Computational Models for Form-Function Synthesis in Engineering Design

U. Flemming, J. Adams, C. Carlson, R. Coyne, S. Fenves, S. Finger, R. Ganeshan
J. Garrett, A. Gupta, Y. Reich, D. Siewiorek, R. Sturges, D. Thomas, R. Woodbury

EDRC 48-25-92

Computational Models for Form-Function Synthesis in Engineering Design

U. Flemming, J. Adams, C. Carlson, R. Coyne, S. Fenves, S. Finger, R. Ganeshan
J. Garrett, A. Gupta, Y. Reich, D. Siewiorek, R. Sturges, D. Thomas, R. Woodbury

Engineering Design Research Center
Carnegie Mellon University
Pittsburgh, PA 15213

March 1992

Abstract. This report introduces the problem of form/function synthesis for designed artefacts that are intended to perform a specified function and consist of physical parts assembled in space. The report briefly reviews how this problem is perceived in selected engineering disciplines, including architectural design, and summarizes the resulting challenges. It characterizes the computational models underlying selected EDRC projects that address these challenges according to a uniform, domain-independent format. It classifies these models into generic strategies and attempts to link these strategies to appropriate types of design problems.

This work has been supported by the Engineering Design Research Center, a NSF Engineering Research Center.

Table of Contents

1 Introduction

This report focuses on the synthesis of engineered artefacts composed of *parts* or *components* that are assembled in space. When we speak about the *form* of an such an artefact, we refer to the sum of its physical properties, from the materials out of which its components are built to their geometric shapes and the spatial relations that exist between components or a component and the whole. The *function* of an artefact denotes its intended behavior as well as its actual performance in a concrete environment and includes intended or unintended side-effects caused by its use (or mere existence).

In this report, we use the terms *form variable* and *physical characteristic* to describe any individual aspect of an artefact's form that is variable, in principle at least, and thus represents a choice to the designer. These variables are also called *design* or *independent variables* in the literature. We use the terms *function variable* and *functional characteristic* to refer to any aspect of an artefact's function, purpose, performance or behavior that the artefact may or may not accommodate. These variables are also called *performance* or *dependent variables* in the literature.

Engineers tend to use the words "function" and "behavior" interchangeably, and the present report makes no exception. It is, however, interesting to note that qualitative physicists, for example, make a distinction between these words; that is, the design's function is what it is used for, while its behavior is what it does. For example, the function of a clock is to display the time, but its behavior might be the rotation of hands or the electronic display of digits. Similarly, a motor may be designed to function as a prime mover, but may also function as a door stop because it has additional behaviors due to its mass.

We make the observation that initial specifications for designs are usually of two types: Some specifications describe at an abstract level the intended function or desired behavior of the overall system, while others describe restrictions or requirements on the form of the final design. The former are usually called *functional* or *behavioral* and the latter *physical* requirements or specifications. For example, the requirements for a vibration absorber might include the behavior of the device in terms of frequency and rejection ratio and might also specify physical properties such as allowable size and weight. Taken together, the physical and functional specifications express the design objective. Physical specifications for a design may or may not be given, while at least some functional specifications must be given, since the functional specifications express the central aspect of the design objective.

Although functional and physical specifications are independent in the functional domain, they are coupled in the physical domain, because any physical arrangement results in a specific set of behaviors. The physical and functional characteristics of individual components depend on one another, and the behavior of the whole design depends strongly on the configuration and interaction of components. Figure 1 illustrates this for a small portion of the form and function variables that play a role in the design of a building: any form variable is likely to influence more than one function variable, and any function variable is likely to depend on more than one form variable. There are, in addition, possible dependencies between variables of the same kind. For example, placing an object in a certain location restricts the placement of other objects, or maximizing the daylight in a room may have adverse effects on heat gains during the summer or heat losses during the winter (see the example given in Section 2.1).

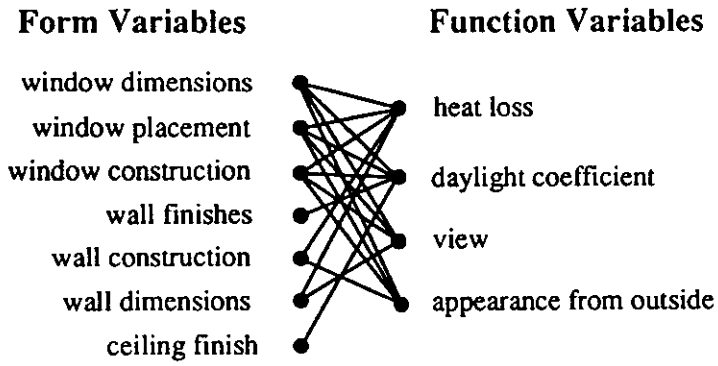

Figure 1: Interaction between form and function variables in building design

Form or function variables can thus not be considered in isolation during the design process. On the other hand, neither humans nor computers can handle *all* form and function variables and their interactions simultaneously. The challenge is to devise a synthesis process that divides the overall task into more manageable subtasks by focusing on selected variables at any time, while allowing for due consideration of interactions and dependencies.

In Section 2, we review the problem of form-function synthesis in several disciplines and compare the approaches and challenges in each discipline. Section 3 characterizes the computational models underlying selected EDRC projects that confront these challenges. Section 4 attempts to link successful computational models to generic design problems, and Section 5 identifies open issues and work for the future.

2 Background

In this section we give an overview of the salient features of form-function synthesis for the disciplines of architectural, mechanical, structural, and digital electronics design. The section concludes with a summary of the challenges that confront these disciplines.

2.1 Architectural Design

Reflections about the relation between form and function in architecture are at least as old as the earliest written treatise to survive from antiquity, Vitruvius' *de Architectura* from the first century A.D. The debate continues to the present day; it cannot be reviewed here in any depth, and we restrict ourselves to a brief characterization of major positions.

At one extreme are the "functionalists" who adhere to Louis Sullivan's famous dictum that "form follows function" [1]. Probably the most radical exponent of this school was Hannes Meyer, who led the Bauhaus from 1928 to 1930. He greatly admired works produced by engineers, which he characterized as "function times economy". He declared that "building is a technical not an aesthetic process" and that "the function diagram and economic program are the main guiding principles in a building scheme" [2].

This type of "functional determinism" has come under attack by architects and theoreticians who view it as reductionist [3] or fallacious ([4], pp. 186-208). Their arguments cannot be reviewed here in detail. But it is important for the following discussion to point out that a narrow-minded functional determinism does not hold up under its own premises. A closer inspection of the function of a building reveals that the criteria by which it is assessed are extremely varied; they may conflict with each other because they compete for the same limited resources (area, costs), or contradict each other for technical or other reasons such as conflicting interests of client, users or the public at large. This situation has been demonstrated by Radford and Gero [5], who show that for the design of a window in a specific situation, daylight performance and summer thermal performance conflict in the sense that improving one criterion might lead to a point where further improvements can only be achieved by diminishing the other criterion. The designs that have reached such a point form a Pareto-optimal set or "frontier" of alternatives each of which exhibits a particular tradeoff between the two criteria (see Figure 2). None of these alternatives is *prima facie* optimal; that is, a decision is by no means as automatic as the radical functionalists assumed and must rely on value judgments.

Furthermore, it is generally difficult to start from a preconceived combination of functional characteristics because it is often not even clear at the outset what *types* of components may be involved in the design of a building. In the Kimball Art Museum in Fort Worth, Texas (designed by Louis Kahn), for example, the roof structure consists of a series of beams with a curved cross section, two of which can be combined to form a vault that lets light in from the top and diffuses it in combination with a suspended screen, which also diffuses sound. Thus we have two components that perform structural, lighting and acoustic functions aside from playing a major role in the overall architectural composition. But we can imagine alternative solutions in which the same functions are performed by different types, even different *numbers* of components; e.g. the function of light diffusion may be carried out by skylights that are not part of the roof structure. Thus, mappings from function to form are generally not unique, and a straight-forward functional determinism appears not only undesirable, but actually impractical.

The other extreme in the form/function debate is occupied by architects for whom the form of a building is its only interesting aspect and who "build against function", as the architect Philip Johnson put it. This

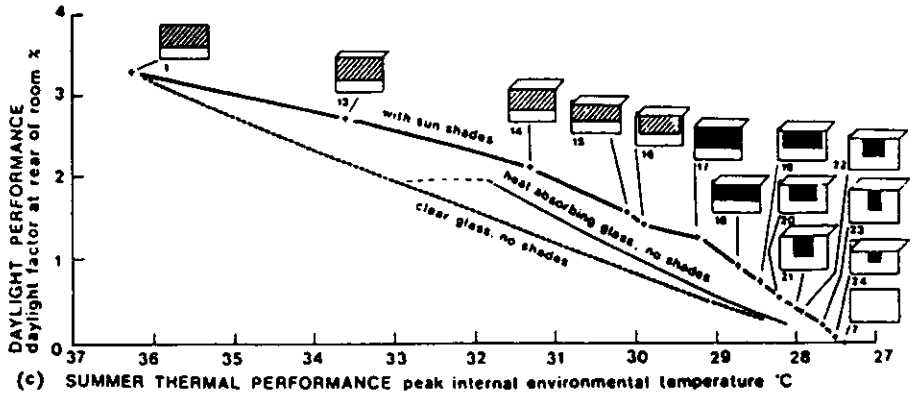

Figure 2: Tradeoffs between alternative window designs (after [5])

position may appear incomprehensible to observers outside the circles in which these debates take place; but the very fact that it can be formulated at all points out that the relation between a building and its function may be rather loose indeed. This is demonstrated by the high degree of adaptability to varying uses that buildings have shown over time. It also explains why one can enjoy the buildings in a foreign city without knowing for which purpose they were built.[1]

There have also been attempts to reconcile the two aspects in a programmatic way. An example is the requirement established in the 19th century by various proponents of the "Picturesque" esthetic (introduced into the United States by the landscape architect A. J. Downing) that the form of a building "express its purpose." However, this is not the same as saying that it be functional and may result in rather dysfunctional buildings. An example is the well-known Engineering Building at Leicester University in England, which separates from each other and expresses clearly its main functional components (lecture halls, offices, labs and shops). The result is a substantial increase in the exterior surface area and complicated joints, which cause severe maintenance and operating problems.

It may be surprising that a field as old as architecture has not solved what appears to be one of its most central problems. But one has to realize that the problem is of rather recent origin. Traditionally architects had to deal with only a small number or well-understood building types, worked closely with their clients and employed slowly evolving craft techniques. Within such a well-established context, a form/function dichotomy did not arise and indeed was not discussed in the literature. All this changed with the industrial revolution, which introduced a host of new building functions and construction technologies. Some of the uncertainties created in this way, particularly those that were connected to the performance of the physical structure and systems of a building, could be dealt with in a rational, scientific way and became the domain of the emerging engineers. The extreme positions outlined above can be viewed as attempts to relocate architects in this changed world: the functionalists essentially by blurring the distinction between architects and engineers and the formalists by making it as wide as possible.

---

[1]One should keep in mind that the statements quoted here must be seen in their respective contexts, namely as polemics: the functionalists attacked the academic tradition which proved unable or unwilling to deal with new techniques or building types, and the critics of functionalism rebelled against the drabness that can result when architecture is reduced to "function times economy", which became epitomized by the speculative office building in the form of a faceless glass box.

No generally accepted position has emerged from this debate. Practitioners have to define their own attitude. That the form/function dichotomy can be resolved without sacrificing one aspect for the other is demonstrated not only by Sullivan's buildings, but also by recent examples like the Kimball Art Museum mentioned above, which performs well as a museum and is a memorable piece of architecture.

Attempts to systematize and formalize architectural design in connection with computer-aided design tend to favor some form of generate-and-test; that is, they assume that a design must be specified to a certain level of completeness before it can be evaluated. This solves some of the problems created by the complex interactions between form and function variables in architecture that were described above: It is generally easier to evaluate a design according to multiple criteria and to *discover* conflicts than to develop a design directly from multiple and conflicting criteria. In Stiny's elegant (but rather abstract) formalization, two descriptions of the evolving design are constructed and maintained in parallel, one describing the form of the design (and constructed by a shape grammar) and one that extracts properties of the design and describes it "in terms pertaining to, for example, purpose, function, and use, meaning, type or form" [6]; the latter can thus form the basis for analysis and evaluation. The LOOS/ABLOOS systems described in Section 3.10 implement this idea for a practically relevant problem domain.

A more general treatment of form/function issues in architecture based on a generate-and-test approach can be found in [7]. This and similar treatments envisage an iterative process that cycles through synthesis and analysis steps and explores the interactions between form and function variables in the concrete context of the given design problem. That is, design is essentially viewed as *search* through a space of possibilities. At this level of generality, this approach is indistinguishable from search-based approaches developed in some engineering disciplines. The more innovative methodological contributions made by architects to this evolving approach are not in the area of analysis, which has to rely heavily on engineering techniques, but in the area of synthesis; architects in particular have been at the forefront of experiments with *generative grammars*, rule-based mechanisms that are able to model constructive reasoning and highly context-dependent decisions especially about the geometry of designs. The GENESIS, GRAMMATICA and LOOS systems described in Sections 3.3 and 3.10 demonstrate this approach. (See [8] for a more general treatment.)

An example from practice where alternatives have been deliberately generated and evaluated in terms of tradeoffs is the Hooker Office Building in Niagara Falls, NY. The architects (the Cannon Partnership) developed three alternative facade systems, two conventional and one innovative, and evaluated them in terms of construction and operating costs [9]. But this example is by no means representative for current practice.

The need for multiple-criteria performance evaluations in architectural design implies that only weak formal methods are available for comparing designs. This explains why classical optimization techniques play only a minor role in this field: a multitude of (possibly conflicting) criteria makes it difficult to arrive at a defensible objective function for the overall design that could be used to structure the search for best solutions. But optimization may be useful in narrowly circumscribed subproblems (see [10] for examples). In general, building design is a satisficing endeavor; that is, architects and their clients must be content with solutions that represent an acceptable overall mix of advantages and disadvantages.

But even if inferences from function to form are generally difficult in architectural design, certain research directions appear underexplored. Not every design problem requires the invention or novel assembly of components. For certain recurring combinations of functional requirements, prototypical component configurations are known. They could be retrieved from a properly constructed database and

parametrically adjusted in accordance with a specific situation, including the systematic variation of form variables *via the variation of function variables* [11].

Any discussion of form/function synthesis in architectural design, however brief, should mention Alexander's *Notes on the Synthesis of Form* (which should more appropriately be called *Notes on the Analysis of Function* according to G. Stiny) because of its broad recognition which extends into related disciplines. Alexander suggests in this book a process that groups functional requirements (expressed as "misfits") hierarchically according to the degree to which they have to be considered *together* in a solution [12]. This approach has been criticized from various points of view (see for example [4]) and by Alexander himself. He replaced his earlier approach by a *pattern language* [13], which has recently been formalized and reviewed as a possibility to capture domain knowledge and solution prototypes in architectural CAD [14].

Architects who are willing to incorporate aspects of rational decision making into their design process and to explore carefully and systematically form/function relations in the context of a given problem need an integrated decision support environment that puts various form generation, analysis and simulation tools at their disposal and aids in the rapid specification and evaluation of design alternatives, particularly at early design stages. Such an environment is not available at the present time, nor is a decision-making process in place that would enable the various disciplines and actors involved in the design of a building to coordinate their efforts.

2.2 Mechanical Engineering

The designer of mechanical parts often moves directly from a statement of functional requirements to sketches of the artefact that will meet those requirements. Experienced designers can answer the question: "Given the functional requirements, what should the artefact look like?" Indeed, most mechanical design CAD systems are based on the assumption that the design proceeds from geometry. That is, the CAD system enables the designer to create and manipulate the geometry of a design and then to analyze it for the required behaviors.

