in a number of papers we have explored many aspects of constraint solving. A particular issue that is under-appreciated is the mathematical fact that a well-constrained problem may have multiple solutions, and that there have to be rules by which a constraint solver selects the solution that is intended by the user. It remains an open problem how to communicate multiple solutions to a user, and by which means the user would navigate conveniently among them, assuming that he or she should not be required to have a deep understanding of the underlying mathematics.

A third technical problem of substantial technical depth is the *persistent naming problem*. Roughly speaking, the procedural description of a design in Erep describes not only a specific part, but an entire class whose members are obtained by varying dimensions and parameters. Since the Erep feature descriptions reference elements of the prior geometry, some of which may be artifacts of feature collision and so do not exist in any form in the description of the prior features, we need a way to identify the referenced structures, and this identification has to remain invariant over the class. In [2, 4], these issues were discussed and addressed. Other work on the subject includes [16, 20].

Persistent naming raises the question of what, in semantic terms, is a variational class. A persistent naming algorithm provides a procedural semantic. Descriptive, nonprocedural attempts at giving a rigorous semantics include [33].

4 User-Defined Features

Standard features are predefined by the CAD system. Common examples include holes, ribs and slots. Those features are supplemented by some mechanisms to define simple features on-the-fly, such as profiled extrusions and revolutions, as well as modifying features such as blends, rounds and chamfers. Standard features may be all that the CAD system provides. Although sufficient for many applications in principle, this core vocabulary is not necessarily convenient and should be supplemented by customized features defined by the user. Such a *user-defined feature*, or UDF, may be specific to an application and therefore complement the core feature vocabulary already provided by the CAD system.

A particular feature is used repeatedly when designing parts. Each use requires placing the feature in a particular locale and, in many cases, accounting for different feature dimensions and parameters. This varied use of a feature should be made as convenient as possible for user-defined features. Moreover, since different applications have different feature needs, it is natural to archive sets of predefined features in a library. The features so archived may be either standard features or user-defined features.

4.1 Requirements

We will distinguish the definition of a user-defined feature, or UDF, and the use of a user-defined feature. The *UDF definition* is a design process that creates the feature prototype. A *UDF use* is the instantiation of a particular UDF prototype in a concrete design in progress. We impose the following requirements:

1. A UDF can be defined using standard features that the CAD system already provides and/or other, previously defined UDFs.

2. The end-user must be able to place and orient the UDF in the part being designed by means of geometric gestures, using standard insertion procedures. Geometric entities of the design to which the UDF is attached may be referred to.

3. User-defined features must provide definitional mechanisms to check the validity of instantiation and attachment.

4. User-defined features must support nongeometric attributes and variational constraints.

The first requirement ensures that implementing UDFs requires minimal enhancements of the CAD system. A UDF built from one standard feature alone will be called *simple*. UDFs that are not simple will be called *compound* UDFs.

The second requirement ensures that the design process remains visual and does not change fundamentally.

Features carry information that relates to design considerations other than the creation of a net shape. A comprehensive formalization of this additional information remains open. So, we advocate a conservative set of attributes and evaluation mechanisms that support some of the nongeometric design information and functional considerations, and discuss how to inform users when the use of a UDF is inconsistent with the guidelines that were incorporated into its definition.

Since parametric and variational design is an accepted paradigm, we believe it is paramount to devise a UDF structure that supports these concepts. Therefore, we concentrate on a variational notion of user-defined features. Other notions might include importing an arbitrary shape as feature, such as the "general material removal feature" of STEP AP 224. We do not consider such features here because their use and inclusion does not support parametric design. Subject to such limitations, they could be added without difficulty to this, more general, framework.

The *functionality* of a part is its behavior, [35, 37]. Consider a connecting rod of a piston engine. The rod's function is to transmit the stroke from the piston to the crankshaft. Its functionality is given by the maximum force it can transmit. Functional attributes express functionality and can be captured by means of constraints that represent interrelationships as well as interdependencies between design data and shape data. The constraints include geometric constraints as well as engineering constraints expressed by equations and relationships.

Geometric dimensioning and tolerancing is used to define the limits of acceptable geometry. Dimensioning specifies the nominal (ideal) size of geometry. Tolerancing specifies the limits within which a nominal dimension is permitted to vary. Nominal dimensions can be represented by valuated and symbolic dimensions. We can attach to each nominal dimension an upper and lower limit. This way both 'worst case' and statistical tolerance-analysis techniques, [26], can be applied downstream.

4.2 Shape Creation

We adopt the basic definitional mechanism of the Erep work for the purpose of defining three-dimensional volumes: sweep a closed planar cross section profile along a given trajectory according to a well-defined extent semantics plus a set of attributes. Extrusions and revolutions are special cases. Throughout, we assume closed profiles.

Attributes provide additional information attached to features or elements of features that capture in part the design intent and engineering significance. Among the possible attributes we find topological, functional and tolerancing attributes, textual attributes and user-defined attributes. Minimal support of these attributes in a variational design environment requires a persistent naming schema; [2, 4, 16].

We define a *user-defined* feature informally as a self-contained, parametric, geometric object consisting of

1. a set of generated features, datum features, and modifying features,
2. a set of imported user-defined features,
3. a set of constraints,
4. a set of attributes, and
5. an interface definition.

The definition of a specific UDF prototype begins with the definer importing a previously defined UDF or using as first feature component a standard feature of the CAD system. The UDF definition continues, as the definer sees fit, with the addition of other UDFs or features as components. Constraints and attributes are added throughout. Finally, an interface for the UDF is defined that governs the UDF use and attachment process upon instantiation.

In the definition of the feature components, as well as in the definition of the interface of the UDF, equations may be used to express relationships that must be satisfied. Such equations may be used to determine parameter values through computation. It is also possible to define inequality relations. They are used for validity determination only. For instance, inequalities may be used to define valid ranges of values of parameters and dimensions.

When defining the UDF, the definer may associate specific attributes with elements of the feature components. We are especially interested in topological attributes associated with surface elements or volume elements of the UDF. Such attributes might stipulate that a specific face must be part of the boundary of the part for which the UDF is used. For example, associating with the sides of a channel the attribute *must be boundary* ensures that the channel cannot be used as a step.

