(ARTIFICIAL, INTELLIGENT) ARCHITECTURE: Computers in Design

BY THOMAS P. MORAN

For a profession that is creative in most fields, architecture has been unimaginative in its use of the computer, which it has relegated to trivial design tasks. As a result, the present research with computers by architects has done virtually nothing to improve our design methods or to deepen our understanding of the architectural design process. There has been too little examination of the design process itself as an intelligent activity —or of the computer as an intelligent device. Artificial Intelligence (AI) is a field, mostly within computer science, which explores the mechanisms of intelligent behavior. AI ideas and techniques will have to be applied to the design process if design and the computer are to be brought together into an intelligent and productive union.[1] I believe that this can be done.

There are two recent books—one about computers and one about design methodology—which I will discuss as a vehicle for explaining how computers are now being misused and how I think they should be used in the future. The striking contrast between these books will serve to highlight several basic issues regarding the style and content and the methods and goals of design research.

I. ARTIFICIAL ARCHITECTURE

The first book is *The Architecture Machine*, by Nicholas Negroponte.[2] It is a significant book in that it is directed at architects, it has been well reviewed, and apparently it is now widely read. But it is for these reasons that it must be criticized, and some alternatives suggested. I believe *The Architecture Machine* presents a superficial and McLuhanesque world of architecturalized computers, which obscures rather than clarifies the critical issues, and hinders productive research.

Program Learning

The author has built a working system, called URBAN5; and the book is based on his experiences with it. URBAN5 is an interactive system which converses with a designer about a spatial design project. URBAN5 keeps all the records of the design and displays them to the designer. He can request pictures from certain viewpoints, can add or delete surfaces and chunks of space, can assign properties to these elements and specify relations between them, can define new properties, and can call for calculations such as circulation distances or material accounting. URBAN5 checks for conflicts between the relations and properties the designer specifies (i.e.—his problem requirements) and the physical elements he specifies (i.e.—his attempted solution), and it reports the discrepancies to the designer. The basic idea behind URBAN5 is quite sensible. The fact that URBAN5 only works with limited design issues, such as circulation distances, can be forgiven if URBAN5 leads us toward the development of systems that can handle significant design issues.

What I would really like to know is *how* URBAN5 works—what makes it tick: how does it represent the design problem, the constraints, and the forms; what are the basic routines which operate on these items; how are these routines organized into the system; what are the weak spots in the system; what are the limits of its organization? But it is just these kinds of things that the book does not reveal. The important issue here is how can I, as another researcher, benefit from URBAN5—learn from its successes and shortcomings.

But the book's basic strategy is to *avoid* the problems of designing the mechanisms for making machines intelligent. Instead, it seems to concentrate on the interfaces between the machine and the physical world and hope that somehow intelligence will seep in. Thus the book says:

". . . the design process, considered as evolutionary, can be presented to a machine, also considered as evolutionary, and a mutual training, resilience, and growth can be developed."

In other words, if you can't learn from your programs how to make better programs, then let your programs "learn" themselves. What URBAN5 lacks, the author contends, is "evolution"; and he proposes a new machine—the Architecture Machine (AM)—to be the evolutionary successor to URBAN5.

Architecture Machines

The Architecture Machine (AM) is a little computer which is dedicated solely to interacting with its own personal architect. This AM is supposed to be "intelligent" so that it can engage in an evolutionary dialogue with its architect; both man and machine will be able to respond to the other.

The book contends that an AM must not merely consist of a multiplicity of specific design services (as did URBAN5), but that it should be "adaptable," i.e., it should be a "general" machine which can convert itself into a special-purpose device in response to a given situation. Since the AM is coupled to only one designer, it may acquire a distorted picture of the world. Hence it must observe the world directly itself. "It must see, hear, and read, and it must take walks in the garden." It should be able to communicate in English as well as other symbolic languages and in sketches as well as graphic codes; it it should have eyes and ears and arms and legs.

Computo-Humanism

I find rather incomprehensible the romantic notion that the computer should directly sense the environment so that it can get an "objective" view of it. Either the computer observes what we tell it to or, if it is really intelligent, it will have biases of its own; and who is to say its prejudices are preferable to ours?

