The Role of Architecture in Computer-Assisted Design Systems

Scott A. Safier and Mark S. Fox*

January 30, 1990

Abstract

Knowledge-based systems and expert-systems technologies provide software architectures which can assist the mechanical designer. The role of the architecture is to integrate the algorithmic and heuristic processes used during design. Through integration, methods employed by designers can communicate via a common representation of the design.

In this paper, we explore issues of coordination and representation for systems that support mechanical designers. First, we discuss design as a problem-solving activity and the architectural requirements for this type of system. Next, we focus of the components of the architecture, and explore issues of representation and coodination. Lastly, we present the Design Fusion system as an example architecture that utilizes these components to fulfill the requirements.

1 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 of the field of AI, the role of a problem solving architecture is unclear. It is our intent, in this paper, to both motivate the need for architectures in design and demonstrate their application.

The design process is complex; designers bring to bear a variety of methods and techniques throughout. They have many tasks to perform and numerous sources of design data. There are algorithmic solutions to several problems, but each of these solves only part of the 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 known as design.

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

In the following sections, we will discuss architectures to assist designers. First, we present requirements for these architectures. Next, we discuss the techniques used to realize these requirements. Lastly, we discuss the design fusion system[12] as an example of such a system. We will draw our examples from the mechanical design domain, specifically turbine blade design.

*This work has been sponsored by Defense Advanced Research Projects Agency (DARPA), under contract number MDA972-88-C-0047 for DARPA Initiative in Concurrent Engineering (DICE).

2 Requirements

Design systems are created to facilitate the design task and the process used by designers to create an artifact from a specification. The behavior exhibited by designers during this process has been characterized as problem-solving [1, 16]. *To support designers, the architecture of the system should also be a general model of problem-solving.*

The components of a problem-solving system are a database and a a control strategy for manipulating the database. The database contains a 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. In this section, we will examine requirements for coordination and representation in a computer-assisted design system.

2.1 Coordination

Design is a search process that explores the elaboration of portions of a design resulting in a top-down decomposition of 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 artifact or description of an artifact. 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 [2] and Simon [32] assume this model of design – that design is a nearly-decomposable problem with little or no interaction between design subproblems. Alexander [3] later concludes that such problems tend to be artificial – natural problems contain interactions. During problem-solving, designers must cope with interactions between design constraints: the specifications, the goals of the designer, 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.*

2.1.1 Constraint-Directed Problem-Solving

Interactions occur between design constraints that are closely related [25]. Cooperation occurs when there is a dependence between interacting constraints, or one constraint is a generalization of others. Competition arises when one or several constraints 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 artifact 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[14].* For example, Rinderle and Watton [30] 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 with alternate view points of the design, providing insight.

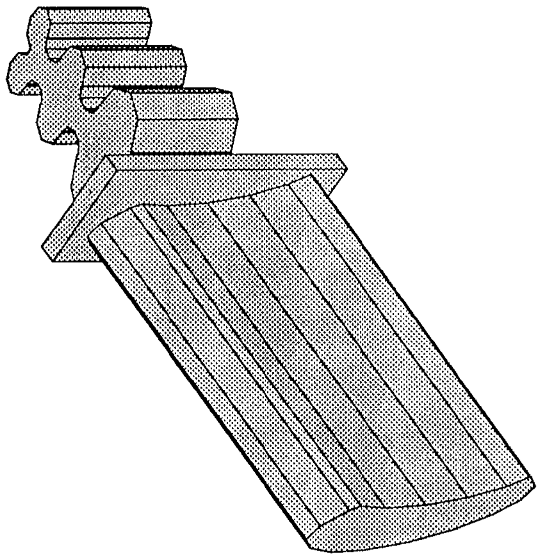


Figure 1: Sample Turbine Blade

2.1.2 Opportunistic Problem-Solving

Opportunistic problem-solving [15] focuses attention on tightly constrained portions of the (design) problem. 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, design decisions which constrain other areas of the design can propagate. These further constrain those areas. 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. [6] 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 andfocusing on the portions of design that are most certain.*

Second, past designs can be opportunistically exploited during design. Finger et al. [12] 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 exisiting part, the design task is selection. A design is routine when similar artifacts 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 artifact. 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.*

Lastly, the architecture should support the opportunisitic 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 actively participate 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.