Two mechanisms exist for encapsulating a UDF. First, the interface specifies which parameters and variables are to be supplied externally upon instantiating the UDF. The order of instantiation is not fixed as we discuss later. Second, equations and geometric constraints define how other variables and attributes are valuated from the interface information, without explicit user action.

Properly encapsulated, a UDF is a meaningful structure that can be treated as a single unit even though it may have many components. UDFs constitute a useful mechanism for customizing a CAD system to specific application needs without the need to reimplement parts of the CAD system. As we will show, UDFs can be implemented simply, yet they offer great flexibility.

5 Topological Attribute Definition, Maintenance and Use

Topological attributes on feature elements are attractive because they can be supported by many CAD system kernels and are useful for defining validity conditions of features. Specifically, the kernel operations of CAD systems will call on the attribute maintenance mechanism in the following four situations:

1. A face, edge or vertex has been created.
2. A face, edge or vertex has been deleted.
3. A face or edge has been subdivided.
4. Two vertices, edges or faces have been merged.

The following topological attributes can therefore be supported *on-line*, that is, the associated actions can be triggered as soon as the corresponding event takes place in the CAD kernel operation, since they are boundary based.

1. *Must be boundary (MBB)*: The attribute requires that the feature's shape element becomes part of the boundary, possibly subdivided, and that the resulting boundary element(s) should not be deleted. If the boundary element is deleted, then an error is signaled.

2. *Must not be boundary (MNB)*: The attribute requires that the feature's shape element should not become part of the boundary, in any part. If a (subdivided part of) the shape element becomes boundary, then the attachment operation fails.

3. *Conditional if on boundary (CIB)*: The attribute requires that the associated shape element should become part of the boundary, possibly subdivided. If not, the feature attachment is canceled.

Maintenance of these attributes requires rules for split and merges. By default, the split elements inherit the attributes of the parent shape element. By default, merging two shape elements with different boundary-based topological attributes is an error. The default rules for merging can be overridden with explicit, user-provided rules.

There are two volume-based topological attributes. Since they are volume-based, they either are attached to the feature volume itself, or else they are associated with volumes for which construction geometry provides additional boundaries.

1. *Must be void (MBV)*: The attributed volume must have an empty intersection with the solid interior, after feature attachment, and throughout the lifetime of the feature.

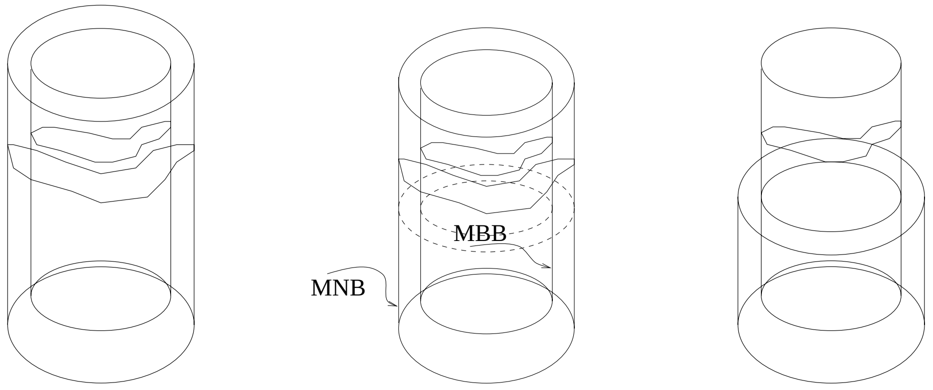

Figure 1: Blind hole feature definition. See the narrative for details.

2. *Must be material (MBM)*: The attributed volume must be contained in the solid interior, after feature attachment, and throughout the lifetime of the feature.

For example, consider a feature definition in which a hole is defined whose interior has the attribute MBV. Around the hole an annular volume has been defined and attributed MBM. Intuitively, the MBM volume stipulates a minimum material condition for the hole. However, the outer boundary of the MBM volume would introduce interior faces and edges into the resulting solid model.

We do not advocate creating a completely general nonmanifold solid model and to maintain validity of all attributes on-line, as is done for example by [7]. The interior volumes so introduced would become subdivided by overlapping volumes associated with other features nearby, and for complex parts we anticipate a combinatorial explosion of interior cells, raising many scaling and performance issues. Instead, we limit nonmanifold structures to surface subdivisions, for example, to maintain face regions that must not be subdivided.

Volumes carrying attributes and interior nonmanifold faces and edges are maintained separately for each feature. They are not incorporated into the solid model of the part. This implies that the validity of these topological constraints must be verified off-line by intersecting the separate structures with the solid model, when requested to do so by the user. Since the volumes are maintained separately, no rules for splits and merges are needed. The conceptual separation into on-line and off-line attribute verification balances efficiency requirements for large-scale design with the support of valid feature use.

A realistic use of features so attributed requires topological rules for matching boundary regions and establishing feature volumes at attachment time, so that sculptured solid boundaries can be accounted for. We explain with an example.

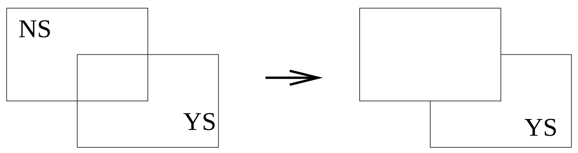

Figure 2: Merge rule for overlapping between YS and NS faces

Example. We want to define a blind hole with a minimum material condition that must be respected. After this hole feature has been placed, subsequent design operations must not invalidate the feature. Figure 1, *left*, is a first attempt at a definition. The inner (void) volume has the attribute MBV, the cup-shaped outer volume the attribute MBM. The two curves indicate boundary intersections with the prior geometry when attaching the feature. The problem here is to establish how the MBM volume is allowed to be clipped while still having a valid feature use. How closely might the intersection curves approach the bottom of the hole before the hole's function is compromised?

Figure 1, *middle*, solves the problem by subdividing the cylindrical faces by the dashed curves. The dashed curves establish a boundary below which the attached feature must not intersect the boundary. The corresponding lower areas of the MBM volume boundary have the attributes MBB on the inside surface and MNB on the outside surface. Any boundary intersection below the dashed curves compromises the feature use.