This pseudo-humanism is further manifested in the book's tendency to anthropomorphize the computer. Not only are humanoid robots going to walk around analyzing the environment, but they will communicate with us at the most subtle levels of language and gesture. E.g., the book bemoans the fact that computer graphics "will not yield the same textural feeling as graphite on paper." The point, of course, is that humanism does not imply human-like machines. This is not to say that the goal of easy and fluid interaction between man and computer is not desireable. The computer is humane when it is used in a socially responsible and productive way—regardless of whether it looks like a machine or not.

Goals and Values

The book expresses the attitude (which I find strange for a text that supposedly espouses machine intelligence) that computers are only adept at handling small details. The human sees a rolling beach while the computer can only see a multitude of sand pebbles, it says. But the computer's alleged "pebble-prejudice" is really the book's own pebble prejudice, and it is a very narrow point of view. It unduly constrains what we can achieve with the computer and, even worse, it leads to grave errors.

Pebble prejudice leads the book into a disastrous confusion in goal structures —global goals and local goals. The book talks about chess-playing programs and says that the global goal of the game, to capture the opponent's king, "has little bearing on

Thomas P. Moran received his bachelor of architecture from the University of Detroit, and studied architectural sciences for two years at Cornell University. His principal interest now is the science of artificial intelligence as applied to the psychology of visual thinking, and as applied to design. He is nearing completion of his PhD in computer science at Carnegie-Mellon University in Pittsburgh.

1. See Herbert Simon. *The Sciences of the Artificial*. M.I.T. Press, 1969.

2. Nicholas Negroponte. *The Architecture Machine*. M.I.T. Press, 1970.

If we are able to clearly understand what we are doing, (design method) and what we are dealing with (the environment) then the computer can be an invaluable aid.

the local actions and the skillfulness of making these moves." It goes on to say

". . . the chess analogy suggests that a machine could learn to play architecture from local design pursuits. . . . With a history of local punishments and rewards, an adaptable machine can evolve without a global set of values and adaptable rules to achieve them."

With regard to values, this suggestion is questionable. Human values and value judgments cannot be swept under the rug. Values underlie all our decisions. Every program we write carries a statement of value. We must face up to this fact. Humane values can be advanced only by our constant attention to them. Any attempt to get around these difficult issues of value (especially by neglect) can only lead to the degradation of our values.

With regard to achieving goals, the book's suggestion seems technically wrong. To assume that global goals can be achieved by merely attending to details is wishful thinking. There is a whole hierarchy of goals embedded, at least *implicitly*, in any program of action, including chess programs. Current chess programs are, in fact, severely limited by their inability to *explicitly* deal with their own goal structures, and so they appear to only deal with local information. But this is one of their weak aspects, and it should not be made into a virtue to be imitated.

What is even worse for the book's argument is the fact that none of the current chess programs *learn* how to play; all their abilities are meticulously built in by their programmers. The chess analogy damages, rather than supports, the case for the AM. Even if there were some basis for believing that the AM could "learn architecture," the book's presentation of learning is completely inappropriate. It describes the AM in terms of such antiquated behavioristic psychology concepts as "a rote apparatus," "a conditioning device," "a reward selector." This reward-punishment type of learning is inadequate for anything but the simplest discrimination tasks (e.g., rats running mazes). For someone to learn complex and subtle issues (like those in architecture) requires complex symbolic interaction—description, explanation, discussion. "Yes/no" is not a rich enough vocabulary. No man or machine is going to "learn architecture" by being given candy and electric shocks!

Computer Maps

As a last example of the pitfalls of pebble prejudice, the book says that "in architecture, most information has a natural disposition—the positional relationship—which can help to organize the proliferation of data." So, what it suggests is the representation of a physical urban area in a computer as a matrix which overlays the area, with all the information about any specific location stored in the corresponding cell of the matrix, i.e., a map in the computer. "The thrust of this sort of data structure argument is that information is treated locally, by positions, and less globally, by attributes."

This characterization of maps is rather over-simplified. Maps are useful devices, of course; but they are only good for representing *static* and location-*specific* kinds of information. This is bulky and tedious data, and it could present an obvious challenge to the speed of the computer. There are now many researchers developing computer systems for the efficient handling of such maps, and some of these are very worthwhile projects. But this kind of information is superficial and does not begin to characterize what architecture is about.

