Concurrent Design

by

S. Finger, M. Fox, F.B. Prinz, J.R. Rinderie

EDRC 24-26-90

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 (eg. 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 presents an overview of a body of research that has resulted from the multi-disciplinary group that is creating this design system. The research falls into four broad categories: 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, reliability, etc. These dimensions are called *perspectives* because each dimension can be thought of as a different way of looking at the design.

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.

The design model is integrated around a shared, domain-neutral representation of the design from which each perspective can extract and reason about features of the design. Constraints are the 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.

This paper presents an overview of a body of research that has resulted from the multi-disciplinary group that is creating this design system. The research falls into four broad categories: geometric modeling, features, constraints, and system architecture. In this paper, we do not present detailed research results which can be found in the individual references.

2. Design Representation

A complete representation of a design 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, and the maintenance procedures. 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.

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. Our architecture does not preclude a perspective from creating its own representation, 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 which are 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.

2.1. 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 [28] discuss the progression from early CAD systems to advanced solid modellers. Voelcker [40] 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 [23] 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[43] and by Gursoz and Prinz[14] 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. Figure 1 gives some examples of non-manifold objects.

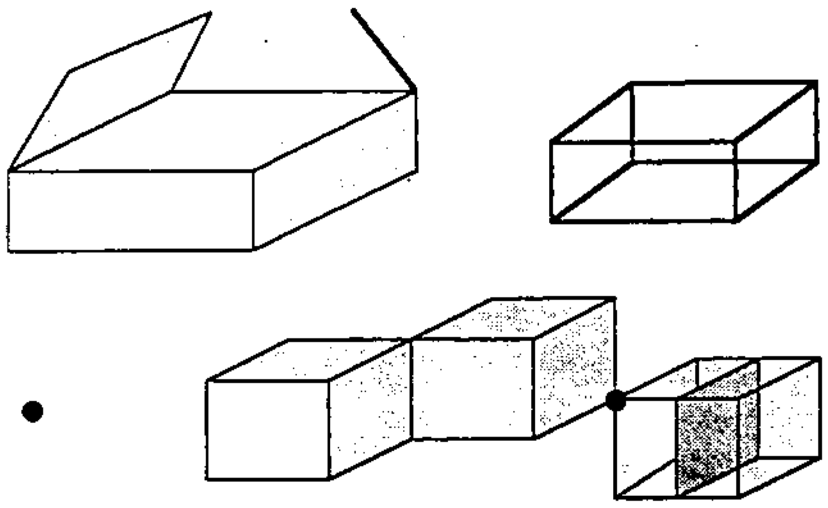
Figure 1: Examples of non-manifold objects

Because one-, two-, and three-dimensional objects can be represented consistently in non-manifold representations, they are highly 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.

2.1. **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; however, the same product design looks quite different when viewed by different expeits. Each perspective emphasizes particular aspects of the design and suppresses certain details in order to evaluate and synthesize. In addition, as the design