The inner cylindrical faces becomes part of the solid boundary, and its MBB attribute can therefore be maintained on-line. The outer cylindrical face would create an interior subdivision of the volume, after the feature has been attached. To maintain it on-line risks a fragmentation of the part volume into numerous cells. The definition of Figure 1, *right*, addresses the issue as follows. First, the MBM volume is reduced to the minimum extent permitted. Defined as a separate volume for the feature, it is maintained as a separate geometric structure. When asked to verify the feature's topological attributes, this volume is re-intersected with the part. If it is not subdivided by the intersection, then the MBM attribute is satisfied, otherwise it has been compromised. □

For the boundary-based attributes MBB, MNB, and CIB we have allowed subdivision of the faces or edges so attributed. We may want to preserve a face or an edge in its entirety, and can do so with the attributes *may be subdivided (YS)* and *may not be subdivided (NS)*. For instance, the bottom face of the blind hole would have the attributes MBB and NS, so that no further subdivision is allowed. The YS and NS attributes have the following default maintenance rules: A NS face or edge may not be split. When merging NS and YS, an internal boundary of the face is maintained that preserves the NS attribute. Overlaps become NS, as shown in Figure 2.

6 Implementation

We assume that the CAD system provides generated features, datum features, and modifying features as explained in Section 3. All these features will be referred to as *standard features.*

Many CAD systems allow open profiles for the generation of cuts and protrusions. Conceptually, we can think of a semantics for open profiles as follows: The open profile is swept, as a closed profile would be, resulting in a surface sheet. The sheet is joined with the existing solid boundary, resulting in the creation of a nonmanifold boundary with new intersection edges and vertices. By a boundary traversal, the combined surface structure is then trimmed to the intended solid boundary.

This conceptually easy process raises many questions. For example, what if the surface sheet does not meet or intersect the solid boundary in some places? Is it always clear on which side of the surface sheet we should add material or subtract it? What if the surface sheet and the solid boundary to which it attaches does not always completely enclose a volume? To endow the process of solid geometry creation from open profiles with a rigorous foundation is beyond the scope of this paper. Therefore, we assume closed profiles throughout.

6.1 User-Defined Features Definition

We split the definition of a UDF into two subprocesses. First, the geometry is defined. Then, component attributes and equational constraints between variables are defined.

6.1.1 Geometry Definition

UDFs definitions are built by defining every component in sequence. The first component so defined is called the *primary* component. The primary component makes no references to elements of any other component. If the UDF definition does not utilize a previously defined UDF, then the primary component must be a standard feature provided by the CAD system, for instance a datum feature. The primary component may also be a previously defined UDF. Subsequent components are defined and placed with respect to the prior components.[1]

Since a feature is either simple or composed of component features, UDFs have a natural hierarchical structure. This structure can be expressed logically as an acyclic directed graph where each graph node is a component and the edges are direct dependencies between them, oriented from the component that

[1]As remarked before, one could allow a general solid as primary feature. This solid would then be "dead geometry," that could be changed only by additional feature components, not by changing parameters or constraints on it.

makes the reference to the component in which the referenced element is defined. The primary component is at the root of the graph.

As explained before, there are two basic ways by which the extent of geometry creation is determined: blind extent and delimited extent. Both methods can be used when defining a UDF.

6.1.2 Attribute Definition

We may associate attributes with feature elements. Such attributes may be topological attributes, functional and tolerancing attributes, textual attributes and user-defined attributes.

In his thesis [3], Charlesworth uses topological attributes to modulate Boolean operations, in an effort to define a symmetric version of feature attachment. A different approach to validate feature semantics is reported by Mandorlini *et al* in [22]. Here, features are standard blind holes, slots, etc. and each type of feature has standard topological constraints associated with the geometric elements that define its boundary. After performing a feature attachment, specific rules are triggered to check whether topological constraints still hold. We use topological attributes also to support validity determination when attaching a feature, as explained before.

6.1.3 Definition of Constraints

Other aspects of design intent and functionality can be captured using constraints that are defined as mathematical equations between variables the design depends on. Variables can represent geometric dimensions as well as technological parameters and engineering variables. Constraints are defined as textual information. The UDF designer is responsible for correctly defining the constraints that apply.

Note that equational constraints exist in combination with geometric constraints. To support such constraints in the CAD system, constraint solvers must be available that decompose the equations and coordinate them with the geometric constraint solving processes. In particular, this integration can be accomplished in a variational sense such that the distinction becomes blurred between which parameters and dimension values are to be supplied explicitly and which ones are to be inferred. This generality can be reflected in the interface definition and allows, when properly executed, much flexibility in the order of attaching a UDF.

6.2 Encapsulation

Encapsulation requires a well-defined functionality and a complete and well-defined interface. Functionality is expressed by the geometry, attributes and equations. The feature definer is responsible for the proper definition. In this task, he is assisted by information hiding through encapsulation. Moreover, encapsulation keeps representational details hidden from the end-user.

The interface of a UDF provides the feature view by which the user generates UDF instances as needed. The interface consists of a name that identifies the feature uniquely, and a set of symbolic parameters.

The parameters in the interface of a UDF feature are variables and datums tags. When instantiated, variables take scalar values, for example real numbers. Datums tags, on the other hand, will refer to geometric elements of the part the UDF is being attached to, after instantiation. They refer to either points, axes, planes or faces. For simplicity, in what follows we shall refer both to variable values and to datum links as *values*.

There are two different categories of parameters: *Independent parameters* and *dependent parameters*. Independent parameters are those whose value must always be provided externally. Dependent parameters are those that are constrained by other entities in the user-defined feature. Their actual value is either provided externally or else derived from the equations and constraints that are solved during the attachment process, depending on the concrete sequence of attachment operations carried out. When appropriately supported by an integrated constraint solver, the distinction between those parameter types need not be made explicitly and is inferred from the particular manner in which the attachment process proceeds.

An important issue for encapsulation is how to include features into the UDF that are already defined. There are two basic ways to do this: The first way to include another UDF is by keeping symbolic links to the the feature and the elements referenced in it. We call such components *dynamic*. Dynamic components offer flexibility because the external feature that will be linked as component at instantiation time is the one that is actually present in the library, at that time. The second way to include another UDF is to generate a local instantiation of the external feature definition. Such *static* components incur less overhead in the feature library management and facilitate validity checking.

6.3 Inheritance

UDFs are built from standard features of the CAD system and/or from other UDFs. As components of the new UDF, they carry with them their own attributes. This provides a natural inheritance for components in the *is-a* sense.