During the process of creating a mechanical part, a designer transforms an abstract functional description for a device into a physical description that satisfies the functional requirements. In this sense, design is a transformation from the functional domain to the physical domain; however, the basis for selecting appropriate transformations and methods for accomplishing transformations is not well understood. Design transformations in circuits [15], software [16], and some architectural applications [17] lead to a degree and type of modularity not well suited to mechanical devices [18].

Good mechanical designs are often composed of highly-integrated, tightly-coupled components, where the interactions among the components are essential to the behavior and economic execution of the design. This assertion runs counter to design methodologies in engineering fields such as software design and circuit design that result in designs in which each component fulfills a single function with minimal interaction. Because of the geometry, weight, and cost of mechanical components, converting a single behavioral requirement into a single component is often both impractical and infeasible. Each component may contribute to several required behaviors, and a single required behavior may involve many components. In fact, most mechanical components perform not only the desired behavior, but also many additional, unintended behaviors. In good mechanical designs, these additional behaviors are often

exploited.[2] By creating a formal description of a limited set of behaviors for mechanical designs and a corresponding description of physical components, designers can generate the description of a physical system that takes advantage of the multiple behaviors of its components.

Because of the difficulties of representing behaviors and their interdependencies, most mechanical designers begin by generating the geometric form of the artefact and then analyzing, for example, its structural, thermal, and kinematic behaviors. This observation can be verified by examining the available mechanical CAD packages. In all of them, the designer synthesizes the artefact by interactively creating the geometry and then invokes analysis tools to check whether the artefact meets its requirements; if it does not, the designer changes the geometry and repeats the cycle.

Finding useful representations for function and behavior for mechanical artefacts is difficult because of the coupling of behavior and form in mechanical parts. Only a few domains within mechanical engineering, such as kinematics, have standard representations that are used in design synthesis. For example, in the transformational system presented in Section 3.5, we begin with a bond graph representation for the behavioral requirements of a gear train. The bond graph representation captures only the requirements for power flow; other requirements must be stated in terms of constraints. Many of the requirements can be tested only after the gear train has been synthesized. Using analysis, requirements such as maximum total mass and volume can then be verified.

2.3 Structural Engineering

The role of structural engineering is, typically, to take the overall form of an artefact determined by other considerations and elaborate it into the detailed form of the structural configuration and its components so as to satisfy functional requirements of strength, safety, serviceability, etc. The overall form may be derived from formal or spatial considerations (e.g., the massing of a building or shape of an automobile), from functional considerations (e.g., the geometric layout of a crossing to be bridged dictated by traffic flow or the layout of a process plant dictated by process requirements), or a combination of the two (e.g., the layout of a ship or aircraft dictated by both the spatial layout of the cargo and the fluid flow around it).

Structural engineering becomes a creative component of the overall design process of multifunctional artefacts or systems when the determination of the overall form is affected by the incorporation of structural functionality considerations. Even when this opportunity is limited, structural design deals with the determination of the detailed form of the structural load-resisting system. Thus, the form/function "debate" occurs at two levels.

One level concerns the overall form of the structure: is it driven by spatial considerations or by its load-carrying function? An elegant example of the latter is Eiffel's design for his 300-meter tower: the slope of the legs at any level is such that the axial forces acting along the legs intersect the resultant of the wind loads above that level; thus, there is no bending anywhere, resulting in a very efficient structure. Weaker forms of function-driven form are Fazlur Khan's Hancock building in Chicago, hyperbolic cooling towers, and most long-span bridges. In most cases, however, function is subordinate to form:

---

[2]This statement does not contradict the design axioms put forth by Suh [19, 20]. The design axioms state that good designs maintain independence of functional requirements and minimize the information content of the design. Suh points out that by integrating functions into a single component, information content may be reduced without compromising the independence of functional requirements.

architects, plant designers, or highway layout engineers define the overall form, and structural engineers provide the best function they can within the spatial constraints.

The second level concerns the physical form of the structure: which function determines the form? Typically, a structural system is an assembly of discrete components, each providing multiple functionalities. The strength of a wall may be dictated by gravity and load resistance functions, and its stiffness by the drift limitation (comfort) function. That same wall may, furthermore, serve as an architectural separation, as a thermal and noise barrier, and as a conduit for electrical and mechanical utilities. At this level, structural design is closely related to mechanical and architectural design (see Sections 2.1 and 2.2), in that each component must satisfy several functional requirements, and each functional requirement is satisfied by several components. It may be argued that structural systems are more loosely coupled functionally than tightly integrated mechanical systems and that, for initial synthesis purposes, a dominant function can be selected. However, no single function decomposition can be expected to work for all cases.

Form-function synthesis in structural design is inherently a multi-level process, where each level contributes to some elaboration or refinement of the physical structure (shape, spatial relations and physical properties of components) satisfying (or optimizing) a variety of functional requirements. The components may be concrete (e.g., a beam or wall) or abstract (e.g., a 3-D frame), and the functional requirements may be intrinsic to the load-carrying function (e.g., strength and stiffness) or extrinsic to it (e.g., spatial demands or acoustic or thermal functionalities). A form-function synthesis support environment must provide for convenient navigation over the form and function variables involved. Independent and dependent variables can only be identified in specific subproblems. In the synthesis of a structural system that invariably interacts with other formal or behavioral functionalities of the overall artefact, function-to-form transformations are as likely to occur as form-to-function transformations.

2.4 Digital Electronics Design

In digital electronics design, the designed artefact is composed of an interconnection of pins on various components such as transistors, gates, macrocells, chips, or boards. The form of the artefact is considered at two distinct levels of detail: *logical form* or *structure*, which does not include geometric information, and *physical form* or *geometry*, which includes geometric information. Thus, digital electronics designers have three well-accepted perspectives of the design: its function or behavior, its structure, and its geometry. Each of the perspectives is an axis on a Y-chart that has been used by several researchers [21, 22] to classify various synthesis activities. The mapping from function to structure is called *logical design*, and the mapping from structure to geometry is called *physical design*. Each axis has several levels of abstraction as shown in Figure 3. Some form variables (e.g. board size and the set of available components) are provided as constraints to the synthesis process.

Physical design consists of two subtasks: placement and routing. In placement, geometric information (usually from an existing library) is added to the components in the structure and they are positioned on an area of specified dimensions so as to satisfy a number of (possibly conflicting) goals. These goals include increasing routability of the interconnections, reducing signal cross talk, and reducing the heat dissipation levels; usually the primary goal is routability -- minimizing the total estimated interconnection distance. In routing, pins on the placed components are properly interconnected by positioning conductor paths in the specified area subject to constraints (such as minimum width of the conductor, minimum distance between conductors, and maximum number of conducting layers). Note that the problems of

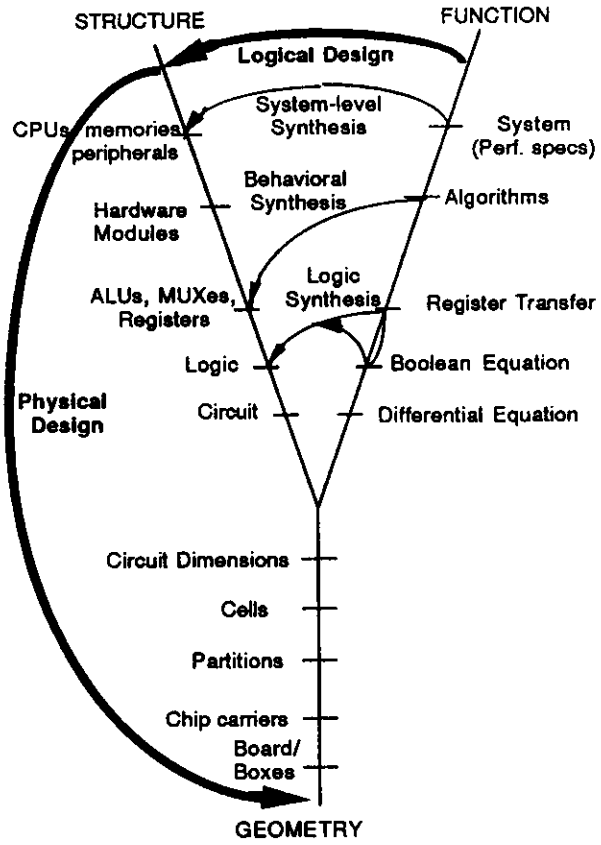

Figure 3: Levels of abstraction in digital design

placement and routing are intimately related; however, historically they have been treated separately due to the inherent computational complexity of the total problem. Physical design has been addressed for both inside an integrated circuit and on printed circuit boards. Several commercial physical design tools exist. We do not discuss physical design further in this report.

The synthesis of form from function in logical design can be classified into three levels: logic synthesis, behavioral synthesis, and system-level synthesis. In logic synthesis, a set of Boolean expressions and finite state machines is mapped into an interconnection of gates and memory devices. Generating a structure that satisfies the specified function is a relatively straightforward procedure. The difficulty is in optimizing the form for a set of criteria (e.g. least size or least time delays). Boolean algebra provides a strong theoretical basis for the optimization algorithms used in logic synthesis.

In behavioral synthesis, an algorithmic description of a portion of a computer, such as a CPU, is mapped into an interconnection of register transfer elements, such as ALUs, registers, multiplexors, and buses. Generating the form for a given behavior can be divided into subtasks. Finding feasible solutions to the subtasks is relatively simple, but the search for optimal solutions has been shown to be NP-complete. Even though the components are modular and fulfill a single function, a good design makes maximal use of each component. For example, the same component can perform the same function on different operands. Heuristic algorithms that generate near-optimal solutions have been developed. The SAW system encapsulates a set of these algorithms (see Section 3.8).

In system-level synthesis, a set of high-level functional requirements (e.g. instruction set, amount and type of input-output devices, or clock speed) is transformed into an interconnection of chips (e.g.

processors, memory, peripheral chips, transceivers, and latches). In this level, generating a feasible solution is difficult because the mapping from functional requirements to an interconnection of chips is not well understood. The MICON system (see Section 3.7) uses a knowledge-based approach in which the mapping is provided by a human trainer in an incremental fashion.

While the goals of these synthesis tasks are different, they have several common characteristics. In each, the design space is large, which rules out brute-force generate-and-test approaches. In addition, no good evaluators exist for partially designed artefacts; thus, incremental generate-and-test is not suitable. None of the steps has a single objective; rather, each pursues several competing objectives so that many pareto-optimal solutions exist. Furthermore, the design space is discrete and discontinuous due to, for example, step functions in memory sizes. Hence gradient descent (greedy) methods will not yield good results. An example design space is shown in Figure 4.

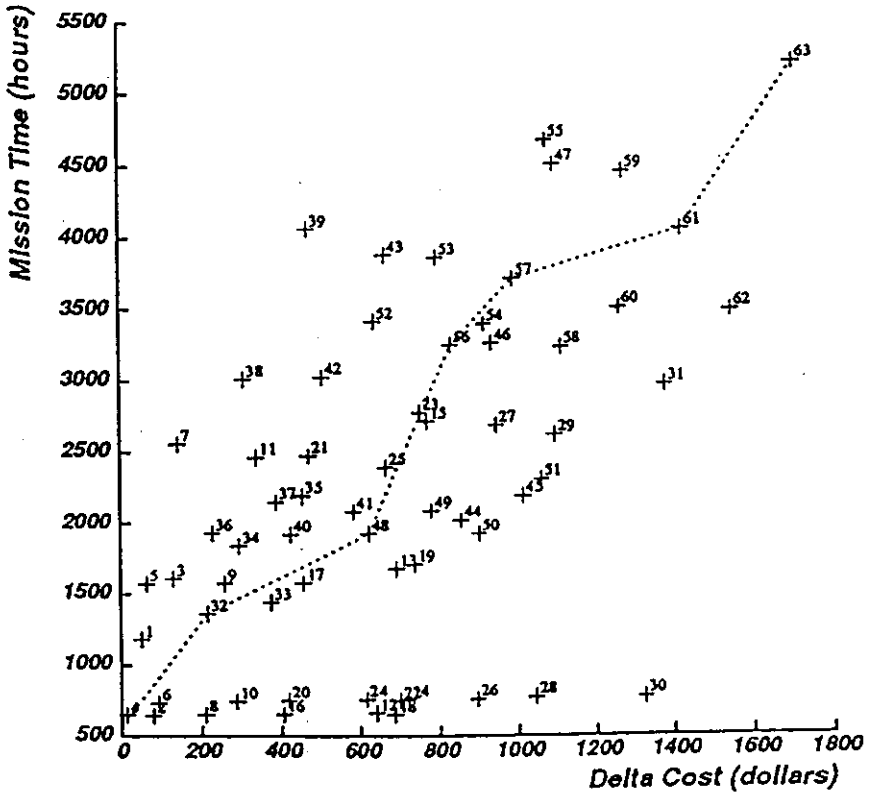

Figure 4: Discrete design space for enhancing the reliability of a computer (after [23])

Logic synthesis tools have been in use for a decade. Over the last five years, commercial versions of these tools have gained widespread acceptance. The trend toward raising the abstraction level of input from the designer has lead to commercial behavioral synthesis tools in the last couple of years. This trend can be extrapolated to the availability of commercial system-level synthesis tools in the coming years.

2.5 Summary

"Form" is a loaded term in architectural design, which makes this field unique among the disciplines included in the present section, and the role attributed to form introduces a philosophical dimension into the form/function debate that is missing in the engineering disciplines. But this is not to say that aesthetic

aspects are absent from engineered artefacts. The pioneers of the modern movement in architecture were attracted to these artefacts also, and sometimes primarily, because of their aesthetic qualities: the artefacts demonstrated in concrete ways the starkly novel effects that can be achieved when pure geometric forms derived from the function of an object are assembled in space unimpeded by conventions that demand the employment of elements from a classical vocabulary and applied ornament.

It is also worth noting in this context that the *appearance* of an object may play an important role in its design, for example, when its marketability depends on it. An obvious example is the influence of the stylist on the form of an automobile, which may override engineering concerns.

When it comes to more tangible functional aspects, the disciplines reviewed in the present section share a core of concerns that center around three related sets of issues: (1) Functional specifications normally do not map uniquely into component configurations. A possibly large space of alternatives is available, and the structure of this space is often ill-understood and precludes the employment of standard search strategies. (2) The search for alternatives is complicated in particular by the multifunctionality of components (in architectural, mechanical and structural design) or by the fact that single-function components can be used in multiple ways (in digital electronics design). This makes it difficult to concentrate at any given time on a single functional characteristic or behavior aspect and thus eliminates some obvious problem decompositions. (3) Functional characteristics may conflict with each other for various reasons. This makes it important to take tradeoffs into account, which complicates the evaluations that must be performed during synthesis and makes optimization difficult to integrate into this process.

Among the disciplines reviewed here, digital electronics design has come closest to an accepted decomposition of an overall design problem into more manageable subproblems (from behavior specification to logical and physical design). It can also rely on formal behavior specifications and well-understood transformations from behavior to form (or structure) at least below the system level. Mechanical engineering appears to occupy the other extreme: it has a particular need for developing a theoretical base for formal specifications of behavior and function/form transformations.

3 Computational Models for Form/Function Synthesis

3.1 Overview

The issues introduced in the previous section are addressed by many EDRC projects. The present section characterizes the computational models underlying selected EDRC projects.

The tasks addressed by the projects vary greatly, and an individual project may solve only part of an overall design problem. For example, layout or placement problems often assume that a functional specification and a list of components are given; that is, they rely on information that may have to be explicitly generated in prior steps. To link the computational models underlying the projects with types of general design tasks, we distinguish the following *generic (sub)tasks* that must be faced in the design of any assembly of parts:

1. *Problem specification*: the specification of the desired behaviors or functions, physical constraints, context and other factors influencing the design
2. *Component specification and logical design*: the identification of the components to be assembled in space. Only in simpler problems can this task be reduced to the selection of parts from a predetermined set. Logical design is the derivation of required connections between the selected components. In some domains, the connections are implied by the component types and do not have to be designed explicitly.
3. *Physical design*: the specification of the shape and placement of components including their physical connections
4. *Analysis and Evaluation*: the analysis of the design according to various constraints or criteria and the evaluation of the design for the desired behaviors.

