Concurrent Design

Susan Finger, Mark S. Fox
Friedrich B. Prinz, James R. Rinderle

Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

Given the initial functional specifications for a product, a designer must create the description of a physical device that meets those requirements. The final design must simultaneously meet cost and quality requirements as well as meet the constraints imposed by activities such as manufacturing, assembly, and maintenance. Mechanical designs are often composed of highly-integrated, tightly-coupled components where the interactions are essential to the behavior and economic execution of the design. Therefore, concurrent rather than sequential consideration of requirements, such as structural, thermal, and manufacturing constraints, will result in superior designs.

Our goal is to create a computer-based design system that will enable a designer to concurrently consider the interactions and trade-offs among different, and even conflicting, requirements. We are creating a system that surrounds the designer with experts and advisors that provide continuous feedback based on incremental analysis of the design as it evolves. These experts and advisors, called *perspectives*, can generate comments on the design (e.g. comments on its manufacturability), information that becomes part of the design (e.g. stresses), and portions of the geometry (e.g. the shape of an airfoil). However, the perspectives are not just a sophisticated toolbox for the designer; rather they are a group of advisors who interact with one another and with the designer.

This paper focuses on the motivation and integration of the research that has resulted from the multi-disciplinary group creating this design system, called Design Fusion. The research falls into broad areas: geometric modeling, features, constraints, and system architecture.

1. Introduction

In creating a concurrent design system for mechanical designers, our goal is to infuse knowledge of downstream activities into the design process so that designs can be generated rapidly and correctly. The design space can be viewed as a multi-dimensional space in which each dimension is a different life-cycle objective such as fabrication, testing, serviceability, and reliability. An intelligent design system should aid the designer in understanding the interactions and trade-offs among different, and even conflicting, requirements. We are creating a system that surrounds the designer with experts and advisors that provide continuous feedback based on incremental analysis of the design as it evolves. These experts and advisors, called *perspectives*, can generate comments on the design (e.g. comments on its manufacturability), information that becomes part of the design (e.g. stresses), and portions of the

To appear in ***Applied Artificial Ingelligence Special Issue on AI in Manufacturing***

geometry (e.g. the shape of an airfoil). The perspectives are not just a sophisticated toolbox for the designer; rather they are a group of advisors who interact with one another and with the designer.

The design methodology is integrated around a shared, dynamic, domain-neutral representation of the design. The shared representation includes the geometric model of the design as well as the features, constraints, and design record. Constraints are the language by which perspectives communicate with one another and with the designer. The design record contains the design decisions that led to the creation of a constraint or feature. The perspectives are coordinated through a blackboard architecture which uses a heterarchical control structure.

2. Design System Architecture

Designers use a variety of methods and techniques throughout the design process. They have many tasks to perform and numerous sources of design data. Some subproblems have algorithmic solutions; however, no single algorithmic solution exists for the design problem in its entirety. Human expertise is required to integrate the subproblems, provide the missing pieces, and guide the process known as design. Recently, with the development of knowledge-based system technologies, software has been created that can participate directly in the design process by making design decisions[1].

The design system architecture has two roles. First, it provides an interactive environment that enables the designer to control the available resources that consist of data, knowledge, methods, and algorithms. Secondly, the architecture provides a group problem-solving environment in which knowledge-based systems contribute to the design process. The Design Fusion architecture is based on the blackboard model of problem solving [8] illustrated in Figure 1. The architecture has four major components: the blackboard, knowledge sources, search manager, and user interface.

The blackboard provides a shared representation of the design and is composed of a hierarchy of three panels. The *geometry panel* is the lowest level representation of the design and uses a non-manifold geometric model of the design. The *feature panel* is a symbolic level representation of the design. It provides symbolic representations of features, constraints, specifications, and the design record. The *control panel* contains the information necessary to manage the operation of the system.

Perspectives and methods are the two types of knowledge sources. *Perspectives* represent knowledge of different stages in the product life cycle. Each perspective may criticize design decisions or generate new design information. Using perspectives that communicate through a blackboard architecture enables us to partition the design knowledge. Each perspective can define its own internal set of features, constraints, and variables, so that inconsistent requirements, names, and definitions are contained within the perspectives. Communication occurs through the common language of the shared representation. *Methods* provide standard analysis capabilities to the system. Three methods are currently being used: feature extraction, constraint management, and mathematical programming.

The *search manager* provides a means for dynamically coordinating the perspectives. The system cycles through four stages of control: perspective identification, perspective selection, perspective execution, and constraint management. At the beginning of a cycle, that is, after a design decision has been posted to the blackboard, any number of perspectives may have contributions to make. The search manager must

[1]Examples of knowledge based systems that make design decisions include XCON [2], PRIDE [21] and ALADIN [17].

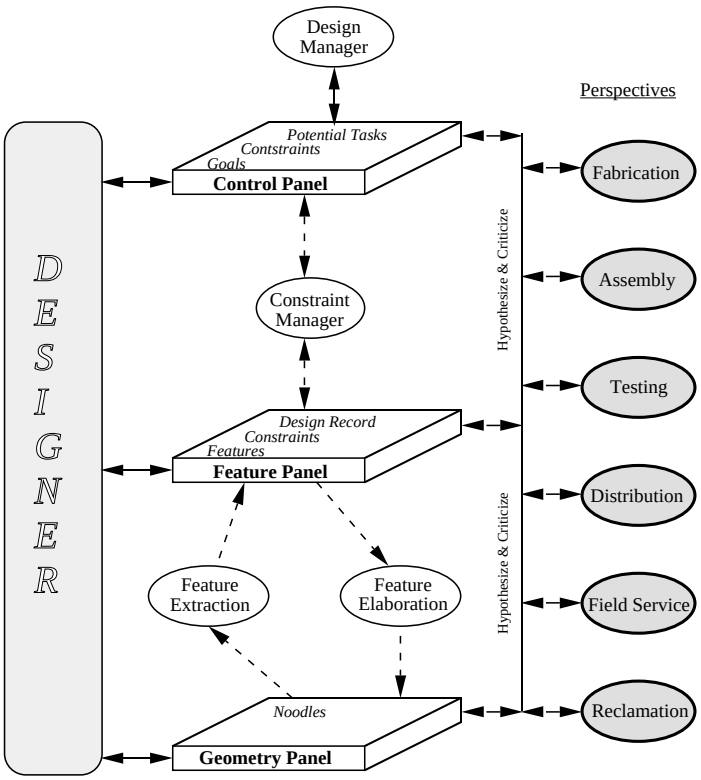

Figure 1: Design Fusion system architecture

decide the sequence of contributions and control their execution.

The *user interface* provides the designer with a complete interactive environment for designing. It provides the user with the ability to define specifications and constraints, to select from a library of existing designs, and to modify designs. The user also has the capability to confirm or override the systems suggestions at each stage in the search manager's decision cycle.

3. Design Representation

Our system is based on the concept of a shared representation. The shared representation of the design is maintained on the blackboard, and all comments, constraints, and design changes are made in terms of it. Perspectives may create local representations for reasoning and analysis, but communication is always through the shared representation. During the design process, large quantities of information about a design are used and generated. We have made the decision to include in the shared representation only those attributes that may be of interest to more than one perspective. Using perspectives enables us to partition the design knowledge into manageable chunks, while allowing us the flexibility to add new information to the representation. For example, the manufacturing perspective may have a constraint on the maximum length of a cast turbine blade. As long as this constraint is not violated, it remains within the perspective; however, if it is violated, the manufacturing perspective would post the constraint on the

blackboard.

