6

Design Fusion: An Architecture for Concurrent Design

MARK S. FOX, SUSAN FINGER, ERIC GARDNER,
D. NAVIN CHANDRA, SCOTT A. SAFIER and MICHAEL SHAW
Center for Integrated Manufacturing Decision Systems, Carnegie Mellon University, Pittsburgh, PA 15213, USA

INTRODUCTION

Recent research in design has incorporated artificial intelligence (AI) problem-solving architectures in the construction of systems that aid designers. For many people outside the field of AI, the role of a problem-solving architecture is unclear. It is our intent, in this chapter, to both motivate the need for architectures in design and demonstrate their application in one particular system: Design Fusion.

Design is a complex process; designers bring to bear a variety of methods and techniques. They have many tasks to perform and numerous sources of design data. There are algorithmic solutions to several tasks, but they each solve only part of the design problem. None is an entire solution in itself, and very few are integrated. It is human expertise that integrates these design resources, provides the missing pieces, and guides the process.

The role of a problem-solving architecture is to integrate design methods and algorithms around a shared representation. In particular, an architecture provides a means for dynamically coordinating their application as required by the problem and the designer. For example, integration permits communication between tasks without designers having to execute multiple routines. A shared representation gives the various modules a common vocabulary, releasing designers from the task of data transformation (e.g. transforming between coordinate systems). Integration and a shared representation facilitate the solving of large

Knowledge Aided Design
ISBN 0-12-298250-9

portions of the design problem. Designers are freed to concentrate on the design, not the process.

The platform for our exploration of problem-solving architectures for design is Design Fusion, a system whose goal is to support concurrent design (Finger *et al.*, 1992). 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 multidimensional space in which each dimension highlights a different life-cycle objective such as fabricability, testability, serviceability, reliability, etc. These dimensions are called *perspectives*. Each perspective can be thought of as a different way of looking at the design.

Design Fusion is a computer-based design system that enables a designer to consider concurrently the interactions and trade-offs among different, and even conflicting, requirements, arising from one or more life-cycle perspectives. It 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). 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.

In the following, we first review requirements for a problem-solving architecture as motivated by the design task. We then provide an overview to our architecture and its relationship to other work. Next a detailed description of the architecture is provided, focusing on representation, constraints, protocols and version management, followed by a description of the "design manager", which controls the direction of design problem solving. Lastly we provide a detailed trace of the operation of the Design Fusion system.

REQUIREMENTS

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 behaviour exhibited by designers during this process has been characterized as problem-solving (Simon, 1968; Akin, 1986; Gero and Coyne, 1987). The components of a problem-solving system are a database, procedures, inference rules and a control strategy. The database contains a representation of the design and its specifications. The control strategy selects and applies procedures and rules that modify the database. To support designers, the architecture of the system should incorporate a general model of problem-solving.

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. Second, the architecture provides a group problem-solving environment in which knowledge-based systems contribute to the design process. In this section, we will examine several design behaviours and identify requirements for an architecture to support these roles.

At the minimum, the architecture should reduce the "drudgery" of the design process. There are many computational tools available to the designer for analysis and deduction. *The architecture should be able to aid in the selection of the most appropriate tool to apply, "massage" the design data to conform to the tool's input requirements and apply it.*

Design is often a search process that elaborates portions of a design in a top-down manner, decomposing the design problem into smaller more manageable subproblems. Subproblems are further decomposed until solvable problems are found. At any particular time during the design process, the solving of a subproblem is the goal of the designer, with the ultimate goal being the creation of an artefact or description of an artefact. To satisfy goals, designers make changes to the design. The design progresses through multiple levels of abstraction and refinement until all goals are satisfied. *The architecture should support the process of goal formulation and decomposition, and the selection of subtasks upon which to focus design attention.*

Early works of Alexander (1965) and Simon (1968) assume this model of design—that design is a nearly-decomposable problem with little or no interaction between design subproblems. Alexander later concludes that such problems tend to be artificial—natural problems contain many interactions (Alexander, 1968). During problem-solving, designers must cope with interactions between specifications, the goals of the designer,

[1] Examples of knowledge-based systems that make design decisions include XCON (Bachant and McDermott, 1984), PRIDE (Mittal *et al.*, 1985) and ALADIN (Hulthage *et al.*, 1990).

and the design decisions made during the design process. *The architecture should provide support for the management of these interactions. It should identify which subtasks interact and what impact decisions will have.*

Interactions occur between design goals that are closely related (Mostow, 1985). Cooperation occurs when there is a dependence between interacting goals, or one goal is a generalization of others. Competition arises when satisfaction of one or several goals precludes the satisfaction of others. For example, consider the turbine blade in Figure 1. Its design has both functional requirements (e.g. lift weight and efficiency) and physical requirements (e.g. expected life of the blade). In the latter case, there is a relationship between fatigue and life, and between stresses (steady-state and vibratory) and fatigue. Designers understand these relationships and have techniques to resolve the interactions. A designer may perform a steady-state stress analysis, estimate life, evaluate the design based upon the results, and modify the physical characteristics of the artefact until the life requirement is approximated. Only at this point does the designer begin to consider vibratory stress, and repeats the process.

Designers must make trade-offs when design decisions conflict with other decisions or specifications and the designer's understanding of current trade-offs affects the overall design process. *The design system should provide support enabling options to be evaluated, preferences to be specified, and constraints to be relaxed* (Fox, 1983; Navin chandra, 1987).

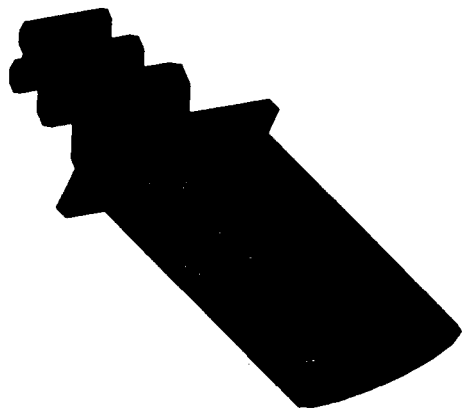

Figure 1. Sample turbine blade.

For example, Rinderle and Watton (1987) discuss one technique to evaluate critical design relationships. They propose transformations between variable systems as a means of understanding these critical relationships. Alternative variable systems are chosen for physical and functional characteristics. Transformations give the designer alternative view points of the design, providing insight.

Opportunistic problem solving focuses attention on tightly constrained portions of the (design) problem (Hayes-Roth and Lesser, 1977). Resolving these portions of the design problem creates *islands of certainty* which can be exploited in other areas of the design. Consider our example of a turbine blade design. It has very specific aerodynamic characteristics that tightly constrain the shape of the airfoil. Once the airfoil has been designed, decisions that constrain other areas of the design can be propagated. For example, the length of the blade affects its fatigue life. When designing to meet a fatigue life specification, focusing the problem-solving process on blade length is reasonable.

Opportunities can be exploited in three ways. First, there is opportunism in elaboration. Ullman *et al.* (1987) suggest that designers add detail to one area of the design while leaving other areas at various incomplete levels. Opportunism permits movement between areas that have the greatest degree of certainty. *An architecture should support the identification and focusing on the portions of design that are most certain and/or constrained.*

Second, past designs can be opportunistically exploited during design. Finger *et al.* (1988) identify four types of design: selection, routine design (e.g. configuration, parameterized), extrapolation, and novel. When there is a direct mapping between the specification and an existing part, the design task is selection. A design is routine when similar artefacts have been constructed, providing appropriate problem-solving methodologies. Extrapolation is using the process of a past design to guide the creation of a new but similar artefact. Extrapolation indexes directly into past designs, exploiting them during the design process. Routine design too can be viewed as combination of selection, constraint analysis and indexing into prior designs or generalizations of prior designs. *An architecture should support opportunistic selection from past designs, choosing between components of past designs to create new but similar components in the current design.* This has come to be known, more recently, as "case-based" design (Navin chandra, 1988; Sycara and Navin chandra, 1989).

In the last decade, industry has recognized the effects of isolating design from life-cycle concerns. This realization has lead to the formation of design teams which bring together the expertise of people familiar with life-cycle activities so that a better design is generated. In parallel, research in design automation has also turned towards the incorporation

of life-cycle concerns. This has taken the form of compilations of knowledge that are used to critique a design once generated; for example, the Design for Assembly work of Boothroyd (Boothroyd and Dewhurst, 1983). *The architecture should support the opportunistic viewing of the design from multiple perspectives*. Each perspective is a model of some life-cycle activity, such as manufacturing or service, providing databases and computational routines to support that perspective's design effort. Movement between these perspectives should be opportunistic, allowing any one to participate actively in the design while the others evaluate and critique the design effort. This movement permits application of algorithms and heuristics at appropriate portions of the design process.