Without additional mechanisms, this inheritance style is too rigid. We provide therefore for selective inheritance. Consider a UDF component in which some of the attributes are changed. This is effected when importing the component, by modifying those attributes. We require in this case that the UDF be a static component. For example, a blind extrusion extent might be so converted to a *from-to* delimited extrusion extent.

It is also possible to impose additional constraints and attributes on the UDF that is imported as component. This method of selectively refining the UDF component is especially appropriate for dynamic feature components.

We could add a scope mechanism to structure inheritance hierarchically. Such mechanisms would be easy to add, but may not be needed. Multiple inheritance does not come up in our design of UDFs since previously defined UDFs always become UDF components.

6.4 Feature Attachment

Feature attachment is the process by which the user includes a new feature in the part under design. This process must be intuitive and convenient. We consider feature attachment as a two-step process: attachment definition and attachment evaluation.

For the attachment definition, the UDF to be attached is selected by its name. Then, the user provides interactively the minimal and sufficient information needed to define the actual size, position, orientation and extent of the UDF instance. This information is passed to the UDF through the parameters of the interface.

After retrieving the UDF from the library, the system displays a feature template generated with default values. The user carries out the attachment definition by giving values for variables and links for datums. The latter can be done visually. Once all required information has been supplied, feature attachment evaluation proceeds automatically. We require that the order in which the UDF components are evaluated and placed respect the precedence ordering of the component graph (Section 6.1.1).

Depending on the interface definition, complex dependencies on parameters and constraints may be created. In simpler situations, every parameter valuation or datum pair mating is propagated in order to determine which other dependent parameters become known according to the equations and constraints in the definition of the UDF. The process of mating geometric elements and assigning values to dependent variables goes on until the attachment is fully defined.

Since no specific attachment sequence is enforced, users may define values for variables and designate mating pairs at their convenience in any order. The only restriction is that independent datums must be linked through explicit mating operations and that, at some point, values for independent variables must be given explicitly. In the attachment evaluation, the actual geometry is computed according to the values given in the attachment definition. It is carried out by the system.

6.5 Validity

UDFs validation can be separated into two steps: First, when the feature is defined, there should be a preliminary definition validation. Then, when the UDF is attached to a part or to another feature, there must be a validation for the attachment operation.

6.5.1 Definition Validation

Once a UDF has been defined and before it is stored in the features library, a process to check definition validity should be available. Because of the potential shape variability, however, such checks do not exempt the system from checking the validity of the instantiation. Moreover, specific constraints such as some topological attributes depend on the interaction of the instantiation of the UDF with the part in which it is used.

Some validation checks are trivial, including syntactic correctness of constraints, coherence of parameter range definitions, and so on. More difficult is to verify that the UDF will instantiate and is properly constrained. Here we restrict to evaluating the UDF with the default values provided by the definer. While this check does not guarantee that the UDF will instantiate correctly for other values chosen by the user, it will preclude a variety of possible errors in the definition.

6.5.2 Attachment Validation

After a UDF has been attached to a part, some tests should ascertain validity. Routine checks include the verification of dimensional, equality, and inequality constraints; the absence of self-intersection of profiles; and so on.

The verification of the topological attributes is an important check that establishes the semantic validity of the feature use. For example, assume that we want to guarantee that a channel should not become a step inadvertently. If the lateral position of the channel on a face is given by a distance parameter from an edge, the value of the parameter might be used to reason that the channel bounds material on both sides. However, this is not sufficient to guarantee validity. Using a *must be boundary* attribute for both channel walls is a more reliable validation test.

It is desirable to revalidate a feature use as the design progresses. For example, we may install a channel on a block properly, but may then invalidate this feature with a subsequent profiled cut that compromises one of the channel walls. Thus, we should re-evaluate some of the feature constraints. Re-evaluation can be the consequence of editing a feature, or of adding or deleting another feature. For efficiency, re-evaluation should be triggered or bypassed by tracking feature collision, for example using bounding boxes and maintaining dynamic box trees;

e.g., [14] Chapter 3.

7 Erep Representation

We exploit the extensibility of the Erep language defined in [15] because it already provides naturally many of the mechanisms needed to support UDFs such as persistent naming and variational constraint solving.

A UDF Erep has two parts: A *header* and a *body.* The header encompasses all the information that defines the UDF feature interface and the equations and relationships that variables in the feature design must fulfill. The body contains the list of component features from which the UDF feature is built.

In general, attributes can be represented in textual form, properly grouped under syntactic headers and descriptors. Since we are especially interested in topological attributes we discuss how they can be included in the Erep.

Topological attributes that must be attached to boundary elements in the feature can be associated with the topological element in the cross section profile that will generate them. This form of attribute attachment can be handled appropriately by minor changes to the persistent naming schema used by the Erep system. Topological attributes that must be attached to volume elements require naming volumetric elements. We attach the attributes to loops of the cross section profile that generate volume.

As described in Section 6.1.1, a UDF has a natural hierarchical structure defined by the fact that each feature component is built with respect to already existing feature components. The hierarchy is rooted in the primary feature component.

Once actual values have been given for the set of parameters that define the shape and position of the UDF instance, the evaluation of the Erep representation can be performed by evaluating first the primary feature and then evaluating each feature component according to the hierarchy in the UDF.

8 Case Study

We illustrate some of the proposed ideas with the following UDF design example. Consider a connecting rod in an internal combustion engine. An important function assigned to the connecting rod is to help lubricate a number of the engine parts. For example, the rod should force oil to the piston pin bushings. To perform this function, connecting rods have several drilled lubricant passages as illustrated in Figure 3.[2]

[2]This design is from a diesel engine design studied by the ARPA MADE program some years ago.

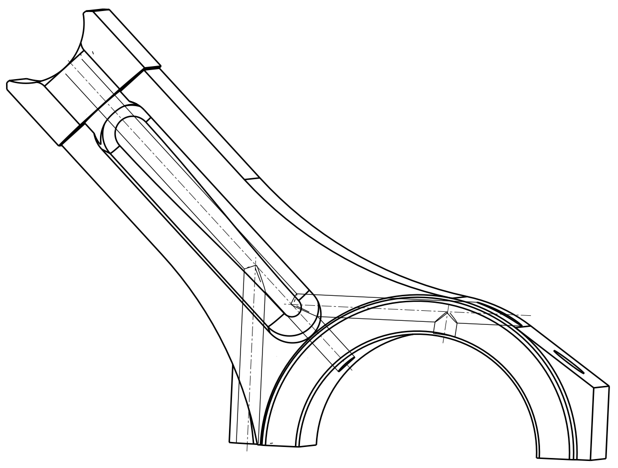

Figure 3: Connecting rod.