If a complete representation of a design could be constructed, it would include attributes like the initial specifications, the geometry with dimensions and tolerances, the material and structural properties, the manufacturing and assembly sequences, the design history including versions and configurations, the bill of materials, the maintenance procedures, and so on. Depending on the design domain, the importance of representing particular attributes will vary. We have focused on representing the geometry, features, and constraints associated with a design.

3.1. Geometric Representation

The representation of geometry has been an active area of research over the last fifteen years. In a review paper, Requicha and Voelcker [29] discuss the progression from early CAD systems to advanced solid modellers. Voelcker [41] also discusses the limitations of current geometric models as design systems because they can only represent the geometry of a completed geometric object rather than an evolving design. Discussions along similar lines can be found in Nielsen [24] as well as in Gursoz and Prinz [15].

In surface boundary representations, known as *b-reps*, objects are modeled by representing their enclosing shell. The basic elements of a b-rep are faces, edges, and vertices. The topology of an object is made explicit by giving the connections between its elements, and the geometry of the object is made explicit by giving coordinates to the vertices, giving lengths to the edges, etc. In constructive solid geometry (CSG), objects are modeled as boolean combinations of a set of primitive solids; that is, an object is constructed by adding and subtracting the basic primitives. An object is represented as a binary tree in which the terminal nodes of the tree are solid primitives, and the intermediate nodes are boolean operations that operate on the primitives to create the desired object.

Both the b-rep and CSG approaches were created to represent solid objects in R^3 space. These models are not able to represent incomplete objects. The non-manifold geometric modeling systems created by Weiler [44] and by Gursoz and Prinz [15] address this issue. These representations build upon the boundary representations, but they are able to represent the more complex adjacency patterns such as dangling edges or nested cones that can occur in non-manifold objects.

Because one-, two-, and three-dimensional objects can be represented consistently in non-manifold representations, they are well-suited to design systems. With non-manifold representations, the design can include a center line of a hole, a parting plane for a mold, and internal boundaries for a finite-element mesh, as well as the enclosing shell of the designed object. Figure 2 shows the evolution of the construction of a solid from a wireframe in a non-manifold representation.

3.2. Feature Representation

Our research in feature-based representations of designs has been motivated by the realization that geometric models represent the design in greater detail than can be utilized by designers, process planners, assembly planners, or by the rule-based systems that emulate these activities. Experts often abstract geometry into features like ribs, parting planes, and chamfers. To date, our research has been on defining and recognizing shape features, that is, features that are derivable from the geometry and topology of the design. We represent shape features using a graph grammar based on the non-manifold representation. The geometry of the designed object and the feature definitions both use the non-manifold representation, so features can be recognized by matching the graph representing the feature with a

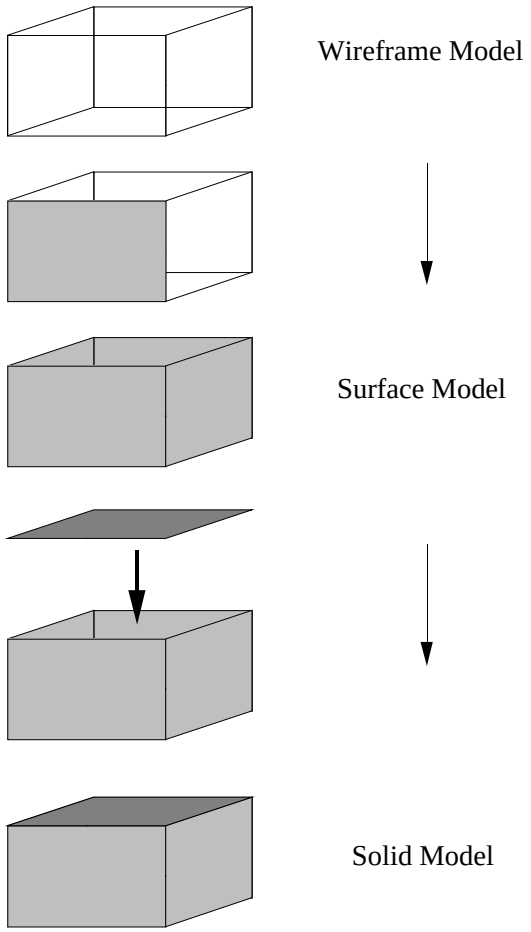

Figure 2: Evolution of a solid model

subgraph of the graph representing the design.

Many features are only used locally by perspectives. For example, the manufacturing perspective may make a preliminary process plan based on the manufacturing features recognized in the design. However, a feature or a feature interaction may cause the manufacturing perspective to generate a comment to the designer giving a warning or advising a change in the design. Because the features are defined in terms of the shared representation, the perspectives can communicate by referring their features to the shared representation. So, even though the designer may use a different term for a feature or may chunk the geometry differently, the manufacturing feature can be highlighted on the geometric display.

For example in Figure 3, a designer and a manufacturer each have a set of features defined. The designer sees two slots, defined by their width and depth, that serve a functional role in meeting a design requirement. The manufacturer is concerned with making the artifact and not only sees the two slots but also the wall created between them. A manufacturing analysis of this wall indicates that it is too thin to be milled to the given tolerance. Although the designer lacks the wall feature, the manufacturer's

definition is used to improve the design. The shared model is a basis of communication via feature definitions for the two perspectives.

3.3. Constraint Representation

The representation shared among perspectives must include not only the evolving product geometry and features, but it must also include the allowable limits on geometry, the relationships among behavior and geometry, and other constraints. The set of constraints asserted by any one perspective is an encoding of the life-cycle concerns of that perspective. The collection of all constraints is the set of currently relevant life-cycle concerns that determine the acceptability of a design alternative. Each perspective, when commenting on the design or suggesting design changes, can view all posted constraints and therefore suggest modifications that minimize conflict. Additionally, the design perspective may characterize design trade-offs by evaluating competing constraints. As the design evolves, features are added and modified causing individual perspectives to assert additional constraints and to modify or retract existing constraints. In this way the collection of constraints is an embodiment of the evolving life-cycle constraints on an acceptable design.

