

Process Models for Design Synthesis

Mary Lou Maher

Studies in design methodology provide various structured approaches to the design process. Many books provide definitions and elaborations of the design process: In the structural engineering field, such books include Holgate (1986) and Lin and Stotesbury (1981). More generally, various design methods and techniques are described in Alexander (1964) and Jones (1970). These design methods share the characteristic of prescribing a general set of tasks to be performed by the designer. One problem with design methodologies is that such approaches prescribe what a designer should do but not how. Human designers tend to not use such methodologies because they stifle creativity through a prescription of structured activities. There is a need to identify models of design processes that facilitate design rather than prescribe a design process.

Models of design processes provide guidance in the development of knowledge-based systems for design. The basis for such models comes from research in design theory and methodology as well as problem solving in AI. Three models are presented: decomposition, case-based reasoning, and transformation. Each model provides a formalism for representing design knowledge and experience in distinct and complementary forms.

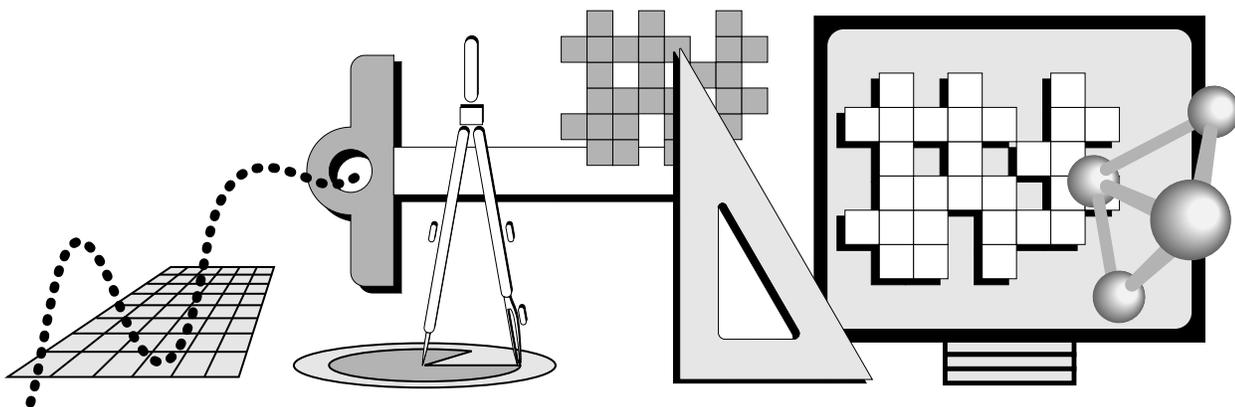
The use of AI techniques in developing design programs provides formalisms for representing problem-solving processes. In this article, the use of AI techniques in modeling design process-

es is elaborated with the presentation of three models that formalize the representation of design knowledge within a design program.

For the purpose of establishing the relevance of the models presented here, a structured approach to the overall process of design is adopted in which the design process comprises three phases: formulation, synthesis, and evaluation.

Design formulation involves identifying the requirements and specifications of the design problem. The result of design formulation is sometimes referred to as the design brief or program or, more simply, the definition of the design problem. *Design synthesis* includes

There is a need to identify models of design processes that facilitate design rather than prescribe a design process.



During design synthesis, the form of the design solution is identified.

