ARTICLES

Using an expert system for design diagnosis and design synthesis

Abstract: *This paper describes the concepts which allow an expert system to be used for both design diagnosis and design synthesis. An example of the implementation of these concepts is presented in the domain of preliminary design of domestic kitchens in the expert system PREDIKT. PREDIKT carries out both design diagnosis and design synthesis using the same knowledge base and utilises an existing expert system shell which has forward- and backward-chaining capabilities. The significance of graphical interaction with expert systems in design domains is demonstrated.*

RIVKA OXMAN
JOHN S. GERO

Architectural Computing Unit
Department of Architectural Science
The University of Sydney
NSW 2006
Australia

1. Introduction

1.1 Design as a problem–solving paradigm

The design process has been frequently modelled as a recursive interaction of the activities of analysis, synthesis and evaluation. During the development of a design, the progress towards a solution is achieved by generating new states from current ones. Such problems in which information is accumulated during the process of solution are representative of the class of problems which have been referred to as "ill–defined problems" [1]. Design can be interpreted in a state–space representation as an initial state which is transformed using expert knowledge into a series of solution states. Within this conceptual framework, problem–solving can be seen as a process of searching through alternative solution states which satisfy certain goals. Design, therefore, shares many characteristics of general problem–solving processes. The behaviour of designers in a process which incorporates automated problem–solving techniques has been studied by Akin [2]. From this model of design we can extract several kinds of problem–solving activities to support this process in a computer–aided design environment.

1.2 Knowledge engineering and expert systems in design

Knowledge in design may be thought of as the tool whereby the designer conceptualises the semantic content of a certain domain and by which he represents his ideas about that domain as the syntactical relations between symbols (domain knowledge) and actions by which these symbols and relationships are manipulated (control knowledge). Knowledge can be described symbolically as well as mathematically. A knowledge–based view of design intends to render the knowledge through which design solutions are generated explicit and amenable to the process of computation. Knowledge engineering is a field which deals with the building of knowledge–based programs. Expert systems are a branch of this larger area of knowledge engineering; they deal with the development of programs which simulate the behaviour and incorporate the knowledge of rational human experts in a specific domain and tend to operate in an interactive manner with users.

This paper discusses the application of knowledge engineering and expert systems techniques in design. The specific domain of application is that of architectural design.

1.3 The traditional role of expert systems

Expert systems which have been developed in other domains contain features which provide directions for the development of expert systems for design. The knowledge incorporated within an expert system consists of facts and heuristics. The facts constitute the body of information and the heuristics are methodological statements and the rules of good guessing which together characterise expert level decision making within the field. The traditional role of expert systems in well–circumscribed domains has been that of a diagnostic tool. Waterman [3] has provided a detailed description of a variety of expert systems and their applications, very few of them are related to design or even have a design component. Of those listed XCON (also known as R1) [4] appears to be the only one in continuous commercial use. It is only now that expert systems technology is being examined for its general applicability in the domain of design. In the subsequent sections, we shall elaborate upon the technical aspects of various approaches and discuss their relevance to design, in general, and to architectural design, in particular.

2. Expert systems in design synthesis and design diagnosis

Two approaches for the application of expert systems in the design process will be considered.

1. *Design Synthesis*: the expert system is capable of design generation.
2. *Design Diagnosis*: the expert system can function as a design critic to evaluate, criticise and recommend corrections in design.

In both modes of operation solutions are generated before they are analysed and evaluated. In any expert system one of the key factors in its construction is the separation of the knowledge base and the control mechanism. Both the way in which knowledge is represented and the way in which it is applied in the inference mechanism must, in design application, recognise the recursive nature and the multiple modes of design paradigms.

2.1 Expert systems in design synthesis

In the automated process of design synthesis, design is treated as a search through a space of solution states. New design states evolve through a process of analysis, evaluation and regeneration. Generation is achieved through the firing, or instantiation, of a set of transformational rules, where an initial state is transformed to a subsequent or final state by means of rule–generated operations. Figure 1 shows a small space of possible solution states in A with an initial state selected by the human designer. Transformational rules encode knowledge about how to generate a new state from a current state, this is how partial solutions are generated as in B. Constraints prevent the execution of one rule and another rule comes into play as in C.

One of the first attempts to represent design knowledge (implicitly) according to such a rule–based conception is known as *shape grammars* [5]. Design knowledge is represented by syntactical transformational rules for the manipulation of formal patterns. The shape grammar deals with shapes represented by line segments, symbols and labels; the elements and the set of transformational rules relevant to a particular design sub–domain, such as a formal style, constitute a syntactical grammar. The control mechanism for the generation of solutions in such systems is known as 'default control' of forward-chaining and back–tracking. The mechanism generates all solutions exhaustively from the initial state to the completed design. In real world situations, design is less a process of exhaustive search through solution space than it is one of recursive, goal–directed search from one state to the next.

Design obviously deals with another broad area in addition to that of syntactic operations. One of the characteristics of architectural design is that of being problem solving in a semantically rich task domain [6]. Syntactic aspects of shape grammars could be extended into a multi–attribute design vocabulary — a type of system which would include the semantic as well as the