A design record tracks the design decisions that led to the creation of a constraint or feature. Design records are defined by the perspective which generated the decision, the type of processing that led to the

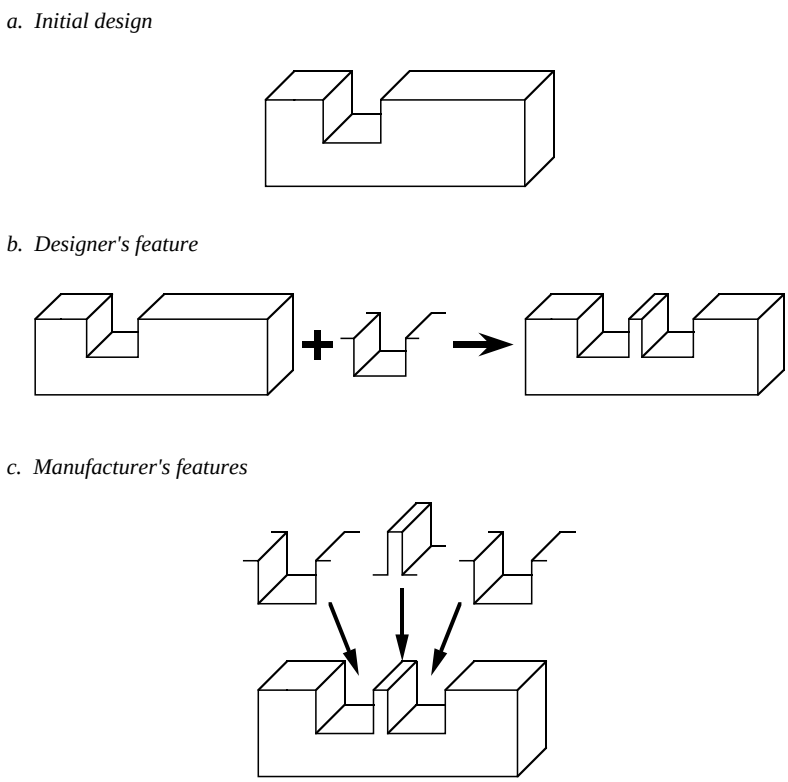

Figure 3: View-point specific feature interaction

decision, and the information upon which it was based. This information can be used to maintain design consistency when underlying assumptions of the design change or to track constraint violations back to the sources.

3.4. Quantitative and Qualitative Representations

Qualitative representations provide a means for reasoning about complex systems without the need for quantitative data. Most design systems perform quantitative analysis of the results of the design process. Numeric algorithms, given numeric input, produce numeric descriptions for properties of the design. One problem with numeric models is that the underlying relationships are often lost or hidden in quantitative representations. In addition, these underlying relationships cannot be manipulated symbolically. Qualitative representations extend quantitative representations by making implicit relationships explicit and accessible.

Procedural knowledge provides a representation for the processes necessary to perform some tasks. It can be algorithmic, such as a finite element analysis program, or heuristic, such as problem-solving. The design representation requires both algorithmic and heuristic information, one augmenting the other. Some tasks have algorithmic solutions that result in some relationship between design parameters. Other tasks use heuristic methods, such as pattern directed search that guides the problem-solving process.

Consider the pattern, or production, in Figure 4. The production is composed of a condition and an action. When the condition is satisfied, the action is invoked. In this case, the stress concentrations in a turbine blade shank are computed when a shank geometry is proposed by the designer. Production systems use pattern-directed search to encapsulate operational descriptions for problem-solving tasks.

IF

1. *there is a constraint on the life-time of the blade;*
2. *and there is a proposed geometry for the shank;*
3. *and the stress concentrations in the shank are unknown;*

THEN

1. `execute the finite element model on the shank geometry`

Figure 4: Sample production

---

4. Features

Features provide both an abstraction mechanism and a mechanism for communicating among experts in a heterogeneous environment. Our approach is to describe features using a graph grammar. Because the designed object is an element in the language generated by this grammar, features can be recognized by parsing the graph representing a feature against the graph representing the object. We provide a representational link between the low-level geometric representation and the high-level design abstractions by formalizing a language to express classes of high-level objects in terms of the low-level ones. Given this language, we are able to extract the high-level elements from the neutral low-level geometric representation.

The use of features derivable from the geometry, that is, *form features* is an area of active investigation in mechanical design [10]. Other researchers have constructed systems that extract features from two-manifold solid models. Using a boundary representation, these systems define features as patterns, and instances of the pattern are extracted from the model [9, 11, 13, 18, 33]. Other research in using features in CAD systems has focused on single domains. Woo [45] utilizes decomposition using form features to perform structural analysis. Shah has looked at mapping features between domains [35]. Several researchers, including Unger and Ray [40], Cutkosky and Tenenbaum [5], Chang et al. [4] and Hayes [16] have explored the use of features in constructing process plans for parts. Many research groups are currently working on feature-based design systems. The two of most interest here are Dixon *et al.* [6] and Cutkosky and Tenenbaum [5]. Our approach differs from these in that we do not use a predefined set of features to build and represent the design.

Figure 5 illustrates several features, all labeled *hole*. From a functional point of view, a designer might specify a hole only by its centerline, radius, and purpose (e.g. alignment) while a manufacturer might define a hole by its location, radius, and manufacturing process (e.g. a punched hole). Both the designer and manufacturer use the label *hole*. While the features labeled as holes are similar, they are not identical. The difference of perspective for characterizing the concept *hole* necessitates differing feature definitions. The ability to represent both manifold and non-manifold objects is essential in describing partial designs or referring to conceptual elements such as center lines or symmetry planes.

4.1. Representation Formalisms

Our work on feature grammars builds on the work of Stiny [37] who first created shape grammars based on the formalisms of linguistics. Using a formal grammar, instances of a class of objects can be generated based on a sequence of production rules. We use a graph grammars, the class of grammars that operates on two-dimensional graphs. A tutorial on graph grammars and their applications is given by Ehrig [7]. Our approach to defining features is based on Pinilla's work [26]. He defines form features by a context-sensitive graph grammar called an *augmented topology graph grammar*. This grammar represents features as topological and geometric entities and permits pattern-directed recognition and generation of salient features from a solid model.

To describe the boundaries of 3D objects, which are inherently two-dimensional, we have created a grammar whose domain is the graphs representing an object's topology augmented with geometric information. Both non-manifold and manifold objects can be represented with this augmented topology

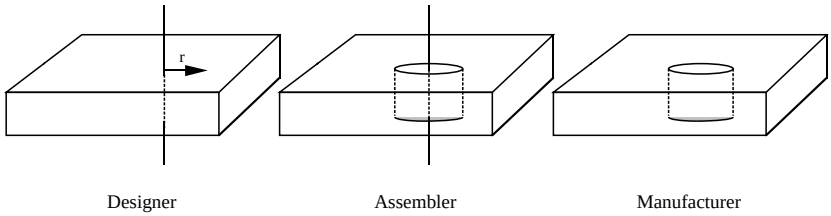

Figure 5: A Hole from three different perspectives

graph (ATG). Topology is encoded by four primitives: nodes (points in 3-space), edges, faces and loops (the enclosing boundaries of faces). Each primitive is represented as a vertex in the augmented topology graph.

These elements form a graph structure in which the nodes contain the topological elements and the arcs contain relationships between the elements. The relationships are both topological and cross-referencing geometric information, such as adjacencies and distances or angles between elements, as well as self-referencing geometric information, such as face area or edge solid angle. The self-referencing information is represented by self-loop arcs. Thus, we achieve a uniform representation of properties in the arcs and simplify the labeling of the nodes of the graph. Figure 6.a and 6.b illustrate a simplified model and its associated augmented topology graph. In [26], we present a more complete description of the grammar, productions, and embedding rules.

4.2. Feature Grammars

In order to parse a design to recognize its features, a set of features must exist in a representation consistent with the representation of the design. Each feature is represented by its faces, edges, and nodes, as well as dimensional characteristics. The features are stored in a graph that represents the adjacencies and relationships between features, providing a base for further abstraction.