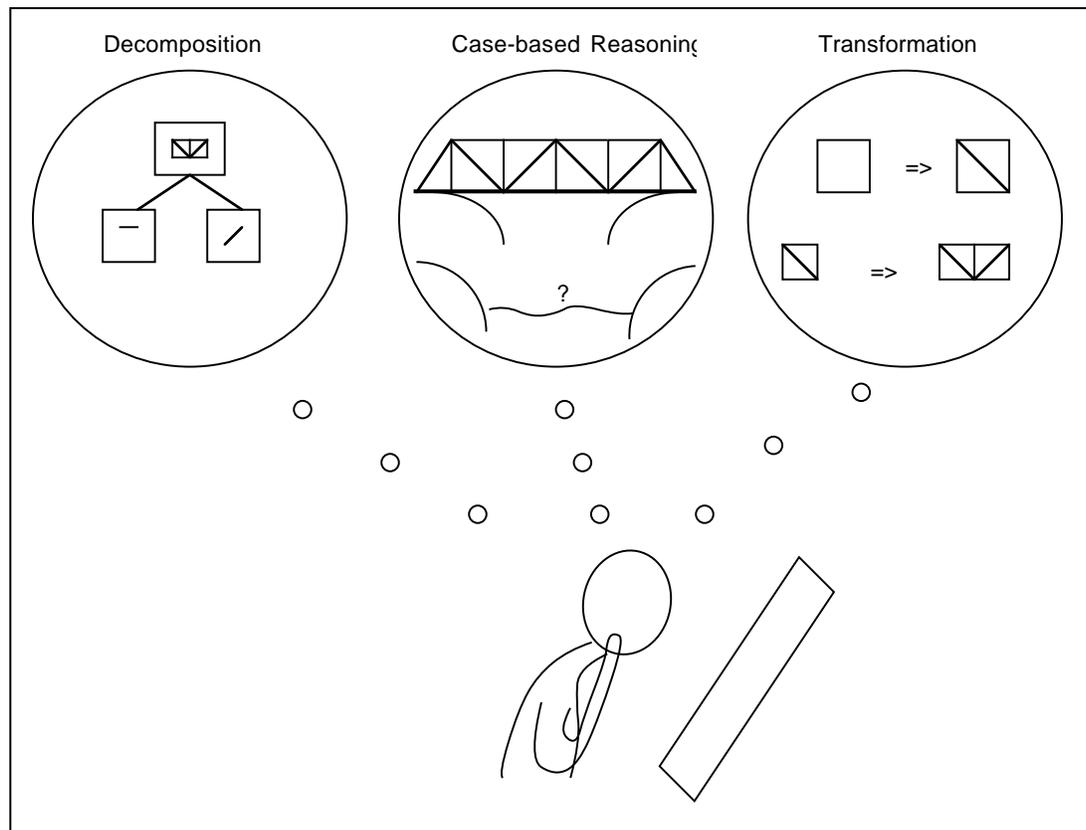


Figure 1. Three Models of Design Processes.

the identification of one or more design solutions consistent with the requirements defined during formulation and any additional requirements identified during synthesis. *Design evaluation* involves interpreting a partially or completely specified design description for conformance with expected performances. This phase of the design process often includes engineering analysis.

Although the phases might not be addressed in the order prescribed for the entire design process and are often carried out recursively, there is an inherent order in which designers approach a design problem. Typically, a designer starts with a definition of the design problem, identifies one or more potential design descriptions, and then evaluates the design. The variation occurs in the revisions of the requirements or descriptions and the iterations of the various phases.

The identification of different phases in the design process is a beginning for the formalization and understanding of design. What is missing from such methods is how each phase is completed. The use of a structured approach is acceptable for identifying various

design activities but not executing them. When a particular methodology does provide a procedure for executing a design activity, it becomes too constrained to be helpful in most design situations. Rather than prescriptions for the execution of design activities, it might be more helpful to identify descriptive models that allow reasoning and creativity within the formalized representation of design.

The focus in this article is on the synthesis phase. During design synthesis, the form of the design solution is identified. This phase of the design process is not well supported by computer-based tools unless the design problem can be formulated in mathematical terms; for example, optimization techniques are used during synthesis when the design problem can be formulated as an objective function and constraints. The recent use of knowledge-based systems for the synthesis of design descriptions has shown promise and forms the basis for the models described here. Experienced designers resort to trial and error less frequently than novice designers when they synthesize designs, suggesting that the use of knowledge-based systems to represent experi-

ence might aid in synthesizing designs. The major issue then is the explicit representation of design experience in a knowledge base.

During design synthesis, a designer considers a design space that contains the knowledge that is used to develop the design solution. A human designer does not need to explicitly identify his(her) design space; it is implicitly developed and expanded as s/he gains experience. A design program, however, does contain an explicit representation of the relevant design space. The nature of the knowledge in the design space must be explicit when considering a knowledge-based approach to design.

Simon (1969) presents design as a problem-solving process and, even more specifically, as a search process. The implication of search as a model for design processes is that design knowledge can be expressed as goals and operators. As a general approach to modeling design, search provides a formalism for expressing design knowledge; however, it does not directly address some of the intricacies and idiosyncrasies of design problems. Considering design as problem solving is a beginning to understanding and modeling design, but design problem solving has some additional characteristics that can be exploited by more explicit models.

It is interesting to consider the three phases of design as a search process within a design space: Design formulation involves identifying the goal(s) of the design problem. Design synthesis involves the search for one or more design solutions through the selection and application of operators. Design evaluation involves assessing whether the goal(s) have been satisfied.

Variations in both the goals and the state-space descriptions as the design process proceeds and the difficulty in predetermining the relevant operators are some of the issues that are not readily addressed in using search for solving design problems. The goals of the problem can change during the problem-solving process, which can indicate a different design space to be searched. One reason design has been difficult to implement as a search process is this change in problem definition during the problem-solving process. One way of dealing with this difficulty is to identify models of the design synthesis process, assuming that formulation has occurred. Another way is to allow synthesis to proceed even with a change of goals. The implication here is that using search as a model for the design process is too general; more specific models that use search in various ways are needed to bridge the gap between a model of

