Design Compilation for Feature-Based, Constraint-Based CAD[*]

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

Abstract

We discuss the architecture of a solid design system we are developing. The system is based on a high-level design representation (Erep) that allows graphical design editing. In our system, design is feature-based and constraint-based. The system leverages existing solid modelers that need be neither feature-oriented nor constraint-based.

Our system implements a design compilation paradigm that relies on a strict separation of a modeler-independent design representation, recording a generic design, from the native boundary representation used by a particular modeling kernel that would construct instances of the generic design. Consequently, the core modeling system can be exchanged, and design instances can be compiled to a variety of different modeling systems.

In this paper we explain the information flow between the system components. In particular, we explain how to separate the user interface, the design compiler, and the underlying core modeling engine. Elsewhere we have addressed the related issues of feature attachment semantics, persistent naming of feature collision elements, and geometric constraint solving.

1 Introduction

The feature-based design paradigm is by now well-established in current CAD systems and is well-represented by a sizable segment of the research literature; see, for instance, the review by Pratt [14]. Complementing this paradigm,

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

Appeared in Proc. 3rd ACM Symp on Solid Modeling and Appl, 1995, 13–20.

constraint-based design has emerged as an important design accelerator and as a tool for expressing design intent. The impact of constraint-based feature-based CAD is underscored by the fact that virtually all major CAD vendors now offer systems that are based on this design approach, including Parametric Technology, AutoDesk, ComputerVision, Intergraph, and SDRC. There are also some research systems that venture into this paradigm to varying degree, including [5, 7, 13, 16, 19, 21].

One of the central goals of design with features is this: Instead of designing geometric shape alone, conceptualize shape design with stereotypical shape elements (the features) and add, if possible, information supporting manufacturing processes and documenting design intent. While design features and manufacturing features are generally not the same, they are often related. The conversion from design features to manufacturing features is possible in certain manufacturing domains. Design intent, however, is difficult to formalize. Nevertheless, it is argued by some to be expressed by the constraint schema and feature attachment by which a design has been defined; e.g., [16, 17].

Often, feature-based design has been approached as an extension of the CSG paradigm; e.g., [20]. At least, the CSG approach anchors the feature representation to a well-defined semantics of the design operations, although it does limit the design vocabulary. Commercial systems, in contrast, usually allow a more extensive design vocabulary, albeit at the price of not clearly documenting the semantics of the shape operations that have been implemented [9].

The requirements of a feature-based design system have been addressed by Faux [5], Hoffmann and Juan [10], Pratt and Wilson [15], Shah and Rogers [18]. In our opinion, an ideal feature-based design system should allow users to freely add, delete and modify constraints and features. The resulting design specification should then be processed automatically, to derive a particular shape that satisfies the constraints, and analyze it with respect to manufacturability, performance, etc. The technological obstacles to achieving this freedom in design include limited techniques for reliably solving 3D geometric constraints [11], a well-defined semantics for feature attachment [4] and a robust and reliable naming schema for identifying topological entities in a generic design [2, 12].

Feasible approaches to feature-based design systems have been described and justified by Faux [5], Pratt [14], Rossignac [16] and Solano and Brunet [20]. In their view, the feature model and the associated shape model are tightly integrated. In fact, Pratt states explicitly the desirability of linking closely the feature representation with an associated boundary representation defining form [14], arguing its necessity because of the interrelationship of form and function in mechanical CAD.

While we affirm this interrelationship, we believe it advantageous to replace tight integration with design compilation, raising the level of abstraction. The logical extension of the integration argument would be that feature attachment operations evolve to closely resemble the geometric operations in an underlying solid modeling system. Such tight integration is undesirable because it encourages nonstandard design representations that mix problem-specific generic information with software-specific instance information. An integrated implementation is difficult to adapt to new application domains and is cumbersome to change. Moreover, it impedes neutral product data exchange in environments where design in progress is to be exchanged, for example in distributed, collaborative manufacturing.