We use three levels of abstraction for recognizing features. At the lowest level is a non-manifold solid

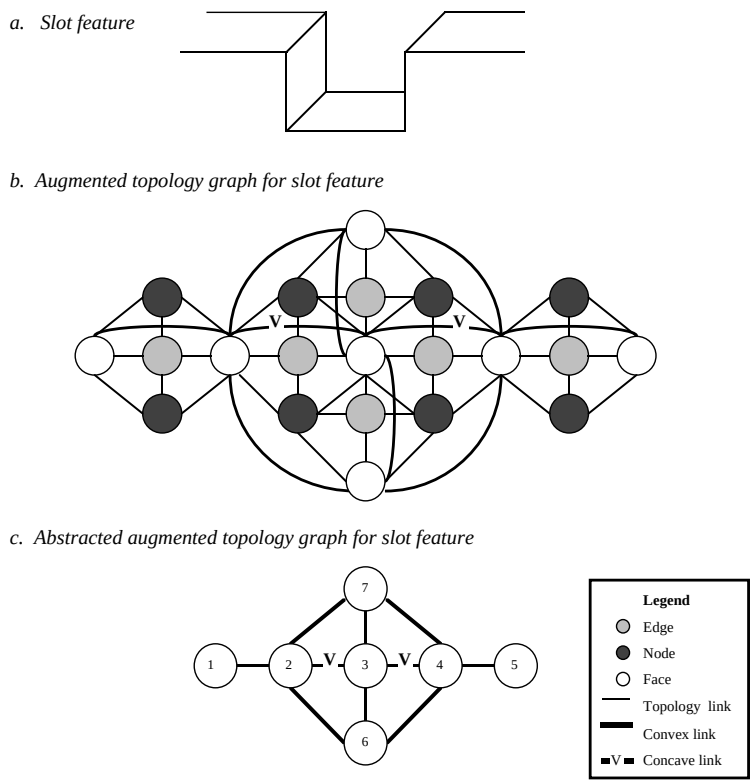

Figure 6: Example: A slot feature

modeller. This level provides complete information for representing a solid, including all topological and geometric data about the model. The intermediate level of abstraction is the augmented topology graph. This level captures the geometric relationships from the input grammar and maintains a non-manifold representation. The most abstract level represents manifold features and any manifold portion of non-manifold features such as geometric and topological relations between faces. Any non-manifold portion of these feature is represented at the intermediate level. By providing multiple levels of abstraction, we reduce the search space and concentrate the search on the areas most likely to match particular features.

The power of the recognition system relies on the completeness and specificity of the feature descriptions. A recognizer that requires enumeration of all possible cases will be slow and inefficient as well as difficult to implement and maintain, since the number of cases is often infinite. The description must be able to express classes of objects, not only instances of them. The recognizer must be able to recognize individual instances of those classes from the general description. Such a description conforms to the formalism of a language that can be described with a grammar. Under this formalism, a class of features is described by a grammar whose starting symbol is a canonical form of the feature and a *finite* set of rewrite rules that generate all possible instances of the class. Figure 7 illustrates a simple rule for splitting walls in a slot, that is, for creating a new instance of the slot feature.

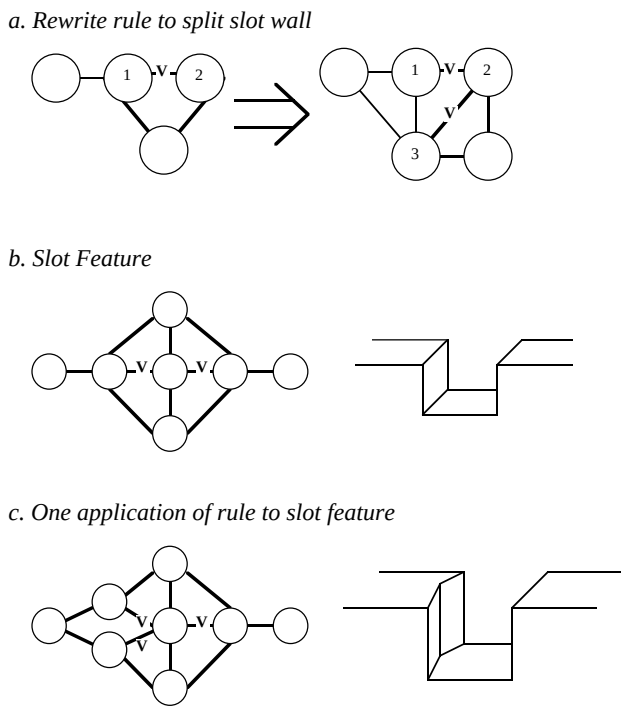

Figure 7: A simple rewrite rule for a slot grammar

4.3. Feature Recognition

Using our methodology, the recognition of a feature is reduced to identifying a subgraph within the object's ATG. The subgraph can be generated by a grammar associated with the feature. For a complete recognizer in a domain, a number of grammars must be developed, one for each feature class considered. A *feature extractor* finds the complete set of feature instances derivable from the productions of an augmented topology graph grammar given the grammar and an augmented topology graph representing the geometry of a design. Because both the geometric model and the feature definitions are represented as graphs, the problem of feature extraction is the problem of finding isomorphic subgraphs, an NP-complete problem.

Feature recognition occurs in two phases. In the first phase, called *grammar compilation*, a feature grammar is processed to enable incremental processing. In the second phase, *feature parsing*, the compiled grammar is used to extract features from a solid model. Because this method of feature extraction occurs incrementally, features can be extracted from an incomplete model as the model is constructed. As the graph representing the model is built, its vertices can be mapped on the recognizer. As features come into existence, they are found by the recognizer.

During grammar compilation, the graph defining a feature is transformed into an equivalent graph that allows for more efficient processing. The vertices in the input feature grammar are classified according to the number and type of arcs connecting the vertex. Then, these classes are ranked so that those with the fewest member vertices have highest priority. The recognizer is constructed as a directed graph from vertices with the higher priority to vertices with the lower or equal priority. Consider the slot feature in Figure 6; its most distinguishing characteristic is its bottom face. By focusing on vertices in the design model that are characteristic of bottom faces, the fraction of the model that must be searched can be reduced. In this feature, as in many other features, the manifold edges and nodes contain little information that is useful in parsing. For two-manifold features, geometric relationships between faces provide the most useful information. For example, in Figure 6.a and b, all edges in the feature are connected to two faces and two nodes. The structure of the nodes and edges is the same no matter where they occur in the feature. They are not useful for discriminating between high-level features. The feature in Figure 6.b removes these ambiguous nodes and includes only those attributes that provide less ambiguous information, that is, the geometric relationships between faces.

Applying the feature compilation procedure to Figure 6.b, the vertices of the graph are collected into four classes: those with a single convex connection (Class A), those with three convex connections (Class B), those with two convex connections and one concave connection (Class C), and those with two convex connections and two concave connections (Class D). The class with the highest priority, Class D, is the starting vertex for the recognizer. Vertices are added to the recognizer, one for each vertex in the input graph, based on the priority of the class containing the vertex. Directed links are added from vertices with higher priority to lower. If vertices have equal priority, the direction is assigned arbitrarily. The feature recognizer for the slot is shown in Figure 8.