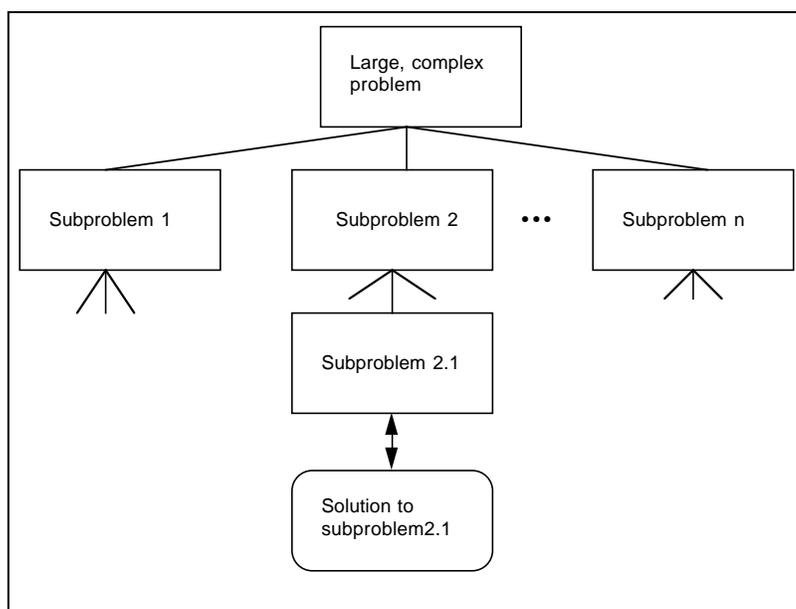


Figure 2. Decomposition Model.

design and the eventual representation of design knowledge and experience.

The use of AI techniques combined with research in design methodology provides an opportunity to exploit the results of both to produce an understanding of design problem solving. The early efforts in using AI techniques in design resulted in expert systems capable of designing specific artifacts using rule-based approaches. In many cases, the experience in developing rule-based design systems led to the generalization of design problem solving in which the knowledge base was no longer made up of unstructured rules. The justifications for this transition from an unstructured rule base to a design-oriented knowledge base included ease of knowledge acquisition and an increased understanding of design problem solving.

Three Models of Design Synthesis

In generalizing design problem solving, three distinct models of design synthesis can be identified: decomposition, case-based reasoning, and transformation (figure 1). These models are distinct in their associated formalisms for design knowledge. The models are not necessarily cognitive models; although they might match various approaches humans take when producing design solutions, the correspondence has not

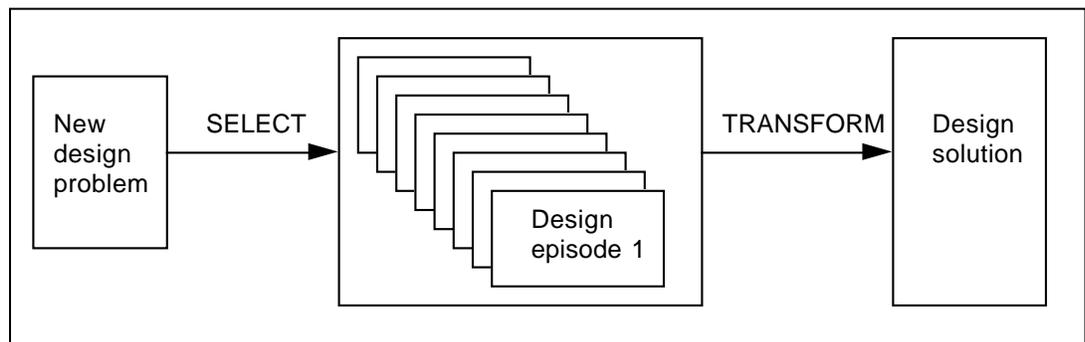


Figure 3. Case-Based Reasoning Model.

been adequately tested. The distinction among the models lies in the representation of design knowledge rather than in their appropriateness for a specific design domain or phase of design. The purpose of identifying more than one design model is in identifying appropriate formalisms for representing design knowledge.

Decomposition (figure 2) is perhaps the most ubiquitous model of design. It follows directly from the development of design methodology and has been shown to be useful in the development of knowledge-based design systems. The idea of dividing large complex problems into smaller, less complex problems is well accepted and practiced. It is possible to consider all models of design as some form of decomposition. When we consider a knowledge-based approach to representing design knowledge, the decomposition model has provided a clear position about the type of design knowledge needed. Specific knowledge-based systems for design by decomposition have been developed that identify specific languages for describing design knowledge. Examples of such languages include DSPL (Brown and Chandrasekaran 1985), Edesyn (Maher 1988), and Vexed (Steinberg 1987). The issues associated with this model include the appropriateness of decomposing and assuming that solutions to loosely coupled subproblems will combine to form a good design solution and the ease of specifying design knowledge in the domain as decomposition and recombination knowledge.

Case-based reasoning is a model of design that directly uses design experience in the form of episodes rather than compiles and generalizes it. The model (figure 3) uses analogic reasoning to select and transform specific solutions to previous design problems to

be appropriate as solutions for a new design problem. This model is attractive because the knowledge acquisition for developing generalized representations of design knowledge in a particular domain can be difficult and time consuming. The issues associated with using this model for design include the identification of the necessary information about a design episode to reason about its applicability in a different context, the meaning of a similar design, and the transformation of the solution from the original context to the new context. Although human designers appear to be good at using this type of analogy, it is difficult to automate it.