In some projects, such as MICON, these tasks are followed by the construction of a physical prototype.

This list does not imply that the tasks are executed in a linear sequence. They may not even appear as separate processes in many approaches; that is, some tasks may be intimately related and solved together; for example, analysis and evaluation may be performed while components are selected or placed. Rather, the tasks indicate types of generic subproblems with common characteristics that must be solved in one way or other in the design of assemblies of parts.

The following sections indicate the generic tasks addressed by each project and describe the computational model used for each task (or combination of tasks) in the following general terms:

1. *Overall architecture and approach*: A brief overview intended to put the detailed information given later in its proper context
2. *Inputs*: The form and content of the specification that describes the design problem to be solved (which may be only part of an overall design problem; that is, the inputs may be outputs from other processes).
3. *Form and function variables*: An identification of the form and function variables directly or indirectly handled by the approach
4. *Representation of form and function variables*: The symbolic representations on which the operators below work
5. *Operations performed on the representations*: The mechanisms used to work on these representations when solving a design problem

6. *Control*: The control strategy under which the operations are executed

7. *Outputs:* The form and content of the solution description.

These terms form the basis for comparisons that suppress domain-specific details. However, the approach taken by a project may reflect domain-specific objectives and may become understandable only against this background. Each project description therefore starts with a brief summary of its overall goals and objectives. It concludes with a, possibly preliminary, evaluation of its contribution to form/function synthesis.

The project descriptions start with projects that cover, or may cover, both logical or conceptual design: Representing and Recording Design Intent, the grammar interpreters GENESIS and GRAMMATICA, the Solids Grammar Using Spatial and Functional Attributes in Structural Design, Transforming from Behavior to Structure in Mechanical Design. They continue with projects concentrating on logical or conceptual design: Conceptual Design, MICON[3], SAW and BRIDGER. The Layout Synthesis project deals with physical design.

It is important to keep in mind that the computational models described below are not meant to represent models of complete design processes. They are intended to provide a basis for the development of tools that are able to augment the capabilities of human designers, not to replace them. For example, human judgment is necessary to supply and iteratively refine the problem specification on which a computational process can be run. It may also have to make decisions between competing alternatives and their associated tradeoffs. The models and their implementations demonstrate furthermore that human judgment may be called upon for more practical reasons: to control the overall process or to supply crucial knowledge that is missing from the knowledge base available to a process.

These models therefore must be seen as a parts of a *computer-supported design environment* (CSDE), where the interaction between computational mechanisms and human designers is both well understood and effectively supported.

3.2 Representing and Recording Design Intent

3.2.1 Objectives

Our goal is to develop a model of design decision-making that explicitly identifies the designer's intent; we also aim to develop a representation for the record of the design process. We assume that the designer begins with functional and behavioral objectives and that these objectives are refined until the specifications are met in a design artefact. In addition, we make the explicit assumption that operators exist for the generation of alternative solutions which meet objectives or subobjectives. Thus, the designer does not manipulate the form of the artefact itself; rather, the designer manipulates the objectives and selects alternatives and refinements, thereby revealing the intent behind the decisions. We are developing this system for constructed facilities in the domain of civil engineering (see [24]).

Our approach is to maintain a record of the design process that starts from the highest-level objectives, including all their refinements, and leads to the final form of the artefact. A documentation of the design

[3]The MICON project itself covers the full design task, but the part described here focuses on logical design

process is useful for:

- explanation - to explain how and why a particular decision was made;
- verification - to determine if characteristics of the final design are consistent with the intended characteristics;
- modification - to predict the effect of making changes to the design; and,
- re-use - to synthesize a design from a previous design with a similar specification.

3.2.2 Generic Design Tasks

The goal of this work is to capture the intent behind the designer's decisions that transform function into form. Therefore, it is not focussed on particular design tasks, but rather on capturing the process that occurs in any tasks.

3.2.3 Computational Model

Overall Architecture and Approach. Our design model consists of the following entities: objectives, decisions, alternatives, assumptions and operators.

- *Objectives* are statements of the desired attributes of a design at different levels of abstraction that drive the design process. Examples of types of objectives are 1) functionality, 2) aesthetics, 3) economy, 4) constructibility or manufacturability, 5) maintainability (the artefact must be maintained during its service life), and 6) disposability (the artefact must be disposed of at the end of its service life). Another class of design objectives is concerned with planning the design process. For example, process objectives might be to minimize the time to complete a design and to minimize the number of design iterations. These objectives guide the decision process during design. We include constraints in our definition of objectives. Objectives may have importance associated with them.
- An *assumption* is an arbitrary requirement imposed to account for lack of information.
- An *alternative* represents a different way in which some of the objectives may be achieved in the design. For example, the sound isolation of a space in a building from a noise source may be achieved either by 1) placing the space far from the source (sound intensity reduces with distance) or, 2) using an acoustic obstruction between the space and source. Usually one of the alternatives is pursued.
- A *design decision* describes an aspect of the design artefact at some level of abstraction (e.g. material = A36). Note that we use the term decision to describe the result of the decision-making activity. In the structural design domain, the values of design decisions can be materials like A36 steel, reinforced concrete, or glued-laminated timber; points and lines; structural properties like rigid or non-rigid; load-transfer mechanisms like bending, shear or axial force; and shapes like wide-flanges, rectangular sections etc.

The design process is characterized by the following activities:

1. The activity of *focusing* determines which objectives to refine or evaluate.
2. The activity of *refining* an objective results in the generation of other objectives or a decision. The subobjectives represent conditions that must exist in the design for the original objective to be satisfied. It is possible that there are different ways (alternatives) to achieve an objective.
3. The activity of *evaluation* determines the feasibility of the alternatives with respect to the other objectives in the design.

4. The activity of *selecting* involves choosing the alternative that best satisfies a set of objectives. The selection process makes use of knowledge about the relative importance of the objectives in accordance with some criteria specified by the designer.

Inputs. A design problem is defined as a conjunction of objectives. The objectives may be incomplete or conflicting and may change over the course of the design process. We also assume that the operators necessary to transform objectives into realizable structures are given. These operators may be analytical, grammatical, heuristic or logical. While the framework we are proposing is a general framework, we are focusing on the design of constructed facilities.

Form and Function Variables. Form and function variables are represented directly or indirectly through objectives, assumptions, alternatives or design decisions.

Representation of Variables. All data (including decisions) are represented as variable bindings. A type is associated with every variable. The types are related in aggregation and specialization hierarchies. Variables are classified along two dimensions: 1) entity vs characteristic, and 2) design vs exogenous variable. An entity is a variable type that is represented as some aggregation of geometry only or both geometry and material types. A characteristic is variable type that represents a property measured over these entities. Exogenous Variables are those variables whose values are known beforehand, while the values of design variables are assigned over the course of the design process.

An objective is a relationship over variables. An objective is an instance of a class. Two kinds of classes are distinguished: 1) reduce and 2) non-reduce. The reduce class expresses the requirement to reduce the value of some design characteristic. Upon evaluation, this class of objective returns the value of the design characteristic under the current variable bindings. In case of the non-reduce class, the result of an evaluation is a yes or no answer indicating whether or not the objective is satisfied. Importance may be specified for a group of objectives either ordinally or by using weights. Assumptions are represented in the same way as objectives. Alternatives maybe a conjunction of objectives or a set of variable bindings.

The representation of operators has not been a major focus of this work because we believe that it is not very important to our goals. We have identified the forms for some operators. For example, an evaluation operator is a function (or procedure) associated with an objective that determines a result, given the current variable bindings for the variables involved.

The design state contains the set of the objectives, assumptions, variable bindings, alternatives, and operators at the current point in the design process. The record of the design process is viewed as a tree whose nodes represent the design state and whose arcs represent the decision-making activities that change the design state.

Operations. The design process may be represented as the following cycle:

- Identify a set of objectives as focus for refinement from the current set of active objectives;
- Refine the objectives in focus to generate alternatives;
- Identify from the active set of objectives a set of objectives to be used to evaluate the alternatives;
- Evaluate the alternatives with respect to the objectives;
- If all alternatives are eliminated, resolve the objectives; else select an alternative for inclusion in the design.

This cycle is repeated until the objectives are satisfied and the designer is satisfied with the the final artefact. The designer may modify objectives when not satisfied with the artefact. The designer determines on which objectives to focus for both refinement and evaluation with some automated assistance. The activities of refining and evaluating may be performed by operators that are invoked when specific combinations of decision and objective descriptions are present. Selection may be performed automatically if enough information is present; else, the designer must provide information for the selection. The task of resolving objectives is performed by the designer. We provide some facilities (e.g., backtracking, tracing effects of changes to objectives and variables etc.) to assist in this task.

Control. The control of the process rests with the designer. The designer determines the objectives for focus and resolves conflicts among objectives.

Outputs. The design process is documented in the design record. The information in the record includes the high-level objectives that were considered, how they were refined, the sequence in which the objectives were considered, interactions between objectives, and how interactions were resolved to arrive at the final design.

A number of operations may be supported by the information in the record. Changes in the values of design decisions and objectives can be propagated through the activity network to determine their effect on the design. Since the activities establish relationships between design decisions and objectives, decisions and objectives that are affected by changes in the values of other decisions or objectives can be identified.

The record can be used to explain the derivation history of a design decision. For example, how was the decision to use A242 steel made?

3.2.4 Contributions and Challenges

We are currently creating a prototype to demonstrate our approach. The primary contribution of this work is the identification of a model for design as a refinement of objectives and the identification of a computational paradigm for the representation of the components in the model.

The current model does not account for the reasons why objectives are addressed in some sequence by the designer. In addition, the computational representation for operators has not been completely addressed. It may not be possible for the designer to express all of the objectives in a computable form. Designers may find it difficult to articulate their objective, although they are able to use it to evaluate the design alternatives. Although we allow for informal textual descriptions of objectives, they are interpreted by the designer and are treated separately from computable representations. We need to look at the integration of informal and computable representations. The practicality of this approach needs to be tested through case studies. Finally, it may be possible to go over the record of a design process *a posteriori*, identify places where wrong decisions were made, and find rules that might, at the very least, improve the process for the same design problem.

3.3 Grammar Interpreters

3.3.1 Objectives

The Grammar Interpreters project has pursued the goal of developing a *complement* to human designers in the form of two capabilities: to support the expression of constructive transformations of designs within a design process; and to permit the systematic exploration of the possibilities given by the transformations. The project was motivated by the situation described in Section 2.1, a motivation it shares with the Layout Synthesis project (see Section 3.10). A particular point of concern for the project have been design situations in which *design moves* (or transformations) accumulate through experience and are used repeatedly over time, either within one design task or across design tasks. Such situations are typical in many disciplines, particularly in architecture, where successful design ideas develop slowly over many projects, are adapted to a new context for each project, and are used many times within a project. For example, successful building details are developed slowly over many projects and are used individually in many contexts within a project.

The project sits firmly within the so-called *search paradigm* for design. That is, it considers the specification (by constructive rules) of a discrete derivationally-connected space of designs and guided movement through it as a metaphor for design action. Within this paradigm it accepts the notion of *rules* as heuristic, contingent and general devices that define the topology (derivational connections) of the spaces under consideration. That rules are heuristic means that they are not definitive - they guarantee neither formal nor functional properties of designs. Instead they are a means to encode strategies (often based on experience) for making designs. That rules are contingent means that they are subject to change, both at the start of a given process and within it; changing rules means that the spaces of designs described by the rules change dynamically. That rules are general means that few of their formal properties (for example, monotonicity) can be relied upon in structuring the control of rule application.

The above position has the following implications:

- *Movement in design space is exploration.* Since little can be said of the structure of design spaces, the researchers involved have adopted *exploration* as a principal metaphor for movement through a search space. Exploration presumes changing design goals and a changing means of achieving goals. It especially presumes a weak link between means and ends; that is, *a priori* knowledge of the effects of rules on designs is not, in general, available.
- *People are a part of the process.* First, rules are heuristic devices defined and changed by people. Grammatical systems acting on rules become amplifiers of human action. Second, evaluations of performance according to multiple criteria means that only weak formal methods are available for comparing designs. Questions of judgment arise in such situations; people must assume this role in the absence of means to make judgments automatically.

Work in the project is aimed at expanding the present understanding of grammatical systems in design by developing new grammar formalisms, algorithms for matching, applying and controlling grammar rules, implementations of the first usable grammar interpreters, and applications of these grammar interpreters to design problems. Results of the project are intended to be largely domain-independent; thus there has been a focus on grammars over general geometric representations.

In its early stages, the project has comprised largely independent investigations into subsets of the above issues. Two of these investigations, which have resulted in the GENESIS and GRAMMATICA grammar interpreters, are reported here. While working on these projects, we have become increasingly aware that a serious test of grammatical ideas lies in the preparation and application of a system spanning all of the

issues. The major goal of the project is to create and use such a system in a comprehensive design context.

3.3.2 Generic Design Tasks

By their embodiment of a *recognize-act cycle*, grammar interpreters constitute a partial, but fundamental, technology for addressing all design tasks that proceed by the application of discrete transformations. Extant grammar interpreter architectures combine component selection, logical design, and physical design under a largely user-controlled search strategy.

The next sections describe the computational models underlying the GENESIS and GRAMMATICA systems.

3.3.3 Computational Model: GENESIS

Overall Architecture and Approach. GENESIS is a solid grammar interpreter that is the fruit of an effort to create and demonstrate a formalism for applying rules to representations of solids objects [25]. Unlike LOOS/ABLOOS it is not, nor did it aspire from its inception to be, a comprehensive design system. Rather, it is the product of addressing one issue in a larger research program. Nonetheless, in its present form it has been found useful for generating designs of certain classes.

A set of rules defines a space of designs which is then explored for interesting designs. Guided by observation of their effects on a design space, rules may be changed; the resulting new design space can then be explored anew.

Input. A set of rules and a starting form are given. The goals of the design task presently remain in the mind of the user.

Form and Function Variables. The primary form variables are collections of rigid solid objects and text placed in relation to these objects. Higher order form variables such as alignment, adjacency, color, and material properties can be expressed in terms of these basic variables. The basic function variables are those that can be expressed as logical predicates over form variables. The function variables used in a particular application are dependent upon the rule set involved.

Representation of Variables. A solid is represented as a boundary-representation solid model. In such models the boundary of the solid is divided into *topological elements* each of which represents a portion of the boundary. A decomposition into topological elements is made to allow the location and shape, the *geometry*, of a single element to be represented by simple mathematical expressions. GENESIS can represent general topologies of the boundaries of r-sets but is presently limited to linear (planar) geometries. Text is represented as labels associated with a representation's topological elements. Sets of solid objects are represented as sets of boundary representations. Function variables are represented as logical predicates, expressed in CLP(R), a Prolog-like language, over the form variables above.

Operations. Rules are *matched* to solids by searching for conditions or features of solids. These are expressed as clauses in first order logic. Explicit conditions of a given solid correspond to axioms about the boundary representation. Clauses, in the form of Horn clauses, allow deduction of complex conditions from simpler conditions. In this way, arbitrarily complex conditions may be specified using deductive reasoning on the solid representation. Locating a condition of a solid then becomes a matter of satisfying a goal clause that specifies the desired condition.

Rules are applied by a variety of operations based on the primitive operators comprising the Euler operators, geometry assignments, and label assertions. Operators for parametrically creating primitive solids, for performing local operations on these solids, and for computing the boolean and unary shape operations are built using the primitive operators. Operations may be organized into sequences, that is, programs.