During feature parsing the compiled recognizer is used to find features in the design model. First, the vertices representing faces in the design model are mapped into the same classes that were defined in the input grammar. Then, connectivity from the feature grammar is verified in the model, guided from the least frequently occurring classes to the most. Again, consider the compiled feature in Figure 8 and the model represented in Figure 9. First, the face-vertices in the model are mapped on to all the classes defined by the grammar. Note that this is not a unique mapping. There may be vertices in the model that are not represented by any class in the recognizer; and likewise, there are vertices in the model that map

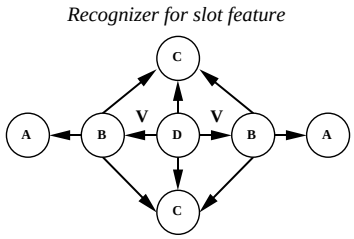

Figure 8: Feature recognizer for a slot

on to more than one class in the grammar. For example, faces that are convex with three other faces, like class C in the example, are also members of class A, a face that is convex with another face. The assignment of classes to vertices in the model is shown in Figure 9.c. Finally, if the vertices in the feature and the design model match, then the connectivity in the feature recognizer must be verified. In Figure 9.e, all the nodes in the recognizer have been identified and verified, so the slot feature has been matched in the design model.

This method of feature matching is a special case of the rete match algorithm [12]. The rete algorithm is an efficient method of matching many patterns to many objects. The rete match process has two steps. First, patterns are matched with objects in a working memory. Second, interpattern dependencies are verified. These two phases correspond to the mapping of face-vertices on to classes and verifying that the faces are configured properly. The algorithm presented here tunes the more general rete algorithm to the problem of feature extraction.

In [32], we present the complete algorithm for feature extraction. Because our method of feature recognition is bottom-up, features can be extracted from an incomplete model while the model is being constructed. As the graph representing the model is built, the vertices of the graph can be mapped on the recognizer. As features come into existence, analyses can be performed, and the designer can be given feedback on the design as it evolves.

5. Constraints

In the context of engineering design, a constraint can be thought of as a required relationship among design features and characteristics. Constraints may embody a design objective (e.g. weight), a physical law (e.g. $F = ma$), geometric compatibility (e.g. mating of parts), production requirements (e.g. no blind holes), or any other design requirement. Collectively, the constraints define what will be an acceptable design. The number, diversity, and variable context of constraints make finding an acceptable design a difficult task. Furthermore, finding the design that satisfies all the constraints is only possible when the constraint network represents all design alternatives, when it is complete and consistent, and when it results in a unique solution. These conditions are rarely, if ever, met. Perhaps more importantly, just a solution to a set of constraints does not necessarily contribute to the designer's understanding of the relative impact of various constraints and therefore does not assist the designer in identifying alternative design configurations that are not governed by similar constraints.

Design constraints are usually numerous, complex, and highly nonlinear. Our objective is to provide the designer with insights about the critical interactions among features, redundant requirements, and

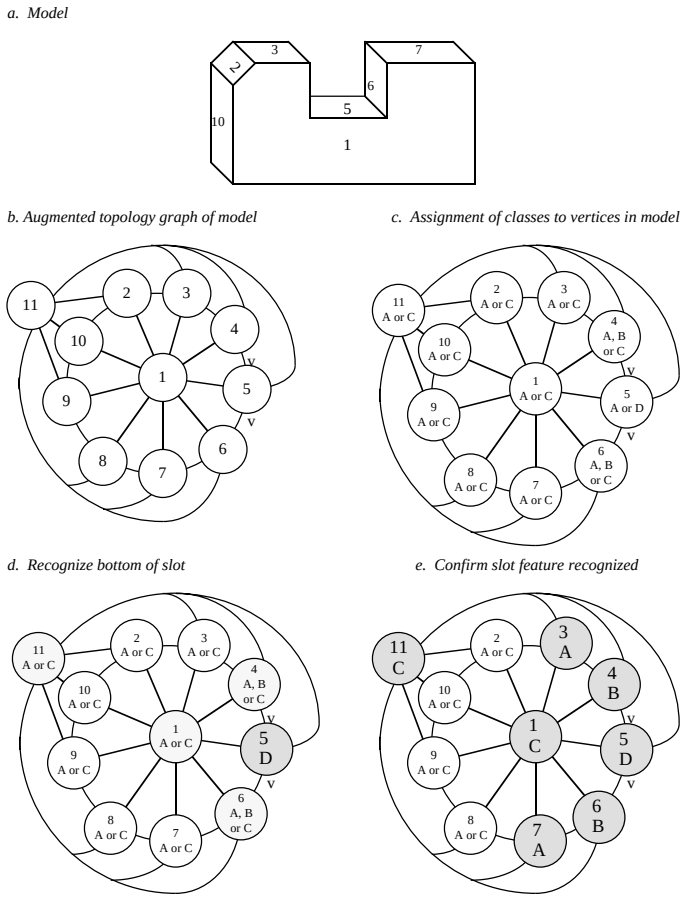

Figure 9: Recognition of a slot feature in a design model

inconsistencies. This information is useful to the designer even if the constraints can be solved numerically because a purely numerical solution does not facilitate understanding of the design task.

In many cases, it is difficult for a designer to understand the nature of a solution or deadlock, particularly if constraints refer to each other in a circuitous structure. Some of this difficulty can be alleviated by identifying suitable transformations on constraint networks that result in directed rather than circuitous structures. The numerical evaluation of circuitous constraints is relatively straightforward. The algebraic transformation is significantly more difficult, especially if the goal is to find transformations that have physical significance to a designer and that augment a designer's insight into the design problem [43].

In design, a small set of constraints often is critical in determining many other design relations. The ability to identify and address these critical constraints early in the design process is important to the designer. As different perspectives impose new constraints on the design the importance of identifying bottle-neck constraints becomes even greater. We are currently exploring several different techniques for identifying the bottle-neck constraints.

5.1. Monotonicity Analysis and Interval Methods

Monotonicity analysis [25] facilitates the simplification of a constraint network and the identification of inappropriately bounded constraint networks. Unfortunately, most engineering design constraints do not exhibit the global monotonicity required for the application of monotonicity analysis; however, *regional* properties of functions can be exploited. The regional information can then be reassembled to draw global inferences. We are using a methodology based on interval analysis to represent, utilize, update, and reassemble regional information.

Using the monotonicity principles can result in the deletion of constraints and reduction in size and complexity of the model when variables are regionally monotonic. Similarly, different constraints may become active and dominant in different regions; hence we gain leverage by exploiting regional information. We apply *interval methods* to represent, abstract, and manipulate regional information.

Interval arithmetic is used as the basis for evaluating algebraic relations containing interval variables, yielding interval results. By using interval methods, we can characterize regional monotonicities, regional feasibilities, etc. of design constraints. The four basic arithmetic operators produce exact intervals, but the representation of higher level functions in terms of these basic arithmetic operators introduces some difficulty. Conservative interval calculation destroys the one-to-one correspondence between intervals on arguments and intervals on functions. This has important implications for design systems, in which it is often necessary to determine what range of arguments will satisfy a range on a function itself. The extent to which a computed interval deviates from the actual interval determines how strong the inferences are that can be made on the intervals on variables.

Some specific techniques can be used to mediate against the expansion of intervals. One such approach is the centered form of functions based on a Fourier expansion of the intervals and is described by Moore [22]. Other heuristics, for example dealing with even exponents, are also useful. In addition, several ad-hoc methods obtain less conservative intervals and even exact intervals.