2.2 Representation

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.* It should be consistent over these multiple view points 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 artifact at various levels of abstraction. Included in a design representation must be:

function Artifacts are designed to meet some desired behavior. A representation of function would permit reasoning about an artifacts behavior and its role in a complex system;

geometry and topology Behaviors 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 [21] defines a configuration as a collection of versions of partial designs. 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.

2.3 Designer Interaction

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 toward 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 [17] 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 behavior.

The architecture should provide mechanisms for each of these requirements.

Once again consider a designer of a turbine blade. A designer focusing on structural properties will see different design features than when the focus is on aerodynamic properties. If the designer is analyzing 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.

2.4 Summary

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.

3 Architectual Components

As a general model of problem-solving, representation and coordination are the components of an architecture. Representation is the encoding of facts and assertions about an artifact's evolving design in a manner that permits answering questions that arise during the design process. Coordination defines when and how information is used by the system. In this section, we will explore how these components can be used to fulfill the requirements described in the previous section.

3.1 Representation

The role of a representation is to allow answers to a set of questions at a level of complexity appropriate to the design task. Rinderle [29] argues that designers' not only reason about the form and function of an artifact, but also its fabrication and other down-stream activities. Any representation of the design must answer questions about not only its form, but also its function and the influences of downstream activities.

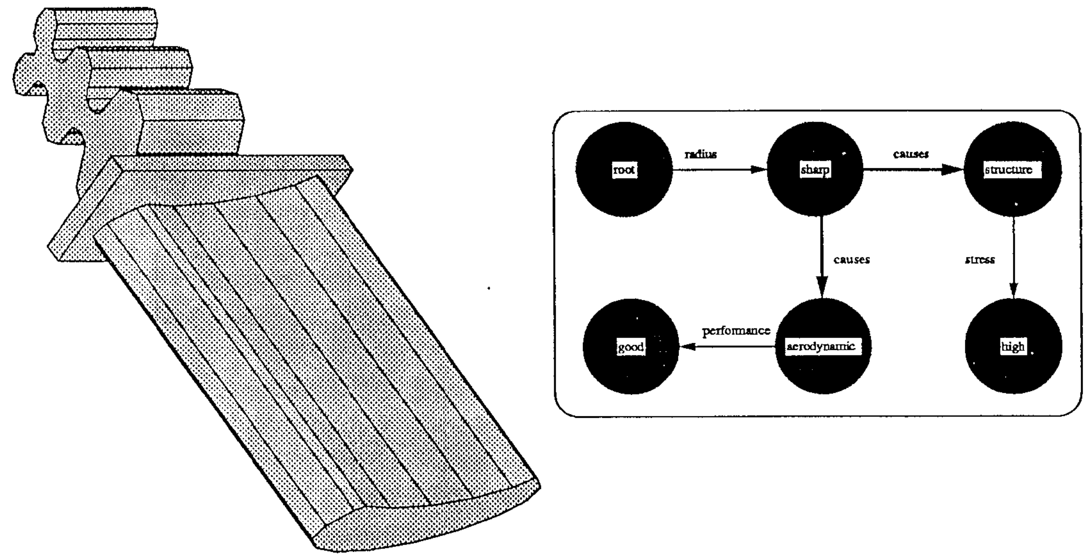
3.1.1 Geometric Modelling

The geometry of the artifact is a neutral representation of the evolving design, and much research has been done into representations for solid models [28]. Wire-frame representations are easy to compute, but are inherently ambiguous because they lack topological information. Representations based upon computation solid geometry provide topological information, but rely on a finite set of primitives to construct solids. Solids not realizable from the provided primitives cannot be constructed. Also, the operations used to combine primitives can be computationally prohibitive.

Two-manifold boundary representations and non-manifold representations [18] also provide topological information. Although boundary representations permit easy computation of free-form solids, they can be memory intensive. Non-manifold representations recognize valid realizable solids, and can also depict non-realizable solids.

