On User-Defined Features

Christoph M. Hoffmann
Robert Joan-Arinyo

Report LSI-96-57-R
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

October 25, 1996

Abstract

Feature-based design is becoming one of the basic design paradigms of CAD systems. In this paradigm the basic unit is a feature and parts are constructed from 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 to generate and use user-defined features in a feature-based design paradigm is proposed. 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.

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

*While on leave in the Department of Computer Sciences, Purdue University. Partially supported by a CIRIT fellowship of the Government of Catalonia under grant 1996BEA400110.
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 with 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 define a universal set of features would lead to a potentially unmanageable number of them 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 the need to reimplement a new CAD system or system interface.

While the feature mechanisms provided by commercial systems provide some of the needed mechanisms, they fall short in part because of the absence of compounding features. Feature definition languages such as the ASU test bed, [23], 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.

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. Moreover, the mechanisms are compatible with constraint-based design paradigms and can be implemented in a neutral way using our Erep approach [14].

2 Previous Work

There are many papers reporting on languages to represent features, [7], [8], [9] [10], [14], [16], [17]. [19], [26], [24], [25]. The motivations given to introduce these languages fall into several categories: Facilitating solid modelling, geometric reasoning, improving manufacturability. To introduce these languages fall into several categories: Facilitating solid modelling, geometric reasoning, improving manufacturability, CAD/CAM integration, consistency verification, unification
of solid modelling tools, etc.

Although the important role that features libraries can play in CAD/CAM systems was recognized some years ago, [20], very few works have addressed explicitly the problem of defining and using features libraries. Furthermore, the papers generally illustrate the concepts with simple and incomplete examples. Papers given rationals supporting the concepts exposed are an exception. Among them we can find the following.

Luby et al. present in [18] a system that builds with 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. Authors claim that modules are available which let the designer add, create, modify and delete both macro- and co-features. Constructive details are not given.

In [21, 22] and [23], Shah et al. present the ASU shell. It is a testbed for rapid prototyping of feature based applications. The library of generic fetaures is in the form of a list of properties. Each feature has a feature type identifier, a name, a list of generic, compatible features, and the solid representation. The solid representation of the form features are stored as a Constructive Solid Geometry.

In [7] Duan et al. report on a solid modelling 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 and. Furthermore, users can to organize their own features library dedicated to a specific application. Unfortunately, the paper does not give any details about how these goals are achivied.

De Kraker et al. in [5] and Dohmen et al. in [6] 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 features library is a library of LOOKS procedures and defining a new feature means to write new code for it.

3 User-Defined Features

Standard features are predefined by the CAD system. Common examples include holes and slots. Supplemented by some mechanisms to define simple features on-the-fly, such as profiled extrusions and revolutions, standard features may be all that the CAD system provides. Although sufficient for many applications, this core vocabulary is not necessarily convenient and should be supplemented by customized features defined by the user. Such user-defined features may be specific to an application and therefore complement the core feature vocabulary.

A particular feature is used repeatedly when designing parts. Each use requires placing the feature in a particular locale and, in many cases, with
different feature dimensions and parameters. This varied use of a feature should be made as convenient as possible. 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.

3.1 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 in a concrete design in progress. Clearly, the definition of the UDF should be parametric and/or variational (in the sense explained in [13]). Moreover, instantiation should preserve some or all of the definitional flexibility. Therefore, upon editing a concrete design, all UDFs that are used and have not been explicitly changed by the user can be automatically reinstantiated according to the implications of the changes. The use of a UDF in variant design should be compatible with the variational schema of the design itself. The following constitutes a minimal set of requirements on user-defined features (UDF):

- A UDF can be defined using standard features that the CAD system already provides and/or other, previously defined UDFs. This ensures that UDF implementation 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 end-user must be able to place and orient the UDF in the part being designed by means of geometric gestures and using geometric references already defined. using standard insertion procedures.

- Instantiation of the UDF must be supportive of the parametric or variational design paradigm of the CAD system.

