بسم الله الرحمن الرحيم

AIN SHAMS UNIVERSITY

Faculty of Engineering

COMPUTER-AIDED ARCHITECTURAL DESIGN TECHNIQUES

BY

SAMIR SADEK HOSNY

M.Sc. Architecture, Ain Shams University

A Thesis Submitted in fulfilment of
The Requirements of The
Ph.D. Degree in
ARCHITECTURE

SUPERVISED BY

Prof. Dr. MOHAMED ZAKI HAWASS
Professor of Architecture
Architectural Engineering Department
Faculty of Engineering,
Ain Shams University

and

Prof. Dr. MOHAMED ABDELHAMID SHEIRAH
Professor of Automatic Control
Electronics and Computer Engineering Department
Faculty of Engineering,
Ain Shams University

CAIRO, EGYPT
(1990)

To

MY PARENTS,

MY WIFE, and

MY SONS, AMR and ALAA

BOARD OF EXAMINERS

Signature

1. Prof. Dr. Emam Mohamed Shalaby
Professor of Architecture,
Head of the Architectural
Engineering Department,
Faculty of Engineering,
Ain Shams University,
Cairo, Egypt.

2. Prof. Dr. Mohamed Tawfik Abdel Gawad
Professor of Architecture,
Head of the Architectural Department,
Faculty of Fine Arts,
Helwan University,
Zamalek, Cairo, Egypt.

3. Prof. Dr. Mohamed Zaki Hawass
Professor of Architecture,
Architectural Engineering Department,
Faculty of Engineering,
Ain Shams University,
Cairo, Egypt.

4. Prof. Dr. Mohamed Abdel Hamid Sheirah
Professor of Automatic Control,
Electronics and Computer
Engineering Department,
Faculty of Engineering,
Ain Shams University,
Cairo, Egypt.

Date : 2/3/1991

STATEMENT

This dissertation is submitted to The Architectural Engineering Department, Ain Shams University for the Degree of Ph.D. in Architecture.

The work included in this thesis was carried out by the author in the Architectural Engineering Department, Faculty of Engineering, Ain Shams University, Cairo, Egypt, and in the Architectural Engineering Department, The Pennsylvania State University, Pennsylvania, U.S.A., from June 1987 to November 1990.

No part of this thesis has been submitted for a degree or a qualification at any other University or Institution.

Date : 14/12/1990

Signature :

Name : Samir Sadek Hosny

ACKNOWLEDGEMENT

I wish to express my deepest gratitude to all my supervisors for their supportive guidance during my staying at The Pennsylvania State University in the U.S.A. and after my return to Ain Shams University in Cairo, Egypt.

I am deeply grateful to professor Dr. Mohamed Zaki Hawass, Professor of Architecture, The Architectural Engineering Department, Faculty of Engineering, Ain shams University, Cairo, Egypt, for his guidance, useful suggestions and valuable comments for this study.

Many thanks are also due to professor Dr. Mohamed Abdel Hamid Sheirah, Professor of Automatic Control, Electrical Engineering Department, Faculty of Engineering, Ain Shams University, Cairo, Egypt, for his encouragement, concern, and enthusiasm about this work.

I am deeply indebted to Dr. Victor E. Sanvido, Associate Professor, The Architectural Engineering Department, and Dr. Loukas N. Kalisperis, Associate Professor, The Architectural Department, The Pennsylvania State University, Pennsylvania, U.S.A., for their continuous guidance and supervision, and their valuable discussions and revision of this work during my staying at The Pennsylvania State University.

ABSTRACT

The (ICAAD.DSS) model presented in this study provides a framework for an integrated computer-aided architectural design decision support system. The framework is based on a unified approach to computing in architecture which is based on a holistic view of the architectural design process. The proposed model shifts the focus from product to process, and views the design problem as a goal-oriented, problem-solving activity that allows a design team to identify strategies and methodologies in the quest for design solutions.

The framework will include; the different users of the system, the task environment and the decision support system. The different building stones for the DSS framework are: the user interface or the browser, the communicator or the dialog generation and management system, the databank, the database management system, the program bank, and the expert system component.

The browser supports the visual representation of the design structure, where the different nodes and their interconnecting links of the model are displayed on a canvas of virtually unlimited size.

The communication between the man-man and man-machine is achieved by using a dialog generation and management system working through a blackboard architecture, in which the blackboard serves as a global representation of the design solution.

The databank or the program bank includes all the needed information and application programs for the different activities encountered in the design process, they are accessible to all the users on a "read only" basis to prevent data corruption.

The proposed database management system provides functions like: creation and deletion, a dictionary, updating, querying and retrieval, sharing, protection and recovery. Integration, consistency, and maintenance of relationships between the stored data is achieved through

using a single integration point; the moving bi-directional database concept.

This study introduces a new environment for the use and integration of computers in the architectural design process.

KEY WORDS: Computer-aided architectural design, Integration, DSS (Decision Support System), Problem-solving, ICAAD.DSS (Integrated Computer-Aided Architectural Design Decision Support System.

TABLE OF CONTENTS

Page

ABSTRACT i

TABLE OF CONTENTS iii

LIST OF FIGURES v

LIST OF ACRONYMS viii

GLOSSARY OF TERMS x

CHAPTER (1)
INTRODUCTION 1
References for chapter 1 8

CHAPTER (2)
THE EVOLUTION OF DESIGN METHODS 10
2.1. The history of design methods 12
2.2. First generation methods 13
2.3. Early design models 16
2.4. Criticism of first generation methods 39
2.5. Second generation methods 41
2.6. Comparison between first and second generation methods 55
References for chapter 2 59

CHAPTER (3)
PROBLEM SOLVING AND SEARCH IN ARCHITECTURE 66
3.1. Problem solving as a model for architecture . 66
3.2. The "Problem Space" and the "Task Environment" of architecture 70
3.3. Problem finding and methods for search in architecture 73
3.4. Artificial intelligence 85
3.5. Expert systems and architecture 89
References for chapter 3 99

CHAPTER (4)
COMPUTER APPLICATIONS IN ARCHITECTURE 105
4.1. Different areas of architectural computing .. 105
4.1.1. Design applications 106
4.1.2. Technical applications 125
4.1.3. Production applications 133
4.1.4. Business and management 140
4.2. Different computer-aided design tools 143
4.2.1. Representation 144
4.2.2. Simulation 148
4.2.3. Generation 153
4.2.4. Optimization 159
References for chapter 4 164

Page

CHAPTER (5)
A FRAMEWORK FOR AN INTEGRATED COMPUTER-AIDED ARCHITECTURAL DESIGN DECISION SUPPORT SYSTEM 176
5.1. The architectural design process 177
5.2. The "BASED" model: An information processing model for the architectural design process .. 179
5.3. The proposed approach 188
5.4. A conceptual model for the decision support system, the (ICAAD.DSS) model 200
5.4.1. The dialog generation and management system (DGMS) 200
5.4.2. The database or the databank (DB) 205
5.4.3. The database management system (DBMS) 210
5.4.4. The Program Bank (PB) 212
5.4.5. The expert system component (ES) 213
References for chapter 5 219

CHAPTER (6)
THE DATA STRUCTURE FOR THE ICAAD.DSS MODEL 223
6.1. Information levels for the architectural design process 223
6.2. Different data models 234
6.3. Which data model best represents the architectural design problems? 239
6.4. The moving/bidirectional database concept ... 244
6.5. The internal representation of a design abstraction 247
References for chapter 6 263

CHAPTER (7)
THE VISUAL REPRESENTATION OF THE DESIGN STRUCTURE (THE BROWSER), AND THE SUGGESTED USER INTERFACE .. 268
7.1. The browser 268
7.2. The suggested user interface (hypermedia) ... 269
References for chapter 7 282

CHAPTER (8)
CONCLUSIONS, SUMMARY AND RECOMMENDATIONS
Conclusions 283
Summary 285
Recommendations 290

APPENDIX 292

LIST OF FIGURES

Figure — Page

CHAPTER (2)

(2.1) Alexander's model 17
(2.2) Alexander's sets and diagrams 21
(2.3) Broadbent's model 23
(2.4) Wade's model 27
(2.5) The main phases of design according to Archer's model 29
(2.6) Layout generation using DOMINO 33
(2.7) Backtracking by the DOMINO floor plan layout program 35
(2.8) The three-stepped process of design according to Asimow 39
(2.9) A diagram for the IBIS method 47

CHAPTER (3)

(3.1) Newell's and Simon's map of the problem solving process 72
(3.2) A tree structure for a depth-first search ... 76
(3.3) The decision making structures 76
(3.4) A tree structure for a breadth-first search . 78
(3.5) The generate-and-test process 80
(3.6) The hill-climbing process 81
(3.7) The means-end-analysis process 84
(3.8) The induction process 84
(3.9) The commercial offspring of artificial intelligence 87
(3.10) Expert systems in abstraction the Medieval Architectural Consultant 93
(3.11) A residential design proposed by the Mies Generator 95

CHAPTER (4)

(4.1) A perspective tour through the streets of Washington D.C. 108
(4.2) The shadow projection from "One Magnificent Mile" 109
(4.3a) A 3-dimensional view of a building, rendered to give a more realistic effect 111
(4.3b) A wireframe, and planes filled in as solids . 112
(4.4) Exterior perspective views, by the ARK-2 system 116
(4.5) A 3-dimensional isometric by the OXSYS system 116
(4.6) An adjacency matrix, a beginning step in space planning 119
(4.7) A trial floor plan, generated from functional requirements 121

Figure		Page
(4.8)	A computer-generated room finish schedule	122
(4.9)	Computer-generated 3-dimensional contours	123
(4.10)	A map produced by printing variable data values	124
(4.11)	The use of computers in structural design	127
(4.12)	Energy performance analysis output in a graphical form	130
(4.13)	Contour plot (plan view) of room surface luminance values	132
(4.14a)	Each layer in the drawing can be accessed and plotted alone by itself	135
(4.14b)	Use of multiple layers	136
(4.15)	Different shapes for columns could be evaluated	138
(4.16)	A 2-D, wireframe, and a solid modeling representation	146
(4.17)	An output from the SOLAR5 simulation program	150
(4.18)	The output from SYMAP and SYMVU programs	152
(4.19)	An isometric output from the program TOPAZ	154
(4.20)	An extract from the transformation rules for Frank Lloyd Wright's Prairie Houses	156
(4.21)	Designing according to panel architecture rules after the Miesian style	157
(4.22)	Multi criteria optimization for a school design problem based on simulation	161

CHAPTER (5)

(5.1)	The BASED process	181
(5.2)	The cyclical (iterative) design process	182
(5.3)	The BASED model of the design process for one phase	183
(5.4)	Design states and transitions	184
(5.5)	The different design phases and procedures in the ADP	186
(5.6)	The CAAD process for a single phase	189
(5.7)	The suggested framework for the decision making system	191
(5.8)	The three levels of the external environment	199
(5.9)	The ICAAD.DSS model	201
(5.10)	The dictionary/directory system, the moving/bidirectional database	208

CHAPTER (6)

(6.1)	The three levels of data abstraction	227
(6.2)	The three levels of design abstraction	227
(6.3)	The DD/DS philosophy	229
(6.4)	A sample hierarchical database	235
(6.5)	A sample network database	236
(6.6)	A room database from a relational database model	236

Figure

Page

(6.7) A sample generic E-R diagram 238
(6.8) An E-R diagram showing a room-surface scheme 239
(6.9) The three dimensional view for the moving/bidirectional database 245
(6.10) Defining the different phases of design 253
(6.11) Different levels of abstraction and detail in the ADP 262

CHAPTER (7)

(7.1) The browser's screen 268
(7.2) An example of hypermedia's data structure ... 271
(7.3) A conceptual window of a "sliding door" object 274
(7.4) A graphical window of a "sliding door" object 276
(7.5) A textual window of a "sliding door" object . 277

LIST OF ACRONYMS

The following terms and acronyms are used in this thesis:

ADP = Architectural Design Process.
AI = Artificial Intelligence.
AIDA = Analysis of Interconnected Decision Areas.
AIA = American Institute of Architects.
BASED = Briefing, Analysis, Synthesis, Evaluation and Decision.
BF = Breadth-First.
CAD = Computer-Aided Design.
CAAD = Computer-Aided Architectural Design.
CADD = Computer-Aided Design and Drafting.
CPM = Critical Path Method.
CRT = Cathode Ray Tube.
DB = Database or Databank.
DBMS = DataBase Management System.
DD/DS = Data Dictionary/Directory System.
DDL = Data Definition Language.
DF = Depth-First.
DGMS = Dialog Generation and Management System.
DML = Data Manipulation Language.
DQL = Data Query Language.
DSS = Decision Support System.
EDP = Electronic Data Processing.
E-R = Entity-Relationship.
ES = Expert System.
GAT = Generate and Test.
G.D.S. = Graphic Decision Systems, or Graphic Design Systems.
HC = Hill-Climbing.
HDF = Heuristic-Depth-First.
HOK = Hillmuth, Obata and Kassabaum.
HVAC = Heating, Ventilating and Air Conditioning.
IBIS = Isuue Based Information System.

ICAAD.DSS = Integrated Computer-Aided Architectural Design Decision Support System.

M-E-A = Means-End-Analysis.

OOP = Object Oriented Programming.

PB = Program Bank.

PERT = Program Evaluation and Review Technique.

RIBA = Royal Institute of British Architects.

SOM = Skidmore, Owings and Merrill.

TE = Task Environment.

TOPAZ = Technique for Optimal Placing of Activities into Zones.

GLOSSARY OF TERMS

This glossary contains most of the significant terms likely to be encountered in this thesis.

Algorithm: A sequence of steps or instructions for solving a problem. As contrasting with heuristic.

Analogy: The representation of one phenomenon by a different phenomenon, as in the representation of time by viewing different views of the same object from different viewing points.

Analysis: The process of resolving or separating a problem, process, situation, or object into its component parts and discovering the relationships between those parts, or the results of that process.

Appraisal: (In CAD), the process of judging or evaluating a design proposal.

Architectural computing: The use of computers in the architectural design process.

Artificial intelligence: The endowing of machines with some of the characteristics of intelligence (learning, recognition, etc.) normally associated with humans. It is the discipline that is concerned with programming computers to do clever humanoid things, but not necessarily in a humanoid way.

Association matrix: A matrix in which numbers indicate the relative importance of the association between facilities or opaooo. Used in facility planning.

Attribute: In computer graphics, any label or characteristic associated with an entity, such as the cost of the item represented by an object. An attribute occupies one or more fields in a record.

Backtracking: Executing an operation in reverse.

Backward chaining: A goal-driven method of reasoning that proceeds from the desired goal to the facts already known.

Browsing: Sifting through data listings in search for something.

CAAD: Computer-aided architectural design, which is architectural design supported by structured data and adequate computer programs to elaborate these data.

CAD: Computer-aided design, or design supported by structured data and computer programs to elaborate these data.

Cathode-ray tube (CRT): visual display device (the monitor) in which a beam of electrons is directed to produce an image on the surface of an evacuated glass tube.

Cluster diagram: A diagram showing the associations between different facilities in facilities planning; usually derived from an association matrix. Same as Bubble Diagram.

Code: Rules used to convert data from one representation to another. Same as Encode.

Compilation: The process of translating an algorithm written in a high-level programming language into machine code for a particular computer. The program that does this is called a compiler.

Computer architecture: The overall manner in which the various components of a computer are integrated to form a single system.

Data: General expression used to describe any fact, number, group of characters or symbols, or other value used by a computer program.

Database: A collection of structured inter-related data stored together in a way that facilitates the addition, modification, and removal of required data for various applications.

Database management system (DBMS): A computer program, or a set of programs that organizes and provides access to a database.

Data dictionary: A facility that tells what kind of data are there in the system.

Data directory: A facility that tells where is the needed data located.

Decision support system (DSS): Is an interactive computer-based system that helps decision makers utilize data and models to solve unstructured problems.

Design methods: It is a specific branch of design theory that copes with processes designed in order to increase the quality of design activities.

Digitizer: A device for converting drawn shapes into digital data, usually by pointing to the vertices of a shape with a special stylus or digitizer pen while the shape lies on a special table that is able to record the position of the pen.

Error message: A message that a program gives to indicate that an error has occurred, which may or may not require user intervention.

Expert system: A computer program that models the performance of human experts within their domains of expertise. It uses knowledge, facts and reasoning techniques to solve problems that normally require the abilities of human experts.

Facilities planning: The process of laying out rooms, spaces or parts of an organization within an existing building or as part of a new building design.

Generation: The synthesis of a solution by a computer program.

Geometric modeling: The representation of the geometry and topology of an object.

Hardware: Physical equipment, such as a computer, display screen, disk drives, etc., as opposed to software.

Heuristics: Rules of thumb. Rules based on experience rather than on axiom. These are criteria, methods, or principles for deciding which among several different courses of action promises to be the most effective in achieving a certain goal.

Icon: A symbol used to represent a command or mode of use. Usually the icon depicts some object familiar to the user: eg., an image of a pen in a computer drafting or painting program.

Inference engine: A mechanism in an expert system that infers new knowledge from already existing knowledge.

Information processing: The production of information by accessing and processing data on a computer; traditionally called electronic data processing.

Input: Information or data entered into a computer via a keyboard, tape, graphic device, or any other means.

Interface: A boundary between two computer components or systems, or between a computer and a human user.

Iterate: To repeat.

Layer: Computer-aided drawings are often subdivided into layers, analogous to overlaid sheets of tracing paper. Different layers may be displayed or hidden at will. One layer might be used for textual annotation, another for dimensions, another for electrical layout, another for plumbing, etc.

Linear programming: A technique for finding an optimum combination when there may be no single best one.

Lock: A mechanism activated by a multiuser system when a record from a file is accessed, temporarily preventing other users from accessing the same record.

Mainframe: The largest, fastest, and most expensive class of computers capable of supporting many computer terminals.

Mapping: A mathematical process in which elements of one group of objects are associated with one or more elements of another group of objects.

Menu: A list of options, usually either displayed on a screen, or fixed to a digitizer tablet, from which the user can select the needed program action.

Meta rules: Rules about rules.

Microcomputer: A small and relatively low-cost class of computers.

Mouse: A device that fits in the palm of the hand, and is moved over a flat surface to cause a corresponding relative movement of a pointer on a display screen.

Natural language: Any human-human language, such as Arabic, or English.

Operations research: A branch of applied mathematics that aims to improve decision making by building mathematical models.

Optimization: The process of trying to find the very best solution in terms of efficiency, cost, time, or some other criterion or combination of criteria.

Output: The results or messages produced by a computer.

Package: A collection of programs forming an integrated whole.

Pascal: A high-level programming language, named for Blaise Pascal (a French mathematician) designed to support the concepts of structured programming.

Pixel: The smallest area on the screen of a raster display whose characteristics (color and brightness) can differ from those of its neighbors.

Procedure: Any sequence of actions, as in an algorithm.

Program: A series of statements or instructions that directs a computer to perform a sequence of operations. An application program deals with something in the real world outside the computer. A utility program is concerned with the mundane processes of handling files and writing programs (rather than with applications outside the computer), while

a system program is one that handles very fundamental operations deep inside the computer.

Prompt: The character(s) output by a program to show that it is ready to accept input.

Read: To retrieve data from mass storage.

Scrolling: The process of moving lines up and down a computer screen so that the text or pictures appear to scroll off the top and bottom of the screen.

Shape grammar: A grammar for manipulating shapes according to rules, analogous to linguistic rules.

Simulation: The process of using or operating a model encoded in a computer to learn about the behavior of the reality being modeled.

Software: A computer program.

Solid modelers: Programs that model solid objects.

Spatial synthesis: The generation of spatial layout or organization, as in layout or facilities planning.

Stylus: A hand-held pointer for indicating choices from a menu, or positions on a drawing, such as a digitizer pen, or a light pen.

Tablet: A small digitizer board, often used with a stylus and a superimposed menu as a means of indicating commands to a drafting system.

Terminal: Any device, such as a screen and keyboard, or printer and keyboard, at which data can be input to and output from a computer. Other synonyms are VDU and workstation.

Touch screen: A computer screen with infrared sensors placed around the frame (or with a special transparent grid laid on the screen) that allows a user to pick a menu item by just pointing or touching it.

Transparency: The more a program hides the computer's internal operations from a user, the more transparent it is.

Update: To bring a record up to date by erasing its previous contents and replacing them with more recent data.

User: Any one who uses a computer

Window: A bounded area created within a computer screen to contain a separate image from the rest of the screen.

Wire frame: A form of drawing in which every line comprising the drawing is visible, as though made of wire.

Word processor: A computer program, or entire computer system that facilitates the typing, editing, and printing of documents.

Workstation: In a computer drafting system, the workstation is the combination of display screen, keyboard, and perhaps a tablet and/or stylus or mouse for each user.

Zoom: A term used in computer graphics (analogous to its original use in photography), it refers to examining a drawing from a greater or lesser distance in order to fit either more or less of the drawing on the screen.

CHAPTER 1

INTRODUCTION

CHAPTER 1

INTRODUCTION

"If computer-aided architectural design techniques are to be developed to their full potential, it will necessitate a diversion of professional effort away from the design of individual buildings in favor of a commitment to designing better design methods and models." (Maver, T, W. 1988).

Computer-aided design has paralleled developments in design methodologies. Design methodology in this respect is a specific branch of design theory that copes with processes designed in order to increase the quality of design activities. Computer-aided architectural design (CAAD) is: architectural design supported by structured data and adequate computer programs to elaborate these data. In its most developed form, CAAD is a decision support system to be employed in architectural design processes. (Bax, 1986).