The consequence of using a neutral generative representation is that this representation must be reevaluated when a feature is edited. The value of this concept lies in its ability to avoid dependence on the specific capabilities of a core modeling system, and the great flexibility it offers in adapting to new application domains. Moreover, the concept serves demands for a neutral product data representation. However, several important and challenging technical issues, such as persistent naming of feature collisions, must be solved to support this neutral representation. With the exception of Kripac's thesis, we have found no concrete work in the literature that explains the mechanisms necessary to implement this paradigm. Commercial proprietary implementations embody partial solutions of the technical problems as we have characterized in [9]. We have addressed the problems in a number of reports and papers [1, 2, 4, 3, 6]. A variational approach to editing design is found in the work by Gupta and Turner [8]. This approach ignores the feature construction steps and directly operates on Brep models.

By severing the tight connection between a core modeling system and a design interface that presents a complex design and editing vocabulary, we can derive a number of important benefits at low cost [10]. In particular, core modeling systems with a minimal shape operation repertoire can be leveraged into sophisticated design systems. This requires carefully formalizing the information content of high-level design, and mapping it by a compilation process to a sequence of lower-level design operations carried out by the core modeling engine, supported by auxiliary computations and data structures. This can be done in a way that minimizes dependence on the core modeling system, so that the high-level design representation can be compiled, as it were, to different core modeling systems, while retaining the full flexibility of redesign. In this paper, we characterize the architectural elements of this approach to design.

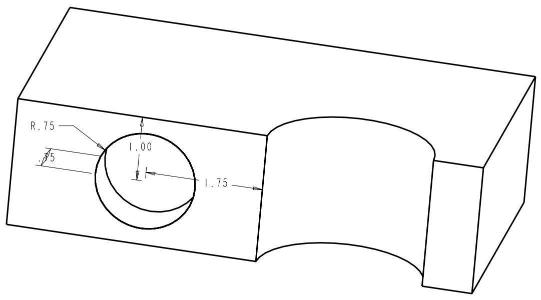


Figure 1: Constraint-based feature definition

2 Layered Representation of Feature-Based Design

Figure 1 shows an example of a slot feature that has been defined from constraints. If the constraints are attached to the boundary representation of the solid, an unnecessary dependence on the core modeling system has been created. Moreover, if the positional constraints placing the slot are changed, complicated geometric operations on the boundary representation have to be executed to update the model accordingly. What is needed instead is a high-level generic representation in which to express the feature definition and the constraint schema that governs its shape and position.

In [10], we have proposed a layered representation of feature-based design that separates the generic design definition, and hence the design intent, from the geometric representation that may be used by a solid modeler that is used only to construct instances of the generic design on demand. In this representation, design intent is stored in an unevaluated and modeler-independent representation, called an *editable representation (Erep)*. The geometry to which an Erep model corresponds, for specific constraint values, is stored in an evaluated representation, for instance as a particular boundary representation (Brep). The Brep representation is constructed by a solid modeling subsystem from the Erep model. The Erep model is independent from the Brep model.

In our design system, the user constructs features serially, based on variational constraints and attachment attributes. Since the definitions and design operations are expressed in a graphical user interface (GUI), there is an interaction between the unevaluated Erep model, the current instantiated corresponding Brep model, and the graphical user interface. Figure 2 shows this interaction of models.

It is desirable that this interaction between the model representations and the GUI be carefully controlled so as to not create any dependency of the unevaluated model construction on the particulars of the core modeling system and