3.1.2 Life-cycle Problem Solving

All these representations provide geometric or topological information, but provide no information about the artifact's function and no method of reasoning about the life-cycle effects on the artifact. These representations of solids must be augmented to permit such reasoning. AI research into representation has focused on two primary methods: semantic networks and rule-based systems. Semantic networks are a graph-based scheme for encoding information as relationships between nodes in a graph. Brachman [4] identifies five levels for describing semantic networks. Each level is distinguished by the characteristics of the links between nodes. The *implementation level* characterizes a semantic network simply as a graph. Pointers join nodes, but no specific meaning is assigned to the structure. This level of abstraction provides a dynamic, persistent memory scheme. The *logical level* interprets semantic networks as predicates in a



IF $r_{root} \equiv$ SHARP **THEN**
structural-comment `sharp root radii increase stresses`

IF $r_{root} \equiv$ SHARP **and**
a perspective other than aerodynamics has commented on the root **THEN**
aerodynamic-comment `sharp root radii increase performance`

Figure 2: Augmented turbine blade

predicate calculus. Nodes in this abstraction represent logical relationships such as AND, OR, or SUBSET-OF between atoms or elementary data structures. The *conceptual level* depicts links between conceptual objects as domain-independent relationships, such as spatial, temporal or causal. The *epistemological level* extends the notions of conceptual objects and the properties and structure that define them. At this level, conceptual objects can be related taxonomically to allow inheritance of description from a super-class to a sub-class.

Brachman's fifth level is the *linguistic level.* It is distinct from the other levels because the primitives of the network are language dependent. Also, unlike the other levels, the meaning of these primitives is expected to change over time. Context-free grammars are an example of linguistic level network.

Rule-based systems provide a method of heuristic search [26] based upon a *recognize-act cycle.* Each rule (or *production*) encodes a pattern/action pair that reads and modifies a database environment. When the pattern matches the data in the environment, the associated action is performed. If multiple rules are satisfied simultaneously, a *conflict-resolution strategy* chooses among them for one to execute. The cycle of matching patterns and executing actions usually continues until no more rules can be applied.

Semantic networks and production systems can augment solid models to provide qualitative and quantitative abstractions of the design. For example, consider the turbine blade in Figure 2. From this model, a quantity for the radius of curvature at the root can be derived. This number can be applied to formula to determine stresses in the blade, manufacturability, aerodynamic properties, *etc.* However, these computations can be expensive and may not provide the desired feedback to the designer. A qualitative abstraction of the root radius can be used to generate *common-sense* suggestions equally useful to the designer, but without extensive computation. If the root radius is determined to be `sharp`, the system can inform the designer that `sharp root radii increase stresses` or `sharp root radii increase aerodynamic performance`. While this feedback lacks detail, it provides useful information which the designer can act upon.

There has been much recent research on defining a *qualitative physics* – a qualitative causal calculus for reasoning about physical systems [8, 22, 20]. In these models, the behavior of a physical system is defined by a set of simultaneous equations of functional relations augmented by an asymmetric computational mechanism to model causality. For example, de Kleer and Brown [9] impose a heuristic mechanism based upon the dynamics of the systems to derive causality while Iwasaki and Simon [20] determine causality algebraicly using a model of what variables directly affect other variables. These models have been used to simulate mechanical [23] and molecular-genetic [34] systems, to diagnosis faulty circuits [10], and in the design of alluminum alloys [24].

The latter system, Aladin, is particularly interesting because it demonstrates how qualitative reasoning can augment quantitative models in a design activity. To design a new alloy, qualitative and quantitative reasoning occurs as appropriate for each of the microstructure of the material, the properties of the alloys, the alloy composition, and the thermo-mechanical manufacturing process. The two levels are coupled by first designing a material at a qualitative level, then at the quantitative level. For example, the determination of which alloying elements to add occurs at the qualitative level; the determination of how much of each element to add occurs at the quantitative level.

Qualitative knowledge is equally useful for conceptual design systems. Qualitative simulation can be used to determine the manufacturability of an artifact, its structural properties, and other facets of the artifact of interest to the designer. Detection of problems areas by such simulations can then be used to direct the designer's attention to those areas for more detailed, quantitative analysis.

3.2 Organization

The role of the architecture is to coordinate design activities to minimize the effort of the designer. Design reasoning is not monolithic; there does not exist a single algorithm that given the right inputs, outputs a complete, optimized design. As described earlier, it is a search process that requires the reasoned application of many methods and representations. Any problem solving organization must be able to encapsulate these methods and apply them where appropriate.

3.2.1 Encapsulation

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.

The first step towards organizing 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 artifact could be fabricated. Its input would be the geometry of the artifact, and its output would be confirmation of its manufacturablity or a list of problems with the current design associated with a degree of confidence in this assertion.