To gather facts together onto a map is to "organize" those facts only in the most trivial sense. Planning maps of a city portray very little of the environmental structure of that city. The really interesting information to get at is that which goes through a professional planner's mind when he is looking at these maps. Until we are willing to come to grips with the abstract concepts and the general principles of environmental design and to formulate them so that we can explicitly deal with them, our computer-oriented research will continue to be merely exercises which yield nothing to the intellectual development of architecture.

II. INTELLIGENT ARCHITECTURE

You will have observed, no doubt, that very little has been said about architecture or design methods. This is exactly the problem! All the romantic fantasies about what wonderful things the computer is going to do for us in design have to be put aside. The real problems lie within architecture and design itself. The computer is not a panacea, nor is it just another gimmick. If we are able to clearly understand what we are doing (design method) and what we are dealing with (the environment), then the computer can be an invaluable aid. But if we are unclear about where we are going, the computer will lead us its own way.

In order to focus on the issues of design itself, I would like to discuss the *Pattern Language* developed by Christopher Alexander and his associates at the Center for Environmental Structure (CES). Their report[3] is about a specific type of design method as applied to a particular building type. It says nothing about computers, but their results can be used to suggest an avenue of research into the application of computers to design. My discussion of their work will concentrate on its methodological aspects.

Let me first state that I am in complete agreement with Alexander's basic tenet that it is the structure of the environment that we must understand. ("A form has a definite, substantial, functional structure," whose inner nature must become clear to us.[4]) If we can do this, the use of the computer will follow quite naturally, as I hope to show. Actually, the CES group does not advocate the use of the computer in connection with their work; they feel that it will detract from the central issues (e.g., environmental structure) and purposes (better design) of their work. I believe that their studies are the most advanced empirical work with a solid methodological commitment in the whole field of design theory. If the computers in design are ever to be pulled from their present rut, then they must participate in the development of theory and method. They can then contribute to the precision and refinement of the theory and provide practical service in the use of the method.

Pattern Language

As designers, we are swamped with information. The question is whether it is useful information and whether it is in the right *form* to be most useful in the design process. It is to this issue that the pattern language addresses itself. In general, design can be seen as the process of converting a set of abstract requirements into a concrete physical form which realizes them. Given a specific requirement, there are probably many spatial configurations which will satisfy that requirement. If some of them are known, they should be recorded. Such a record is a "pattern." An organized set of patterns concerning a specific type of object (e.g., a building type) constitutes a "pattern language" which is capable of "generating" all the instances of that type of object.

3. Christopher Alexander, Sara Ishikawa, Murray Silverstein. "A Pattern Language Which Generates Multi-Service Centers." Center for Environmental Structure (2531 Etna St., Berkeley, Cal.), 1968. (CES has a more recent report: "Houses Generated by Patterns." 1969.)

4. Christopher Alexander. "The Question of Computers in Design." *Landscape*, vol. 14, no. 3, 1965.

If we are unclear about where we are going, the computer will lead us its own way.

The CES report examines a building type called a "multi-service center"—a building housing many diverse kinds of community services. The choice of a new, and hence, nebulously defined, building type was good since its fundamental structure had to be consciously thought out. The expression of the results of this thinking provided a test for the appropriateness of the pattern language. The results are 64 pattern rules which (the authors feel) capture the functional essence and structure of the multi-service center in terms of reusable, easily applicable, "generic design ideas". The bulk of their report is a complete listing of these pattern rules. The patterns deal with general features of the building, its relation to its site, the physical and functional subparts of the building and their interrelations. Some of the patterns are very general and apply to almost any building type. I will not review the patterns here, but instead will focus on one particularly simple pattern [5] in some detail.

Pattern Rule

Imagine the situation where a sidewalk runs alongside a building. One desirable goal is that people on the sidewalk stop at the building and look in and become familiar with what's inside (and then maybe enter). But people on sidewalks have a tendency to keep moving even though they may be curious to stop and look. They have a conflict of tendencies. The way to resolve this conflict is to place "street-niches" in the building along the sidewalk so that people can step off the sidewalk into them and linger. A street-niche should be level with the sidewalk, should be at least five feet deep, and should contain a display of some sort. This is the pattern rule.

The formulation of a rule in terms of behavioral tendencies is an outgrowth of Alexander's earlier work on "relational theory." The problems in design, he postulated, are to reduce the conflicts between people's tendencies, and this is done by finding an appropriate spatial arrangement. These spatial relations are the "atoms of environmental structure." [6] This way of formulating a pattern reflects the style and values of the CES group, but many other ways are possible. When other people begin to design patterns independently, other styles and values will be manifested. I think such a diversification will be good for the development of this methodology.