The architecture of a design system coordinates the group problem-solving activity of these perspectives, including coordinating negotiations between competing perspectives, and providing a vehicle for the perspectives to view and modify the design. *A shared representation also supported by the architecture provides the perspectives with the vocabulary to evaluate the design and conduct negotiations* (Sycara, 1990). It should be consistent over these multiple viewpoints and interpretable by them, allowing answers to a set of questions at a level of complexity appropriate to the design task.

The representation should be precise and complete, capable of representing all interesting aspects of the artefact at various levels of abstraction. Included in a design representation must be:

- *Function*. Artefacts are designed to meet some desired behaviour. A representation of function would permit reasoning about an artefact's behaviour and its role in a complex system.
- *Geometry and topology*. Behaviours are realized through geometries, which must be represented by the system.
- *Design constraints*. An explicit representation of design constraints enables opportunism when evaluating design decisions.
- *Goals and specifications*. The representation must track the current goals of the design process, and evaluate design decisions against desired specifications.
- *Design evolution*. The representation must be capable of managing an evolving design, and permitting the designer to back-up to prior instantiations of the design.

The architecture must also provide support for an evolving design. As a design evolves, multiple versions are created over time. Katz defines a configuration as a collection of versions of partial designs (Katz, 1985). The architecture should manage the changing design, and allow the designer to view any portion of the design history, and revert to a prior design if necessary. Also, inconsistencies in the design inevitably arise during the design process. These inconsistencies should be tolerated by the system, but also tracked. The designer should be notified of inconsistencies when appropriate.

The architecture of a computer-assisted design system should also represent the current focus of attention *of the designer, and coordinate the activities of the system towards this focus*. The designer's focus of attention affects the manner in which the design is evaluated. The system should share this focus, and use it to select the subset of its data that is relevant to the designer's current activity.

Grosz (1986) identifies three requirements for representing focus:

1. The ability to distinguish between data relevant to a particular task and that which is superfluous.
2. Recognize objects of implicit focus. When the designer is focused on a portion of the design, its subcomponents and components which it interacts with are implicitly under focus.
3. Provide a mechanism for shifting focus. As the design changes, the focus of attention of the designer will shift. These shifts should be recognized by the system and be reflected in the system's behaviour.

The architecture should provide mechanisms for each of these requirements.

Consider the turbine blade design task. A designer focusing on structural properties will see different design features than when the focus is on aerodynamic properties. If the designer is analysing stress concentrations in the shank, the system should not point out problems with the manufacturability of the airfoil. The system may, however, comment on stress concentrations in the fir-tree portion of the shank or between the shank and the platform, which are both implicitly under the designer's focus of attention.

In summary, the role of the architecture is to integrate partial solutions to design problems around a shared representation. It should support problem solving by managing interactions between design constraints, providing the designer with data to facilitate making trade-offs. The system should also exploit tightly-constrained areas of the design, opportunistically elaborating areas at various levels of abstraction, using past designs to guide the current process, and moving between multiple perspectives that affect the design. A common representation enables these perspectives to view and comment about the design. Lastly, the system should share a common focus of attention with the designer, enabling the system to bring appropriate information and techniques to the design problem.

OVERVIEW OF DESIGN FUSION ARCHITECTURE

The novelty of the Design Fusion architecture lies not in the incorporation of a unique representation or method, but in the integration of representations and methods, as identified by our requirements, in a single operational system. In essence, Design Fusion has become a testbed for the integration of individual design representations and methods. In order to ascertain the efficacy of the Design Fusion architecture, we have applied it to the design of a jet engine blade and a robot arm, and it is currently being applied to the design of a power transformer.

The Design Fusion architecture is based on the blackboard model of problem solving (Erman *et al.*, 1980; Nii, 1986a,b) illustrated in Figure 2. The architecture has four major components: the blackboard, knowledge sources, search manager, and user interface.

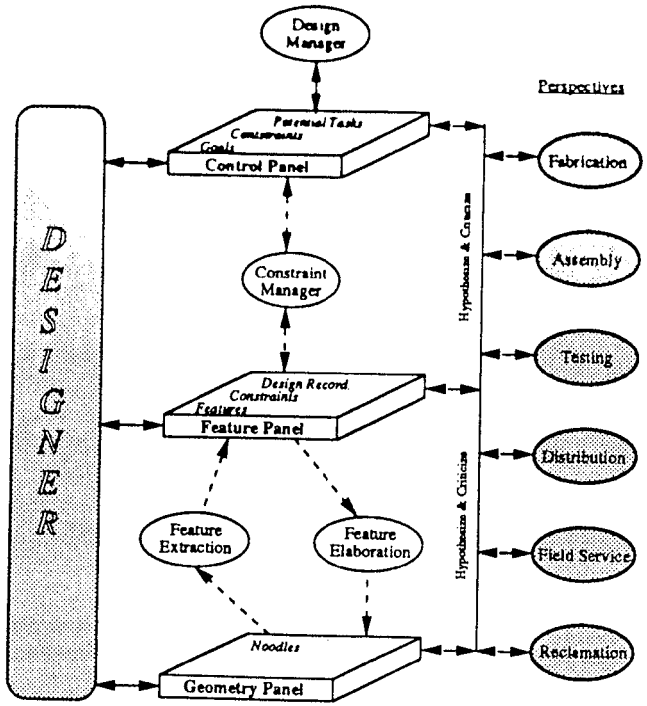

Figure 2. Design Fusion system architecture.

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 lifecycle. Each perspective may criticize design decisions or generate new design information. Use of 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. Inconsistencies and conflicts in goals inevitably arise during the design process. These inconsistencies are tolerated by the system but are also tracked. The designer is notified of inconsistencies when appropriate. *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, several perspectives may have contributions to make. The search manager must decide the sequence of contributions and control their execution. The system keeps a record of design decisions that led to the creation of a constraint or feature. Design records are defined by the perspective that 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 consistency when underlying assumptions of the design change, or to track constraint violations back to the sources.

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 can also confirm or override the system's suggestions at each stage in the search manager's decision cycle.

In the remainder of the chapter, we will examine these four components of the system in more detail, followed by a detailed example of the system designing a fan blade.

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 could post the constraint on the blackboard.

If a complete representation of a design could be constructed, it would include the following attributes: 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.

The shared representation of the design artefact spans three panels: geometry, feature and control. The following defines what is represented on each.

Geometry panel

The geometry panel focuses solely on the representation of the artefact's 3D geometry. The geometry of the artefact is a neutral representation of the evolving design. Design Fusion employs a non-manifold geometric representation. The representation of geometry has been an active area of research over the last 15 years. In a review paper, Requicha and Voelcker (1982) discuss the progression from early CAD systems to advanced solid modellers. Voelcker (1988) 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. We have found non-manifold representations well-suited for capturing an evolving design, where part of the model may be a true solid while other parts are still 2D line sketches. Discussions along similar lines can be found in Nielsen *et al.* (1987) as well as in Gursoz *et al.* (1990).

In surface boundary representations, known as *b-reps*, objects are modelled 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 modelled 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 modelling systems created by Weiler (1986) and extended by Gursoz *et al.* (1990) 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 centre line of a hole, a parting plane for a mould, and internal boundaries for a finite-element mesh, as well as the enclosing shell of the designed object. Figure 3 shows the evolution of the construction of a solid from a wireframe in a non-manifold representation.

Feature panel

A major problem with geometric representations is that they provide no information about the artefact's function and no method of reasoning about the life-cycle effects on the artefact. A geometric representation must be augmented to permit such reasoning. The following describe three types of qualitative knowledge represented in the feature panel: features, constraints and the design record.

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 and assembly planners, or by the rule-based systems that emulate their

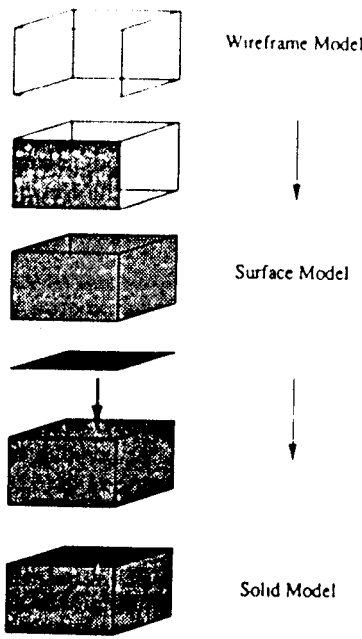

Figure 3. Evolution of a solid model.

activities. Experts often abstract geometry into features like ribs, parting planes, and chamfers (Dixon *et al.*, 1987) which provide for a qualitative hierarchical description of the artefact.