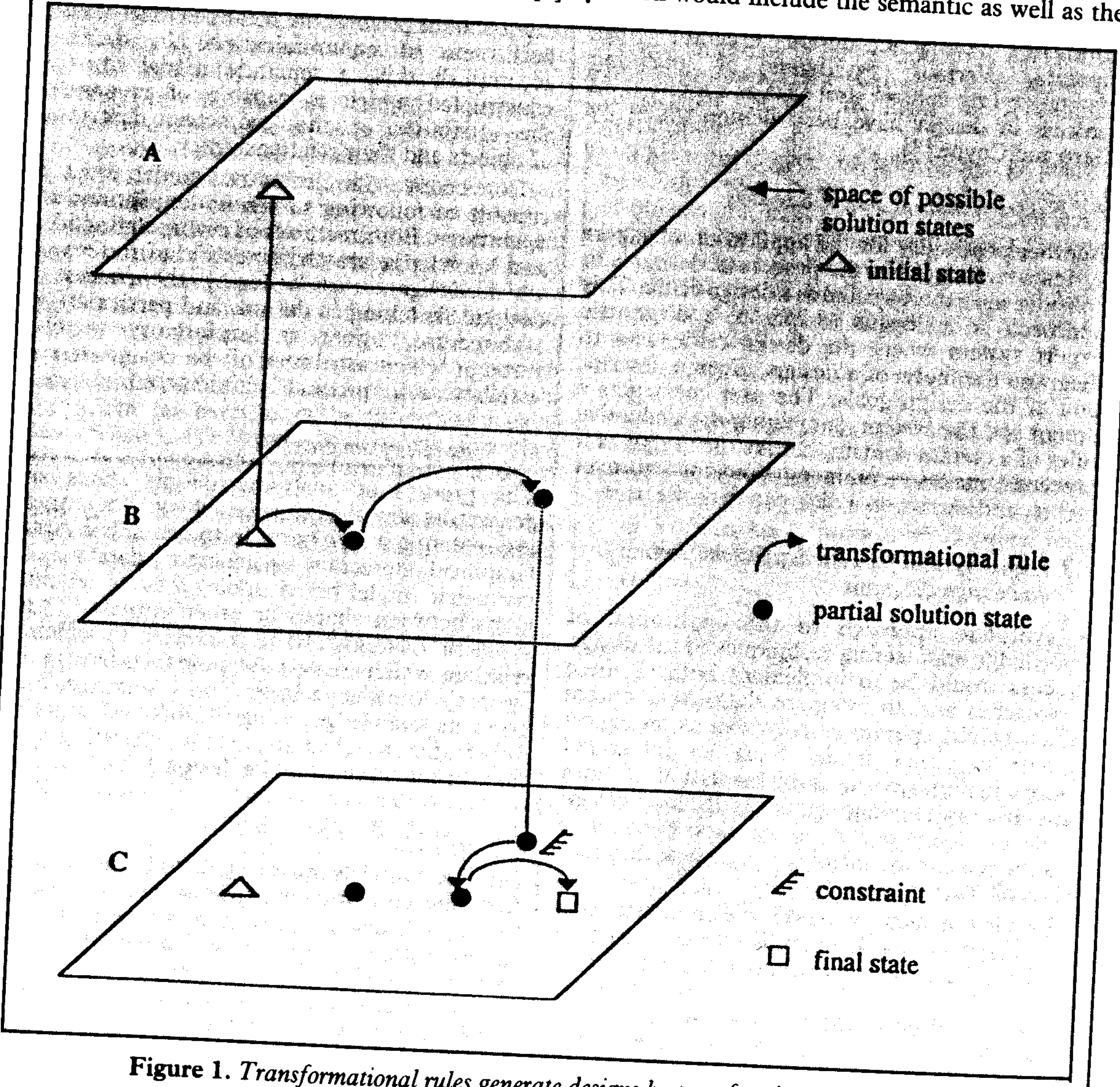


Figure 1. *Transformational rules generate designs by transforming an initial state through partial solution states to a final state which is the design*

syntactic representation of design knowledge. The semantic representation of design knowledge provides a potential basis for both the organisation of design knowledge, as well as for the definition of goals, or the provision of control in search procedures. A linguistic paradigm of design has been discussed by Gero and Coyne [7]. Design rules can be written which are made up of facts about objects and their relationships. The semantic aspects of a design domain can be represented as goals which are the attributes of the objects in their final, or desired, state. In such a manner we can potentially limit the search by using a goal–directed, backward–chaining strategy in which the system knows what kinds of knowledge or actions are necessary in order to achieve specific goals. This appears to be one of the ways used by the human designer to reduce the theoretically large problem space to a manageable size.

Semantic rules at a level of abstraction can become a default paradigm for planning which deals with the selection and ordering of syntactic design rules. These create states at a higher level of abstraction than that at which more 'simply motivated' syntactic design executor rules operate. Various paradigms of planning demonstrating control strategies as a multi–level process in design have been demonstrated by Gero and Coyne [8].

2.2 Expert systems in design diagnosis

Another possibility for the application of expert systems techniques in the process of design is to consider the expert system as a design critic. This approach is analogous to that of a diagnostic expert system where the design critic tries to diagnose the faults of a design, given a description of the design goals. The user constructs a design and the system, containing the evaluative rules of a certain domain, checks the design as it proceeds, makes recommendations for improvements, and carries on a dialogue with the user.

2.3 Expert systems for both design synthesis and design diagnosis

A synthetic approach to the application of knowledge engineering techniques to the design process would be to implement both of these approaches and to integrate them in a system which would operate equally well as an expert system in either mode. Such an integrated approach requires the implementation of both generative and evaluative rules at different design levels (or stages) within the same knowledge base and the use of the control mechanism at different levels of abstraction.

Implicit in such a system is the emphasis upon the importance of a capability for constructing partial design solutions. Partial solutions, once generated, can be evaluated against criteria or desired attributes and provide the context to guide the search and to complete the design solution. This method can procedurally limit combinatorial explosion. Furthermore, in any design session, learning through and about the process which generates partial solutions has value; it can be used to modify, or refine, the initial statement of goals.

The aim of this paper is to discuss three primary methodological issues in the design and implementation of such a system.

1. The conversion of textual (semantic) representation to graphical (syntactic) representation and *vice versa*.
2. The provision of input and response by graphic as well as textual means within the traditionally interactive nature of the expert system dialogue between user and system.
3. The use of the same knowledge base for both design synthesis and design diagnosis and for bringing partial solutions to the state of completed designs.

3. Graphical interaction

Designers express their thoughts, concepts and designs both verbally and graphically. An intelligent system for design in a computerised environment should provide the means to map between both sets of communication. In order to accomplish this, a semantic model must be constructed which is capable of recognising the attributes of the geometric description of objects and their relationships.

Conversely, the geometric model must be capable of following semantic descriptions and operations. Both methods of manipulating ideas and knowledge are characteristic of the way in which designers work during the process of design. Sketching in design, and particularly in architecture, offers an exploratory medium through which attributes of the design emerge which were not previously anticipated.

3.1 Semantics from graphics

The process of analysing design states and generating new ones must be achieved by extracting semantic information from geometric input. Graphical interaction can generate a database of a geometric model based upon syntactic relationships between shapes or other entities. These syntactic relations can be converted to semantic relations which create a dynamic facts base in the system's knowledge base. The system can then apply its knowledge, using the inferred semantic information to select actions for generating solutions or for diagnosing the design.

3.2 Graphics from semantics

Graphical models must be able to be constructed from the semantic descriptions which make up the textual dialogue between the user and the expert system. When interaction with an expert system is based upon a textual dialogue, a semantic facts base can be derived from queries made by the system of the user. The system should infer attributes from this information in order to instantiate attributes which are, in turn, interpreted as a set of syntactical relationships between objects which can then be used to generate graphical constructs. In this manner, the system simultaneously produces from the facts base the definition of the solution state of the design problem, a goal state which is expressed semantically and converts this to a goal state which can be executed by syntactical operations and displayed graphically. Clearly, knowledge which provides the mappings between semantics and graphical syntax needs to be specifically provided to the system.

3.3 Procedural knowledge and declarative knowledge

Representations of knowledge can be both procedural and declarative. Prolog [9] offers a ready means to achieve both forms of representation. Logic programming provides a rich medium with which we can represent object attributes, and through which we can write rules incorporating these attributes in an inference process. When semantic goals have to be interpreted, procedural knowledge is utilised to set up a graphic display. When syntactic relations between shapes is set, declarative knowledge can be employed to interpret semantic content from an existing display.

Graphical images are usually interpreted semantically by humans; there is no concern with the process used to produce the image. A graphical image is a declarative representation. When using a computer to produce a graphical image, procedural processes and representations must be used. This difference between the image and the means of producing it is of fundamental importance because it defines the mappings which are required to interpret and produce graphics. It also means that a computable semantic model of the screen graphics needs to be kept. This has been one of the difficulties in using traditional CAD systems since they store their models syntactically in the form of data and procedures. The semantics $\rightarrow$ syntax and syntax $\rightarrow$ semantics transformations are fundamental to any interactive graphics interface to a design expert system.