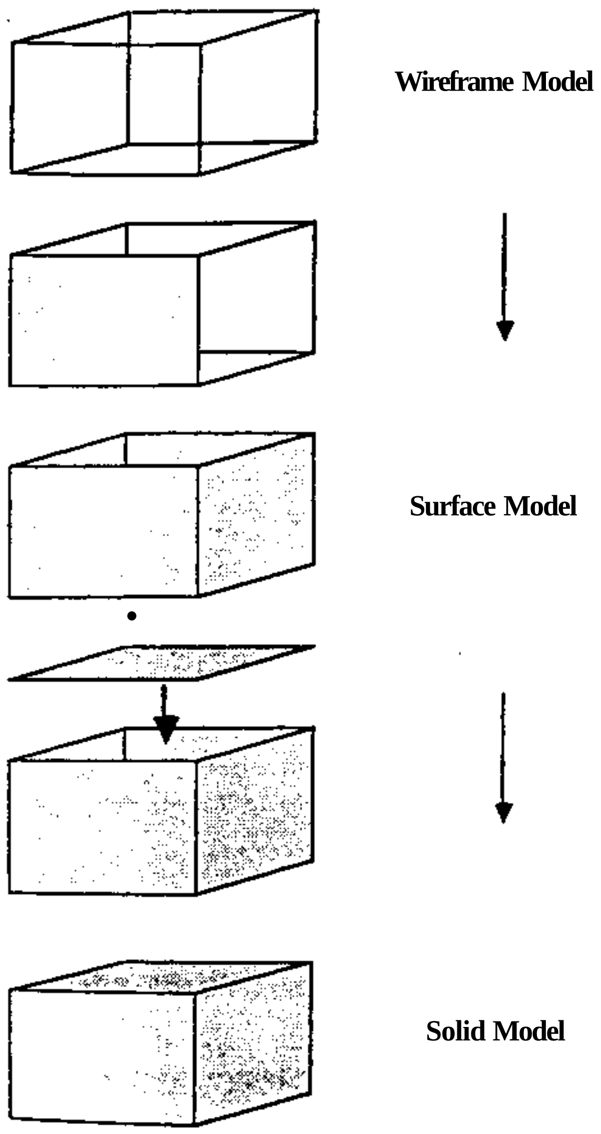

Figure 2: Evolution of a solid model

evolves, so does the view from each of the perspectives; that is, what is emphasized and what is suppressed changes depending on the current state of the design. Figure 3 illustrates a turbine blade as viewed from several different perspectives.

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 objea and the feature definitions both use the non-manifold representation, so features can be recognized by parsing a feature graph against the graph of the object.

Because each perspective views the design differently, each perspective defines its own set of features.

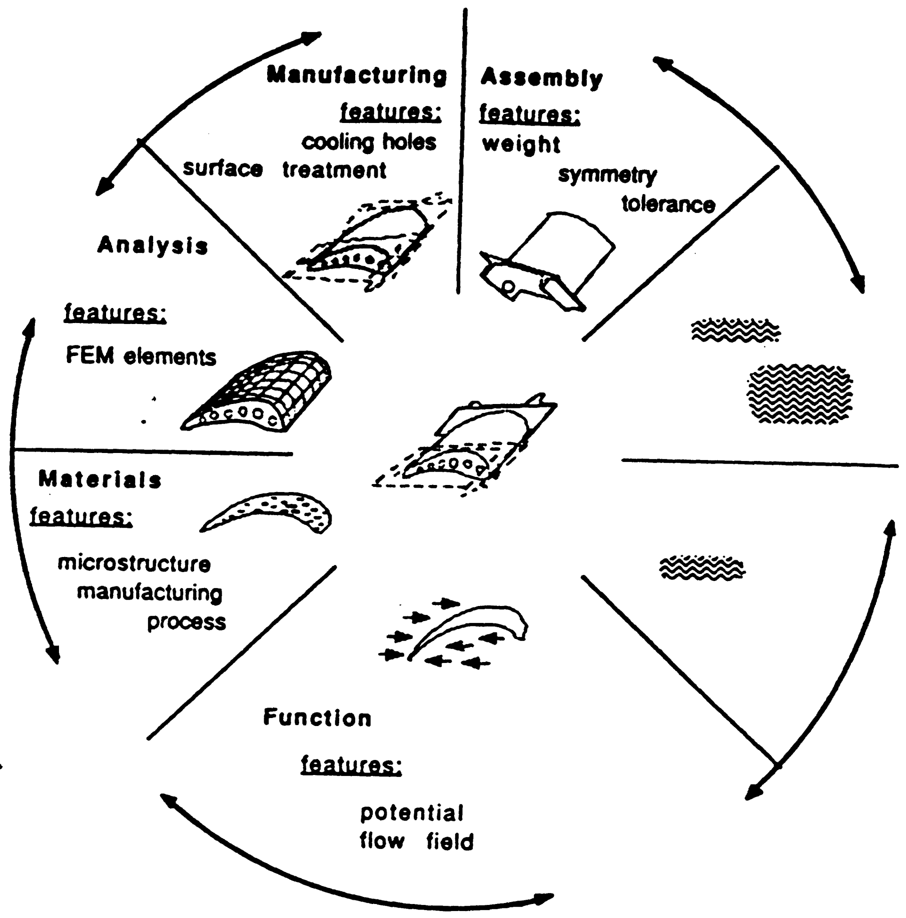

Figure 3: The features of a turbine blade as viewed from several perspectives.