One aspect of the pattern rule, its *form,* has implications for design methodology. The form of a pattern rule is:

if C then A because P.

In words, if the condition **C** is present in a design problem, then do action **A** (which may be a set of abstract spatial relations to be incorporated into the problem), because there is a problem **P** which usually occurs under the condition **C** and which can be solved by doing **A.** The **because** part of the pattern not only contains the statement of the problem, but also contains the assumptions or the empirical evidence for believing that the problem exists, is important and is solved by the action of the **then** part.

Design Knowledge

The collecting of a large body of patterns would be an excellent way to accumulate and organize design knowledge. To codify all of our present design knowledge in this way would be a tremendous effort, but there are some reasons why such an effort would be worthwhile:

1. The pattern *if-then-because* rule is a canonical format for stating design principles, i.e., connections between function and form. This format is very flexible since there is no restriction on what goes into the three parts. (The formulation in terms of tendencies is just one possibility.) Each pattern is a discrete, self-contained statement since the assumptions, evidence, and rationale behind it are all a part of it. Hence, patterns can be inserted, deleted, or modified independently of each other. The collection of patterns can be seen as an ever-changing body of knowledge (evolutionary, if you prefer).

2. The patterns are useful to the designer because they are in a form which is directly applicable to his problems.[7] The patterns do not do the designing themselves. The action components of the patterns allow infinite variety in their application to specific problems. The designer has to be very careful and imaginative in his execution of them: The patterns help the designer basically by breaking his problem into subproblems, by formulating these subproblems, by introducing appropriate concepts and relations, by grouping related aspects of his problem together, and by serving as an exhaustive checklist.

3. An important feature of the pattern rule is that the designer can assess its validity for himself and, hence, can apply it much more sensibly to his problems. From the point of view of methodology, it is not so important how good each pattern is, but only that each one is transparent and open to criticism and can be improved over time. The designer, working on a specific project, can feedback his experience with the patterns so that they can be refined and certified; or he may even invent new patterns.

4. The most significant impact of the patterns is that they put the emphasis on the deeper (abstract, conceptual, functional) aspects of design. They provide a way of explicitly dealing with these kinds of issues. The patterns allow us to deal with conceptually distinct alternatives to design problems, not just narrow variations on a single theme.

5. As a body the patterns open up a lot of interesting empirical questions. How many patterns are needed to be generally useful? (Many thousands, probably.) Many of them would be very specialized, but many others would be widely applicable. What about the possible variations of style and content of the patterns? How much variation is tolerable? Can technical problems be formulated in pattern rules? Can many different people contribute to the same collection of patterns without it losing its coherence? Such questions can only be answered by trying to build and use a large collection of patterns.

6. The body of patterns also have a scholarly and academic interest. It would constitute an invaluable empirical base for assessing our current state of design knowledge. Such scholarly, detached study could reveal weaknesses, redundancies, or contradictions and could propose consolidation and refinement of groups of patterns. And finally, they would be the basis for a theory of environmental structure (if one is ever to be formulated).

Language

We have not yet touched on the question of the *organization* of the collection of patterns. Although they are essentially independent, the patterns do, as a body, possess a structure—a very complex structure. For the designer, the problem is one of *indexing.* At any point in time, how can he find out which patterns apply to his problem? Does he have to search the whole collection each time? This is very inefficient. The CES group believes that the designer will learn the patterns; and, once he has done so, he will simply "know" when they apply; and he will become

5. Pattern 34 of the MSC report.

6. Christopher Alexander & Barry Poyner. "The Atoms of Environmental Structure." R & D Paper, Ministry of Public Buildings & Works (London), 1967.

7. For some evidence on the use of patterns see: Roger Montgomery, "Pattern Language." *Architectural Forum,* January 1970.

. . . it is not so important how good each pattern is
but only that each pattern is transparent
and open to criticism and can be improved over time.

fluent in design.