3.2.2 Coordination

The computer-assisted design system is required to coordinate these multiple encapsulation's (which we call agents) to produce an artifact from a specification. The competing goals of the designer and the life-cycle perspectives, the interactions between specifications and geometry of the artifact, *etc* provide many sources of complexity in this system. Complexity yields chaos. The architecture constrains this chaos by applying a structure to the system [13]. 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 [5] use a hierarchy to manage complexity in a system to perform routine design. Specialists that design specific components of an artifact are organized in a uniform hierarchy. More specific specialist occur at lower levels in this hierarchy which are invoked from higher, more general levels in the hierarchy.

Heterarchical systems have numerous disjoint agents available for particular tasks. Coordination in this type of system is by negotiation and contract based upon the marginal cost of the task [7, 33, 31]. Entities compete for tasks with all other forms of control eliminated between units. Each entity in the heterarchy pursues its own goals in correspondence to the needs of another.

Heterarchical systems provide a stucture 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. The Design Fusion system described in the next section applies heterarchical coordination to the design problem.

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 coodinate the activities of the agents by exploring and evaluating alternate design processes before committing to an acceptable one [16]. Systems must also schedule the activities of the agents, selecting appropriate agents to evaluate and participate in the design process at appropriate times [19, 5]. The architecture must provide either planning, scheduling or a combination of both to coordinate the activities of design agents.

3.3 Summary

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.

4 Design Fusion

Design Fusion [12] uses a blackboard architecture to provide a heterarchical organization for multiple life-cycle perspectives to view and comment on a shared representation of the design. It is based on three underlying concepts:

1. Integrating life-cycle concerns through the use of views from multiple perspectives, where each perspective represents a different life-cycle concern such as manufacturing, structures, materials, *etc*;

2. Representing the design at various levels of abstraction and granularity through the use of features, where features are the attributes that characterize a design from the viewpoint of any perspective;

3. Generating and pruning the design space through the use of constraints.

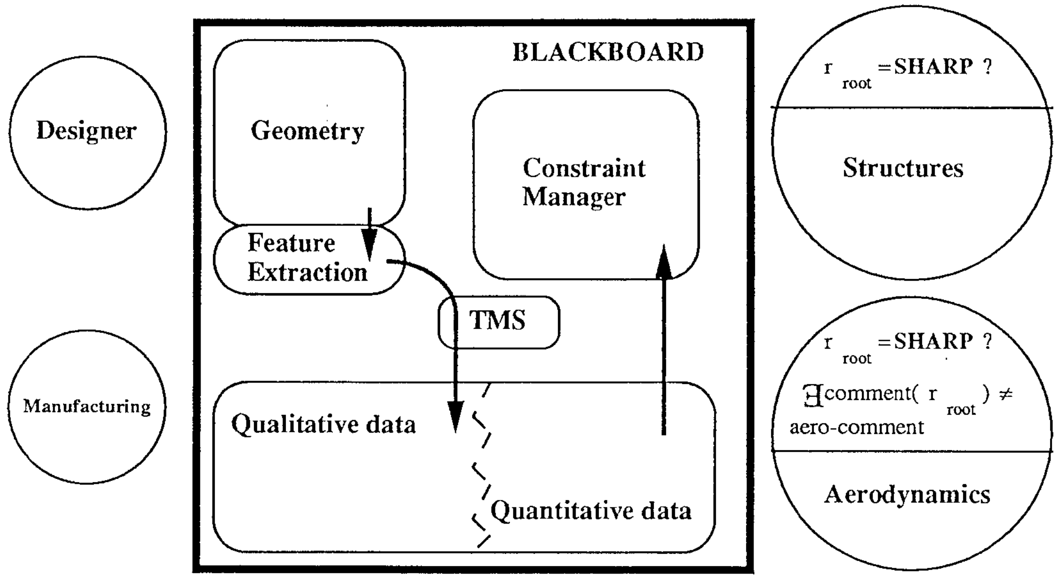


Figure 3: Flow of control in Design Fusion

Figure 3 depicts the organization of the system. The designer and three life-cycle perspectives are represented: manufacturing, structures and aerodynamics. Each of these perspectives encapsulates expertise for the relevant specialty. They act as proxies, providing feedback to the designer based upon qualitative and quantitative models. The perspectives monitor the design through the blackboard, which represents the design in terms of geometry (and topology), constraints, and a database of qualitative design features and quantitative parameter values derivable from the geometry.