And, because the features are defined in terms of the shared representation, the perspectives can communicate by referring their features to the shared representation.

2.3. 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 which 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.

2.4. 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. On 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.

```
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

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.

3. Features

One motivation for creating feature-based design systems is that a product design is viewed differently as the design evolves and as viewed by different experts. 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 a feature graph against the graph of 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. For a more detailed discussion of the representation and the algorithms, see [91, [25], [30], and [31].

3.L Life-Cycle Features

Features provide a domain-neutral foimalism for extracting information firom a geometric model for life-cycle analyses* Individual life-cycle perspectives view an evolving design through their own set of features and analyze the design based on these features. Because features are defined for specific purposes, the features of one perspective need not have any correspondence to the features defined for another perspective.

In Figure 5, a designer and a manufacturer each define features to view a shared model The designer sees in the artifaa two slots, defined by their width and depth, which serve a functional role in meeting a requirement of the design* The manufacturer is concerned with making the artifact and not only sees the two slots but also the wall created by 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 neutral geometric model is a basis of communication via feature definitions for the two perspectives.

a. Initial design

b. Designer's feature

c. Manufacturer's features

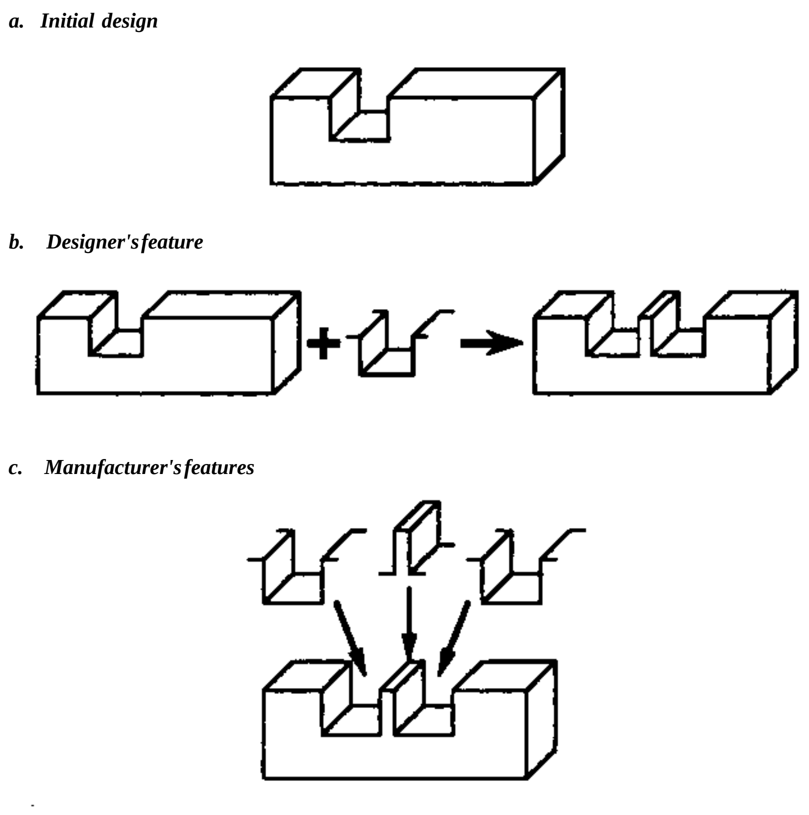

Figure 5: View-point specific feature interaction

Previous research in using features in CAD systems has focused on single domains. Woo [44] utilizes decomposition using form features to perform structural analysis. Several researchers, including Unger and Ray [39], Cutkosky and Tenenbaum [5], Chang et al. [4] and Hayes [16] have explored the use of

features in constructing process plans for parts.

3-2. **Feature Definitions**

Two approaches to using features in design have been explored. One is to restrict the designer to a set of labeled, primitive features to use in the design. A restricted set of features limits the expressiveness of the design system and allows only one interpretation of the design. This approach has the advantage that feature extraction is relatively straightforward because the features on the design can be tracked as they are added.

The other approach is to use a neutral geometric representation of the design and to extract the locations of the features when required. We have taken the second approach because we must support multiple perspectives that each require a different grouping of primitive elements into features.

Figure 6 illustrates several features, all labeled *hole*. From a functional point of view, a designer might specify a hole only by it's centeriine, 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 *re not identical. The difference of perspective for characterizing the concept *hole* necessitates differing feature definitions.