In the Design Fusion system, features are the primary representation by which perspectives communicate their beliefs and suggestions. Each component, subassembly and assembly is defined by a single feature, which in turn is refined into subfeatures. Consequently, the design artefact is defined by a network of features.

Parameters represent various values associated with features. Parameters may represent geometric characteristics such as width or height, computed physical characteristics such as weight or volume, or computed behavioural characteristics such as volume flow for a turbine blade.

Figure 4 depicts a partial representation of a fan blade. The *part-of* link provides a device by which a feature hierarchy is built. The *revises* link provides the mechanism by which design histories are stored. These links are created by the blackboard functions that create new branches in the design evolution tree. The *constrains* link connects a constraint node to the two or more parameters and/or specifications that it constrains. The

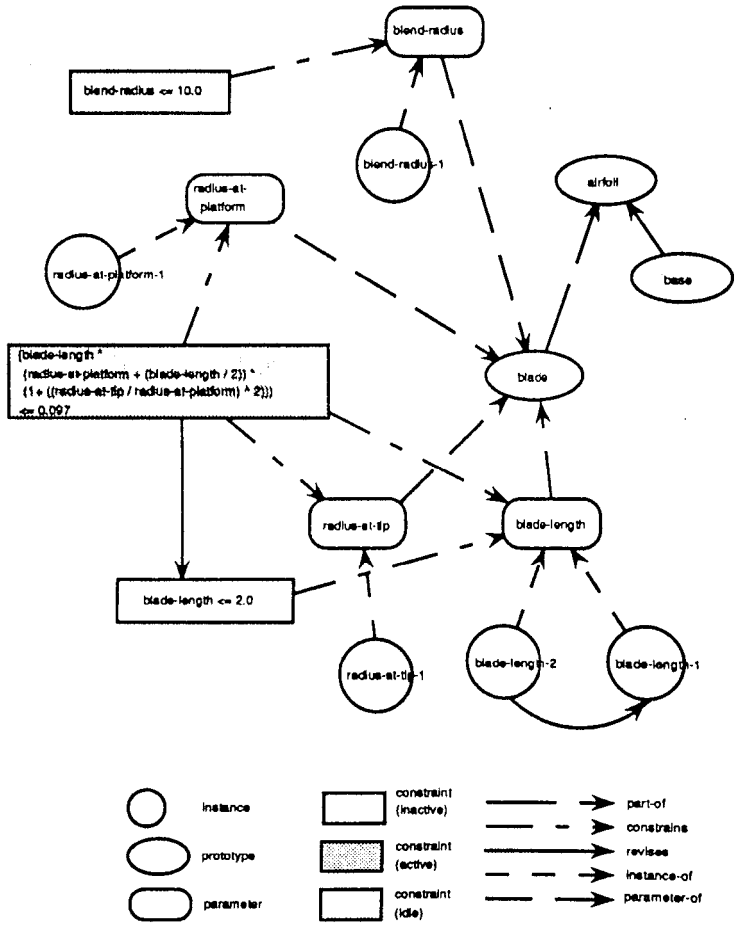

Figure 4. Sample design evolution tree.

instance-of link connects an instance of a blackboard entity to its prototype. This link is created by the blackboard protocol functions: bb-post, bb-assert, bb-revise, bb-select, bb-derive and bb-refine. The *parameter-of* link connects a parameter prototype to the feature that it is associated with.

For all features on the feature panel, there exists a mapping to the geometry on the geometry panel that it describes. Many features are only used locally within a perspective. 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 geometric 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 5, 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 artefact and not only sees the two slots but also the wall created between them. A manufacturing analysis of this wall indicates that it it 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.

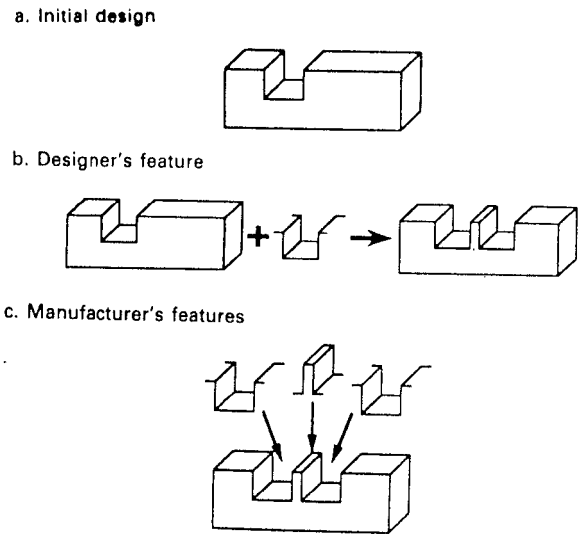

Figure 5. Viewpoint-specific feature extraction.

Constraint representation

The representation shared among perspectives must include not only the evolving product geometry and features, but must also include the allowable limits on geometry, the relationships among behaviour 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.

Constraints have three distinct states:

- *Idle constraints* have not yet been introduced to the design process. For example you would not want to consider a constraint on blend radius if no value for blend radius has yet been selected.
- *Active constraints* have been introduced to the design process and are in the set of constraints currently under consideration.
- *Inactive constraints* have been introduced to the design process but are not in the set of constraints currently under consideration.

The terminology used to refer to state changes of constraints is as follows:

- *Activation* refers to the change of state from idle to active.
- *Postponement* is the process whereby an active constraint becomes inactive.
- *Reactivation* refers to the change of state from inactive to active.

Constraints represent physical laws that constrain various aspects of the artefact being designed (i.e. design specifications, parameter values, and fabrication and life-cycle requirements). The various types of constraints are numeric-implicit, algebraic-equality, algebraic-inequality and predicate.

- *Numeric-implicit* constraints are unidirectional or black-box constraints. Given values for the set of input variables you can obtain values for the output variables. An example of this would be a finite-element analysis of the geometric model.
- *Algebraic-equality* constraints are bidirectional and are expressed

as algebraic equality functions. For instance the efficiency of a turbine blade can be expressed as an algebraic equality in terms of the rotor loss coefficient, the stator loss coefficient and the theoretical pressure coefficient.

- *Algebraic-inequality* constraints are bidirectional and are expressed as algebraic inequality functions. For instance, in the design of turbine blades you may want to assert that the blend radius be less than or equal to 0.0015 m. You might also want to constrain the blade length by specifying it as an algebraic inequality in terms of radius-at-tip and radius-at-platform.
- *Predicate* constraints are unidirectional and are expressed as a function whose input is a set of keyword value pairs corresponding to the constrained parameters.

Design record representation

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 underlying assumptions of the design change or to track constraint violations back to the sources.

Design Fusion maintains a *design evolution tree*. The design evolution tree is useful for:

- Tracking design history.
- Reverting to a prior design.
- An explanation facility. A facility such as this would be able to explain how the propagation of constraints leads to the assignment of various feature attributes.
- Dependency-directed backtracking from constraint violations.

Every node of the design evolution tree is annotated with its design history number. The design history number is initialized to zero. When a new branch of the design evolution tree is created, the design history number is incremented and that value is assigned to every new node.

The current geometric modeller, Noodles, does not have any mechanism for maintaining design history. Since the physical *features* reside solely within Noodles, we need some mechanism to handle backtracking as it pertains to the geometric model. When backtracking, we could simply reassert whatever was revised by the node that is being backtracked. For example, in Figure 4, if we were to backtrack such that *length'* was no longer valid then we would simply have to reassert *length* to Noodles causing a new geometric model to be calculated.

Design versions are used to keep track of different versions of the

design. They are distinct from design histories in a conceptual sense. Whereas design histories are useful for tracking each significant step in the design process, design versions are used to track larger milestones. This is the mechanism by which prior designs are maintained.

Reason maintenance

Design Fusion utilizes a reason maintenance system (RMS) to ensure that all assertions on the blackboard have valid justifications. All of the first-level blackboard protocol functions (bb-assert, bb-post, bb-revise, bb-select, bb-refine, and bb-retract) take a list of justifications as an argument. Justifications are other entities that must currently be on the blackboard. The RMS is called immediately following any changes to the blackboard. Each time the RMS is called, it verifies that every justification for each entity on the blackboard is itself still on the blackboard. When the RMS encounters an entity for which this condition does not hold, it queues that entity for deletion by calling bb-retract.