The set of lubricating passages in a connecting rod has a well defined functionality. If a particular lubricating system is assumed, the number of passages, the type of each passage as well as the topological relationships between the passages will be fixed. To realize an instance of such a lubricating system, one would only have to define the geometric dimensions of each component passage and place the passages with respect to the rod. Clearly, such a set of lubricating passages fits the idea of UDFs.

We begin with an explanation of how to define the UDF prototype. Then we will show how the end-user can instantiate and attach the UDF prototype in a particular connecting rod design.

8.1 Definition of the User-Defined Feature

The set of lubricating passages of the rod in Figure 3 can be represented abstractly by the diagram in Figure 4. The set consists of four cylindrical passages with coplanar axes. The passage labeled l_1 is the one to be drilled into the connecting rod shaft.

First, we design the passage labeled l_1. We begin by defining the datum planes shown in Figure 5. They are a natural choice for features with an extruded extent semantics and should probably be predefined by the system. Datum planes DP_2 and DP_3 will be used as references to define the passage. DP_2 is defined as a plane normal to DP_1, through the y-axis, and DP_3 is defined as a plane normal to DP_1 through the z-axis. Datum plane DP_4 will delimit the passage extent and is defined as an offset of DP_1 at a distance l_1.

Once the datum frame has been set up, we sketch the passage profile on the sketching plane DP_1 and properly annotate the sketch with geometric con-

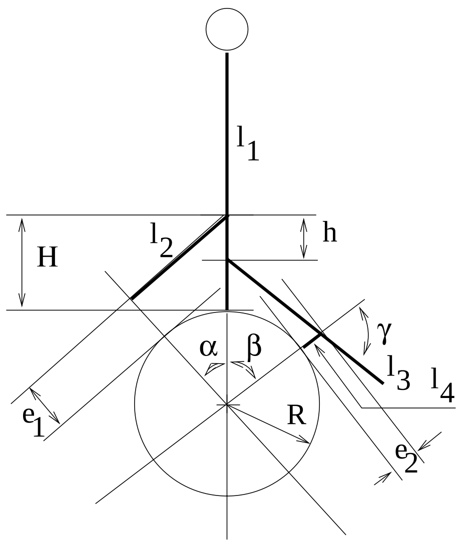

Figure 4: Lubrication passages system drilled in a connecting rod with constraint schema.

straints. See Figure 6. Then, we define the passage extent by assigning the *from* face explicit semantics to the sketching plane and the *to* face explicit semantics to datum plane DP_4. Since the sketching plane now is involved in an extent that is not blind, a datum plane DP_{100}, a copy of the sketching plane, is created by the system to support the *from* face. This allows us to place the sketching plane independently from the from and to extent planes. Figure 7 shows the resulting partially defined UDF.

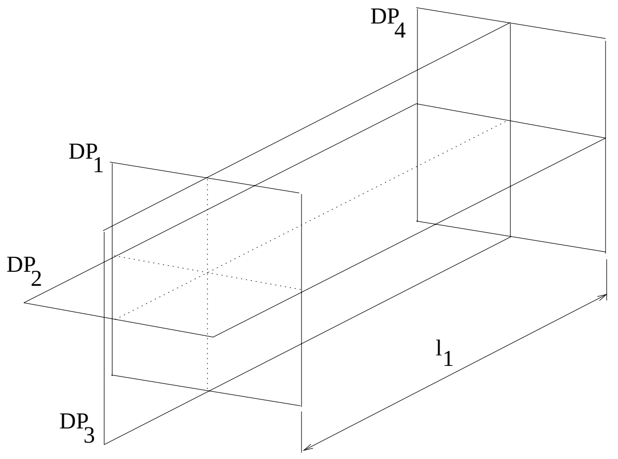

Figure 5: Datums for a generic passage.

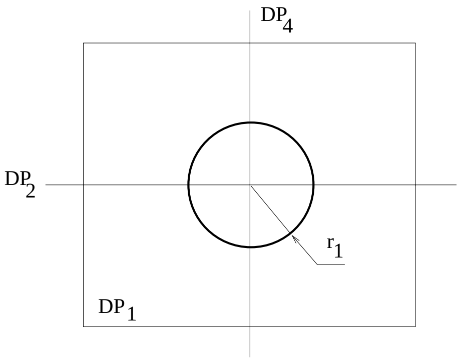

Figure 6: Profile and constraints for the first passage.

Now we define another basic component in the UDF, the passage labeled l_2. This is illustrated in Figure 8. As before, we embed the definition into the frame of the four datum planes shown in Figure 5. To properly place this frame we can use spatial variational constraints, or else define auxiliary datums with parametric spatial constraints. We define datum plane DP_6 as offset of DP_1 at distance $-R$. Now we can define the datum axis DA_1 as the intersection of plane DP_6 and DP_3. To attach the framework for the second passage, we require datum plane DP_7 to go through DA_1 at angle $90 - \alpha$ with DP_3, and DP_8 to go through DA_1 at an angle of 90 with DP_7. Then DP_9 is a parallel offset of DP_7 at distance $R + e_1$.

The profile of the new passage is defined on the sketching plane DP_8. Figure 9 shows the first passage projected onto the sketching plane, the profile defined, and the geometric constraints. To complete the geometry we define the extent of the attached passage. We assign the *from* face explicit semantics to

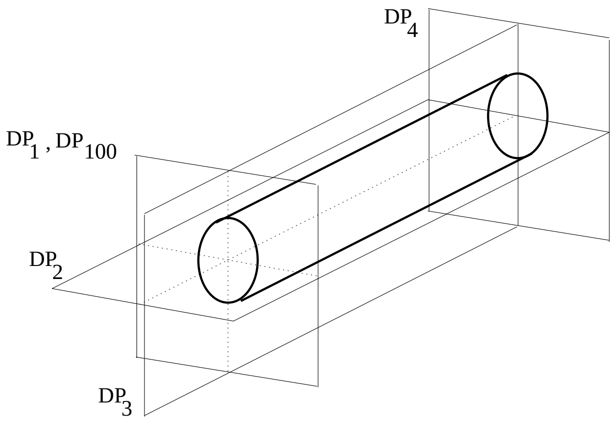

Figure 7: User defined generic passage.