- User-defined features must provide definitional mechanisms to check the validity of instantiation and attachment. If a UDF instance is invalid, because of incorrect instantiation or because of implied consequences of other design steps, then this invalid use should be communicated to the user so it can be corrected. More advanced approaches would include automatic recovery strategies.

- Since applications require more than geometric information, and in order to capture design intent, user-defined features must support nongeometric attributes and variational constraints.
• User-defined features must provide mechanisms to inherit properties and attributes from their components and to selectively refine the inherited properties and attributes.

3.2 Definitions

We assume a feature-based design paradigm along the general lines set forth in [13]. Moreover, we assume that designs may involve the use of both geometric and nongeometric variables in a variational constraint schema; [13].

A basic definitional mechanism for three-dimensional volumes is to sweep a closed planar cross section profile along a given trajectory according to a well-defined extent semantics, plus a set of attributes. It is important that this process has a rigorously founded semantics; [12]. The result of this operation will be called a sketched form feature.

A profile is defined in a sketching plane. The sketching plane can be the support of an already existing planar face or a datum plane. In this paper, all cross sections to be swept are assumed closed. We will later discuss briefly the semantic problems associated with the use of open profiles. The definition of the profile is based on variational constraints that are solved when the feature is created. Constraints can also be used to position the profile relative to existing geometry. In this case, the constraints position the profile with respect to the projection of the existing geometry onto the sketching plane. When the actual shape of the feature is defined giving current values to a set of parameters, we say that the basic feature is parametrically defined.

If the trajectory is a line segment that is not parallel to the sketching plane, then the basic feature is called an extrusion. If the trajectory is a revolution around an axis not perpendicular to the sketching plane, then the basic feature is a revolved feature. For a discussion on the semantics of extruded and revolved features as well as an explanation of the extent semantics of attaching the feature see [4].

Attributes are additional information attached to 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, 3, 15].

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

1. a set of sketched form 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.

Each of the sets may be empty, but the set of sketched form features and the set of imported user-defined features cannot be empty simultaneously.

The definition of a specific UDF 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, adding other UDF's or features as components. Finally, an interface for the UDF is defined that governs the UDF use and attachment 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. However, they are used strictly for validity determination. 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 valued 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.

4 Implementation

We assume that the CAD system provides generated features, datum features, and modifying features; [14]. The generated features include profiled extrusions and revolutions that are used as protrusions and cuts with the attachment semantics described in [4]. Datum features consist of points, lines and planes. Modifying features include chamfers, blends and rounds. 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? 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.

4.1 User-Defined Features Creation

We split the process to create a UDF into two subprocesses. First, the geometry is created. Then, components attributes and equationals constraints between variables are defined.

4.1.1 Geometry Creation

UDFs definitions are built by defining sequentially every component. 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.

As a result of the sequential definitional process of the components, 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 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 in [4], there are two basic ways by which the extent of geometry creation is determined: blind extent and delimited extent.

A blind extent is determined from an explicit dimension value such as length or sweep angle subtended. Since the extent definition logically depends on the sketching plane, it must be possible to refer to the sketching plane when defining the UDF, and the sketching plane must be placed when instantiating the UDF later.
A delimited extent is determined at feature attachment time from the existing geometry specifying, for example, a sweep that ranges from a particular face to the next face in the sweep path. Delimited extents involve explicitly or implicitly identifying a face or a plane. In this case, the exact position of the sketching plane is not required, but the sketching plane orientation in space must be understood. Therefore, we distinguish between the sketching plane and delimiting planes for delimited extents definitions.

4.1.2 Attribute Definition

We may associate attributes with feature elements. Such attributes may be topological, functional and tolerancing attributes, textual attributes and user-defined attributes. We concentrate on topological attributes as a particular way to capture design intent.

A topological attribute is an attribute that is associated with a topological element of the UDF, i.e., with a vertex, an edge, a face, or a UDF component. Briefly, the attribute stipulates the role of the topological element in UDF instantiation. Topological attributes can be used to validate the geometry. Examples include:

Must be void: This attribute can be associated with a feature component volume that removes material from a part. The attribute implies that no other construction step may add material to the void region generated by the feature component.