3.4 Types of knowledge

Design processes involve trial and error in order to produce successive states of a design solution for consideration. The description is a record of decisions which express the current state of the design. This process requires mappings between various kinds of models. For example, in architecture, there will be mappings between a geometrical model, an architectural model and a performance model [10]. In a manual design process, the designer rapidly executes, interprets and evaluates according to his or her own informal mental models. Knowledge-based programs can provide expert assistance in such design processes by mapping between generation and its interpretation or evaluation.

Three types of knowledge are significant in a design expert system, (Table 1). The first of these, *syntactical knowledge*, here deals with the connection of an object with its domain or other objects and with the data which supports facts. It is not concerned with the meaning of an object and its attributes. For example, when a designer locates a window in a wall graphically, syntactical knowledge is used to find the horizontal and vertical distances from the boundary of the wall. In design, such knowledge aims to maintain the consistency of the database of geometrical descriptions.

The second type of knowledge is *semantic knowledge* which is concerned with the meanings of objects and here deals with the relationships between objects and between their attributes. Design rules can be specified which generate relationships and attributes. Predicates which generate facts about relationships between objects may include:

— positional relations — "window is above sink";
— relations of inclusion — "window is in north wall";
— comparative relations — "west wall is short";
— relations which define a ratio of states — "area of kitchen has a proportion between length and width";
— and so on.

Table 1. *Types of knowledge needed in a design expert system*

Type	*Definition*	*Example*
Syntactical knowledge	Knowledge which maintains the connection of the object with its domain and with other objects; often geometric in nature for physical	If rightmost vertex of object A is $[X_a, Y_a, Z_a]$ and leftmost vertex of object B is $[X_b, Y_b, Z_b]$ then distance between A and B is $\sqrt{((X_a-X_b)^2 + (Y_a-Y_b)^2 + (Z_a-Z_b)^2)}$.
Semantic knowledge	Knowledge concerned with the meanings of objects: normally deals with relationships between objects and between their attributes.	If area of window is greater than 15 per cent of area of floor then light is sufficient.
Evaluation	Knowledge to interpret a design in terms of implicit attributes.	Knowledge driven procedural programs.

Design which is generated according to rules may be interpreted through models of performance. Therefore, specification of rules for interpretation is necessary in order to encode relevant knowledge which can be used for evaluation. Hence, the third type of knowledge is *performance* or *evaluation knowledge*. It is the evaluative knowledge which is used to check the validity of a state solution against the system's knowledge base of performance requirements. For example, an evaluative rule can check the relevant properties of a window (dimensions, location, material, etc) and calculate other associated factors, such as lighting and ventilation, according to the domain specific codes and requirements.

The utilisation of these three types of knowledge and their significance in the operation of such an expert system for design can be demonstrated by a typical predicate of the system.

```
check–execute:
    locate object
    generate facts
    interpret (find 'the approval of the object')
locate object:
    execute and display the syntactical relations
    between entities of an object
generate facts:
    produce semantic relations between objects
interpret:
    check the validity of a design solution against
    rules in the system's knowledge base
```

Designers make use of all three classes of knowledge and apparently utilise morphisms between them which change their function. Thus, performance knowledge at one time becomes a goal set whilst at another time it becomes part of a constraint set. Syntactical knowledge is used both to generate designs and to check generated designs, and so on. This richness in knowledge utilisation makes the domain of design particularly interesting.

4. An expert system for the preliminary stages of design of a kitchen

A system, PREDIKT (PREliminary DesIgn of KiTchens), which elaborates the issues raised in the last section is described. PREDIKT is employed as a design synthesiser and critic in the preliminary stages of the design of a domestic kitchen. The development of this system is concerned with the following points:

1. the system provides several modes of interaction between the user and the system;
2. the system employs the same knowledge base for both design synthesis and design diagnosis; and
3. the system converts semantic representation to graphical representations and vice versa.

The present version of PREDIKT runs on SUN Microsystems workstations, is written in Quintus Prolog and consists of three basic subsystems:

1. a stand–alone expert systems shell, BUILD [11];
2. a knowledge base in the form of design rules related to the domain of kitchen design; and
3. the system converts semantic representation to graphical representations and *vice versa.*

4.1 The expert system shell: BUILD

The shell, BUILD, contains the fundamental components of an expert system: an interface facility; an inference mechanism; an explanation facility; and a state description — these are described in Table 2. BUILD is written in Prolog and has interfaces to Prolog, other languages and, hence, graphics.

4.2 Design rules as a knowledge base

Knowledge about the spatial characteristics and layouts of elements in design domains such as the kitchen are found in a variety of sources. The most common of these are books in the form of general design guides for use in the initial stages of architectural design — the programming and planning stages. These books provide spatial standards and other useful information such as description and sizes of activities and equipment, functional requirements and recommendations for ideal relationships between elements. They may provide standard or critical dimensions for functional elements and provide necessary ergonometric data, as well as hints based on experience and good design practice. This information may come in the form of tables, diagrams, flow diagrams of relationships or activities, or in the form of alternative and recommended plan layouts. Design rules attempt to encode experiential and phenomenological knowledge in a formal and structured way. Knowledge accumulated from experience as a designer results in heuristics. Heuristics enable the designer to focus quickly on important facts of existing conditions and to match through knowledge the appropriate patterns and elements which fit the needs of the design requirements and the design goals.

Such knowledge and experience, once organised, can be encoded in a production rule–based formalism. The resulting system can thus utilise the knowledge of human design experts as the basis for the design and composition of architectural plans. The knowledge base of the kitchen design expert system contains such rules. Rules are built using an object–attribute–value approach. Each object is inferred by a preprocessor and its attributes determined. Consider the following rule:

if
'size of kitchen room'is__'small'
then
'shape of counter'is__'straight counter'.

Within this rule, the relationship between object, attribute and value is as follows:

object	*attribute*	*value*
kitchen room	size	small
counter	shape	straight

Table 2. *Fundamental components of the BUILD shell*

Component	*Description*
Interface facility	Provides for dialogue between the designer and the shell. Dialogue is in a restricted English. Provides for mouse input and windowing capabilities.
Inference mechanism	Carries out reasoning tasks which control the strategy of the shell's execution. It supports both goal driven and data driven processes. In the data driven mode, the given data can be used to infer all that can be inferred or can be restricted to specified topics.
Explanation facility	The shell interacts with the knowledge base and the inference mechanism to explain why an answer is needed at a particular point during the session, or how a question can be answered. The querying strategy elicits the required information from the user in a 'top down' mode. The user then can give information at a high level of abstraction. The user can ask the system either 'how' or 'why' questions. 'How' directs the system to search for knowledge needed to satisfy the request. 'Why' directs the system to provide an answer within the context of a set of rules, and within the framework of the question currently being asked. The shell can explain how a certain conclusion was reached, or explain why a specific conclusion could not be reached.
State description	The shell can display the facts which have been found to be true or false during a particular session. These facts which may have been inferred during a session and entered into the dynamic facts base, can be either completely or partially removed at the end of a session.