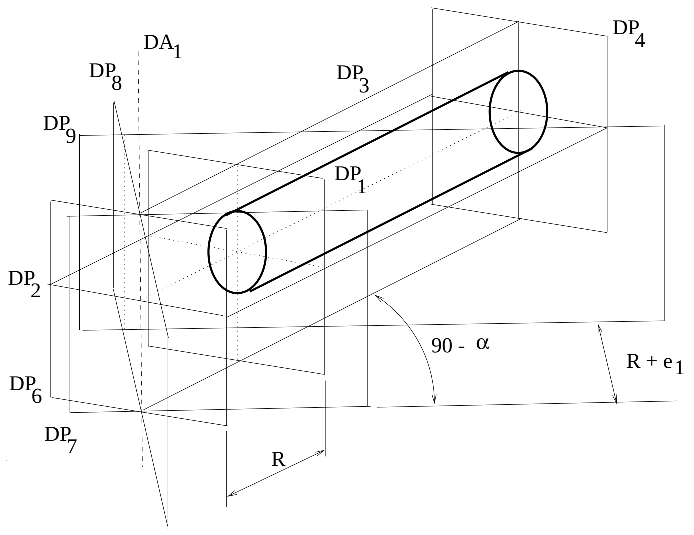

Figure 8: Datums for the second passage.

datum plane DP_8. As before, since the sketching plane now is involved in an extent that is not blind, a copy of the sketching plane, not shown in Figure 9, is created by the system to support the *from* face. The *to* face is assigned to the cylinder of the first passage. Figure 10 shows the resulting UDF. We continue in this manner until all feature components have been placed. Figure 11 shows the final UDF.

Now we can define equations between the variables in the new compound feature. For example,

$$r_1 \quad = \quad 0.25 \times t$$

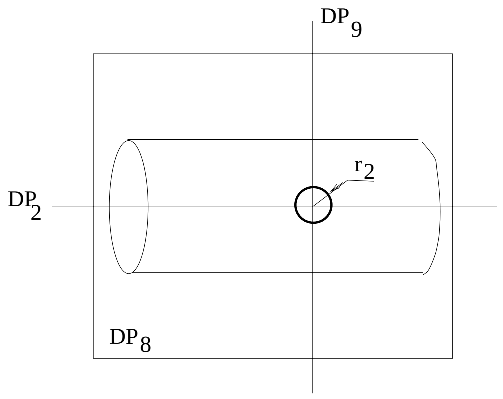

Figure 9: Profile and constraints that define the cross section of the attached passage.

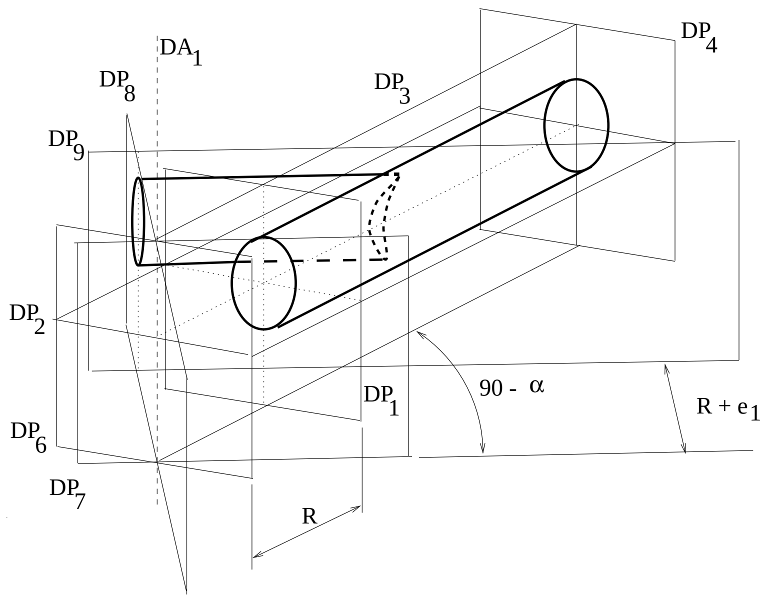

Figure 10: User-defined feature after attaching a second passage.

$$\begin{aligned} r_2 &= r_1 \\ r_3 &= r_1 \end{aligned}$$

Where t is the rod shaft thickness. We also can define some relationships, such as

$$\begin{aligned} H/3 &< h \\ 2H/3 &> h \end{aligned}$$

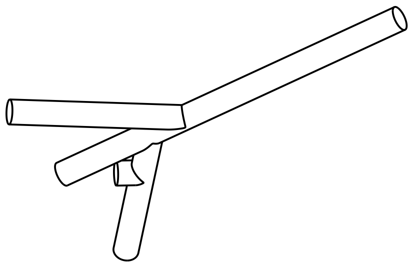

Figure 11: User-defined feature.

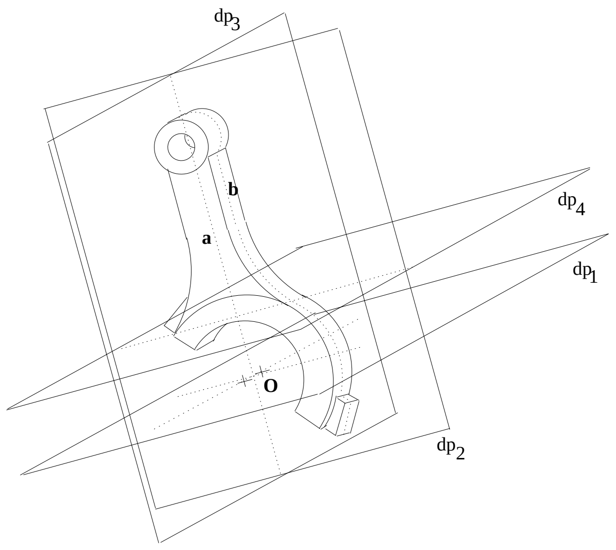

Figure 12: Connecting rod under design to which a lubricant passage system must be attached.

or others involving variables that represent geometries in the lubricating system or technological knowledge. Here, H and h are as in Figure 4.