Control. Presently GENESIS has only weak methods of control for operator application. This is not so much by design as it is a result of focusing on the grammatical mechanisms underlying GENESIS. Users can either interactively control the application of rules according to the rule matches found by GENESIS or can request a single depth first descent into the search space.

Output. The output is a set of solid models that is either a member of the language of the grammar initially specified or any of the sentential forms of the grammar.

3.3.4 Computational Model: GRAMMATICA

Overall Architecture and Approach. The GRAMMATICA project aims to lay a theoretical foundation for *grammatical programming*, a nondeterministic programming paradigm whose fundamental computational mechanism is rule unification. Grammatical programming captures the essence of conventional grammars, thus retaining the ease of expression that has made them useful practical devices for describing design spaces. But grammatical programming goes beyond conventional grammars; it supports programming styles from purely declarative to purely procedural, permits the modular construction of grammars, and enables the straightforward description of both the discrete and continuous aspects of design spaces. By correcting these deficiencies of conventional grammars, the GRAMMATICA project seeks to make grammars useful in practical design contexts.

GRAMMATICA is a domain-independent language for describing design spaces. The user begins by constructing sets of primitive designs and transformations of designs using a domain-specific syntax. Design transformations may be specified using a rewriting rule constructor, but they are not in general limited to rewriting rules. Both designs and transformations may be parametric.

From the primitive transformations, the user constructs compound transformations using the operators of a relation algebra: 1) *composition*, which sequences transformations, 2) *union*, which chooses nondeterministically between transformations, 3) *iteration*, which repeats transformations, and 4) *failure*, which tests the applicability of a transformation to a target structure. A GRAMMATICA program consists of a set of mutually recursive design transformations. Any element of that set may be applied to an initial set of designs to generate a space of designs.

The first GRAMMATICA prototype, currently being implemented using the constraint logic programming language CLP(R), will read a GRAMMATICA program as a whole, and then provide commands for exploring the design spaces that the program describes.

Input. The task faced by the GRAMMATICA user is not the solution of a design problem *per se*, but rather the description of a space of designs. The desired space is specified by the formulation of primitive design transformations and the construction of compound transformations from the primitives.

Form and Function Variables and their Representations. Because GRAMMATICA is a domain-independent formalism, it does not prescribe a representation. We will be exploring its use with a general representation of geometric structures: labels, points, vectors, and frames of reference (affine

transformations) grouped into hierarchical structures of lists and sets. Function variables may be expressed within this representation as terms and logical predicates in the hierarchy, and by constraints attached to designs.

Operation and Control. It is useful to distinguish between two levels of control in the description and exploration of design spaces. The topology of the design space itself is determined by control structures that govern when which rules may be applied to which designs when. GRAMMATICA's transformation-combining operators provide this control. The course of a search through a space is controlled by yet another mechanism. Although search control is not prescribed by the GRAMMATICA formalism, the GRAMMATICA implementation will provide the user with a set of commands that generate selected designs in a space, for example: the designs immediately derivable from a given design, the designs along a random path through the space, or the designs that result from breadth-first and depth-first enumerations of the space.

Output. The output of a grammatical program is in effect the design space rooted at a given set of initial designs. As most design spaces are infinite, or practically so, the output of a grammatical program exists only virtually; access to it is given by the search commands described above.

3.3.5 Contributions and Challenges

The main contributions of the project thus far have been 1) to show that useful grammar interpreters can be implemented, 2) to show that the control of rule applications can be incorporated directly within a grammatical formalism, and 3) to develop a homogeneous model for describing both the discrete and continuously-varying aspects of design spaces. Prototype applications have been written that generate a wide range of designs, including Queen Anne houses, computer cabinets, building structures and a variety of geometric curiosities. The diversity of the design classes currently created gives credence to the design metaphor of using changeable heuristic rules to define spaces of possibilities that are then explored for particular designs.

Experience with these applications has underscored the need for work in all four areas of the project. First, all grammatical formalisms have limitations at the present time (in applicability and implementability), and applications exist that are not served well by any existing grammar formalism. For example, non-parametric shape grammars have a strong characterization of emergent form, and we understand their implementation; yet rules are not parametric and must therefore be too specific. On the other hand, the structure grammars used in GRAMMATICA both support parametric rule specification and produce parametric designs, yet can only weakly employ notions of emergence. It seems that there remains much to be done with grammatical formalisms. Second, we have only a weak understanding of algorithms for matching, applying and controlling grammar rules, and of the complexities of those algorithms. Building fast and reliable interpreters demands that such issues are addressed. Third, issues of how people can use grammar interpreters to accomplish design tasks have been opened with every investigation in the project, but to date no solutions have emerged that are sufficient to form a general paradigm for implementations. Fourth, experience with applications indicates that there are many areas of design in industry which could benefit from the use of grammar interpreters. Essentially, any area in which forms of multi-component designs display invariance across multiple projects would be a likely candidate for application of grammatical methods. For example, computer cabinets from a given company typically have forms that are stylistically coherent across a number of models and components. By capturing the essence of the style in a grammar, the space of possible cabinet designs can be conveyed to a computer, which can then support the search for solutions to cabinet design problems.

At present, researchers in the grammar interpreters project are pursuing four main goals:

1. Developing and applying GENESIS: Presently GENESIS is a usable prototype system for defining and applying rules. It is being used in a number of contexts, especially the design of building structures. The experience gained from these explorations is being used to refine its present implementation.

2. Developing the theory and practice of grammatical programming: If grammatical programming is to be useful and reliable, it must be based on an intuitive syntax and sound semantics. Present work is focused on specifying a denotational (and an equivalent operational) semantics of grammatical programming. A demonstration implementation based on the theory is being written.

3. Devising intuitive means of grammatical programming: Grammatical programming is an expressive means of describing spaces of designs. The notation is terse, though, and this motivates an attempt to devise simple ways to specify grammatical programs, perhaps via graphical interaction.

4. Devising means of exploring parametric design spaces: The ability to create parametric designs is only one part of exploring spaces of parametric designs. Once a parametric design is generated, the possibilities defined by its parameter space must be understood. It would be best if the two processes (grammatical generation and parametric variation) could be coupled tightly in a system so that they could be used almost simultaneously to define and guide an exploration of design possibilities.

3.4 A Solids Grammar Using Spatial and Functional Attributes in Structural Design

3.4.1 Objectives

The objective of this project is to incorporate form and function considerations into the generative grammar paradigm. Shape, solids, and structure grammars, as presently used, concentrate on form. They provide a formalism for marking shapes with temporary labels which serve to control the growth of shapes. The functional properties of these shapes require persistent labels that represent the functional attributes. Form-function synthesis grammar rules need patterns of {shapes, markers, functional attributes} both in the condition (LHS or antecedent) and in the action (RHS or consequent) parts of the rules. This study is an exploration of the type of attribute algebra needed to complement the shape algebras for simultaneous form-function synthesis.

The specific domain is the class of high-rise buildings characterized as tube structures, in which the envelope of the building serves as the major load-carrying element. Therefore, architectural considerations on shape, massing, and visual impact, are coupled to structural considerations of strength and stiffness.

3.4.2 Generic Design Tasks

The grammar rules implement the following tasks:

- extension of the problem specification by supplying parameters defining usage categories for the building,
- specification of building components and components of load-carrying structure
- physical design of building and structure components, and
- analysis and evaluation of structural system.

3.4.3 Computational Model

Overall Architecture. The prototype system is built on top of the GENESIS grammar interpreter. Additional predicates have been provided to manipulate functional attributes represented as persistent labels and to invoke external procedures.

Inputs. Synthesis knowledge is represented as a set of rules. The conditions (LHS) of the rules may involve both shapes and functional attributes. The actions may create or modify shapes, create or modify functional attributes, or invoke operations, such as performing a finite-element analysis of the current structural configurations. The synthesis problem to be solved is represented by an initial object of indeterminate shape with attributes representing the building program and the building site.

Form and Function Variables. Form variables are 3-D solid objects. The objects may be real (e.g., the face of the tube or the columns and spandrel beams composing the tube), virtual (e.g., the initial shape with no geometry) or auxiliary (e.g., display of forces by means of solids, the heights of which are proportional to the axial forces or shears in the elements). The function variables are the architectural attributes (e.g., occupancy type and shape preferences), structural attributes. (e.g., material type and strength) and derived behavior attributes (e.g., forces and moments).

Representation of Variables. Objects are represented as boundary representation solids. Some dimensions of the objects are real (i.e., they define dimensions of the building or component), while others are arbitrary (i.e., serve only for display purposes). Functional variables are represented as persistent labels on the solid objects.

Operations. The operations follow the GENESIS architecture. Rule conditions are evaluated by matching on patterns of shapes and attributes. Rule actions perform definition and shaping of real and virtual objects and the definition, assignment, and evaluation of functional attributes.

Control. Control is presently a user-driven dialog. The system displays currently applicable rules and the user chooses to apply or not to apply a rule.

Output. The output is a building model (a hierarchy of objects and their functional attributes) satisfying the architectural and structural requirements.

3.4.4 Contributions and Challenges

The contribution to the domain of building design is the formalization of architectural and structural synthesis in the form of grammar rules. This provides a much finer-grained representation of domain knowledge than the global representation in HI-Rise [26], so that individual rules may be evaluated and critiqued by practitioners. We are currently undertaking a knowledge acquisition effort by interviewing five teams of engineers and architects, using the current prototype as a starting point.

The methodological contribution is the development of an attribute algebra to provide a semantic complement to the syntactic capabilities of solids shape grammars.

Among the challenges are the following:

- development of robust functional decompositions and hierarchies that support reasoning about multiple functionalities;,
- development of top-down and bottom-up mappings between functional entities and discrete

components (i.e., functions mapping into assemblies of components as well one component mapping in to several functions), and

- development of seamless integration between spatial and functional attributes in the grammar formalism.

3.5 Transforming from Behavior to Structure in Mechanical Design

3.5.1 Objectives

The goal of this project is to create a methodology by which design specifications for a mechanical system can be transformed into the description of a configuration of mechanical components. Both behavioral and physical *requirements* as well as behavioral and physical *characteristics* of the available mechanical components must be represented to execute such a transformational approach to design. Because the interactions of components are important, the representation of the behaviors of mechanical components must be linked to the representation of their physical characteristics; that is, relationship between form and function of components must be captured.

3.5.2 Generic Design Tasks

To study the transformational process, we are working in the the domain of gear box design. Clearly, one reason for selecting this domain is that gear box design is a well-understood, highly-parameterized area of mechanical design. Nevertheless, we believe that our representation and transformation formalism will be applicable in other mechanical design domains, particularly to the class of design problems that we call *configuration design*. By configuration design, we mean designs composed from standard component families but for which allowable configurations are not specified *a priori*.

3.5.3 Computational Model

Overall Architecture and Approach. Our strategy is to transform the behavioral specification to generate different power path topologies, each with the same overall behavior. Each power path topology in turn can be realized in many different shafting and gear configurations. Note that the transformations are one to many; that is, the initial behavioral requirements can be transformed into many possible power path topologies. In turn, each power path topology can be transformed into many possible configurations and each configuration can be transformed into many different geometric instantiations. We are creating a methodology that can generate an efficient sequence of transformations leading to a final design that meets the design requirements.

Physical and behavioral specifications and characteristics can be represented as combinations of abstract primitives. They are abstract because each primitive corresponds to only one behavioral or physical characteristic. Individually they do not correspond to any particular component or configuration of components, but collectively they may represent the design specifications or the form and behavior of components.

We employ a *specification graph* composed of abstract behavioral primitives that do not necessarily correspond to any physical components. To arrive at a design, we transform the specification graph, preserving the system behavior, to obtain a graph that corresponds to a collection of components. the

transformation process is guided by the goal of function integration and incidental behavior principles and by knowledge of the available components.

Inputs. The behavioral requirements for the design are stated in a bondgraph. Additional physical requirements are expressed as constraints on the final configuration. (For example, the total mass must be less than x.) A database with the available components must also be specified, and each component must have associated behavioral and geometric models. The geometric model of a component includes models of the allowable interconnections with other components. The set of behavior-preserving transformations is given (rewrite rules).

Form and Function Variables. The form variables are the geometry and topology of the components and their allowable interconnections. The behavioral variables are power flows, transformations, and connections.

Representation of Variables. We use bond graphs to represent the design requirements. A major advantage of using bond graphs is that we can define transformation rules that alter the structure of the bond graph but that do not alter the dynamic behavior of the system represented by the graph. The implications of this statement are important. Because we can transform the specifications graph to represent many different physical systems, we do not impose an initial structure or configuration on the physical design; that is, we do not require an *a priori* decomposition of the design specifications.

The behavioral characteristics of components use the representation that is used for specifications, *i.e.* bond graphs. Each component has a bond graph associated with it that represents its behavior *in isolation.* One crucial difference between specifications and components is that the behaviors and physical characteristics of a component are inherently linked; no single characteristic, either behavioral or physical, can be obtained in isolation. For example, a spur gear may be selected for its power transformation characteristics, but its inertial characteristics are implicitly selected as well. The representation of components therefore requires linked models for form and behavior, while the representation of specifications has independent models for form and behavior.

Component interactions are both physical and behavioral. For example, meshing gears share a bond with a common pitch velocity. In the physical configuration, this translates to the requirement that their pitch elements must touch. The model of a component in a configuration therefore must include its location and orientation. Behavioral models may also include characteristics such as mass and volume that can be derived from the physical models. These behavioral models may be defined for the entire configuration or for individual components. Modeling issues are discussed in detail in [27].

Operation and Control. A grammar for bond graphs defines transformations that preserve the overall behavior of the system [11]; however, some of the graphs represent physically realizable designs while others do not. In addition to knowing the general rules for behavior-preserving transformations, we need to know the constraints imposed by the components. Guidance in selecting which transformation to apply comes from the physical requirements of the system, from the physical characteristics of the components, and from the relationship between geometry and behavior of the components.

Outputs. Currently, the output of the system is a set of constraints, which when solved, result in a physically realizable configuration for a gear box which meets the behavioral and physical requirements.

3.5.4 Contributions and Challenges

We have created a methodology which, for a limited class of design problems, transforms the functional specifications into a physical system. We implemented this methodology in a test bed using the Constraint Logic Programming environment provided by CLP(R) [28].

Many research issues remain open even for this limited class of problems. For example, not all constraints can be propagated forward without backtracking and the generation of alternatives must be controlled using domain knowledge. In addition, the representation of behavior is limited to lumped parameter dynamic systems, a small subset of mechanical design problems.

3.6 Conceptual Design

3.6.1 Objectives

Form-function synthesis processes need not only specifications, but also systematic methods to produce and manage these specifications early in the evolution of a design. We describe a representation of and a systematic approach to conceptual design. The central idea underlying the approach is to capture design intent through a chain of functional description and reasoning. The model of intent highlights the dependencies among the sub-functions of a design and aids in form synthesis.

3.6.2 Generic Design Tasks

Our model allows an initial rough functional specification to be refined concurrently with component selection and, to a limited degree, physical design.

3.6.3 Computational Model

Overall Architecture and Approach. Our approach to conceptual design is based on [29] and employs functional descriptions of products or processes according to a set of linguistic and hierarchic rules. Within this approach, existing artefacts are analyzed by deriving their function logic in a bottom-up fashion. New artefacts, in general, are synthesized in a top-down fashion [30]. The *function block diagram* (FBD) employed by us is given a three-tiered structure consisting of 1) *function blocks* (compact verb-noun descriptors of what the design does rather than what it is) with links to other blocks, 2) *allocations* (constraints, performance requirements, specifications and resources), and 3) *components* (artefacts that satisfy the given function).

The development of form occurs on at least two levels in this model concurrently with the development of function. An abstract representation of form resides in an allocation list, set of specifications which is always linked to a given subfunction. A concrete representation of form may be developed from the allocations by either decomposition to obvious components or synthesis within a supported function.