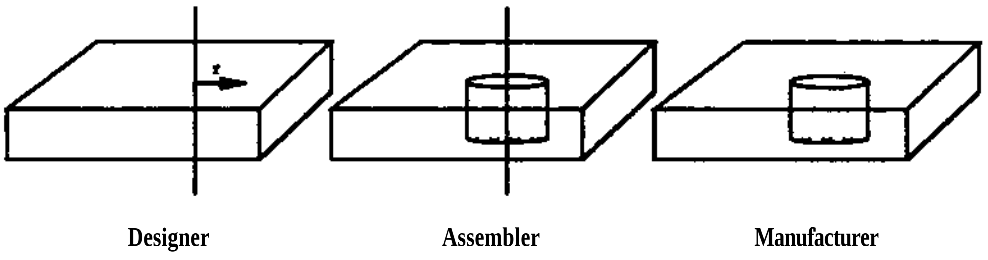

Figure 6: A Hole from three different perspectives

Figure 6 illustrates that the ability to represent both manifold and non-manifold objects is essential in describing partial designs or referring to *non-existent* elements such as center lines or symmetry planes. The design feature, *hole*, is a non-manifold feature defined in terms of its centeriine and radius - lines in 3-space not associated with any surface.

Our approach to defining features is based on Pinilla's [25] work. 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.

Because features can be arbitrarily complex, complete enumeration of all possible features is impractical. A grammatical formalism permits the specification of families of related features. Through the application of transformations to base features, a complex feature can be recognized as an instance of the base feature. For example, Figure 7 depicts a grammar for a slot family. The definition permits the single face representing the bottom of the slot to be split. So, a model that contains a slot with six faces can be recognized as a member of this family through one application of the rewrite rule.

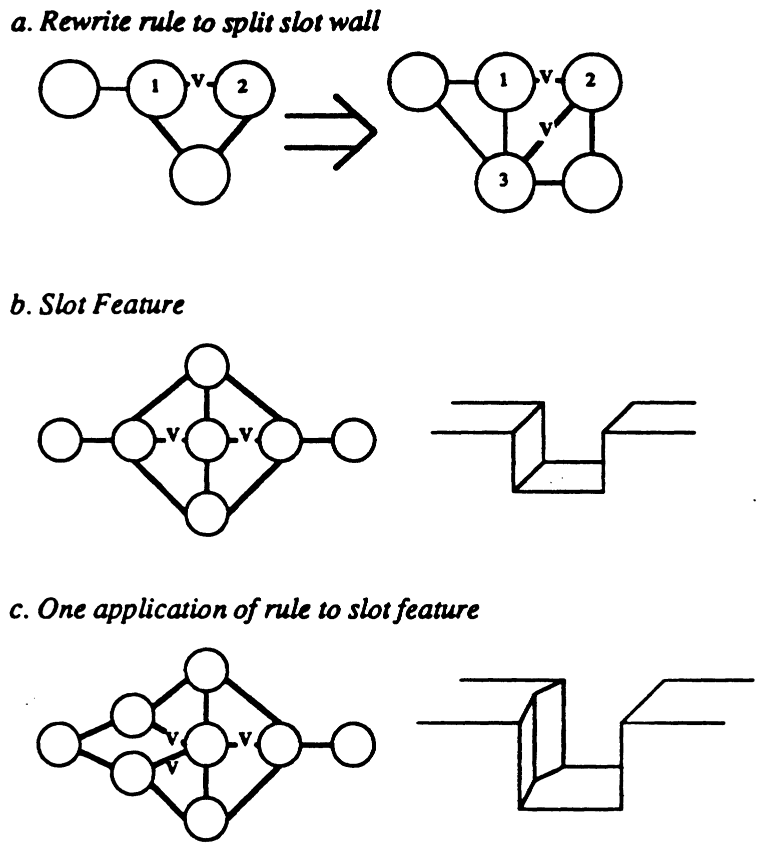
a. Rewrite rule to split slot wall
1
V
2
1
V
2
V
3
b. Slot Feature
V
V
c. One application of rule to slot feature
V
V
V

Figure 7: Definition of a slot feature family

3.3. Feature Recognition

In [30], we present an algorithm for feature extraction. We define and extract features from a solid model. 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.

Several 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. Sakurai and Gossard's system [32] uses graph matching to recognize instances of features taught to the system, but lacks a formal method of defining features. Henderson's work [18] is similar to Gossaid's. Although he does not develop a formal representation, he uses feature-specific heuristics to anchor the search. Falcidicno and Giannini [8] and Floriani [10] use a pattern-directed language to define feature classes in a two-manifold boundary representation.

We use three levels of abstraction for recognizing features. At the lowest level is a non-manifold solid modeller. This level provides complete information for representing a design, including all topological and geometric data about the modeL The intermediate level of abstraction is the augmented topology graph of Pinilla. This level captures the geometric relationships from die input grammar and maintains the 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.

4. 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 desiga 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 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 [42].

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 these bottle-neck constraints.

4.1. Monotonidty Analysis and Interval Methods

Monotonicity **analysis [24] facilitates the** simplification of **a** constraint netwoik and **the** identification of inappropriately **bounded constraint networks.** Unfortunately, most engineering **design** constraints do not exhibit the global monotonicity required for the application of monotonidty 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 informatioa 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 in [21]. 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.