In order to develop these systems, one has to study the structure of the design process. Design methodology focuses on these processes. On the other hand, one has to know what is possible in the field of information and computer technology. CAAD as such, is a field of application of a theory of design process and decision making in a context of available computer techniques, networks and communication systems. In accordance, this study will address the following subject:

DESIGN METHODS, AN APPROACH TO A COMPUTER-AIDED ARCHITECTURAL DESIGN PROCESS.

However, computing in architectural design as it is being implemented today, is not integrated into the basic design process (Kalisperis, 1988). Recent developments in CAAD program packages have provided some compatible programs, enabling architects to perform different tasks using the same

data. The scope of such programs is limited to small tasks and simple manipulations within small projects (Beheshti and Monroy, 1987). The true integration of computers into the architectural design process is hardly progressing (Orr, 1985). The current computer aided design systems still do not meet the needs and expectations of the architect (Lindhult, 1987), and the need for a "flexible" architectural design decision support system is imperative (Kalisperis, 1988).

THE PROBLEM STATEMENT:

The lack of computer integration in the architectural design process prevents the architect from using the full capabilities of computers, thus limiting his abilities to: input design information, test alternative strategies accordingly, use the output from a certain program as an input to another, and view the results in a realistic form (Greenberg, 1984). Besides, the different computer-aided design application programs are evaluative and generative, they function individually not as a part of a system, thus making cross referencing and connections impossible. They were not interrelated and did not cover the architectural design process in its totality (Kalisperis, 1988). The different CAD tools are not packaged into a system that follows naturally through the design process, which makes the Architect unable to "enter" it at any desired mode or phase (Broadbent, 1987).

The emphasis in this thesis then, will be to introduce the computer as early as possible in the design process, and present a framework that will attempt to integrate the whole computer-aided architectural design process, based on a holistic view of the architectural design problem solving. If the architect works totally in the realm of pencil and tracing paper until an idea is fully developed, then he is

depriving himself of a potentially valuable design tool, and the computer will remain in the domain of the CAD operator.

THE VALUE OF SOLVING THE PROBLEM:

The solution, presented in a model form, will allow for an integrated interaction between the architect and the computer with the intent of enhancing creativity, and supporting decision making. It should enable the architect to start with any phase of the architectural design process and deal with any level of design. It will also shift the emphasis of CAD systems from being "product oriented" to "design procedure oriented", i.e., it will shift the focus from "product" to "process".

The thesis will develop a framework based on: i) knowledge of design methods (theories, concepts and practices), and ii) knowledge of computer capabilities (computer hardware and software).

The proposed framework of this study will allow for an interactive dialog between the architect and the computer, making the computer not only a useful tool, but also a reliable partner through all the phases of the architectural design process. By integrating the computer at this level, architects can model and visualize architectural design ideas in new and exciting ways.

It is believed that the proposed framework will expand the role of computers from their already established ability in representing and evaluating design solutions, to the ability of assisting in the creative and judgmental functions of the design process as well.

THE GOAL OF THE STUDY:

The purpose of the research is to develop a framework that makes possible the achievement of an integrated computer-aided architectural design process.

THE OBJECTIVES OF THE STUDY:

The study should accomplish the following objectives:

1. Define the different design methods in architecture.
2. Understand the key issues of problem solving, and decision making in the architectural design process.
3. Illustrate the significance of computers and their different applications in architecture.
4. Construct a model to provide a framework for an integrated computer-aided architectural design (CAAD) process.
5. Define the data structure for the model.
6. Identify the needed user interface and conclude.

THE SCOPE OF THE RESEARCH AND THE LIMITATIONS:

The focus of this research is on the architectural design process aided by computers. The result is not a computer program or software, but a framework for an integrated computer-aided architectural design process, through which computers will be able to assist the architect in his decision making throughout the entire design process.

THE RESEARCH METHODOLOGY:

The methodology for conducting this research included four types of activities; i) reviewing different literature, ii) conducting field trips to select architectural schools, and attending a conference on design methods and computer

based models, iii) attending computer courses in which computer tools are used in the design process, and iv) analyzing the results to develop the model.

1. Objective number "1" is met by reviewing the history of different "design methods" and "models of the design process". The search included "first generation methods", which were best suited to the solution of "well constrained" or "well behaved" design problems, as well as "second generation methods", which dealt with "wicked" or "ill-structured" design problems as a result from the difficulties experienced in the application of the "first generation methods". An assortment of books, periodicals, papers, technical reports and thesis were consulted.

2. Objective number "2" is met by reviewing the literature to understand the key issues of problem solving and decision making in the architectural design process. The methodology for achieving the above objectives was through a general literature survey of different sources such as books, periodicals, papers, technical reports and thesis, etc. The result of those two objectives achieved is the second, and the third chapters in this thesis respectively.

3. In order to achieve objective number "3", it was important to review appropriate literature and conduct field trips to select architectural schools. Attending the Engineering Design Research Conference, held at the University of Massachusetts at Amherst, during the period from the 11th to the 14th of June 1989 was most beneficial. It was also important to attend some computer courses in which computer tools are used in the design process, such as: G.D.S. on the VAX (main frame),

and AUTOCAD on the PC (microcomputer), to explore the capabilities of the different computer systems.

4. To provide a framework for an integrated computer environment, it was necessary to build a model of the computer-aided architectural design process, which will attempt to provide a conceptual framework for the integration of the whole architectural design process. This model considers the architectural design process as a goal-oriented, problem-solving activity, where each state can be obtained by the application of an operation to a previous state, in order to identify different strategies and methodologies in the search for architectural design solutions.

5. To achieve objectives 4, 5, and 6, it was necessary to design the model, define the data structure, and design the suggested user interface respectively, which required knowledge of design methods, as well as knowledge of computer hardware and software.

The professionals' opinions for the evaluation and feedback, were available on a personal interview basis, as well as through correspondence through the mail service. Accordingly, some field trips had to be conducted to select architectural schools, to get the academics' opinion about the suggested model for the new computer environment.

The criteria for the selection of the architectural schools, was based on:

a. They should be well known for their use of computers in the architectural design process.
b. Their willingness of participation in the study.
c. Their being within reasonable travelling distances from Penn State; the home town of the author during his data

collection for this thesis, to facilitate the process of data collection, analysis and feedback.

The criteria for the selection of the personnel included:

a. They should be well known for their comprehensive writings about the subject of computer-aided design in general, and about computer-aided architectural design in particular.
b. They should have been studying or conducting research in the computability of design for at least 5 years.

This period of 5 years, is a minimum requirement for experience so that the interviewee would have had already accomplished a personal point of view regarding the computability of design and/or the computability of the architectural design process in particular.

The feedback from the interviews was very important to the refinement of the model for the proposed framework.

REFERENCES FOR CHAPTER 1:

Bax, M. F.:
"Domain Theory: Applications for CAAD." In Open House International, vol. 11, no. 2, 1986.

Beheshti, M.; and Monroy, M.:
"Requirements for Developing an Information System for Architecture." In CAAD Futures '87. Proceedings of the Second International Conference on Computer-Aided Architectural Design Futures, Eindhoven, The Netherlands, 20-22 May, 1987. Edited by Tom Maver and Harry Wagter. New York: Elsevier, 1988.

Broadbent, G.:
Design in Architecture. John Wiley & Sons, Ltd., New York, 1988, first published 1973.

Greenberg, D. P.:
"The Coming Breakthrough of Computers as a True Design Tool." In Architectural Record, September, 1984, pp. 150-159.

Kalisperis, L. N.:
A Conceptual Framework for Computing in Architectural Design. A Ph.D. Dissertation. The Architectural Department. The Pennsylvania State University, 1988.

Lindhult, M. S.:
"Towards an Intuitive Computer-Aided Design Process." In Proceedings of the 1987 Conference on Planning and Design in Architecture. Boston, Massachusetts, August 17-20, 1987. Edited by J. P.

Protzen. New York, The American Society of Mechanical Engineers, 1987.

Maver, T. W.:
Introduction for CAAD Futures '87. Proceedings of the Second International Conference on Computer-Aided Architectural Design Futures, Eindhoven, The Netherlands, 20-22 May, 1987. Edited by Tom Maver and Harry Wagter. New York: Elsevier, 1988.

Orr, J.:
"The Merits of Design Automation." In Computer Graphics World, January, 1985, pp. 83-84.

Protzen, J. P. ed.:
Proceedings of the 1987 Conference on Planning and Design in Architecture. Boston, Massachusetts, August 17-20, 1987. New York, The American Society of Mechanical Engineers, 1987.

CHAPTER 2

THE EVOLUTION OF DESIGN METHODS

CHAPTER 2

2. The Evolution of Design Methods.

2.1. The History of Design Methods.

2.2. First Generation Design Methods.

2.3. Early Design Models.

2.3.1. Alexander's Model, The Decomposition Theory.

2.3.1.1. Criticism of the Decomposition Theory.

2.3.2. Broadbent's Model.

2.3.3. Wade's Model.

2.3.4. Archer's Model.

2.3.5. Tabor's Model, The Space Allocation Model.

2.3.6. Jones's Model.

2.3.7. Other Models.

2.4. Criticism of First Generation Methods.

2.5. Second Generation Design Methods.

2.5.1. The IBIS Model.

2.5.2. The Pattern Language.

2.5.3. Squatters Technique.

2.5.4. Design as a Learning Process.

2.5.5. Function-Analysis Techniques.

2.5.5.1. Information Analysis.

2.5.5.2. Speculation.

2.5.5.3. Evaluation.

2.5.5.4. Synthesis.

2.5.5.5. Recommendations.

2.6. A Comparison Between First and Second Generation Design Methods.

CHAPTER 2

THE EVOLUTION OF DESIGN METHODS

In architecture, there are a number of independent theories known as "models of the design process", each relating to a particular aspect of architecture. Formulation of architectural design models is a result of the strong contemporary concern for definition of the design process and the creation of designed artifacts. They result from attempts to understand and externalize what actually takes place during the design process, and from the desire to devise ways to improve this process. According to Jones' (1966) terminology, it is changing from a "blackbox approach" to a "glassbox approach".

Although a certain number of design methods have been developed for architectural design, the majority of methods available have been "imported" from general design models and engineering design in particular (Gregory, 1966).

Architectural design method was developed by the great rationalist Viollet-le-Duc (Kalisperis, 1988). Method, as he sees it, requires meticulous attention to the "programme of requirements," the selection of an appropriate method of construction, and proper regard for the nature of materials."

Contemporary concern for the architectural design process, and design methods in general, has been strongly influenced by developments in applied mathematics and systems science which occurred during the middle of the century (Bazjanac, 1974). This has also been confirmed by Broadbent (1973, p. 272) who, in his analysis of contemporary thought on architectural design, writes that: "the new math, with a certain amount of statistics, has been almost as influential in the development of new design methods as all the other

sources and disciplines put together." This change in thought about the architectural design process closely followed the discoveries in operations research, computer, cognitive, and management sciences. The new concern about design was initially affected by theoreticians who brought ideas from mathematics to architecture (Bazjanac, 1974).

According to Cross (1966), the definition of a method in design is: the process, the procedure, and the practice of that design.

According to Heath (1984), a method in action is the construction and search of specific design problem spaces for specific design tasks. Even if it is conceded that the general goals of design and the general extended problem space of architecture can be described in some way, it need not be admitted that any useful generalizations can be made about the individual case (Heath, 1984). Method might still be individual and personal, so much confined to the black box of the individual designer's mind that it cannot be explained or clarified.

It is possible, that each new architectural problem requires a new and different method. Or again, method might be specified to building types, so that, for example, there would be one method for libraries, another method for schools, another for houses, and so on.

This chapter will show that it is impossible for method to be wholly or even largely subjective or internal. On the other hand, it will be argued (through the demonstration of different design models) that attempts to construct a single method for all architectural problems have not been wholly successful because of the variety of the different types of problems requiring different methods. However, method is not specific to each individual task: there are broad classes of

problems, but these classes are not identical with, and are much fewer than, building types.

2.1. The History of Design Methods:

The history of design methods as a recognized subject with impact on architecture, covers around thirty years, starting with the Oxford Conference on Design Methods (Jones and Thornley, 1963). Alexander's seminal book Notes on the Synthesis of Form was published in 1964, and the Birmingham Symposium, subsequently published as The Design Method (Gregory, 1966, took place in 1965). The first conference to deal specifically with design methods in architecture, was held at Portsmouth, England, in 1968 (Broadbent and Ward, 1969), and in that same year, the American Design Methods Group held its first international conference at Cambridge, Massachusetts (Moore, 1970).

A comprehensive, detailed, and critical review of the first design methods generated during the 1960's is found in Christopher Jones' book: Design Methods - Seeds of Human Futures, (Jones, 1970). Broadbent, as well, did the same for the more specifically architectural developments in his book: Design in Architecture, (Broadbent, 1973).

The year 1972 marks a decisive change in the direction of thinking on the subject: a point at which certain difficulties that had been experienced in the application of the theories and methods developed in the previous decade were, if not resolved, at least explained, by a new and broader theory. The new theory was brought to general notice by a discussion that appeared in the fifth anniversary report of the Design Methods Group in January, 1972 (Rittel, 1972b), marking the start of what became known as the second generation design methods. This chapter will address the

first and second generation design methods, as well as the different design process models associated to them.

2.2. First Generation Design Methods:

During the ten years from 1962 to 1972, a number of design methodologies were introduced. A common characteristic of these "first generation" methods (Rittel, 1972), was the view that the design process was a sequence of well defined activities based on the assumption that the ideas and principles of scientific methods could be applied to the design process as well (Popper, 1963), and (Kuhn, 1962).

If we agree with Wade's assumption (1977), in describing design in the most abstract way as a transformation A -> B, where A is some initial state, and B is a terminal state, and the two are linked by a vector represented by the arrow (->), then the first-generation methods had centered on the vector term. That is, they assumed that the present state of affairs and the final goal were known or could readily be discovered, and attempted to develop better methods of organizing the transformation of the initial state into the final state. The objective of these methods was in fact to find an algorithm, a logically rigorous set of rules for producing a satisfactory, or even optimum result. This "systems" approach had proved its effectiveness in military and space technology, where the actual goal was extremely simple and precise, though the equipment needed to realize it was extremely complex. The problems, in fact, were very well constrained by the laws of physics on the one hand, and by the limits of human psychophysical performance on the other (Heath, 1984), which is not the case in architectural design and planning. In architecture and town planning, the goals are complex and vague, whereas the equipment needed to realize them is extremely simple.

Studer, R.:

"Christopher Alexander's Notes on the Synthesis of Form; Review." In Architectural Association Journal, March, 1965.

Summers, L. H.:

Algorithms and Computational Methods in Building Design. A report submitted to the National Science Foundation (NSF), and Computer Integration in Construction (CIC), Pennsylvania, 1988.

Tabor, O.:

"Analyzing Communication Patterns." In The Architecture of Form, edited by L. March. London: Cambridge University Press, 1976.

Wade, J.:

Architecture, Problems and Purposes. New York: Wiley-Interscience, 1977.

CHAPTER 3

PROBLEM SOLVING AND SEARCH IN ARCHITECTURE

CHAPTER 3

3. Problem Solving and Search in Architecture.

3.1. Problem Solving as a Model for Architecture.

3.2. The Problem Structure: "Problem Space" and "Task Environment" of Architecture.

3.3. Problem Finding and Search Methods in Architecture.

3.3.1. Recognition.
3.3.2. Depth-First Search.
3.3.3. Breadth-First Search.
3.3.4. Generate and Test.
3.3.5. Hill-Climbing.
3.3.6. Heuristic Search, or Means-End Analysis.
3.3.7. Induction.

3.4. Artificial Intelligence.

3.5. Expert Systems and Architecture.

CHAPTER 3

PROBLEM SOLVING AND SEARCH IN ARCHITECTURE

In this chapter, a definition of the "problem space", as well as the "task environment" is presented, an overview of the different problem finding techniques and search methods in architecture is presented. The issues of artificial intelligence, as well as the potential of expert systems in architecture are addressed.

3.1. Problem Solving as a Model for Architecture:

It has been argued that first-generation models for architecture, are very narrow in their scope. They are based on a philosophy of "historical determinism", where civilization and environment are determinants of art form, or "personal determinism" where the individual becomes the means of understanding the art work (Abell, 1957). They attempted to generate a design methodology for architectural design problems from a limited base. On the other hand, second generation models, attempted to create specific tasks of architectural design from a more general model of activity.

This broader model, which supports architectural complexity as well as architectural contradiction, is derived from research into problem solving activity in general, and specifically from the work of Newell and Simon (1972), which incorporates much of what has been said by other writers such as Poincare (1952), and Polya (1945) and goes beyond them in seeking to construct a theoretical psychology of problem solving.

Newell and Simon's definition of a problem is:
"A person is confronted with a problem when he wants something, and does not know immediately what series of

actions he can perform to get it". (Newell and Simon, 1972). The method, or sometimes is said to be: the model of the activity, is that list of actions that must be performed to get from the problem state to the solution state, the "design"; the actions themselves are the problem solving process or the design process.

Later they continue: "To have a problem implies (at least) that certain information is given to the problem solver: information about what is desired, under what conditions, by means of what tools and operations, starting with what initial information, and with access to what resources. The problem solver has an interpretation of this information-exactly that interpretation which lets us label some part of it as goal, part as side conditions, and so on. Consequently, if we provide a representation for this information (in symbol structures), and assume that the interpretation of these structures is implicit in the program of the problem solving IPS {IPS is an acronym for "information processing system" for which an appropriate analogy may be mind.}, then we have defined a problem". (Newell and Simon, 1972, pp. 72,73).

They define the IPS as "a system consisting of a memory, containing symbol structures, a processor, effectors and receptors". Receptors gather information from the environment and effectors manipulate the environment through motor behavior. Memory contains individual symbols of tokens that stand for objects and other symbols and their relations. The processor is basically a symbol manipulator with the following functions: i) It converts the information provided by the receptors into a code that is internally consistent with the symbol structures of the system, ii) it transforms internal symbols and their relations, and iii) converts internal symbols into code that can be transmitted to the external world or the environment by the effectors. Newell and Simon

suggest that the processor consists of "atomic processes" a "working memory", and an "interpreter" that determines the sequence in which the processes are performed as a function of the symbol structures present in the working memory (Kalisperis, 1988).

Atomic processes represent the basic functions of the processor. When these are activated in particular sequences, the functions that the processor is responsible for are performed. The working memory provides the representational medium in which these operations take place. The interpreter determines the sequence in which these atomic processors are executed, so that the functions of the processors are performed properly.

Heath (1984) reports that the idea that problem solving is the distinctive feature of architectural activity is not new. As Reyner Banham put it that architecture and problem solving are two interchangeable ideas, he says "A problem well stated is a problem solved" (Banham, 1965).

Haider (1986) pointed out that Straus, Thorsen and Thorsen define problem as the perception of a situation requiring resolution, and problem solving as "situation changing" or "conflict resolving", and states that "In its most general sense, problem solving includes the actions involved in perceiving, analyzing, remembering, planning, alternative generating, evaluating and synthesizing".

The term problem solving is applied in architecture in a generic sense, and does not indicate sequential problem solving. Kalisperis (1988), pointed out that it is evident from Alexander's (1971) definition of selfconscious and unselfconscious design, as well as Newell's and Simon's (1972) definition of problem, that selfconscious architectural design does involve a "problem".

"To describe all of architectural design problem solving is clearly an abstraction" (Heath, 1984). He argues that not only is problem solving a small part of the total architectural design activity, but also there are many kinds of problems, he observed that much of what architects do at present, does have the character of puzzle solving, in that the objective is known; it is clear that there is an answer, and the area of work or amount of information involved is quite limited. The sizing of a lintel for a small opening in a masonry wall, or an eaves gutter for a roof, are examples. Again, it is the fact that much of architecture is merely puzzle solving that makes architecture possible in a reasonable time, and at a reasonable cost (Heath, 1984, p. 127).

According to Archea (1987): "architects approach a design in a manner that is antithetical to the problem solving ... architects are puzzle makers. They are primarily concerned with unique design contexts ... At the onset of the design process, neither the goals that the designer will attempt to meet, nor the elements or attributes that will be drawn upon to meet these goals can be stated by the designer. While both may exist in some pre-visual or pre-verbal form; neither can be fully articulated until the design process has been completed. The objective of design in architecture is to achieve a satisfactory fit between a "kit of parts" ... What the architect seeks is a set of combinatorial rules that will allow him or her to achieve a satisfying effect. In effect, the architect makes a puzzle in which each part of assemblage has a uniquely satisfying proposition in relation to the whole. The only problems which are addressed in this process are those which the architect introduces to satisfy his or her quest for satisfactory effect (Archea, 1987, pp. 32-33).

Design problems, of which architectural design problems are a subclass, are not puzzles. Design problems are

distinguished from other problems by the fact that their goal state cannot be predefined, in the sense that the designer do not know exactly what he is trying to achieve. We may know in general terms, for example, that we want to design a house for a family of six on a steeply sloping site, but before we can actually achieve this thing, our needs and wishes have to be made specific, and once that is done, producing the design may not be too hard. More generally, following Wade (1977), we can say that design problems are concerned with "closure of the terminal state", or deciding what it is that is to be done, and whether it can be done, and according to Simon (1970), establishing the "most satisficing solution". Additionally since architectural design problems are ill-defined problems, certain assumptions have to be made which in turn are influenced by the degree of severity, and the conceptual tools the architect/designer is using. Most architects find it difficult to articulate how they design (Archea, 1985; and Haider, 1986).

In the problem solving literature, a problem with a proper answer - that is an answer which is identifiable as correct - is called "well defined". On the other hand, a problem without criteria for establishing whether a solution is correct is called "ill defined". By this definition, the architectural design problem is considered to be ill defined (Wade, 1977).