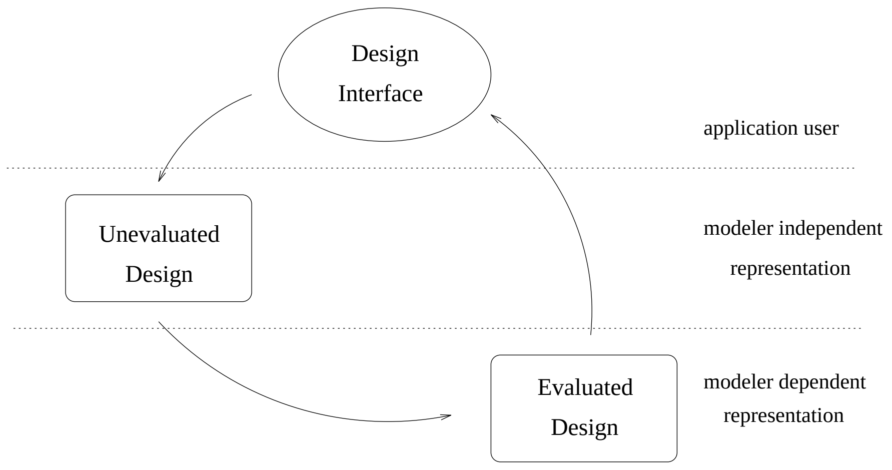

Design Interface
application user
Unevaluated Design
modeler independent representation
Evaluated Design
modeler dependent representation

Figure 2: In our system organization, the user interacts with a graphical representation of the evaluated design instance. The design operations are abstracted and recorded in a modeler-independent unevaluated representation (Erep). This generic representation, in turn, can be translated mechanically into a design instance.

its native representation. The independence is obtained by mediating the interaction between the GUI and the design instance representation by our design compiler. The information required by the GUI is obtained by specific formal operations implemented in the design compiler. Furthermore, the abstraction of a design operation is also done by the design compiler following specific instructions the GUI presents based on user design operations. The functional organization of the architecture is summarized in Figure 3.

3 The Unevaluated Design

The Erep language was first proposed by Hoffmann and Juan in [10], where the main benefits of organizing a CAD system around this language were articulated. The Erep specification given in that paper omits a number of technical issues that concern the abstraction of design gestures into the generic Erep level and the details of recording those abstractions. These details have been addressed elsewhere [2, 4, 3]. The feature operations themselves can be categorized into three broad groups: generated features, modifying features and datum features, now briefly characterized from the point of view of achieving modeler independence.

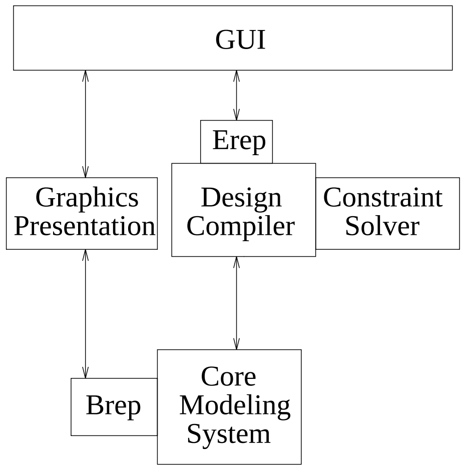


Figure 3: System architecture: The unevaluated model is translated by a compiler to operations executed in a core modeler when specific constraint values and attributes are prescribed. As explained, the GUI has to interact with the visual presentation of the evaluated model to allow graphical design and editing.

3.1 Generated Feature

A generated feature creates a new shape by sweeping a profile and relating the swept volume to the prior existing geometry. We describe the profile, the trajectory, and the attributes that define feature attachment. Clearly, most of this can be described independently of the core modeling system. Although perhaps not apparent at first, the cross section description must include how to display constraints, and must include an identification of which of the many possible solutions of the geometric constraints defining the profile. The profile description is therefore analogous to specifying a 2D drawing, and there are neutral standards that can be used for this purpose.

In addition to profile and trajectory descriptions, we need to identify specific shape elements of the prior geometry. These include a specification of the plane in which the profile must be constructed, faces that might limit the extent of the sweep, as well as projected faces edges and vertices referenced for the purpose of defining the constraints on the profile and its relative position. Here, a good naming schema is required that identifies the shape elements without using the boundary representation; [2]. Moreover, the naming schema must be assessed in view of the editing operations allowed; [3]. For instance, when changing the position of the round slot in Figure 1, the position of the blind hole must also change.