42. 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 OTC another can be determined. In die 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 [37], Mackworth [20], Boming[2], Sussman and Steele[36], Gosling [13], Popplestone [26], Steward [35] 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. .

43. Interval Criticality, Dominance, and Activity

The large number of constraints which arise in a concurrent design environment make it useful to characterize die 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 [19,29].

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.

4.4. 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 [27]. 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.

4.5. 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 [35]. 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 woik of Serrano [34] and Steward [35], subsets of the constraint set can be identified and isolated to be solved simultaneously. The algorithm consists of two stages: *watching* 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-degrce-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 [22], we present algorithms that help to identify the best variables to select to simplify a given constraint problem. We al^o 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 [41]. 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.

5. System Architecture

The role of an architecture is to integrate design methods and algorithms around a shared representation. In particular, an architecture provides a means for dynamically coordinating the application of methods as required by the problem and the designer. For example, integration permits communication between tasks without the designer having to execute multiple routines. A shared representation gives the various modules a common vocabulary, releasing the designer from tasks such as data transformation (e.g. transforming between geometric representations). Integration and a shared representation facilitate *the* solving of large portions of the design problem. Designers are freed to concentrate on the design, not the process.

Two components of the design system are the design database and the control strategy for manipulating the database. The database contains the current representation of the design and its specifications. The control strategy coordinates a set of operations that manipulate the database as the design changes. The choice of these operations depends upon the control strategy and the representation of the data.

Perspectives are represented as knowledge sources that can view the designs in parallel. Once a change occurs and is propagated, each perspective can comment on the change and can also add information to the design.

5.1. Heterarchical Control

The architecture coordinates and manages the multiple design activities. The competing goals of the designer and the different life-cycle perspectives as well as the interactions between specifications **and geometry of the artifact provide many sources of complexity in this system. The architecture** constrains this **complexity by applying a structure to the** system **[11]. Hierarchical structures** provide one or many levels **of** authority; heterarchical structures permits competition **between agents that are** cooperating toward **a** common goaL