3.2. The Problem Structure: "Problem Space" and "Task Environment" of Architecture:

Problem solving in general, constitutes a continuous series of events causing decisions along the time dimension. It starts with a certain state at a particular point in time, goes through intermediate stages, and then terminates either with a solution state or resignation at a later point in time. Often there are causal relationships that associate these

individual events sequenced over time. To develop a representation of the architectural design process, these static states must be selected in such a way that they are most informative, and explicitly specify the transformation necessary to establish the causal relationships between each state, its predecessors and its successors. In order to understand the taxonomy of such discrete event representations, the notion of "problem space" and "task environment" need to be considered.

According to Newell and Simon (1972): the description of the overall map of the problem solving process is illustrated as in figure (3.1):

"Proceeding inwards from the task environment, there is the problem space, in which problem solving takes place as search; then the methods which are the means by which search takes place; and finally the production system, which is the program organization by which the methods are realized in terms of elementary processes and basic characteristics of the IPS." (Newell and Simon, 1972).

The "task environment" may be roughly described as "situation" (Newell and Simon, 1972). They argue that it is impossible to describe the task environment itself. However, they suggested two ways to talk about it that do not involve misleading abstraction. One is to consider "alternative, isomorphic presentations of the same environment. We mean by the structure of the environment precisely the set of invariants that are preserved under translation from any one of the isomorphs to any other" (Newell and Simon, 1972). The other, they continue, "is to construct a hypothetical problem space that is objective only in the sense that all the representations that human subjects in fact use for handling the problem can be embedded as specializations in this larger space." In their work, this hypothetical space is constructed

by collecting different protocols of problem solving behaviors, from which different problem spaces can be extracted.

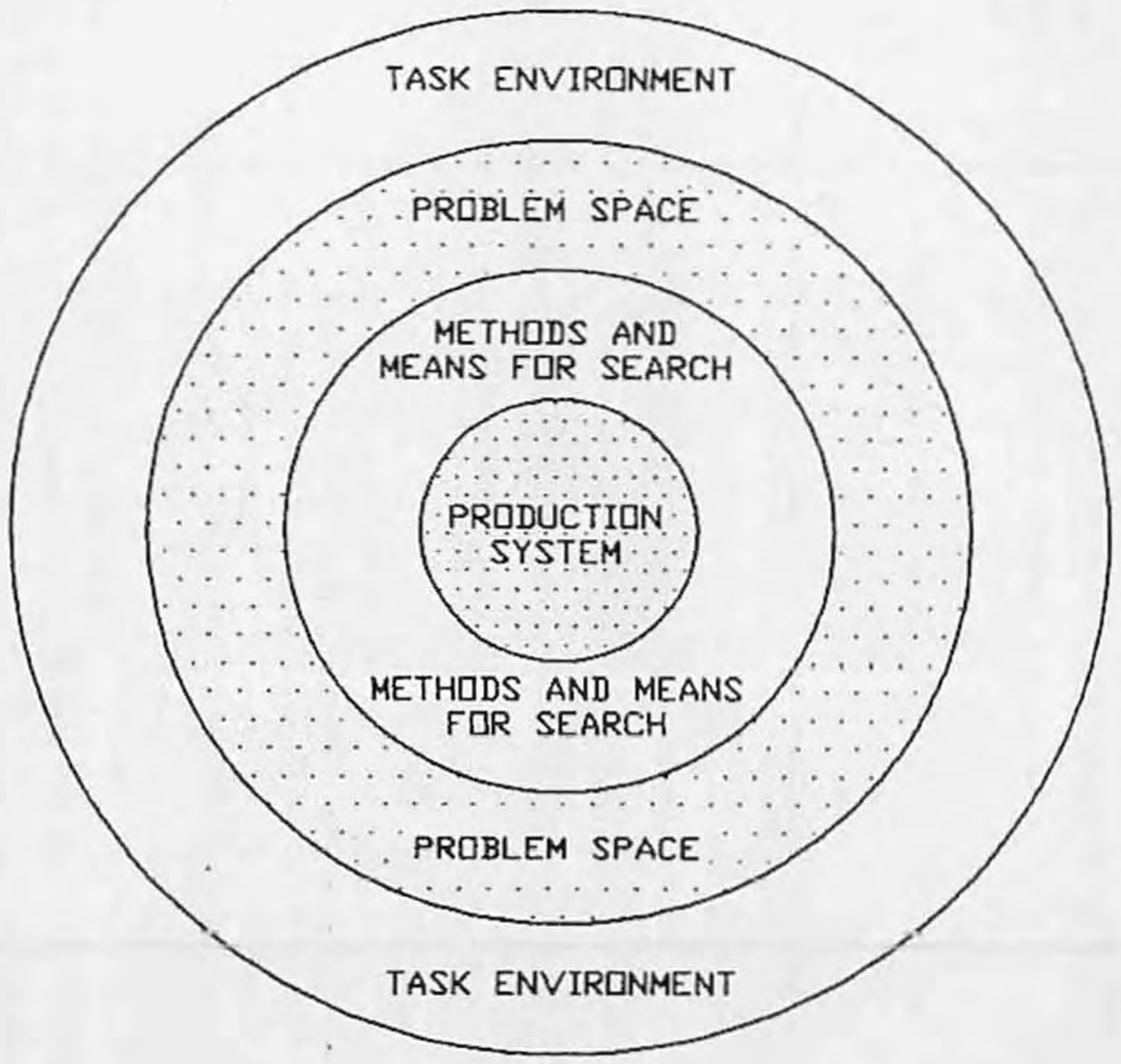


Fig. (3.1), Newell's and Simon's
map of the problem solving process.

On the other hand, the "problem space" is the problem solver's internal representation of the task environment (Newell and Simon, 1972). The construction of a problem space out of the elements provided by the task environment, is the first task of the problem solver. However, they presented little information about how the problem space is achieved. Different problem solvers may, and probably will use different problem spaces, since they will abstract from the task environment in different ways. It is also important to point

out here, that whether or not the problem solving effort will be successful depend largely on whether or not the problem space is a "good" representation of the task environment.

In the complex environment of architectural design problems, this may not be easily accomplished. The task environment may not contain a solution due to the conflicting demands inherent in the problem. According to Haider (1986), "this peculiarity, renders architectural problems different from puzzles".

3.3. Problem Finding, and Methods for Search in Architecture:

Search techniques consist of three main components (Barr, 1981). The first component, the database, describes the current domain-dependent situation as well as the goal. The second component is a set of operators that are employed to manipulate the database. The third component is the control strategy that decides which operators to apply, and in which sequence.

An equivalent architectural database example would consist of the building program, the site description, and the client's particular needs and requirements as the database. The architect's knowledge of how to transform a program requirement into the description of a three-dimensional space are the operators. The design strategy that the architect chooses to produce a final solution is equivalent to the control strategy.

Once an appropriate problem space is constructed, the next step for the problem solver is to proceed to search it for a solution. Newell and Simon (1972), identify three basic methods for search in design: recognition or knowing the answer; generate and test; and heuristic search (means-end-analysis). In addition there are some other methods for

search, like: depth-first-search, breadth-first-search, hill climbing, and induction.

According to Akin (1986), and Schmitt (1988), architectural search takes place on general, global levels, and on specific, local levels. Typical search methods employed on a global level are depth-first search, and breadth-first search. On a local level, specific goals of the designer can be achieved with an array of possible search strategies, of which the generate-and-test method, hill-climbing method, and heuristic search or means-end-analysis are particularly suited for computer implementations.

3.3.1. Recognition:

Recognition, according to Heath (1984), might appear trivial, but it is the basic procedure of selfconscious design, and also the most penultimate stage of more complex design procedures.

The triviality of the problem must owe something to the problem solver's repertoire of methods. The problem is not easy for all solvers. According to Newell and Simon (1972), recognizing the answer is considered to be the one universal method for solving problems. Recognizing an answer means that the answer was already in memory, and was simply evoked by the act of understanding the question.

Recognition processes are important for solving problems, but not because they can be used directly for solving hard problems, but because problem solving often proceeds by reduction; a hard problem is solved by replacing it with ostensibly easier problems. Eventually, if this reduction process is to succeed, some subproblems must actually be solved and not just reduced further. Recognition is very

often the way this last step is accomplished. It is said that the solver reduced the problem to something he already knew.

Recognition is applicable to all problem formulations because it is not sensitive to the details of the formulation. By the same token, when a problem is solved by recognition the usual measures of problem difficulty do not apply, for the result is immediate.

3.3.2. Depth-First Search:

This method is considered to be one of the basic methods of search in design. It is characterized as the allocation of the designer's attention to the siblings of a parent node before moving to the next parent node of the same depth, in a tree-like search space.

In a depth first search method, the nodes in the tree would be accessed as in fig. (3.2), only one node at each level is examined, unless failure requires the search to back up to explore alternatives previously ignored. This implies that the designer does not move to a next major issue, or another parent node before exploring all siblings of the current issue, or current node.

In terms of the potential success of a search process, the depth first method has some disadvantages; as a search method in a tree-like or lattice-like problem structures, refer to fig. (3.3), depth first search: i) does not guarantee optimal search of the tree, ii) does not insure finding a solution, iii) does not insure a balance of emphasis between multiple issues of equal priority that may be implied by the levels of the tree or lattice structure, and iv) becomes even more inefficient if the branches of the search tree which are being examined contain cyclic paths or are very large.

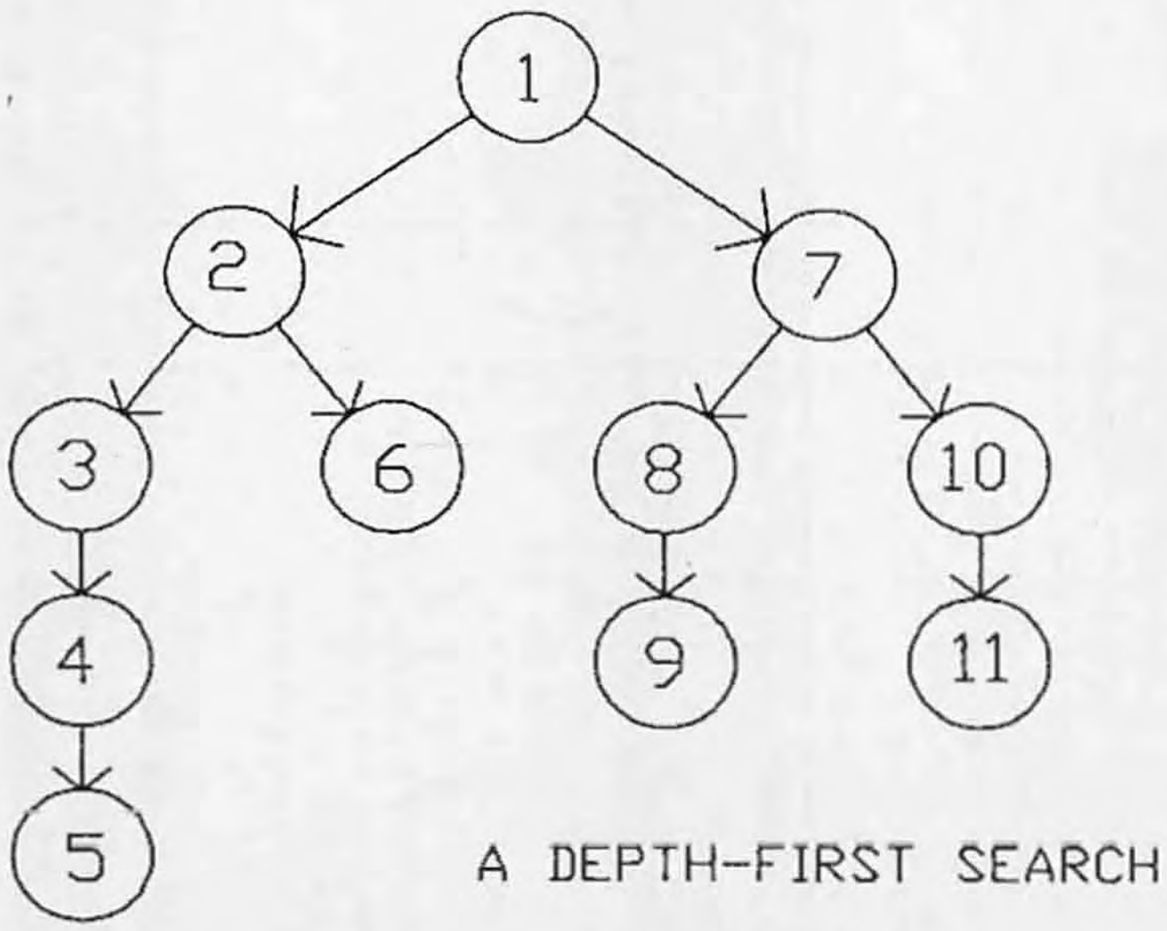


Fig. (3.2): A tree structure for Depth-First Search.

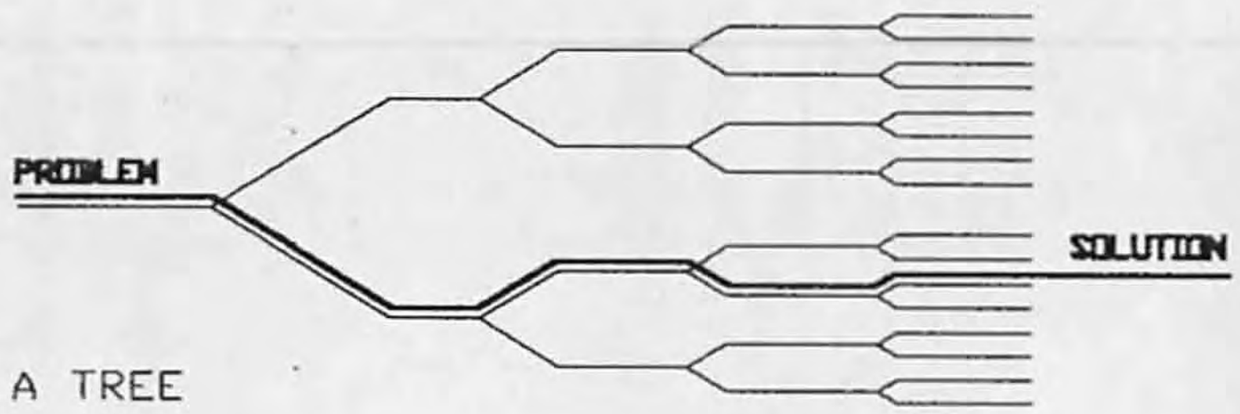


Fig. (3.3): The Decision making structures,

The tree represents sequential, linear logic, but the

the lattice allows decisions to jump from one path to another.

An in-depth investigation of such a design problem may render a certain design infeasible. This is more likely to occur when the information needed for the investigation is either open-ended (such as construction methods and different materials), or non-decomposable (as in the case of the different relations that exist between cost and aesthetics).

If the designer is using a depth first method only, then he will be oscillating endlessly between the two ideas, or will be lost in an almost infinitely large branch of the tree.

Even in some cases where open-ended or cyclic conditions do not exist, depth-first search still presents a major obstacle in achieving optimality through a sequential decision making process. As one selects a construction method, then a structural plan, and then a room layout, it is virtually impossible to ensure optimality, especially when each decision is working off from the constraints of the earlier decisions. Each new decision has to work within a narrow set of alternatives which are defined as a function of earlier decisions. To achieve optimality, earlier decisions must be made with cognizance of the decisions to follow. This requires parallel investigation which is not possible in the case of a depth-first search method.

3.3.3. Breadth-First search:

In a breadth-first search, the designer explores the problem's structure horizontally instead of vertically. He examines all nodes on one level, before considering any other node on the next lower level, as in fig. (3.4). Akin (1978) pointed out the difference between depth-first and breadth-first search as the reorganization of the examined nodes.

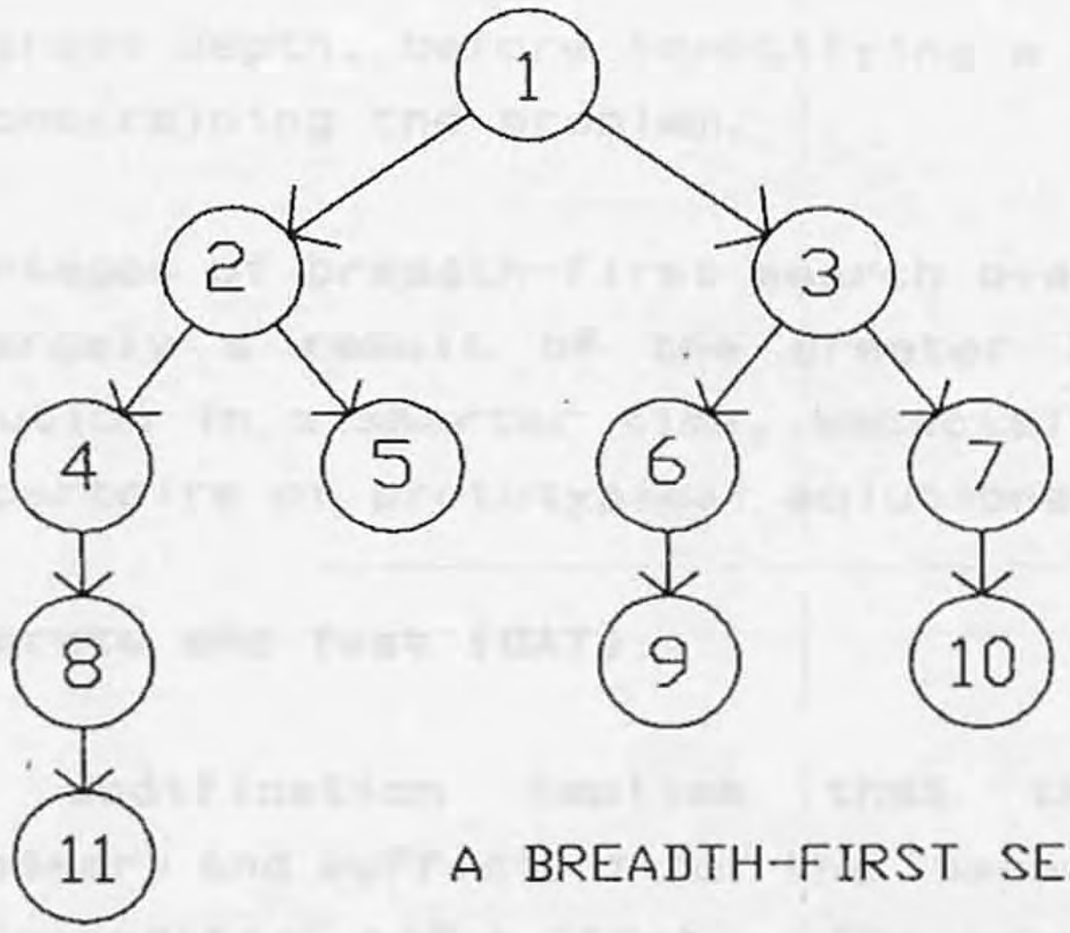


Fig. (3.4): A tree structure for a Breadth-First-Search.

Consider a design problem for which strong typological solutions exist. The speculative high-rise office building, for example, can be successfully resolved by selecting a solution type from a handful of archetypal plan organizations that respond to a few critical constraints of the problem: maximum rentable floor space and minimum investment. A depth-first search of the same problem domain would undoubtedly result in much greater effort being expended towards reinventing some of these archetypes. This requires a thorough knowledge of the performance of many building types in terms of the efficiency of circulation, structure, energy use, and construction materials. But in breadth-first search, little time is needed for each one of these categories initially. Once the suitable type is identified, all of the remaining effort is allocated to that type. This means that the designer has to study each problem component in a lateral sequence, not allowing the opportunity to search any one

component in great depth, before identifying a solution type and further constraining the problem.

The advantages of breadth-first search over depth-first search are largely a result of the greater likelihood of finding a solution in a shorter time, especially when there is a large repertoire of prototypical solutions available.

3.3.4. Generate and Test (GAT):

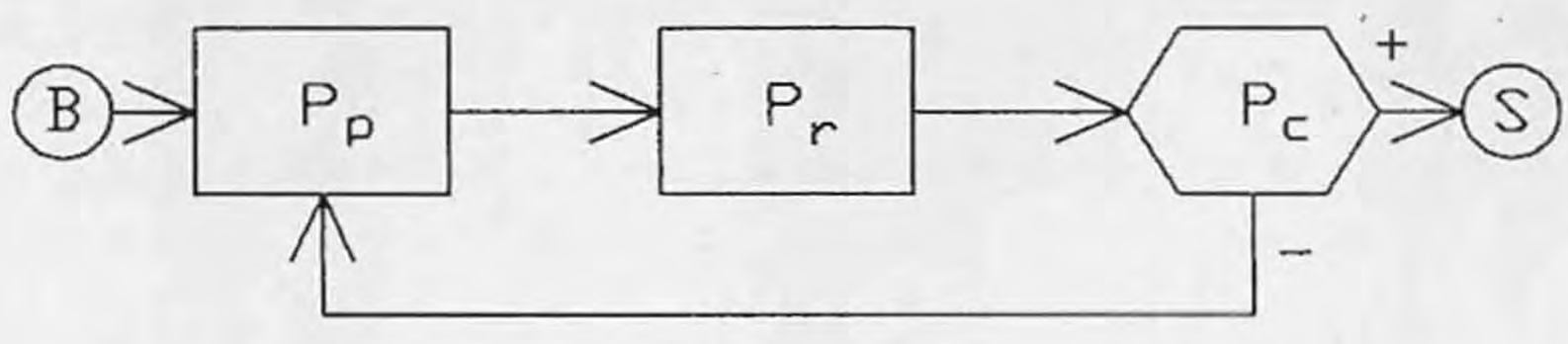
Newell's codification implies that the primitive processes necessary and sufficient for the "Generate and Test" method are a "generator" and a "test". The generate and test is quite commonly used in architecture, notably in all those cases that are cases of selection, and a number of these cases are available from which, on the basis of more or less explicit criteria, we can make a choice.