Transformation is a holistic approach to design, similar to case-based reasoning, but uses generalizations rather than specific episodes, like decomposition. In the transformation model (figure 4), the design knowledge is expressed as a set of transformational rules in which the left-hand side (LHS) of the rule is replaced by the right-hand side (RHS) of the rule. The most common application of the transformation model is manifested as grammars. The issues associated with using this model are the representation of the design description, the control in selecting an eligible transformational rule, and the termination of the application of rules.

In the following subsections, the definition and use of these models are elaborated. Currently, the models are ill defined, and many issues need to be resolved before they can become domain-independent formalisms defined in sufficient detail to be computer environments for knowledge-based design. In addition, examples are presented of systems that have been implemented that use one of the models. The purpose of the following subsections is to illustrate that such models of design are not domain dependent—the

examples are drawn from different domains—and that the models provide an understanding of the nature of the design knowledge that needs to be acquired to implement knowledge-based systems for design.

Decomposition

Decomposition by definition means that something is decomposed; it also implies a recomposition. The early expert systems for design synthesis implicitly used the decomposition model. Hi-Rise (Maher and Fenves 1984; Maher 1988), an expert system for the preliminary design of high-rise buildings, used both rule-based and frame-based representations of design knowledge. R1 (Xcon), an expert system for the configuration of computer systems, used a rule-based approach. In both cases, the representation of the design knowledge was dictated by the language in which the system was implemented: PSRL (Rychener 1984) in the case of Hi-Rise and OPS5 (Forgey 1981) for R1. These expert systems led to the generalization of the design approach to define a decomposition language for developing a knowledge base. Hi-Rise led to Edesyn (Maher 1988) and R1 to Salt (Marcus, Stout, and McDermott 1988). These two systems were not unique in this sense. Many efforts in developing design expert systems led to the development of languages for decomposition in the anticipation that this approach would facilitate the development of additional design expert systems and provide a formalism for representing design knowledge.

The identification of various languages for describing design knowledge has raised a number of issues in the application of the decomposition model to complex design problems. Several questions are raised when applying this model to the development of a knowledge base for design: What is decomposed? How is the problem decomposed? Is the decomposition fixed? How does recomposition occur?

What is decomposed? It is easy to say that the design problem is decomposed into subproblems, but what is the meaning of decomposition in design? There are at least two meanings to the decomposition of a design problem into subproblems: (1) to decompose a domain of design knowledge, say, structural design, into the various physical components that are used to construct design solutions, say, walls, slabs, and so on, or (2) to decompose into the various functions that must be provided for by a design solution, for example, resisting various types of load and providing open space, until a component can be

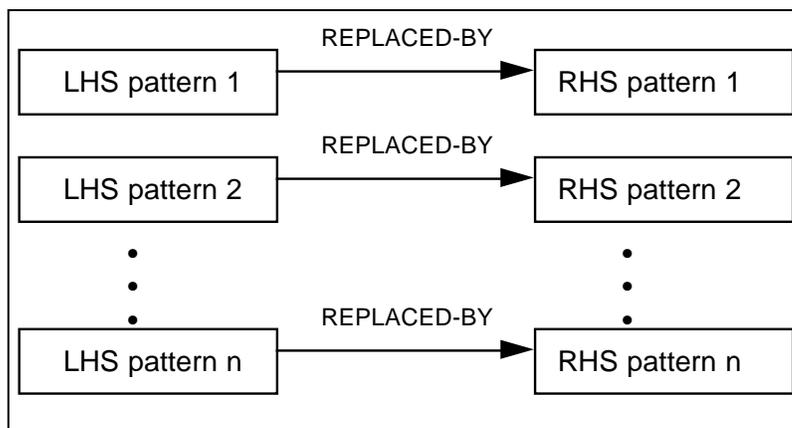


Figure 4. Transformation Model.

identified that will provide a specified function.

The first approach to decomposition is an object-centered approach in which the design knowledge is organized around the physical systems and components that a particular domain is concerned with. The second approach considers a functional decomposition to a sufficient level of detail in which a function to form mapping can occur.

The issue of function versus form as a basis for decomposition is not resolved yet. The point here is that design knowledge comprises various types of knowledge, for example, function and form, and recognizing these types of knowledge facilitates the formalization of the design process and, subsequently, the development of a solution. One approach to dealing with the decomposition of the function versus form dilemma is to ignore the distinction between function and form and identify a uniform representation for either function or form. In a search process approach to design, this uniform representation is typically called a *goal*. Another approach is to dictate a specific representation for functional decomposition and a specific representation for form decomposition.