Hierarchical structures vest decision-making authority in one or more agents. Simple hierarchies have a single decision maker, either one agent or **a** group of cooperating agents. Uniform hierarchies apply multiple levels of decision-making to filter competing information. Information moves up a uniform hierarchy, while decision-making control propagates down from higher-levels. Brown and Chandrasekaran [3] use **a** hierarchy to manage complexity in **a** system that performs routine design. Specialists that design specific components of an artifact are organized in **a** uniform hierarchy. More specific specialist are invoked from the higher, more general levels in the hierarchy.

Heterarchical systems have many disjoint agents available for particular tasks. Coordination is by negotiation and contract based upon the marginal cost of the task [6,38,33]. Agents compete /or tasks with all other forms of control eliminated between units. Each agent in the heterarchy pursues its own goals in correspondence to the needs of another. Heterarchical systems provide a structure for problem-solving in design systems with multiple perspectives. Negotiations between multiple competing perspectives decide particular strategies to pursue in the design process. A shared representation of the design supports inter-perspective communication.

Design agents can be organized in either hierarchical or heterarchical structures that determine the decision making process, but the problem of coordinating these agents remains. Systems that plan coordinate the activities of the agents by exploring and evaluating alternate design processes before committing to an acceptable one [12]. Systems must also schedule the activities of the agents, selecting appropriate agents to evaluate and participate in the design process at appropriate times [17,3]. The architecture must provide either planning, scheduling or a combination of both to coordinate the activities of design agents.

The role of an architecture is to integrate partial solutions around a central representation. Problem-solving architectures are composed of a database and a method of coordinating operations on the database. Semantic networks provide **a** representation for encoding qualitative and quantitative design data in a shared database. Hierarchical and heterarchical structures can be used to coordinate the access of multiple design agents to the database.

5.2. Black Board Architecture

Our architecture, based upon the blackboard model [7], is shown in Figure 8. Each design version, composed of features connected by constraints, is represented in the *design blackboard.*

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. Inconsistent requirements, names, and definitions are contained within the perspectives because the communication is through the shared representation. Inconsistencies and conflicts in goals inevitably arise during the design process. These inconsistencies are tolerated by the system but are also tracked.

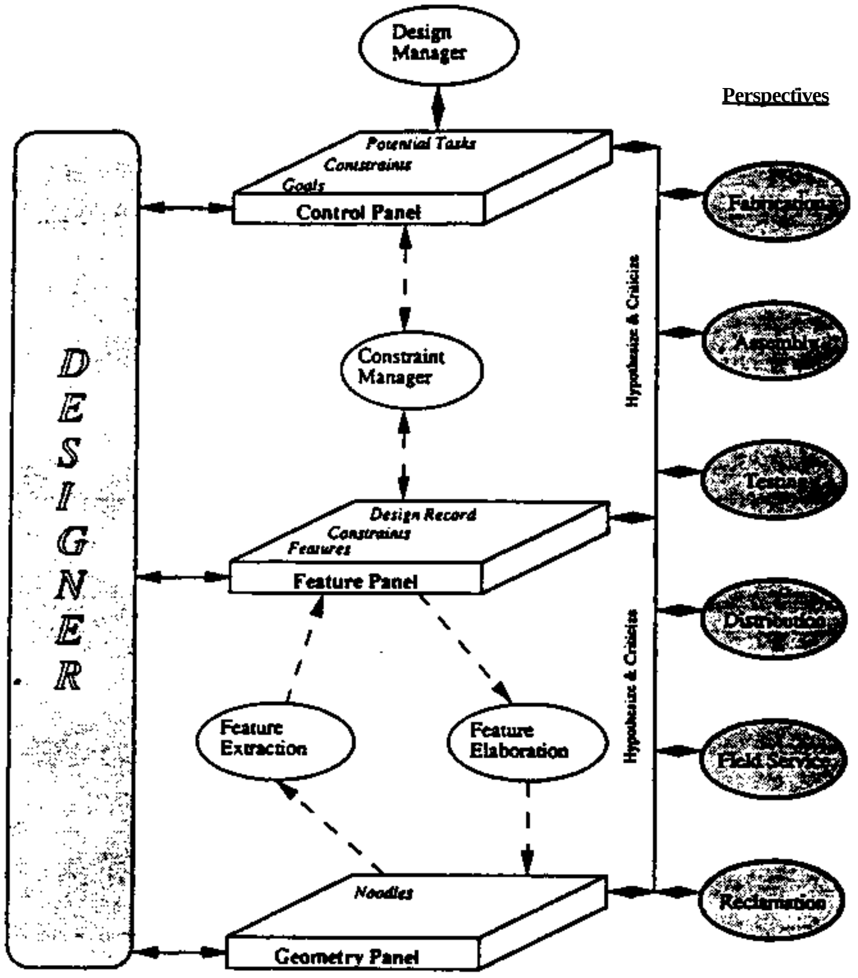

Figure 8: Design Fusion system architecture

The designer is notified of inconsistencies when appropriate.