It is also possible for this process to be applied to whole buildings, as in what Broadbent (1973) calls "iconic" design (refer to chapter 2 in this thesis), a situation in which the available building types are all standardized, and the designer simply selects the building type appropriate to the task in hand, the task itself being defined in such cases by adaptation to the building type as much as the building type by adaptation to the task.

Generate and test is a good method, only when the generation and the testing of solutions are both easy and cheap, and the size of the problem space is not very large. But this is not always the case in architectural design. Akin (1978), pointed out that this is a weak method par excellence. All that must be given is a way to generate possible candidates for a solution, plus a way to test whether they are indeed solutions.

A "generator" is a process that takes information specifying a set, and produces elements of that set, one by one. It should be viewed as autonomously "pushing" elements through the system. Hence there is a flow of elements from the generate process to the test process. On the other hand, a "test" is a process that determines whether some condition or predicate is a true input, and behaves differentially as a result.

According to Newell (1970), there are two different possible outputs; satisfied (+), and unsatisfied (-). According to Akin (1986), in the case of architectural design, the representation of generated instances, especially in the form of sketches and writings, must also be an integral part of this method. Thus any combination of the projection- confirmation- representation sequence will approximate the GAT paradigm adequately for design, like in fig. (3.5). Here, the projection process is the equivalent of generating new instances; the confirmation process is the equivalent to the testing of new instances;and the representation process signifies the codification of the results of each of these processes.



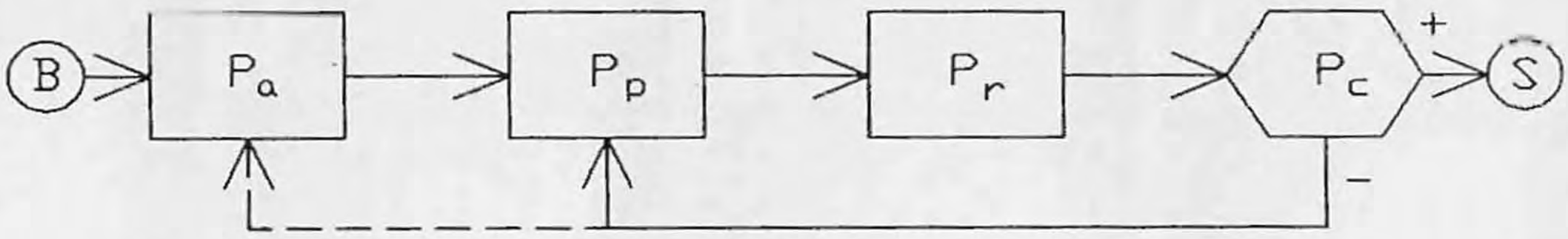
B BEGIN.
P_p PROJECTION OF INFORMATION: GENERATE A PARTIAL SOLUTION.
P_r REPRESENTATION OF INFORMATION: REPRESENT SOLUTION.
P_c CONFIRMATION OF INFORMATION: TEST IF SOLUTION MEETS GOAL.
S STOP.

Fig. (3.5): The Generate-and-Test process.

3.3.5. Hill-Climbing (HC):

Hill-climbing (HC) is a variation of the GAT method. Each newly generated solution is accepted if it represents an improvement over the best solution developed so far. That is, new candidate elements are generated by taking a step from the present position. Thus the highest element so far , plays a dual role, both as the base for generation of new elements and as the criterion for whether they should be kept (Newell, 1970).

The difference between this and the GAT method is in the test applied to evaluate the generated solutions. In the GAT method, the solution is accepted if it satisfies the goal of the designer. In the case of HC method, only the solution better than the best solution developed so far is acceptable, as in fig. (3.6).



B BEGIN.
P_a ACQUISITION OF INFORMATION: RETRIEVE BEST-SO-FAR SOLUTION.
P_p PROJECTION OF INFORMATION: GENERATE NEW SOLUTION.
P_r REPRESENTATION OF INFORMATION: RECORD SOLUTION.
P_c CONFIRMATION OF INFORMATION: COMPARE PARTIAL SOLUTION WITH BEST-SO-FAR SOLUTION.
S STOP.

Fig. (3.6): The Hill-climbing process.

The primitive operations needed in this method, to replicate the portion that corresponds to GAT, are projection, confirmation, representation, as well as acquisition of information to enable the comparison of the most recently generated solution to the best one already generated. The assumption is that, the designer generates new solutions with the help of the projection process, these solutions have to be compared against the best-so-far solution. The acquisition process retrieves the best-so-far solution from memory and enables its comparison against the current solution. If the new solution is closer to the designer's goal it is stored as the best solution so-far.

3.3.6. Heuristic Search, or Means-End-Analysis:

Architectural problem spaces are in many cases not fully determinate, and obviously involve so many alternatives. Heuristic search reduces the number of alternatives actually considered in any complex design problem. The essence of heuristic search, is to make use of information already obtained to guide the remaining steps of the problem solving process; the search process is redefined as a search for information which will limit the area of search, ultimately to the point at which generate and test (GAT) or recognition methods become practicable. All of the game-playing and theorem proving programs make use of this method, as well as many of the management science applications.

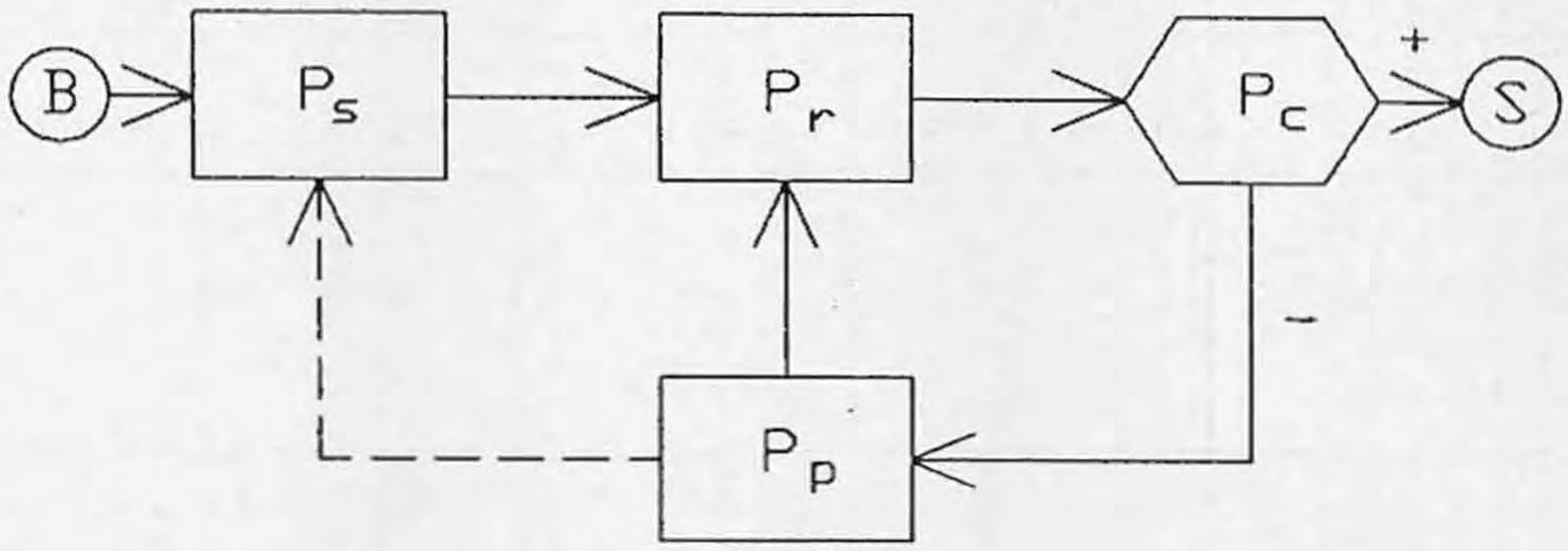
The most elementary variant of the method assumes a space of elements, the problem space, which contains one element representing the initial position (an initial state), and another representing the final or designed position (a solution state). The means-end-analysis process could be illustrated as in fig. (3.7).

3.3.7. Induction:

According to Newell (1970), induction, as a weak method for problem-solving requires i) a mapping between the given data and the predicated data, and ii) a given form for this mapping which conforms to the data. Fig. (3.8), shows the different processes in an induction process. This method is clearly a version of "hypothesis-and-test". However, the latter term is used much more generally than to designate the class of induction problems handled by this method. Furthermore, there is nothing in hypothesis-and-test which implies the use of match; it may be only generate-and-test. "Consequently, we choose to call the method simply the induction method, after the type of task it is used for" (Newell, 1970).

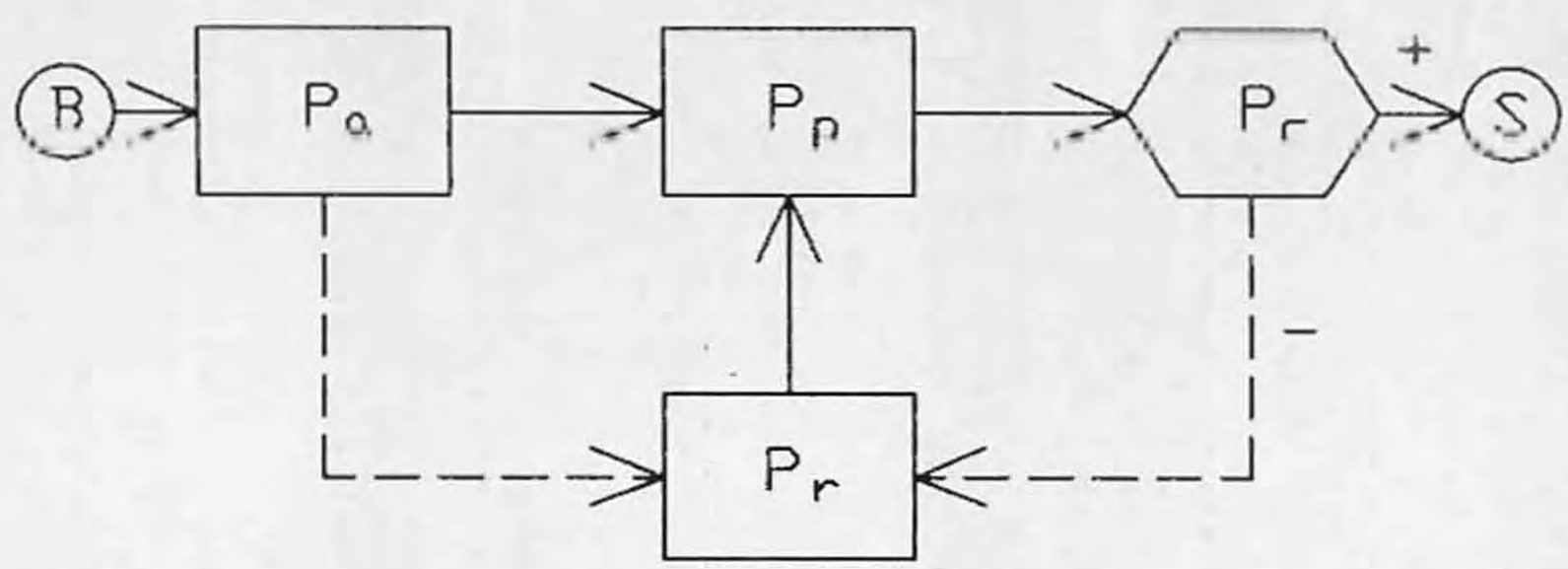
In design, there are simply too many points of departure to generate, let alone examine, all possible solutions. Rapoport (1969) discusses the notion of criticality as a function of the freedom available to the designer in making personal choices as opposed to satisfying explicit constraints.

As new building technologies become available the criticality of design constraints diminishes and the extent of freedom in choice of alternative solutions increases. Similarly, as buildings become more complex, users become more demanding. This increases the constraint and thus a greater criticality results. Often technological innovations are seen as means of relief from the increasing criticality. This notion is significant in understanding how designers search their solution space, and use style as a mechanism for balancing this formula (Akin, 1987).



B BEGIN.
P_s REGULATION OF CONTROL: SELECT HEURISTIC METHOD.
P_r REPRESENTATION OF INFORMATION: RECORD PARTIAL SOLUTION.
P_c CONFIRMATION OF INFORMATION: DOES SOLUTION MEET GOAL?
P_p PROJECTION OF INFORMATION: APPLY METHOD.
S STOP.

Fig. (3.7): The Means-End-Analysis process.



B BEGIN.
P_a ACQUISITION OF INFORMATION: SELECT RULES THAT MATCH PREDICATE.
P_p PROJECTION OF INFORMATION: PROJECT NEW INFORMATION.
P_c CONFIRMATION OF INFORMATION: DOES NEW INFORMATION MATCH DESIRED GOAL?
P_r REPRESENTATION OF INFORMATION: RECORD INFORMATION.
S STOP.

Fig. (3.8): The Induction process.

Presently there is a proliferation of architectural styles (Jencks, 1984) that underscore the need to reduce the problem space, and the number of alternatives to be considered. Most contemporary stylistic "theories" are aimed at developing systems of constraints which can be imposed on the design problem thus reducing the large number of degrees of freedom. Eclecticism, mannerism, historicism, and post-modernism are some of the more recent stylistic trends that introduce systems of constraints on design problems creating a more manageable problem space. The architect acquires an "a priory" palette of forms dictated by the style (Akin, 1987).

Methodological reduction of choices is significant in dealing with the differences of well-defined and ill-defined problems. In 1969, Simon presented "optimization", and "satisficing" as the alternative suitable for ill-defined problems. He argues that in well-defined problem domains there is either a unique or an optimal solution with respect to an objective function that is specified a priori. However, in design there are no means of finding optimal solutions, let alone objective function(s) with which to optimize. "Consequently, in design another metric of evaluation is used, that of satisficing a given set of minimum criteria. This is manifested in present architectural practice through the development of criteria using stylistic preconceptions" (Akin, 1987).

3.4. Artificial Intelligence:

From one point of view, artificial intelligence started in 1834 or shortly thereafter, when Charles Babbage suggested the possibility of having his analytical engine play chess. It was not until the 1940's, with the emergence of the digital computer, that the interest in artificial intelligence was renewed (Kalisperis, 1988).

According to the Webster's New-World dictionary of computer terms, the definition of artificial Intelligence is; the branch of computer science that studies how smart a machine can be, which involves the capability of a device to perform functions normally associated with human intelligence, such as reasoning, learning, and self-improvement.

According to Simon (1982), artificial intelligence is defined as "the discipline that is concerned with programming a computer to do clever, humanoid things-but not necessarily to do them in a humanoid way". The whole aim of artificial intelligence is to induce cleverness into the behavior of computers by writing programs that incorporate the heuristic devices that people use.

Within the last ten years, as of this writing in 1990, some results of AI research have indicated that many concepts, procedures and techniques developed in AI laboratories have great commercial value. Fig. (3.9), shows some commercial offspring of the AI research conducted during the last several years. The five most active areas are: 1) natural language, 2) robotics, 3) improved human interfaces, 4) exploratory programming, and 5) expert systems. For more details, see Harmon et al (1988).

Among all these AI activities, two are of great significance in the architectural design computability; and these are the natural language, and the expert systems.

A natural language is any language that humans speak, e.g. Arabic, English, French, or any other spoken language. Some research is trying to develop computer hardware and software that will allow computers to interact with people in a natural language, focusing on developing natural language interfaces for existing databases. Thus, designers may ask for information in a database either by typing or speaking a

request, just as they would ask an assistant for similar information. The natural language program (often called a frontend, since it stands between the user and the existing database) can convert the user's typed or spoken request into a set of database query commands to obtain the information from the database program. This will alleviate the need for the architect to learn a programming language in order to be able to retrieve a certain piece of information.

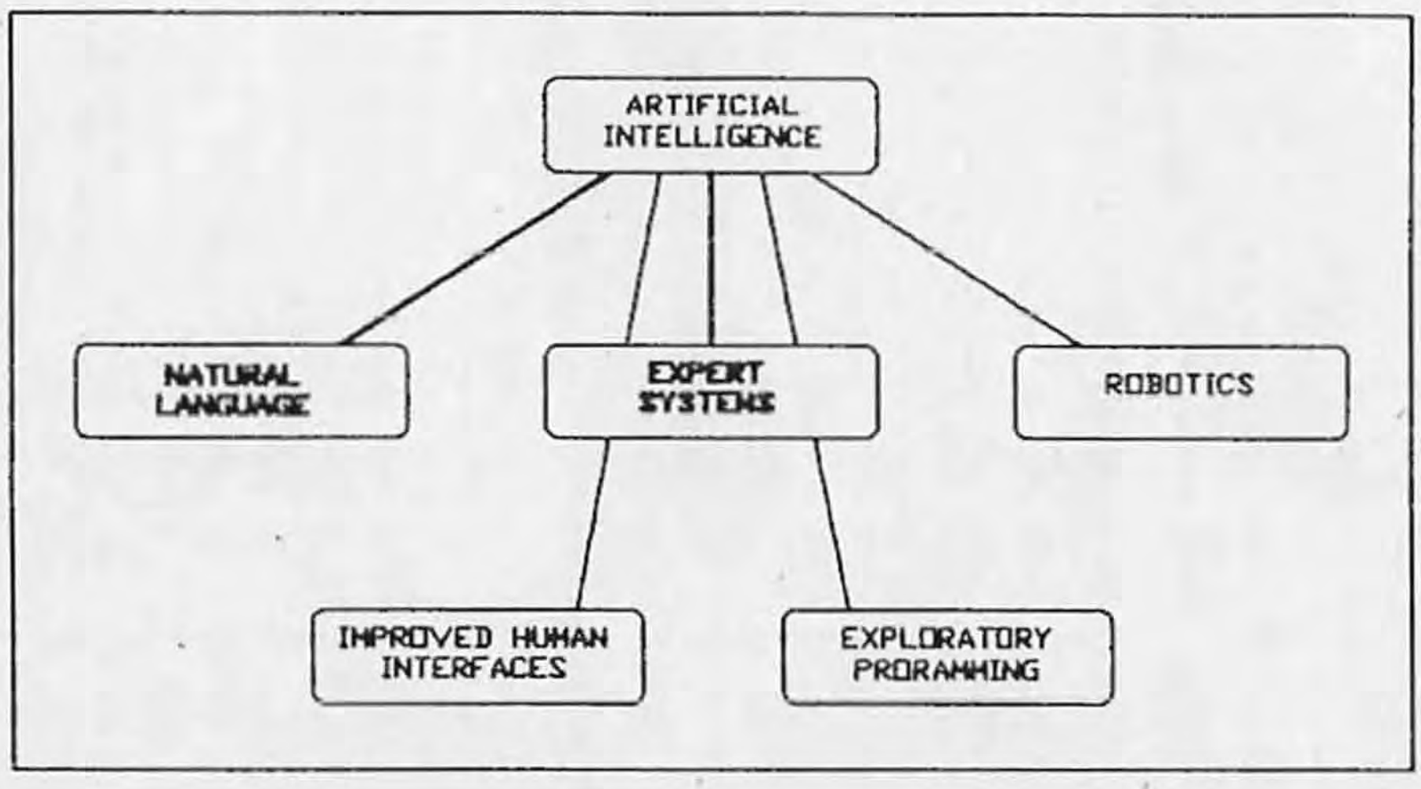


Fig. (3.9): The commercial offspring of artificial intelligence.

On the other hand, Expert Systems, are programs that utilize methods and techniques derived from AI for constructing human-machine systems with specialized problem-solving expertise. These problem-solving systems got their name to suggest that they function as effectively as human experts at their highly specialized tasks. Expert systems rely mainly on heuristics, or rules of thumb, rather than on mathematical certainty; allowing users to look at problems even when they have incomplete information, which is usually the case in architectural design problems, at least at the beginning when the whole solution space has not been discovered yet.

Regarding the computability of architectural design, the introduction of artificial intelligence as a possible method of integrating computers into the architectural design was inspired by the success of its integration into other fields such as medicine and engineering. During the middle of the 1980's, researchers started investigating the implications of artificial intelligence in architectural design computability, which will be addressed in the next part of this chapter.

Assuming that researchers are capable of achieving artificial intelligence, it can be said that it will be possible to devise better machines, or even to program existing ones so that they can outperform human intelligence in most forms of mental activity. In fact, it can be argued that present machines would be able to do this now, if people were smart enough to write the right kind of programs. The limitation then, is not in the machine, but in the man himself. Here then is the paradox. In order to make a machine which appears smarter than man, man himself must be smarter than he is at present.

Taking advantage of knowledge derived from research in computer and cognitive sciences, it is possible to create programs that simply exhibit intelligence. Such programs can and will assist human intelligence in problem solving. Similar to the effect that the industrial revolution had on the mechanization of mundane, unskilled labor, intelligent programs will replace mundane "unskilled" human thought processes. Where the machine is seen as the extension of the human, intelligent computer systems (expert systems) are the extension of the human brain (Voelker, 1985).

3.5. Expert Systems and Architecture:

As described above, expert systems are an outgrowth of (AI), a field that has for many years been devoted to the study of problem solving using heuristic, to the construction of symbolic representations of knowledge about the world, the process of communicating in natural language, and the process of learning from experience. Expertise is often defined to be that body of knowledge that is gained over many years of experience with a certain class of problem.