3.2 Modifying Features

Adding rounds and chamfers are operations that modify the existing geometry in a way that is understood from a few parameters. The user selects a list of edges and vertices in the existing geometry to be rounded with a chosen radius. The user may have to specify also end-conditions that determine how the round should terminate. Chamfers similarly require some shape parameters plus a list of edges and vertices. To support graphical editing and design, the selection requires interacting with the evaluated model constructed by the core modeler. Again, the unevaluated representation must record generic names identifying the selected edges. To accomplish this, the operations on the evaluated model are formalized, for instance as line/solid intersection, and an abstraction mechanism translating a specific shape element of an evaluated model to a generic characterization is implemented by the design compiler.

3.3 Datum Features

Datums are auxiliary geometric entities that serve to define sketching planes, aid placing features and define sweep extents. Our system uses points, lines and planes that are defined, one at a time, by a set of 3D geometric constraints. These constraints reference prior existing geometry. Since we solve parametric 3D constraint systems, the 3D constraints have a unique solution.

3.4 Modeler Independence

The system architecture contains as major components the GUI through which design is expressed graphically, the design compiler that interprets the generic Erep model, and the core modeling system that instantiates specific geometry in its own native representation. The information flow between these components falls into three broad categories.

1. Construct and match generic names. Because of the variability introduced by geometric constraints, it is not possible to base a naming schema on information derived from coordinate systems.

2. Solve geometric constraints. At least for 2D profiles, variational constraint solving has become the norm. This raises difficult issues of identifying the correct solution from a potentially exponential number of mathematically possible ones. It also implies that this identification should ideally be independent of the particular solver strategy the system uses.

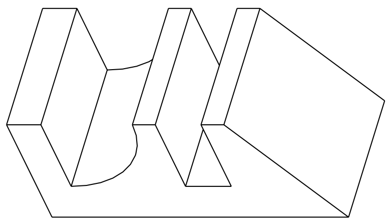
3. Follow a complete and unambiguous feature attachment semantics. Some design systems define generated features with open profiles. This implies that feature definitions and attachments are based on surface operations rather than on volume operations which give many opportunities for incomplete specifications.

When this information flow is formalized, independence from a specific core modeling system can be achieved.

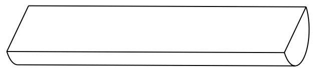
4 Design Compilation

The generic Erep model is processed by a design compiler that interprets each feature definition and compiles for each a sequence of modeling operations and auxiliary semantic computations. The modeling operations are done by a core modeling system, in our case ACIS. Auxiliary computations include maintaining tables and solving constraint problems. They are done by the design compiler itself. Datum features are managed by the compiler, but could be instead managed by the core modeler. Communication between the GUI and the design compiler is through a textual description, ordinarily a temporary file. Although this impacts speed, it does ensure a complete formalization of the information flow.

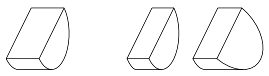
For a generated feature, compilation proceeds as follows: A sketching plane is identified, for example by matching a generic face name with specific face of the boundary representation maintained by ACIS. The existing geometry is projected orthographically onto this plane. Next, the cross section is properly constructed. This includes matching some names of existing projected geometry referenced by constraints. The constraints are solved resulting in a wire in the sketching plane. The wire is skinned and swept, so constructing a proto solid. The interference of the proto solid with the existing geometry is computed and the proto feature is trimmed according to the rules described in [4]. Finally, the actual feature is constructed by a Boolean operation. See also Figure 4. Thus, three distinct steps are carried out: matching the named references to geometric references [2, 3]; defining and placing the feature by solving specified constraints; and attaching the feature based on an unambiguous semantics[4]. The table summarizes which of these steps is needed for the other two classes of features.



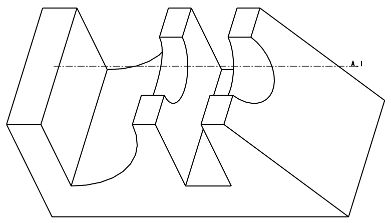
(a) existing geometry to be cut



(b) proto feature



(c) intersection with existing geometry; complement volumes not shown.