Control panel

The control panel maintains a description of the state of the design process. The state description is used by the design manager knowledge source to decide what activity to perform next. The following information comprises the state description:

- *Phase description*. Design Fusion cycles through four phases: Perspective monitor checking, Perspective prioritization, Perspective selection and invocation, and Constraint checking.
- *Invocation list*. Design Fusion maintains a list of perspectives that can be invoked.
- *Process trace*. In order to identify cycles in the reasoning process. Design Fusion maintains a process trace that records each perspective invocation, the features that caused the invocation and the features that were added/modified.

KNOWLEDGE SOURCES

Opportunism in problem solving can be viewed as pattern-directed application of heuristics and procedures. Changes in the design give rise to many cooperating and conflicting opportunities. In order to prevent opportunities from degenerating into chaos, the architecture must organize the exploration of alternatives posed by them.

Perspectives

The first step towards organization is to encapsulate this knowledge of opportunities so that the architecture can intelligently select which to apply. Perspectives give rise to the need to modularize the procedures and heuristics applicable to one particular view. The specification of a perspective, including the designer's perspective, must include its function, its desired input and expected outputs, and the validity and certainty of its results. For example, the function of the manufacturing perspective would be to determine whether an artefact could be fabricated. Its input would be the geometry of the artefact, and its output would be confirmation of its manufacturability or a list of problems with the current design associated with a degree of confidence in this assertion.

Perspectives embody a particular, life-cycle view of the design artefact. They may be comprised of tables, algorithms and/or heuristics, and may criticize proposals and decisions placed on the blackboard and/or make decisions and place them on the blackboard. A perspective is composed of two parts: one or more monitors that specify a pattern on the blackboard that will cause the perspective to examine whether it wishes to respond to the new information by performing an action, and the action component that may be invoked by the design manager.[2] With these representations, a perspective does the following:

1. Recognizes features relevant to its goals.
2. Informs the design manager of its wish to perform a task.
3. If selected by the design manager, it performs its task and updates the blackboard.

Perspectives included in the first application of Design Fusion to fan blade design include: Aerodynamics, Structures, Manufacturing and the Designer.

Methods

While perspectives may opportunistically request invocation during the design process, methods are designed to participate as a step in the overall Design Manager's control cycle. In each iteration, the Constraint Manager prioritizes requested perspective invocations, selects a perspective, invokes it, checks constraints and then extracts/revises features (the latter has been in theory but not in practice).

[2] This is similar to what was defined by the Hearsay-II system (Erman *et al.*, 1980).

Design manager

It is the responsibility of the Design Manager to control the design process. At present we run the system "open loop"; it is the human designer who has the final say as to which perspective is to be invoked next. In future versions, we plan to give the Design Manager greater control over the design process.

The Design Manager is implemented as a rule-based system in which the rules manage the design cycle (i.e. prioritize perspectives, select, etc.), while other rules provide domain knowledge relevant to each step in the cycle. Consider the rule

IF there is a constraint on the life-time of the blade;
AND there is a proposed geometry for the shank;
AND the stress concentrations in the shank are unknown;
THEN Invoke the Structures perspective to perform a finite element analysis on the shank geometry

It selects the Structures perspective to perform a stress analysis when a shank geometry is proposed by the designer. At this time, very little domain-specific design management knowledge has been developed owing to our reliance on the designer for guidance.

Constraint manager

The Constraint Management System (CMS) in Design Fusion provides a facility for manipulating and checking constraints posed by the various perspectives (Rinderle and Krishnan, 1990; Navin chandra *et al.*, 1992). The CMS treats the given constraints as a graph of constraints and parameters which can be manipulated and used in the following ways:

1. *Posting*. Perspectives are able to send constraints to the system's constraint graph at any time during a product's development cycle. Whenever a constraint is received, it is automatically linked into the constraint graph, and hence becomes part of the design's environment.
2. *Consistency*. When a perspective makes a change to a feature, it may request the CMS for a consistency check.
3. *Checking and evaluation*. Constraints and algebraic expressions may be evaluated at any time and values can be propagated from one part of the design to another.
4. *Coordination*. Constraints and parameters belonging to different perspectives are logically partitioned in the constraint graph. This helps in relating the design decisions made by one discipline to the goals and constraints of another discipline, thus providing a coordination function.

The constraints are maintained in a dynamic constraint graph. The constraint graph is represented as a tripartite graph of constraints, parameters and entities. Each constraint points to the parameters it *constrains* and each parameter points to the constraints that it is *constrained-by* (Figure 6). It is possible to post constraints to the graph from a variety of different perspectives. For example, one might load the design constraints together with the manufacturing constraints to find possible conflicts. Such conflicts are found well before values are selected for all the variables in the design. It is for this reason that the constraint graph representation is useful in coordinating multiple perspectives. In addition, it makes it possible to provide early feedback about impending problems that occur due to incorrect decisions made within a perspective, or problems that occur due to interactions across perspectives. For example, while a designer is working on the intricacies of the electrical connections in a large power transformer, the designer might be alerted to a problem with maximum allowable size for transportation.

As the CMS uses a single, uniform representation for all constraints, there is no differentiation between functional, geometric, manufacturing and other so-called life-cycle constraints. Methods used to refine the design by processing constraints are applied uniformly to all the

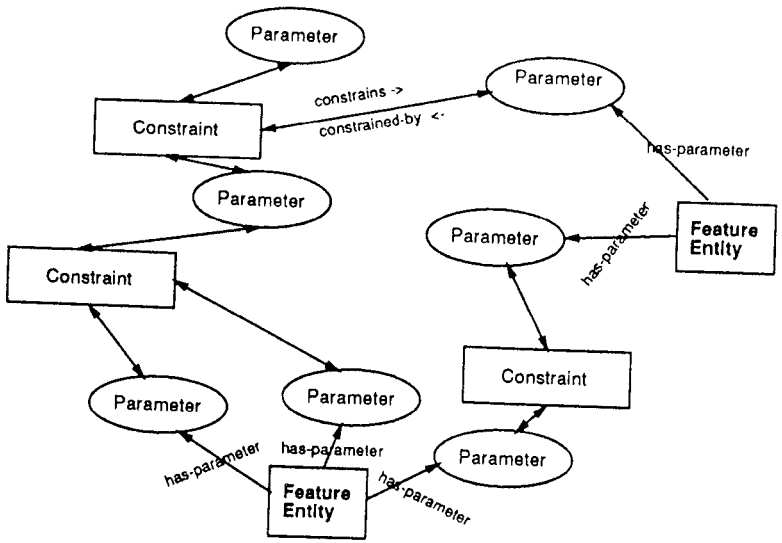

Figure 6. Bipartite graph of variables and constraints.

constraints, regardless of their origin. The computer does not distinguish between constraints that are functional and those that have traditionally been considered downstream of the design phase. All constraints, upstream or downstream are considered equal. At any stage in the design process, the computer will focus its attention on the constraints that have been violated, regardless of their origins. It is for this reason that our approach achieves concurrency.

Feature extraction

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 perspectives. Features provide both an abstraction mechanism and a mechanism for communicating among perspectives 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 Pinilla *et al.* (1989), Safier and Fox (1989a), Finger and Safier (1990) and Safier and Finger (1990).

Design database

In many design domains, there is a lot of similarity between artefacts. It would be convenient to have a way of generating a new design by modifying a previous but similar design. Design Fusion supports this methodology through the concept of *prior designs*. It is currently possible for the designer to load a prior design and use that as a basis for a new design.

Ongoing work involves developing a sophisticated design librarian to keep track of prior designs and be able to search for particular attributes. This would allow finding the prior design that most closely resembles the current design goals.

DESIGN MANAGEMENT

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 artefact 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 behaviour. 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 to then 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 not only obfuscates the problem but increases the problem-solving complexity to the point of being unmanageable. An 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 used 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 stage 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. Owing 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, multilevel 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 7 defines the lower level protocol. Work on the upper level protocol is underway.

- 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 7. Low-level protocol definition.

DETAILED EXAMPLE

In this section, we provide an annotated trace of the Design Fusion system as it has been used to design a fan blade. Each cycle of the system is divided into four steps: checking monitors, prioritize perspectives, invoke perspective, and checking constraints. At the end of each cycle the annotation appears in italics.

0001 Design Manager

Checking Monitors:
No monitors activated

Prioritize Perspectives:
Designer (default-to-designer-control): 1.

Invoke perspective Designer: executing default-to-designer-control activity.
Loading a perspective: manufacturing.
Manufacturing posting pending proxy: (BLADE-LENGTH ≤ 0.2).

Checking Constraints:
No inconsistencies found.