Rules may incorporate variables. For example:

```
if
    'area in sq metres'is A and
    'width in cm' is W and
    'length in cm' is L
then
    A is W * L/10000.
```

The process of determining specific values for attributes which are stored in the dynamic facts base can be accomplished in a number of ways: through an interactive dialogue with the user; by selecting an object from a menu and interpreting its attributes from a graphic display; or through an inference process.

Typical rules from the knowledge base are listed below:

```
if
    'area of kitchen in sq m' is_greater_than
      30.0 and
    'width of kitchen in cm' is_greater_than
      210 and
    'length of kitchen in cm' is_L and
    'acceptable dimensions'
then
    'classification of kitchen size' is_large

if
'location of window' is_'in corner' and
'length of window in cm'is_greater_than
  140.0
then
    'size of window is big enough'

if
    'number of windows' is_1 and
    window is_on Wall1 and
    'long wall has window' and
    shape is_'straight wall'
then
    window share_wall_with counter and
    counter is_on Wall1
```

In this system all control knowledge is vested in the expert system shell.

4.3 The Semantic Interpreter

The design rules which are related to the domain of residential kitchen design are encoded as previously described. Between the knowledge base and the graphics display is a semantic interpreter, (Figure 2). The purpose of the interpreter is to convert graphics to facts and *vice versa*.

The general conversion transformation is of the form

$$\text{semantics} = \tau\,(\text{syntax})$$
or
$$\text{syntax} = \tau'\,(\text{semantics})$$

where τ and τ' represent the knowledge needed to carry out the transformation. (In certain logic programs τ and τ' are homomorphic and represent forward and backward deduction processes.)

When the user displays ideas graphically, the system converts the syntactic relations between objects to facts which can be checked against the rules in the knowledge base. This is used in the process of diagnosis when the system is operating

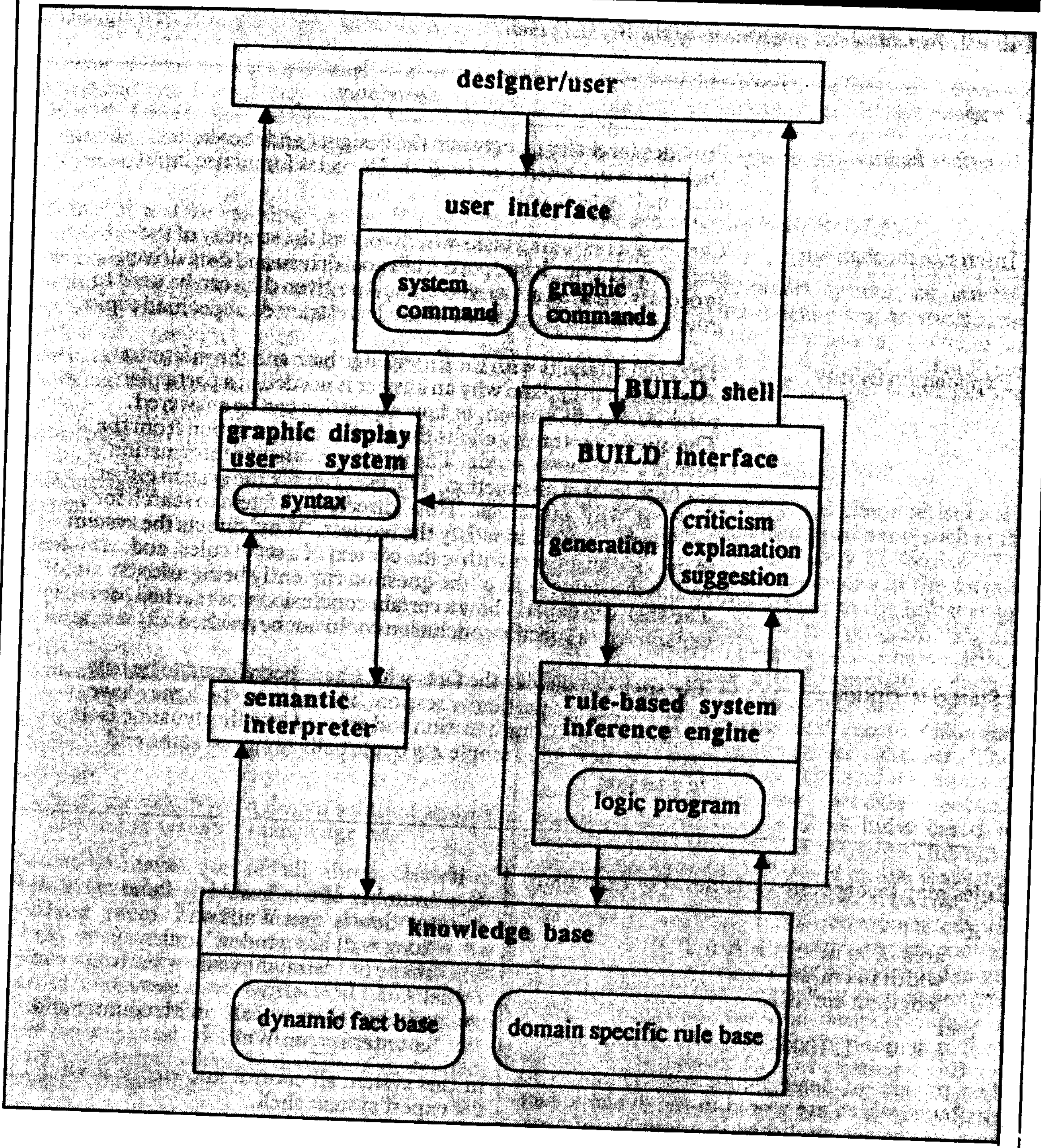


Figure 2. *Data flow in PREDIKT*

as a design critic and analysing design decisions which have been made. Consider the following simple example. The designer wishes to input the plan information of the kitchen graphically. This can only be performed syntactically by specifying the x- and y-coordinates of two opposing corners of the plan (assuming it is rectangular) usually by pointing with a mouse. If these two corners have coordinates (X_1, Y_1) and (X_2, Y_2) then τ can be written in Prolog as

```
locate_area:
    Dx is X2 — X1,
    Dy is Y2 — X1,
    draw_area (Dx, Dy),
    assert (length of kitchen is_Dx),
    assert (width of kitchen is_Dy),
    assert (area of kitchen is_Dx*Dy).
```

This transformation results in three semantic facts being asserted. Other transformations may be considerably more complex than this example.

Alternatively, when the user describes ideas and defines goals textually, the interpreter converts semantic relations and displays the resultant syntactic relations of elements graphically. This mode of operation can be employed in the process of design synthesis when the system operates as a generative system.