Inputs. Inputs to conceptual design depend on whether the design proceeds from the function or from the artefact. In top-down design, the inputs are verbal, syntactic, and numeric descriptors which evolve with understanding and negotiation. In bottom-up or reverse design, the input is the set of descriptors embodied by each of the given components and subassemblies of an existing design. Since there is no evidence of design proceeding exclusively in the top-down mode, descriptors from related but independent designs will always be present.

Form and Function Variables. In principle, any form or function variable of interest to the design team can be taken into account.

Representation of Variables. The general form of the function block diagram (FBD) is shown in Figure 5. The function block contains the function name expressed as a generic noun/verb pair to describe the function of the product or process. The verb must be active; the noun must be measurable. The nodes to the left of a function node represent the reason why a function is included (a higher level function). The nodes to the right are functions describing how the function is performed (lower level functions). Links connect each high level function with its lower level functions according to this how/why relationship. Links other than how/why may also be expressed, such as causal, temporal, informational, alternative, and revisional [31]. Strict hierarchy is not required since a more specific function may satisfy more than one less specific function in the diagram.

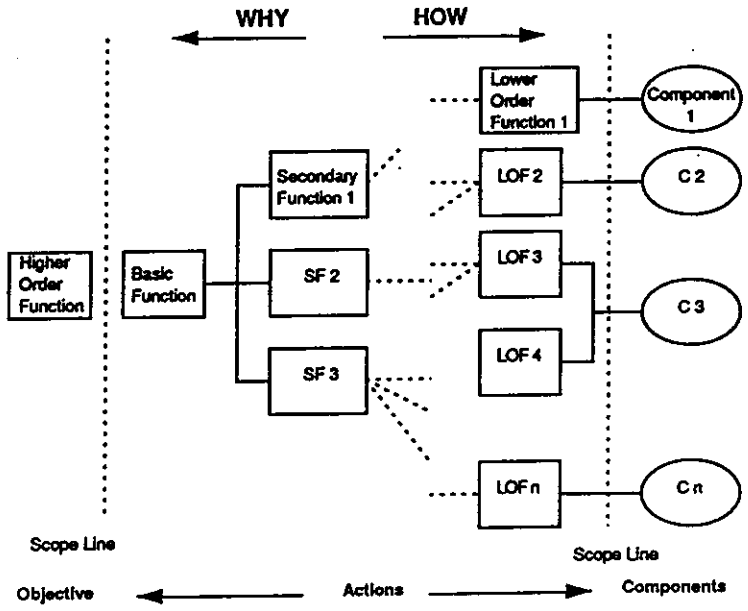

Figure 5: Function block diagram

Candidate artefacts and/or supported functions (coded domain-specific knowledge, formulas and examples) are specified to satisfy the lowest level functions, which in turn are embodied by the higher-level ones.

Operations and Control. The basic function of the design is established by agreement of the design team. If the basic function cannot be accomplished by a single known component, it is decomposed into several functions which collectively perform the function. These secondary functions may then be translated into components or recursively decomposed. The function decomposition process continues until 1) the decomposition process is out of scope for the project, 2) there exists a synthesis technique which will complete the decomposition or propose artefacts, or 3) each function can be mapped into a component or structure that will accomplish it directly.

During the development of the function logic, and ideally before detail design begins, the designer must address the issue of which agent will perform each identified function. This process is known as function allocation. In addition to resources, the allocation list contains constraints, performance requirements and component specifications. We have adopted a structure to manage this information based on the verb classification of signal, energy, or material as suggested by [32].

The allocations for each function block are developed and passed to the neighboring blocks by inheritance rules. Frequently, the allocation list attached to the basic function is not known precisely at the beginning of the process. As the lower level functions are satisfied by artefacts or supported functions, new categories of information are specified, passed up to the basic function, and inherited by certain lower level functions of the structure. Thus, the allocation list supplies what needs to be known as the design evolves. Its values drive the process of type, number, and dimension synthesis. In addition to the rules for function block analysis and synthesis, one can apply specific steps to detect the existence of linkages in the function structure.

Outputs. The principal output is a valid function block diagram with its allocations. The allocation list attached to a function contains the relevant design specifications, performance requirements, resources, and component specifications that would drive domain-specific synthesis tools. In some cases, the allocations of the lowest level function block will identify a component with definite geometric and material properties. The details of the function block diagram may include revisions, alternatives, and process-dependent functions. The allocation lists will include behavioral and compliance information.

The design history is captured by the function block diagram throughout the development of the product since a record is kept of the alternatives that are discarded and the revisions made. The logic behind each design can therefore be understood by both the expert and novice designer at any time. Moreover, the function block diagram can be a means of communication between designers in different domains.

3.6.4 Contributions and Challenges

Conceptual design intent is captured by the FBD in the form of how/why logic throughout the development of the product because a record is kept of the alternatives that are discarded and the revisions made. Special link types connect functions through information transfer, causality and temporal constraints at a high level. Thus, the logic behind each design can be understood by both the expert and novice designer at any time. The FBD is also a means of communication between designers in different domains because the interaction of domain-based functional decisions is explicitly expressed.

Two substantial challenges lie ahead. First, a large body of experience shows the FBD to be an effective design and analysis tool when used in the context of a *design team.* Design and implementation of an interactive software environment which encourages this approach in a more personal context is clearly needed. Our colleagues have constructed a generic block/link data management system [33] with which to carry out exploratory studies in the use of our model. Second, the outputs from the conceptual process need to be integrated with the inputs of the other domain-specific approaches described in this paper. The limitations to sufficiency of the data structures and the level of input detail would be determined by this task.

3.7 MICON: Synthesis by Composition

3.7.1 Objectives

The overall goal of the MICON project is to provide an integrated set of tools to design and manufacture, in a short period of time, a computer system from high-level input specifications [34]. Specifically, we have developed a system that automates the synthesis of the electronics in computer systems by composing off-the-shelf microprocessor family components. Input to the system is a set of high-level specifications that describe the functionality of the computer. The output of MICON synthesis tools is a *netlist*, a list of components and interconnections of pins on these component.

In addition to satisfying requirements on functionality, the electronics synthesis process must consider the design from several other perspectives, such as reliability and testability. MICON includes a design advisor for each perspective. The design advisor analyzes and makes recommendations to the synthesis tool from its perspective. Design advisors are not discussed further in this section, and the discussion is limited to MICON's core synthesis tools.

3.7.2 Generic Design Tasks

The MICON approach addresses a combination of logical design and analysis and evaluation tasks. Physical design (placement of components and routing of the interconnections) is done by external commercial tools.

3.7.3 Computational Model

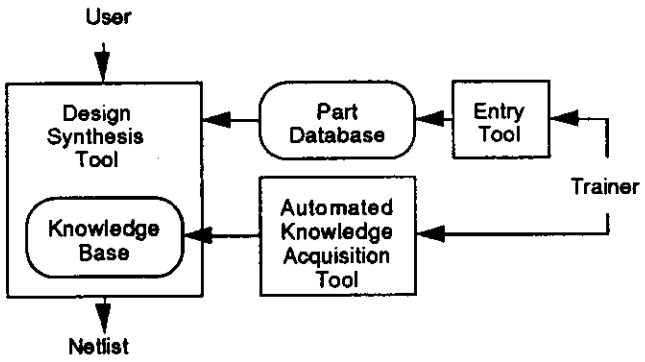

Figure 6: MICON architecture

Overall Architecture and Approach. The MICON system is based on the concept of *design reuse*, that is, the capture, dissemination, and automatic reuse of design expertise. The architecture of the core MICON tools is shown in Figure 6. Information about individual components is stored as a part-model in a *database*. Information about designing with these components is stored as design templates with related constraints in a *knowledge base*. MICON's synthesis tool prompts the user to input specifications, uses the database and knowledge base to build a design that satisfies the specifications, and generates the corresponding netlist.

All design information is organized and entered into the database and knowledge base by one or more MICON trainers, usually expert hardware designers. Part-models are added to the database by an

interactive *entry tool.* Templates are added into the knowledge base by an automated *knowledge-acquisition tool.* The trainer inputs schematics and equations that generalize the templates (so that they can be reused in other similar design situations) and compiles it into the knowledge base. A novel feature of MICON is that the synthesis tool actively supports knowledge acquisition by prompting the user when it needs additional knowledge.

The accumulated design expertise can be utilized by a MICON user to generate a customized design in a short period of time. A novice user can leverage the expertise of the trainers and produce novel combinations of previously taught design fragments, but cannot produce a totally novel design.

Inputs. The desired computer system is specified by a set of attributes (termed specifications). These attributes describe the functionality of the computer (e.g. type and clock-speed of the microprocessor, amount and type of memory, number and type of input/output devices) and ceilings on resources (e.g. board size, cost, power). The user is prompted for each of these specifications. The number of input specifications is variable. For example, if the user specifies two serial input/output devices, two sets of specifications are input.

The other inputs come from the MICON trainers over a period of time. These include the following:

- A functional hierarchy for organizing parts used in building a computer. The hierarchy is built incrementally based on abstraction and problem decomposition. Leaf nodes of the hierarchy are physical parts or components used to build the design. Non-leaf nodes of the hierarchy are abstract parts that are defined by the trainer to organize physical parts using *part-of* and *is-a* relations.
- A model for each part object that is a list of attribute-value pairs. Abstract parts also have a list of specification attributes that a refinement of the part must satisfy; the value of these attributes is to be computed based on how the part is used in the design.
- Constraints between attributes associated with part objects. For example, a relation may exist between the specification attribute of a part and attributes of its successors in the functional hierarchy; another equation may indicate how the specification attribute of a part is computed.
- Interconnection of pins on parts to result in a correct design (templates).

The first two sets of inputs are stored in the database, while the last two sets of inputs are compiled into the knowledge base.

Form and Function Variables. The form variables are the part objects and the connections of pins on the parts. The function variables are the specifications associated with each part object.

Representation of Variables. The part-objects are added incrementally in the design process starting at the root node of the functional hierarchy resulting in a directed acyclic graph, called the *design hierarchy.* Each part-object has several attributes associated with it that may be static or may be computed during the synthesis process. The part-objects are represented as objects that are linked into the design hierarchy by link-objects. Each part-object also has any number of attribute-objects linked to it. Specifications are special types of objects associated with the abstract part-objects.

Operations. Operations include the following:

- *Computing specifications.* Either the user is queried for a value or the value is computed using equations (constraints) input by the trainer.

- *Search for function (select parts).* This operator refines a part into its successor parts in the functional hierarchy. When the functional hierarchy indicates that a part is decomposed into its successor parts, all the successor parts are selected. When a part is to be refined into one of the successor parts, a two step process selects a part to be added to the parts list:
  - successors that do not satisfy the constraints input by the trainer are pruned.
  - if more than one successor part is satisfactory, a penalty function heuristic is used to select one.
- *Search for structure (connect parts):* Depending upon the computed specifications and the selected parts, this operator may add additional parts to the parts-list and add connections to the netlist.
- *Compute other attributes:* Attributes associated with the part-object are computed using equations (constraints) input by the trainer.

Since successor parts must connect with the boundary of the predecessor part in the design hierarchy and can only influence the attributes associated with the predecessor part-object, no explicit aggregation/disaggregation rules are needed.

Control. The design process starts by designing the root abstract part of the functional hierarchy. Designing an abstract part is equivalent to executing the operations in the following sequence, termed a *design cycle*:

1. Compute specifications. If specifications are not complete, then return.
2. Search for function.
3. Search for structure.
4. Compute other attributes.

A design cycle is illustrated in Figure 7, which shows where in the sequence form and function variables are addressed. A design cycle is executed for the root abstract part, which introduces additional abstract parts, each of which must go through its own design cycle. Note that the design is data-driven -- a design cycle is executed only when all specifications for a part are available. Design cycles are repeatedly executed for non-leaf parts until the design has reached the leaf parts in the functional hierarchy.

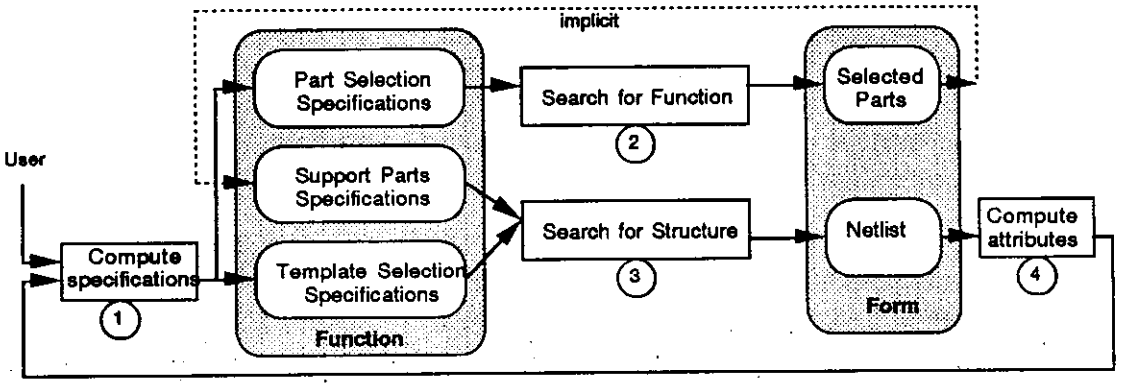

Figure 7: The MICON design cycle

Outputs. The output comprises the parts-list and netlist.

3.7.4 Contributions and Challenges

We have developed a novel synthesis-by-composition problem-solving method. This method has been demonstrated to be effective for designing computer systems; it has been used to generate several realistic "industrial strength" designs. The method has also shown to be applicable to other problem domains (window regulator design and sequencing tools). The method creates new designs by composing pre-stored fragments of designs; an approach for automatically acquiring and generalizing these fragments has also been developed.

The method relies heavily on pre-stored design information. Challenges include the verification of the existing design information and verification of new information as it enters the system. Also, correct-by-construction is not adequate and verification of designs generated by the system is an outstanding research issue. Finally, the method produces a satisfiable solution; additional methods for optimizing the solution are needed.

3.8 Synthesis of Integrated Circuits: The System Architect's Workbench (SAW)

3.8.1 Objectives

The overall goal of the the System Architect's Workbench (SAW) project [35] is to develop and demonstrate synthesis techniques for digital systems design. The work concentrates on synthesis tools for the specification of the control schedule and data path to implement a desired behavior. In these synthesis steps, operations in the behavioral description are assigned to control steps, registers are assigned to hold values, functional units are assigned to perform operations, and data path interconnections are generated to connect the registers and functional units.

The SAW project has also begun to address methods for modifying the behavioral description prior to synthesis. For instance, the behavior may be partitioned so that it can be implemented as two separate integrated circuits.

3.8.2 Generic Design Task

The principal design task undertaken by SAW is that of component specification and logic design. The designs produced by SAW specify register transfer (RT) level components and their interconnection. Recently, however, SAW has expanded to encompass physical design so that the designs it creates can be quickly realized using field programmable gate arrays.

3.8.3 Computational Model

Overall Architecture and Approach. Figure 8 illustrates the organization of SAW [36]. A mixed behavioral and structural description, written in the Verilog language [37], is provided as input. The system works from an internal data/control flow representation called the *value trace* (VT). The programs that are internal to SAW transform the behavior and synthesize a control sequence and a data path structure that implement the transformed behavioral description. Two alternative synthesis paths are currently available: one in which the control sequencer and data path structure are synthesized separately (the CSTEP/EMUCS path), and one in which the two synthesis tasks are done concurrently (the SAM path). In either case, the result is a register-transfer (RT) level design expressed in Verilog.