A non-manifold representation scheme [18] for geometry provides a neutral representation of the design. The perspectives extract features from the geometry, and calculate parameter values from the features. Pinilla et al. [11] describe a formal method to describe shapes and features that relate closely to very low-level models of objects. This provides an interlingua for the perspectives to communicate about an artifacts form.

Parameters extracted from the design are recorded in the database. A truth-maintenance system (TMS) tracks the dependency between the data in the database and the geometry from which it was derived. It is used to detect when assertions in the database are no longer supported by the current geometry. Revisions are also explicitly represented in the database as relations between assertions.

Perspectives monitor database changes, and are invoked to comment on the design at appropriate times. For example, in Figure 3 the structural perspective is monitoring the design for sharp root radii. Likewise, the aerodynamic perspective is monitoring that no other perspective comments on sharp root radii.

When a monitoring condition is satisfied, the life-cycle perspective is invoked. It may express preferences about the design by commenting to the designer or imposing constraints on the design. For example, when the structural perspective recognizes a blending surface in the design, it can impose a constraint on the sharpness of the surface. When this constraint is violated, it can comment to the designer its concern about radii sharpness. Likewise, when the aerodynamic perspective detects an airfoil in the design, it can express its preferences on the radii of blending surfaces by imposing constraints on the design. It is important to note that aerodynamics notion of *sharpness* and structures notion of *sharpness* need not be the same. Each perspective defines its own *sharpness*-feature through the feature extraction language.

These perspective-specific features are extracted from the neutral geometric representation.

A constraint manager coordinates constraints imposed by the designer and perspectives. It has three functions. First, it detects violated constraints. When a constraint violation occurs, the perspective that imposed the constraint is invoked to process the violation. Second, it can propagate known values to determine consistent assignments for other design variables. Lastly, the constraint manager facilitates opportunistic reasoning. There are times during the design process when the designer desires to perform some analysis, but all the inputs are not known or have not been specified. Through constraints, these inputs can be identified, and plans generated to acquire the necessary design parameters.

Together, these components form a problem-solving architecture that meets the requirements enumerated in Section 2. It permits opportunism through a shared representation of design assertions and constraints. Multiple perspectives encapsulate qualitative and quantitative models to provide the designer feedback from life-cycle proxies. The system remains focused by observing changes to a database of facts about a design coordinated by the designer.

5 Conclusion

We have shown how artificial intelligence techniques can provide an architecture for computer-assisted design. The role of this architecture is to integrate around a common representation partial solutions to portions of the design problem. This architecture is a general model of problem-solving to free designers from the process of design and permit them to concentrate on the design.

The architecture must meet a set of requirements to fulfill this role. As a general model of problem-solving, the architecture must support the problem-solving process. The architecture must support management for design constraints and multiple design view-points, or perspectives. It must support communication and negotiation for these perspectives by providing a common representation of the design. A representation for the focus of the designer must also be supported. Databases represented as semantic networks and control from pattern-directed inference and architectural structure are the components of an architecture which are brought together to satisfy these requirements.

The Design Fusion system is an example of a system that uses these components. It combines multiple life-cycle perspectives which view a common design through perspective-specific features. The goal of this effort is to create the underlying theories and methodologies for a computer-based system that will assist in creating mechanical designs that meet their function, cost and quality requirements while simultaneously meeting the constraints imposed by life-cycle activities such as manufacturing and service.

6 Acknowledgments

We wish to thank Susan Finger for all her help in the preparation of this paper. We also wish to thank Caroline Hayes, Gilad Amiri, Monica Cellio, Mark Perlin, Chris Young and all the members of Carnegie Mellon's Design Fusion project for all their pertinent comments on the paper.

References

[1] O. Akin. *Psychology of Architectural Design.* Pion Limited, London, United Kingdom, 1986.

[2] C. Alexander. *Notes on the Synthesis of Form.* Harvard University Press, Cambridge MA, 1965.

[3] C. Alexander. A city is not a tree. *Ekistics*, 139:344–348, 1968.

[4] R. J. Brachman. *On the Epistemological Status of Semantic Networks*, pages 191–216. Morgan Kaufmann, Palo Alto CA, 1985.

[5] D. C. Brown and B. Chandrasekaran. Knowledge and control for a mechanical design expert system. *IEEE Computer*, 19(7):92–100, July 1986.