Datum planes DP_1, DP_2, and DP_3 could be taken to be independent parameters in the attachment process, but since the user has a choice which parameters to valuate and which datums to link, we leave it to the constraint solver to determine dependencies during attachment.

Finally we assign topological attributes to boundary elements or to the feature volumes. An example of boundary attribute is declaring that the lateral, cylindrical faces of the holes must always contribute to the boundary of the part. An example of volume attribute is to define that the holes must be always void, that is, no other construction can add material to the void region generated by the holes in the part. As discussed in Section 5, this may involve a subdivision of the cylinders.

8.2 Attachment

Assume that the connecting rod being designed is as shown in Figure 12 and that we want to attach to it the lubrication passage system defined before as a UDF. Moreover, we assume that we attach first the passage to be drilled in the rod shaft.

We start the attachment by defining in the connecting rod the positions of the basic datum planes dp_1, dp_2 (this is the drawing plane) and dp_3. See Figure 12. We define dp_2 as offset from the side face **a** of the rod at a distance equal to half of the shaft's thickness. Datum plane dp_3 is an offset from the

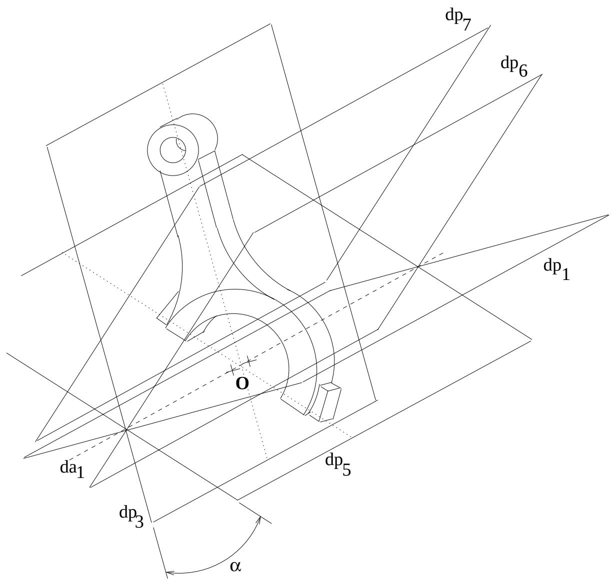

Figure 13: Attachment process.

side face **b** of the rod at a distance equal half of the rod shaft width. dp_1 is a plane perpendicular to both dp_2 and dp_3 through the center **O** of the rod big end. Now we define datum plane dp_4 is an offset from dp_1 at a distance equal to the rod big end internal radius R.

The lubricant passage to be drilled in the rod shaft is attached simply by mating dp_1, dp_2, dp_3, and dp_4 on the part with DP_6, DP_2, DP_3 and DP_1 in the UDF.

Since the rod thickness is already fixed, the UDF parameter r_1, and therefore the cross section profile, is defined. The passage extent can be fixed by assigning the *from* and *to* semantics to the inside faces of the rod's crankshaft end and piston end, respectively. Here, the datum planes DP_{100} and DP_4 in the UDF point to these faces. Note that this is not a mating process, since the terminating faces are not planar. This is why we distinguish between the planes DP_1 and DP_{100} in the UDF prototype.

Figure 13 illustrates how to position the second passage with respect to the first one. We first define datum axis da_1 as the line common to dp_1 and dp_3. Then dp_5 is a datum plane through da_1 at an angle α with dp_3. Next, we define dp_6 perpendicular to dp_5 through da_1, and dp_7 as offset of dp_6 at distance $R+e_1$. See also Figure 4. Mating the sketching plane DP_8 with dp_5 and DP_9 with dp_7 in the part fixes the position of the second passage. Alternatively, we may want to place DP_8 and DP_9 based on explicit constraints to the part. For example, we could make DP_8 coplanar with the assembling faces of the rod at the crankshaft side, thereby not valuating α directly. The valuation of the parameters r_2 and

r_3 is automatic from the constraint equations and completely determines the cross section. The passage extent can be fixed, for example, by assigning the *through next* explicit semantics to the *to* face in the UDF definition.

The attachment process proceeds until every component in the UDF has been attached to the part under design.

9 Summary

We have presented a propcedural mechanism for generating and deploying user-defined features in a feature-based design paradigm. The mechanism presented addresses customization needs in a simple, effective way. The usefulness of the mechanism relies on three basic capabilities: Use of standard tools provided by the CAD system, parameterization of UDFs, and graphical interaction.

Using standard mechanisms already available in the CAD system, the designer does not have to aqcuire new skills. Second, parameterization and constraints, we obtain a convenient way to instantiate user-defined features according to specific design requirements. Third, positioning and orientating the UDF in the part being designed by means of geometric gestures on geometric references embeds UDF attachment seamlessly into the part design process.

The Erep language provides a strong foundation for expressing user-defined features. It provides already many of the mechanisms needed to support user-defined features such as parameters, attributes, geometric and technological constraints, persistent naming, and variational constraint solving. Moreover, Erep is a textual, neutral representation, that is not committed to a core solid modeling system, and thus focuses on the required information content without the distractions of specific implementation details of the core modeling system.

Acknowledgements

This work was developed while R. Joan-Arinyo was on leave in the Department of Computer Sciences, Purdue University, partially supported by a CIRIT fellowship of the Government of Catalonia under grant 1996-BEAI-400110.

Hoffmann has been supported in part by ONR Contract N00014-96-1-0635 and by NSF grants CDA 92-23502 and CCR 95-05745.

References

[1] W.F. Bronsvoort, R. Bidarra, M. Dohmen, W. van Holland, and K.J. de Kraker. Feature modeling for concurrent engineering. In I. Horvart and T. Varady, editors, *International Symposium on Tools and Methods*

for Concurrent Engineering'96, pages 46–55. Technical University of Budapest, 29-31 May 1996. Budapest, Hungary.

[2] V. Capoyleas, X. Chen, and C.M. Hoffmann. Generic naming in generative, constraint-based design. *Computer Aided Design*, 28(1):17–26, 1996.

[3] William W. Charlesworth. *Set operations under topological constraints.* PhD thesis, Dept. of Mech. Engr., Purdue University, 1996.

[4] X. Chen and C.M. Hoffmann. On editability of feature-based design. *Computer Aided Design*, 27(12):905–914, 1995.