One of the hallmarks of an expert system is that it is constructed from the interaction of two very different people; a domain expert, or a practicing expert in some domain; and a knowledge engineer, or a specialist skilled in analyzing an expert's problem-solving processes and encoding them in a computer system. The best human expertise is the result of years, perhaps decades, of practical experience, and the best expert system is one that has profited from contact (via the knowledge engineer) with a human expert.

Foremost, among the characteristics defining an expert system, are excellent performance - accuracy, speed, and cost-effectiveness of information gathering techniques. Expert systems are also typified by a collection of other properties, many of which are taken for granted in human experts, such as: a) the ability to explain and justify answers, either by relying on a theory, or citing relevant heuristic rules, or appealing to past case histories; b) the closeness of reasoning procedures to those used by human experts (the system is not a mysterious black box using obscure mathematical formula); c) the ability to deal with uncertain or incomplete information about the current problem situation; d) the ability to summarize and point out features of the problem situation that were most important in leading to an answer, including information about which other factors might

still have an effect, if they were to become known; e) the use of verbal or symbolic encodings for knowledge, most of which is readily communicated in natural language; and f) the ability to grow gradually by adding new pieces of knowledge, usually in the context of solving an unfamiliar problem.

These qualities make the expert system more effective as a consultant, since there is some way of backing up answers and of building confidence in the system's abilities. Also included are the possibilities of improving the system by conversational means, and of using the system as a tutor or trainer.

Inherent complexity of a problem area and scarcity of good human experts are prime motivating factors for building expert systems. Building an expert system often helps to systematize a body of knowledge, so that it can be widely dispersed. An expert system is often a good means for pooling the expertise of a number of specialists, to produce a system that is more effective than any of them working alone. Fully automated systems can use the capabilities of an expert system to avoid the need for human intervention in many of the routine day-to-day failures and emergencies.

The kinds of problems that are most amenable to the expert system type of approach, are those requiring knowledge intensive problem solving, i.e., where years of accumulated experience produce good human performance. Such domains have complex fact structures, with large volumes of specific items of information, organized in particular ways.

The advantages of an expert system are significant enough to justify major efforts to build them: a) decisions can be obtained more reliably and consistently, b) explanation of the final answers is an important side product, c) a problem area can be standardized and formalized through the process of

building an expert system for it, d) a consultation mode on difficult cases could be obtained, where humans may overlook obscure factors, e) an expert system can often serve as an example of good strategy in approaching a problem, which might be useful in training situations, f) expert systems can be more easily expandable than conventional software, so that they can gradually be improved as their problem domain evolves.

Architectural Expert Systems are expected to solve many of the problems of the first generation tools for abstraction and evaluation, i.e., supply computer aided abstraction and evaluation tools with reasoning capabilities.

Although the concept of expert systems has matured, and produced very impressive results in areas such as medicine, geology and engineering; its applications in architecture so far are limited. One reason of this limitation is the graphic character of traditional architectural representation and the fact that this abstraction hides most of the meaning of the drawing, not from the designer, but from the computer. The languages and shells used to build expert systems that exist today were not developed for architectural applications (Kalisperis, 1988). Rather, architectural researchers have selected existing systems for their implementations.

Successful implementations, although limited in scope, were developed for spatial design (Flemming 1986), abstraction (Schmitt, 1986), architectural planning (Schmitt 1988b), participatory design (Kalay and Swerdloff 1987, Kalay et al 1985), composition (Gero and Sambura 1985), geometric reasoning using knowledge based systems (Woodbury 1988), and passive solar energy (Rosenmann et al 1988, Rosenmann and Gero 1989).

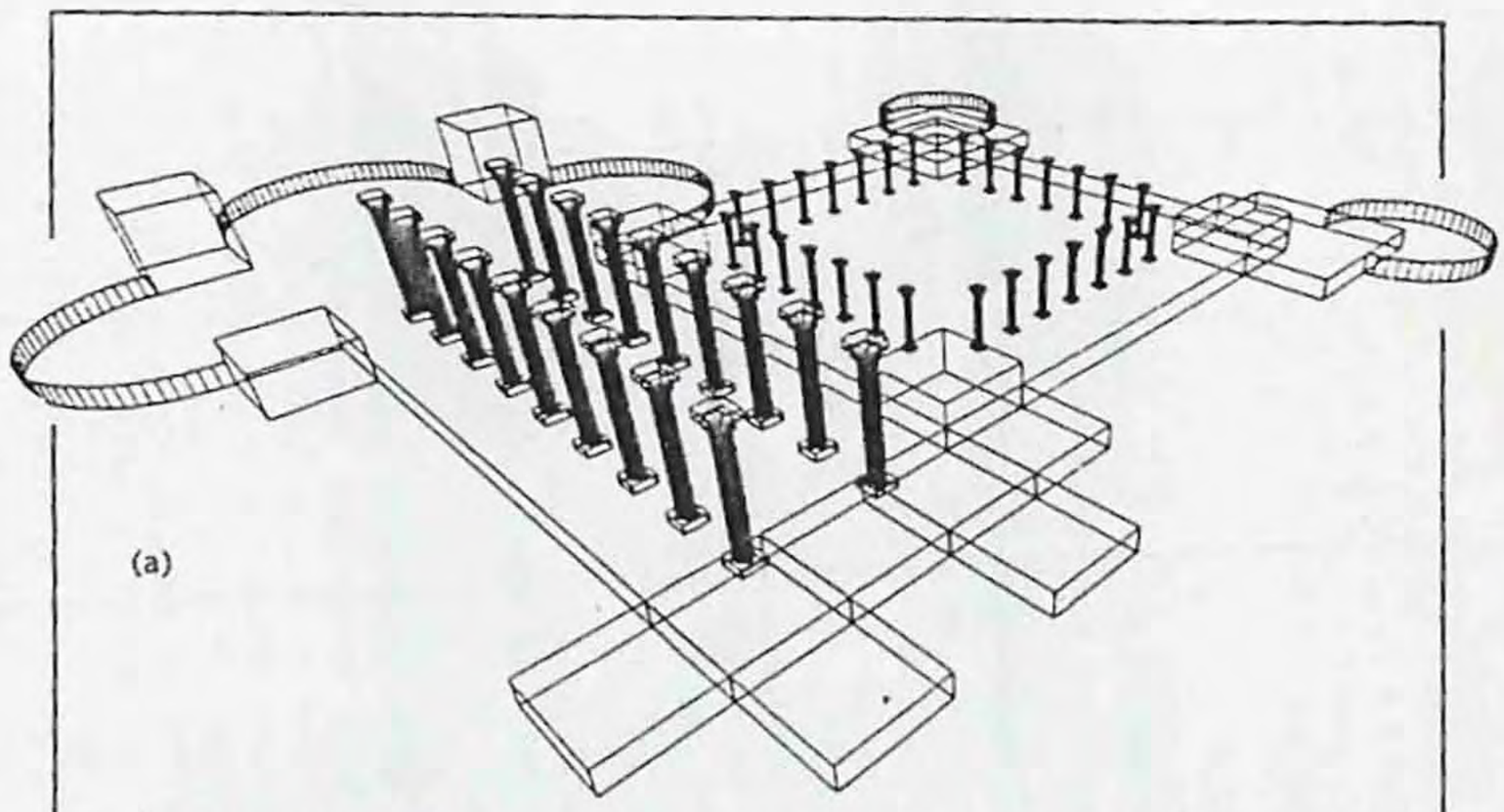
Some examples of expert systems in architecture follows:

Example 1: Expert Systems in Abstraction;
The Medieval Architectural Consultant:

This program focuses on medieval religious French architecture. The reasons for this are: the encountered building types can easily be classified and abstracted, and the interdependency of qualitative-political and quantitative-technological factors is evident and can be formalized (Schmitt, 1987).

The Medieval Architectural Consultant combines several programs: a graphics program provides the expert system with a graphical output and a graphical knowledge acquisition module, while drafting and database management programs provide the system with functional utilities. The system deals with the relationship between the construction of religious buildings in medieval France and historical conditions. It features the generation and graphic display of various possible religious building types, their graphic insertion into a map of France, and the inference of a historical statement from the distribution type and number of the buildings in a certain area of France, as in fig. (3.10).

The system is implemented at a high level of control abstraction, and can therefore be easily changed to represent other aspects of design, history, or evaluation. All historical and factual knowledge is represented in the form of rules. The program is a hybrid implementation with the expert system controlling procedural support programs. The users begin by defining the kinds of buildings (e.g., Cathedrals, or Non-Cathedrals), that will later be used for the history session. Selections include the time period, building type, building location, religious order, and the functional situation of the builder.



(a)

```
;_____________________________________________________
(p get_answer3
    (question_3)
    (religion catholic)
-->
    (remove 1)
    (write (crlf) Building)
    (write (crlf) |1   Cathedral|)
    (write (crlf) |2   Non Cathedral|)
    (write (crlf) (crlf) |Your choice:|)
    (bind <x> (accept))
    (write (crlf) |_____________________|)
    (make answer3 <x>))
;_____________________________________________________
(p bad_answer3
    (answer3 { <>1 <>2 })
-->
    (remove 1)
    (make question_3))
;_____________________________________________________
(p answer3_is_1
    (answer3 1)
    (building <a> <b> <c>)
-->
    (remove 1)
    (modify 2 ^kind cathed
```

(b)

(c)

Fig. (3.10): Expert Systems in Abstraction.

The Medieval Architectural Consultant

Example 2: Expert Systems in Design Generation,
The "Mies Van Der Rohe" Generator:

Two of the well known projects of Mies Van Der Rohe; the Barcelona Pavilion, and the court house projects are examples of a particular type of architectural language; panel architecture. The notion of language is introduced to explore the analogy between architectural and natural language. Nouns, verbs, and grammar of natural language are related to symbols, relationships between symbols, and design rules in architectural language, respectively. The knowledge base consists of a set of design elements (slabs, roofs, and panels), as well as a number of possible relations between these elements. The design rules determine the order in which these elements will be combined (the control structure).

According to Schmitt (1987b), a typical session with the Mies generator would begin with the user defining the slab of the building, and then graphically defining the different kinds of panels or selecting them from a database. In a second step, possible relations between panels must be established graphically, or again selected from a database. In the third step, the inference engine is initiated and will place the panels according to the rules established in the control structure.

If panels cannot be placed because they would intersect with another panel, the length of the panel will automatically be reduced, and an other placement attempt would be made. At any given stage in the generation process, the user may interrupt the process for manual editing, and if desired, the automatic and the manual generation could be mixed.
Fig. (3.11) shows the result of a typical working session with the Mies Generator.

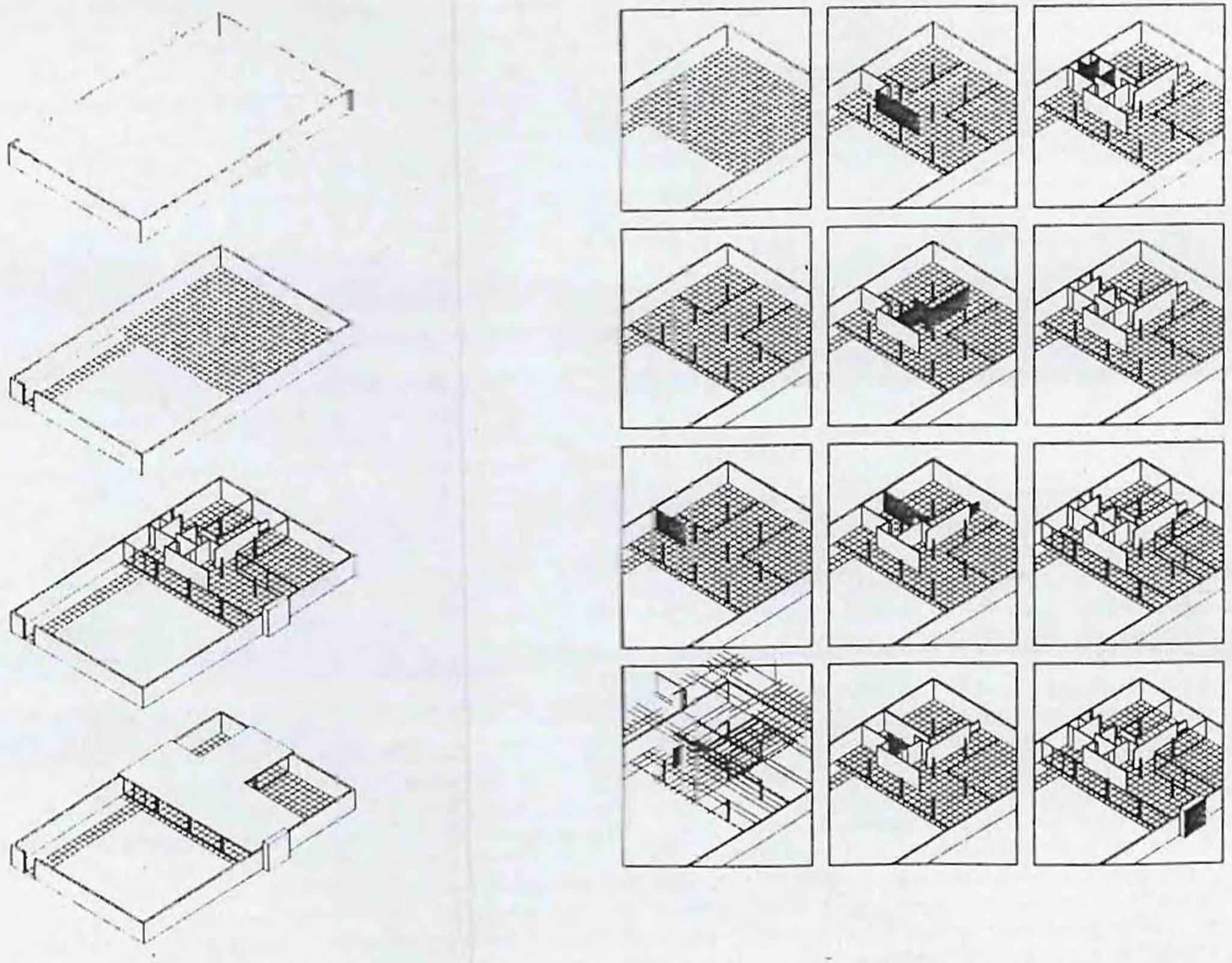


Fig. (3.11): A residential design proposed by the Mies Generator.

Other examples of architectural expert systems could be described as prototype refinement on a global level, and of simulation and optimization on a local level (Schmitt, 1988). Prototype refinement means that a typical prototype for a particular building type is chosen at the beginning of the design process which is subsequently changed and refined (Gero and Maher, 1987). Simulation includes operations on an abstract model of the design to predict consequences of design decisions (Schmitt, 1987). While optimization involves finding optimal solutions for one or more pre-defined design parameters (Radford and Gero, 1986).

Although some limited applications exist in architectural expert systems, they have not been introduced in the design process yet. Design problems require extensive knowledge based on the definition of the problem and the problem space. Knowledge applicable to architectural problems is very much problem definition dependent.

The problem with architectural expert systems exists primarily in the extraction of knowledge or information and the thought process of the expert that lies at the core of the expert system. The main problem is that the expert is often unable to describe how he reaches a specific conclusion; he cannot adequately describe or define his reasoning and search or thought processes, or even describe his knowledge systematically (Doyle, 1984).

"Experience has taught us that much of their (experts) knowledge is private to the experts, not because they are unwilling to share publicly how they perform, but because they are unable. They know more than they are aware of knowing". (Doyle, 1984).

Sometimes the knowledge can be externalized by the "careful, painstaking analysis of a second party, or

sometimes by the experts themselves operating in the context of a large number of highly specific problems" (Feigenbaum, 1961).

Currently, the only means of collecting knowledge in architecture is protocol analysis, which is "the recorded behavior of the problem solver or the designer in the form of sketches, notes, audio or video recordings" (Akin, 1986). Though protocol analysis produces more "rich and comprehensive data" than conventional methods, it is nevertheless faced with the previously discussed problem of externalizing knowledge and thought or search processes, and is criticized as being inappropriate in experimental work. With both, protocol and conventional methods of analysis, there still remains room for human error. Indeed the human thought process can be readily misunderstood, or misrepresented because we lack a fundamental understanding of human thought.

Knowledge-based systems take years to develop and can thus become very costly. This is particularly true in a situation where the problem is constantly changing, and the nature of each problem is in fluctuation, like in architectural design. In this case, the knowledge base established for one problem often does not apply to the next, and the rules must both be altered and adjusted for and within each individual problem.

The architectural design process itself poses unique problems for computer applications. Simon (1973) in his article: The Structure of Ill Structured Problems, pointed out that the nature of the field, the lack of common architectural language, and the need to gain improved understanding of problem-solving thought processes in conjunction with the many facets of the problem space have thus far inhibited development of expert systems in architectural design.

So far, the study has addressed the issues of different generation design methods, problem solving, problem finding and search in architecture, artificial intelligence, and expert systems in architecture.

In the following chapter, different computer applications in architecture will be addressed. These will include; different areas of architectural computing, as well as, different computer-aided design tools.

REFERENCES FOR CHAPTER 3:

Abell, W.:

The Collective Dream in Art. New York, Shocken, 1957.

Akin, O.:

"How Do Architects Design?" In J. Lactombe's ed., Artificial Intelligence and Pattern Recognition in Computer-Aided Design. Amsterdam: North-Holland, 1987.

"An Exploration of the Design Process." In Nigel Cross's ed., Developments in Design Methodology, John Wiley & Sons. New York, 1984.

"A Formalism for Problem Restructuring and Resolution in Design." In Planning and Design. Vol. 13, 1986.

Psychology of Architectural Design. Pion Limited, London, 1987.

Archea, J.:

Architecture's Unique Position Among the Disciplines, Puzzle-Making vs. Problem-Solving. In The Design Process, pp. 20-22, September, 1985.

"Puzzle Making: What Architects Do When No One Is Looking." In Yehuda Kalay's ed., Computability of Design, John Wiley & Sons, New York, 1987.

Banham, R.:

"Where Are You Universal Man Now That We Need You...?" In RIBA Journal, No. 7, p.295, 1965.

Barr, A, a,d Feigenbaum, E.:
The Handbook of Artificial Intelligence. William Kaufmann, Inc., Los Altos, CA, 1981.

Broadbent, G.:
Design In Architecture. John Wiley & Sons, Ltd., New York, 1973.

Cross, N.:
Developments in Design Methodology. John Wiley & Sons, Ltd., New York, 1984.

Doyle, J.:
"Expert Systems Without Computers or Theory and Trust in Artificial Intelligence." In The AI Magazine, pp. 59-63, Summer, 1984.

Feigenbaum, E. A. :
"The Simulation of Verbal Learning." In Proceedings of the Western Joint Computer Conference, pp. 121-129, New York: ACM, 1961.

Flemming, U.; Rychener, M.; Coyne, R.; and Glavin, T.:
"A Generative Expert System for the Design of Building Layouts." In Z. Sriram, and A. Adely's ed., Applications of Artificial Intelligence to Engineering Problems. Berlin: Springer-Verlag, 1986.

Gero, J. S.:
Design Optimization. New York: Academic Press, 1985.

Gero, J., and Maher, M.:
"A Future Role of Knowledge-Based Systems in the Design Process." In CAAD Futures, '87. Eindhoven

University of Technology, The Netherlands, May, 1987.

Gero, J., and Sambura, A.:
"A Framework for a Computer Integrated Design Environment - CIDE." In K. Bo and F. M. Lillehagen's ed., CAD Systems Framework. Amsterdam, North-Holland Publishing, 1985.

Haider, J.:
A Conceptual Framework for Communication-Instruction in Architectural Design. Ph.D. Dissertation, The Architectural Department, The Pennsylvania State University, 1986.

Harmon, P., Maus, R., and Morrissey, W.:
Expert Systems Tools and Applications. John Wiley & Sons, Inc. New York, 1988.

Heath, T.:
Method in Architecture. John Wiley & Sons, Inc. New York, 1984.

Jencks, C.:
The Language of Post-Modern Architecture. New York: Rizzoli, 1984.

Kalay, Y.:
Computability of Design. A Wiley Interscience Publication. John Wiley & Sons, New York, 1987.

Kalay, Y., and Swerdloff, L.:
"A Partnership Approach to Computer-Aided Design". In Yehuda Kalay's ed. Computability of Design. A Wiley-Interscience Publication. John Wiley & Sons, New York, 1987.

Kalay, Y., Harfmann, A., and Swerdloff, L.:
"ALEX: A Knowledge-Based Architectural Design System." In Proceedings of ACADIA '85, 1985.

Kalisperis, L. N.:
A Conceptual Framework for Computing in Architectural Design. A Ph.D. Dissertation. The Architectural Department. The Pennsylvania State University, 1988.

Newell, A.:
"Heuristic Programming: Ill-Structured Problems." In J. A. Arnofsky's ed., Progress in Operations Research. Vol. 3. John Wiley & Sons, Ltd., New York, 1970.

Newell, A., and Simon, H.:
Human Problem Solving. Englewood Cliffs, New Jersey: Prentice-Hall, 1972.

Poincare, H.:
Science and Method. Translated by F. Maitland, New York: Dover Publications, 1952.

Polya, G.:
How To Solve It. Princeton, New Jersey: Princeton University Press, 1945.

Radford, A., and Gero, J.:
Design By Optimization in Architecture and Building.
Van Nostrand Reinhold Company, New York, 1986.

Rapoport, A.:
House, Form and Culture. Englewood Cliffs, New Jersey: Princeton-Hall, 1969.