Must be material: This attribute can be considered the opposite of the "must be void" attribute. Here, no other construction step may remove material from the solid region generated by the feature component.

Must be boundary: This attribute specifies that the associated boundary element of the feature must contribute to the final boundary in the part. If the element is an edge or a face, some part of the edge or face must be on the boundary of the part in which the UDF is used.

Must not be boundary: This attribute specifies that the boundary element in the feature must not contribute at all to the final boundary in the part. Note, however, that the feature must contribute to the final part.

Conditional if on boundary: This attribute specifies that a UDF is instantiated on a part depending on whether the associated topological element exists after instantiation. For example, consider a hole feature abstracted by a cylinder that is concentric with the hole and has a larger radius. We subdivide the cylinder perpendicular to its axis and associate the attribute with the lower portion. Then an instance of the hole feature is attached to the part if and only if the lower half of the cylinder does not intersect the part’s boundary. In effect, this attribute enforces a minimum material condition and a minimum hole depth.

Topological attribute are associated with a topological element by picking the element and specifying the attribute.
Topological attributes may require nonmanifold operations and nonmanifold solid representations. Where associated with internal boundaries and volumetric elements, special care is required so that, in the course of the design, we avoid a combinatorial explosion of subdivided cells and internal boundary elements.

4.1.3 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. It is the responsibility of the UDF designer to correctly define the constraints that apply. To support such constraints in the CAD system, solvers must be available that decompose the equations and coordinate them with the geometric constraint solving processes.

4.2 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 designer 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 is provided by 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.

The concepts of dependent and independent parameters can be formalized as follows. Let us denote by $C$ the set of all the tags in the UDF which represent symbolic geometric constraints, by $D$ the set of symbols representing datums, by $Q$ the set of variable symbols involved in the constraints associated with the
UDF, and let us denote by $F$ the set of names of geometric elements belonging to component features that are cross referred. The set of geometric elements on which the UDF is built is defined by $V = D \cup F$. Furthermore, let us denote by $E_1$ the set of valued geometric constraints defined between elements in $V$, and by $E_2$ the set of symbolic geometric constraints defined between elements in $V$. Now $E = E_1 \cup E_2$ is the set of tags which represent geometric constraints defined between elements in $V$. We will refer to the graph $G(E, V)$, naturally induced by a UDF, as the feature graph. It will play an important role in UDF attachment process.

Once the definition of a UDF is completed the parameters are defined. Independent and dependent variables are fixed as a result of the UDF definition. Each variable in the UDF belongs to one of the two following sets. One set is defined by those variables in $Q$, or in $E_2$ or in both. The other set is defined by those variables not in $Q$ nor in $E_2$. The first set is the set of dependent variables, $CP = E_2 \cup Q$. The second set is the set of independent variables, $P = C - CP$.

The case of datums is different. Since datums are always nodes in the feature graph, their tags are always related through constraints to other geometric elements. Hence, datums should be always dependent parameters. However, in our definition we have provided for independent datums. In doing so we have provided the designers with a tool to enforce that links for some datums be always given explicitly by the end-user of the feature at attachment time. Independent datums are designated explicitly by feature designers at creation time.

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.

### 4.3 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 particular, if the UDF is simple, inheritance in the is-a sense is obtained.

Without additional mechanisms, this inheritance style is too rigid. We provide therefore for selective inheritance and subtyping. Consider a UDF component in which some of the attributes are changed. This is affected 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 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.

4.4 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 feature instance. This information is passed to the UDF through parameters.

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.

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 feature graph. 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.

