AI Magazine Volume 10 Number 3 (1989) (© AAAI)

Knowledge-Based System Applications in Engineering Design: Research at MIT

Duvvuru Sriram, George Stephanopoulos, Robert Logcher, David Gossard, Nicolas Groleau, David Serrano, and Dundee Navinchandra

Advances in computer hardware and software and engineering methodologies in the 1960s and 1970s led to an increased use of computers by engineers. In design, this use has been limited almost exclusively to algorithmic solutions such as finite-element methods and circuit simulators. However, a number of problems encountered in design are not amenable to purely algorithmic solutions. These problems are often ill structured (the term ill-structured problems *is used here to denote problems that do not have a clearly defined algorithmic solution), and an experienced engineer deals with them using judgment and experience. AI techniques, in particular the knowledge-based system (KBS) technology, offer a methodology to solve these ill-structured design problems. In this article, we describe several research projects that utilize KBS techniques for design automation. These projects are (1) the Criteria Yielding, Consistent Labeling with Optimization and Precedents-Based System (CYCLOPS), which generates innovative designs by using a three-stage process: normal search, exploration, and adaptation; (2) the Concept Generator (CONGEN), which is a domain independent framework for conceptual or preliminary design; (3) Constraint Manager (CONMAN), which is a constraint-management system that performs the evaluation and consistency maintenance of constraints arising in design; (4) the distributed and integrated environment for computer-aided engineering (DICE), which facilitates coordination, communication, and control during the entire design and construction/manufacturing phases; and (5) DESIGN-KIT, which can be envisioned as a new generation of computer-aided engineering environment for process-engineering applications.*

The types of problems that engineers normally solve are bounded by the derivation-formation spectrum. In derivation-type problems, the solution to the problem consists of identifying an outcome or hypothesis from a finite set of possible outcomes known to the problem solver. In formation-type problems, however, the problem solver only has the knowledge of how to form the solution. The problem solver then utilizes several problem-solving techniques to arrive at a solution.

The problem of design falls at the formation end of the spectrum. *Design* can be viewed as the process of specifying a description of an artifact that satisfies constraints arising from a number of sources by using diverse sources of knowledge. The constraints can be prespecified or can evolve during the design process. In this article, we provide an overview of some representative research projects that utilize AI techniques, in particular knowledge-based systems (KBSs), for engineering design automation at the Massachusetts Institute of Technology (MIT). The design process is described in the next section, followed by a discussion of the classes of design activities in Types of Design Activity. Several research projects in design automation are described in the subsequent five sections.

Design can be viewed as the process of specifying a description of an artifact that satisfies constraints arising from a number of sources by using diverse sources of knowledge.

Design Process

The design process can be viewed as an iterative six-step process:

1. Problem Identification: The problem, necessary resources, target technology, and so on, are identified at this stage.
2. Specification Generation: Design requirements and performance specifications are listed.

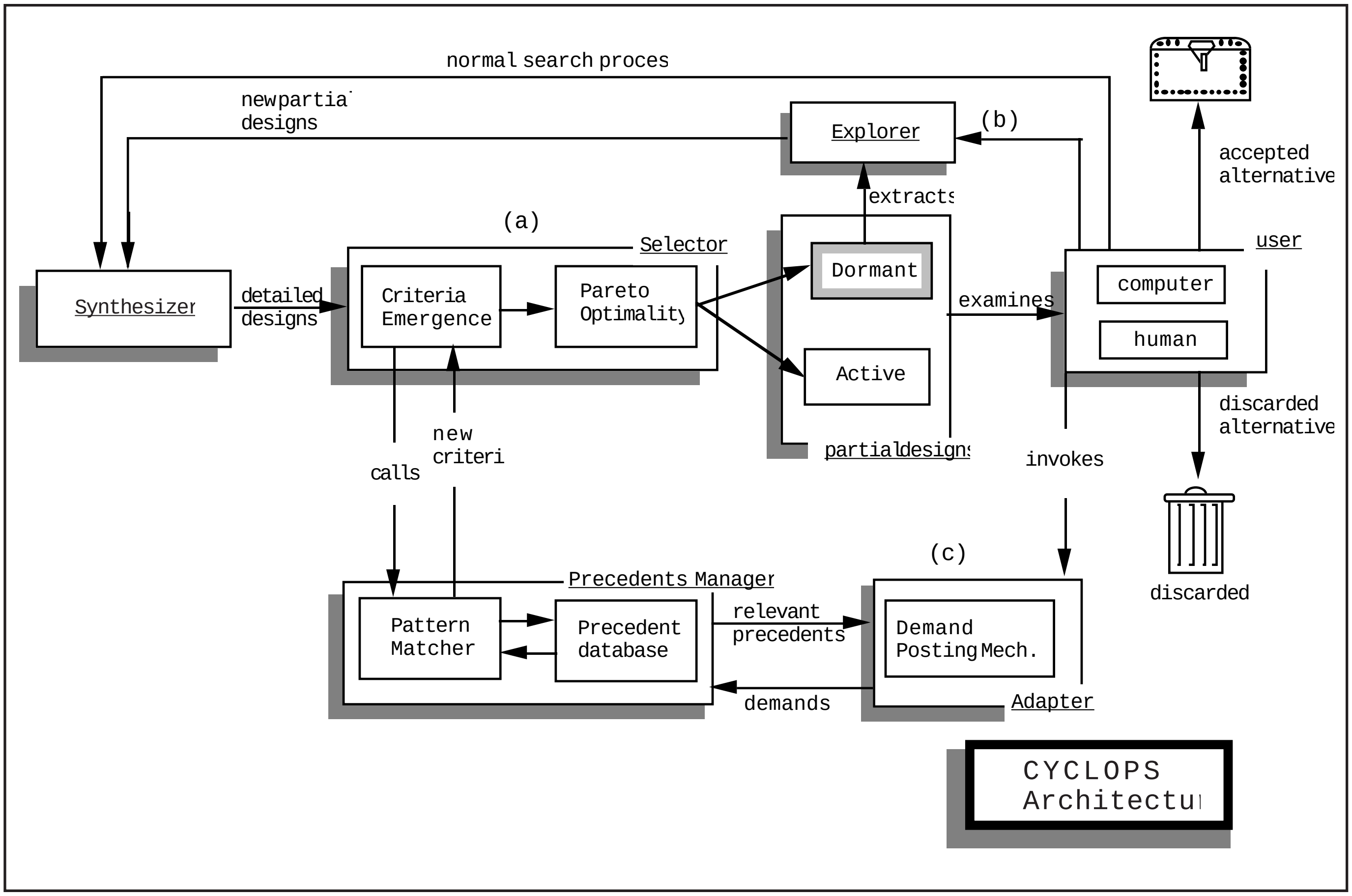


Figure 1. Architecture of CYCLOPS.

3. Concept Generation: The selection or synthesis of preliminary design solutions satisfying a few key constraints is performed. Several alternative designs might be generated.
4. Analysis: The response of the system to external effects is determined using an appropriate model for the system.
5. Evaluation: Solutions generated during the concept generation stage are evaluated for consistency with the specifications. If several designs are feasible, then (normally) an appropriate evaluation function is used to determine the best possible design to refine.
6. Detailed Design: Various components of the system are refined so that all applicable constraints (or specifications) are satisfied.

Significant deviations might exist between the component properties assumed or generated at the concept generation stage and those determined at the detailed design stage, which necessitates a reanalysis. The process continues until a satisfactory or optimal design is obtained.

Types of Design Activity

It is useful to classify design into various categories. These design classes can be thought of as being bounded by the creative-routine spectrum. These classes, which are process-based rather than product-based, are described as follows.

- **Creative design.** An a priori plan for the solution of the problem does not exist, where the plan is an abstract decomposition of the problem into a set of levels that represent choices for the component or object hierarchy of the problem. Rather than using a convergent line of reasoning, the designer uses a divergent thought process. The key element in this design type is the transformation from the subconscious to the conscious.
- **Innovative design.** The decomposition of the problem is known, but the alternatives

for each of its subparts do not exist and must be synthesized. The designer uses some fundamental principles of the domain to develop the alternatives; these alternatives might be a novel combination of existing components. One must note that a certain amount of creativity comes into play in the innovative design process. Further, a system considered as innovative in one culture might not seem innovative in another culture.

- **Redesign.** An existing design is modified to meet changed functional needs.
- **Routine design.** An a priori plan of the solution exists. The subparts and alternatives are known in advance, perhaps as a result of either a creative or an innovative design process. The solution involves finding the appropriate alternatives for each subpart that satisfy the given constraints.

At the creative end of the spectrum, the design process might be nebulous (hazy), spontaneous, chaotic, and imaginative, whereas at the routine end, the design is precise, predetermined, systematic, and mathematical. Brown and Chandrasekaran (1985) classify the first two classes as class I and class II designs and the fourth as a class III design. Several applications developed at MIT that span the creative-routine design spectrum are described in the following sections.

Innovative Design: CYCLOPS

CYCLOPS is an experiment in innovative design. The system was developed to investigate the following observations about design: (1) Designers work with multiple objectives and constraints but are not always bound by them. (2) Designers are not mere satisfiers: They like to produce optimal designs. (3) New criteria emerge as design progresses. (4) Past design examples are extensively used in problem solving.

The CYCLOPS system is based on the thesis that innovative designs can be obtained by exploring a wide variety of alternatives. Innovative designs are not obtained through some deliberate attempt at producing them but by generating lots of design alternatives and throwing away the bad ones. Consequently, the ability of a system to innovate depends on how well it can generate diverse alternatives that break away from the norm and the governing constraints. This idea is based on the observation that new alternatives sometimes serendipitously lead to interesting solutions. Our system explores design alternatives by relaxing the governing criteria.

The system's exploratory capabilities are supplemented by its ability to reason from design cases (we use the words precedent and case interchangeably in this section). The ability to use knowledge drawn from experiences inside or outside the current domain helps produce innovative solutions. In particular, a design appears novel if it incorporates knowledge acquired from past experiences in domains different from the current one.