Rosenmann, M., Gero, J., Tolhurst, S., Postmus, A., & Radford, A.:

"SOLAREXPERT, An Expert System for Passive Solar Energy Design in Housing; Stage 1." In S.V. Szokolay's ed., People and Technology - Sun, Climate and Building. University of Queensland, Brisbane, pp. 121-128, 1988.

Rosenmann, M., and Gero, J.:

"SOLAREXPERT, An Expert System for Evaluating Passive Solar Energy Designs." In B. H. V. Topping's ed., Artificial Intelligence Techniques and Applications for Civil and Structural Engineers. Civil-Comp Press, Edinburgh, pp. 131-139, 1989.

Schmitt, G.:

"Expert Systems in Design Abstraction and Evaluation." In Yehuda Kalay's ed., The Computability of Design. John Wiley & Sons, New York, 1987.

"Expert Systems and Interactive Fractal Generators in Design and Evaluation." In CAAD Futures '87. Proceedings of the Second International Conference on CAAD Futures, edited by T. Maver, and H. Wagter. Eindhoven, The Netherlands, 20-22 May, 1987.

Microcomputer Aided Design, for Architects and Designers. A Wiley Inter-science Publication. John Wiley & Sons, New York, 1988.

"ARCHPLAN: An Architectural Planning Front End to Engineering Design Expert Systems." In Michael D. Rychener's ed. Expert Systems for Engineering Design. The Academic Press, Inc. CA. USA. 1988b.

Simon, H.:
The Sciences of the Artificial. Cambridge, Massachusetts: MIT Press, 1969.

Simon, H.:
"Style in Design." In Proceedings of the Environmental Design Research Association Conference. Pittsburgh: Carnegie Mellon University, Department of Architecture, 1970.

"What Computers Mean for Man and Society." In H. Simon's ed., Models of Bounded Rationality. Cambridge, Massachusetts: MIT Press, 1982.

"The Structure of Ill-Structured Problems." In N. Cross's ed., Developments in Design Methodology. John Wiley & Sons, Ltd., New York, 1984.

Voelker, W.:
"Psychological Aspects of the Design Process." In Crit 11, pp. 6-11, 1985.

Wade, J.:
Architecture, Problems, and Purposes. John wiley & Sons, Ltd., New York, 1977.

Woodbury, R.:
Representation and Manipulation of Geometric Information in a Knowledge Based Expert System." Ph.D. Dissertation, Carnegie Mellon University, 1988.

CHAPTER 4

COMPUTER APPLICATIONS IN ARCHITECTURE

CHAPTER 4

4. Computer Applications In Architecture.

4.1. Different Areas of Architectural Computing.

4.1.1. Design Applications.

4.1.1.1. Conceptual Modeling.

4.1.1.2. Integrated Design Systems.

4.1.1.3. Space Planning.

4.1.1.4. Computer Mapping.

4.1.1.5. Site Planning and Land Analysis.

4.1.2. Technical Applications.

4.1.2.1. Structural Analysis.

4.1.2.2. Financial Analysis, and Cost Estimating Programs.

4.1.2.3. Thermal Performance, and Mechanical Analysis.

4.1.2.4. Lighting and Acoustics.

4.1.3. Production Applications.

4.1.3.1. Working Drawings.

4.1.3.2. Specifications.

4.1.4. Business and Management Applications.

4.1.4.1. Office Accounting.

4.1.4.2. Comprehensive Financial Management.

4.1.4.3. Project and Personal Scheduling, Construction Management.

4.2. Different Computer-Aided Design Tools.

4.2.1. Representation of Computer-Aided Drafting.

4.2.2. Simulation.

4.2.3. Generation.

4.2.4. Optimization.

CHAPTER 4

COMPUTER APPLICATIONS IN ARCHITECTURE

One of the original purposes of the computer was to relieve humans from tedious and menial tasks. In February, 1946, the first electronic digital computer, called ENIAC, was developed at the University of Pennsylvania. It was built by John Mauchly, Jr., Presper Eckert, and John von Neumann. It was to calculate various mathematical functions for the purposes of military warfare in World War II, which up till then was being done manually, or by electro-mechanical means. (Radford, and Stevens, 1987), and (Gamboa, 1987).

Computer programs which exist today, aid, automate and accelerate a wide variety of tasks in architectural, engineering and design offices. Many of these applications have been available for decades. Others are relatively new. Most are continually being refined and improved as computer technology improves. An increase in computer power and speed, enhanced high level languages, and the knowledge gained from years of experience have all contributed to the creation of increasingly useful and efficient applications.

4.1. Different Areas of Architectural Computing:

As a useful tool to organize this chapter, the major areas of computer applications in architecture have been grouped into four categories. The first is titled design applications and encompasses a broad range of programs that aid the architect during the design development stage of a project. The second group deals with technical applications, including engineering and other mathematical type analysis which help in evaluating and elaborating a certain design concept. Production aspects of the firm are considered next, dealing mainly with detailed design, specifications and

working drawings. Finally, the business and management section deals with programs that are not for use on client projects, but instead are for in-house use to help in organizing and running a more efficient office.

4.1.1. Design Applications:

Unlike what the title of this section implies, computers do not think and design the way a designer designs. Rather the techniques described here are all tools used to produce alternatives, give suggestions, and provide as much information as possible to support the designer/user to make a certain decision regarding different alternative solutions. It is important to point out here that, building designs will never be fully automated, produced in mass by a room full of computers. The computer will never replace the architect or the engineer; it is only a tool to enhance the designer's knowledge, and allow him to design better buildings than he would without the help of the computer.

4.1.1.1. Conceptual Modeling:

In architectural design, the most menial of jobs is the drafting task. Computer drafting was introduced to the profession to alleviate as well as to expedite that particular aspect of the design process. Depending on the capabilities of the computer system, electronic drafting could mean the production of floor plans, sections and elevations. It could also mean generating three dimensional (3-D) views of the building directly from the two dimensional (2-D) orthographic drawing. These 3-D views are also called geometric modeling and could be used as a base to manual rendering or as a means of testing and evaluating the massing of structures.

Available computer application programs permit the architect to create a 3-D simplified block model of a building

within a graphic computer system early in the preliminary design phase. Perspectives , isometrics and orthogonal views for any part of the building can be shown on the screen, and can be accessed from any angle. If the area surrounding the building (other buildings, streets and landscaping) is input, it is possible to assess a particular design's impact on the visual environment as well, by showing sight lines down various streets. This technique is especially helpful and valuable in the early preliminary design stage, to help the architect evaluate alternative designs.

These systems can also be used as an interesting marketing tool. Graphic output can be produced showing the different views from any window in any floor inside a building, looking in any direction away from the building. This has a great appeal to clients who will be working or residing in a building, and wish to choose the view from their window before the building is even built.

Another useful and interesting feature, is the use of slide-shows or "walk-throughs". This capability enables the architect to give a complete representation about his building even before it is built. He can use a "slide-show" to demonstrate different views inside or outside the building, or he can even use the "walk-through" capability to show the client how his building is going to look like from the inside, as if the client or the viewer was moving towards a certain direction. As he moves around, he can explore more views of the interior. Naturally, with animation capabilities, the "walk-through" would be a smooth change of scenes rather than a "slide show" effect. Figure (4.1) illustrates the movement through a 3-dimensional path to demonstrate different views.

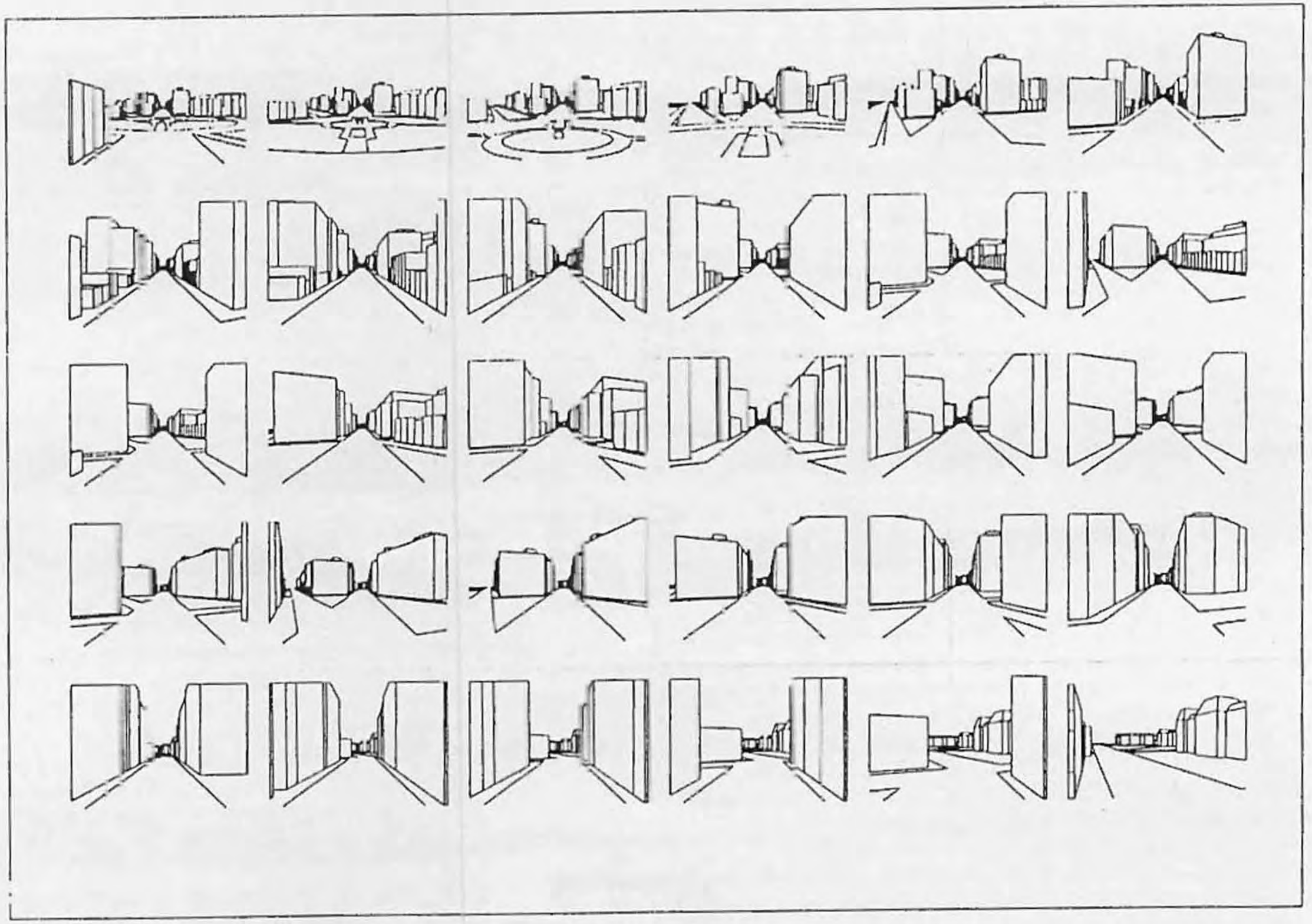


Fig. (4.1): A perspective tour

through the streets of Washington D.C.

In addition, the computation of the shadow patterns produced by buildings can easily be incorporated into conceptual modeling routines. This calculation is not complex, but involves a great deal of number manipulation.

The computer then is considered to be a useful tool in this situation. It is very simple to find the shadow that a building will cast on the ground, as well as onto surrounding buildings. A more complex calculation is needed to determine the location of a shadow cast into a room through a window, because of obstructions caused by the wall surface, sunshading devices, and so on. In SOM's architectural firm, the computer shadow projection from "One Magnificent Mile", in figure (4.2); a project in Chicago, Illinois, was used to convince zoning officials that the building would not prevent the sun from reaching the nearby beach. The program that generated the shadows was written in one day for the zoning presentation (Iyengar, 1985).

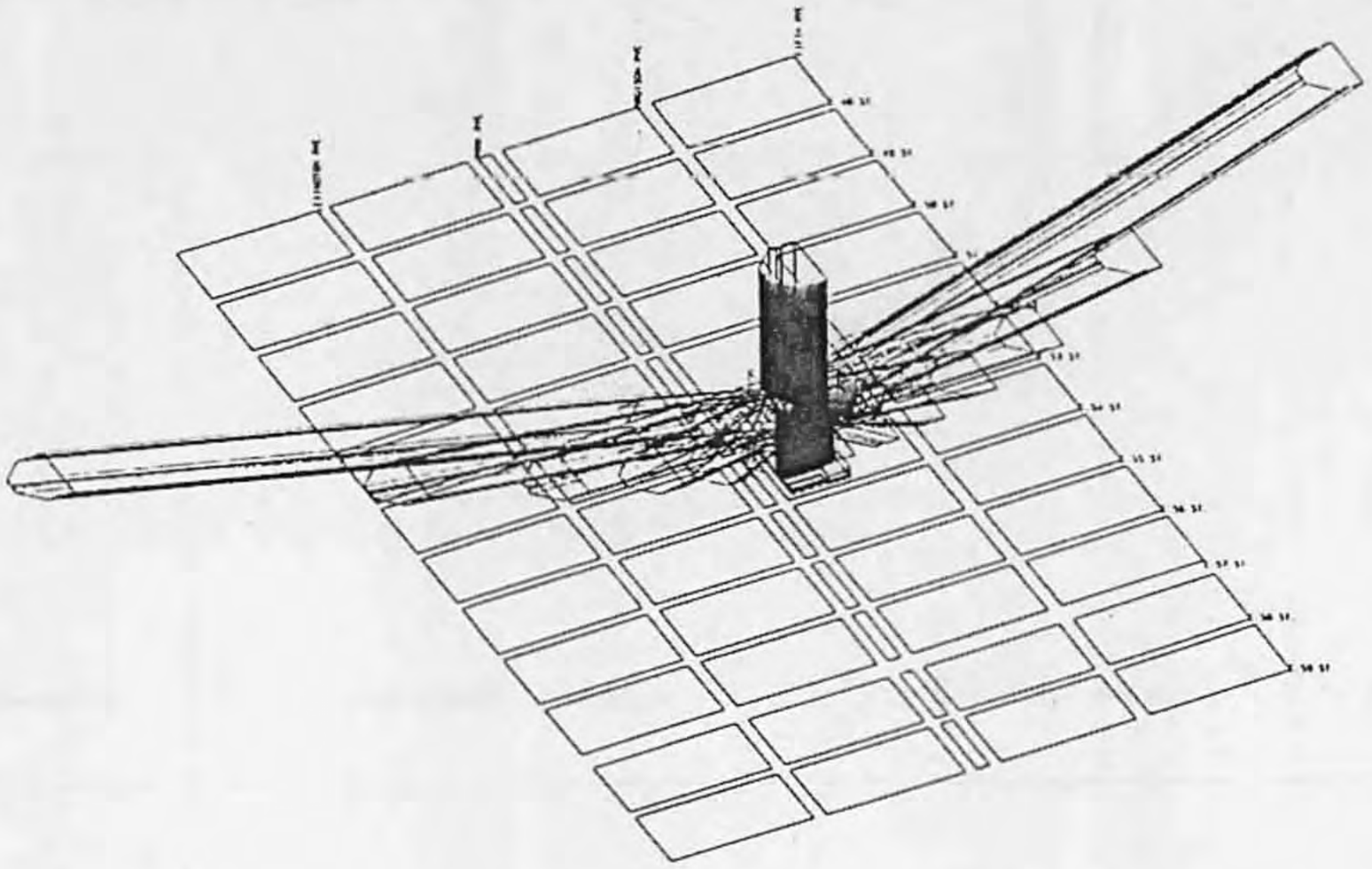
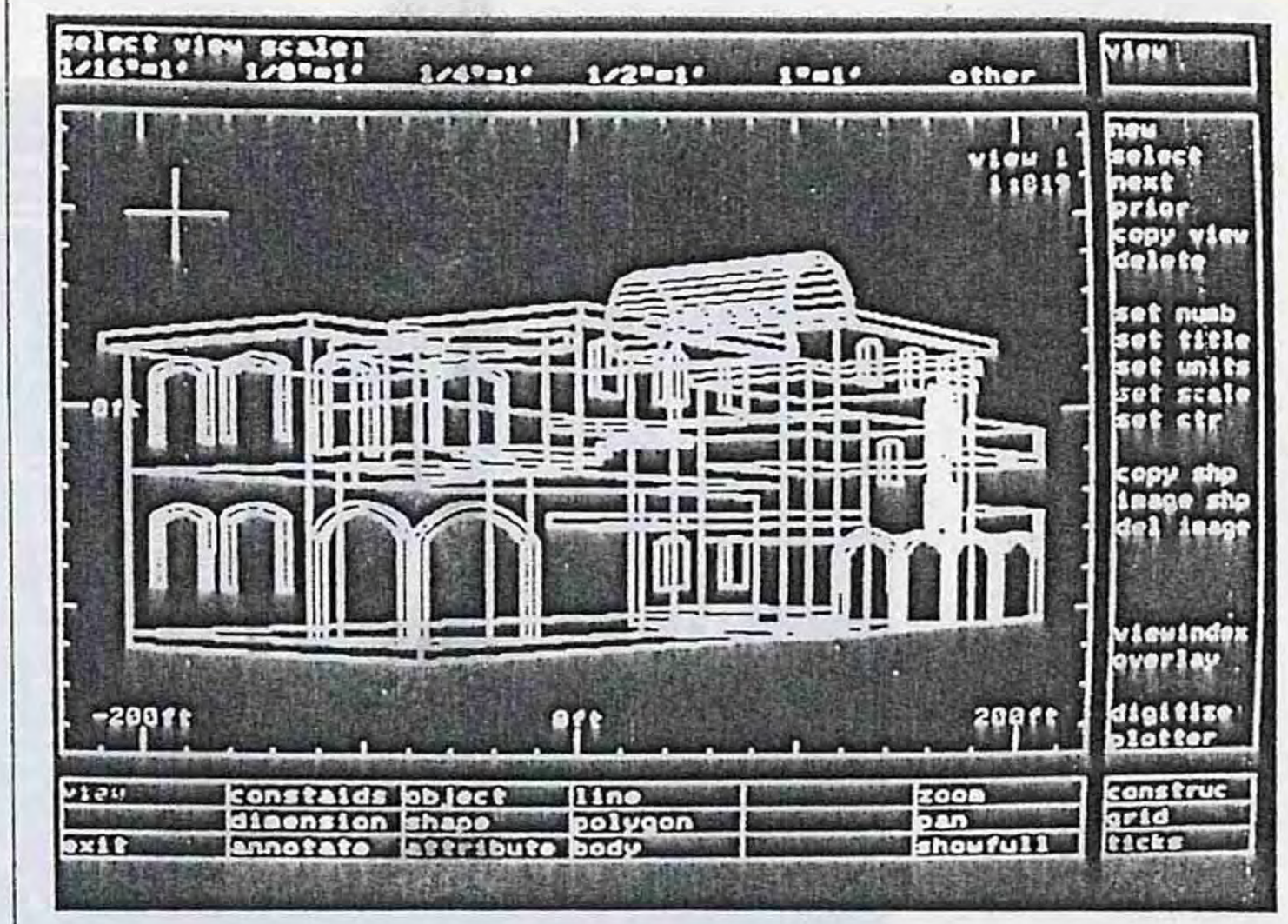


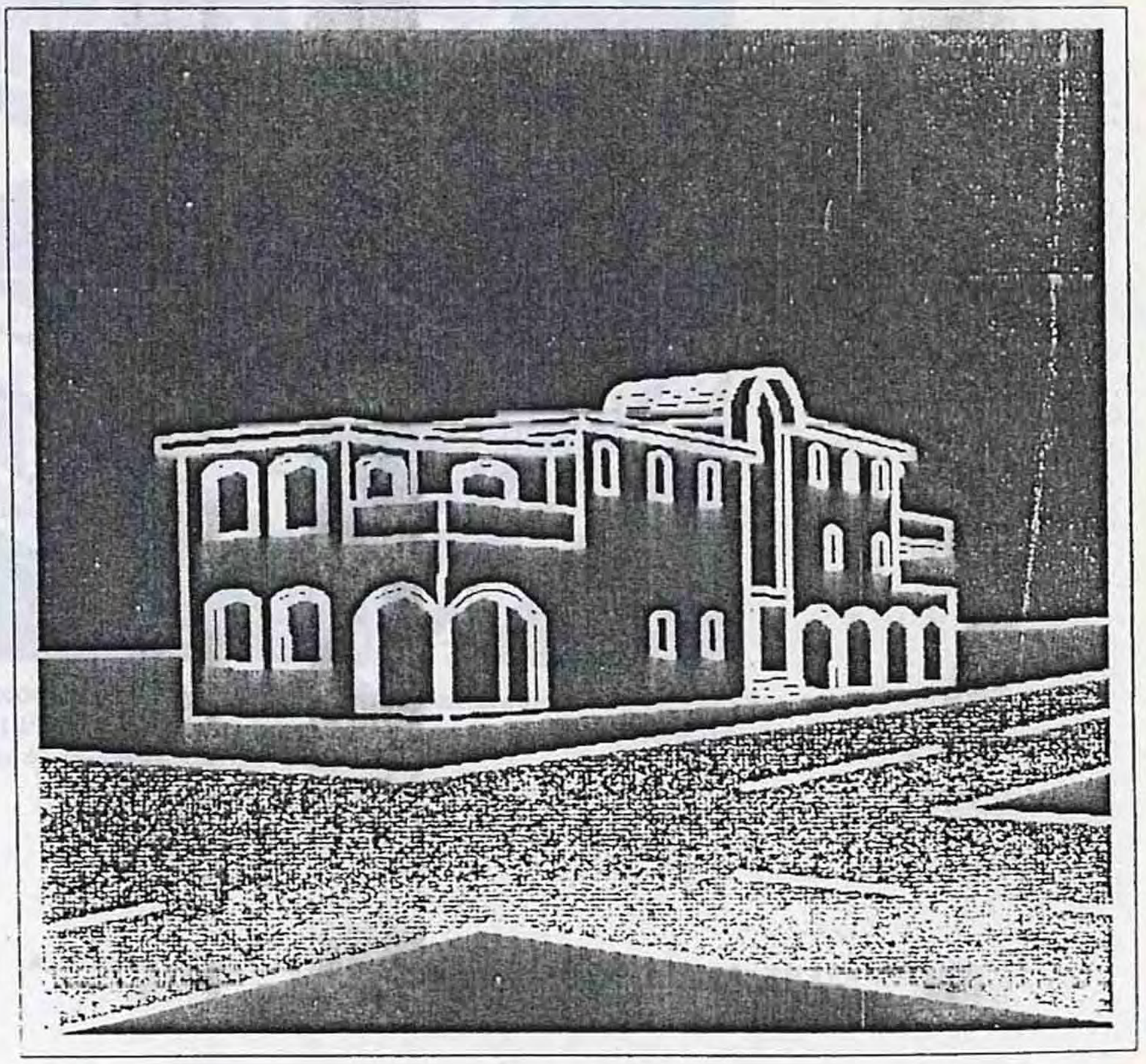
Fig. (4.2.): The shadow projection from One Magnificent Mile.