[6] L. A. Stauffer D. G. Ullman and T. G. Dietterich. Preliminary results of an experimental study of the mechanical design process. In M. B. Waldron, editor, *Proceedings from the NSF Workshop on the Design Process*, pages 145–188, Oakland, CA, February 8-10 1987. Ohio State University.

[7] R. Davis and R. G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20:63–109, 1983.

[8] J. de Kleer and J. S. Brown. The origin, form and logic of qualitative physical laws. In *Proceedings of the Eighth International Joint Conference on Artificila Intelligence*, pages 1158–1169, 1983.

[9] J. de Kleer and J. S. Brown. Theories of causal ordering. *Journal of Artificial Intelligence*, 29(1):33–62, 1986.

[10] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Journal of Artificial Intelligence*, 32(1):97–130, 1987.

[11] J. M. Pinilla et al. Augmented topology grammar for feature recognition. In *1989 NSF Engineering Design Conference*, June 1989.

[12] S. Finger et al. The design fusion project: A product life-cycle view for engineering design. In H. Yoshikawa and T. Holden, editors, *IFIP WG 5.2 Workshop on Intelligent CAD*, pages 165–172, September 1988.

[13] M. S. Fox. Organization structuring: Designing large complex software. Technical Report CMU-CS-79-155, Carnegie Mellon University, Pittsburgh PA, December 1979.

[14] M. S. Fox. *Constraint Directed Search: A Case Study of Job-Shop Scheduling*. PhD thesis, Carnegie Mellon University, 1983.

[15] M. S. Fox. Observations on the role of constraints in problems-solving. In *Proceedings of the Sixth Canadian Conference on Artificial Intelligence*, pages 172–187, May 1986.

[16] J. S. Gero and R. D. Coyne. *Knowledge-Based Planning as a Design Paradigm*, pages 289–323. Elsivar Publishing, 1987.

[17] B. J. Grosz. *The Representation and Use of Focus in a System for Understanding Dialogs*, pages 353–362. Morgan Kaufmann, Palo Alto CA, 1986.

[18] E. L. Gursoz and F. B. Prinz. Corner-based representation of non-manifold surface boundaries in geometric modeling. Dfm-edrc, Carnegie Mellon University, Pittsburgh, PA, 1988.

[19] F. Hayes-Roth and V.R. Lesser. Focus of attention in a distributed logic speech understanding system. In *Proceedings of the International Joint Conference on Artificila Intelligence*, pages 27–35, 1977.

[20] Y. Iwasaki and H. A. Simon. Causality in device behavior. *Journal of Artificial Intelligence*, 29(1):3–32, 1986.

[21] R. H. Katz. *Information Management for Engineering Design*. Springer-Verlag, New York NY, 1985.

[22] B. Kuipers. Commonsense reasoning about causality: Deriving behaviour from structure. *Journal of Artificial Intelligence*, 24:169–203, 1984.

[23] B. Kuipers. Qualitative simulation. *Journal of Artificial Intelligence*, 29(3):289–337, 1986.

[24] I. Hulthage M. D. Rychener, M. L. Farinacci and M. S. Fox. Integration of multiple knowledge sources in aladin, an alloy design system. In *Proceedings of the Fifth National Conference on Artificila Intelligence*, pages 878–882, 1985.

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

[26] A. Newell and H. Simon. *Human Problem Solving*. Prentice-Hall, Englewood, New Jersey, 1972.

[27] B. Ostrofsky. *Design, Planning, and Development Methodology*. Prentice-Hall, Inc, Englewood Cliffs NJ, 1977.

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

[29] J. R. Rinderle. Implications of function-form-fabrication relations in design decomposition strategies. *Computers in Engineering*, 1, 1986.

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

[31] A. Sathi and M. S. Fox. *Constraint Directed Negotiation of Resource Reallocations*. Morgan Kaufmann, Los Altos, CA, 1989.

[32] H. A. Simon. *The Architecture of Complexity*, chapter 7, pages 192–229. MIT Press, Cambridge MA, 1968.

[33] K. Sycara. *Resolving Adversarial Conflicts: An Approach Integrating Case-Based and Analytic Methods*. PhD thesis, School of Information and Computer Science Georgia Institute of Technology, Atlanta, GA, 1987.

[34] D. S. Weld. The use of aggregation in causal simulation. *Journal of Artificial Intelligence*, 30(1):1–34, 1986.