There were no other monitors activated so the designer perspective was invoked by calling the default-to-designer-control activity. The designer chose to load the manufacturing perspective. A proxy for the manufacturing perspective was received at that time and the pending proxy was posted to the blackboard. A proxy is a constraint created by a person or system external to Design Fusion to be used during the design process.

0002 Design Manager

Checking Monitors:
Firing monitor PENDING-PROXY (designer):
Designer posting comment: Retrieve proxy BLADE-LENGTH-CONSTRAINT.

Prioritize Perspectives:
Designer (default-to-designer-control): 1.

Invoke perspective Designer: executing default-to-designer-control activity.
Loading a perspective: aerodynamics.

Checking Constraints:
No inconsistencies found.

The pending-proxy monitor was activated due to the new pending-proxy on the blackboard. The activation of this monitor resulted in a comment to the designer that the proxy was pending and should be retrieved. Again the default-to-designer-control-activity was executed and the designer chose to load the aerodynamics perspective.

0003 Design Manager

Checking Monitors:
No monitors activated.

Prioritize Perspectives:
Designer (default-to-designer-control): 1.

Invoke perspective Designer: executing default-to-designer-control activity.
Loading a perspective: structures.

Checking Constraints:
No inconsistencies found.

As before, the default-to-designer-control activity was executed and the designer loaded the structures perspective.

0004 Design Manager

Checking Monitors:
No monitors activated.

Prioritize Perspectives:
Designer (default-to-designer-control): 1.

Invoke perspective Designer: executing default-to-designer-control activity.
Loading a prior design: b1001–0725 1989a—pressure: 180, flow rate: 8.0, efficiency: 78.5.
Designer posting constraint: (≥ GOAL-VOLUME-FLOW 8.6).
Designer posting parameter: VOLUME-FLOW = 8.6d0.
Designer posting constraint: (≥ GOAL-PRESSURE-RISE 180).
Designer posting parameter: PRESSURE-RISE = 180.
Designer posting constraint: (= GOAL-RPM 1440).
Designer posting parameter: RPM = 1440.
Designer posting parameter: BLADE-LENGTH = 0.2d0.
Designer posting parameter: DENSITY-MEDIUM = 4428.
Designer posting parameter: BLEND-RADIUS = 0.00254d0.
Designer posting parameter: RADIUS-AT-PLATFORM = 0.25.
Designer posting parameter: RADIUS-AT-TIP = 0.5.
Designer posting parameter: NUMBER-OF-BLADES = 4.
Designer posting parameter: X-TILT = 0.0.
Designer posting parameter: Y-TILT = 0.0.
Designer posting parameter: AERO-BLADE = 0.
Designer posting parameter: AERO-EFFICIENCY = 0.785.

Checking Constraints:
No inconsistencies found.

The default-to-designer-control activity was executed and the designer requested that a prior design be loaded. The design was loaded onto the panels from the design database. The loading caused the posting of a number of parameters specified by this prior design.

0005 Design Manager

Checking Monitors:
Firing monitor BLEND-RADIUS-CHECK (aerodynamics):
Aerodynamics posting activity: Reveal a constraint on blend-radius.

Prioritize Perspectives:
Aerodynamics (reveal-blend-radius-constraint): 0.

Designer (default-to-designer-control): 1.

Invoke perspective Aerodynamics: executing reveal-blend-radius-constraint activity.
Aerodynamics retracting activity: Reveal a constraint on blend-radius.
Aerodynamics posting comment: Aerodynamics: Blend Radius should be less than 0.0015 m.
Aerodynamics posting constraint: (BLEND-RADIUS ≤ 0.0015).

Checking Constraints:
(≤ blend-radius 0.0015): inconsistent.

The monitor blend-radius-check was activated and as a result an activity to **reveal** *that a constraint on blend radius was posted to the blackboard. The activity reveal-blend-radius-constraint was executed causing a constraint on blend radius to be posted to the blackboard along with a comment to alert the designer to the new constraint. The new constraint on blend-radius was flagged by the constraint manager as being inconsistent with the current value of blend-radius.*

0006 Design Manager

Checking Monitors:
Firing monitor INCONSISTENT-CONSTRAINTS (designer):
Designer posting comment: Handle inconsistent constraints.|
Designer posting activity: Postpone any inconsistent constraints.

Prioritize Perspectives:
Designer (default-to-designer-control): 1.
Designer (postpone-inconsistent-constraints): 2.

Invoke perspective Designer: executing postpone-inconsistent-constraints activity.
Designer retracting activity: Postpone any inconsistent constraints.

Checking Constraints:
No inconsistencies found.

The monitor inconsistent-constraints was activated as a result of blend-radius-constraint being inconsistent. As a result, the activity postpone-inconsistent-constraints and a related comment to the designer were posted to the blackboard. The designer chose to execute this activity which marks the previously inconsistent constraints as postponed. The constraint checking mechanism ignores constraints that are tagged as postponed and thus reported "No inconsistencies found".

0007 Design Manager

Checking Monitors:
No monitors activated.

Prioritize Perspectives:
Designer (default-to-designer-control): 1.

Invoke perspective Designer: executing default-to-designer-control activity.
Setting a Constraint: (≥ goal-volume-flow 10.0).
Designer revising constraint: (≥ GOAL-VOLUME-FLOW 8.6) → (≥ GOAL-VOLUME-FLOW 10.0).
Designer revising parameter: VOLUME-FLOW = 8.6d0 → VOLUME-FLOW = 10.0.
RMS retracting parameter: AERO-EFFICIENCY = 0.785.
RMS retracting parameter: AERO-BLADE = 0.
RMS retracting parameter: RADIUS-AT-TIP = 0.5.
RMS retracting parameter: RADIUS-AT-PLATFORM = 0.25.

Checking Constraints:
No inconsistencies found.

The default-to-designer-control activity was executed and the designer chose to revise the specification, goal-volume-flow. The revision of volume-flow causes the RMS to retract the entities aero-efficiency, aero-blade, radius-at-tip, and radius-at-platform, all of which were originally derived from volume-flow.

0008 Design Manager

Checking Monitors:
Firing monitor SOLVE-FOR-BLADE-SECTIONS (aerodynamics):
Aerodynamics posting activity: Generate an optimal turbine blade from specifications.
Aerodynamics posting comment: Aerodynamics: Specifications have changed. Solve for new aerofoil.|

Prioritize Perspectives:
Designer (default-to-designer-control): 1.
Aerodynamics (solve-for-new-air-foil): 10.

Invoke perspective Aerodynamics: executing solve-for-new-air-foil activity.
Aerodynamics retracting activity: Generate an optimal turbine blade from specifications.
Aerodynamics retracting comment: Aerodynamics: Specifications have changed. Solve for new aerofoil.|
Aerodynamics revising parameter: BLEND-RADIUS = 0.00254d0 → BLEND-RADIUS = 0.0025d0.
Aerodynamics revising parameter: BLADE-LENGTH = 0.2d0 → BLADE-LENGTH = 0.25d0.
Aerodynamics posting parameter: AERO-BLADE = 2.
Aerodynamics posting parameter: RADIUS-AT-PLATFORM = 0.25d0.

Aerodynamics posting parameter: RADIUS-AT-TIP = 0.5d0.
Aerodynamics posting parameter: AERO-EFFICIENCY = 77.4d0.

Checking Constraints:
No inconsistencies found.

The monitor solve-for-blade-sections was activated due to the retraction of aero-blade in the previous cycle. This monitor posts the activity solve-for-new-air-foil and a related comment to the blackboard. The designer chose to execute the solve-for-new-air-foil activity. The activity and comment were retracted from the blackboard and a new blade was generated from the current parameters and specifications resulting in revision of the parameters blend-radius and blade-length and the posting of the parameters aero-blade, radius-at-platform, radius-at-tip and aero-efficiency.

0009 Design Manager

Checking Monitors:
No monitors activated.

Prioritize Perspectives
Designer (default-to-designer-control): 1.

Invoke perspective Designer: executing default-to-designer-control activity.
Retrieving a proxy: blade-length-constraint.
Designer posting constraint: (BLADE-LENGTH ≤ 0.2).

Checking Constraints:
(≤ blade-length 0.2): inconsistent.

The designer chose to retrieve the pending proxy blade-length constraint, resulting in the blackboard posting of a constraint on blade-length. The new constraint on blade-length was found to be inconsistent by the constraint checker.