4.4 Backward-chaining and forward-chaining operations on the knowledge base

Backward-chaining as goal chaining

The following example of PREDIKT's operation illustrates this mechanism. The goal of this example is to determine an appropriate shape for the kitchen counter. Attributes in the knowledge base represent properties such as the size of the room and the location of activities and equipment. PREDIKT infers the recommended shape of the kitchen's counter from available alternatives. Backward-chaining identifies the needed attributes as subgoals and other rules are examined to conclude values for this subgoal, e.g. size of kitchen, functions and type of movement in the kitchen. The results are passed forward and conclusions are drawn about the main goal. Occasionally PREDIKT will request information from the user, such as "What is the desired area of the kitchen?" That is, the values of attributes can be established in any session by various processes depending on the mode of PREDIKT's operations: by inference from other rules; by querying the user in a design dialogue; or by interpreting the information in the current state of the design as displayed graphically.

A typical goal chaining tree is shown in Figure 3.

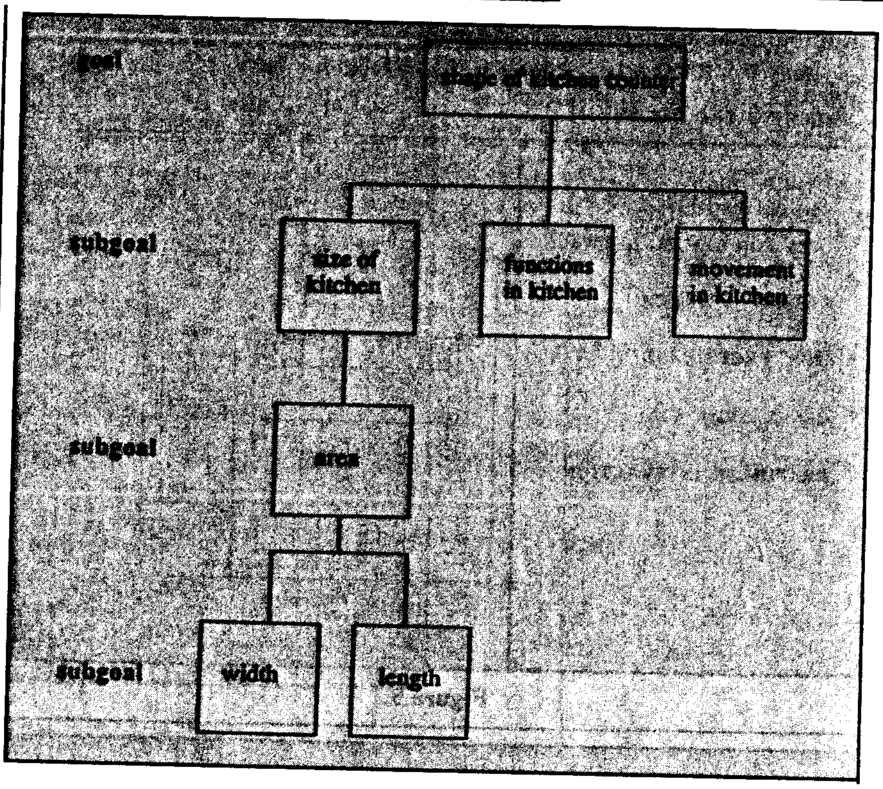


Figure 3. *Goal chaining tree*

Forward-chaining as generation

In forward-chaining, the rules are examined to determine whether or not they are applicable, given the information on hand. When executed, this is equivalent to generation in a state-space. Given a floor area of a specific kitchen, the system can examine all of the possible solutions for different types of movement and activity, and can propose alternative counter shapes for kitchens of the same area. A typical forward-chaining fact tree is shown in Figure 4.