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

[6] D. H. Brown Associates, Port Chester, NY. *SDRC's VGX Technology*, 1997.

[7] J. de Kraker, M. Dohmen, and W. Bronsvoort. Maintaining multiple views in feature modeling. In *Proc 4th ACM Symposium on Solid Modeling and Applications*, pages 123–130, Atlanta, GA, May 14-16 1997.

[8] K.J. de Kraker, M. Dohmen, and W.F. Bronsvoort. Feature validation and conversion. In D. Roller and P. Brunet, editors, *CAD Tools for Products.* Springer Verlag, Berlin, 1996.

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

[10] W. Duan, , J. Zhou, and K. Lai. FSMT: a feature solid-modelling tool for feature-based design and manufacture. *Computer Aided Design*, 25(1):29–38, 1993.

[11] A. Ghandi and A. Myklebust. A natural language approach to feature based modeling. In *Advances in Design Automation*, volume 1 of *Computer Aided and Computational Design*, pages 69–77. ASME, 1989.

[12] D.C. Gossard, R.P. Zuffante, and H. Sakurai. Representing dimensions, tolerances, and features in MCAE systems. *IEEE Computer Graphics and Applications*, pages 51–59, March 1988.

[13] P.H. Gu, H.A. ElMaraghy, and L. Hamid. FDDL: A feature based design description language. In H.A. ElMaraghy, W.P. Seering, and D.G. Ullman, editors, *Design Theory and Methodology DTM89*, Design Technical Conferences Proceedings, pages 53–63, New York, 1989. ASME.

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

[15] 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 Modeling for Product Realization*, pages 129–164. North Holland, 1993.

[16] J. Kripac. *Topoogical ID System. A mechanism for persistently naming topological entities in history-based parametric solid models.* PhD thesis, Czech Technical University, 1993.

[17] T. Laakko and M. Mäntylä. Feature modelling by incremental feature recognition. *Computer Aided Design*, 25(8):479–492, 1992.

[18] T. Laakko and M. Mäntylä. A feature definition language for bridging solids and features. In G. Allen J. Rossignac, J. Turner, editor, *Second Symposium on Solid Modeling and Applications*, pages 333–341. ACM Press, 1993. Montreal, Canada.

[19] T. Laakko and M. Mäntylä. Incremental constraint modelling in a feature modelling system. *Computer Graphics Forum*, 15(3):367–376, 1996.

[20] R. Lequette. Considerations on topological naming. In M. Pratt, R. Sriram, and M. Wozny, editors, *Product Modeling for Computer Integrated Design and Manufacture*, pages 394–406. Chapman and Row, 1997.

[21] S.C. Luby, J.R. Dixon, and M.K. Simmons. Creating and using a featrures data base. *Computers in Mechanical Engineering*, pages 25–33, November 1986.

[22] F. Mandorlini, U. Cugini, H.E. Otto, and F. Kimura. Modeling with self validation features. In C. Hoffmann and W. Bronsvoort, editors, *Proc 4th ACM Symposium on Solid Modeling and Applications*, pages 88–96, Atlanta, GA, May 14-16 1997.

[23] A. Middledicth and C. Reade. A kernel for geometric features. In C. Hoffmann and W. Bronsvoort, editors, *Proc 4th ACM Symposium on Solid Modeling and Applications*, pages 131–140, Atlanta, GA, May 14-16 1997.

[24] E.H. Nielsen, J.R. Dixon, and G.E. Zinsmeister. Capturing and using designer intent in a design-with-features system. In *Design Theory and Methodology – DTM'91*, volume 31, pages 95–102. ASME, 1991.

[25] H.-M. Rho and D. Sheen. A part feature description model for process planning of rotational parts. In *Advances in Design Automation*, volume 1 of *Computer Aided and Computational Design*, pages 87–91. ASME, 1989.

[26] U. Roy, C.R. Liu, and T.C. Woo. Review of dimensioning and tolerancing: representation and processing. *Computer Aided design*, 23(7):466–483, September 1991.

[27] O.W. Salomons, F.J.A.M. van Houten, and H.J.J. Kals. Review of research in feature based design. *Journal of Manufacturing Systems*, 12(2):113–132, 1993.

[28] J. Shah, H. Dedhia, V. Pherwani, and S. Solkhan. Dynamic interfacing of applications to geometric modeling services via modeler neutral protocol. 1996.

[29] J.J. Shah. Feature transformations between application-specific feature spaces. *Computer-Aided Engineering Journal*, 5(6):247–255, 1988.

[30] J.J. Shah and M.T. Rogers. Expert form feature modelling shell. *Computer Aided Design*, 20(5):515–524, 1988.

[31] J.J. Shah and M.T. Rogers. Feature based modelling shell: Design and implementation. In *Proc. ASME Conf. Computers in Engineering, Vol 1*, pages 255–261, San Francisco, 1988.

[32] J.J. Shah and M.T. Rogers. A testbed for rapid prototyping of feature based applications. In J.J. Shah, M. Mäntylä, and D.S. Nau, editors, *Advances in Feature Based Manufacturing*, Manufacturing Research and Technology, 20, chapter 18, pages 423–453. Elsevier Science B.V., 1994.

[33] V. Shapiro. Necessary conditions for boundary representation variance. In *Proc. 13th Symp. on Computational Geometry*, pages 77–86, 1997.

[34] L. Solano and P. Brunet. Constructive constraint-based model for parametric CAD systmes. *Computer Aided Design*, 26(8):614–621, 1994.

[35] L.E. Taylor and M.R. Henderson. The roles of features and abstraction in mechanical design. In *Desigh Theory and Methodology – DTM'94*, volume 68, pages 131–140. ASME, 1994.

[36] J. Tikerpuu and D.G. Ullman. General feature-based frame representation for describing mechanical engineering design developed from empirical data. In *Proc. ASME Conf. Computers in Engineering, Vol 1*, pages 245–253, San Francisco, 1988.

[37] D.G. Ullman. The evolution of function and behavior during mechanical design. In *Desigh Theory and Methodology – DTM'93*, volume 53, pages 91–103. ASME, 1993.

[38] M. von Rimscha. Feature modelling and assembly modelling - a unified approach. In F.-L. Krause and H. Jansen, editors, *Advanced Geometric Modeling for Engineering Applications*, pages 203–213. Elsevier Science Puibishers B.V., 1990.