CYCLOPS, whose domain is landscape design, operates in three modes: normal search, exploration, and adaptation. These modes are described in the following subsections.

Normal Search Mode

In normal search mode, CYCLOPS uses a multiobjective version of the A* algorithm—Pareto-Optimal A*—to find all nondominated solutions (provided they exist). The search process uses two modules: the synthesizer and the selector (figure 1). The synthesizer takes partial designs and adds detail to them by instantiating new variables. The selector checks for dominance and places the designs in a dormant or an active list. Dominance is determined by plotting the partial designs on a multiobjective pareto graph and selecting the pareto-optimal points. An example of a pareto graph for two objectives (O1 and O2) is shown in figure 2. After selection, the active designs are returned to the synthesizer for further detailing.

Exploration Mode

In the exploration mode, CYCLOPS relaxes the governing criteria and searches alternatives outside the original solution space using an augmented version of the Pareto-Optimal A* algorithm. The explorer module generates new alternatives by relaxing the constraints and objectives that bound the space (figure 1). In effect, relaxation increases the size of the solution space, allowing the system to examine designs in the state space that normally would have been pruned off (Navinchandra and Marks 1986). Two types of criteria are used in CYCLOPS: constraints and objectives. Both criteria define the solution space as a subset of the larger state space of possible designs. We can explore the space by relaxing either of these criteria.

Relaxing constraints produces new alternatives, some of which can fortuitously lead to improved designs. For example, a landscape designer who relaxes the constraint that all homes be on slopes less than 8 percent to some higher value (say 10 percent) makes available plots of land that are on a slope between 8 percent and 10 percent. It is possi-

CYCLOPS performs this kind of reasoning by matching against the subgoals and relations in the causal explanation underlying the precedent.

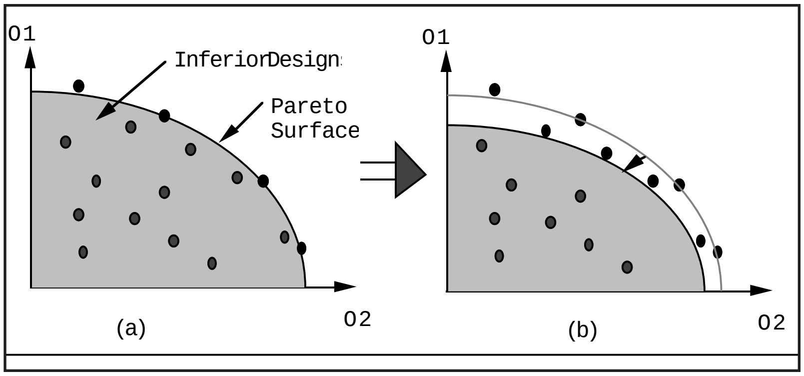


Figure 2. Pareto Slip.

ble that some of these new alternatives might provide opportunities such as better soil conditions or better view. Relaxation overcomes the artificial precision built into a criterion. When we say that the slope should be less than 8 percent, it does not mean that lots with slightly higher slopes of say 9 percent or 10 percent be completely avoided.

Objectives can be relaxed by deliberately pushing the pareto surface toward the origin (figure 2). The figure shows that only a few of a large group of designs are nondominated and lie on the pareto surface. The curve defines the trade-off between objectives O1 and O2. The situation depicted in figure 2a can be relaxed by deliberately pushing the pareto surface toward the origin, as shown in figure 2b. In effect, all solutions that lay just below the pareto surface are available for consideration. It is in this way that new alternatives can be brought into consideration through objective relaxation.

The alternatives generated by exploration are passed to the selector through the synthesizer. The selector tries to see if potential opportunities exist in the generated designs, which is done by emerging new criteria. The selector has a criteria-emergence submodule that matches design alternatives against precedents (figure 1). The ability to emerge new criteria is implemented by providing the selector with a database of past experiences (precedents). In its simplest form, a *precedent* is a record of an experience or episode; it is a record of the conditions and the effects experienced. The effects can be physical or emotional. All precedents are input by the programmer or knowledge engineer. The program matches design alternatives to the precedents in the database. If a design has certain characteristics that match a favorable past experience, the current design is viewed as favorable. For example, when CYCLOPS was given the problem of finding a location for a new house, it considered alternatives by normal search and exploration. During exploration, it relaxed a constraint on the maximum allowable slope and found a location high on a hillside. On examining this alternative, it was reminded of a precedent about how an elevated location provides a good panoramic view. After examining the precedent, the program extracted a new criterion about view. It then reevaluated the existing alternatives using the new criterion. New criteria often change the focus of the problem-solving process. This phenomenon is called *criteria emergence*. The matching algorithm is explained next.

Adaptation Mode

In the adaptation mode, CYCLOPS uses precedent knowledge to debug design problems. This knowledge is achieved by the adapter module (figure 1). The adapter uses a technique called *demand posting*, where demands are posted to the precedents database manager, and appropriate precedents are retrieved. The debugger works by recursively applying the following steps: (1) looking for and recognizing bugs in a given design; (2) explaining the causes of bug; (3) repairing the bug by recalling and applying relevant precedents; and finally, (4) reexamining the design for new bugs. The third step is the central function of the precedent-based debugging process.

The demand-posting technique helps CYCLOPS solve design problems by drawing analogies. Whenever CYCLOPS encounters a design problem that does not directly match any past experience, it tries to reason by analogy. Drawing an analogy requires the ability to match a precedent and the target problem even though they have different characteristics. For example, a landscape designer faced with the problem of locating a house on a steep slope might solve the problem by placing the house on stilts. The designer might get this idea by reasoning analogically from a precedent about how villagers in Thailand put their huts on stilts to avoid flooding. Notice that the purpose for using stilts in the base precedent is different from the purpose of the target problem. The matching is not based on surface features of base and target but on a deeper understanding of why the huts are put on stilts. CYCLOPS performs this kind of reasoning by matching against the subgoals and relations in the causal explanation underlying the precedent. An explanation is usually a trace of how the different parts of the precedent relate to subgoals of the overall goal of the precedent. If a match is found, design strategies used in the relevant parts of the precedent are applied to the target problem. The process is continued until all bugs in the target problem are eliminated (Sycara and Navinchandra 1989).

The advantage of this technique is that it can match a base and target even if their main goals or surface features are radically different. For this reason, the program appears to reason analogically (Gentner and Toupin 1986).

The debugger has to retrieve past problem-solving precedents that provide appropriate repair strategies. An obvious way of finding such precedents is to use a description of the current bug as a cue into memory. Often, however, a cue might not retrieve relevant precedents, even if they exist in memory, because the cue might not have the right symbols in its representation or might be too vague or overly specific. We need to generate new cues that although related to the original cue are different enough to retrieve useful precedents. Several techniques for cue transformation have been suggested: elaboration (Kolodner 1980), condensation (Kolodner 1988), tweaking (Kass and Leake 1988; Schank 1986), and adaptation (Sycara 1987).

Other techniques for finding relevant precedents have been suggested by researchers working on the psychological aspects of human creativity and problem solving. The developers of techniques such as brainstorming (Osborn 1953) and synectics (Gordon 1961) suggest that creative problem solving is predicated on retrieving and using a wide variety of precedents. The techniques they use to facilitate this process are based on asking many questions (Recently, this idea was also examined in the AI literature [Schank 1986, 1988]). Questions serve as cues into memory and help retrieve precedents that normally would not have come to mind. CYCLOPS performs this kind of reasoning by redefining the given problem. For example, given problem X, the program first asks, “Is there a known way of eliminating X?” If it cannot find an appropriate precedent, it asks, “What are the causes of X?” “If it is not known how to eliminate X, can its causes be eliminated?” “Can its effects be reduced or eliminated?” Roughly speaking, the idea is to reduce a bug into subbugs that can relate either directly or analogically to precedents in memory. The reasons underlying a given bug are determined by developing a causal explanation for the existence of the bug. This questioning process is recursively applied until a solution is found (Navinchandra 1988, forthcoming).

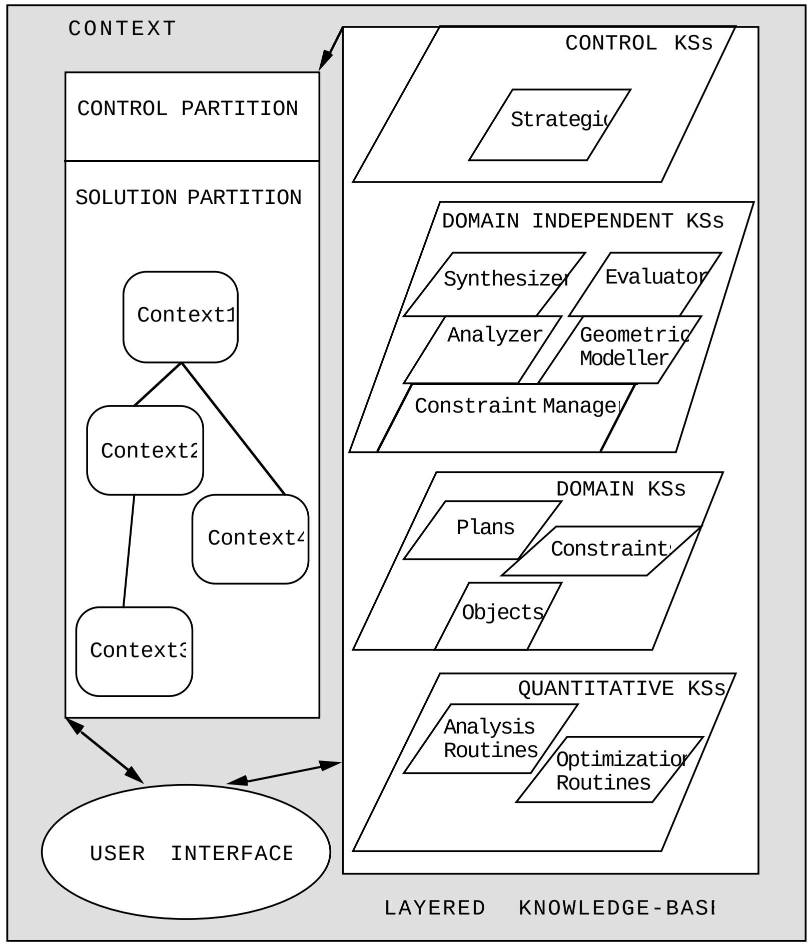


Figure 3. Overview of CONGEN.