5.2. Constraint Propagation in Design

When a design decision is made, constraints can be used to propagate the decision to other parts of the design. For example, once a motor shaft diameter is specified, it is possible to determine some characteristics of other components such as the bearings. Depending on the topological structure of the constraint network, propagating and checking the consistency of constraints is difficult. In addition, a designer needs not just the solution, but also needs an understanding of the nature of the solution. In particular, a designer needs to understand how certain design decisions or variables were set, how those variables depend on other design variables, and the leverage that design variables and constraints have upon other design decisions. We address this need by providing a solution and an explanation of the solution that tracks the dependencies in a constraint network and evaluates the impact of a decision on other design variables.

Another important issue in the satisfaction of a constraint network is the scope of changes in a design that result from a single design decision or a change in a constraint. When changes can be localized, understanding the nature of the constraints is straightforward. However, a small change that at first may appear to be local, may in fact propagate across the entire design space. The effects of such changes are difficult to track and understand.

Intervals can be effective for representing and reasoning about design parameter values. Interval values

can also be propagated through a set of constraints so that potential constraint violations can be detected. By propagating design decisions through constraints, the effect of some design parameters on one another can be determined. In the process, redundant constraints are identified and eliminated. The intervals of the parameters can also be refined in this process.

Any variable can affect any other variable if there is a chain of constraints connecting them. Propagation can occur in any direction; it is not the case that one variable in a constraint must be selected *a priori* as being dependent while all others are regarded as independent. As constraints are propagated and as intervals narrow, specifications may be found to be inconsistent with other constraints, thereby identifying violations and redundancies before design decisions are made. Interval propagation provides insight about a design without the need to choose specific values for design parameters. We believe that the ability to draw important inferences about a design problem early in the process is important in concurrent design.

A large body of research exists on solving constraint propagation problems including that of Sutherland [39], Mackworth [20], Borning [3], Sussman and Steele [38], Gosling [14], Popplestone [27], Steward [36], Sadeh [31], and Serrano [34]. These techniques provide a core of solution methods directly applicable to algebraic constraints in real variables. Based on these methods, we are developing propagation techniques applicable to constraints among interval variables. Some important differences exist when dealing with interval constraints: the distinctions between equality and inequality constraints change, constraints may be evaluated when any number of interval variables are not yet specified or even when all intervals are finite, and a single constraint may be evaluated many times to obtain additional design information.

5.3. Interval Criticality, Dominance, and Activity

The large number of constraints which arise in a concurrent design environment make it useful to characterize the relative importance of each constraint. Some constraints are active; their presence influences the design. Constraints that are known to be inactive can be eliminated without influencing the design. Some of the active constraints are critical; they determine a part of the design solution. Most critical constraints are inequality constraints that are satisfied as equalities in the final design. Some constraints dominate others; satisfying the dominant constraints insures the satisfaction of the others. Dominated constraints are inactive and can be deleted.

Constraints in design may not be globally monotonic, active, dominant or critical, but may have these properties within a region. Therefore, the concepts of constraint criticality, dominance, and activity defined over regions are more effective in identifying the critical constraints and eliminating the insignificant ones. Interval methods can again be used to characterize regionally dominant, critical and active constraints [19, 30].

Constraint dominance is an especially useful property for the following reasons:

- Dominance is transitive; dominance relationships can be propagated.
- Dominance often is context independent; the dominance relationship between two constraints may be independent of objective and other constraints.
- Context-dependent properties, like constraint activity and criticality, can be identified using constraint dominance.
- Dominance can help manage constraints in a concurrent design setting where constraints may

be dynamically asserted. The significance of newly asserted constraints can be evaluated by examining their interaction with currently dominant constraints.

In a concurrent design setting where life-cycle constraints can be dynamically asserted, the effect of a newly introduced constraint can be studied by testing for dominance against the currently dominant constraint in different regions of the design space. If a new constraint dominates the currently critical constraint in some region of the design space, then the new constraint is critical. Thus, the transitivity of dominance can be used to prove criticality of a new constraint.

5.4. Global Optimization

Global optimization of a general, nonlinear, nonconvex objective function subject to nonlinear constraints is an unsolved problem. There is no single best method to attain a globally optimal solution. Most traditional nonlinear programming techniques are local methods and can get stuck in local valleys. Also, only under strong assumptions about the function can a solution be guaranteed to be globally optimal.

Interval methods have been used to solve the global optimization problem [28]. The methodology behind these approaches for unconstrained optimization is as follows:

- Use interval methods to represent regional information.
- Exploit the bounds provided by the interval method to guide the branch and bound search strategy in which regions of the design space which have lower bounds are examined first.
- Use a subdivisioning procedure to accelerate the search by yielding tighter bounds.

To solve the constrained optimization problem, these methods successively subdivide the constrained design space until they arrive at a part of the space that satisfies all the constraints. Due to the the extreme conservatism of interval calculations and the nonlinearity of the constraints, it is difficult to obtain a region that satisfies all the constraints through interval calculations. On the other hand, it is not necessary that each and every constraint be satisfied in every region through interval calculations. A large portion of the constraints are dominated in some regions and can be deleted from those regions.

5.5. Reduction of Computational Complexity

Design problems often have large numbers of complex constraints that must be satisfied to complete a design task. Because it is impossible to guarantee the simultaneous solution of a large set of design constraints, we have investigated algorithms for planning and simplifying constraint satisfaction. Satisfying a large number of constraints does not imply that all the constraints must be solved simultaneously. We have developed algorithms for finding coupled constraints and for creating a solution plan that minimizes the need for simultaneous solution.

The simplest type of constraint sets are those that do not need to be solved simultaneously. Constraint sets are said to be *serially decomposable* if the constraints can be solved serially, yielding the value of one new variable for each constraint evaluation [36]. We have also found that estimating the value of critical variables can sometimes uncouple equations, thereby reducing or eliminating simultaneity.

A serially decomposable constraint set can be ordered using a simple row and column elimination algorithm. This algorithm fails if the constraint set is not serially decomposable. An algorithm for assessing the decomposability of a constraint set, prior to ordering, has been proposed by Rane *et al.* [19].

When a constraint set is not serially decomposable, portions of the constraint set must be solved simultaneously. Using algorithms based the work of Serrano [34] and Steward [36], subsets of the constraint set can be identified and isolated to be solved simultaneously. The algorithm consists of two stages: *matching* and *ordering*. A matching should be maximal, that is, the maximum number of possible matchings should be found. This is achieved using a standard bipartite matching algorithm [1]. A maximal match determines which variable is computed from which constraint, but does not determine the order of solution. The *ordering* of the computation is based on variable-constraint matching. These dependencies can be represented as a directed graph among variables. When these dependencies are circuitous, a group of constraints, said to comprise a *strong component*, must be considered simultaneously. Strong components can sometimes be broken or simplified by estimating the value of one of the variables in the strong component. The process is analogous to untying knots in a string. Untying a large knot might either reveal smaller knots or might eliminate the knot altogether. By breaking a strong component, single-degree-of-freedom search can be performed on one variable instead of solving for all the variables simultaneously.

It is our hypothesis that this idea can be extended to larger problems. In [23], we present algorithms that help to identify the best variables to select to simplify a given constraint problem. We also present experiments that show that in many cases it is possible to eliminate simultaneity by estimating the value of just one variable.