How is the problem decomposed? The assumption behind decomposition is that each subproblem can be solved independent of the other subproblems. However, this assumption is an idealization rather than a reality for any problem definition. The rule of thumb in decomposition is to decompose into nearly independent subproblems or loosely coupled subproblems. Usually, this decomposition is easier said than done. However, because many have been successful in solving complex problems using decomposi-

tion, there must be something to it. A well-understood problem is easier to decompose than one that has not yet been explored.

Is the decomposition fixed? A fixed decomposition implies that it is not altered for a specific problem and is used without modification for all problems presented. At a general level of abstraction, a fixed decomposition might be possible. For example, it is useful to decompose design into formulation, synthesis, and evaluation at a general level. Such a decomposition is not as useful for a more detailed view of a specific design problem. As soon as a more detailed approach to design is required, a single decomposition is inadequate.

For example, the decomposition of the design of a structural system might be (1) design the lateral load resisting system, (2) design the gravity load resisting system, and (3) design the foundation. This decomposition makes sense when the structure is a building with more than five stories. Variations can occur when (1) the structure is a single-family residence, where the lateral load resisting system is insignificant and where the order of the decomposition might change but not the subproblems; (2) the structure is underground, where the lateral load resisting system cannot be considered separate from the gravity load resisting system, and the nature of the subproblems must be reconsidered; and (3) the structure is an offshore oil platform, where the foundation and the various structural systems for lateral and gravity loads are integrated. Once it has been determined that the decomposition should not be fixed, a representation that accommodates variations must be identified.

How does recomposition occur? The decomposition of a design problem into subproblems implicitly assumes that recomposition will occur. The fact that recomposition is implicit, rather than explicit, indicates that it is not an obvious process. Recomposition can occur implicitly; that is, by stating the solutions to the subproblems, the entire problem is solved. Recomposition can, however, introduce complications. Considering the subproblems as independent problems is only an idealization of reality. Putting the subproblems together must take into account the interactions. One way of representing the interactions is as constraints; the issue of recomposition then becomes one of constraint satisfaction.

The various computer environments that have been developed for design by decomposition answer each of these questions through the identification of a structured language or

syntax for describing a knowledge base. As an example, Edesyn provides two major representation structures for the representation of design knowledge: the system and the constraint. The representation of various systems allows the design knowledge base to explicitly represent the decomposition knowledge discretely, although the decomposition of any one system can vary as design proceeds. Edesyn provides a uniform representation for form and function, leaving the distinction to the knowledge base developer. The representation of various constraints allows the recomposition knowledge to be explicitly stated. What we gain from the implementation of the decomposition model as domain-independent computer environments for design is a better understanding of decomposition and its associated problems but, more importantly, the realization that models of design are not domain specific.

Case-Based Reasoning

Case-based reasoning in design involves the generation of a design solution using the solution or solution process from a previous design problem as a basis. This model of design synthesis requires episodes of design cases rather than generalizations about a design domain. Examples of a case-based reasoning approach to design include Struple (Zhao and Maher 1988) and Bogart (Mostow, Barley, and Weinrich 1989). Struple uses a database of building design solutions, finds partial matches to a new design problem, and provides a set of relevant design components for the solution of the new design problem. Bogart uses a library of circuit design plans, allows the user to select a relevant previous design, and replays the previous design plan for a new design problem. These two systems are based on previous efforts in developing decomposition languages for design (Edesyn in the case of Struple and Vexed in the case of Bogart) and the need to more directly use experience because the knowledge acquisition of generalized decompositions was proving to be difficult.

As a process model, case-based reasoning involves several operations. One set of operations is (Darpa 1989) (1) recall relevant cases from case memory, (2) select the most promising case, (3) construct a solution for the new problem, (4) test the solution, (5) evaluate the results, and (6) update case memory by storing the new case.

In design synthesis, the operations of most interest are centered on case retrieval, selec-

tion, and modification. With the assumption that case memory is large (that is, many design cases are stored), retrieval becomes a search problem in a large space. The selection of a case among potential cases requires recognition of the relevance of each case and how close the case is to providing a solution to the design problem. The modification of the case for the new design problem raises the issues of what should be changed and what should stay the same. It can be assumed that the changes are based on the results of a partial match, where the features of the case that did not match should be changed, but such local changes might not be sufficient.

There are guidelines for the development of a case-based reasoning system, regardless of whether the problem is design, planning, or diagnosis, as in the following:

Case memory organization: The extremes in representing cases in memory are to represent each case in its entirety (Reisbeck 1988; Stanfill and Waltz 1986; Koton 1988; Hammond 1986; Rissland and Ashley 1988) or break each case into pieces (Carbonell 1983; Kolodner 1988; Hinrichs 1988). When breaking a case into pieces, there are various approaches, such as conceptual clustering (Fischer 1988) and discrimination networks (Feigenbaum 1963).

Indexing: The most obvious way to index cases is to use appropriate features as indexes. The selection of a set of indexes can be fixed, which is not flexible, or the selection can be based on inductive-learning methods to identify predictive features (Lebowitz 1987) or explanation techniques (Mark and Barletta 1987) or to define a vocabulary associated with a task-oriented approach to problem solving (here, design and planning are examples of tasks) (Hammond 1986).