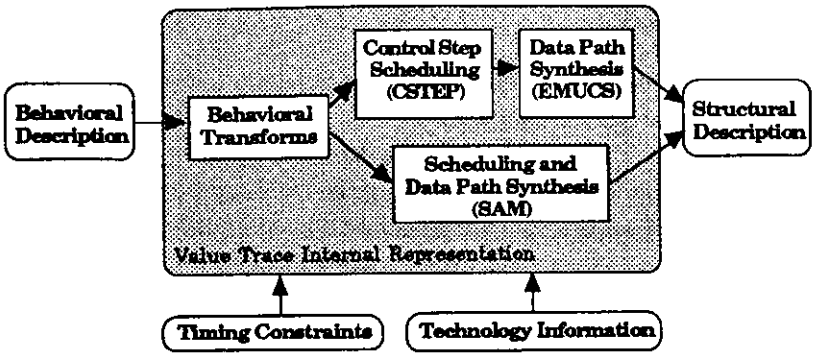

Figure 8: SAW architecture

Inputs. The input to SAW consists of a behavioral description, timing constraints, and technology information. The behavioral description is a formal specification of the logical operation of the system being synthesized. It is expressed using Verilog, a hardware simulation language, so that its operation may be verified by simulation prior to synthesis. Timing constraints embody the temporal relationship between the input and output signals of the system as well as the desired frequency of the system clock. Technology information describes the technology-specific properties (e.g. delay and cost) of the RT-level components (e.g. registers, MUX'es, and functional units).

The inputs to SAW are the results of an earlier design process. The behavioral description may be a simulation model that was used during system verification, or it may have been developed expressly for synthesis. The timing constraints are usually imposed by interface requirements. The difficulty of producing a behavioral description is ensuring that it is correct and interacts properly with other parts of the system. The use of Verilog aides the process of developing behavioral descriptions by providing a unified model for simulation and synthesis [38].

The synthesis problem for SAW is to choose and connect the RT-level components that implement the behavior described by the input. Furthermore, the design must meet the timing constraints and must be realizable given the technology information. It is desirable that the design be not only correct, but also optimal in terms of either physical size, throughput, or some combination of the two.

Form and Function Variables. The form variables of the resulting RT-level design are the set of RT-level components, the interconnection of the components, the states of the control sequencer, and the *next-state* and *output* functions of the control sequencer.

The important function variables are the physical size of the design (the amount of circuit area) and the throughput of the design (the number of inputs that can be processed in a given amount of time). SAW attempts to optimize the design by reducing the physical size and increasing the throughput of a design when possible. It should be noted that there are many other function variables of an RT-level design, such as maximum clock rate and power dissipation, which are not explicitly optimized by SAW.

Representation of Variables. The internal representation of the behavior, the VT, is a directed, acyclic graph in which the vertices represent operations and the edges represent values. As synthesis proceeds, the VT is modified and other structures are built to describe the evolving form of the design. When the design is complete, the structure of the design is translated back into Verilog and written as output.

The tools that operate on the VT during synthesis have their own representations for the problems they solve. Conflict graphs and compatibility graphs are used by many tools to represent the constraints within a design. The SAM synthesis path [39] uses a force-based representation [40] in which alternatives are viewed as "forces" pulling on the decision being made.

The function variables are derived from the evolving form of the design. Even though SAW has does not try to meet goals for physical size or throughput, it does try to optimize these variables. By allowing more than one value to share a register or more than one operation to share a functional unit, SAW attempts to reduce the area of the design. By chaining multiple operations into a single control step or by duplicating hardware along a critical path (thus increasing the computational parallelism) SAW attempts to increase the throughput. The representation of the function variables in any synthesis subproblem depends on how the subproblem is formulated. If the subproblem formulation is based on a conflict graph, for instance, the area of the design (e.g. the number of registers needed) may be analogous to the number of colors needed to color the graph.

Operations. Many of the operations applied to the representations are primitive. These operations include the following: assigning a color to a node of a graph, assigning an operation to a control step, and assigning hardware (functional unit, MUX, or register) to a node or edge in the VT. These operations are used to construct portions of the RT-level design (e.g. controllers, data paths).

Some operations, such as behavioral transformation (modifications to the structure of the VT), architectural partitioning, and redrawing process boundaries, have a greater impact on the final design. They are used to transform a behavioral description into one that is more suitable for synthesis under the current constraints. For instance, a large description may need to be partitioned so that it can be realized with two physical integrated circuits. Also, a process that must react quickly to any of several events may best be implemented as a number of separate processes.

Control. The techniques used by the SAW tools are all constructive in nature; that is, they attempt to build an optimal design by making a sequence of carefully chosen design decisions. Furthermore, the control structure of the decisions making process is typically heuristic search. Tools that use graph-based problem representations often employ some form of graph-coloring or clique-partitioning heuristic.

The control mechanism used by the SAW synthesis path relies on the forces in the problem formulation to guide not only the decisions it makes, but also the order in which it makes them. Decisions are made based on the strength of the forces applied by the various alternatives. The forces are defined so that they express the global as well as local ramifications of a given alternative. In addition, the forces are updated as the design progresses. In this way, a simple best-first search strategy can result in near-optimal results even for highly complicated problems.

Some of the high level operations implemented by SAW (applying behavioral transformations and re-forming processes, for instance) must be applied by hand. No formal control structure yet exists for these operations.

Outputs. Regardless of the synthesis path taken through SAW, the output is always a Verilog description of a register-transfer (structural) level design. The use of Verilog to describe the output of SAW enables the results of SAW to be verified by simulation.

3.8.4 Contributions and Challenges

SAW represents an attempt to design for a given function while optimizing form variables (i.e. area and throughput). It does so by allowing an abstraction of the form variables to guide the choices it makes as it assembles and connects components. Even though the problem of finding an optimal form for a given function is intractable, the algorithms embodied by SAW are able to find near-optimal solutions quickly.

The major challenge remaining in this work is to apply it to synthesis problems in which the desired behavior may be realized with a wide variety of functions as well as forms. In these synthesis problems the mapping from behavior to structure is much less straightforward. For example, if the desired behavior was "a system that facilitates document processing," the possible designs (e.g. an IBM PC, an Apple Macintosh, or a Unix workstation) vary greatly not only in form but also, at a lower level, in function. Such problems pose a significant challenge to synthesis because the design spaces are quite large and the tradeoffs are as yet poorly understood.

3.9 Design of Bridges (BRIDGER)

3.9.1 Objectives

BRIDGER is a system that assists in the preliminary design of cable-stayed bridges [41]. Its objective is to explore the effectiveness of machine learning techniques for the generation and use of design knowledge about form-function relations. The system uses different techniques suitable for acquiring knowledge for specific design subproblem. The project articulates the issues involved in the selection of machine learning techniques for building other learning programs [42].

3.9.2 Computational Model

Overall Architecture. BRIDGER has two main modules: synthesis and redesign; both integrate learning and problem solving. In the synthesis system, learning mechanisms incrementally build synthesis knowledge structure from design examples. Examples can be obtained from two sources: the literature or design scenarios. The learning process is denoted by the arrows labeled (1) in Figure 9. Problem solving uses this knowledge to synthesize candidate bridges for new specifications. The synthesis process is shown by arrows (2) in the figure. Since the knowledge created is heuristic by nature, candidate designs are usually inadequate in some aspects; they might violate constraints or be inefficient in some manner. The redesign system resolves these problems. Candidate designs are passed to a critic that analyzes them and submits them to a redesign module, if necessary. The iteration through analysis and redesign results in acceptable designs. The knowledge used by the redesign system is also generated by learning mechanisms that use examples of redesign scenarios and user explanations.

Inputs. A design problem is specified by a list of property-value pairs that mainly determine the size of the bridge (e.g., the LENGTH and WIDTH properties). Optionally, any attribute that describes a bridge can be specified as a requirement; this may impose constraints in addition to the required function.

Form and Function Variables. The primary function of a bridge is to carry a given traffic volume between its ends. The variables that account for this functionality are the length, the width, and the number of decks. The secondary function of a bridge is to support the loads exerted by the traffic on the bridge. The form variables describe how the load carrying elements are placed in space. They include the layout of stays, number of stays, tower height, stay spacing, and main span. The form variables

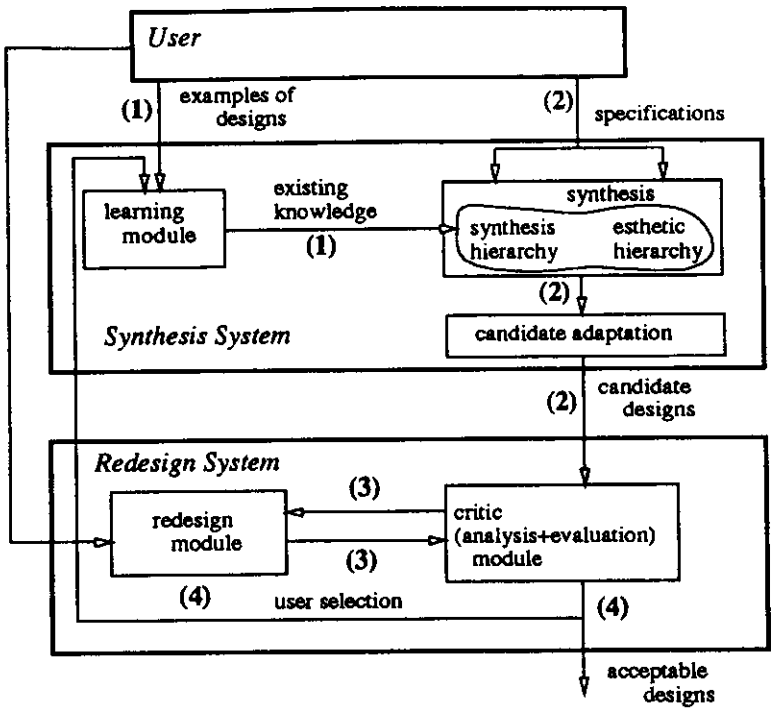

Figure 9: BRIDGER architecture

influence loads additional to the traffic load, such as wind, and the dynamic behavior of the bridge. The secondary function and the form of bridges strongly interact.

Representation of Variables. Form and function variables have the same representation: they are all represented by attribute-value pairs combined together into a complete description of the bridge.

Knowledge is organized in two classification hierarchies [43]. One hierarchy is built from the properties explicit in bridge descriptions and is called the *synthesis hierarchy*. The second hierarchy is generated from artificial examples; each example is composed of the original specification properties and several *aesthetic* properties and is called the *aesthetic hierarchy*. An example of an aesthetic property is *MSL-RATIO*, the ratio of main span to crossing length. The aesthetic properties were pre-selected as part of the problem analysis in the domain of cable-stayed bridge design; they are known to encapsulate important domain knowledge, but the exact content of this knowledge is not known.

Operations. BRIDGER uses the two hierarchies and several synthesis methods to generate candidate designs. The synthesis strategies can be described as variations on hierarchical classification. First, BRIDGER uses the synthesis hierarchy to synthesize several candidate designs. The hierarchical classification proceeds by assigning attribute-value pairs to the design. These attributes may address function or form depending on the knowledge generated from past experience. If a form attribute is assigned before a function attribute, it suggests that the form attribute value was more significant than the function attribute value for a class of previous design, and vice versa. No prior ordering is exercised on the assignment of attributes, and the assignment may be completely interleaved.

The synthesized candidates will never fit exactly the specification of the new problem. The candidate serves as raw data for a new candidate that satisfies the specification. In this sense, the synthesized candidate is always viewed as a prototype.

BRIDGER retrieves a set of aesthetic property-value pairs from the aesthetic hierarchy, using the same methods that are used for synthesis. Therefore, there are no generally fixed *a priori* aesthetic values; rather, they are context-dependent. The current context, which is the specification of the new design, is used to access aesthetic values in the aesthetic hierarchy. Aesthetic values also evolve in time as aesthetic properties of new bridges are assimilated into the aesthetic hierarchy. The aesthetic attribute values are used, in a process called *adaptation*, to scale the candidate designs to fit the problem specification.

The product of the adaptation stage is a bridge description that can be analyzed by a finite-element program. The analysis results are used to devise redesign modifications that are executed until the bridge satisfies the requirements and the governing design codes. These steps are denoted by arrows (3) in Figure 9. After each redesign modification, the bridge is again proportioned using the aesthetic properties.

Control. BRIDGER exercises a heuristic hill-climb control in the learning and synthesis processes. The control of the scaling process is fixed to a pre-defined order of scaling operations.

Output. BRIDGER outputs a set of complete descriptions of candidate designs. The candidates are presented graphically and in tabular form. The user selects a partial set and submits it to the synthesis module for further training as denoted by arrows (4) in Figure 9.

3.9.3 Contributions and Challenges

Beside its general contribution to understanding the role of learning in design, BRIDGER demonstrates an approach to combining form and function in synthesis without any presumption about the relative importance of either type of concern. A pre-defined ordering on the form or function attributes is easy to implement; this may allow BRIDGER to handle intent.

BRIDGER's approach is limited in the representation of artefacts it can handle (lists of attribute-value pairs). Expanding the scope of the approach necessitates an extension to hierarchical representations or graph structures.

3.10 Layout Synthesis

3.10.1 Objectives

The Layout Synthesis project originally pursued the goal of developing a complement to human designers in the form of two capabilities: to systematically enumerate solution alternatives with interesting tradeoffs; and to be able to take into account a broad and open spectrum of possibly conflicting criteria, constraints or requirements. That is, the system was motivated by the situation described in Section 2.1 and was meant to assist designers in exploring the form possibilities available and their performance with respect to multiple concerns. Although applied initially to layout tasks in the domain of building design, the system was intended from the beginning to be easily adaptable across domains and disciplines, given the importance of layout, floorplanning, or placement tasks in various disciplines.

While working on these and subsequent applications, we discovered that we were developing mechanisms that could also support other design methodologies and began to view these mechanisms as a flexible and expandable toolkit from which various design strategies could be implemented. The goal is now to develop this toolkit and a general framework for the implementation and exploration of such strategies,

with particular emphasis on large layout problems. The present section limits itself to a description of the original approach, which was implemented through the LOOS system [44] and its successor, ABLOOS (Abstraction-based LOOS) [45, 46].

3.10.2 Generic Design Tasks

The ABLOOS approach combines physical design with analysis and evaluation under a control strategy that alternates between the two tasks.

3.10.3 Computational Model

Overall Architecture. The standard strategy provided by ABLOOS for solving layout problems is a form of hierarchical generate-and-test, where the term *hierarchical* is to be understood in a double meaning. First, the system develops solutions incrementally by placing one object at a time and is able to evaluate and prune intermediate solutions in order to avoid computational problems often associated with a blind generate-and-test approach; this is the meaning given to the term in the AI literature [47]. Second, the system allows for a layout task to be hierarchically decomposed into subtasks, each of which represents a complete layout problem at a particular level of abstraction that can be solved with the generate-and-test method used in LOOS. The overall architecture of the first ABLOOS version is shown in Figure 10.

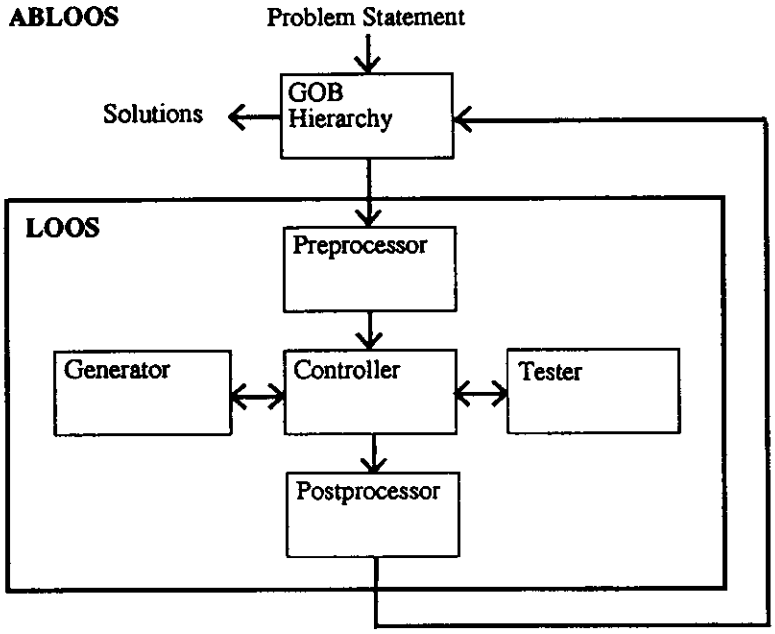

Figure 10: ABLOOS architecture

Inputs. The problem decomposition for a particular layout problem is represented by a hierarchy of *goal objects* called "GOBs", each of which represents a complete layout task at a particular level of abstraction (this problem decomposition is determined by the user at the present time). Each layout task in this hierarchy is defined by a list of components to be allocated (some of these can be preplaced) and by a collection of constraints or criteria that are associated with each component type and define a feasible layout.