The notion of using bipartite matching and the strong components algorithm together was originally suggested by Wang [42]. The algorithms were used to solve Gaussian matrices for solving sets of equations using Newton-Raphson-like methods. Serrano [34] applied a similar algorithm for finding strong components in sets of constraints. The aim of his work was to concentrate solution on components and to avoid solving the entire constraint set simultaneously. Both these efforts are aimed at bi-directional constraints. We have extended the algorithms to uni-directional constraints. We have also developed the notion of breaking strong components using heuristic approaches.

6. Controlling The Design Process

The architecture provides a group problem-solving environment in which the designer and the perspectives cooperate in the generation of a design. Both the designer and the perspectives have the opportunity to generate and test design decisions, enabling the simultaneous participation of all perspectives throughout the design process rather than *ex post* critique. The competing goals of the designer and the different life-cycle perspectives as well as the interactions between specification of the requirements and the specification of the artifact provide many sources of conflict during the design process. Consequently it is necessary to determine dynamically which of the perspectives' contribution dominates at each stage of the design process. Specifying a blackboard architecture is not sufficient to specify the system's design behavior. The designer manager's role is to coordinate the activities so that they are cooperative and coherent.

The philosophy that underlies the group problem solving strategy is a least commitment approach. Rather than making specific design decisions immediately, constraints are imposed successively until commitments must be made. The implication is that problem solving is constraint directed; however, it is not possible to state all the constraints on a design and then to solve them. In addition to the fact that the initial constraint set may be unsolvable, it is also true that the constraint set changes over time as decisions are made and different parts of the design space are explored. Perspectives represent a partitioning of knowledge relevant to some stage in the product life cycle. Much of the knowledge may

not be relevant to the current design task, and depending on the path taken by the designer, many of the constraints within the perspectives may never be relevant to a particular design problem. Therefore posting all of the constraints on the blackboard at the outset would not only obfuscate the problem but increase the problem solving complexity to that point of being unmanageable. The alternative is to let each perspective determine the *relevance* of its knowledge to the situation at hand, and then *reveal* whatever knowledge is relevant in the form of a constraint.

Design is an exploration among alternative designs and among the methods to generate and evaluate them. At any point in the design process, the designer and the perspectives may have many contributions to make. The computer resources and the designer's time are limited, so decisions have to be made on which paths to explore and which methods to use. An open issue is the determination of which perspective dominates at any state in the design process when contributions may conflict, overlap, or be tangential. The current demonstration version of Design Fusion leaves the selection to the designer, but we believe that the appropriate approach is based on an analysis of the existing constraints.

Inconsistencies and conflicts in goals inevitably arise during the design process. Dealing with inconsistencies in the constraint network is another area of research. Due to the conflicting goals and variations of knowledge of perspectives, revealed constraints can lead to inconsistencies. These inconsistencies are tolerated by the system but are also tracked. Our approach is to use a dependency representation so that the sequence of decisions, and ultimately the core hypotheses that lead to the inconsistency, can be identified and retracted when necessary.

The search manager's control abilities are made possible through the definition of a precise, multi-level protocol that defines how a perspective can make contributions. The lower level protocol focuses on integrating the contributions of each perspective through the assertion, derivation, and retraction of constraints. The upper level focuses on the postponement, relaxation, and satisfaction of constraints. Figure 10 defines the lower level protocol. Work on the upper level protocol is underway.

7. Conclusion

We have implemented the first version of the design system that embodies the research presented in this paper. This system, known as Design Fusion, has enabled us to test and refine our ideas on concurrent design. In the process of implementing the Design Fusion system, we have

- created a method for defining and recognizing non-manifold features and have begun to implement an efficient algorithm for recognizing features in an evolving design
- created an architecture that integrates partial solutions to portions of the design problem based on a common representation
- created new algorithms for reasoning about constraints using interval methods and regional partitioning.

The Design Fusion system supports concurrent design by enabling the simultaneous consideration of life-cycle constraints. It uses a shared representation of the design which can be parsed using perspective-specific features. It uses constraints as a language by which perspectives communicate with one another and with the designer. The perspectives are coordinated through a blackboard architecture that uses a heterarchical control structure.

- ASSERT:
  - Assigns a value or constraint to a feature.
  - Causes a new branch to be created in the design evolution tree.
  - Cannot be retracted.
- POST:
  - Assigns a value or constraint to a feature.
  - Causes a new branch to be created in the design evolution tree.
  - Can be retracted.
- REVISE:
  - Modifies a value or constraint of a feature.
  - Maintains the same branch of the design evolution tree.
  - Can be retracted.
- DERIVE:
  - Assigns a value or constraint to a feature.
  - Maintains the same branch of the design evolution tree.
  - Is retracted automatically if a posting or revision it depends on is retracted.
- RETRACT:
  - Removes a value or a constraint from a feature.
  - Causes a new branch to be created in the design evolution tree.

Figure 10: Low Level Protocol Definition

Acknowledgments

This work has been sponsored in part by Defense Advanced Research Projects Agency (DARPA) under contract No. MDA972-88-C-0047 for the DARPA Initiative on Concurrent Engineering (DICE) and by the National Science Foundation under the Engineering Research Centers Program, Grant CDR-8522616.

We wish to acknowledge the work of Yung-Cheng Chao, Eric Gardner, Jerry Griffin, Levent Gursoz, V. Krishnan, D. Navinchandra, Harold Paxton, Miguel Pinilla, Scott Safier, Atul Sudhalkar, and Christopher Young, all of whom have contributed many ideas and substantial time to the creation of the Design Fusion system.

References

1. Aho, A. V., Hopcroft, J. E. and Ullman, J. D., *Data Structures and Algorithms,* Addison-Wesley Publishing Company, Reading, Massachusettes, 1983.

2. Bachant, J. and McDermott, J, "R1 Revisited: Four Years in the Trenches," *AI Magazine*, Vol. 5, No. 3, 1984, pp. 21-32.

3. Borning, A., "ThingLab- A Constraint Oriented Simulation Laboratory," Technical Report, Xerox Palo Alto Research Center, 1979.

4. Chang, T. C., Anderson, D. C. and Mitchell, O. R., "QTC - An Integrated Design/Manufacturing/ Inspection System for Prismatic Parts," *Computers in Engineering 1988*, American Society of Mechanical Engineers, San Francisco, CA, August 1988, pp. 417-426.

5. Cutkosky, M. R., Tenenbaum, J. M. and Muller, D., "Features in Process-Based Design," *Proceedings of the International Computers in Engineering Conference*, American Society of Mechanical Engineers, San Francisco, CA , July 1988.

6. Dixon, J. R., "Designing with Features: Building Manufacturing Knowledge into More Intelligent CAD Systems," *Proceedings of ASME Manufacturing International-88*, American Society of Mechanical Engineers, Atlanta, Georgia, April 17-20 1988.

7. Ehrig, H., "Tutorial Introduction to the Algebraic Approach of Graph Grammars," *Graph-Grammars and their Applications to Computer Science*, Springer-Verlag, New York, Lecture Note Series 1987, pp. 3-14.

8. Erman, L. D., Hayes-Roth, F. Lesser, V. R. and Reddy, D. R., "The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty," *Computing Surveys*, Vol. 12, No. 2, 1980, pp. 213-253.

9. Falcidieno, B. and Giannini, F., "Automatic Recognition and Representation of Shape-Based Features in a Geometric Modeling System," *Computer Vision, Graphics, and Image Processing*, Vol. 48, 1989, pp. 93-123.