If the interface specifies explicitly which parameters and datums are independent, and a precomputation has determined that with such a set of independent parameters the attachment evaluation is deterministic, then simple spatial constraint solving suffices to evaluate the attachment, and this is normally the case. However, it is also possible that a set of parameters and datums is specified
without designating explicitly which ones are independent. Here, a constraint
solver is needed that determines, from the set of currently specified parameter
values and mating pairs, which other parameter values and mating pairs can be
computed as a consequence of the constraints of the UDF definition. In that
case, variational solvers will be needed.

The concepts of dependent and independent parameters can be formalized
as follows. Let us denote by $C$ the set of all the tags in the UDF which represent
symbolic geometric constraints, by $D$ the set of symbols representing datums,
by $Q$ the set of variable symbols involved in the constraints associated with the
UDF, and let us denote by $F$ the set of names of geometric elements belonging
to component features that are cross referred. The set of geometric elements on
which the UDF is built is defined by $V = D \cup F$. Furthermore, let us denote
by $E_1$ the set of valuated geometric constraints defined between elements in $V$,
and by $E_2$ the set of symbolic geometric constraints defined between elements
in $V$. Now $E = E_1 \cup E_2$ is the set of tags which represent geometric constraints
defined between elements in $V$. We will refer to the graph $G(E, V)$, naturally
induced by a UDF, as the feature graph. It will play an important role in UDF
attachment process.

Once the definition of a UDF is completed the parameters are defined. In-
dependent and dependent variables are fixed as a result of the UDF definition.
Each variable in the UDF belongs to one of the two following sets. One set is
defined by those variables in $Q$, or in $E_2$ or in both. The other set is defined by
those variables not in $Q$ nor in $E_2$. The first set is the set of dependent variables,
$CP = E_2 \cup Q$. The second set is the set of independent variables, $P = C - CP$.

The case of datums is different. Since datums are always nodes in the fea-
ture graph, their tags are always related through constraints to other geometric
elements. Hence, datums should be always dependent parameters. However,
in our definition we have provided for independent datums. In doing so we
have provided the designers with a tool to enforce that links for some datums
be always given explicitly by the end-user of the feature at attachment time.
Independent datums are designated explicitly by feature designers at creation
time.

4.5 Validity

UDFs validation can be separated into two steps: First, when the feature is
defined, there should be a definition validation. Then, when the UDF is attached
to a part or to another feature, there must be a validation for the attachment
operation.
4.5.1 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.

4.5.2 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., [11] Chapter 3.

5 Erep Representation

We exploit the extensibility of the Erep language defined in [14] because it already provides naturally many of the mechanisms needed to support UDFs such as persistent naming and variational constraint solving.
As in [14], the syntactic definitions are not meant to be a comprehensive design. Our aim is to show that our Erep approach provides suitable infrastructure on which to build the UDF mechanism, without a commitment to an specific application view or core CAD system.

A UDF 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.

5.1 Attribute Representation

In general, attributes can be represented in textual form, properly grouped under convenient 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 volume elements can be represented as qualifiers of the cross section profile. In the example given in Appendix A, the line CUT VOID in the DESCRIPTION header of the cross section, stipulates that the component feature removes material that must be void.

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. As an example, in Appendix A, the line e1 boundary in the COMPONENTS header of the BigSlot cross section defines that the face in the slot generated by the edge e1 must contribute to the boundary of the UDF instance.

5.2 Erep Evaluation

As described in Section 4.1.1. UDF have 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 for the set of parameters that define the shape and position of the UDF instance are given, 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.

6 Case Study

As an example, we consider first the problem of designing a UDF. Then we will show how an end-user can attach the already defined UDF to a given part in an ongoing design process.
6.1 Definition of a User-Defined Feature

We are to design a UDF composed of two extruded holes with the respective main axis in two different, parallel planes as illustrated in Figure 1. The longitudinal axis of the holes meet at an angle $\alpha$ that can range between $\alpha_1$ and $\alpha_2$.