Execution of CYCLOPS

CYCLOPS is started in the normal search mode. If no solutions are found or if the user makes a request, the program goes into the exploration mode. The adapter is only invoked by the user. This restriction was placed because adaptation is computationally expensive, and hence, it is not possible to try and adapt each and every design produced by the synthesizer.

Implementation Details

The first version of CYCLOPS was implemented at the MIT Intelligent Engineering Systems Laboratory in Franz Lisp™ on a Vax™ machine running the UNIX™ operating system. The CYCLOPS interface uses the X window system developed by Project Athena at MIT. CYCLOPS has been tested on landscape design problems with 10 to 15 land

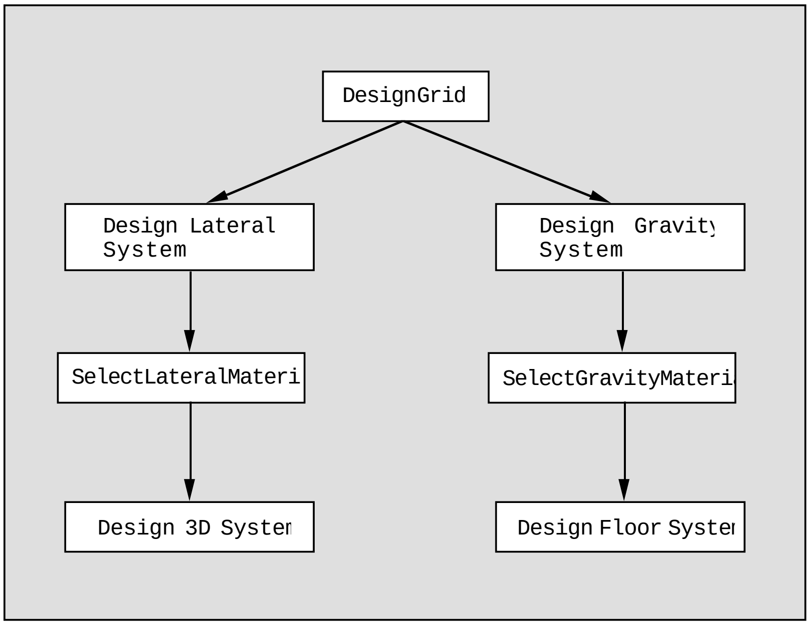


Figure 4. Partial Design Plan for Preliminary Structural Design.

uses, about a dozen constraints, and half a dozen objectives. The program usually adds three or four new objectives during a design process. The knowledge base has 25 precedents.

Contact: D. Navinchandra or D. Sriram.

Routine Design: CONGEN

CONGEN is being developed as a domain-independent framework for conceptual (or preliminary) design. This system includes some of the features incorporated in earlier knowledge-based expert systems (KBESs) such as ALL-RISE (Sriram 1987) and Pinch Roll Interactive Design Expert (PRIDE) (Mittal, Dym, and Morjaria 1985) that were developed for conceptual design. CONGEN consists of a layered knowledge base, a context mechanism, and a friendly user interface, as shown in figure 3. The knowledge base and the context are described briefly in the following subsections.

Knowledge Base

The knowledge base consists of a number of knowledge sources (KSs) that are organized into several layers or levels. Briefly, we are incorporating four KSs into CONGEN. Strategy-level KSs determine the appropriate domain-independent KS to fire, depending on the information provided in the control partition of the context. Because this level is used to control various tasks, such as the activation of other KSs, it comprises the task control knowledge. Domain-independent KSs (DIKSs) perform specific tasks involved in design in a domain-independent manner; DIKSs can be viewed as KBES shells. In the current implementation, we are incorporating four DIKSs. First, the synthesizer takes a set of specifications (or constraints) and generates one or more conceptual designs. A description of the synthesizer is provided in the next subsection.

Next, the evaluator performs a preliminary evaluation of all the feasible alternative solutions that are generated by the synthesizer. The evaluator acts on a network of object templates; this network exists in the domain KS level. The root object of this network contains details of the evaluation, such as features needed for evaluation, and the evaluation function. The child nodes (or objects) represent various features; the value of each feature is determined by traversing through the alternative solution, which is represented as a tree in the solution partition of the context.

Third, the geometric reasoner KS is an intelligent computer-aided design (CAD) graphics system that performs the following tasks when fully implemented: (1) understands engineering sketches and drawings, (2) generates geometric models and reasons about these models, and (3) performs interference checking between design objects.

Fourth, the constraint manager KS performs the evaluation and consistency maintenance of constraints arising in design. The constraint manager is further described in the next section.

Domain KSs contain knowledge for a particular domain. These KSs are used by DIKSs. Design plans, goals, constraints, objects, and analysis procedures are some KSs that can be incorporated at this level. For example, for preliminary (or conceptual) structural design (Sriram 1987), the design plan is represented as shown in figure 4. The classes of constraints that arise in structural design (civil and mechanical) can be categorized into (1) synthesis constraints, which affect the generation of feasible configurations, including spatial requirements and heuristics that represent the designer's style or experience; (2) interaction constraints, which arise from the interaction of structural subsystems, including the compatibility of materials and structural behavior; (3) causal constraints, which represent equations of equilibrium, compatibility relationships, and the response of structural systems to their environment; and (4) parametric constraints, which are constraints on parameters (attributes) of components, including strength and serviceability constraints. Sample synthesis and interaction constraints for preliminary (conceptual) structural design follow.

Synthesis Constraint

IF	material is concrete, AND
	30 <= no. of stories <= 40, AND
	the formwork is not expensive
THEN	the constraint is satisfied

Interaction Constraint

IF	3D.lateral.load.material =
	3D.gravity.load.material
THEN	the constraint is satisfied

Quantitative KSs contain the analytic knowledge and reference information required for analysis and design.

Synthesizer KS

The synthesizer KS takes a set of specifications (or problem-specific constraints) and generates one or more preliminary designs. It can be viewed as a problem solver for a consistent labeling problem (Haralick and Queeney 1982; Mackworth 1977) and can be defined by the following tuple:

$\{U, L, C, T, R, \delta\}$,

where

$U = \{u_1, u_2, ..., u_n\}$ is a set of units that need to be assigned; these units can be hierarchically decomposed into subunits.

$L = \{l_1, l_2, ..., l_n\}$ is a set of labels that can be assigned to U; the elements of L can be known in advance or generated using a known function.

C is a set of constraints that need to be satisfied by every feasible solution. (Four types of constraints—interaction, synthesis, causal, and parametric—were described earlier. Other design problems can incorporate similar categories of constraints. In Mackworth [1977], the interaction constraints were classified into arc constraints when two units are involved in a constraint and path constraints when more than two units are involved in a constraint; the synthesis and parametric constraints were classified under node constraints.)

$T = \{(u_1, l_1)\ (u_2, l_2)\ ....,\ (u_n, l_m)\}$ is a set of legal assignments.

R is a set of feasible solutions.

δ is a set of operators that set up the sequence of unit assignments, that is, hierarchical refinement.

In the current KBES for design, the units are normally represented as parts of a design plan. For example, the units in a preliminary (conceptual) structural design system (figure 4) are 3D-lateral-mat (from the select lateral material task), 3D-gravity-mat (from the select gravity material task), and so on.

Context

The context consists of all the solutions generated during conceptual design. It is divided into two parts: the control partition, which is used for storing general information, and the solution partition, which comprises a tree of contexts. Multiple solutions (or partial solutions) to the design problem can be obtained from the leaf node contexts.

Implementation Details

CONGEN is being developed on three platforms: SUN, IBM PC, and Macintosh II. The SUN and Macintosh versions are being implemented in Parmenides/Frulekit, which is a frame-based–rule-based language developed in Common Lisp at Carnegie-Mellon University by Jaime Carbonell's research group; the user interface for the SUN version will be developed in the X Window environment. The IBM PC version is being developed in KAPPA™, which is a C-based hybrid programming environment marketed by Mega-Knowledge Inc. We hope to release the synthesizer and the constraint manager KS modules for the Macintosh during the Summer of 1990.

Contact: D. Sriram.

CONMAN

Constraints are continually added, deleted, and modified throughout the development of a new product. For example, the initial set of specifications might be augmented, changed, or refined as the design progresses. The resulting constraint set can contain conflicting or unrealizable requirements. The management of these constraints throughout the evolving design is a nontrivial task. The constraints are often numerous, complex, and contradictory. In complex designs, where form, function, and physics strongly interact, it is difficult to track all relevant constraints and parameters and understand the basic design relationships and trade-offs. Effective tools for constraint management facilitate the conceptual design process. These tools should form an integral part of any design automation system.

A constraint-management system should have three functionalities.

1. Evaluation. One obvious elementary function of a constraint manager is to evaluate the set of constraints for given values of known parameters. This ability is essential to

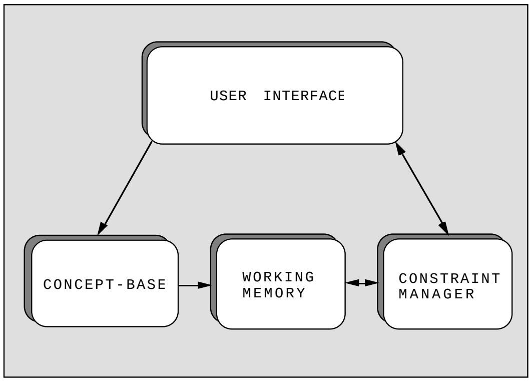


Figure 5. Overview of Concept Modeler.

enable the designer to examine the basic relationships and trade-offs between design parameters.