Form and Function Variables. The primary form variables are the primitive spatial relations *above/below* and *right/left* in terms of which different layout structures or topologies can be defined and

generated. Any realizable structure defines upper and lower bounds for the (possibly continuous) coordinates of the objects involved, which are also stored explicitly because of their importance during evaluation. Higher-order form variables such as alignments can be expressed in terms of these basic variables.

The basic function variables handled by the system in its present version are expressed as constraints that define a feasible solution and are either satisfied or not by a design generated by the system.

Representation of Variables. A set of spatial relations or a specific topology is represented by a directed graph, where the upper and lower dimensional bounds implied by the structure are attributed to the respective nodes. Such an attributed graph is called a *configuration.*

The evaluation of a design in terms of applicable constraints is represented in an *evaluation record*, a list of declarative statements about the satisfaction or violation of these constraints.

Operations. Alternative layout configurations are generated by two sets of operators working on the graph representation: *Generation rules* that generate well-formed configurations from well-formed configurations by insertion of one object at a time, and dimensional *propagation rules* that update the dimensional bounds after each such insertion. The rules are combined in a generator able to accept any configuration and to expand it in all possible ways. The formalization of the rules as recursive rewrite rules makes it possible to prove important formal properties of the generator.

Aggregation and *disaggregation rules* allow for transitions between the levels of abstraction and decomposition defined by the goal hierarchy. *Test procedures* are used to evaluate the shape and placement of components in a layout for constraint satisfaction. These procedures are invoked by a simple control regime and combined in a tester.

Control. The search for feasible layouts at any level is governed by a controller; it calls the generator to expand selected intermediate solutions by adding one component at a time; passes them to the tester for evaluations; and uses these evaluations to prune intermediate states and to initiate the next generate-and-test sequence according to a branch-and-bound strategy (in the first system versions). The overall problem is solved by recursively solving the subtasks represented by the goal objects. This solution can be done top-down or bottom-up.

Output. The output is one or more layouts that violate the same (minimum) number of constraints.

3.10.4 Contributions and Challenges

Applications have been written, at least in preliminary versions, for the following domains: the layout or remodeling of residential kitchens, the layout of apartments on a floor, floorplanning of analog and digital computer boards, the layout of service cores in high-rise office buildings and the arrangement of components for single-board computers in an enclosing box. The latter two are applications in 2 1/2 and true 3D space, respectively.

The number of objects allocated in the applications vary from 4 to approximately 60. The generate-and-test approach has proven efficient for generating all alternatives with interesting tradeoffs for problems with up to 12 objects. This suggests a rough rule of thumb for the decompositions needed to handle larger problems by the same approach: one level of decomposition per order of magnitude in the number objects to be allocated.

Although the framework would allow employment of other methods that can be implemented from the toolkit or built from scratch, the original generate-and-test approach proved very effective in dealing with complex interactions between form and function variables as they occur in layout synthesis. One advantage is the *conceptual clarity* that results when the two aspects are handled by two distinct components, a generator dealing with formal or syntactic properties (in this case, basic spatial relations between objects and their dimensions) and a tester able to interpret layouts semantically in terms of their performance. This clarity is further enhanced by the distinction between generation and propagation rules, which reflects a conceptually clean distinction between the handling of the topology and geometry of a layout. A consequence of this clarity is not only that it aids system designers, but also makes the system very easy to understand by users. A second advantage is the generality of the approach and the resulting adaptability of the system. The independence of generator and tester make the generator applicable across domains and allows domain-specific testers to deal with any set of criteria, be they consistent or contradictory. A third advantage emerged during development work when it was realized that the generate-and-test architecture allows for *seamless transitions to and from other design methods or modes* such as interactive editing or parameter optimization through standard LP or NLP techniques.

Among the remaining challenges are the following:

- creating a declarative representation of domain knowledge to overcome limitations of the current representation of test knowledge,
- creating a user interface for the acquisition of domain knowledge,
- defining the problem decompositions and the exploration of different design methods and alternatives, and
- creating the architecture for the framework and support provided for design explorations (e.g. version control and transitions between design methods).

3.11 Summary

We present in Table 1 an overview of the computational models employed by the systems described in this section. The next section provides a more general comparison.

Project	Inputs	Function variables	Form variables	Representation of function var.s
Design Intent	Conjunction of objectives	Objectives and sub-objectives	Alternatives to satisfy objectives	Variable bindings
GENESIS	Starting configuration Set of rules	Predicates over form variables	Properties of collections of solids and labels on solids	Horn clauses over boundary rep. of solid models
GRAMMATICA	Specification of design space	Predicates over form variables	Properties of geometric prim.s and labels on these	Horn clauses over structures of lists and sets
Spatial and Functional Grammar	Initial state	Functional attributes of building comp.s	Geometric prop.s of building components	Persistent labels
Transforming from Behavior to Structure	Beh. requ.s Available components	Power flows Transformations Connections	Geometric and topological prop.s of components	Bond graph
Conceptual Design	Highest level function	Functions with associated constraints etc.	Components that satisfy lowest level functions	Function blocks with allocations connected by links
MICON	Specifications Parts hierarchy Design templ.s	Specifications	Parts Connections	Specification attributes
SAW	Behavior Constraints Technology descr.	Physical size I/O timing Clock rate	Components Connections	Control/data flow graph
BRIDGER	Specifications Synth. hier. Aesth. hier.	Volume Support Site char.s	Physical prop.s of load-carrying elements	Attribute-value pairs
Layout Synthesis	Problem decomp. Components with assoc. attributes	Constraints Criteria	Spatial rel.s Dimensional bounds	Eval. record Test functions

Table 1: Summary of computational models

form var.s	Operators	Control	Outputs
Variable bindings	Select, refine, evaluate, resolve objectives	Interactive construction of design record	Design record
Boundary representation solid models	Grammar rules	Interactive, depth-first search, or random walk	Solid models of designs
Hierarchical structure of lists and sets	Primitive and compound transformations	Various regimes	Selected members of design space
Boundary representation	GENESIS rules	Interactive	Solids model of building with attributes
Bond graph	Transformation rules on bond graphs	Branch-and-bound	Set of constraints defining configuration that meets requ.s
Allocations to function blocks	Decomposition of functions Allocations	Interactive	Function block diagram with allocations
Design hierarchy	Compute spec.s Select, connect parts Compute attributes	Recursive cycle through operations	Parts list Net list
Design hierarchy	Binding behavioral elements to physical structures	Graph coloring Clique partitioning Force-directed methods	Structural description of the logic design
Attribute-value pairs	Selection Adaptation	Hill-climbing	Candidate designs
Graph with attributed vertices	Gen., prop. rules Aggr./disaggr. rules Test rules	Branch-and-bound	Feasible design alternatives

Table 1 continued

4 Computational Models in Form-Function Synthesis: A Classification

4.1 Overview

The approaches underlying the computational models described in the preceding section appear to fall naturally into three broad classes. The first of these encompasses *top-down* or *refinement strategies*, in which an overall description of an artefact or design objective is elaborated through several levels of abstraction until a sufficiently detailed physical specification has been achieved.

A second and contrasting class is formed by *bottom-up strategies*, which are also sometimes called *constructive strategies*. These strategies construct the physical structure of an artefact in a step-wise fashion *at the same level of abstraction*. The refinement strategies attempt to generate at each level of abstraction structures that are *complete* relative to the granularity associated with that level; the bottom-up strategies, in contrast, work more or less at the same level of granularity and produce a complete structure only at the end.

Given the descriptions collected in the preceding section, it appears necessary to add to these classes a third one formed by strategies that *start with a highly-structured description* and perform transformations on this structure until a sufficiently detailed physical specification has been found. The transformations themselves may include refinement or constructive operations. It appears appropriate to call these approaches *middle-out strategies*, although it should be noted that our usage of this term is not entirely the same as its usage in the AI planning literature.

We can add additional stratifications to this classification when we observe that within each strategy, one of two contrasting inference mechanisms may be at work: from function to form or from form to function, which correspond roughly to goal- and data-driven processes, respectively. The following sections attempt to characterize these strategies in more specific terms. We emphasize the prerequisites for each strategy and the characteristics of the generic design problems for which they show promise.

4.2 Top-Down or Refinement Strategies

4.2.1 Function-Driven Strategies

This strategy is demonstrated by the model underlying the Conceptual Design project, which proceeds by decomposing an initial overall goal or function statement recursively into conjunctions of (sub-)goals or -objectives until physical components can be selected directly to satisfy a particular subgoal. Alternative refinements may exist at any level, and the resulting hierarchy need not be strict; e.g. a component may satisfy more than one objective. The model does not employ a particular strategy for refinement and selection at a particular level, which are left to the design team, but formalizes the expression of the subgoals (which are called "lower-level functions").

This type of goal refinement can also be represented by the recording mechanism underlying the Design Intent project, which also suggests a particular refinement strategy that generalizes the approach employed by [48]: selection of a focus objective from a given conjunction to be satisfied, followed by evaluations according to the remaining objectives.

We also view MICON as performing function-driven refinement. The MICON model refines not so much

functions or objectives, but *parts* through several levels of abstraction. We included this approach in the present class of strategies because the refinements are clearly function- (that is, specification-) driven.

An important prerequisite for following the present strategy is that knowledge about possible refinements be accessible in some form. The first two projects rely on interactions with designers for this purpose and provide an interactive recording mechanism that gives the process a particular structure . This structure reflects a particular *formal* model of the design process, but has not yet been implemented through a computational model in the strict sense that would allow computers to take a more active part in the decision-making process. The benefit is that neither project depends on pre-stored knowledge, that is, on routine situations or repetitive design problems. Conceptual Design explicitly intends to stimulate the discovery of new solution concepts by requiring an entire design team to think about options and by providing a structured framework for the exercise: the decompositions result in foci for the team's deliberations. Design Intent is conceived as a generic model for the entire design process. It is conceivable that, as experience with classes of design problems accumulates, the projects may contribute to the establishment of promising decompositions where they do not exist at the present time.

MICON also recognizes this difficulty and provides a mechanism to elicit refinement knowledge when needed from experts through its tutoring component. Thus, a knowledge base for automated refinements is built gradually over many problem runs.

Any refinement hierarchy represents explicitly the couplings between goals and subgoals or between goals and components. More global interactions across goals or components have to be handled by evaluations after selections have been made. We will call such global evaluation mechanisms that are invoked *after* decisions have been made *critics* in the following. In fact, we will see that almost no approach can do without these.

4.2.2 Form-Driven Strategies

Form-driven refinement strategies implement some form of part refinement through levels of abstraction. But in contrast to MICON, the refinement is driven more by given part decompositions than by behavior specifications, although the latter may be important for selecting appropriate refinements at the various levels. Global interactions again are handled by critics.

This strategy is typically employed when the problem specification consists mainly of *physical specifications* that suggest at least good guesses about appropriate classes of component configurations. An example are the physical specifications from which BRIDGER starts (required length, traffic volume, site characteristics).

Each part at some level of abstraction can of course be viewed as defining a goal or objective: that of finding a refinement into less abstract subparts that satisfy the specifications associated with it. The Design Intent approach therefore can also work for form-driven refinement strategies.

Grammar-based approaches are applicable to form-driven refinement when the refinements are less specification- than context-directed (where the context is the current state of the design). This becomes particularly easy when the representation on which the grammar rules work is appropriate for any abstraction level. For example, a rectangular solid representing a partition at one level of abstraction can be refined into an internal frame covered by plaster board on its two sides, where each refined component is again represented as a geometric solid. Some of the grammars that have been written in GENESIS can

be viewed in this way; another example are the decompositions employed by ABLOOS. These processes reflect very closely the progression through *scales* that characterizes e.g. building design.

Specifications that significantly influence refinements may become part of the context as attributes and thus become an integral part of the matching and replacement mechanisms employed by grammar-based systems; to find such a unified treatment of form and function attributes is the explicit goal of the solids grammar using spatial and functional attributes for structural design.

The difficulties that arise with the present strategy are similar to those mentioned in the preceding section. Knowledge about appropriate refinements must be accessible and pre-stored if the process is to be automated, which may restrict applications to routine situations or make interactions with designers mandatory. BRIDGER demonstrates an interesting third possibility: a system able to learn from its own experience.

4.3 Bottom-Up or Constructive Strategies

4.3.1 Function-Driven Strategies

Bottom-up or constructive strategies build the description of an artefact through a sequence of decisions *at the same level of abstraction*, which is usually the lowest level aimed at. This is obvious for form-driven approaches like the ones described in the next section that add components one-by-one. But the WRIGHT project (which is not an EDRC project) demonstrates that a bottom-up approach can also be function-driven [49, 50]. WRIGHT solves layout problems similar to LOOS, but employs a complementary strategy. Rather than constructing layouts incrementally by adding one component at a time and performing intermediate tests, it looks directly at the constraints to be satisfied and builds a solution by satisfying *one constraint at a time*. In order to be able to do this, it requires that all constraints be uniformly expressed as equations or inequalities in the form variables (in this case, the coordinates of the objects to be allocated) and that the structural alternatives that exist for solving any constraint be explicitly stored as a disjunct of constraints. The result is a *disjunctive constraint satisfaction problem* that is solved by selecting disjuncts and testing for compatibility using established numerical techniques. The problem formulation generated by WRIGHT is equivalent to a mixed integer non-linear programming (MINLP) problem. This suggests that MINLP approaches generally can be subsumed under the present strategy, a connection that we cannot pursue further in the present report.

In this strategy, the coupling between form and function variables is explicit and tighter than in any other strategy. Interactions are automatically taken into account by the constraint satisfaction technique employed. This is one of the advantages of the approach. A disadvantage is that it requires an explicit listing of all possible ways of satisfying the constraints up-front, which limits the types of constraints it is able to consider; in particular, constraints that may involve an arbitrary number of objects pose difficulties (see [51] for a more detailed comparison with the LOOS approach).

Another problem is generic to the bottom-up strategies: the combinatorial explosion of the search space as the problems involve larger numbers of objects or constraints. ABLOOS attempts to overcome this difficulty by dividing the overall problem into subproblems each of which can be solved by its constructive problem-solver. A similar approach is conceivable for WRIGHT in addition to the sophisticated look-ahead and sequencing strategies it uses to reduce the number of constraints that must be explicitly considered.

4.3.2 Form-Driven Strategies

Under this category fall the classical hierarchical or incremental generate-and-test approaches as employed by LOOS and other grammar-based systems described in preceding sections; a predecessor is DENDRAL [52]. A prerequisite is that solutions can be constructed incrementally, e.g. by adding one part at a time, and that intermediate evaluations can be carried out with some certainty (as guaranteed by the monotonicity properties of the LOOS generator).

Another, more fundamental prerequisite is that the constructions employed guarantee that every promising solution can, in principle at least, be generated; this may require a considerable amount of theory formation as indicated by both DENDRAL and LOOS. Since these mechanisms can be developed independent of the tests to be executed, the coupling between form and function variables can remain loose, which is one of the reasons for the generality or flexibility of the approach. On the other hand, the search space tends to explode with the number of objects to be handled and forces functional concerns to be considered as early and effectively as possible.