Let us first design the hole with radius $R$ in Figure 1. A set of convenient datum planes can be defined as follows. Since we are designing the new feature from scratch, first we define the datum plane $DP_1$ as the sketching plane. See Figure 2. Datum planes $DP_2$ and $DP_3$ will be taken as reference to define the hole. $DP_2$ is defined as a plane normal to $DP_1$ through axis $Y$ and $DP_3$ is defined as a plane normal to $DP_1$ through axis $Z$. Then, datum plane $DP_4$ that will delimit the hole extent is defined as an offset of $DP_1$ at a distance $l_1$.

Once the framework of datums is set up, we sketch the hole profile on the sketching plane $DP_1$ and properly annotate the sketch with geometric constraints. See Figure 3.

Next, we define the hole 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 is involved in an extent that is not blind, datum plane $DP_{100}$, a copy of the sketching plane, is created by the system to support the from face. Figure 4 shows the resulting partially defined UDF.

Now we proceed to define the second basic component in the UDF, the hole with radius $r$ in Figure 1. The process is illustrated in Figure 5. First we define the datum plane $DP_5$, where the axis of the hole will lay, as an offset of plane $DP_2$ at a distance $h$. Then, datum plane $DP_6$ is built as an offset of $DP_4$ at a
distance \( l_2 \) and \( DP_7 \) is an offset of \( DP_3 \) at a distance \( e \). We can define now the datum axis \( DA_1 \) as the intersection of plane \( DP_6 \) and \( DP_7 \). This datum axis will be taken as the reference to place and orient the second hole. Datum plane \( DP_3 \) is built as a plane through \( DA_1 \) and at angle \( \alpha \) with \( DP_7 \). Note that \( DP_7 \) and \( DP_3 \) effectively define the orientation of the axis of the second hole with respect to the first one.

The profile of the new hole is defined on the sketching plane \( DP_9 \) which is built as a datum plane through \( DA_1 \) normal to \( DP_8 \). Figure 6 shows the first hole projected onto the sketching plane, the profile defined and the geometric constraints. To complete the geometry we define the extent of the attached hole. We assign the from face explicit semantics to datum plane \( DP_{10} \), which is an offset of \( DP_5 \) at a distance \( l_3 \), and the to face to the wall of the first hole. Figure 7 shows the resulting UDF.

![Figure 2: Datums for a generic hole.](image)

![Figure 3: Profile and constraints for the first basic hole.](image)
Figure 4: User defined generic hole.

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

\[
\begin{align*}
\epsilon &= 0.30 \times v \\
r &= 0.25 \times R
\end{align*}
\]

And we can define some relationships too,

\[
\begin{align*}
h &< 0.5 \times r \\
l_1 &\geq 1.5 \times l_2
\end{align*}
\]

Figure 5: Datums for the second hole.
Figure 6: Profile and constraints that define the cross section of the attached hole.

\[
\alpha > 90 \\
\alpha < 180 \\
l_3 \leq 0.9 \times l_1
\]

Then, datum planes \( DP_1 \) and \( DP_2 \) are defined to be independent parameters.

Finally we can assign topological attributes to boundary elements or to the feature's volume. An example of boundary attribute is declaring that the lateral 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. This volume attribute could be simply inherited from the feature components if it would be defined in them.

An Erep representation example for the UDF defined above is given in Ap-

Figure 7: User-defined feature.
6.2 Attachment

Assume that the part under design is that in Figure 8 and that we want to attach to it the UDF built above. Furthermore, assume that we want to attach first the hole with the largest radius. Figure 9 illustrates the attachment process.

We start the attachment by giving values to independent datum planes \( DP_1 \) and \( DP_2 \). The sketching plane \( DP_1 \) is mated with face \( a \) in the part and the datum plane \( DP_2 \) is mated with face \( b \). To position the cross section, we define first in the part a datum plane \( dp_1 \) which is mated with \( DP_3 \) in the UDF. By giving a value for parameter \( R \) the cross section profile is defined. The hole extent is fixed, for instance, by assigning the from and to semantics to the part faces \( a \) and \( c \) respectively, what results in datum planes \( DP_{100} \) and \( DP_{4} \) pointing to these faces.