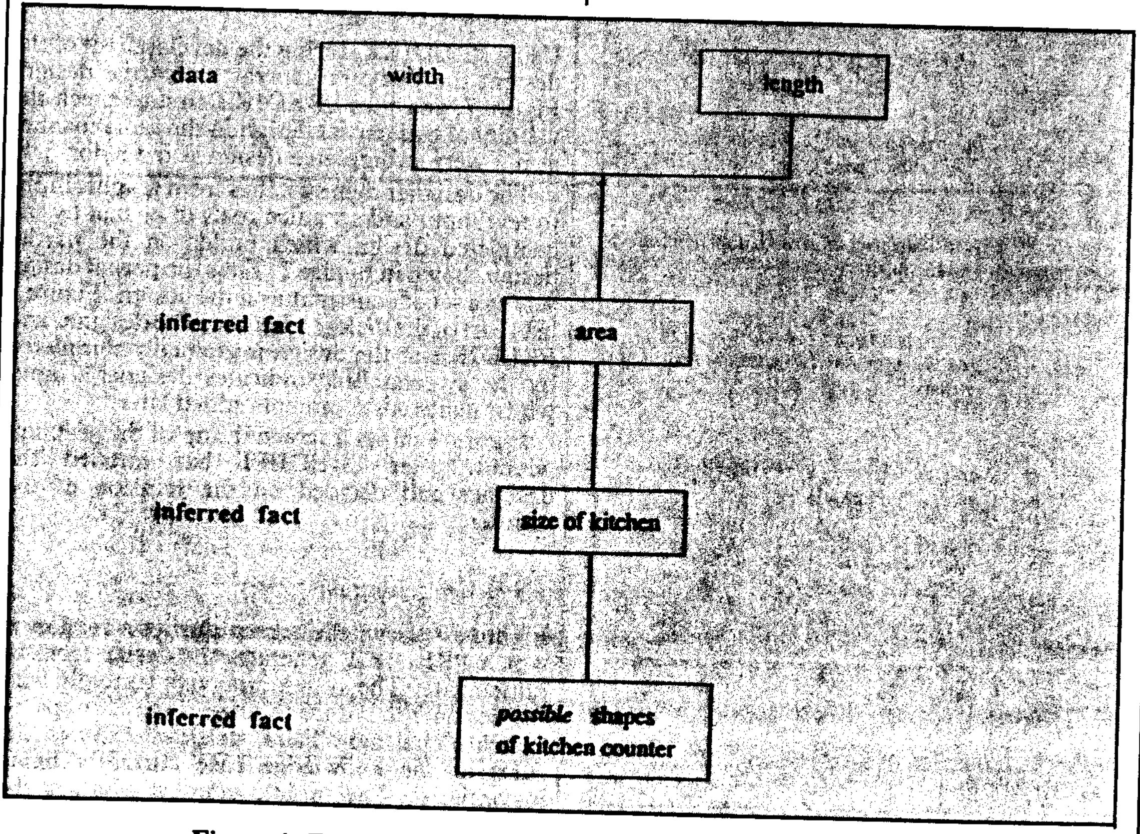


Figure 4. *Forward chaining (data driven) as generation in a state-space*

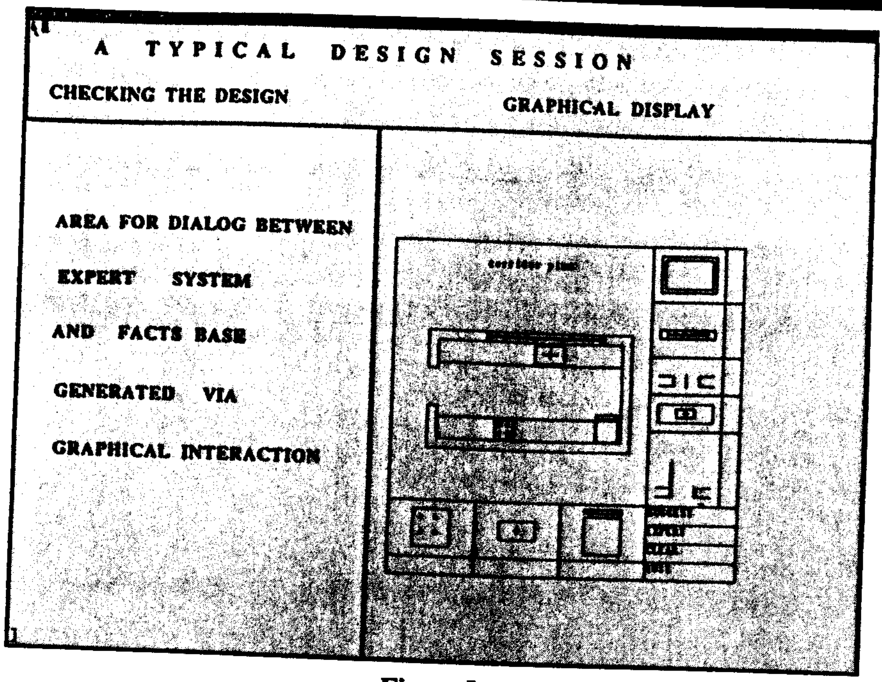
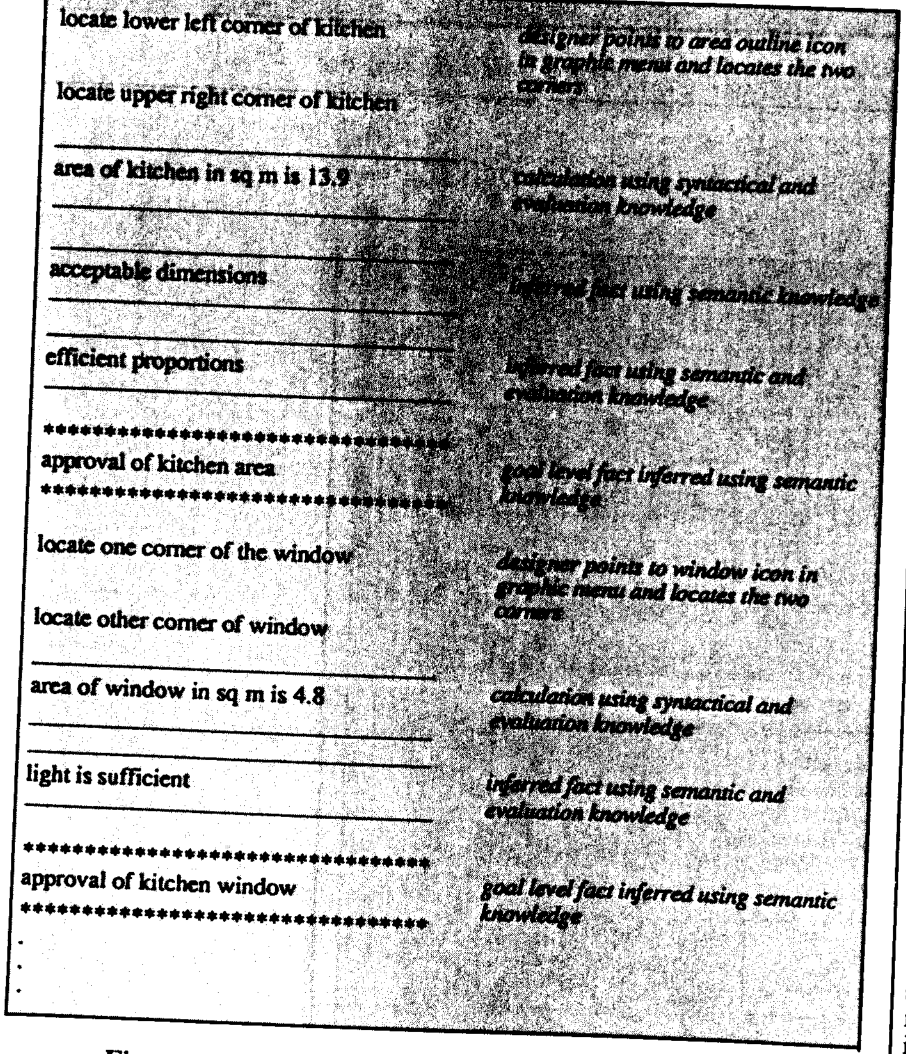


Figure 5.



locate lower left corner of kitchen	*designer points to area outline icon in graphic menu and locates the two corners*
locate upper right corner of kitchen	
area of kitchen in sq m is 13.9	*calculation using syntactical and evaluation knowledge*
acceptable dimensions	*inferred fact using semantic knowledge*
efficient proportions	*inferred fact using semantic and evaluation knowledge*
***********************************	
approval of kitchen area	*goal level fact inferred using semantic knowledge*
***********************************	
locate one corner of the window	*designer points to window icon in graphic menu and locates the two corners*
locate other corner of window	
area of window in sq m is 4.8	*calculation using syntactical and evaluation knowledge*
light is sufficient	*inferred fact using semantic and evaluation knowledge*
***********************************	
approval of kitchen window	*goal level fact inferred using semantic knowledge*
***********************************	
.	
.	
.	

Figure 6. *Part of a text script during a design diagnosis session. Italics are added comments*

5. Graphic interaction

Three modes of interaction are possible between the designer and PREDIKT:

Design diagnosis: the designer uses PREDIKT to check a completed design described graphically;
Design development: the designer develops a partial design, has it checked by PREDIKT, and then interacts with it to generate a completed design which satisfies the implicit constraints from this partial initial state; and
Design generation: the designer interacts with PREDIKT from the beginning of the design process and generates a completed solution utilising the knowledge in PREDIKT as goals and constraints.

Figure 5 shows the screen during a typical design session. The left–hand window is for text input and output whilst the right–hand window is for graphic input and output.

5.1 Design diagnosis

In design diagnosis the designer describes the design entirely graphically (using the graphic window in Figure 5). The semantics implicit in the graphics are interpreted as facts and placed in the facts base. PREDIKT now checks the design by interacting with the facts base using its knowledge base. A typical diagnosis text script is shown annotated in Figure 6, italics are comments added later. In this session PREDIKT checks the design as it proceeds.