(d) subtraction from the existing geometry with the selected volumes

Figure 4: Two phases of feature attachment.

4.1 Constructing and Matching Persistent Names

The user executes a graphical design or editing operation that may involve referencing a geometric entity. The user visually identifies a vertex, edge or face. The GUI constructs a line of sight through the mouse position and asks the core modeler to construct a list of possible Brep entities the line intersects. Depending on the mode, the nearest entity is selected, or else another intersected entity based on a visual dialogue with the user. The required Brep entity is passed to the design compiler which returns a persistent name for it so that the design operation can be understood generically. This requires an association of names with Brep entities, a computation done by the compiler and recorded either in an internal data structure, or else by exploiting mechanisms supported by the core modeler. In our case, the ACIS attribute mechanism is used to

	generated feature	datum feature	modifying feature
matching names	×	×	×
constraint solving	×	×	
attachment	×		×

Table 1: Compilation steps involved in three classes of features

make the association. The name is returned to the GUI and becomes part of the generic description of the operation.

Conversely, when a previously recorded generic model operation is reinterpreted, a generic name is matched against the existing geometry. In that case, the name is procedurally interpreted as description of a geometric, topological, and historical context that identifies a unique locale on the Brep instance. Matching and naming can be quite complicated because even simple changes of dimension values can have complex shape implications.

4.2 Constraint Solving

Profiles are instantiated by solving geometric constraints. There is an extensive literature on geometric constraint solvers, and we use our own method described in [1]. Finding a neutral representation of the constraint problem is rather straightforward. However, recording the correct solution to the problem is difficult.

Mathematically, a geometric constraint problem is equivalent to a system of nonlinear algebraic equations and has, therefore, more than one solution. The literature on geometric constraint solving for CAD is curiously silent on this point. Commercial constraint solvers apply proprietary heuristics that are not always good, but since the heuristics are proprietary, it is difficult to characterize the domain in which the rules succeed.

As explained in [1], we select the solution obtained by applying a few simple design rules. In view of the difficulty to anticipate the user's intention especially when editing designs, we give the user the option of overruling the solution selection. Since our solver finds solutions by carrying out a sequence of simple construction steps, the particular solution is expressed in terms of the construction sequence and the choices at each construction step. A theoretical analysis of the appropriateness of this approach is given in [6].

This style of recording solutions leaves open the important problem of translating this record so that other constraint solvers working on different principles can find the same solution. In principle, one could enumerate all solutions. However, since the number of solutions of a constraint problem is exponential in the number of constraints, this approach is unacceptable in practice. More research is needed before this problem can be addressed effectively.

The constraints for defining datum features represent a small set of 3D parametric constraints. We solve them inside the compiler with algebraic equations.

4.3 Feature Attachment

Generated features and modifying features require a feature attachment step. Attachment is the result of several operations in the underlying modeling system whose results are interpreted by the design compiler. A detailed feature attachment semantics and its implementation has been reported in [4]. For a generated feature, its 2D profile is first placed in 3D space with respect to a sketching plane. A rotation matrix defines the orthographic of the 3D geometry onto the sketching plane. The 2D constraint solution defines the cross section. Here we need to record the nature of the solution. The manner in which the profile and the feature attributes are used is illustrated in Figure 4. Here a profile is extruded into the proto feature shown in Figure 4 (b). In Figure 4(a), we specify the curved face to be the *from* face and the rightmost face to be the *to face*. Then the Figure 4(d) is the geometry after the cut.

The particular sequence of operations of the core modeler that implement this semantics depend, of course, on the core solid modeler. In ACIS we rely for the most part on Boolean operations and boundary traversals. When using other modelers. such as Parasolid, more specific surfacing operations could be used.

For a modifying feature, the attachment is directly converted to a Brep modification operation. Rounding and chamfering operations have become common in most commercial solid modelers available today.