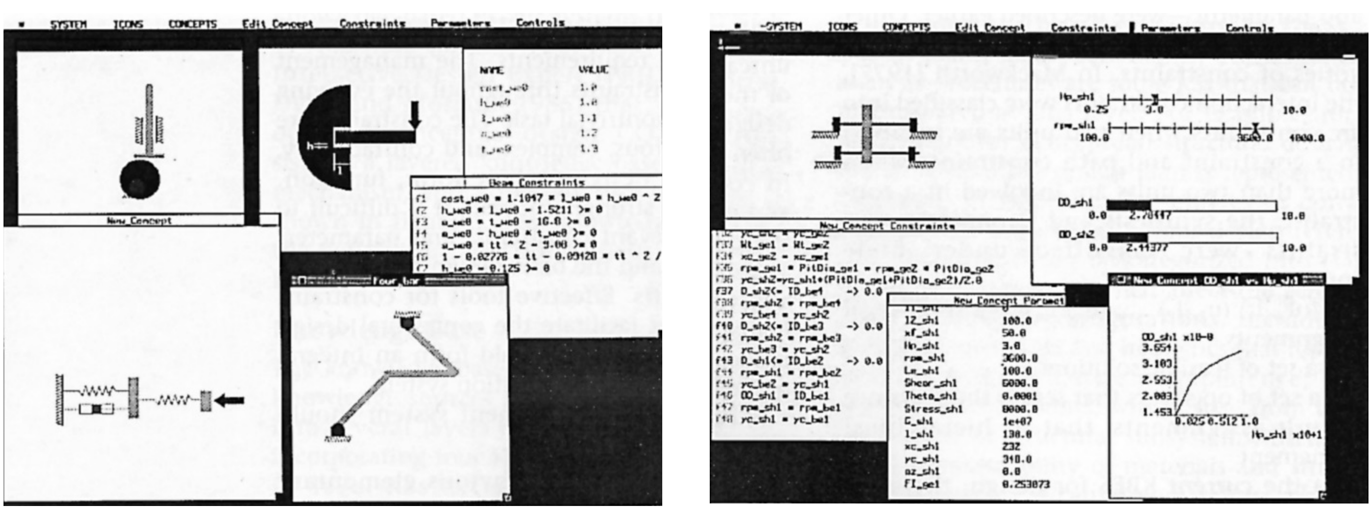
2. Minimization of computation. During evaluation, the computational effort should be minimized. This effort is generally accomplished by identifying and isolating those constraints relevant to a particular computation.

3. Consistency maintenance. Because constraints are continually added, deleted, and modified throughout the conceptual design process, it is possible for a constraint set to contain inconsistencies. If a constraint set contains inconsistencies and cannot be evaluated, it is desirable that the system be able to identify the redundant or conflicting constraints.

CONMAN has these functionalities and was implemented as part of a knowledge-based system—the concept modeler (figure 5). The concept modeler provides the user with a menu of predefined concept models of common mechanical engineering components; other components can be added to the system. Each component is represented as a frame in a concept base. Each frame encodes a set of constraints that predict its performance, establish its physical limits, and define its topological (connectivity) restrictions. In addition, graphic icons and other physical properties are included in each frame (figure 6a). In the current system, the user interactively selects individual components from a menu and specifies the connectivity relationships between these components. The system creates the aggregate concept model in the working memory by fusing the individual component models (figure 6b). The constraint manager then evaluates the aggregate model given the known parameters and identifies redundant or conflicting constraints. Concepts can be stored and retrieved and can be used as components in higher-level concepts. Currently, the constraint manager is being incorporated in CONGEN as the constraint manager KS.

Constraint Manager: Overview

The role of the constraint manager is exemplified by the causal dependency sphere metaphor (figure 7), which encloses a network of parameter relationships and interdependencies. The sphere interacts with the world through a series of devices that can be either input transducers or output actuators. These devices, shown in the figure as cylindrical rods that radially extend from the sphere, have various sizes denoting their relative importance. They can have a scale indicating values, and they can have limits on the values they can attain. The limits can have warning lights to signal a limit has been



a. b.

Figure 6. Concept Examples and Sample Screen from the Concept Modeler.

reached. The limits on the devices can be set by the designer. The designer can query the system by acting on any of the devices and observe the responses on all other devices; the devices respond according to the dependencies set within the sphere. The constraint manager functions in a similar manner. It provides useful information regarding the dependencies among the parameters, even when a feasible solution is found. This information can be used as a guide in searching for a better solution and to gain insight into the nature of the solution space. Details of how the constraint manager represents and processes constraints are provided in the following subsections.

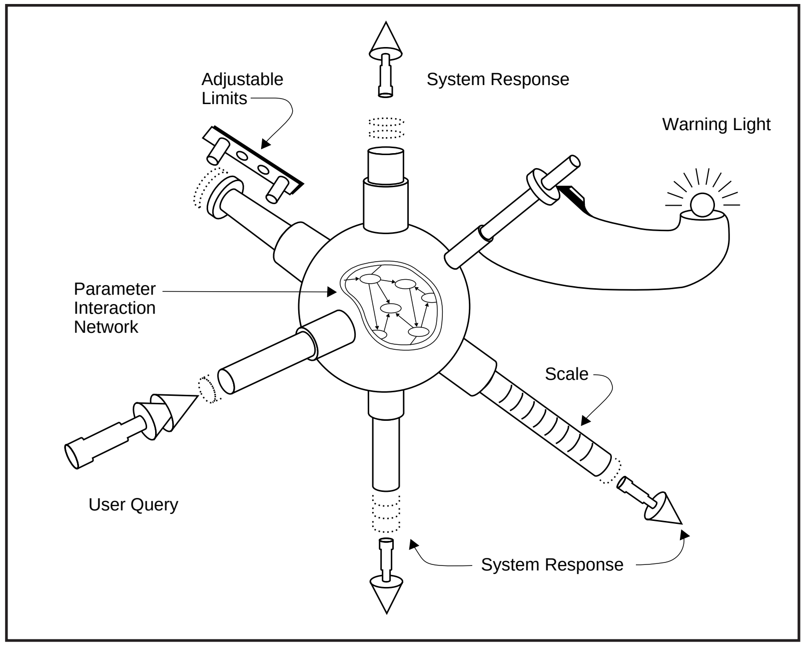


Figure 7. Causal Dependency Sphere.

Constraint Representation

A *constraint* is a set of parameters that behaves according to a specified relationship (or constraining mechanism). The parameters can take on continuous or discrete values. The status of a parameter is its classification as a (known) constant or (unknown) variable. The constraint relationships can be piecewise continuous equalities or inequalities. Individual constraints, as well as systems of constraints, are modeled as a *constraint graph,* which is a directed graph whose nodes denote parameters and whose arcs denote constraint relationships. Arcs are labeled according to the constraint they represent; the same label can appear on more than one arc. This representation is illustrated by considering the cantilever beam subjected to a point load, as shown in figure 8a. The associated constraints and the constraint network (or graph) are shown in figure 8b.

Constraint Evaluation

To evaluate a system of constraints, an assignment or matching of every unknown parameter to a particular constraint relationship must be performed (this matching problem has been extensively studied in the literature; see Serrano [1977] for a literature review). The matching is represented by the bipartite graph G = {V, E}. A graph is said to be bipartite when its set of nodes V is the union of two subsets N and F such that the intersection of these subsets is a null set (that is, $N \cap F = \emptyset$) and such that every member of its set of arcs E connects one element of N with one element in F; $N = \{n_1,\ldots,n_p\}$ is a set of p nodes that correspond to the unknown parameters; and $F = \{f_1,..,f_r\}$ is a set of r nodes that correspond to the set of constraints. Each arch in E $= \{e_1,..,e_k\}$ matches one unknown parameter with one constraint; no two arcs in E can have the same elements in N or F in

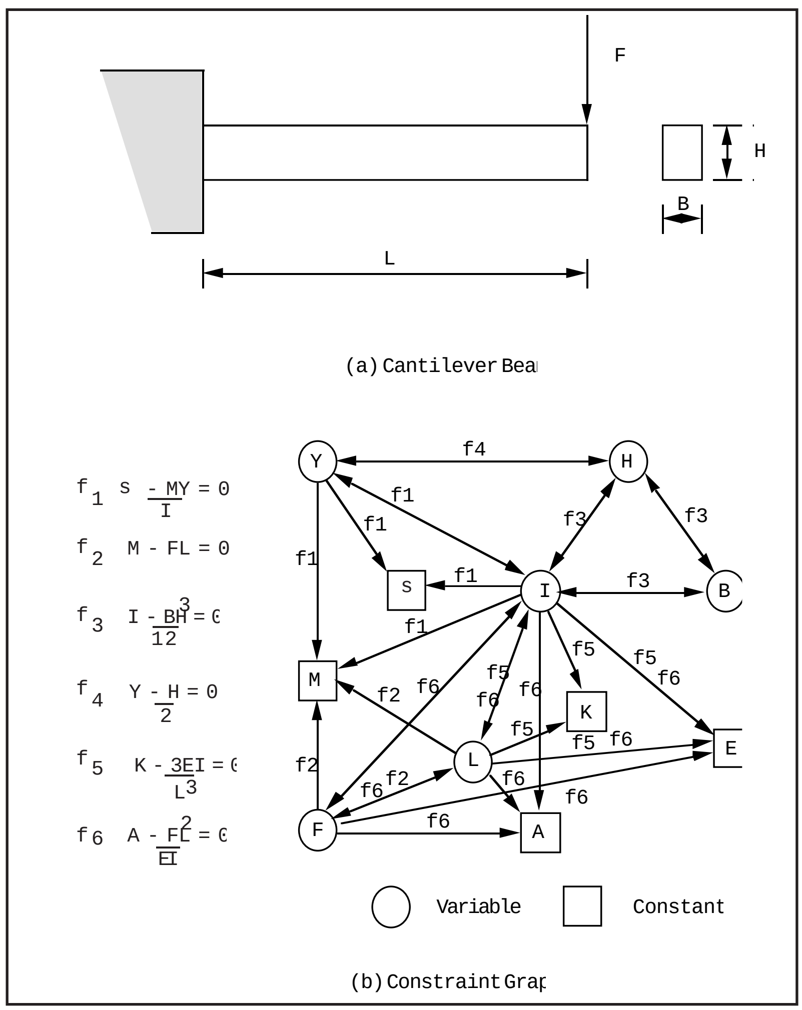


Figure 8. Example of a Constraint Representation.