5.2 Design development

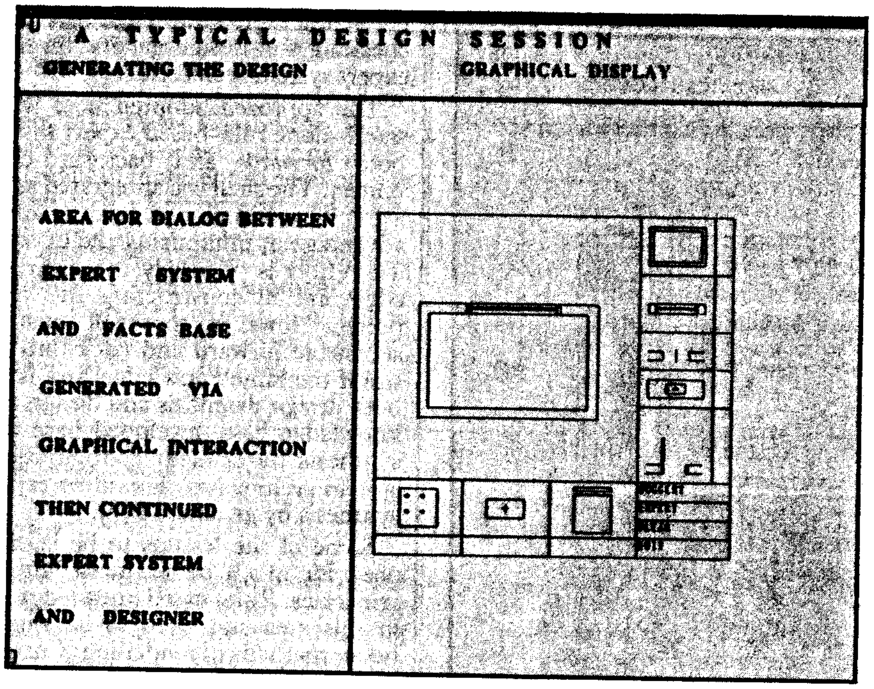
Using PREDIKT during the development of the design, the designer generates a partial design, Figure 7, and uses PREDIKT first to check the validity of this partial design in the same manner as if it were a completed design as in section 5.1.

The designer then switches from graphic input to text input and specifies goals to be met by the completed design which builds on the partial design shown in Figure 7. Thus the partial design acts as a set of constraints on the design. Figure 8 lists part of the dialogue between the designer and PREDIKT as the design is gradually completed by the system. Bold indicates designer's input whilst italics are comments added later.

Figure 9 shows a screen dump of the graphics window after PREDIKT has queried the designer and decided on the location of the counter.

5.3 Design generation

Figure 10 shows the screen during a session in which PREDIKT generates the design from its initial state. The system interacts textually with the designer to reduce the size of the state space–search. The right-hand window displays the rules in the knowledge base currently being instantiated. The middle window displays the dialogue with the user. The bottom left window shows the design graphically as it is being generated. Above it is an information window.



A TYPICAL DESIGN SESSION

GENERATING THE DESIGN

GRAPHICAL DISPLAY

AREA FOR DIALOG BETWEEN
EXPERT SYSTEM
AND FACTS BASE
GENERATED VIA
GRAPHICAL INTERACTION
THEN CONTINUED
EXPERT SYSTEM
AND DESIGNER

Figure 7.

? find counter.	*designer specifies that he/she wants the system to design the counter*
The shape is ?_enter value (how/why)	*PREDIKT queries designer for additional fact*
options for values are: L shape or straight wall or U shape or corridor plan	
? how.	*designer asks how such facts can be found*
dining is in kitchen ?_ enter yes/no (how/why)	*PREDIKT queries designer for alternative fact*
? yes.	
shape is L shape	*inferred fact using semantic knowledge*
window share wall with counter	*inferred fact using semantic knowledge*
** counter is on north wall and east wall **	*goal level fact inferred*

Figure 8. *Dialogue between the designer and PREDIKT as the partial design in Figure 7 is gradually completed by the system querying the designer when needed. Bold text indicates designer input, italics indicate comments added later*

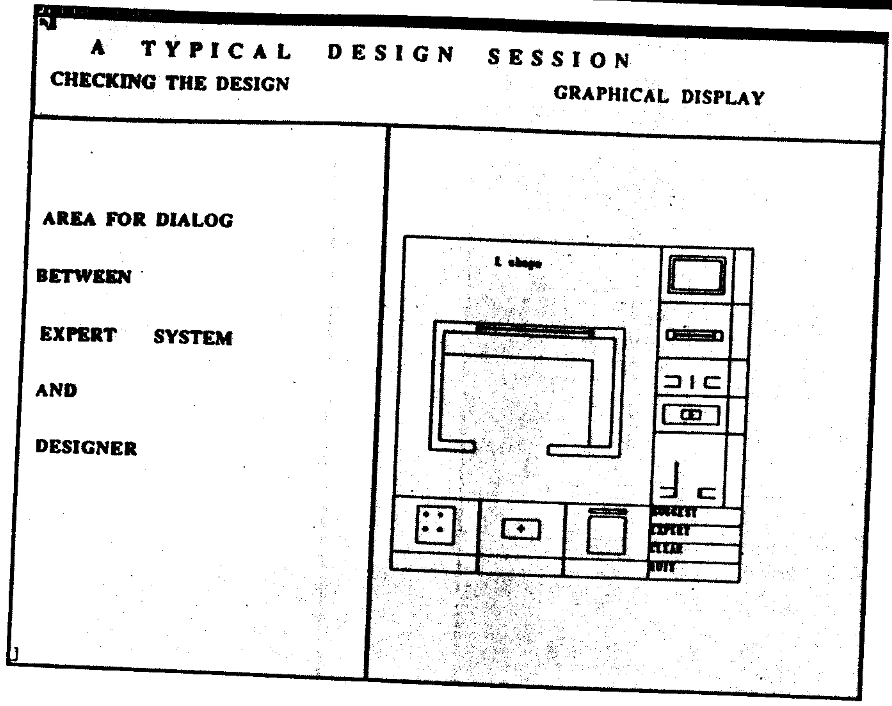


Figure 9

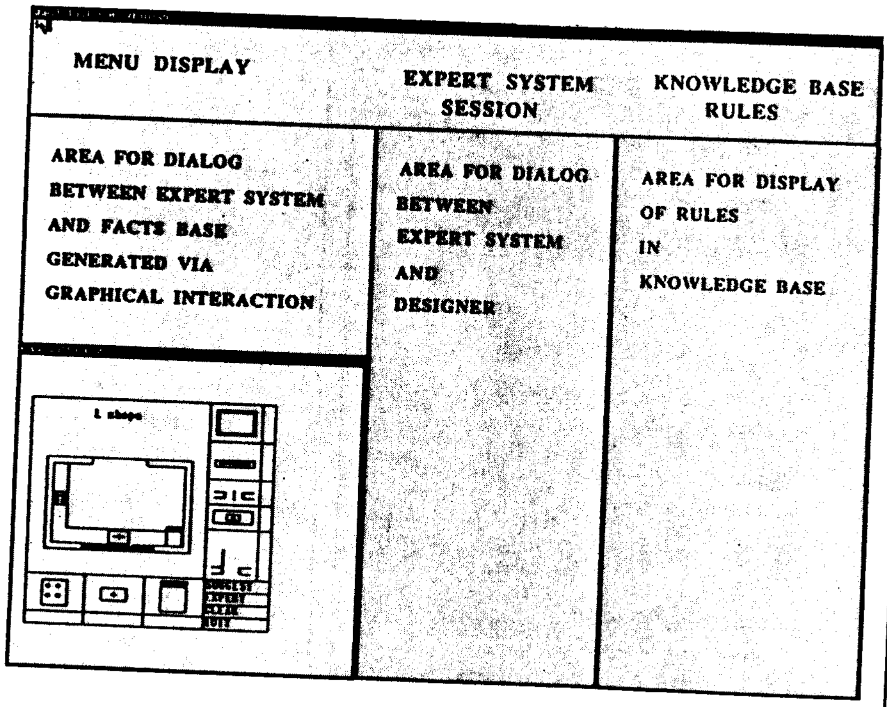


Figure 10.