After attaching a feature, we add certain feature information to the geometric model. This information is needed in defining later features. For instance, a face of the current geometry might be selected as the sketching plane of the next feature whose generic name relies on the added information. In our implementation, we add this information through ACIS attribute mechanism.

5 Conclusions

In this paper, we emphasize in our architectural approach a strict separation of an abstract design representation from the specific capabilities of an underlying core modeling system. Clearly, different core modeling systems can be used, albeit with different design compilers. Consequently, the unevaluated model can be thought of as a vehicle to federate different core modelers. It can also be considered a strategy for leveraging core modeling capabilities, because the unevaluated feature operations recorded in the Erep need not correspond to a single shape operation in the core modeling system.

The unevaluated model can serve as a blueprint for product data exchange standards. Because it separates the representation from underlying modeling systems, it is a neutral design representation. Moreover, the unevaluated representation is ideal for doing collaborative distributed design because it does not presuppose a common CAD system and is much more compact than the evaluated model.

The design compilation paradigm encourages developing efficient techniques for evaluating generic designs. In turn, this work entails evolutionary pressures on core CAD systems. Thus, rather than packaging existing capabilities, the approach encourages developing new capabilities and so is more responsive to user needs.

In future work, the unevaluated representation should be extended to include nongeometric aspects of design, especially design intent data on performance, costing, tolerances, and so on. Moreover, the translation of the features in which the design has been expressed to features oriented towards manufacturing and engineering analysis needs to be explored.

References

[1] W. Bouma, I. Fudos, C. Hoffmann, J. Cai, and R. Paige. A geometric constraint solver. *Computer Aided Design*, page to appear, 1994.

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

[3] X. Chen and C. Hoffmann. Editing feature based design. Technical Report CSD 94-067, Purdue University, Computer Science, 1994.

[4] X. Chen and C. Hoffmann. Towards feature attachment. Technical Report CSD 94-010, Purdue University, Computer Science, 1994.

[5] I. D. Faux. Reconcilation of design and manufacturing requirements for product description data using functional primitive part features-from a three-dimensional feature database. Technical Report R-86-ANC/GM/PPP-01, CAM-I, Arlington, TX, USA, 1986.

[6] I. Fudos and C. M. Hoffmann. Correctness proof of a geometric constraint solver. Technical Report 93-076, Purdue University, Computer Science, 1993.

[7] D. C. Gossard, R. P. Zuffante, and H. Sakurai. Representing dimensions, tolerences and features in MCAE systems. *IEEE CG & A*, 8(2):51–59, 1988.

[8] S. Gupta and J. U. Turner. Variational solid modeling for tolerance analysis. *IEEE CG & A*, 13(5):64–74, 1993.

[9] C. M. Hoffmann. Semantic problems in generative, constraint-based design. In *Practice of Computer Aided Geometric Design — What CAD Systems are Really Capable of.* Teubner Verlag, 1993.

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

[11] Christoph M. Hoffmann and Pamela J. Vermeer. Geometric constraint solving in R^2 and R^3. In D. Z. Du and F. Hwang, editors, *Computing in Euclidean Geometry.* World Scientific Publishing, 1994. second edition.

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

[13] R. Light and D. Gossard. Modification of geometric models through variational geometry. *Computer-Aided Design*, 14(4):209–214, 1982.

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

[15] M. J. Pratt and P. R. Wilson. Requirements for the support of form features in a solid modeling system. Technical Report R-85-ASPP-01, CAM-I, Arlington, TX, USA, 1985.

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

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

[18] J. J. Shah and M. T. Rogers. Functional requirements and conceptual design of the feature-based modeling system. *Computer Aided Engineering*, 5(1):9–15, 1988.

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

[20] L. Solano and P. Brunet. A system for constructive constraint-based modeling. In B. Falcidieno and T. Kunii, editors, *Modeling in Computer Graphics*, pages 61–84. Springer Verlag, 1993.

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