To position the second hole with respect to the first one we first define in the part the datum plane \( dp_2 \) (See Figure 9) which is mated with \( DP_6 \) in the UDF. Then we give values for the distance \( h \), the angle \( \alpha \) and the parameter \( c \). The cross section is fixed giving a value to the radius \( r \). The hole extent can be fixed, for example, by assigning the through all explicit semantics to the to face in the UDF definition. The result of the attachment is shown in Figure 10.

7 Conclusions

The mechanism presented addresses customization needs in a simple, effective way. It could profitably be included in data exchange standards.

The feature abstraction that user defined features represents is a small step towards the more general problem of re-interpreting the feature structure of the design.
References


A Erep Extension for User-Defined Features

An example of extension of the Erep language to accomodate the user-defined feature definition given in Section 4 is as follows.
<ud_feature> ::= UD_FEATURE <name> <stamp>
  <header>
  <features_list>
  END_UDFEATURE

<header> ::= HEADER
  <external_features>
  <independent_variables>
  <independent_datums>
  <dependent_variables>
  <dependent_datums>
  <global_equations>
  END_HEADER

<external_features> ::= EXTERNAL_FEATURES
  <external_features_name_list>
  END_EXTERNAL_FEATURES

<independent_variables> ::= INDEPENDENT_VARIABLES
  <vars_list>
  END_INDEPENDENT_VARIABLES

<independent_datums> ::= INDEPENDENT_DATUMS
  <datums_names_list>
  END_INDEPENDENT_DATUMS

<dependent_variables> ::= DEPENDENT_VARIABLES
  <vars_list>
  END_DEPENDENT_VARIABLES

<dependent_datums> ::= DEPENDENT_DATUMS
  <datums_name_list>
  END_DEPENDENT_DATUMS

<vars_list> ::= { <variable> }

<variable> ::= <name> { <range> } { <default> }

<range> ::= { <MIN value> } { <MAX value> }
  | INSET <val_enum>

<val_enum> ::= <value> { <value> }

23
<default> ::= DEFAULT <value>

<datums_names_list> ::= { <name> }

<global_constraints> ::= GLOBAL_CONSTRAINTS
  EQUATIONS
  <equations_list>
  RELATIONSHIPS
  <inequalities_list>
  END_GLOBAL_CONSTRAINTS

A.1 Other Definitions

The following definitions update and complement those given in [14]. We start with the definition of a generic feature.

<feature> ::= FEATURE <name> <stamp> <type>;
  <references>
  <cross_section>
  <description>
  END_FEATURE

<references> ::= REFERENCES
  <list_references>
  END_REFERENCES

<list_references> ::= <empty>
  | <list_references>, <reference>

<reference> ::= <name> = <reference_type> <feature_face_name>

<reference_type> ::= SIDE_F
  | FRONT_F
  | END_F

<feature_face_name> ::= <f_name>.<cs_name>.<co_name>

<f_name> ::= <name>

<cs_name> ::= <name>

<co_name> ::= <name>
An `<f_name>` is the name of a feature, an `<cs_name>` is the name of a cross section in feature `<f_name>`, and an `<co_name>` is the name of a component in cross section `<cs_name>`.

The `<cross_section>` describes the twodimensional profile that will generate the volumetric feature by sweeping the threedimensional space. It consists of geometry, cross section components, objects and geometric constraints.

```
<cross_section> ::= CROSS_SECTION <name>
    <geometry>
    <components>
    <objects>
    <geometric_constraints>
END_CROSS_SECTION
```

Cross section components are the set of geometric elements like points, straight segments, circles and arcs of circle from which the cross section is built. Geometry describe the geometric loci like points, straight lines and circles which support the cross section components. Objects are sets of components describing a geometric entity like a loop of straight segments or a face. Finally, geometric constraints describe the set of geometric constraints that define the cross section.