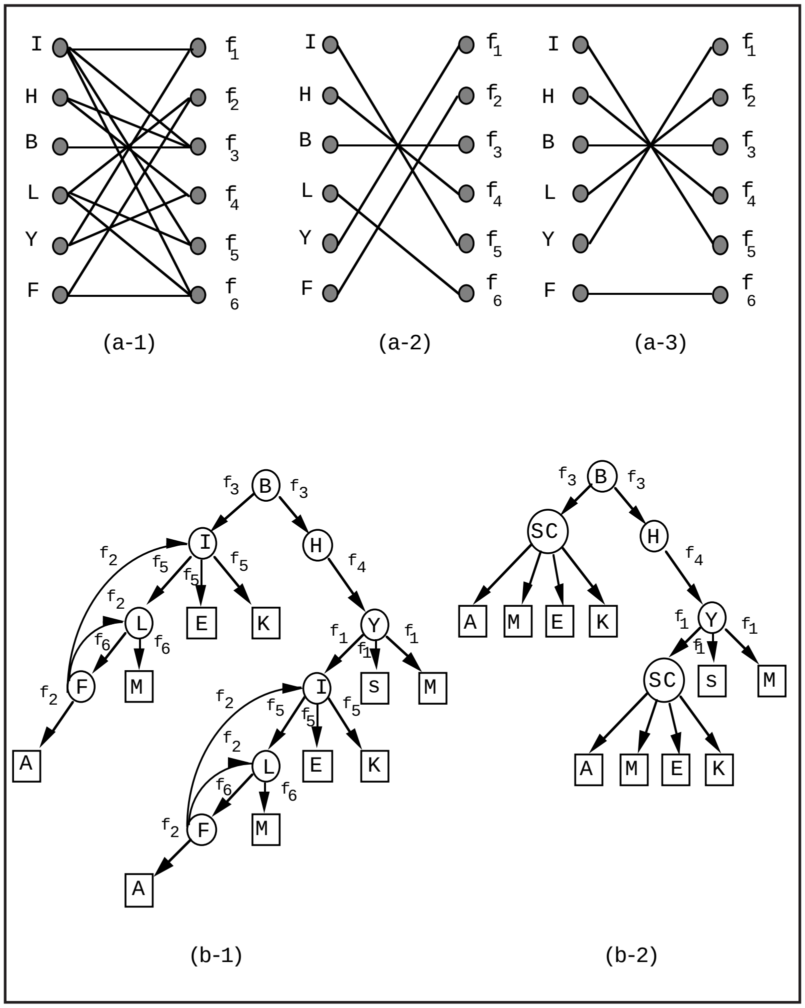


Figure 9. Example of Constraint Evaluation for Potential Variations Created When the Components and Interface Values Become Better Known.

Iteration proceeds toward increasing detail; design personnel can change, and their numbers can expand with increasing level of detail. This process introduces a number of problems in the engineering industry.

common. Therefore, each arc in the bipartite graph corresponds to a particular node (an unknown parameter) and one of its attached arcs (a constraint) in the original constraint graph. In the example introduced earlier, N = {I, H, B, L, Y, F} is the set of unknown parameters; KN = {M, E, A, K, S} is the set of known parameters; and F = {f_1, f_2, f_3, f_4, f_5, f_6} is the set of constraints. A bipartite graph before matching is shown in figure 9a-1, and bipartite graphs depicting two possible matchings are shown in figures 9a-2 and 9a-3.

This matching permits the original constraint graph to be converted to a tree-like structure, shown in figure 9b-1. At every node in the constraint graph representing an unknown parameter, all incoming arcs are deleted except those denoting the unknown parameter's matching constraint. The presence of cycles in the tree makes evaluation difficult because a cycle identifies a constraint subset that must be solved simultaneously. The subset consists of constraints matching the nodes that constitute a strong component in the diagraph. These strong components are collapsed; the resulting tree is shown in figure 9b-2. The unknowns are then evaluated using a reverse topological sort, which is described in Serrano (1977).

Implementation Details

The constraint-management system was implemented on an IRIS workstation in Franz Lisp. Recently, several companies have been developing constraint-management tools as a part of their design automation packages. Example constraint-management tools are Cognition Inc.'s MathSolve™, which is based on early work performed at the MIT Mechanical Engineering-CAD laboratory and is a representative constraint management system; Borland's Eureka™, which is a relatively inexpensive constraint solver available on personal computers but does not help identify the causes of constraint failure, as in Mathsolve™; and Premise Inc.'s variational geometry package which contains a constraint management facility for dealing with equality constraints.

The constraint manager has several limitations, such as the inability to handle inequalities. We are currently pursuing various augmentations to the constraint-management system.

Contacts: D. Serrano, D. Gossard, D. Navinchandra, and D. Sriram.

Cooperative Engineering Design

Most engineering projects involve a large number of components and the interaction of multiple technologies. The components included in the product are decided in an iterative design process. In each iteration, interfaces and interface conditions among these components are designed with slack to account for potential variations created when the components and interface values become better known. Iteration proceeds toward increasing detail; design personnel may change, and their numbers expand with increasing level of detail. This process introduces a number of problems in the engineering industry.

The problems facing the engineering industry in the United States will be high-

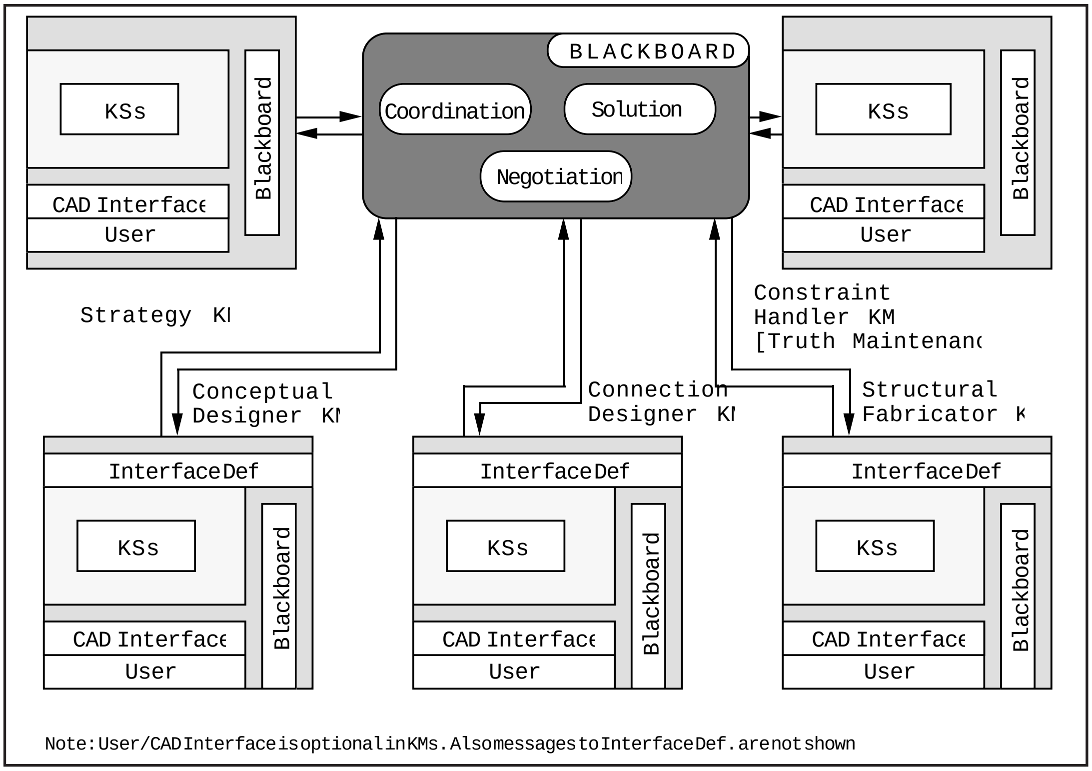


Figure 10. A Conceptual View of DICE for Design and Construction.

lighted by considering the design and construction of structures.[1] On a single project, interacting design technologies often come from separate firms or functional groups within a firm, and little coordination exists between designers and contractor(s) during design. Because designers find coordination among themselves difficult, they leave this task to construction managers or the contractor. Thus, working drawings, used to inform the contractor of the product, lack detail. Shop or fabrication drawings are required from the contractor to document details, but potential conflicts among trades are often unrecognized until construction begins.

Several undesirable effects are caused by this lack of coordination: (1) The construction process is slowed, and work stops when a conflict is found. (2) Prefabrication opportunities are limited because details must remain flexible. (3) Opportunities for automation are limited because capital-intensive, high-speed equipment is incompatible with work interruptions from field-recognized conflicts. (4) Rework is rampant because field-recognized conflicts often require design and field changes. (5) Conservatism pervades design because designers provide excessive slack in component interfaces to avoid conflict. (6) The industry is unprepared for the advent of automated construction because the need for experience in design limits choice to available materials placed by hand.

All these problems decrease productivity. In addition, failures, such as the 1981 Kansas City Hyatt Regency collapse where two skywalks in the lobby of the hotel collapsed, occur more often then they should. Overcoming these problems requires significant changes to the design process, together with superior computer-integrated design and construction-manufacturing (CIDCAM) tools.

Computer-aided tools, which are collectively called DICE, are currently being developed with these objectives[2]: (1) to facilitate effective coordination and communication between various disciplines involved in engineering; (2) to capture the process by which individual designers make decisions, that is, what information was used, how it was used, and what it created; (3) to forecast the impact of design decisions on manufacturing or construction; (4) to interactively provide designers with detailed manufacturing process or construction planning; and (5) to develop intelligent interfaces for automation.