Alexander likes to call the collection of patterns a "language," and he likes to refer to the designer as a "speaker" of the language. A quick look at modern linguistics will show why *language* is an attractive concept. It is the work of Noam Chomsky which has brought about a new and rigorous way of dealing with the structure of language.[8] Chomsky makes a distinction between language "competence" and "performance." The former refers to what we intuitively know about the structure of our language (e.g., given any string of words, we can judge whether it is a "grammatical" sentence or not). The latter concept refers to how we actually use language (e.g., our everyday speech is grammatically very sloppy, although we usually aren't aware of it). Since performance involves too many non-linguistic factors, he wants to explain competence. To do this he proposed the notion of a "generative grammar" as the vehicle for formally characterizing language competence. The grammar is a system of rules which explicitly enumerates the sentences, and only the sentences, of a given language. Not only are the sentences produced, but also their structural descriptions are produced. These descriptions have many levels, two of which are their "deep structure" and their "surface structure." Surface structure is just phrase structure (like the sentence diagrams we used to draw in grade school). Deep structure reflects the logical relations which underlie a sentence's meaning. E.g., the sentences "Brutus killed Caesar," "Caesar was killed by Brutus" and "it was Brutus who killed Caesar" all have essentially the same deep structure even though they are different on the surface. Chomsky's grammar gives the central role to syntax, i.e., to symbol-manipulating rules which make no reference to meanings of words. Hence, he characterizes a language by purely formal rules which can generate the sentences of the language.

How about the pattern language? In a sense the patterns represent what a designer needs to know to design a certain type of building—his design competence. The pattern language is also generative, a "system of generating principles." Most important, the patterns exhibit what is generic among many different-looking buildings, the conceptual and functional structure, underlying them—their deep structure, if you will.

This analogy with language is very powerful. But beyond this inspirational comparison I think the analogy is weak. For one thing, a grammar is descriptive while the pattern language is normative. The pattern language is quite different in form from a linguistic grammar because the pattern language is essentially *performative* in nature; it is concerned with exactly *how* buildings are designed and not just with abstract descriptions of buildings. In a performance system, one is not concerned with the distinction between syntax and semantics since both aspects (structure and meaning) are inextricably combined in the productive mechanism. The pattern language is really closer in spirit, if not in form, to many AI studies of language understanding. In form, the pattern language actually descends from another field.

8. For a readable introduction to Chomsky's work see: John Lyons. *Noam Chomsky*. Viking Press, 1970.

Production Systems

The formal ancestor of the pattern language is the *production system*, which was first used as a tool in mathematical logic. A production system is a generative system, i.e., it defines a set of objects by giving rules which enumerate the set. A production system consists of condition-action rules called *productions:*

$$\begin{array}{c} C_1 \rightarrow A_1 \\ C_2 \rightarrow A_2 \\ \cdot \\ \cdot \\ \cdot \\ C_n \rightarrow A_n \end{array}$$

These productions manipulate symbolic *objects*. The condition part of a production can be thought of as a statement which, when applied to an object, is true or false (if it is true, we say it *matches* the object). The action part of the rule is arbitrary; but it is applied to the matched object, thus changing the object (and we regard this transformed object as a new object). The following is an *operating procedure* for controlling the application of productions:

```
(1) INPUT an object; and let it be the
    current object.
(2) Call the current object "Z".
(3) Try to match Z with all the condi-
    tion parts C1, C2 . . . , Cn.
(4) If no conditions match Z, then
    OUTPUT Z and STOP.
(5) If one or more condition parts
    match Z, then choose one of them
    and call it Ci.
(6) Apply the action part Ai (which is
    paired with Ci) to object Z trans-
    forming it into a new object which
    is now the current object.
(7) Go back to step (2).
```

This system is *non-deterministic*, i.e., for any given input object, there are many possible output objects, depending on the choices made in step (5).

The production system can also be used as a language for specifying actions; it is a perfectly general programming language. E.g., the operating procedure above could have been defined by productions. Further, we can think of the productions as representing the abilities of a problem-solver (i.e., what it knows how to do), the input object as a problem, and the possible output objects as solutions (or at least as results of the problem-solver's responses to the problem). Productions are an attractive way to characterize human thinking behavior.[9]

Design Process

It is clear, I hope, that the pattern language is an elaborate production system and that patterns are productions. (Since I think the word "language" is overdrawn, I will simply refer to the *pattern system* from now on.) The objects the pattern system deals with are conceptual—design problems, partially specified designs, design solutions. The method of the pattern system is basically that of the designer executing the operating procedure above. He has the freedom to make the choice in step (5), i.e., he can decide which pattern to apply next. The designer introduces further variation in step (6) where he applies the **then** (action) part of the pattern to his problem. A pattern rule is more than a production since it has a **because** part (which should be incorporated into the design as part of its rationale). A nice feature of a production system is its flexibility of allowing *dynamic responses* to conditions even if the system did not cause them. That is, a production system does not specify the order in which its rules are to be applied; the ordering is determined solely by the conditions of the problem. If an outside agent (e.g., a client) suddenly changes the object (the design problem) which the pattern system (the designer) is working on, then patterns can immediately be invoked which deal with these new conditions.