0010 Design Manager

Checking Monitors:
Firing monitor INCONSISTENT-CONSTRAINTS (designer):
Designer posting comment: Handle inconsistent constraints.|
Designer posting activity: Postpone any inconsistent constraints.
Firing monitor CHECK-BLADE-LENGTH (aerodynamics):
Aerodynamics posting activity: Generate an optimal turbine blade from specifications.
Aerodynamics posting comment: Aerodynamics: Constraint violation.| Blade length = 0.25d0 should be less than or equal to 0.2.| Solve for new airfoil.|

Prioritize Perspectives:
Designer (default-to-designer-control): 1.

Designer (postpone-inconsistent-constraints): 2.
Aerodynamics (solve-for-new-air-foil): 10.

Invoke perspective Aerodynamics: executing solve-for-new-air-foil activity.
Aerodynamics retracting activity: Generate an optimal turbine blade from specifications.
Aerodynamics retracting comment: Aerodynamics: Constraint violation.| Blade length = 0.25d0 should be less than or equal to 0.2.| Solve for new airfoil.|
Aerodynamics revising parameter: AERO-BLADE = 2 → AERO-BLADE = 3.
Aerodynamics revising parameter: BLEND-RADIUS = 0.0025d0 → BLEND-RADIUS = 0.0025d0.
Aerodynamics revising parameter: BLADE-LENGTH = 0.25d0 → BLADE-LENGTH = 0.2d0.
Aerodynamics revising parameter: RADIUS-AT-PLATFORM = 0.25d0 → RADIUS-AT-PLATFORM = 0.3d0.
Aerodynamics revising parameter: RADIUS-AT-TIP = 0.5d0 → RADIUS-AT-TIP = 0.5d0.
Aerodynamics revising parameter: AERO-EFFICIENCY = 77.4d0 → AERO-EFFICIENCY = 75.5d0.

Checking Constraints:
No inconsistencies found.

Two monitors, inconsistent-constraints and check-blade-length, were both activated because of the constraint on blade-length that arrived via proxy in cycle 9. They posted activities and comments for postponing the inconsistent constraints and solving for a new airfoil, respectively. The designer choise to execute the solve-for-new-air-foil activity. As in frame 8, the activity and comment were retracted from the blackboard and a new blade was generated from the current parameters and specifications, resulting in revision of the parameters aero-blade, blend-radius, blade-length, radius-at-platform, radius-at-tip and aero-efficiency. Note that the inconsistent contraint from cycle 9 was resolved by this activity.

0011 Design Manager

Checking Monitors:
No monitors activated.

Prioritize Perspectives:
Designer (default-to-designer-control): 1.
Designer (postpone-inconsistent-constraints): 2.

Invoke perspective Designer: executing default-to-designer-control activity.
Setting a Constraint: (≥ goal-life 100 000).
Designer posting constraint: (≥ GOAL-LIFE 100 000).

Designer posting parameter: MINIMUM-LIFE = 100 000.

Checking Constraints:

No inconsistencies found.

The activity default-to-designer-control was executed and the designer chose to initialize the specification for goal-life. As with the revision of goal-volume-flow and volume-flow in frame 7, the parameter minimum-life is revised to satisfy minimally the goal-life specification.

0012 Design Manager

Checking Monitors:

Firing monitor ASSERT-LIFEMAP-ACTIVITIES (structures):

Structures posting activity: Approximate minimum life.

Structures posting activity: Generate life map.

Firing monitor SATISFY-GOAL-LIFE? (structures):

Structures posting comment: Structures: This design meets the life goal of 100 000.0.| The current life is 100 000.0.

Prioritize Perspectives:

Designer (default-to-designer-control): 1.

Designer (postpone-inconsistent-constraints): 2.

Structures (approx-life): 2.

Structures (compute-life-map): 8.

Invoke perspective Structures: executing approx-life activity.

Structures revising parameter: MINIMUM-LIFE = 100 000 → MINIMUM-LIFE = 75 788.

Structures revising constraint: (BLADE-LENGTH ≤ 0.2) → (BLADE-LENGTH ≤ 0.174 114d0).

Structures posting parameter: OPTIMAL-X-TILT = −0.007 778d0.

Structures posting parameter: OPTIMAL-Y-TILT = −0.020 786d0.

Structures posting parameter: OPTIMAL-BLEND-RADIUS = 0.004 613d0.

Structures posting parameter: OPTIMAL-LENGTH = 0.174 114d0.

Checking Constraints:

(≤ blade-length 0.174 114d0): inconsistent.

The monitor assert-lifemap-activities was activated owing to the instantiation of the parameter minimum life in the previous frame. Assert-lifemap-activities posted activities for approximating the expected minimum life and generating a life map of the current airfoil. The designer chose to execute the approx-life activity. The parameter, minimum-life, and the constraint on blade-length were revised and the parameters optimal-x-tilt, optimal-y-tilt, optimal-blend-radius, and optimal-length were posted to the blackboard. The constraint checker signalled that the new blade-length constraint was inconsistent.

0013 Design Manager

Checking Monitors:

Firing monitor SATISFY-GOAL-LIFE? (structures):

Structures posting comment: Structures: This design does not meet the current goal of 100 000.0 cycles.| The current life is 75 788.0. The following changes would improve fatigue life.| Change Tilt Angle: The current tilt angle is (x = 0.0, y = 0.0).| The optimal tilt is (x = −0.007 778d0, y = −0.020 786d0).| Blend Radius: Change the blend radius from 0.0025d0 to 0.004 613d0.| Length: Reduce the length of the airfoil to approximately 0.174 114d0.|

Firing monitor CHECK-BLADE-LENGTH (aerodynamics):

Aerodynamics posting activity: Generate an optimal turbine blade from specifications.

Aerodynamics posting comment: Aerodynamics: Constraint violation.| Blade length = 0.2d0 should be less than or equal to 0.174 114d0.| Solve for new airfoil.|

Prioritize Perspectives:

Designer (default-to-designer-control): 1.

Designer (postpone-inconsistent-constraints): 2.

Structures (approx-life): 2.

Structures (compute-life-map): 8.

Aerodynamics (solve-for-new-air-foil): 10.

Invoke perspective Aerodynamics: executing solve-for-new-air-foil activity.

Aerodynamics retracting activity: Generate an optimal turbine blade from specifications.

Aerodynamics retracting comment: Aerodynamics: Constraint violation.| Blade length = 0.2d0 should be less than or equal to 0.174 114d0.| Solve for new airfoil |

Aerodynamics revising parameter: AERO-BLADE = 3 → AERO-BLADE = 4.

Aerodynamics revising parameter: BLEND-RADIUS = 0.0025d0 → BLEND-RADIUS = 0.0025d0.

Aerodynamics revising parameter: BLADE-LENGTH = 0.2d0 → BLADE-LENGTH = 0.15d0.

Aerodynamics revising parameter: RADIUS-AT-PLATFORM = 0.3d0 → RADIUS-AT-PLATFORM = 0.35d0.

Aerodynamics revising parameter: RADIUS-AT-TIP = 0.5d0 → RADIUS-AT-TIP = 0.5d0.

Aerodynamics revising parameter: AERO-EFFICIENCY = 75.5d0 → AERO-EFFICIENCY = 71.5d0.

RMS retracting comment: Structures: This design does not meet the current goal of 100 000.0 cycles.| The current lift is 75 788.0. The following changes would improve fatigue life.| Change Tilt Angle:

The current tilt angle is (x = 0.0, y = 0.0).| The optimal tilt is (x = −0.007 778d0, y = −0.020 786d0) Blend Radius: Change the blend radius from 0.0025d0 to 0.004 613d0 Length: Reduce the length of the airfoil to approximately 0.174 114d0

RMS retracting parameter OPTIMAL-LENGTH = 0.174 114d0.
RMS retracting parameter OPTIMAL-BLEND-RADIUS = 0.004 613d0
RMS retracting parameter OPTIMAL-Y-TILT = −0.020 786d0.
RMS retracting parameter OPTIMAL-X-TILT = −0.007 778d0.
RMS retracting constraint (BLADE-LENGTH ≤ 0.174 114d0).
RMS retracting parameter: MINIMUM-LIFE = 75 788.
RMS retracting comment: Structures: This design meets the life goal of 100 000.0.| The current life is 100 000.0.
RMS retracting activity: Generate life map.
RMS retracting activity: Approximate minimum life.

Checking Constraints:
No inconsistencies found.

The monitors satisfy-goal-life? and check-blade-length were activated. The firing of satisfy-goal-life? resulted in a comment to the designer that the current design does not meet the specification for goal-life. The firing of check-blade-length resulted in the posting of an activity and comment to generate a new air foil. The designer chose the solve-for-new-air-foil activity. The activity and comment were retracted from the blackboard and a new blade was generated from the current parameters and specifications, once again resulting in revision of the parameters aero-blade, blend-radius, blade-length, radius-at-platform, radius-at-tip and aero-efficiency. In addition, the RMS retracted the comment posted by the satisfy-goal-life? monitor, the parameters optimal-length, optimal-blend-radius, optimal-y-tilt, optimal-x-tilt, and minimum-life, the constraint on blade-length and the activities for generating a life map and approximating the minimum life. Once again the inconsistent constraint from cycle 12 was resolved by this activity and no inconsistent constraints were found.