Some modeling programs, as contrasted with most drafting systems, are developed on the concept of solid modeling. This means that instead of storing a wall or a column as a series of lines (as most drafting systems do), the computer recognizes items as being "solids" that cannot be arbitrarily penetrated or intersected. This concept is also useful in the design development and working drawing stages, to check for interferences such as pipes passing through beams, etc. Ray Clark from SOM firm pointed out that they are not only working on software that checks for interferences among the various systems in a building, but also automatically corrects the conflict, by resizing a duct or moving a plumbing line, for example (Iyengar, 1985).

Color also has become an important feature in computer graphics. The use of solid areas of color to fill in a computer drawing of a building is extremely useful to help visualize a design. A modeling system with color capabilities can provide the architect with invaluable visual information early in the design process. Input devices such as the stylus and the digitizing tablet are utilized in a pencil and paper fashion (Burden, 1985).

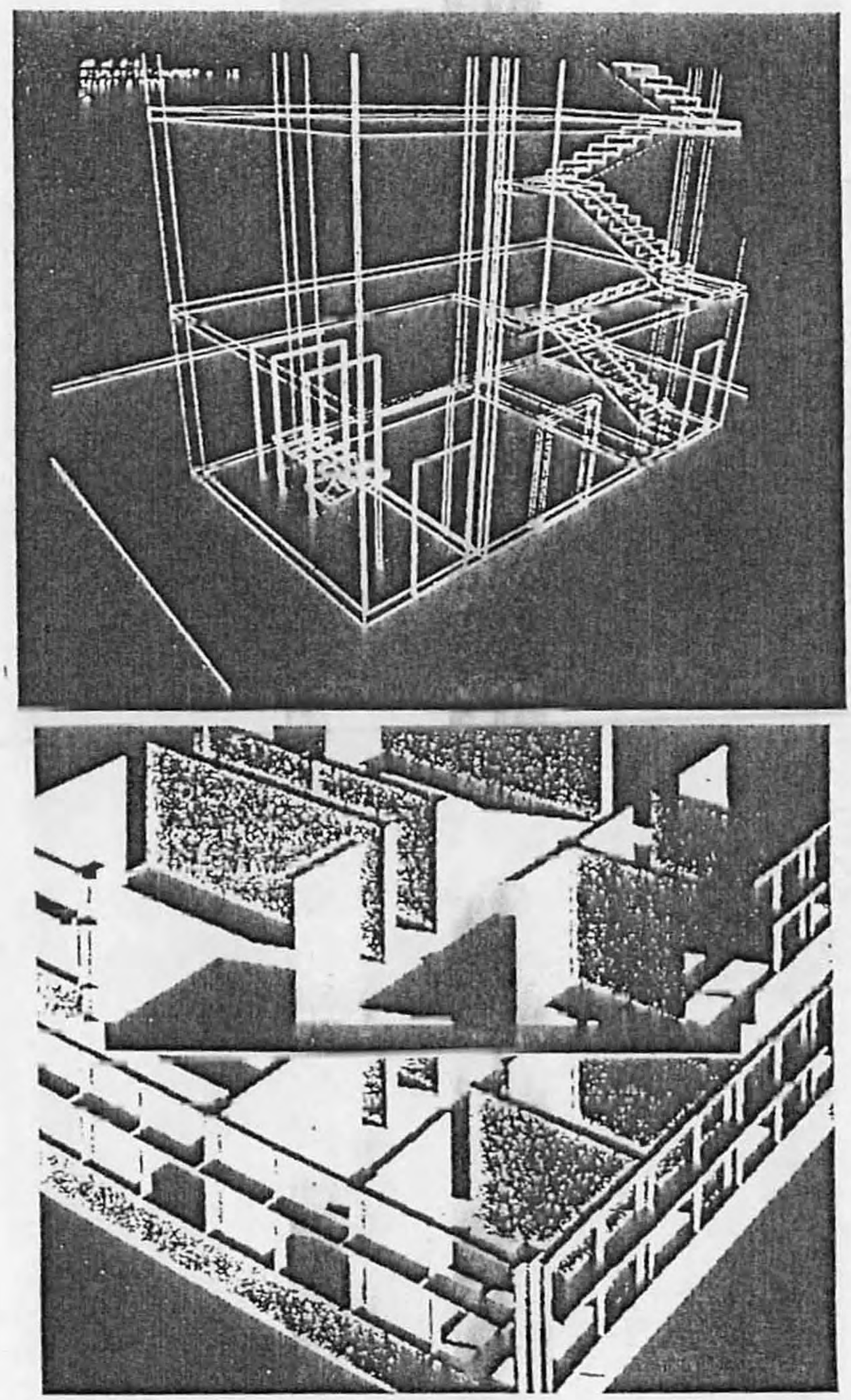
Sketches are drawn with the stylus on the digitizing tablet or on the screen (using a light pen) and even text can be written, then the computer emulates the free hand drawing or the hand writing and produces neat lines and text, which is now known as neuro-networks, or neural-networks. Wire-frames could be generated from plans to produce isometrics or perspectives. Hidden lines could then be removed to give a realistic view. A paint program could then color the surfaces with a wide choice of shades or colors to create a more realistic effect, as in fig. (4.3).





The pictures in this exposition have been produced by students in the School of Architecture & Environmental Design in SUNY-AB. using the WORLDVIEW system for three-dimensional modeling, courtesy of Maedl Computer Research Corporation, Inc. (running on a VAX-11/750), and a DIGITAL PAINT system for coloring them, courtesy of DIEM software (running on an AEDS-11 system).

Fig. (4.3a): A 3-Dimensional view of a building, rendered to give a more realistic effect.



Conceptual modeling graphic output: (top) wire frame drawing (*Courtesy of Charles M. Eastman.*); (bottom) Planes filled in as solids to create a more realistic representation of an interior space. (*Courtesy of Skidmore, Owings, and Merrill.*)

Fig. (4.3b): A wireframe, and planes filled in as solids.

4.1.1.2. Integrated Design Systems:

The conceptual modeling schemes mentioned above, are used primarily as a tool for visual analysis of a design. While a visual evaluation is important, it is certainly not the only criteria needed for making a trade-off between the different alternatives, or for the evaluation and the selection between several designs. The need for a much larger range of evaluation techniques has led to the concept of integrated design systems. The basis of these systems is a model of a building design that is input into a computer database. The intent is to develop a model that is complete down to every necessary detail and the smallest piece of information. If this can be achieved, ideally the architect will no longer need to work in the medium of pencil and paper; all design development, design detailing, working drawings specifications and construction documents will evolve from the computer model.

Today an architect must draw and record graphic, as well as non-graphic information about every item that makes up a building. For example, a floor plan will show the size of a door and its corresponding opening, but additional information on that door is recorded elsewhere, this information will include material, thickness, manufacturer and fire rating. On the other hand, an elevation will indicate the size and location of a window, but the architect, the engineer and the contractor also need to know the frame material, the number of panes and the U-value and type of glass. An integrated computer system will tie all relevant non-graphic information to each graphic item. This collection of information is called a database.

Once the necessary data is input into the computer, the database contains enough information to support a variety of application programs. These programs can interface with the

database, and provide numerical evaluations such as the expected building costs, heat loss and energy use, and daylighting values. Unfortunately, in most cases today, one database is not available that can handle all of these programs. Therefore, similar and repetitive information must be input separately for each program, which is known by data redundancy. An integrated approach would avoid this time consuming repetition or this data redundancy.

Another benefit of this integrated approach is the ability, due to the interactive nature of computers, to easily alter data such as U-values, type of materials used, and percentage of window area; by trying different values for these variables, while running analysis programs, it is easy to see how these changes affect the performance of a certain building.

It is obvious that the objective of an integrated computer-aided architectural design system is to eliminate the drudgery of the routine work in architecture, and to free the architect in order that he may perform in the capacity of which he is best suited, namely imagination, intuition, and creativity. A modern, systematic design procedure must offer explicitness in its process, in order to enable the architect to make his decisions more rationally, and communicate more effectively.

According to Gero (1977), an integrated computer-aided design system cannot be made up of one gigantic program that does all the tasks that an architect undertakes. This would be too complicated as well as too tedious to be of use. In order for a process to be classified as "computer-aided", certain characteristics must be present. There needs to be a certain dialog between the three components of an integrated computer-aided design system: viz. the user, the data and the machine; (Gero, 1977). This implies that an answer cannot

simply result from the input of a set of data. Instead, the computer and designer have to trade information in order to get a solution.

Two of the examples of integrated systems are: the ARK-2, and the OXSYS systems.

The ARK-2 system is considered to be the first major interactive design system. The name ARK-2 comes from the words: "Architectural Kinetics- Man and Machine", (Reynolds, 1980). The system attempts to integrate nine programs through a common database and user-machine interface. The nine programs are designed to carry out the following functions: produce a database, assist the architect in the development of schematic layouts, produce drawings, calculate areas, generate perspective views, aid the development of the project specification, carry out critical path network analysis for project scheduling, assist designers by providing a mathematical scoring program to produce high scoring solutions, and finally, to provide the architect with a simple office management reporting system. The ARK-2 system is in use in the USA, Europe and Australia. Figure (4.4) shows one of the outputs of the ARK-2 system.

The OXSYS system represents a highly integrated interactive design system, specially centered on the Oxford Regional Health Authority's Oxford Method of Construction, although it could be applied equally well to other construction methods. The system can be used in various stages of the design process; starting from (briefing) or room data development, then designing different alternatives through the design development stage, and ending with the working and shop drawings and schedules in the detailed design stage. The system is prototypical of many integrated computer-aided building design systems.

The system accepts information from the design team, and gradually assembles a fully detailed model of the building in the computer's memory. Using this model, the system carries out a large set of evaluation procedures to guide the design team towards better designing by predicting the cost and performance of alternative solutions. Figure (4.5), illustrates a computer generated isometric for a hospital building produced by the OXSYS system.

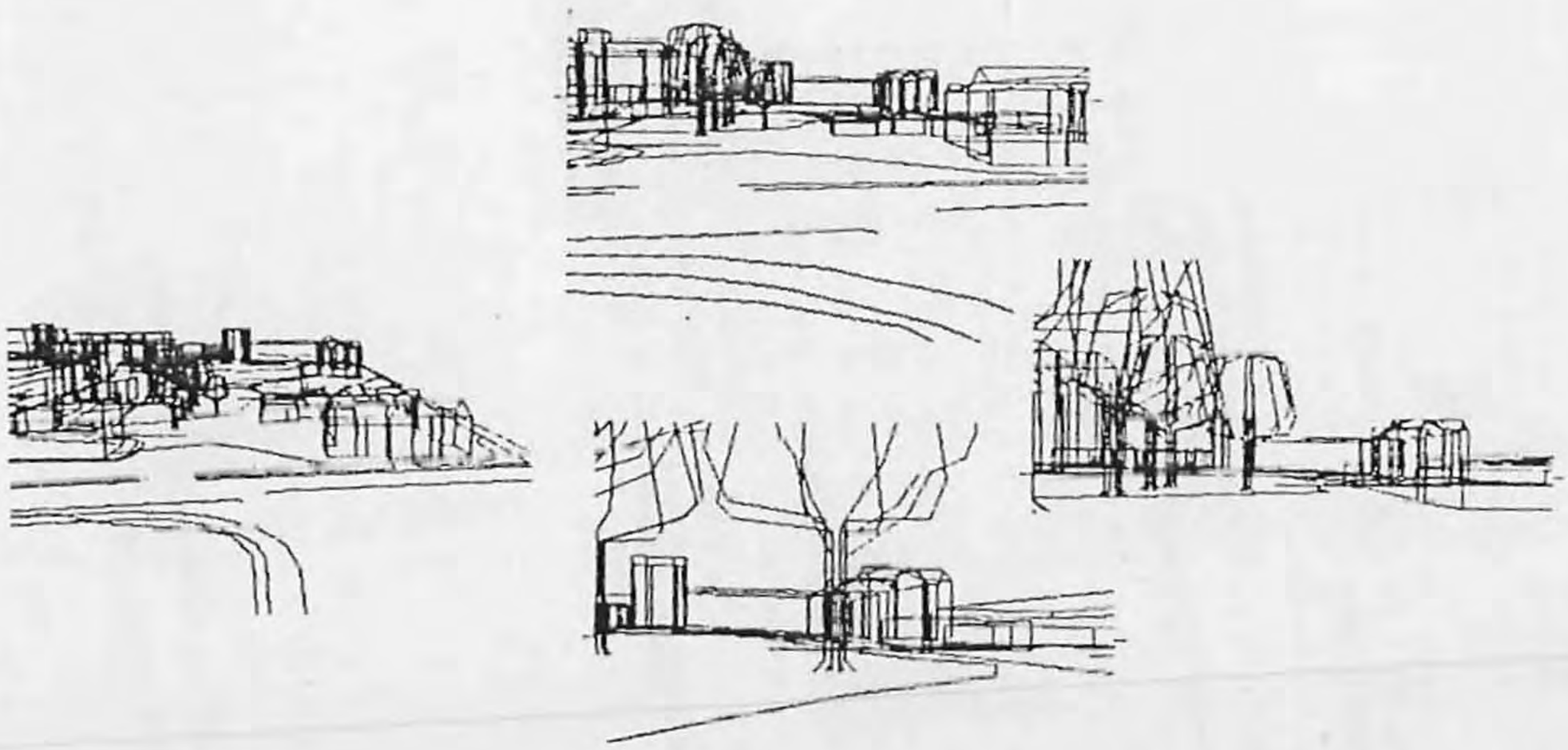


Fig. (4.4): Exterior perspective views, by the ARK-2 system.

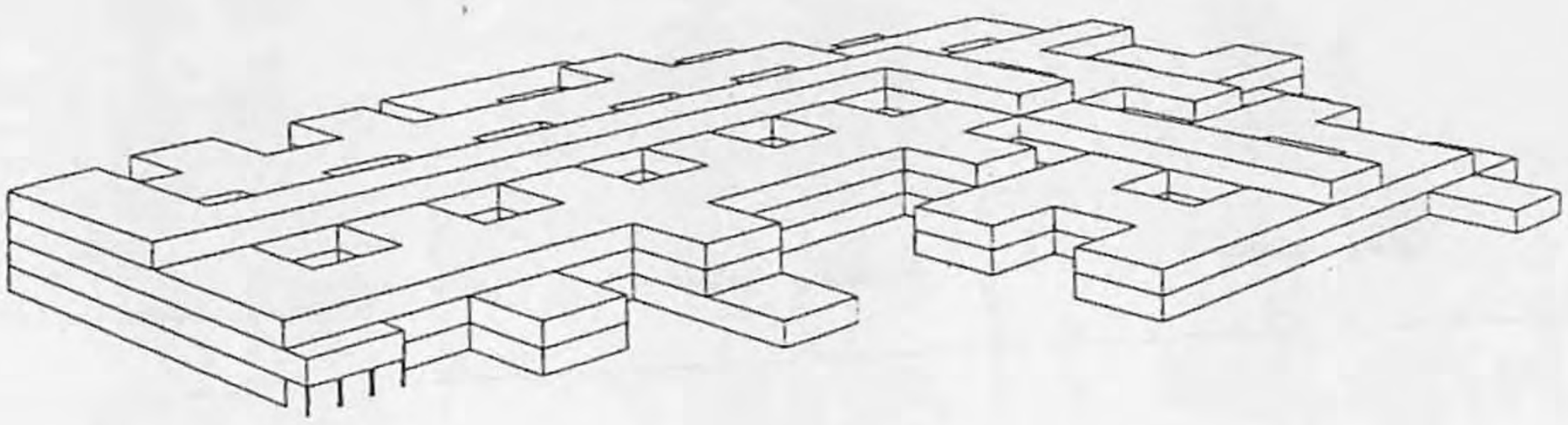


Fig. (4.5): A 3-Dimensional isometric by the OXSYS system.

Unfortunately again, computing in architectural design, as it is being implemented today, is not integrated into the basic design process. The different application programs were developed independently, outside a comprehensive framework, and were presented individually as computational aids for architectural design. Although they were initially received with much anticipation and excitement, their reductionist approach hindered their implementation as integral tools in the architectural design process. These application packages were basically evaluative and generative and functioned individually not as a part of a system, making cross referencing and connections impossible. They were not inter-related and did not cover the architectural design process in its totality (Kalisperis, 1988). The different CAD tools are not packaged into a system that follows naturally through the design process, which makes the architect unable to "enter" it at any desired mode or phase (Broadbent, 1987).

Recent developments in CAAD program packages have provided some compatible programs, enabling architects to perform different tasks using the same data. However, the scope of such programs is limited to small tasks and simple manipulations within small projects (Beheshti and Monroy, 1987). The true integration of computers into the architectural design process is not progressing well. The current computer aided design still do not meet the needs and expectations of the architect (Lindhult, 1987).

In order to facilitate computing in the architectural design process, a flexible decision support system environment needs to be developed, which is the main goal of this thesis.

On the other hand, research and development efforts in the offices of SOM (Skidmore, Owings and Merill), RTKL, and HOK (Hillmuth, Obata and Kassabaum), are geared toward an integrated CAAD system (Gamboa, 1987). This is a

technological answer to the technology based creation of various specialized disciplines involved in the building process. In the days of Vitruvius, the master builder took full responsibility of the design and construction of practically anything. Today, the advancements in technology have produced architectural, structural, electrical and environmental engineers, as well as interior designers and landscape architects, all of whom partake in the responsibility of building. However, the segmentation of this growing responsibility has created problems in communication. With integrated CAAD systems "a closer understanding generated between the various members of the design team is likely to ensue" (Maver, 1986).

4.1.1.3. Space Planning:

Space planning problems are popular tasks to develop into computer programs that support the architectural design process. In the 1970's, approximately 25% of the published papers and the released software programs addressing computing in architecture, were concerned with space allocation (Broadbent, 1973). Quite understandably, the issues surrounding space planning are highly quantifiable. Creating space, especially in its two-dimensional (2-D) form such as floor plans, often becomes a geometric exercise involving a series of repetitive trials and errors until some suitable and logical layout is created. It is also this mathematical characteristic that has persuaded architects to be receptive to the use of computers for space planning (Gamboa, 1987).

Computer assisted space planning is being implemented with increasing frequency in the interior and architectural design fields. All phases of space planning have been successfully computerized, beginning with analysis and proceeding through design (synthesis) and evaluation.

The analysis phase collects and interprets the necessary specifications for the design. The functional and inter-relationships, and the physical size requirement for each space must be determined. Future users of the facility should be interviewed to find out exactly what they do, with whom they interact and what is their minimum and maximum spatial needs. This information can be quantified, entered into the computer and used in the subsequent design phase. See fig. (4.6).

ADJACENCY MATRIX

	ADMIN.	PERSONNEL	ACCOUNTING	PLANNING	RESEARCH	DATA PRO.	EXE. OFF.	GENL. OFF.
ADMIN.	*	1	2	3	3	2	1	2
PERSONNEL	1	*	3	3	3	3	4	3
ACCOUNTING	2	3	*	4	4	1	4	2
PLANNING	3	3	4	*	1	3	1	3
RESEARCH	3	3	4	1	*	3	1	3
DATA PRO.	2	3	1	3	3	*	5	4
EXE. OFF.	4	4	4	1	1	5	*	4
GENL. OFF.	2	3	2	3	3	4	4	*

ADJACENCY RELATIONSHIP

1. VERY CLOSE
2. CLOSE
3. INDIFFERENT
4. FAR
5. VERY FAR

Fig. (4.6): An adjacency matrix,
a beginning step in space planning.

Computer programs in the design (synthesis) phase take the functional and spacial relationships determined in the previous process, and create bubble diagrams and floor plans, as in fig. (4.7).

This phase is generally the most controversial one, since it is here that the computer actually does design. Some planning programs offer an evaluation phase, which calculates the amount of circulation space in each plan, and then compares them. The usual assumption is that the better plan is always the one which minimizes the amount of circulation space.