The architecture uses conceptual knowledge about the design to support the design task. 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 decision, and the information upon which it was based. This information can be used to maintain design consistency when undeilying assumptions of the design change or to track constraint violations back to the sources.

6. 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.

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. R and Ullman, J. D., *Data Structures and Algorithms,* Addison-Wesley Publishing Company, Reading, Massachusettes, 1983.

2. Boming, A^ "ThingLab- A Constraint Oriented Simulation Laboratory," Technical report, Xerox Palo Alto Research Center, 1979.

3. Brown, D. C. and Chandrasekaran, B., "Knowledge and Control for a Mechanical Design Expert System," *IEEE Computer,* VoL 19, No. 7, July 1986, pp. 92-100.

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. Davis, R. and Smith, R. G., "Negotiation as a Metaphor for Distributed Problem Solving," *Artificial Intelligence,* VoL 20,1983, pp. 63-109.

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

8. 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.

9. Finger, S. and Sailer, S. A., "Representing and Recognizing Features in Mechanical Designs," *Second International Conference on Design Theory and Methodology, DTM ⁹90,* ASME, Chicago, September, 1990.

10. 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.

11. Fox, M. S., "Organization Structuring: Designing Large Complex Software," Technical report CMU-CS-79-155, Carnegie Mellon University, December 1979.

12. Gero, J. S. and Coyne, R. D., "Knowledge-Based Planning as a Design Paradigm," *Design Theory in Computer-Aided Design, Proceedings of the IFIP WG 5.2 Working Conference 1985, Tokyo,* Yoshikawa, H. and Waiman, E. A., ed., North Holland, Amsterdam, 198S, pp. 261-29S.

13. Gosling, J., *Algebraic Constraints,* PhD dissertation, Department of Computer Science, Carnegie-Mellon University, 1983.

14. Gursoz, E. L., Choi, Y. and Prinz, F., "Vertex-based Representation of Non-manifold Boundaries," *Second Workshop on Geometric Modeling,* IFIP, New Yoik, 1988.

15. Gursoz, E. L. and Prinz, F. B., "Comer-Based Representation of Non-manifold Surface Boundaries in Geometric Modeling," Technical Report, Engineering Design Research Center, Carnegie Mellon University, 1989.

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.

17. Hayes-Roth, F. and Lesser, V. R., "Focus of Attention in a Distributed Logic Speech Understanding System," *Proceedings of the International Joint Conference on Artificila Intelligence,* 1977, pp. 27-35.

18. **Gavankar, P., Chuang, S. H.9 Henderson, M. R. and 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, January, 15-17** 1990, pp. **167-183.**

19. **Krishnan, V., Navinchandra, D., Rane, P. and Rinderie, 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. *YL,* "Consistency in Netwoik Relations," *Artificial Intelligence,* Vol. 8,1977, pp. 99-118.

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

22. Navinchandra, D. and Rinderie, J. R., "Interval approaches for Concurrent Evaluation of Design constraints," *1989 ASME Symposium on Concurrent Product and Process Design,* American Society of Mechanical Engineers, San Francisco, December 1989.

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

24. Papalambros, P. J. and Wilde, D. J., *Principles of Optimal Design,* Cambridge University Press, New Yoric, 1988.