0014 Design Manager

Checking Monitors:
Firing monitor ASSERT-LIFEMAP-ACTIVITIES (structures):
Structures posting activity: Approximate minimum life.
Structures posting activity: Generate life map.

Prioritize Perspectives:
Designer (default-to-designer-control): 1.
Designer (postpone-inconsistent-constraints): 2.

Structures (approx-life): 2.
Structures (compute-life-map): 8.

Invoke perspective Structures: executing approx-life activity.
Structures posting parameter: MINIMUM-LIFE = 87 748.
Structures posting parameter: OPTIMAL-X-TILT = −0.00867d0.
Structures posting parameter: OPTIMAL-Y-TILT = −0.019 821d0.
Structures posting parameter: OPTIMAL-BLEND-RADIUS = 0.003 875d0.
Structures posting parameter: OPTIMAL-LENGTH = 0.140 512d0.
Structures posting constraint: (BLADE-LENGTH ≤ 0.140 512d0).

Checking Constraints:
(≤ blade-length 0.140 512d0): inconsistent.

The monitor assert-lifemap-activities was activated, resulting in the posting of activities to approximate minimum life and generate a life map. The designer chose to approximate the minimum life. As a result the parameters minimum-life, optimal-x-tilt, optimal-y-tilt, optimal-blend-radius and optimal-length, and the constraint blade-length were posted to the blackboard. The new constraint on blade-length was flagged as inconsistent by the constraint checker.

0015 Design Manager

Checking Monitors:
Firing monitor SATISFY-GOAL-LIFE? (structures):
Structures posting comment: Structures: This design does not meet the current goal of 100 000.0 cycles.| The current life is 87 748.0. The following changes would improve fatigue life.| Change Tilt Angle: The current tilt angle is (x = 0.0, y = 0.0).| The optimal tilt is (x = −0.008 67d0, y = −0.019 821d0).| Blend Radius: Change the blend radius from 0.0025d0 to 0.003 875d0.| Length: Reduce the length of the airfoil to approximately 0.140 512d0.|
Firing monitor CHECK-BLADE-LENGTH (aerodynamics):
Aerodynamics posting activity: Generate an optimal turbine blade from specifications.
Aerodynamics posting comment: Aerodynamics: Constraint violation.| Blade length = 0.15d0 should be less than or equal to 0.140 512d0.| Solve for new airfoil.|
Aerodynamics posting feature: MODEL-NOT-APPLICABLE.
Aerodynamics posting comment: Aerodynamics Warning: aerodynamic model not applicable for blade lengths less than or equal to 0.149 metre.
Firing monitor AERO-NOT-APPLICABLE (structures):
Structures posting activity: Refine structures blade-length constraint.

Prioritize Perspectives:
Designer (default-to-designer-control): 1.
Designer (postpone-inconsistent-constraints): 2.
Structures (approx-life): 2.
Structures (refine-constraint): 2.
Structures (compute-life-map): 8.
Aerodynamics (solve-for-new-air-foil): 10.

Invoke perspective Structures: executing refine-constraint activity.
Structures retracting activity: Refine structures blade-length constraint.
Structures revising constraint: (BLADE-LENGTH ≤ 0.140 512d0) → (BLADE-LENGTH * RADIUS-AT-PLATFORM + (BLADE-LENGTH / 2)) * (1 + ((RADIUS-AT-TIP / RADIUS-AT-PLATFORM) ^ 2))) ≤ 0.097).
Tms retracting comment: Aerodynamics Warning: aerodynamic model not applicable for blade|~~14Tlengths less than or equal to 0.149 metre.|
Tms retracting feature: MODEL-NOT-APPLICABLE.

Checking Constraints:
No inconsistencies found.

The monitor satisfy-goal-life? was activated because the new value for minimum-life posted in the previous cycle did not meet the goal-life specification. A comment was posted to alert the designer to this situation. The check-blade-length monitor was activated because of the inconsistency of the constraint on blade-length. As a result, the solve-for-new-air-foil activity and its related comment. Aerodynamics posted a model-not-applicable feature and its explanatory comment because its internal model does not apply once blade lengths are below 0.149 metres. And finally, the aero-not-applicable monitor was activated owing to the newly posted model-not-applicable feature. This resulted in the refine-constraint activity being posted to the blackboard. The refinement replaced the simple blade length inequality with a more detailed constraint so that the other perspectives could alter their decisions in light of what is constraining structures; if aerodynamics can compute a solution that satisfies the more detailed constraint, then it would be acceptable to structures. The designer chose to execute the refine-constraint activity which refined the constraint on blade-length.

0016 Design Manager

Checking Monitors:
No monitors activated.

Prioritize Perspectives:
Designer (default-to-designer-control): 1.

Designer (postpone-inconsistent-constraints): 2.
Structures (approx-life): 2.
Structures (compute-life-map): 8.
Aerodynamics (solve-for-new-air-foil): 10.

Invoke perspective Aerodynamics: executing solve-for-new-air-foil activity.
Aerodynamics retracting activity: Generate an optimal turbine blade from specifications.
Aerodynamics retracting comment: Aerodynamics: Constraint violation.| Blade length = 0.15d0 should be less than or equal to 0.140 512d0.| Solve for new airfoil.|
Aerodynamics revising parameter: AERO-BLADE = 4 → AERO-BLADE = 5.
Aerodynamics revising parameter: BLEND-RADIUS = 0.0025d0 → BLEND-RADIUS = 0.0025d0.
Aerodynamics revising parameter: BLADE-LENGTH = 0.15d0 → BLADE-LENGTH = 0.2d0.
Aerodynamics revising parameter: RADIUS-AT-PLATFORM = 0.35d0 → RADIUS-AT-PLATFORM = 0.25d0.
Aerodynamics revising parameter: RADIUS-AT-TIP = 0.5d0 → RADIUS-AT-TIP = 0.45d0.
Aerodynamics revising parameter: AERO-EFFICIENCY = 71.5d0 → AERO-EFFICIENCY = 74.5d0.
RMS retracting comment: Structures: This design does not meet the current goal of 100 000.0 cycles.| The current life is 87 748.0. The following changes would improve fatigue life.| Change Tilt Angle: The current tilt angle is (x = 0.0, y = 0.0).| The optimal tilt is (x = −0.008 67d0, y = −0.019 821d0).| Blend Radius: Change the blend radius from 0.0025d0 to 0.003 875d0.| Length: Reduce the length of the airfoil to approximately 0.140 512d0.
RMS retracting parameter: OPTIMAL-LENGTH = 0.140 512d0.
RMS retracting parameter: OPTIMAL-BLEND-RADIUS = 0.003 875d0.
RMS retracting parameter: OPTIMAL-Y-TILT = −0.019 821d0.
RMS retracting parameter: OPTIMAL-X-TILT = −0.008 67d0.
RMS retracting parameter: MINIMUM-LIFE = 87 748.
RMS retracting activity: Generate life map.
RMS retracting activity: Approximate minimum life.

Checking Constraints:
No inconsistencies found.

The designer chose the solve-for-new-air-foil activity causing the parameters aero-blade, blend-radius, blade-length, radius-at-platform, radius-at-tip and aero-efficiency to be revised. These values were selected to satisfy structure's refined constraint. In addition, the RMS retracted the para-

meters optimal-length, optimal-blend-radius, optimal-y-tilt, optimal-x-tilt, and minimum-life

0017 Design Manager

Checking Monitors:
Firing monitor ASSERT-LIFEMAP-ACTIVITIES (structures):
Structures posting activity: Approximate minimum life.
Structures posting activity: Generate life map.

Prioritize Perspectives:
Designer (default-to-designer-control): 1.
Designer (postpone-inconsistent-constraints): 2.
Structures (approx-life): 2.
Structures (compute-life-map): 8.

Invoke perspective Structures: executing compute-life-map activity.
Structures posting parameter: MINIMUM-LIFE = 75 788.
Structures posting parameter: OPTIMAL-X-TILT = 0.007 778d0.
Structures posting parameter: OPTIMAL-Y-TILT = −0.020 786d0.
Structures posting parameter: OPTIMAL-BLEND-RADIUS = 0.004 613d0.
Structures posting parameter: OPTIMAL-LENGTH = 0.174 114.