DICE can be envisioned as a network of computers and users (called knowledge modules [KMs]), where the communication and coordination is achieved—through a global

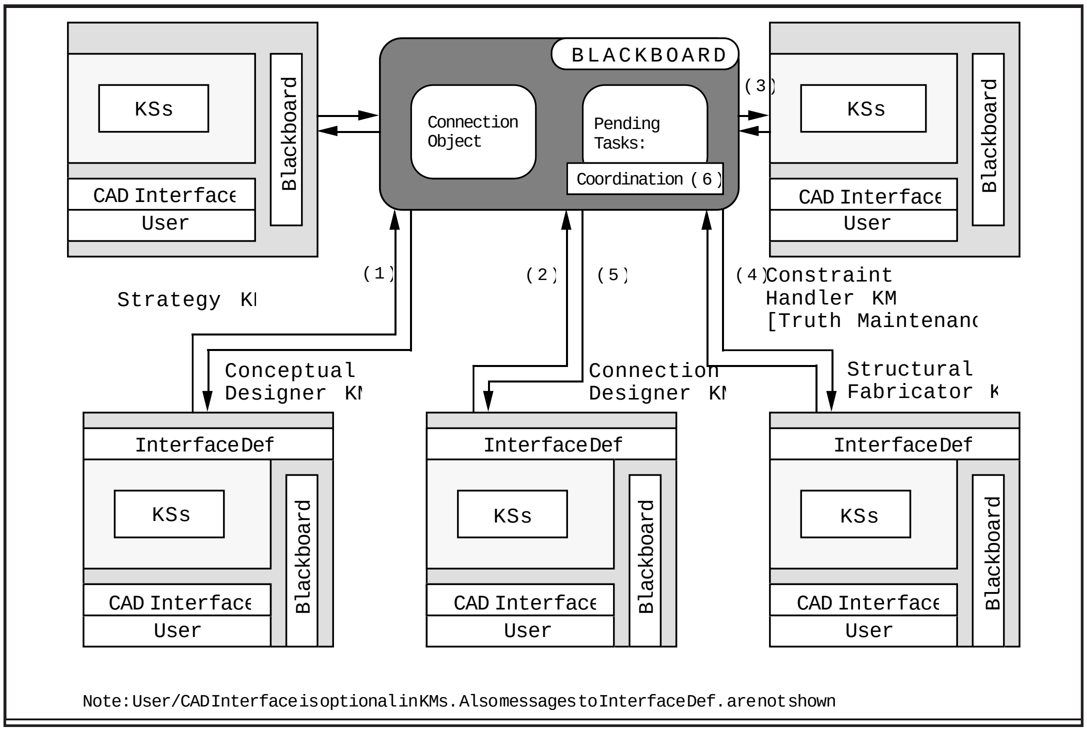


Figure 11. Evaluation and Propagation of Design Decisions.

database called blackboard—by a control mechanism. A conceptual view of DICE for design and construction is shown in figure 10; only a representative set of KMs are shown in the figure.

Blackboard

The blackboard is the medium through which all communication takes place. The blackboard in DICE is divided into three partitions: solution, negotiation, and coordination. The solution blackboard (SBB) partition contains the design and construction (or manufacturing) information generated by various KMs. This design information is represented in the form of an object hierarchy and contains information about the design product and process. The negotiation blackboard partition consists of the negotiation trace between various engineers taking part in the design and manufacturing (construction) process. The coordination blackboard (CORDBB) partition contains the information needed for the coordination of various KMs.

Knowledge Modules

Each KM can be viewed as a KBES, developed for solving individual design- and construction-related tasks; a CAD tool, such as a database structure, that is, a specific database, an analysis program, and so on; a computer user; or a combination of these. A KBES could be viewed as an aggregation of KSs. Each KS is an independent chunk of knowledge, represented either as rules or objects. In DICE, KMs are grouped into three categories: strategy, specialist, and quantitative. The strategy KMs help the control mechanism in the coordination and communication process. The specialist KMs perform individual specialized tasks in the design and construction process, and the quantitative KMs are mostly algorithmic CAD tools.

Control Mechanism

The communication, coordination, data transfer, and all other functions define the control mechanism. The control mechanism performs two tasks: (1) evaluates and propagates implications of actions taken by a particular KM and (2) assists in the negotiation process.

Task 1 is accomplished through methods associated with objects in the object hierarchy of SBB and a truth maintenance system (TMS) that keeps the global database in a consistent state. If two KMs try to access the same object, then the priorities are achieved by the strategy KM, and the scheduling information is stored in CORDBB.

DICE: Trace of Task 1 for Construction Automation

A trace of events for task 1 for design and construction is shown in figure 11 and is outlined as follows: (1) A preliminary design of a building (in the form of objects), which includes loading details and designer's intentions in making certain decisions is posted on SBB partition by the conceptual designer (developed using CONGEN). (2) Let the connection details of a particular joint be represented by the connection object. The connection designer sends a message with details of connections and any assumptions made during the design. (3) TMS checks to see whether earlier assumptions made by the conceptual designer are violated. (4) Associated with the connection object are methods, which indicate the possible KMs that can modify the object. Assume that the fabricator KM is one of them. A message is sent to the fabricator KM to find out whether the connection can be fabricated in the field. (5) The connection designer is notified if any problems are anticipated. (6) Sometimes two or more KMs might want to modify or access a particular object in the SBB partition. This information is stored in the CORDBB partition and is used by the control mechanism.

Implementation Details

During the initial stages, our major focus was the development of (1) utilities for defining the SBB object hierarchy; (2) transactions for posting, modifying, and deleting information in the blackboard; (3) a simulation program to demonstrate the utility of DICE; and (4) a prototype that involves the automatic generation of construction schedules from an architectural drawing. The DICE prototype was implemented on a network of SUN computers in Parmenides/Frulekit. Our current research is addressed at scaling up DICE so that it can be used in the industry.

Contact: D. Sriram and R. Logcher.

DESIGN-KIT: A Knowledge-Based Environment for Process Engineering

A computer-aided process engineering software environment should allow designers to move consistently among the following engineering tasks: (1) conceptual design of processing schemes and evaluation of alternative chemistries, mass, and energy allocations; (2) simulation, economic, and operability analysis of generated process designs; (3) completion of the design by sizing and costing all major equipment; (4) identification of control loop configurations; (5) generation of design operating procedures for start-up, shutdown, and alternative levels of production; and (6) generation of piping and instrumentation diagrams, fabrication isometrics, mechanical details of machinery, structures, and so on.

Each of these tasks is composed of a structure of subtasks, requires different information (qualitative or quantitative) at various levels of detail, and generates information that might be prerequisite for the execution of another task and might involve the user's participation at various levels of interaction.

A programming environment—DESIGN-KIT—that addresses these issues was developed in the Laboratory for Intelligent Systems in Process Engineering (LISPE) in the Department of Chemical Engineering. The structure of DESIGN-KIT and some of the applications that were developed using DESIGN-KIT are presented in the following subsections.

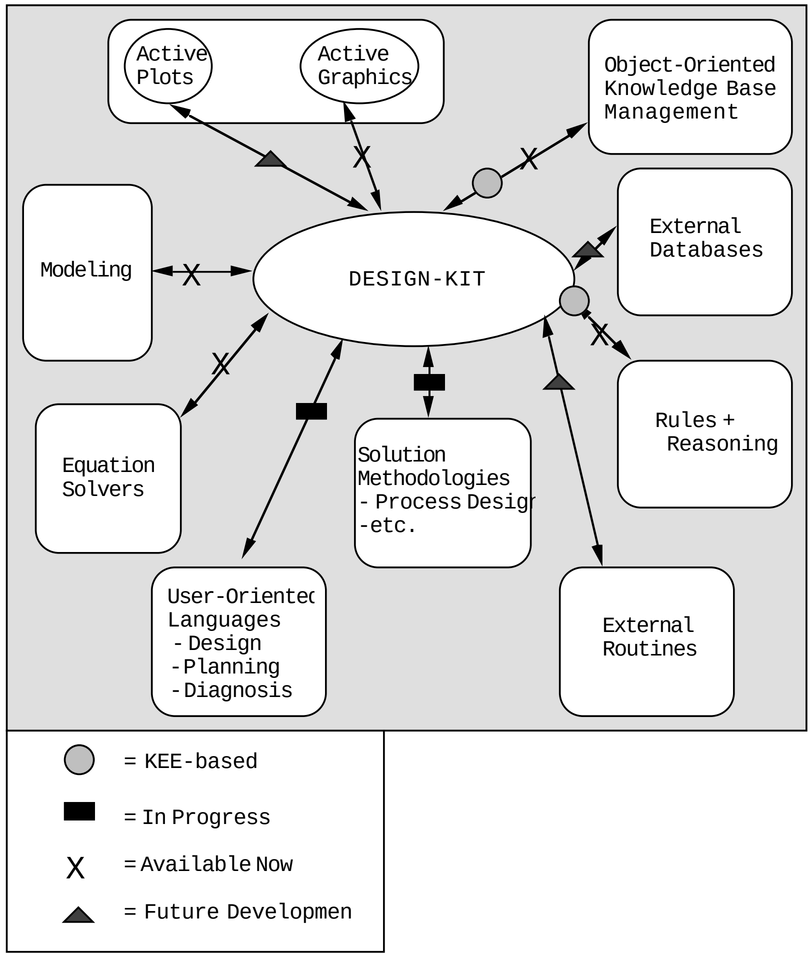


Figure 12. Structure of DESIGN-KIT.

DESIGN-KIT is envisioned as the next generation computer aided engineering environment, with full integration of knowledge-based systems, algorithmic packages, and graphic interfaces.