Retrieval algorithms: There are basically two guidelines in the development of a retrieval algorithm: concept refinement and parallel search. *Concept refinement* assumes that the structure of case memory is hierarchical in nature. The more general features or shared features of various cases are searched first, eliminating large amounts of cases according to these shared features. There are examples in which cases were stored in entirety using redundant discriminate networks (Kolodner 1983; Lebowitz 1983) and in pieces (Kolodner 1988). Parallel implementations use multiple processors so that every case in memory can be simultaneously checked (Stanfill and Waltz 1986).

Selection of the best case: A weighted count of matching features provides one way to select the best case; however, this approach

Case-based reasoning in design involves the generation of a design solution using the solution or solution process from a previous design problem as a basis.

does not take into account that the case itself might determine the importance of a feature. Some approaches to finding the best case are preference heuristics (Kolodner 1988), dimensional analysis (Rissland and Ashley 1988), and dynamically changing weighted evaluation functions (Stanfill 1987).

These guidelines provide a basic understanding of case-based reasoning but do not directly address the issues of case-based reasoning as a model for design synthesis. When storing design episodes as cases, the content, as well as the organization, of a case must be considered. Most design solutions are stored as descriptions of the physical object or system; most commonly, this description is in the form of drawings or geometric models. The synthesis of a design solution for a new problem starts with a set of specifications and requirements. These specifications do not solely consist of geometric information; so, unless the cases include information about the intended function, behavior, or performances of the design solution, retrieval of specific design cases can be distorted and, therefore, not provide a useful basis for the new design. This need is partly addressed by the identification of a representation of design episodes that includes function, behavior, and performance as well as geometric descriptions.

There is also the issue of storing the design solution or the operators used to produce the design description. The advantage to storing the design solution is that many existing cases can be used immediately, augmenting the geometric description with the relevant functions, and so on. The disadvantage is that transforming the old solution to fit the new problem is difficult. The alternative, storing the operators, allows the transformation to the new problem to be an execution of the old solution operators using the new problem statement.

The use of case-based reasoning for design synthesis assumes that the design knowledge is represented in the form of design episodes, defining the type of knowledge needed. The

representation of design episodes requires a formalism that includes geometric description, function, and behavior, so that reasoning about the transformation from the old solution to the new design problem is possible. As a model, case-based reasoning has been explored in many areas, and algorithms and specific programs have been developed. The implications of using this model in design have not been addressed as extensively as with the decomposition model.

Transformation

The transformation model follows a theoretical approach to design in which the initial set of design requirements is transformed into a design solution. The transformation model begins to approach the issue of how transformation occurs. Because grammars provide a formalism for the transformation model, they are used here to define the model. A general rule-based system shares many characteristics of a transformation model, the distinction being a subtle one. In rule-based systems, a predefined control mechanism determines which rule will execute, and there are fewer restrictions on the use of a rule's LHS and RHS. In a transformation model or, more explicitly, in a grammar, there is no explicit control mechanism, and the rules are considered rewrite rules in which LHS is replaced by RHS.

A grammar is more commonly associated with language rather than design or problem solving. Understanding grammars and their application to design requires considering design knowledge as a language and legal design solutions as legal statements in a language. In this sense, we can consider a grammar to be a formalism of design knowledge that can be used to generate a set of legal design solutions or, alternatively, to check whether a design solution is legal. The term legal is being used here in the broadest sense, meaning that a design solution or statement conforms with the formal definition of the language.

The properties of a grammatical formalism were explored by Chomsky (1957). A formal definition of a grammar is

$$G = \{N, T, P, S\} ,$$

where N is a set of nonterminal symbols, T is a set of terminal symbols, P is a set of productions or rewrite rules, and S is a special symbol called the start symbol.

The most well-known and successful type of grammar used in design is the shape grammar. *Shape grammars* use symbols that are based on shapes made up of points and lines. The formal definition and basic properties of shape grammars have been clearly defined

and applied by Stiny and Gips (Gips 1975; Stiny and Gips 1978; Stiny 1980). The application of shape grammars to architectural design has illustrated the ability of shape grammars to formally represent generative design knowledge and capture design style. The most notable applications include grammars that characterize the design of Frank Lloyd Wright's prairie houses (Koning and Eizenberg 1981), Palladian villa plans (Mitchell and Stiny 1978), and Queen Ann-style houses (Flemming 1987).

The considerations in using a grammar to represent design synthesis knowledge include (1) the definition of the terminal and nonterminal symbols and (2) the identification of productions. The definition of the terminal and nonterminal symbols is usually based on a formal representation of shape. The identification of productions provides the domain knowledge, where the productions represent design transformations associated with a specific design domain, for example, the design of rigid-frame structural systems, or a design style, for example, prairie houses.