The `<description>` defines the threedimensional characteristics of the feature.

```
<description> ::= DESCRIPTION
    <volumetric_type>
    <sketching_plane>
    <sketching_reference>
    <trajectory>
    <extent>
END_DESCRIPTION
```

B  Erep Example of a User-Defined Feature

An Erep representation example for the user-defined feature defined in Section 6 is as follows.

```
UDFEATURE TwoHoles 082196
HEADER
INDEPENDENT_VARIABLES
R MIN 10 MAX 30 DEFAULT 20;
END_INDEPENDENT_VARIABLES
```
INDEPENDENT_DATUMS
    dp1;
    dp2;
END_INDEPENDENT_DATUMS

DEPENDENT_VARIABLES
    h    DEFAULT S;
e;
l1;
l2;
l3;
v;
    alpha;
END_DEPENDENT_VARIABLES

DEPENDENT_DATUMS
    dp3;
    dp4;
    dp5;
    dp6;
    dp7;
    dp8;
    dp9;
    dp10;
    dp100;
    da1;
ENDDEPENDENT_DATUMS

GLOBAL_CONSTRAINTS
EQUATIONS
    h = 1.25 * r;
e = 0.3 * v;
RELATIONSHIPS
    r <= 0.75 * R
    l1 >= 1.5 + l2;
    alpha > 90;
    alpha < 180;
    l3 <= l1 / cos alpha;
END_GLOBAL_CONSTRAINTS
END_HEADER

FEATURE BigHole 082196
REFERENCES
   SmallHole
END_REFERENCES

DATUM dp1 DATUM_PLANE;
   CSP_X
END_DATUM
DATUM dp2 DATUM_PLANE;
   NORMAL dp1
   DEFAULT ON CSA_Y
END_DATUM
DATUM dp3 DATUM_PLANE;
   NORMAL dp1
   DEFAULT ON CSA_Z
END_DATUM
DATUM dp4 DATUM_PLANE;
   OFFSET 11 dp1
END_DATUM

CROSS_SECTION bighole
   COMPONENTS
      vertex pt1
      circle cr1
   END_COMPONENTS

CONSTRAINTS
   CONCENTRIC (pt1, c1);
   RADIUS R (c1)
END_CONSTRAINTS

DESCRIPTION
   CUT VOID
   SKETCHING_PLANE dp1
   TRAJECTORY NORMAL;
   EXTENT
      FROM dp100 TO dp4;
   END_EXTENT
END_DESCRIPTION
END_FEATURE

FEATURE SmallHole 082196
REFERENCES
END_REFERENCES

DATUM dp5 DATUM_PLANE;
  OFFSET h dp2
END_DATUM
DATUM dp6 DATUM_PLANE;
  OFFSET -12 dp4
END_DATUM
DATUM dp7 DATUM_PLANE;
  OFFSET e dp3
END_DATUM
DATUM da1 DATUM_AXIS;
  ON dp6
  ON dp7
END_DATUM
DATUM dp8 DATUM_PLANE;
  ON da1
  ANGLE alpha dp7
END_DATUM
DATUM dp9 DATUM_PLANE;
  ON da1
  NORMAL dp8
END_DATUM
DATUM dp10 DATUM_PLANE;
  OFFSET 13 dp9
END_DATUM

CROSS_SECTION smallhole OPEN
  COMPONENTS
    vertex pt1
    circle cr1
  END_COMPONENTS

CONSTRAINTS
  CONCENTRIC (pt1, cr1);
  RADIUS r (cr1)
ENDCONSTRAINTS
END_CROSS_SECTION

DESCRIPTION
CUT VOID
SKETCHING_PLANE dp8
TRAJECTORY NORMAL;
EXTENT
    FROM dp10 TO BigHole
END_EXTENT
END_DESCRIPTION
END_FEATURE
END_UDFEATURE
LSI-96-1-R "(Pure) Logic out of Probability", Ton Sales.


LSI-96-4-R "Skip-Trees, an Alternative Data Structure to Skip-Lists in a Concurrent Approach", Xavier Messegue.

LSI-96-5-R "Change of Belief in SKL Model Frames (Automatization Based on Analytic Tableaux)", Matías Alvarado and Gustavo Núñez.