The design process is not as simple as the above sketch indicates, of course. Production systems usually prescribe serial behavior. But a designer is often involved in trying to handle many design factors simultaneously. This can be a result of bad design procedure, but a certain amount of parallel operation is necessary since design must resolve competing demands for a limited physical space. The pattern system

9. See: Allen Newell & Herbert Simon. *Human Problem Solving*. McGraw-Hill, in press.

A production system is a generative system, i.e., it defines a set of objects by giving rules which enumerate the set.

does allow for some patterns to be applied simultaneously.

This view of the design process as being guided by the complex operation of a production system leads us back to the problem of *efficiently* determining which patterns apply during the course of the design process. Some patterns deal with general issues while others deal with details, and they will be applied in a basically general-to-specific order. Clearly there is a structure of relations among the patterns. Some patterns are related because of the logical relation between their **if** parts (e.g., condition C_i may be the negation of condition C_j). Some patterns are related because they deal with the same physical part of a building (and they should probably be applied simultaneously). We can use these relations to form pointers between patterns, which will suggest to the designer the order in which he will need the patterns.

The CES group presents such a scheme in their report, which they call a "cascade". (See *Figure 1.*) Even as it is, their cascade is a confusing labyrinth; but imagine what a cascade for a thousand patterns would be like—it would be useless. The CES group has been working on this structure and representation problem, and they report considerable progress. They now view the structure in terms of a set of conceptual design units with the patterns serving as complete links between these concepts. They are working on a *pattern book* in which each page presents one pattern with the conceptual units it links up. Thus they represent the pattern structure locally, leaving it to the designer to fit the pieces together.

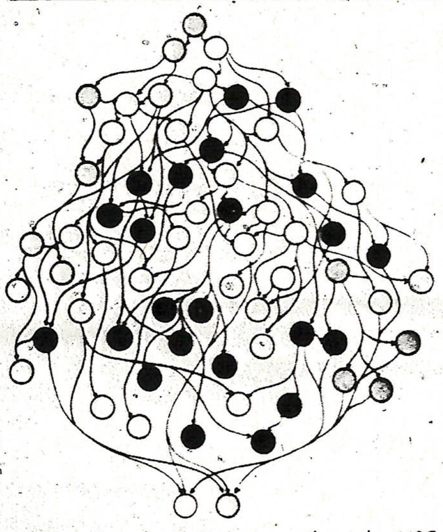
Figure 1. Cascade of 64 patterns (from the MSC report)

Computers Again

These problems of the control process and the structure of the pattern system will need considerable study. But process and structure are central concepts in computer science. I think that the computer (and its science) can be a great help in clarifying these issues; this can be the computer's most fundamental contribution. Let me suggest a way for the computer to be more intimately involved in the pattern system, which might also be of practical benefit.

You are a designer with a real design problem. You also have a typewriter and a display screen which are plugged into a large computer. The computer has in its memories all the known patterns (thousands of them). The computer will be your record-keeper and *adviser*. You will tell it all your design decisions—your conceptual decisions as well as your decisions about physical form. The computer will advise you as to which patterns are applicable to your problem at every step. You will decide whether to use the patterns and how to apply them. Let us call the computer system in this scheme the Architect's Adviser (AA).

AA is a considerably more difficult undertaking than any present computer-aided-design systems. AA's advantage over URBAN5 is that AA is giving real, intelligent, architectural advice and is not just checking for a designer's inconsistencies. AA expands the idea of an URBAN5-type of system by underpinning it with a whole new level—the level of design concepts. AA focuses attention on this conceptual level rather than on the level of graphic detail.