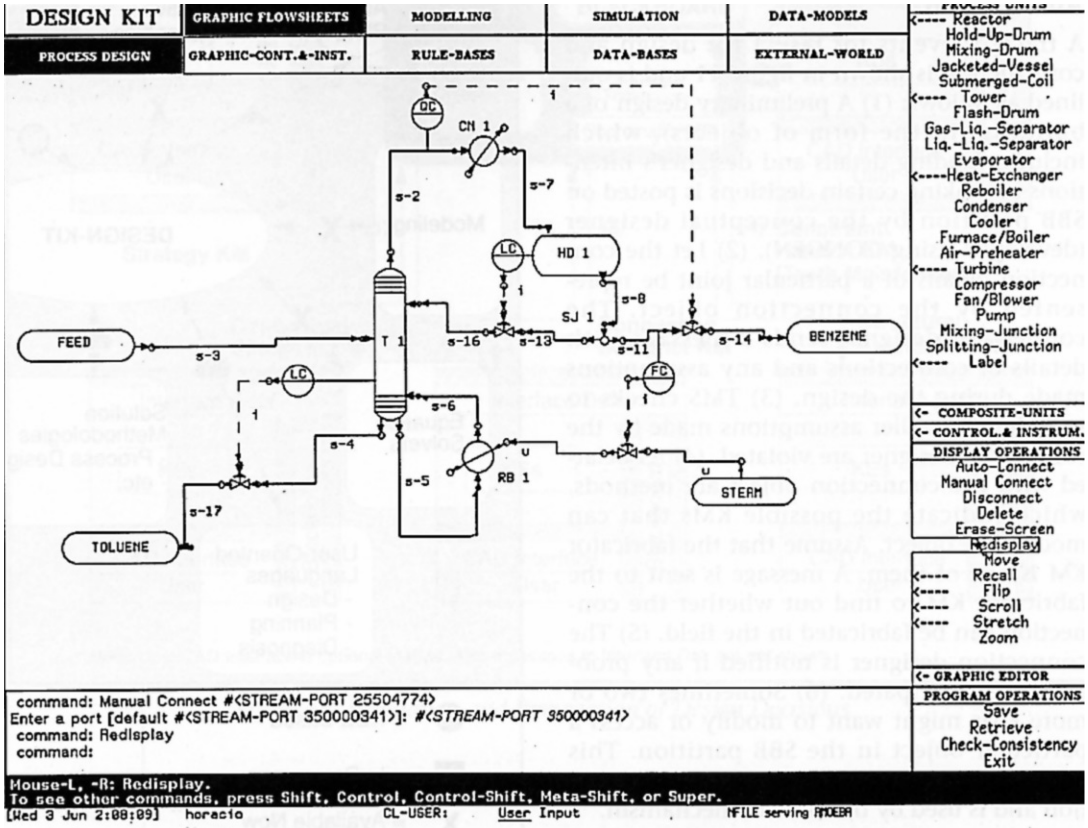


Figure 13. Graphic Interface of the Interactive Synthesis and Analysis of Process Flowsheets within DESIGN-KIT.

Structure of DESIGN-KIT

DESIGN-KIT is envisioned as the next generation computer-aided engineering environment, with full integration of knowledge-based systems, algorithmic packages, and graphic interfaces. DESIGN-KIT is implemented in Common Lisp and KEE™ on a Symbolics machine. The various components (figure 12) of DESIGN-KIT are briefly described in the following paragraphs; further details are provided in Stephanopoulos et al. (1987).

Graphics and Graphic Interfaces. DESIGN-KIT contains three interfaces: (1) the multiwindow interfaces, (2) the graphic objects, and (3) KEE's graphic interface.

Multiwindow Interfaces. These interfaces were constructed using generic objects available in the Symbolics Common Lisp programming environment. For example, figure 13 shows a five-window arrangement: (1) In DESIGN-PANE, the graphic composition of new or the rearrangement of old process flowsheets, control loops, process and instrumentations diagrams, and mechanical equipment configurations takes place, interactively or automatically. (2) SELECT EQUIPMENT contains lists of processing units, mechanical equipment, or components of control loops. (3) DISPLAY OPERATIONS contains commands operating on the graphic objects of the main DESIGN PANE window, that is, connect, disconnect, delete, move object, and so on. (4) PROGRAM OPERATIONS contains commands to retrieve an old process or save the current process or control loop design, exit, and so on. (5) LISP LISTENER allows the user to access the Common Lisp facilities to debug, edit, compile, evaluate, and so on, Lisp code.

Ephemeral windows appear in the graphic interface to support the interactive characterization of processing units, control loop components, and so on.

Graphic Objects. The construction of icons to represent specific processing units, control loop components, or operational paths is based on a set of class objects. The objects are mouse sensitive and can be created, deleted,

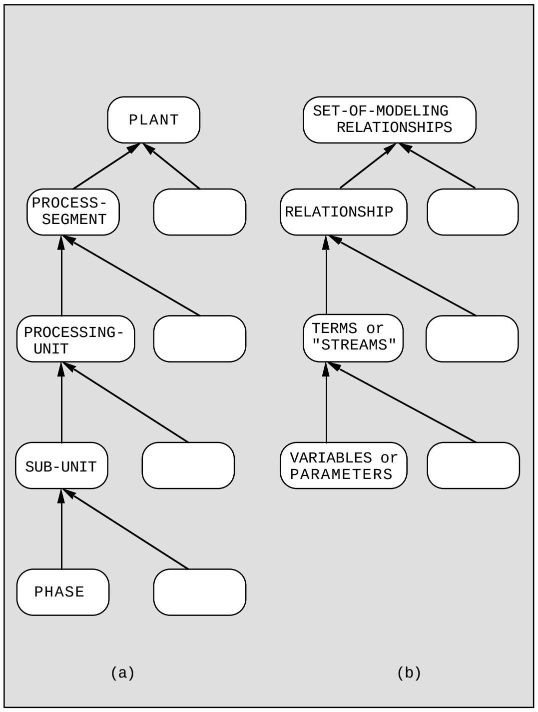


Figure 14. The Dual Hierarchy Used for Representation of Processing Systems.

modified, or rearranged through simple mouse and menu operations.

KEE's Graphic Interface. The facilities of KEE provide additional graphic interfaces. Within the scope of DESIGN-KIT, these facilities are used to (1) display the structure of the data models in a given design or the attributes of specific object; (2) generate new classes of objects or modify the attributes of existing ones; and (3) articulate production rules, and so on.

Object-Oriented Database: Data Models. The graphic images of processing units, complete flowsheets, control loops, or operational strategies are directly connected with data models in an object-oriented database. These data models are structured so that they contain information and knowledge describing what it is we might know about an object, what we would like to automatically infer about an object, and how to infer it. The mechanism of multiple inheritance and KEE's knowledge organization is used throughout DESIGN-KIT to construct the data models for various objects.

Hierarchical Modeling of Processing Systems. The representation of processing systems is achieved through a dual hierarchical

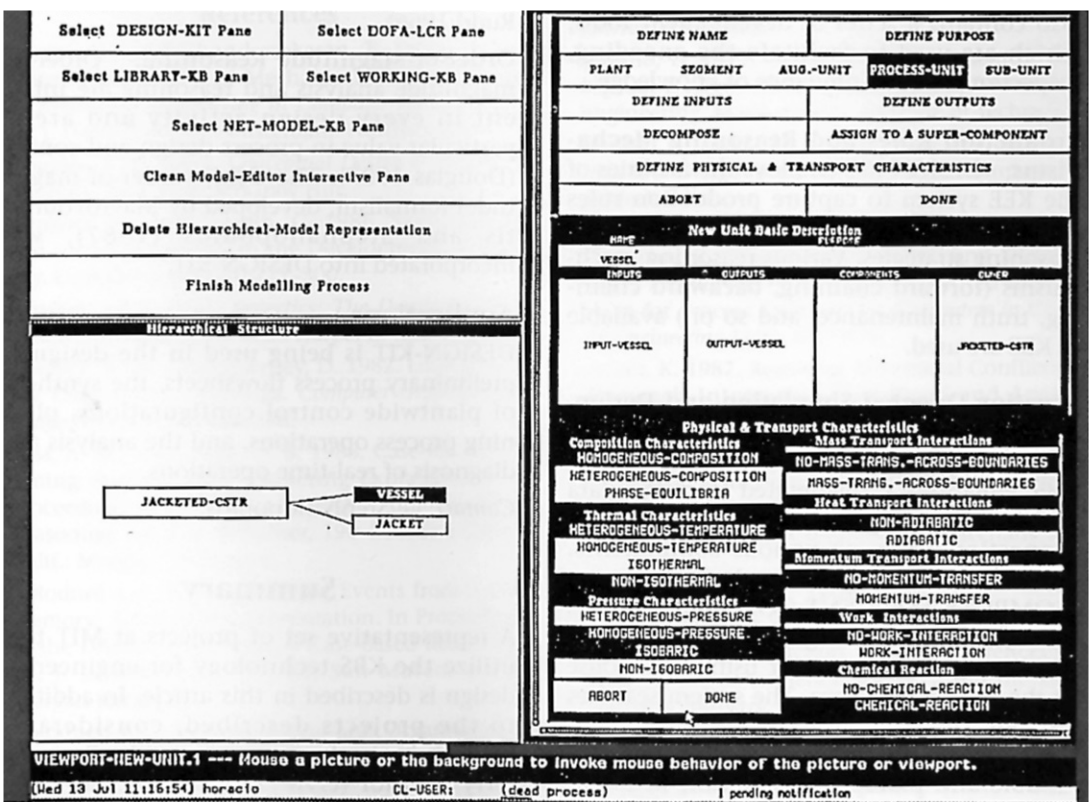


Figure 15. The Interface with the Model Editor.

Failures, such as the 1981 Kansas City Hyatt Regency collapse where two skywalks in the lobby of the hotel collapsed, occur more often then they should. Overcoming these problems requires significant changes to the design process, together with superior computer-integrated design and construction-manufacturing (CIDCAM) tools.

structure depicting the various levels of abstraction of processing systems (figure 14a) and the relationships of the various modeling components (figure 14b). Such a system allows multiple, coexisting levels of abstraction for the various processing entities; consistency of models at any level of detail; and conflict-free specification of design or operational constraints at any level of the modeling components.

The development of process models is supported by a model editor (figure 15), which knows how to construct the modeling relationships using principles from chemical engineering science. Specific relationships can also be entered by the human designer. It also contains a series of development tools, which are used to facilitate the encoding, inspection, and maintenance of knowledge.