Shape grammars and their application provide an example of how grammars can be used to manipulate shape. Other types of grammars address other aspects of design. In some design domains, it is not sufficient to generate design alternatives on the basis of shape alone. The types of grammars that have been identified include graph grammars and attribute grammars (Mullins and Rinderle 1990).

The attractive aspects of a grammatical approach to design are the ability to represent a design space without enumerating the possible design solutions and the formal basis for representing design knowledge. The difficulty in the practical application of grammars to design lies in the definition of a vocabulary that is expressive enough to capture design knowledge. The major barrier to the application of grammars to engineering design is the lack of a formal basis for representing function. Most grammars represent geometry in the form of shapes or geometric models. There are a few experiments with representing both function and geometry in a grammar. Fenves and Baker (1987) define a grammar that considers spatial transformations and an additional grammar that adds context or function to the generated shapes. Finger and Rinderle (1990) describe a bond graph approach to defining a grammar that captures behavior and geometry of mechanical systems.

Although the consideration in developing a grammar for representing design knowledge is defining the vocabulary and the rules, the

The most well-known and successful type of grammar used in design is the shape grammar. Shape grammars use symbols that are based on shapes made up of points and lines.

difficult questions for the practical use of grammars as both a representation and an implementation as a computer program are, How can function be formally represented? and How is rule execution controlled? The first question is still open, although there are some examples of including function in grammars. With the second question, the resulting system is usually made interactive (that is, the user or designer chooses which eligible rule is executed), or an inference mechanism similar to those used in rule-based expert system languages is used.

Conclusion

Three models of the process of design synthesis were presented: decomposition, case-based reasoning, and transformation. The value in identifying multiple models of design lies in the richness of the representation of design knowledge and experience provided by each and in the ability to choose a model that more closely fits the knowledge readily available for the domain being considered. Although each model can be implemented as a computer environment on its own, there is an advantage when the three models can be interchangeably used, the use of a model depending on available knowledge. For example, design decomposition can provide an overall model for design, where each system-subsystem can be designed using case-based reasoning when a relevant case is available, a set of transformations when a grammar is available, or further decomposition when the generalized decomposition is understood. Before such an integrated environment is possible as an implementation, many of the questions and issues raised by the implementations of each model need to be resolved.

Acknowledgment

This work was supported by the Engineering Design Research Center at Carnegie-Mellon University and the National Science Foundation Design Theory and Methodology program.

References

- _. 1989. Case-Based Reasoning, DARPA: Machine Learning Program Plan. In *Proceedings of the DARPA Workshop on Case-Based Reasoning*, 1–13. San Mateo, Calif.: Morgan Kaufmann.
- Alexander, C. 1964. *Notes on the Synthesis of Form*. Cambridge, Mass.: Harvard University Press.
- Brown, D., and Chandrasekaran, B. 1985. Expert Systems for a Class of Mechanical Design Activity: Knowledge Engineering. In *Computer-Aided Design*, ed. J. Gero, 259–283. Amsterdam: North-Holland.
- Carbonell, J. G. 1983. Derivational Analogy and Its Role in Problem Solving. In *Proceedings of the Third National Conference on Artificial Intelligence*, 64–69. Menlo Park, Calif.: American Association for Artificial Intelligence.
- Chomsky, N. 1957. *Syntactic Structures*. Mouton & Co.
- Feigenbaum, E. A. 1963. The Simulation of Verbal Learning Behavior. In *Computers and Thought*, eds. E. A. Feigenbaum and Feldman, 297–309. New York: McGraw Hill.
- Fenves, S., and Baker, N. 1987. Spatial and Functional Representation Language for Structural Design. In *Expert Systems in Computer-Aided Design*, ed. J. Gero, 511–529. Amsterdam: Elsevier.
- Finger, S., and Rinderle, J. R. 1990. Transforming Behavioral and Physical Representations of Mechanical Designs. In *Proceedings of the First International Workshop on Formal Methods in Engineering Design, Manufacturing, and Assembly*, 133–151.
- Fischer, D. H. 1988. A Computational Account of Basic Level and Typicality Effects. In *Proceedings of the Seventh International Conference on Artificial Intelligence*. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.
- Flemming, U. 1987. More Than the Sum of Parts: The Grammar of Queen Anne Houses. *Environment and Planning B. Planning and Design* 14.
- Forgy, C. L. 1981. OPSS User's Manual, Technical Report, CMU-CS-81-135, Dept. of Computer Science, Carnegie-Mellon Univ.
- Gips, J. 1975. *Shape Grammars and Their Uses*. Stuttgart: Birkhauser Verlag.
- Hammond, K. 1986. CHEF: A Model of Case-Based Planning. In *Proceedings of the Fifth International*