Checking Constraints:
No inconsistencies found.

The assert-lifemap-activities was activated because of the new aero-blade posted in the previous frame. As before, this caused activities for approximating the minimum life and generating a life map to be posted to the blackboard. The designer chose to generate a lifemap which resulted in the posting of the parameters minimum-life, optimal-x-tilt, optimal-y-tilt, optimal-blend-radius and optimal-length.

The design of the blade is complete, though the minimum-life goals were not achieved.

CONCLUSION

The Design Fusion system is an example of an AI approach to computer-assisted design. At its core, Design Fusion provides a multilevel shared representation of the design artefact: geometry, features and constraints, and design control. The shared representation provides a means by which various life-cycle perspectives and methods can contribute, along with the user, to the evolving design.

Design Fusion supports an opportunistic approach to design. The designer and perspectives may jump from one feature to another in the design space, from abstract to detailed, from one component to another. In order to manage the design process, Design Fusion tracks all features, their dependencies and constraints, and uses sophisticated constraint-solving and reasoning-maintenance techniques to propagate and retract decisions, respectively.

The Design Fusion system has been used to design fan blades and a robot arm and is currently being applied to power transformer design.

ACKNOWLEDGEMENTS

Portions of this chapter are excerpts from Finger (1991) and Safier and Fox (1989b). This work has been sponsored by the Defense Advanced Research Projects Agency (DARPA), under contract number MDA972-88-C-0047 for DARPA Initiative in Concurrent Engineering (DICE).

REFERENCES

Akin, O. (1986). *Psychology of Architectural Design*. London: Pion Limited.

Alexander, C. (1965). *Notes on the Synthesis of Form*. Cambridge, Mass.: Harvard University Press.

Alexander, C. (1968). A city is not a tree. *Ekistics*, **139**, 344–348.

Bachant, J. and McDermott, J. (1984). R1 revisited: four years in the trenches. *AI Magazine*, 5(3), 21–32.

Boothroyd, G. and Dewhurst, P. (1983). Design for assembly—a designers handbook. Technical Report, Department of Mechanical Engineering, University of Massachusetts, Amherst, Mass.

Dixon, J. R., Cunningham, J. J. and Simmons, M. K. (1987). Research in designing with features. In D. Gossard (ed.) *IFIP WG 5.2 Workshop on Intelligent CAD Systems, IFIP, Cambridge, Mass.*

Erman, L. D., Hayes-Roth, F., Lesser, V. R. and Reddy, D. R. (1980). The Hearsay-II speech understanding system: integrating knowledge to resolve uncertainty. *ACM Computing Surveys*, **12**(2), 213–253.

Finger, S. and Safier, S. A. (1990). Representing and recognizing features in mechanical designs. In *Design Theory and Methodology—DTM '90*. Chicago: ASME, pp. 19–25.

Finger, S., Fox, M. S., Navinchandra, D., Prinz, F. B. and Rinderle, J. R. (1988). The Design Fusion project: a product life-cycle view for engineering design. In H. Yoshikawa and T. Holden (eds) *IFIP WG 5.2 Workshop on Intelligent CAD*, pp. 165–172.

Finger, S., Fox, M. S., Prinz, F. B. and Rinderle, J. R. (1991). Concurrent design. *Applied Artificial Intelligence*, 1991, in press.

Fox, M. S. (1983). Constraint-directed search: a case study of job-shop scheduling. Ph.D. thesis, Carnegie Mellon University, 1983. (CMU-RI-TR-85-7, Intelligent Systems Laboratory, The Robotics Institute, Pittsburgh.)

Gero, J. S. and Coyne, R. D. (1987). Knowledge-based planning as a design paradigm. In H. Yoshikawa and E. A. Warman (eds) *Design Theory in Computer-Aided Design*. New York: Elsevier, pp. 289–323.

Grosz, B. J. (1986). The representation and use of focus in a system for understanding dialogs. In B. J. Groz, K. Sparck Jones and B. L. Webber (eds) *Readings in Natural Language Processing*. Palo Alto: Morgan Kaufmann, pp. 353–362.

Gursoz, E. L., Choi, Y. and Prinz, F. (1990). Vertex-based representation of non-manifold boundaries. In M. J. Wozny, J. U. Turner and K. Preiss (eds) *Geometric Modeling for Product Engineering*. New York: North-Holland, pp. 107–130.

Hayes-Roth, F. and Lesser, V. R. (1977). Focus of attention in a distributed logic speech understanding system. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 27–35.

Hulthage, I., Fox, M. S., Rychener, M. D. and Farinacci, M. L. (1990). The architecture of ALADIN: a knowledge-based approach to alloy design. *IEEE Expert*, **5**(4), 56–73.

Katz, R. H. (1985). *Information Management for Engineering Design*. New York: Springer-Verlag.

Mittal, S. M., Dym, C. L. and Morjaria, M. (1985). PRIDE: An expert system for the design of paper handling systems. *IEEE Computer*, November, 102–114.

Mostow, J. (1985). Toward better models of the design process. *AI Magazine*, **6**(1), 44–57.

Navin chandra, D. (1987). Exploring the innovative designs by relaxing criteria and reasoning from precedent-based knowledge. Ph.D. thesis, MIT, Cambridge, Mass.

Navin chandra, D. (1988). Case-based reasoning in CYCLOPS, a design problem solver. In J. Kolodner (ed.) *Proceedings of the DARPA Workshop on Case-based Reasoning*. Palo Alto: Morgan Kaufmann, pp. 286–301.

Navin chandra, D., Fox, M. S. and Gardner, E. (1992). On the role of constraints in concurrent design. *Journal of Institute for Industrial Engineering*, 1992, submitted.

Nielsen, E. H., Dixon, J. R. and Simmons, M. K. (1987). How shall we represent the geometry of designed objects? Technical Report 6–87, Mechanical Design Automation Laboratory, University of Massachusetts, Amherst, Mass.

Nii, Penny (1986). The blackboard model of problem solving. *AI Magazine*, **7**(2), 38–53.

Nii, Penny (1986). Blackboard systems. Part Two: blackboard application systems. *AI Magazine*, **7**(3), 82–106.

Pinilla, J. M., Finger, S. and Prinz, F. B. (1989). Shape feature description and recognition using an augmented topology graph grammar. In *NSF Engineering Design Research Conference, University of Massachusetts, Amherst*, 285–300.

Requicha, A. A. G. and Voelcker, H. B. (1982). Solid modelling: a historical summary and contemporary assessment. *IEEE Computer Graphics and Applications*, **2**(2), 9–23.

Rinderle, J. R. and Krishnan, V. (1990). Constraint reasoning in concurrent design. In *Design Theory and Methodology—DTM '90*. Chicago, ASME, pp. 53–62.

Rinderle, J. R. and Watton, J. D. (1987). Automatic identification of critical design relationships. In *Proceedings of International Conference on Engineering Design ICED 87*.

Safier, S. A. and Finger, S. (1990). Parsing features in geometric models. Technical Report, Robotics Institute, Carnegie-Mellon University, Pittsburgh.

Safier, S. A. and Finger, S. (1989). Parsing features in geometric models; an abstract. In *SIAM Conference on Geometric Design. Tempe. Ariz.*

Safier, S. A. and Fox, M. S. (1989). The role of architecture in computer-assisted design systems. In *NSF Engineering Design Research Conference, University of Massachusetts, Amherst, Mass.*, pp. 507–520.

Simon, H. A. (1968). The architecture of complexity. In *Sciences of the Artificial*. Cambridge, Mass.: MIT Press, pp. 192–229.

Sycara, K. (1990). Cooperative negotiation in concurrent engineering design. In D. Sriram (ed.) *Cooperative Product Development*. New York: Springer Verlag.

Sycara, K. P. and Navin chandra, D. (1989). Integrating case-based reasoning and qualitative reasoning in engineering design. In J. Gero (ed.) *Artificial Intelligence in Design*. New York: Springer-Verlag.

Ullman, D. G., Stauffer, L. A. and Dietterich, T. G. (1987). Preliminary results of an experimental study of the mechanical design process. In M. B. Waldron (ed.) *Proceedings from the NSF Workshop on the Design Process, Ohio State University, Oakland, Calif.*, pp. 145–188.

Voelcker, H. B. (1988). Modeling in the design process. In W. D. Compton (ed.) *Design and Analysis of Integrated Manufacturing Systems*. Washington, DC: National Academy Press, pp. 167–199.

Weiler, K. J. (1986). Topological structures for geometric modeling. Ph.D. thesis, Rensselaer Polytechnic Institute.