LSI-96-6-R "Compressibility of Infinite Binary Sequences", José L. Balcázar, Ricard Gavaldà, and Montserrat Hermo.

LSI-96-7-R "A Proposal for Word Sense Disambiguation using Conceptual Distance", Eneko Agirre and German Rigau.

LSI-96-8-R "Word Sense Disambiguation Using Conceptual Density", Eneko Agirre and German Rigau.

LSI-96-9-R "Towards Learning a Constraint Grammar from Annotated Corpora Using Decision Trees", Lluís Marquez and Horacio Rodríguez.

LSI-96-10-R "POS Tagging Using Relaxation Labelling", Lluís Padró.


LSI-96-12-R "Using Bidirectional Chart Parsing for Corpus Analysis", A. Ageno and H. Rodríguez.

LSI-96-13-R "Limited Logical Belief Analysis", Antonio Moreno.

LSI-96-14-R "Logic as General Rationality: A Survey", Ton Sales.


LSI-96-17-R “Rewriting in Categories of Spans”, Miquel Monserrat, Francesc Rosselló, Joan Torrens, and Gabriel Valiente.


LSI-96-19-R “Learning Causal Networks from Data”, Ramon Sangüesas i Solé.


LSI-96-21-R “Exact Learning of Subclasses of CDNF Formulas with Membership Queries”, Carlos Domingo.

LSI-96-22-R “Modeling the Thermal Behavior of Biosphere 2 in a Non-Controlled Environment Using Bond Graphs”, Angela Nebot, François E. Cellier, and Francisco Mugica.


LSI-96-25-R “Decision Trees Have Approximate Fingerprints”, Víctor Lavín and Vijay Raghavan.


LSI-96-27-R “Massively Parallel and Distributed Dictionaries on AVL and Brother Trees”, Joaquim Gabarró and Xavier Messegue.


LSI-96-29-R “Bounding the expected length of longest common subsequences and forests”, Ricardo A. Baeza-Yates, Ricard Gavaldà, and Gonzalo Navarro.


LSI-96-31-R “ParaDict, a Data Parallel Library for Dictionaries (Extended Abstract)”, Joaquim Gabarró and Jordi Petit i Silvestre.

LSI-96-32-R “Neural Networks as Pattern Recognition Systems”, Lourdes Calderón.

LSI-96-33-R “Semàntica externa: una variant interessant de la semàntica de comportament” (written in Catalan), Vicent-Ramon Palasí Lallana.


LSI-96-41-R “Semàntica algebraica del llenguatge AL: l’algorisme α” (written in Catalan), V.R. Palasi Lallana.

LSI-96-42-R “Partial Occam’s Razor and its Applications”, Carlos Domingo, Tatsuie Tsujiki, and Osamu Watanabe.

LSI-96-43-R “Transparent Distributed Problem Resolution in the MAKILA Multi-Agent System”, Karmelo Urzelai.


LSI-96-46-R “Analysis of Methods for Generating Octree Models of Objects from Their Silhouettes”, Marta Franquesa and Pere Brunet.


LSI-96-51-R “Mètodes de validació d’esquemes de bases de dades deductives” (written in Catalan), Carles Farré.


LSI-96-54-R “Patterns in Random Binary Search Trees”, Philippe Flajolet, Xavier Gourdon, and Conrado Martínez.

LSI-96-56-R  "Double-Pushout Hypergraph Rewriting through Free Completions", P. Burmeister, F. Roselló, G. Valiente.

LSI-96-57-R  "On User-Defined Features", Christoph M. Hoffmann and Robert Joan-Arinyo.

Hardcopies of reports can be ordered from:

Nuria Sánchez  
Departament de Llenguatges i Sistemes Informàtics  
Universitat Politècnica de Catalunya  
Pau Gargallo, 5  
08028 Barcelona, Spain  
secretlsi@lsi.upc.es

See also the Department WWW pages, http://www.lsi.upc.es/www/