- Conference on Artificial Intelligence. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.
- Holgate, A. 1986. *The Art in Structural Design*. Oxford: Oxford University Press.
- Hinrichs, T. R. 1988. Towards an Architecture for Open World Problem Solving. In *Proceedings of the DARPA Workshop on Case-Based Reasoning*. San Mateo, Calif.: Morgan Kaufmann.
- Jones, J. C. 1970. *Design Methods*. London: Wiley Interscience.
- Kolodner, J. L. 1988. Retrieving Events from a Case Memory: A Parallel Implementation. In *Proceedings of the DARPA Workshop on Case-Based Reasoning*. San Mateo, Calif.: Morgan Kaufmann.
- Kolodner, J. L. 1983. Towards an Understanding of the Role of Experience in the Evolution from Novice to Expert. *International Journal of Man-Machine Studies* 19.
- Koning, H., and Eizenberg, J. 1981. The Language of the Prairie: Frank Lloyd Wright's Prairie Houses. *Environment and Planning B*(8): 295–323.
- Koton, P. 1988. Reasoning about Evidence in Causal Explanations. In *Proceedings of the DARPA Workshop on Case-Based Reasoning*. San Mateo, Calif.: Morgan Kaufmann.
- Lebowitz, M. 1987. Experiments with Incremental Concept Formation: UNIMEM. *Machine Learning* 2(2): 103–138.
- Lebowitz, M. 1983. Generalization from Natural Language Text. *Cognitive Science* 7(1).
- Lin, T. Y., and Stotesbury, S. D. 1981. *Structural Concepts and Systems for Architects and Engineers*. New York: Wiley.
- Maher, M. L. 1988a. Engineering Design Synthesis: A Domain-Independent Representation. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing* 1(3): 207–213.
- Maher, M. L. 1988b. HI-RISE: An Expert System for Preliminary Structural Design. In *Expert Systems for Engineering Design*, ed. M. Rychener, 37–52. San Diego, Calif.: Academic.
- Maher, M. L., and Fenves, S. J. 1984. HI-RISE: A Knowledge-Based Expert System for the Preliminary Structural Design of High-Rise Buildings, Technical Report, R-85-146, Dept. of Civil Engineering, Carnegie-Mellon Univ.
- Marcus, S.; Stout, J.; and McDermott, J. 1988. VT: An Expert Elevator Designer That Uses Knowledge-Based Backtracking. *AI Magazine* 9(1): 95–114.
- Mark, W., and Barletta, R. 1987. Case-Based Reasoning in Manufacturing, Lockheed AI Center, Palo Alto, Calif.
- Mitchell, W., and Stiny, G. 1978. The Palladian Grammar. *Environment and Planning B*(5): 5–18.
- Mostow, J.; Barley, M.; and Weinrich, T. 1989. Automated Reuse of Design Plans. *Artificial Intelligence in Engineering* 4(4): 181–196.
- Mullins, S., and Rinderle, J. R. 1990. Grammatical Approaches to Design. In *Proceedings of the First International Workshop on Formal Methods in Engineering Design, Manufacturing, and Assembly*, 42–69.
- Reisbeck, C. K. 1988. An Interface for Case-Based Knowledge Acquisition. In *Proceedings of the DARPA Workshop on Case-Based Reasoning*. San Mateo, Calif.: Morgan Kaufmann.
- Rissland, E. L., and Ashley, K. D. 1988. Credit Assignment and the Problem of Competing Factors in Case-Based Reasoning. In *Proceedings of the DARPA Workshop on Case-Based Reasoning*. San Mateo, Calif.: Morgan Kaufmann.
- Rychener, M. D. 1984. PSRL: An SRL-Based Production-Rule System, Reference Manual, Dept. of Computer Science, Carnegie-Mellon Univ.
- Simon, H. A. 1969. *The Sciences of the Artificial*. Cambridge, Mass.: MIT Press.
- Stanfill, C. 1987. Memory-Based Reasoning Applied to English Pronunciation. In *Proceedings of the Sixth National Conference on Artificial Intelligence*. Menlo Park, Calif.: American Association for Artificial Intelligence.
- Stanfill, C., and Waltz, D. 1986. Toward Memory-Based Reasoning. *Communications of the ACM* 29(12): 1213–1228.
- Steinberg, L. 1987. Design as Refinement Plus Constraint Propagation: The VEXED Experience. In *Proceedings of the Sixth National Conference on Artificial Intelligence*. Menlo Park, Calif.: American Association for Artificial Intelligence.
- Stiny, G. 1980. Introduction to Shape and Shape Grammars. *Environment and Planning B*(7): 343–351.
- Stiny, G., and Gips, J. 1978. *Algorithmic Aesthetics*. Berkeley, Calif.: University of California Press.
- Zhao, F., and Maher, M. L. 1988. Using Analogical Reasoning to Design Buildings. *Engineering with Computers* 4:107–119.



Mary Lou Maher is currently a senior lecturer in the Department of Architectural and Design Science and deputy director of the Key Centre of Design Quality at the University of Sydney. She is on leave from Carnegie-Mellon University, where she was involved

with the Synthesis Lab at the Engineering Design Research Center.