Production Rules and Reasoning Mechanisms. DESIGN-KIT employs the facilities of the KEE system to capture production rules and execute them within the scope of specific reasoning strategies. Various reasoning mechanisms (forward chaining, backward chaining, truth maintenance, and so on) available in KEE are used.

Equation-Oriented Simulation and Design. A rudimentary symbolic equation solver was implemented as a part of DESIGN-KIT. Symbolic equations are generated from the data models (describing various components) and connections between various components. Consider the following example: The message (COMPUTE-EFFLUENT-COMPOSITION) is sent by the user to the graphic icon of a continuous stirred tank reactor using the mouse on the graphic interface. The system searches through the data model of the reactor and identifies the equation containing the desired variable and places this equation in a list. Subsequently, it examines whether other variables and parameters in the selected equation have their values specified. If not, it attempts to specify additional equations containing the unknown variables and inserts them in the list of equations. This expansion terminates at a user-specified process boundary or when no variables are left unspecified. Once the set of symbolic equations for solving a particular problem has been compiled, special methods simplify the representation by weeding out redundant variables. Subsequently, the incidence matrix is formed with the associated lists of variables and equations. The Lee-Christensen-Rudd algorithm is invoked for the selection of the design variables that renders the simplest set of simultaneous equations to be solved (Lee, Christensen, and Rudd 1966).

Order-of-Magnitude Reasoning. Order-of-magnitude analysis and reasoning are inherent in every design activity and are of particular value in process design and control (Douglas 1987). The O[M] (order-of-magnitude) formalism, developed by Mavrovouniotis and Stephanopoulos (1987), was incorporated into DESIGN-KIT.

Applications

DESIGN-KIT is being used in the design of preliminary process flowsheets, the synthesis of plantwide control configurations, planning process operations, and the analysis and diagnosis of real-time operations.

Contact: G. Stephanopoulos.

Summary

A representative set of projects at MIT that utilize the KBS technology for engineering design is described in this article. In addition to the projects described, considerable research is also being conducted by Professor Jerry Connor (civil engineering), Professor Steven Kim (Laboratory for Manufacturing Productivity), Professor Warren Seering (mechanical engineering), and Professor Karl Ulrich (School of Management). Significant contributions are being made, and it is hoped that advanced design automation tools will be available to the designer in the near future.

A number of projects with similar scope are also being pursued in other research institutions. A forthcoming book entitled *Artificial Intelligence in Engineering Design*, edited by C. Tong and D. Sriram will contain papers describing some of these projects. Other sources for articles on AI in design are a recent book entitled *Expert Systems for Engineering Design*, edited by M. Rychener and published by Academic Press, and several books edited by J. Gero and published by Elsevier Science, North-Holland. ■

Acknowledgments

Portions of this work appeared in some of the references cited here. The work on constraint management was partially supported by Control Data Corporation. The DICE prototype was supported by the Army Research Office. Current additions to DICE are being supported by the Industrial Affiliate Program of the Intelligent Engineering Systems Laboratory. DESIGN-KIT was partially supported by a grant from the National Science Foundation.

References

Brown, D., and Chandrasekaran, B. 1985. Expert Systems for a Class of Mechanical Design Activity. In *Knowledge Engineering in Computer-Aided Design*, ed. J. Gero, 259-290. Amsterdam: North-Holland.

Douglas, J. M. 1987. *Conceptual Design of Chemical Processes*. New York: McGraw Hill.

Gentner, D., and Toupin, C. 1986. Systematicity and Surface Similarity in the Development of Analogy. *Cognitive Science* 10:277–300.

Gordon, W. J. 1961. *Synectics: The Development of Creative Capacity*. New York: Harper & Row.

Haralick, R. M., and Queeney, D. 1982. Understanding Engineering Drawings. *Computer Graphics and Image Processing* 20:244–258.

Kass, A. M., and Leake, D. B. 1988. Case-Based Reasoning Applied to Constructing Explanations. In Proceedings of the DARPA Workshop on Case-Based Reasoning, ed. J. L. Kolodner, 190–208. San Mateo, Calif.: Morgan Kaufmann.

Kolodner, J. L. 1988. Retrieving Events from a Case Memory: A Parallel Implementation. In Proceedings of the DARPA Workshop on Case-Based Reasoning, ed. J. L. Kolodner, 94–103. San Mateo, Calif.: Morgan Kaufmann.

Kolodner, J. L. 1980. Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model. Ph.D. diss., Yale Univ.

Lee, W.; Christensen, J.; and Rudd, D. 1966. Design Variable Selection to Simplify Process Calculations. *American Institute of Chemical Engineers* 12:1104.

Mackworth, A. K. 1977. Consistency in Networks of Relations. *Artificial Intelligence* 8:99–118.

Mavrovouniotis, M., and Stephanopoulos, G. 1987. Reasoning with Order of Magnitude and Approximate Relations. In Proceedings of the Sixth National Conference on Artificial Intelligence, 626–630. Menlo Park, Calif.: American Association for Artificial Intelligence.

Mittal, S.; Dym, C.; and Morjaria, M. 1985. PRIDE: An Expert System for the Design of Paper Handling Systems. In *Applications of Knowledge-Based Systems to Engineering Analysis and Design*, ed. C. Dym, 99–116. New York: American Society of Mechanical Engineers.

Navinchandra, D. Forthcoming. *Exploration and Innovation in Design*. Berlin: Springer-Verlag.

Navinchandra, D. 1988. Case-Based Reasoning in CYCLOPS, A Design Problem Solver. In Proceedings of the DARPA Workshop on Case-Based Reasoning, ed. J. L. Kolodner, 286–301. San Mateo, Calif.: Morgan Kaufmann.

Navinchandra D., and Marks, D. H. 1986. Design Exploration through Constraint Relaxation. In *Expert Systems in Computer-Aided Design*, ed. J. Gero, 481–510. Amsterdam: Elsevier Science.

Osborn, A. F. 1953. *Applied Imagination*. New York: Scribner's.

Schank, R. C. 1988. *The Creative Attitude: Learning to Ask and Answer the Right Questions*. Hillsdale, N.J.: Lawrence Erlbaum.

Schank, R. C. 1986. *Explanation Patterns: Understanding Mechanically and Creatively*. Hillsdale, N.J.: Lawrence Erlbaum.

Serrano, D. 1977. Constraint Management in Conceptual Design. Ph.D. diss., Dept. of Mechanical Engineering, Massachusetts Institute of Technology.

Sriram, D. 1987. *Knowledge-Based Approaches for Structural Design*. Southampton, U.K.: Computational Mechanics

Stephanopoulos, G.,; Johnston, J.; Kriticos, T.; Lakshmanan, R.; Mavrovouniotis, M.; and Siletti, C. 1987. DESIGN-KIT: An Object-Oriented Environment for Process Engineering. *Computers in Chemical Engineering* 11(6): 655–674.

Sycara, K. 1987. Resolving Adversarial Conflicts: An Approach Integrating Case-Based and Analytic Methods. Ph.D. diss., Georgia Institute of Technology.

Sycara, K., and Navinchandra, D. 1989. Integrating Case-Based Reasoning and Qualitative Reasoning in Design. In *AI in Design*, ed. J. Gero, 231-250. Southampton, U.K.: Computational Mechanics.

Notes

1. Manufacturing in the civil engineering industry is known as construction. Several differences exist between the construction industry and the manufacturing industry. For example, in manufacturing, several hundreds of a single type of product are produced, whereas construction involves the production of one-of-a-kind products. However, the overall engineering process is similar. In this article, the terms manufacturing and construction denote the realization or creation of a designed artifact.

2. A large-scale effort with similar objectives is also being supported by the Defense Advanced Research Projects Agency in the form of an industry-university team, with West Virginia University and General Electric playing a major role.

Duvvuru Sriram is an assistant professor of Civil Engineering and the co-technical director of the Intelligent Engineering Systems Laboratory at MIT. He is currently working in the areas of knowledge-based expert systems, object-oriented databases for engineering, and cognitive studies of engineering in problem-solving. He served as the technical chairman of the first and second international conferences on AI in Engineering and was a founding co-editor of the International Journal for Artificial Intelligence in Engineering. In 1989, he was awarded a Presidential Young Investigator award from the National Science Foundation.

George Stephanopoulos is the J.R. Mares professor of chemical engineering at MIT. He is the author of two books: *Chemical Process Control; An Introduction to Theory and Practice*, and *Synthesizing Networks of Heat Exchangers*. Stephanopoulos has been a Dreyfus Scholar and was awarded the Colburn Award of AIChE in 1982, and the C. McGraw Research Award of ASEE in 1986. His research interests are in the area of process systems engineering, product and process design, and process operations and control.

Robert Logcher is a professor of Civil Engineering and the co-technical director of the Intelligent Engineering Systems Laboratory in the Department of Civil Engineering at MIT. Currently, he is working in the areas of knowledge-based expert systems, object-oriented databases for engineering, and applied natural language processing. He has developed numerous software tools for the AEC industry, including ICES STRESS, STRUDL, knowledge-based cost estimating, scheduling, and a number of other CAD systems.

David C. Gossard is a professor in Design, Systems, and Controls Division of the Mechanical Engineering Department at MIT. He is the director of the CAD laboratory where he conducts research on advanced CAE systems for mechanical design. His current research interests include computational methods for conceptual design, manufacturing advisory systems, and engineering applications of neural networks.

Nicolas Groleau is a doctoral candidate in Computer Science and Civil Engineering at MIT. His research interests include blackboard architectures for engineering design and the use of knowledge-based systems for spaceborne scientific experiments.

David Serrano is an assistant professor of Mechanical Engineering, University of Puerto Rico, Mayaguez, Puerto Rico. His research interests include the development of constraint-based frameworks for engineering design. His work on constraint management has been widely utilized in many commercial systems developed for engineering design.

Dundee Navinchandra is a research associate at the Robotics Institute, Carnegie-Mellon University where he is also an adjunct faculty member of the Department of Civil Engineering. His research interests include knowledge representation, design theories, project management, and the application of AI to engineering problems and has several publications in these areas. Navinchandra was involved in the inception of the Intelligent Engineering Systems Laboratory at MIT.