25. 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, June 11-14,1989, pp. 285-300.

26. 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.

27. Ratschek, H. and Rokne, J., *New Computer Methodsfor Global Optimization,* Ellis Horwood Limited, Chichester, England, 1988.

28. Requicha, A. A. G. and Voelcker, H. B., "Solid Modeling: A Historical Summary and Contemporary Assessment," *IEEE Computer Graphics and Applications,* March 1982, pp. **9-24.**

29. Rinderie, J. R. and Krishnan, V.f "Constraint Reasoning in Concurrent Design," *Submitted to ASME Design Theory and Methodolgy DTM '90,* ASME, 1990.

30. Safier, S. A. and Finger, S., "Parsing Features in Geometric Models; An abstract," *SIAM Conference on Geometric Design,* **Tempe, AZ,** November 1989.

31. Safier, S. A. and Finger, S., "Parsing Features in Geometric Models," Technical Report, Robotics Insitute, Carnegie-Mellon University, January 1990.

32. Sakurai, H. and Gossaxd, 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.

33. Sathi, A., and Fox, M.S., "Constraint Directed Negotiation of Resource Reallocations," *Distributed Artificial Intelligence II,* Morgan Kauftnann, Los Altos, CA, 1989.

34. Serrano, D., *Constraint Management in Conceptual Design,* PhD dissertation. Department of Mechanical Engineering, Massachusetts Institute of Technology, 1987.

35. Steward, D. V.f "Partioning and Tearing Systems of Equations," *Journal of SIAM, Numerical Analysis Series B,* Vol. 2, No. 2,1965.

36. Sussman, G. J. and Steele, G. L., "CONSTRAINTS- A Language for Expressing Almost

Hierarchical Constraints," *Artificial Intelligence** No. 14, 198a pp. 1-39.

37. Sutherland, I. E., "Sketchpad - A Man-Machine Graphical Communication System," Technical report #296, MIT Lincoln Lab. Cambridge, Massachusetts, 1983.

38. Sycara, K., "Resolving Goal Conflicts via Negotiation," *Proceedings of the Seventh National Conference on Artificial Intelligence.* St Paul, MN, 1988, pp. 245-250.

39. 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.

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

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

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

43. Wetter, K. J., *Topological Structures for Geometric Modeling,* PhD dissertation, Renssalaer Polytechnic Institute, 1986.

44. Woo, T. C, "Interfacing Solid Modeling to CAD and CAM: Data Structures and Algorithms for Decomposing a Solid/' *Computer-Integrated Manufacturing,* ASME, New Yoric, 1983, pp. 39-45.

Hierarchical Constraints," *Artificial Intelligence*, No. 14, 1980, pp. 1-39.

37. Sutherland, I. E., "Sketchpad - A Man-Machine Graphical Communication System," Technical report #296, MIT Lincoln Lab. Cambridge, Massachusetts, 1983.

38. Sycara, K., "Resolving Goal Conflicts via Negotiation," *Proceedings of the Seventh National Conference on Artificial Intelligence*, St. Paul, MN, 1988, pp. 245-250.

39. 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.

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

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

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

43. Weiler, K. J., *Topological Structures for Geometric Modeling*, PhD dissertation, Renssalaer Polytechnic Institute, 1986.

44. 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.