10. Finger, S. and Dixon, J. R., "A Review of Research in Mechanical Engineering Design, Part II: Representation, Analysis, and Design for the Life Cycle," *Research in Engineering Design*, Vol. 1, No. 2, 1989, pp. 121-137.

11. de Floriani, L., "Feature Extraction from Boundary Models of Three-Dimensional Objects," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11, No. 8, 1989, pp. 785-797.

12. Forgy, C., "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," *The Journal of Artificial Intelligence*, Vol. 19, 1982, pp. 17-37.

13. Gavankar, P., Chuang, S. H., Henderson, M. R. and Ganu, P., "Graph-Based Feature Extraction," *Proceedings of the First International Workshop on Formal Methods in Engineering Design, Manufacturing, and Assembly*, Colorado State University, Colorado Springs, January, 15-17 1990, pp. 167-183.

14. Gosling, J., *Algebraic Constraints*, PhD dissertation, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1983.

15. Gursoz, E. L., Choi, Y. and Prinz, F., "Vertex-based Representation of Non-manifold Boundaries," in *Second Workshop on Geometric Modeling*,Wozny, M. J. and Turner, J. and Preiss, K., ed., IFIP, New York, 1988.

16. Hayes, C. C. and Wright, P. K., "Automating Process Planning: Using Feature Interactions to Guide Search ," *The Journal of Manufacturing Systems*, Vol. 8, No. 1, January 1989, pp. 1-15.

17. Hulthage, I., Fox, M.S., Rychener, M.D. and Farinacci, M.L., "The Architecture of ALADIN: A Knowledge-Based Approach to Alloy Design," *IEEE Expert*, Vol. 5, No. 4, 1990, pp. 56-73.

18. Joshi, S. and Chang, T. C., "Graph Based Heuristics for Recognition of Machined Features from a 3D Solid Model," *Computer Aided Design*, Vol. 20, No. 2, March 1988, pp. 58-66.

19. Krishnan, V., Navinchandra, D., Rane, P. and Rinderle, J. R., "Constraint Reasoning and Planning in Concurrent Design," Technical Report CMU-RI-TR-90-03, Robotics Institute, Carnegie Mellon University, 1990.

20. Mackworth, A. K., "Consistency in Network Relations," *Artificial Intelligence*, Vol. 8, 1977, pp. 99-118.

21. Mittal, S., Dym, C. and Morjaria, M., "PRIDE: An Expert System for the Design of Paper Handling Systems," in *Applications of Knowledge-Based Systems to Engineering Analysis and Design*,American Society of Mechanical Engineers, 1985, pp. 99-116.

22. Moore, R., *Methods and Applications of Interval Analysis*, SIAM, Philadelphia, 1979.

23. Navinchandra, D. and Rinderle, J. R., "Interval Approaches for Concurrent Evaluation of Design Constraints," *Symposium on Concurrent Product and Process Design*, American Society of Mechanical Engineers, San Francisco, December 1989.

24. Nielsen, E. H., Dixon, J. R. and Simmons, M. K., "How Shall We Represent the Geometry of Designed Objects?," Technical Report 6-87, Mechanical Design Automation Laboratory, University of Massachusetts, 1987.

25. Papalambros, P. Y. and Wilde, D. J., *Principles of Optimal Design*, Cambridge University Press, New York, 1988.

26. Pinilla, J. M., Finger S. and Prinz, F. B., "Shape Feature Description and Recognition Using an Augmented Topology Graph Grammar," *NSF Engineering Design Research Conference*, University of Massachusetts, Amherst MA, Amherst, MA, June 11-14 1989, pp. 285-300.

27. Popplestone, R. J., Ambler, A. P. and Bellos, I., "An Interpreter for a Language for Describing Assemblies," *Artificial Intelligence*, Vol. 14, No. 1, 1980, pp. 79-107.

28. Ratschek, H. and Rokne, J., *New Computer Methods for Global Optimization*, Ellis Horwood Limited, Chichester, England, 1988.

29. Requicha, A. A. G. and Voelcker, H. B., "Solid Modeling: A Historical Summary and Contemporary Assessment," *IEEE Computer Graphics and Applications*, Vol. 2, No. 2, March 1982, pp. 9-24.

30. Rinderle, J. R. and Krishnan, V., "Constraint Reasoning in Concurrent Design," *Design Theory and Methodolgy - DTM '90*, American Society of Mechanical Engineers, Chicago, 1990, pp. 53-62.

31. Sadeh, N. and Fox, M.S., "Preference Propagation in Temporal Constraints Graphs," Technical Report CMU-RI-TR-89-2, Intelligent Systems Laboratory ,The Robotics Institute, 1988.

32. Safier, S. A. and Finger, S., "Parsing Features in Solid Geometric Models," *Proceedings of the European Conference on Artificial Intelligence-90*, Pitman Publishing, Stockholm, Sweden, August 1990, pp. 566-572.

33. Sakurai, H. and Gossard, D. C., "Shape Feature Recognition from 3-d Solid Models," *Proceedings of the International Computers in Engineering Conference*, American Society of Mechanical Engineers, July 1988.

34. Serrano, D., *Constraint Management in Conceptual Design*, PhD dissertation, Department of

Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA, 1987.

35. Shah, J. J. and Rogers, M. T., "Feature-Based Modelling Shell: Design and Implementation," *Proceedings of the International Computers in Engineering Conference*, American Society of Mechanical Engineers, July 1988.

36. Steward, D. V., "Partioning and Tearing Systems of Equations," *Journal of SIAM, Numerical Analysis Series B* , Vol. 2, No. 2, 1965, pp. 345-338.

37. Stiny, G., *Pictorial and Formal Aspects of Shape and Shape Grammars*, Birkhauser, Basel, 1975.

38. Sussman, G. J. and Steele, G. L., "CONSTRAINTS-- A Language for Expressing Almost Hierarchical Constraints," *Artificial Intelligence*, No. 14, 1980, pp. 1-39.

39. Sutherland, I. E., "Sketchpad - A Man-Machine Graphical Communication System," Technical Report 296, MIT Lincoln Laboratory, 1983.

40. Unger, M. B. and Ray, S. R., "Feature-Based Process Planning at the AMRF," *Computers in Engineering 1988*, American Society of Mechanical Engineers, San Francisco, CA, August 1988, pp. 563-569.

41. Voelcker, H. B., "Modeling in the Design Process," in *Design and Analysis of Integrated Manufacturing Systems*,Compton, W. D., ed., National Academy Press, Washington, DC, 1988, pp. 167-199.

42. Wang, R. T. R., *Bandwidth Minimization, Reducibility Decomposition, and Triangularization of Sparse Matrices*, PhD dissertation, Computer and Infomation Science Research Center, Ohio State University, Columbus, OH, 1973.

43. Watton, J. D., *Automatic Identification of Critical Design Constraints*, PhD dissertation, Department of Mechanical Engineering, Carnegie Mellon University, Pittsburgh, PA, 1989.

44. Weiler, K. J., *Topological Structures for Geometric Modeling*, PhD dissertation, Rensselaer Polytechnic Institute, Troy, NY, 1986.

45. Woo, T. C., "Interfacing Solid Modeling to CAD and CAM: Data Structures and Algorithms for Decomposing a Solid," *Computer-Integrated Manufacturing*, ASME, New York, 1983, pp. 39-45.

Table of Contents