The question must be posed at this point: does AA have any real advantages over a pattern book? In the short run, no. In the long run, I think, yes. Psychological evidence suggests that a designer can learn a lot of patterns; but I don't think it's ever enough patterns. I conceive of the pattern system as holding the cumulative design knowledge of the whole architectural community. No one designer can or wants to know it all. If a designer spends his whole life doing a narrow range of design, then he is an expert in his area; and he doesn't need advice. But most designers are facing new kinds of design problems all the time; and they can use an expert adviser.

The other advantage of AA over the pattern book is that AA is dynamic while the pattern book is static. A pattern book cannot possibly display all the possible connections between patterns, but can only suggest the most probable connections. This may not be helpful in unusual design situations (where the patterns are most needed). A sophisticated AA would not be strictly bound to the standard connections between patterns. It would be on the alert for unusual situations (although it would take AA longer to recognize these), and it would point these out to the designer.

Let us consider some of the problems to be faced in designing an AA. The talents of both design theorists and computer scientists would be needed. The most important problems at present are in developing the patterns. There are problems of large-scale computer-system design. And there are the problems of programming AA to be intelligent. I will focus on these last problems.

Formalization

At present, patterns are stated in English; and this is quite appropriate at this stage of the game. But AA will do symbolic computations on the patterns. Thus we need a way of giving AA a precise representation of the patterns, i.e., we need an *interface language* which AA can interpret. We will have to recode the patterns into this language, and this will force us to be very precise about the intended meanings of the patterns.

The previously described pattern of a street-niche is shown in *Figure 2*, encoded in a possible interface language. The pattern is expressed as three groups of numbered statements. Statements **1-4** assign names to certain concepts to which the rest of the statements will want to refer. **5** specifies that there must be a certain goal in the design problem for this pattern to apply. The action associated with this pattern is to define a new concept "street-niche" (**6**) and to specify it relative to the rest of the concepts in this pattern (**7-11**). **12** declares that street-niches are actually to be added to the design. The rationale for this pattern is that there is a conflict (**13-14**), that the street-niche resolves this conflict (**15-16**), and that the street-niche attains the goal **5** (**17**).

You probably have found the language in *Figure 2* quite understandable even though you've never seen it before. It would not take too much training to be able to write statements in it also. (While this language is not as desirable as full English, it is better than doing nothing just because computers can't handle English.) The parentheses are used in this language merely for bracketing phrases (surface structure). Even if these parentheses were removed, the statements would probably still be machine-interpretable with existing AI techniques.

When AA interprets the statements in

We need to focus our attention at the deep level— the processes behind insight, creativity, etc.— if we are to make design more intelligent.

IF

1 ("B" IS (A BUILDING))
2 ("F" IS ((A STREET-FRONTAGE) OF B))
3 ("S" IS ((THE SIDEWALK) NEXT-TO F))
4 ("P" IS ((ANY PERSON) ON S))
5 ((ONE GOAL) IS (THAT (P KNOW-ABOUT ((THE ACTIVITY) OF B))))

THEN

6 (DEFINE "STREET-NICHE")
7 ((A STREET-NICHE) IS (A NICHE))
8 ((A STREET-NICHE) IS OPEN-TO S)
9 (((THE FLOOR) OF (A STREET-NICHE)) IS LEVEL-WITH ((THE SURFACE) OF S))
10 ((THE DEPTH) OF (A STREET-NICHE)) IS GREATER-THAN (5 FEET))
11 ((A STREET-NICHE) CONTAINS ((A DISPLAY) OF ((THE ACTIVITY) OF B)))
12 LET (F CONTAIN ((ONE OR MANY) (A STREET-NICHE))))

BECAUSE

13 (P TEND-TO MOVE)
14 (P TEND-TO (STOP AND (LOOK-AT F)))
15 (P CAN GO (INTO STREET-NICHE))
16 (((A PERSON) IN STREET-NICHE) NOT TEND-TO MOVE)
17 (((A PERSON) IN STREET-NICHE) LOOK-AT (THE DISPLAY))

Figure 2. A pattern expressed in an interface language (the surface structure).

Figure 2, it must assimilate them into its own network of concepts. What the results of this interpretation process might look like is pictured in *Figure 3*. Here we see a fragment of AA's own conceptual structure. The circles and boxes are concepts and the arrows relate them. The boxes are concepts AA had previously defined in its memory. The circles are the new concepts that AA created while interpreting statements **1-11**. This network represents the deep structure of these statements. It is on this network of concepts that AA will do its symbolic processing. For example, when AA finishes interpreting statements **12-17**, it could use standard deduction techniques to rigorously *prove* that a conflict exists, that the pattern resolves it, and that the pattern solves goal **5**. Hence, AA can verify that a pattern makes sense.