6. Discussion

This system has demonstrated the applicability of the expert systems technology to design synthesis. Although the knowledge base is small, under 100 rules, it shows that even a simple rule–based approach begins to address elementary configuration type design problems. Of particular significance is the ability of PREDIKT to utilise the same knowledge base for both design checking and design generation using the same expert system architecture.

The approach adopted here was to utilise a stand–alone rule–based expert system shell with both forward- and backward-chaining capabilities. The shell was integrated as a module into a graphic environment. Such an approach has the advantage of minimising the development time. The effort is primarily in acquiring the knowledge and in constructing the semantic interpreter. However, the shell must have directly accessible forward and backward chaining control if the same knowledge base is to be used for both design diagnosis and design synthesis. The knowledge base developed here was for design synthesis by element configuration. The same system architecture should be capable of design synthesis by generation [7].

Some of the lessons to be learnt include the need for much better modelling of the design semantics. Rule–based knowledge bases are weak in this area and need to be supplemented by better methods of modelling. Currently, BUILD is being supplemented by a frame–based semantic modelling system which addresses this issue [12]. The present approach is satisfactory where the design domain in state–space terms is relatively small. When the domain becomes large, additional control knowledge generally in the form of planning knowledge is needed [13].

Expert systems allow a designer, working in a well–defined domain, to adopt a variety of approaches to the final generation of a design — ranging from full manual generation with automated checking, through partial manual generation with the system completing the design, to having the system carry out the entire design.

7. Acknowledgements

This work is supported by grants from the Australian Computer Research Board and the Australian Research Grants Scheme. The prototype version of BUILD was initially developed on a machine kindly loaned by IBM Australia Ltd. The technical assistance of Bala Balachandran in particular and other members of the Architectural Computing Unit is gratefully acknowledged.

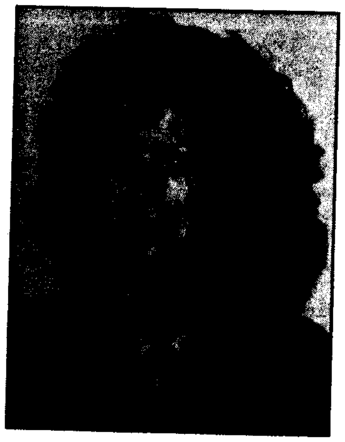
8. References

[1] A. Newell and H. Simon, *Human Problem Solving*, Prentice–Hall, New Jersey, 1972.
[2] O. Akin, *Models of Architectural Knowledge: An Information Processing Model of Design*, PhD Thesis, Carnegie–Mellon University, 1979.
[3] D.A. Waterman, *A Guide to Expert Systems*, Addison–Wesley, Reading, Massachusetts, 1986.
[4] J. McDermott, 'R1: a rule–based configurer of computer systems', *Artificial Intelligence*, **19**, 1982, pp. 135–173.

[5] G. Stiny, 'Introduction to shape and shape grammars', *Environment and Planning B*, 7, 1980, pp. 343–351.
[6] H.A. Simon, *The Sciences of the Artificial*, 2nd edition, MIT Press, Cambridge, 1981.
[7] J.S. Gero and R.D. Coyne, 'Logic programming as a means of representing semantics in design languages', *Environment and Planning B*, **12**, 1985, pp. 351–369.
[8] J.S. Gero and R.D. Coyne, 'Knowledge-based planning as a design paradigm', *Preprints, IFIP Working Conference on Design Theory in Computer-Aided Design*, University of Tokyo, 1985, pp. 261–295.
[9] W.F. Clocksin and C.S. Mellish, *Programming in Prolog*, Springer–Verlag, Berlin, 1981.
[10] W.J. Mitchell, *The Logic of Architecture*, Prentice–Hall, New Jersey, (in press) 1986.
[11] M.A. Rosenman, *BUILD: User's Manual*, Architectural Computing Unit, University of Sydney, 1985.
[12] J.S. Gero, M.A. Rosenman and C. Manago, 'A model-based expert system shell', *IAAIC86*, Melbourne, (to appear).
[13] R.D. Coyne and J.S. Gero, 'Semantics and the organization of knowledge in design', *Design Computing*, **1**, 1, 1986, pp. 68–89.

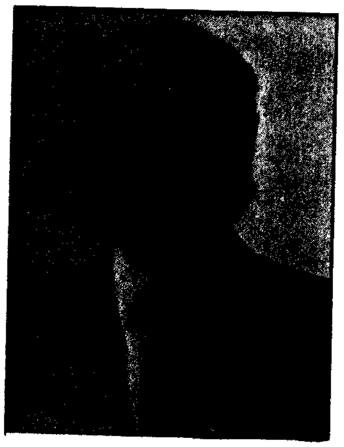
About the authors

Rivka Oxman



Rivka Oxman is currently a researcher in the Architectural Computing Unit in the University of Sydney, on leave from the Israel Institute of Technology — Technion. She received her Bachelor of Architecture and MS in architecture from the Technion. Prior to joining the Technion she practised architecture for ten years. Her research work is on knowledge-based design systems in architectural design. She is particularly concerned with incorporating the semantics of designed objects into knowledge-based systems.

John S. Gero



John S. Gero is Professor of Architectural Science and Director of Research for the Architectural Computing Unit in the University of Sydney. He received his Bachelor of Engineering from the University of New South Wales and his Master and PhD in architecture from the University of Sydney. He has been a Visiting Professor at Columbia University, Strathclyde University (Glasgow), National Institute of Applied Sciences at Lyon and University of California at Los Angeles and has lectured at over 120 universities and research institutes. He has held research positions at University of California at Berkeley, MIT and Harvard University. His research is concerned primarily with knowledge-based design systems and his current work includes developing interactions between expert systems and commercial CAD systems, generative design systems, machine learning in the domain of designs and knowledge-based systems which formulate design systems. He is the co-author/editor of eight books including *Knowledge Engineering in Computer-Aided Design* and *Expert Systems in Computer-Aided Design* (to appear), and over 150 papers. Professor Gero is on the editorial boards of numerous journals including *Design Computing*, *Artificial Intelligence in Engineering* and *Artificial Intelligence in Engineering Design and Manufacture*.