The solids grammar described in section 3.4 represents an attempt to develop a constructive strategy that is form- and function-driven to equal degrees by incorporating functional attributes more closely into the representation of form. This system is still in an experimental state.

4.4 Middle-Out Strategies

4.4.1 Function-driven Strategies

A function-driven middle-out strategy becomes possible when a highly-structured description of behavior is given that can be transformed into a logical or physical design by associating physical components with the elements in the behavior specification. "Highly-structured" means here that the behavior specification contains an explicit representation of the interactions between behavioral components, which allows the transformations to be governed mainly by local constraints or specifications and eliminates, in the ideal case at least, the need for global critics. Both SAW and the transformation project use a graph representation to this end: a Verilog description in the first case and a bond graph in the second case.

This strategy depends for its success on two rather stringent prerequisites. An appropriate behavior description must be given at the outset (as assumed by SAW) or be constructed as a first step in the design process. Equally important is the availability of behavior-preserving transformations, a decidedly non-trivial requirement as demonstrated by the transformation project.

4.4.2 Form-Driven Strategies

Form-driven middle-out strategies are conceivable, albeit not represented by any of the projects described in the present report. Instead of starting with a highly-structured behavior description, they would have to start with a highly-structured *form* description. Many of the prototype-refinement strategies discussed in the literature clearly fit this description (see [53]). They retrieve one or several solution *prototypes* from a database and, in the simplest case, adjust them to a given context or problem specification by "parameter tweaking"; that is, the structure of the prototype is preserved in contrast to the refinement strategies that gradually expand a structure through levels of abstraction and the bottom-up approaches that construct it from its constituent elements. ARCHPLAN, a component of the IBDE system developed at the EDRC, implements aspects of this approach [54].

Strategy	Problem specification	Prerequisites
Function-driven top-down strategies	High-level goal, function or beh. specification	Accessible knowledge about good refinements and form alternatives
Form-driven top-down strategies	(Almost) physical specifications	Accessible knowledge about form refinements and functional implications
Function-driven bottom-up strategies	Collection of simultaneous constraints in form variables	Explicit formulations of constraints
Form-driven bottom-up strategies	Parts to be configured Constraints/criteria to be satisfied	Constructive operators Partial evaluators of functional constraints
Function-driven middle-out strategies	Highly-structured behavior specification	Interactions that can be explicitly represented
Form-driven middle-out strategies:		
Prototype refinement	Collection of solution prototypes	Available prototypes Refinement/adjustment operators
Iterative improvement	Initial configuration Obj. function	Permutation operators

Table 2: Summary of design strategies

Iterative improvement strategies such as simulated annealing represent a second, distinct approach that starts with a highly-structured form description [55]. In contrast to prototype refinement, it permutes the structure itself and tests if these changes result in improvements. Iterative improvement shares with the constructive strategies the advantage that if it works at all, the distinction between routine and non-routine design problems becomes meaningless for practical purposes because it can find, in principle at least, *any* solution in the search space.

Methods for handling interactions	Difficulties	Projects
Local: Propagation of goals/ specifications/constraints Global: Critics	Access to refinement knowledge and form alternatives	Conceptual Design Design Intent MICON
Local: Propagation of specifications/constraints Global: Function critics	Access to refinement knowledge and functional implications	Design Intent Grammar Interpreters EDESYN, BRIDGER, ABLOOS
Constraint satisfaction techn.s Equation solvers	Finding expressions for all constraints Combinatorics of search space	WRIGHT MINLP
Local: Constraint propagation Global: Critics	Combinatorics of search space Reliability of intermediate tests	Grammar Interpreters LOOS
Iteration and prediction	Developing a beh. description Interactions between design decisions	Systems Architect's Workbench Transforming Beh./Structure
Critics	Creating data-base of prototypes	ARCHPLAN
Critics	Finding an appr. objective function	

Table 2 continued

4.5 Summary

Table 2 summarizes the classification suggested in the present section.

5 Challenges and Future Work

Aside from specific challenges faced by individual projects, several general challenges emerge from our survey. First and foremost is the challenge to test a model's applicability in the context of realistic, "industrial-strength" problems. This is less a challenge for the models developed for digital electronics design, which have for the most part passed that test. But the majority of projects described in this report are still in a highly experimental stage and have to be tested in the context of more realistic problems.

Another interesting test would be to apply particular models across domains, particularly those that have been successful in a specific domain. This suggests itself in particular for the digital electronics design systems, which have been developed for specific applications.

These two types of tests or experiments are likely to increase our understanding of the salient characteristics of the underlying models and their relations to the characteristics of generic design problems. The overview presented in the preceding section should only be seen as a first step towards a more detailed and substantive classification.

In conjunction with these investigations, issues of usability and integration of the capabilities offered by successful models into general design-support environment will have to be addressed: What are the respective roles of the human and computational agents involved and how can they evolve through use and time ? What are the desired types of interactions What "degrees of freedom" for specifying and manipulating computational models must be supported? Is there a generic set of interaction types, or are they idiosyncratic to a particular computational model, or will a mixture of both be needed?

References

1. Sullivan, L., "The Tall Office Building Artistically Considered," in *Louis Sullivan. The Public Papers*, Twombley, R., ed., Univ. of Chicago Press , Chicago, 1988, pp. 103-113.

2. Schnaidt, C., *Hannes Meyer: Buildings, Projects, and Writings*, Architectural Book Pub. Co., New York, 1965.

3. Colquhoun, A., "Typology and Design Method," *Arena*, Vol. 83, June 1967.

4. Steadman, P., *The Evolution of Designs: Biological Analogy in Architecture and the Applied Arts*, Cambridge University Press, New York, 1979.

5. Radford, A. D. and Gero, J. S., "Tradeoff Diagrams for the Integrated Design of the Physical Environment in Buildings," *Building and Environment*, Vol. 15, 1980, pp. 3-15.

6. Stiny, G., "A Note on the Description of Designs," *Environment and Planning B. Planning and Design*, Vol. 8, 1981, pp. 257-262.

7. Mitchell, W. J., *The Logic of Architecture. Design, Computation, and Cognition*, MIT Press, Cambridge, MA, 1990.

8. Woodbury, R. F., "Searching for Designs: Paradigm and Practice," *Building and Environment*, Vol. 26, 1991, pp. 61-73.

9. "Rainbow's end. The Hooker Office Building. Niagara Falls, NY", Progressive Architecture, April 1980.

10. Radford, A. D. and Gero, J. S., *Design by Optimization in Architecture, Building, and Construction*, Van Nostrand Reinhold, New York, 1988.

11. Finger, S. and Rinderle, J. R., "Transforming Behavioral and Physical Representations of Mechanical Designs," *Proceedings of the First International Workshop on Formal Methods in Engineering Design, Manufacturing, and Assembly*, Colorado State University, January 15-17 1990, pp. 133-151.

12. Alexander, C., *Notes on the Synthesis of Form*, Harvard University Press, Cambridge, MA, 1964.

13. Alexander, C., *The Timeless Way of Building*, Oxford University Press, New York, 1979.

14. Galle, P., "Alexander Patterns for Design Computing," *Environment and Planning B. Planning and Design*, Vol. 18, 1991, pp. 327-346.

15. Steinberg L., Langrana N., Mitchell, T., Mostow, J. and Tong, C., "A Domain Independent Model of Knowledge-Based Design," Technical Report AI/VLSI Project Working Paper No. 33, Rutgers U., March 1986.

16. Wirth, N., "Program Development by Stepwise Refinement," *Communications of the ACM*, Vol. 14, No. 4, 1971, pp. 221-227.

17. Fenves, S. J. and Baker, N. C., "Spatial and Functional Representation Language for Structural Design," *Expert Systems in Computer-Aided Design, Elsevier Science (North-Holland),IFIP 5.2*, 1987.

18. Rinderle, J. R., "Implications of Function-Form-Fabrication Relations on Design Decomposition Strategies," *Proc. Computers in Engineering*, American Society of Mechanical Engineers, Chicago, 1986, pp. 193-198.

19. Suh, N. P., Wilson, D. R., Tice, W. W., Yasuhara, M., and Bell, A. C., "Application of Axiomatic Design Techniques to Manufacturing," *Production Engineering Division, Winter Annual Meeting*, American Society of Mechanical Engineers, Washington, D.C., 1980.

20. Suh, N. P., *The Principles of Design*, Oxford University Press, Oxford, UK, 1988.

21. Gajski, D. D. and Kuhn, R. H., "Guest Editors' Introduction: New VLSI Tools," *Computer*, Vol. 16, No. 12, 1983.

22. Walker, R. A. and Thomas, D. E., "A Model of Design Representation and Synthesis," *Proceedings of the 22nd Design Automation Conference*, IEEE Computer Society and ACM-SIGDA, 1985.

23. Yount, C. R. and Siewiorek, D. P., "{SIDECAR}: Design Support for Reliability," *Proceedings of the 28th Design Automation Conference*, IEEE Computer Society and ACM-SIGDA, 1991.

24. Ganeshan, R., Finger, S. and Garrett, J., "Representing and Reasoning with Design Intent," *Proceedings of the First International Conference on Artificial Intelligence in Design*, Edinburgh, June 25-27 1991.

25. Heisserman, J., *Generative Geometric Design and Boundary Solid Grammars*, PhD dissertation, Department of Architecture, Carnegie Mellon University, Pittsburgh, PA, 1991.

26. Maher, M.L., and Fenves, S.J., "HI-RISE: An Expert System for the Preliminary Structural Design of High Rise Buildings," Tech. report R-85-146, Carnegie Mellon University, November 1984.

27. Rinderle, J. R. and Balasubramaniam, L., "Automated Modeling to Support Design," *Design Theory and Methodolgy - DTM '90*, Rinderle, J. R.,ed., American Society of Mechanical Engineers, Chicago, 1990, pp. 281-290.

28. Heintze, N., Michaylov, S. and Stuckey, P., "CLP(R) and Some Electrical Engineering Problems," Technical Report CMU-CS-89-139, Department of Computer Science, Carnegie Mellon University, 1989.

29. Miles, L. D., *Techniques of Value Analysis*, McGraw Hill, New York, 1982.

30. Bytheway, C. W., "The Creative Aspects of FAST Diagramming," *Proceedings of the SAVE Conference*, 1971.

31. Sturges, R. H. and Kilani, M. I., "A Function Logic and Allocation Design Environment," *Proceedings for ESD Fourth Annual Expert Systems Conference and Exposition*, Detroit, April 3-5 1990.

32. Pahl, G. and Beitz, W., *Engineering Design*, The Design Council, Springer-Verlag, London, 1984.

33. Subrahmanian, E., Podnar, G. and Westerberg, A., "n-DIM: n-Dimensional Information Modeling - A Shared Computational Environment for Design," Tech. report, Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA, 1989.

34. Birmingham, W. P., Gupta, A. P. and Siewiorek, D. P., *Automating the Design of Computer Systems -- The MICON Project*, Jones and Bartlett, 1992.

35. Thomas, D. E.; Lagnese, E. D.; Walker, R. A.; Nestor, J. A.; Rajan, J. V. and Blackburn, R. L., *Algorithmic and Register-Transfer Level Synthesis: The System Architect's Workbench*, Kluwer Academic Publishers, The Kluwer International Series In Engineering and Computer Science, 1990.

36. Thomas, D. E. (ed.), "The System Architect's Workbench User's Guide. Version 2.0," Research Report No. CMUCAD-91-42, Carnegie Mellon University, May 1991.

37. Thomas, D. E. and Moorby, P., *The Verilog Hardware Description Language*, Kluwer Academic Publishers, 1991.

38. Arnstein, Lawrence F., "Describing Systems for High Level Synthesis in the Verilog Language," Master's thesis, Carnegie Mellon University, December 1990.

39. Cloutier, R. J. and Thomas, D. E., "The Combination of Scheduling, Allocation, and Mapping in a

Single Algorithm," *Proceedings of the 27th Design Automation Conference*, IEEE Computer Society and ACM-SIGDA, 1990.

40. Paulin, P. G. and Knight, J. P., "Force-Directed Scheduling for the Behavioral Synthesis of ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 8, 1989, pp. 661-79.

41. Reich, Y., *Building and Improving Design Systems: A Machine Learning Approach*, PhD dissertation, Department of Civil Engineering, Carnegie Mellon University, Pittsburgh, PA, 1991.

42. Reich, Y., "Macro and Micro Perspectives of Multistrategy Learning," *Machine Learning: A Multistrategy Approach, Vol. IV*, Michalski, R. S. and Tecuci, G.,ed., Morgan Kaufmann, San Mateo, CA, 1992, pp. .

43. Reich, Y., "The Acquisition and Utilization of Basic Aesthetic Criteria in Design," *Artificial Intelligence in Engineering*, Vol. (accepted for publication), 1992.

44. Flemming, U.; Coyne, R.; Glavin, T. and Rychener, M., "A Generative Expert System for the Design of Building Layouts - Version 2," in *Artificial Intelligence in Engineering: Design (Proceedings of the Third International Conference, Palo Alto, CA)*, J. Gero, ed., Elsevier (Computational Mechanics Publications), New York, 1988, pp. 445-464.

45. Coyne, R.F. and Flemming, U., "Planning in Design Synthesis - Abstraction-Based LOOS," in *Artificial Intelligence in Engineering V. Vol 1: Design (Proceedings of the Fifth International Conference, Boston, MA)*, J. Gero, ed., Springer (Computational Mechanics Publications), New York, 1990, pp. 91-111.

46. Coyne, R.F., *ABLOOS: An Evolving Hierarchical Design Framework*, PhD dissertation, Dept. of Architecture, Carnegie Mellon University, Pittsburgh, PA, 1991.

47. Stefik, M. et al., "Basic Concepts for Building Expert Systems," in *Building Expert Systems*, Hayes-Roth, F. et al., eds., Addison-Wesley, Reading, MA, 1983, pp. 56-86, ch. 3.

48. Garrett Jr., J.H. and Fenves, S. J., "A Knowledge-Based Standards Processor for Structural Component Design," Tech. report R-85-157, Department of Civil Engineering, Carnegie Mellon University, Pittsburgh, PA, 1986.

49. Baykan C.A., *Formulating Spatial Layout as a Disjunctive Constraint Satisfaction Problem*, PhD dissertation, Dept. of Architecture, Carnegie Mellon University, Pittsburgh, PA, 1991.

50. Baykan C.A. and Fox M.S., "Constraint satisfaction techniques for spatial planning," in *Intelligent CAD Systems III Practical Experience and Evaluation*, ten Hagen, P.J.W. and Veerkamp, P.J., ed., Springer-Verlag, Berlin, 1991, pp. 187-204.

51. Flemming, U.; Baykan, C.; Coyne, R.F.; Fox, M., "Hierarchical Generate-and-Test vs. Constraint-Directed Search. A Comparison in the Context of Layout Synthesis," *Proceedings AID'92*, Gero, J.,ed., Kluwer Academic Publishers, 1992 [in press].

52. Buchanan, B.; Sutherland, Georgia and Feigenbaum, E.A., "HEURISTIC DENDRAL: a program for generating explanatory hypotheses in organic chemistry," in *Machine Intelligence 4*, Meltzer, B. and Michie, D., ed., Edinburgh University Press, Edinburgh, 1969, pp. 209-254.

53. Gero, J.; Maher, M.L. and Zhang, W., "Chunking Structural Design Knowledge as Prototypes," Working paper, The Architectural Computing Unit, Dept. of Arch. Science, University of Sydney, Australia, 1988.

54. Schmitt, Gerhard., "ARCHPLAN - An Architectural Planning Front End to Engineering Design Expert Systems," in *Expert Systems for Engineering Design*, Rychener, M. D., ed., Academic Press, New York, 1988, pp. 257-278.

55. Rutenbar, R.A., "Simulated Annealing Algorithms: An Overview," *IEEE Circuits and Devices*, January 1989, pp. 19-26.