A more difficult part of AA's processing is to take this interpreted pattern and relate it to the other patterns in its memory. AA will have to use *heuristic* techniques, i.e., techniques which are usually helpful but not foolproof. (Human thinking is characteristically heuristic.) AA will compare the **if** part of the new pattern with the **if** parts of the other patterns for similarity; it will look for other patterns which deal with the same entities; and it will relate the new pattern to these. We can also advise AA about how we think the patterns are related. Using these kinds of strategies, AA will build up and maintain a complex network of relations among the patterns.

AA will also have to build up a description of your design project as you, the designer, tell AA about it. This problem should not be underestimated. Designers are not used to making explicit what's on their minds. But, like a psychoanalyst, AA needs to know if it is to be helpful. You will use the interface language, plus some graphics, to give AA a running account of your project as it evolves. For example, a building description would have to contain such items as:

(a) the problem requirements, criteria, goals, assumptions;
(b) the conceptual entities involved;
(c) the activities and features of the building and its functional parts;
(d) the relationships between (a,b,c);
(e) the physical components of the building;
(f) the relationships of (b,e) to the existing environment; and
(g) any other problem constraints.

You can assume that AA will provide some clerical services, such as graphic and accounting routines, to make your task more manageable. AA will warn you if you give it conflicting information, and it will ask you for information it needs which you have not told it. AA's representation of your design project will be a network similar in form to the one in *Figure 3*.

Given its understanding of the patterns and a description of your problem, AA's most important task is to find patterns which apply to your problem. Designing AA's *pattern matching* processes is probably the most difficult task in this research. These processes will also have to be heuristic. AA will use the relations between patterns as a guide to which patterns are most likely to apply next. Even the matching of a specific pattern takes much care. In the street-niche pattern, for example, AA will have to examine its description of your problem and find entities there which match (have the same local structure as) the entities **B, F, S, P, g,** and **a** in figure 3. If it does make this match, it will report to you where in your problem it found the match. If you then decide to use the pattern, you will tell AA (in as much or little detail as you want) how you apply the street-niches to your building. AA will

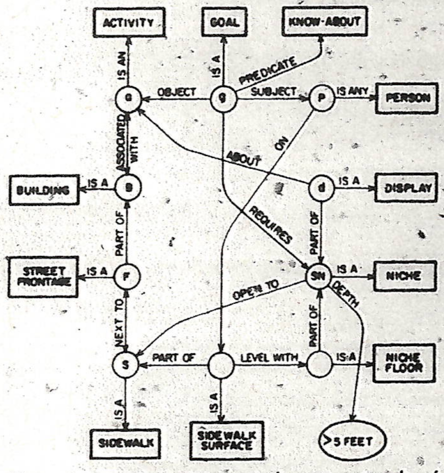

Figure 3. A pattern interpreted as a network of concepts (the deep structure).

record this by inserting entities in your problem which match the entity **SN** in *Figure 3*. AA will tag this new entity in your problem with a pointer to the pattern which caused its creation. If at any time in the future any of the assumptions of the street-niche pattern (or goal 5) become invalid, then AA will ask you to reconsider your decision to employ those entities matching **SN**.

Both the pattern organization and the pattern-matching-processes are difficult computational issues and are ripe for study. If effective schemes could be found for these processes, they would be real advances in the state of the AI art as well as being useful in this design method.

Language Again

Let me conclude by noting the role of language in these processes. Language has two levels—surface and deep. The surface level is the level of interfaces. The deep level is the level of concepts (either in the designer's mind or in AA's memory). The function of interfaces is to transmit concepts. English sentences, the statements in *Figure 2*, diagrams, and drawings are all interfaces. It is more important that concepts get accurately transmitted than which interfaces are used. We should not get hung up at the interfaces; the most interesting and important things happen at the deep level. This is where intelligence, insight, creativity, etc., all live. We should not attribute qualities to surface phenomena which actually emanate from this deeper level. Our understanding of the deep level of concepts is crude at best, and we need to focus our attention at this level if we are to make design more intelligent.