There are two types of available space planning programs. One type would accept input regarding room areas and their adjacency requirements. The algorithm utilizes these input data and later generates various block layouts. Some programs take the algorithm one step further by extending the block layout into three-dimensions (3-D). This produces an image of what the massing would be like when all the block layouts are piled on top of each other. The objective in these kinds of programs is to generate an optimum circulation and space allocation model for a given problem.

The other type of space planning program assumes an existing layout. Given the necessary data, it tests the validity of the layout, and at times generates further suggestions. It should be noted, however, that the solutions generated by both types of programs are not always buildable. The reason for this limitation, is that in space allocation, more data is required aside from areas and adjacencies. It is essential to know where the external walls should lie, the size and location of corridors and doors, the shapes of the rooms and others. These data often create a conflict in requirements. Preferences as well as weights are needed to inform the program of the priorities of the given requirement.

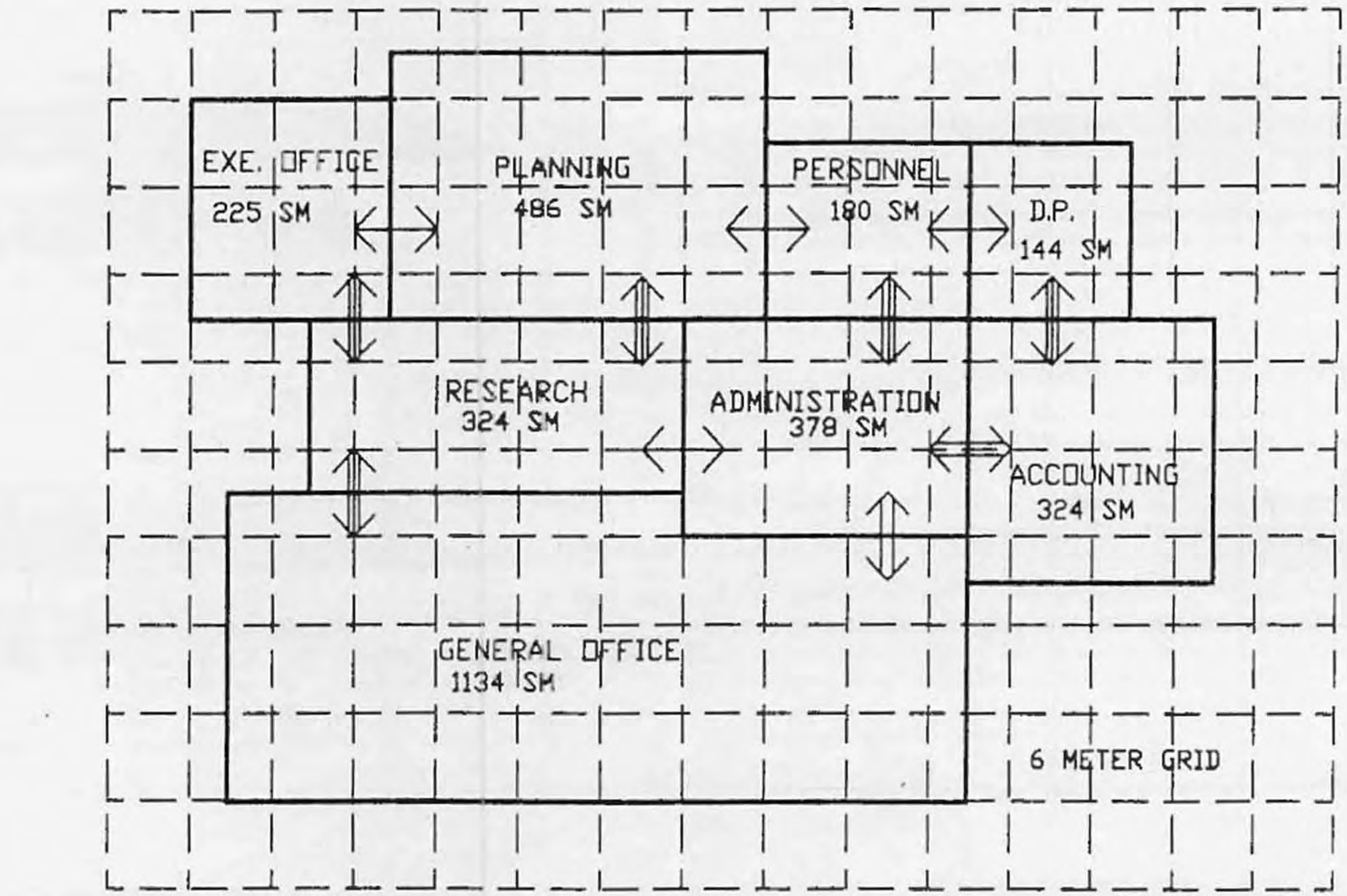


Fig. (4.7): a trial floor plan,

generated from functional requirements.

Even then, space allocation problems usually have too varied a set of constraints that are unsolvable through a computer which is programmed on a pre-defined set of rules (Gamboa, 1987).

Many other programs can be used by interior designers to assist in various aspects of their work. Computer graphics allows the designer to plan furniture layouts by storing the furniture symbols permanently, making a furniture library, and trying different arrangements on a floor plan. Computer generated purchase orders can be produced from these layouts, if the database contains attribute information such as the size, color, material and availability of each piece of furniture.

Interior design and space planning can also take advantage of many non-graphic capabilities of the computer. There are programs available for enabling the user to enter, store, manipulate and print out information on room types, finishes, furnishings and accessories. See fig. (4.8).

ROOM NO	ROOM NAME	FLOOR MATL	FLOOR FIN	BASE MATL	BASE FIN	NORTH MATL	NORTH FIN	SOUTH MATL	SOUTH FIN	EAST MATL	EAST FIN	WEST MATL	WEST FIN
1-103	VESTIBULE	CONC	MWF2	GWB	RB1	GWB	FWP2	GWB	FWF2	GWB	FWP2	GWB	FWP2
1-104	CARPENTRY SHOP	CONC	VAT1	GWB	RB1	GWB	P1	GWB	P1	GWB	P1	GWB	P1
1-105	PROP STORAGE	CONC	VAT1	GWB	RB1	GWB	P1	GWB	P1	GWB	P1	GWB	P1
1-106	PROCESSING ROOM	CONC	VAT1	GWB	RB1	GWB	P1	GWB	P1	GWB	P1	GWB	P1
1-107	PRINT ROOM/DARK ROOM	CONC	VAT1	GWB	RB1	GWB	P	GWB	P	GWB	P	GWB	P
1-108	FILM LOADING	CONC	VAT1	GWB	RB1	GWB	P	GWB	P	GWB	P	GWB	P
1-109A	PRODUCTION CONTROL RM	AFL	C9	GWB	RB1	GWB	FWP2	GWB	FWP2	GWB	FWP2	GWB	FWP2
1-109B	AUDIO CONTROL ROOM	AFL	C9	GWB	RB1	GWB	FWP2	GWB	FWF2	GWB	ROCK	GWB	FWP2
1-110	MASTER CONTROL ROOM	AFL	C9	GWB	RB1	GWB	P1	GWB	P1	GWB	P1	GWB	P1
1-111	ENGINEERING TECH. SHOP	AFL	PL8	GWB	RB1	GWB	P1	GWB	P1	GWB	P1	GWB	P1
1-112	TAPE STORAGE	CONC	VAT1	GWB	RB1	GWB	P1	GWB	P1	GWB	P1	GWB	P1
1-113	RECEPTION	DPF	C8	GWB	RB1	GWB	P1	GWB	P1	GWB	P1	GWB	P1
1-114	CORRIDOR	AFL	C9	GWB	RB1	GWB	FWP2	GWB	FWF2	GWB	FWP2	GWB	FWP2
1-115	OFFICE	DPF	C8	GWB	RB1	GWB	P1	GWB	P1	GWB	P1	GWB	P1
1-116	VIDEO TAPE EDITING ROOM	AFL	C9	GWB	RB1	GWB	FWP2	GWB	FWF2	GWB	FWP2	GWB	FWP2

Fig. (4.8): A computer generated room finish schedule.

(Courtesy of Skidmore, Owings, and Merrill.)

The cost, quantities, colors and manufacturers of every item can be entered and stored in the computer. Reports are produced that can be continually updated through the course of a project.

Computerized space planning has also appeared in contexts other than interior design. Site designers can effectively make use of similar algorithms to generate alternative plans locating buildings, pedestrian and vehicular routes, parking and recreational areas. On a smaller scale, various apartment or office units can be combined into alternative building schemes, and compared to find the effects of various layouts, and building shapes on project costs.

4.1.1.4. Computer Mapping:

Computer mapping techniques were one of the earliest applications of computer graphics. Programs that produce contour maps, topographical perspectives, and grid cell maps are commonplace and easy available. Data to describe a particular site can be input into the computer, and both 2-D and 3-D contour maps can be generated, see fig. (4.9).

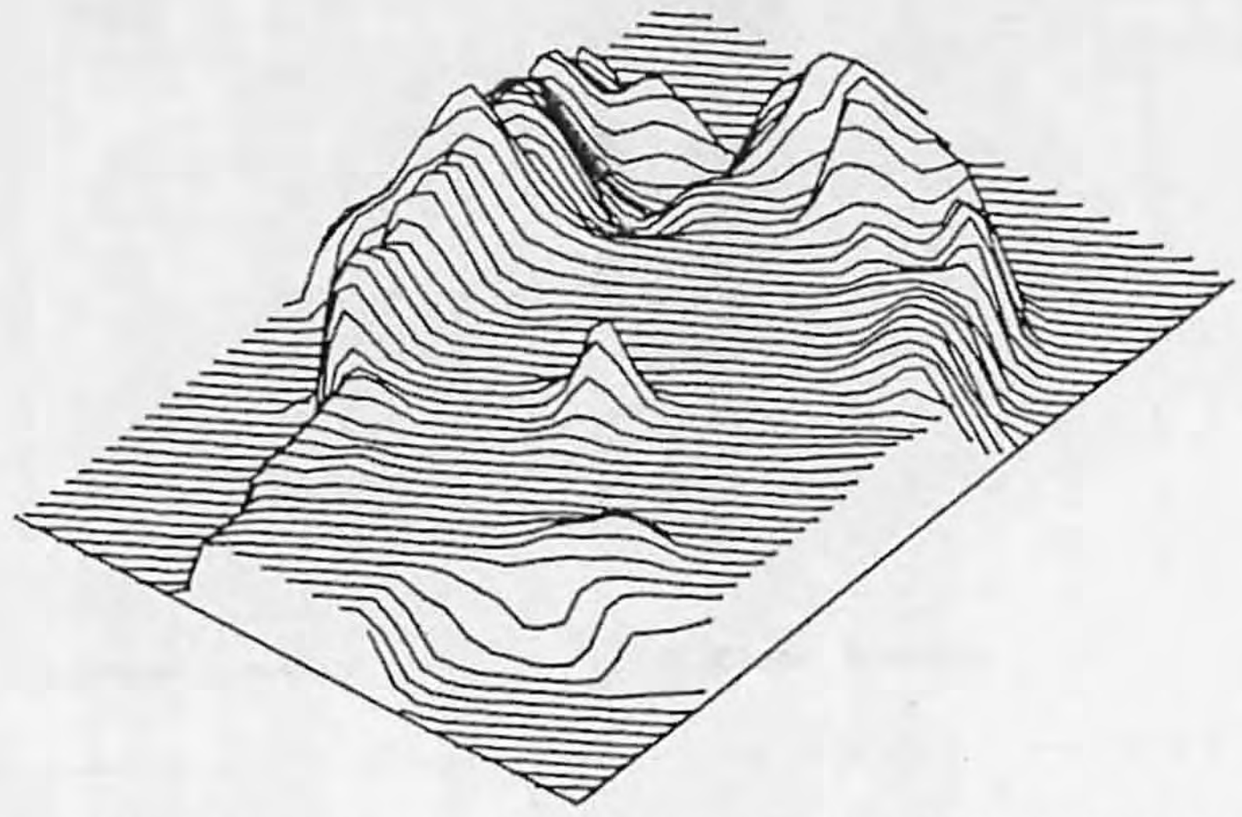


Fig. (4.9): Computer generated 3-dimensional contours.

The same data can be used to run certain processes like cut and fill calculations, slope and water run-off analysis for a site, road layout, land use costs and others.

4.1.1.5. Site Planning and Land Analysis:

In recent years, the computer has become even more valuable to landscape architects and planners as programs have developed which provide the capacity for large and detailed databases allowing intricate land analysis. The computer is useful for town planning, for in this scale, the amount of useful information available is too vast to be handled by the designer in an efficient manner. The types of data that can be included for large scale planning are almost endless. These types of data may include information on soil type , slope, mineral resources, surface water quality, vegetation, sewer and water service, transportation facilities, public properties and existing land use. These variables, as well as many others, can be displayed on maps either singly, or in combination as type of overlay-system. See fig. (4.10).

Fig. (4.10): A map produced by printing variable data values.

All of the programs mentioned in this section, are viable examples of how the computer can assist the architect as he designs, although this is by no means an exhaustive list. With the exception of the design phase of the space planning sequence, no program attempts to do the architect's job for him. Rather, these have all been examples showing how the computer can assist the architect by analyzing the designer's alternatives, by quantifying and organizing vast amounts of data, and by providing new information about a proposed design that the architect would have been unable to receive otherwise.

4.1.2. Technical Applications:

This section will examine some of the many technical applications that architects and engineers use in further designing and refining a building. The common feature that ties them together is their reliance on mathematical calculations. The engineering disciplines which act as consultants to architects are included here, as well as other important applications, such as thermal performance, and cost estimating.

4.1.2.1. Structural Analysis:

Structural engineering was one of the first engineering disciplines to consider computer methods for analysis. Like space planning problems, engineering-oriented activities are highly quantifiable. Program development began in the 1950's, and by the 1960's programs were available to calculate loads, moments, thrusts and shears, and then used these values to select member shapes and sizes, and to provide an estimate of material quantities. Databases include tables of available shapes and their properties, and necessary code restrictions which limit the acceptable members.

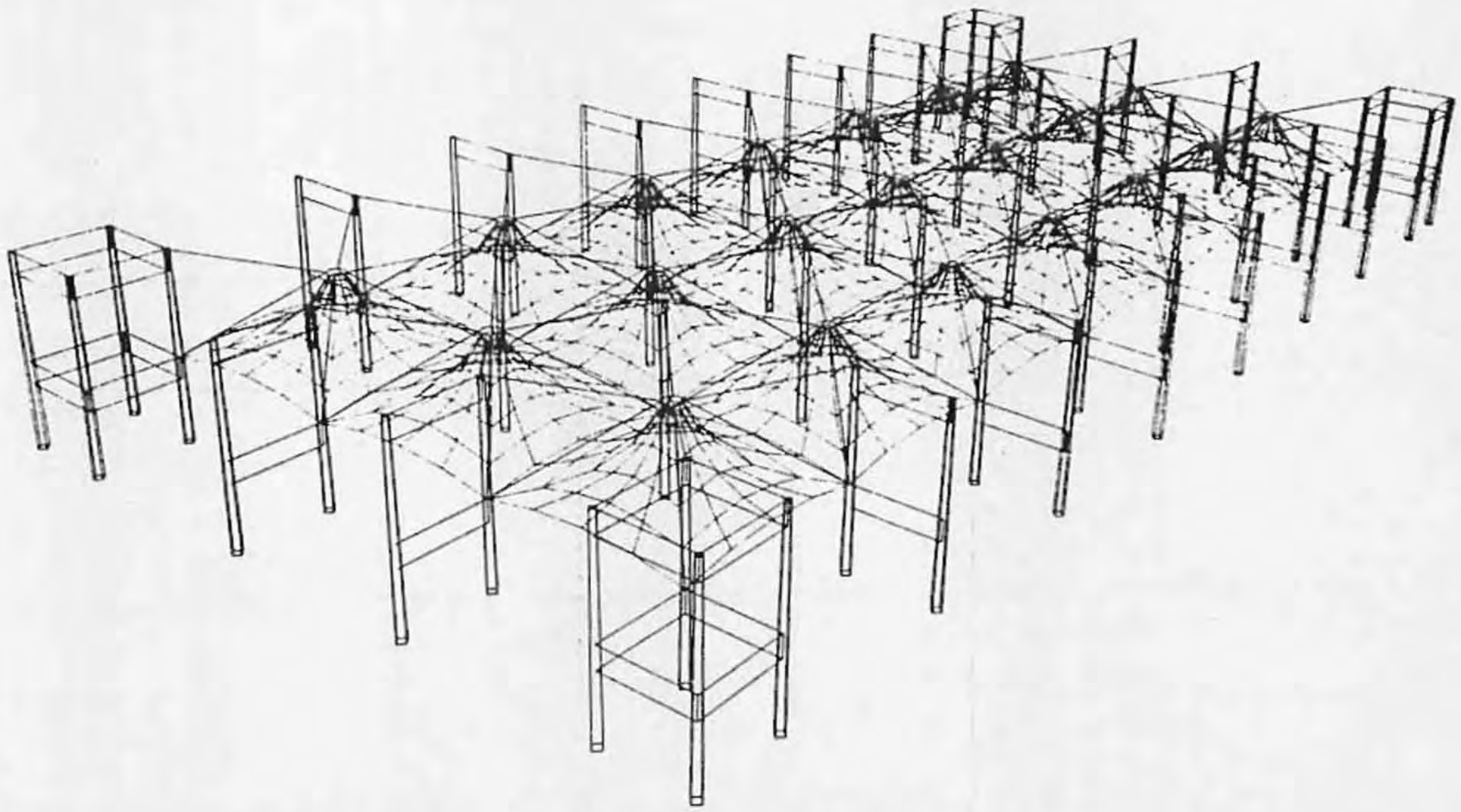
The users of these programs, however, are mostly engineers and technicians, though architects sometimes venture into using these programs, only to give them some preliminary information. These programs, once available only at universities and the large firms, are now being implemented on smaller computers in smaller firms.

More recent structural application programs are concentrating on developing comprehensive systems for information processing of structural data.

The usefulness of an integrated system for structural analysis is apparent, since the output of many of the smaller programs mentioned can become input for another. For example, the output of programs to determine column axial loads, moments, and wind moments and shears, can be input for a column design program whose output in turn can be used to design lower story beams and girders, baseplates and footings.

Other more recent developments are programs which use finite element techniques for irregular shapes, and programs which do detailing and joint design, refer to fig. (4.11). A common structural analysis program is called STRUDL. While analysis of buildings is the most relevant structural application for architects, these programs can also handle calculation for bridges, highways, water supply, sanitary and sewage systems.

Soil analysis programs are another set of valuable tools for architects and engineers, resulting in output such as the structural capacity of the soil, and the effects of water on its composition and strength.



MEMBER 2

LOADING 1 DEAD

DISTANCE FROM START	AXIAL	Y SHEAR	Z SHEAR	Y BENDING	Z BENDING	MAX NORMAL	MIN NORMAL
	/---------- STRESS ----------/						
12.000	-90.1000824	0.0	0.0	0.0	-8.0313644	-82.0687103	-98.1314392
24.000	-89.7677460	0.0	0.0	0.0	31.6678772	-58.0998688	-121.435623
36.000	-89.4354248	0.0	0.0	0.0	58.3164215	-31.1190033	-147.751846
48.000	-89.1030884	0.0	0.0	0.0	71.9145660	-17.1885223	-161.01[illegible]654
60.000	-88.7707672	0.0	0.0	0.0	72.4622040	-16.3085632	-161.2[illegible]2971
72.000	-88.4384460	0.0	0.0	0.0	59.9592285	-28.4792175	-148.[illegible]97675
84.000	-88.1061096	0.0	0.0	0.0	34.4058665	-53.7002411	-122.511978

LOADING 2 LIVE

DISTANCE FROM START	AXIAL	Y SHEAR	Z SHEAR	Y BENDING	Z BENDING	MAX NORMAL	MIN NORMAL
	/---------- STRESS ----------/						
12.000	-51.4300879	0.0	0.0	0.0	-41.8651276	-9.5648804	-93.2951355
24.000	-50.9452820	0.0	0.0	0.0	20.4055023	-30.5397797	-71.3507843
36.000	-50.4605408	0.0	0.0	0.0	63.6407318	13.1801910	-114.101273
48.000	-49.9758148	0.0	0.0	0.0	87.8409729	37.8651581	-137.816788
60.000	-49.4910889	0.0	0.0	0.0	93.0057220	43.5146332	-142.496811
72.000	-49.0063782	0.0	0.0	0.0	79.1353607	30.1289825	-128.141739

Fig. (4.11): The use of computers in structural design.

4.1.2.2. Financial Analysis and Cost Estimating Programs:

Financial and cost estimating programs are used both for pre-design and post-design phases. Economics today play an important role in any project. The necessity for accurate cost estimating is recognized, by all members of the building team; the architect, the contractor and the owner. By computerizing the process, results could be obtained faster, and should become more precise, better organized and free from mathematical errors.

Investment strategies are more complex and varied today than yesterday. People always want to know what their money will bring them. Statistics and projections are a central issue and concern in architecture. With the help of computer programs, the leading architectural firms will be able to project how their client's investment will be spent and how much it will earn even before the first sketches are executed.

Estimating should first occur in the early stages of a project, and continually be refined as schematics become design development drawings. Therefore, several levels of computer programs may be useful. Programs run in the early stages of a building's design are approximate only, and use the expected area and functions of each zone to estimate the cost.

For more accurate estimates after a design is complete, a typical database must have a description of the building's components, and materials, as well as the units of measurement of each component. By entering the materials and quantities of a building proposal into the computer, the program will calculate costs. A good estimating program will also consider a variety of related factors such as, overhead, profit, insurance, taxes, local conditions affecting labor and market prices, and expected inflation.

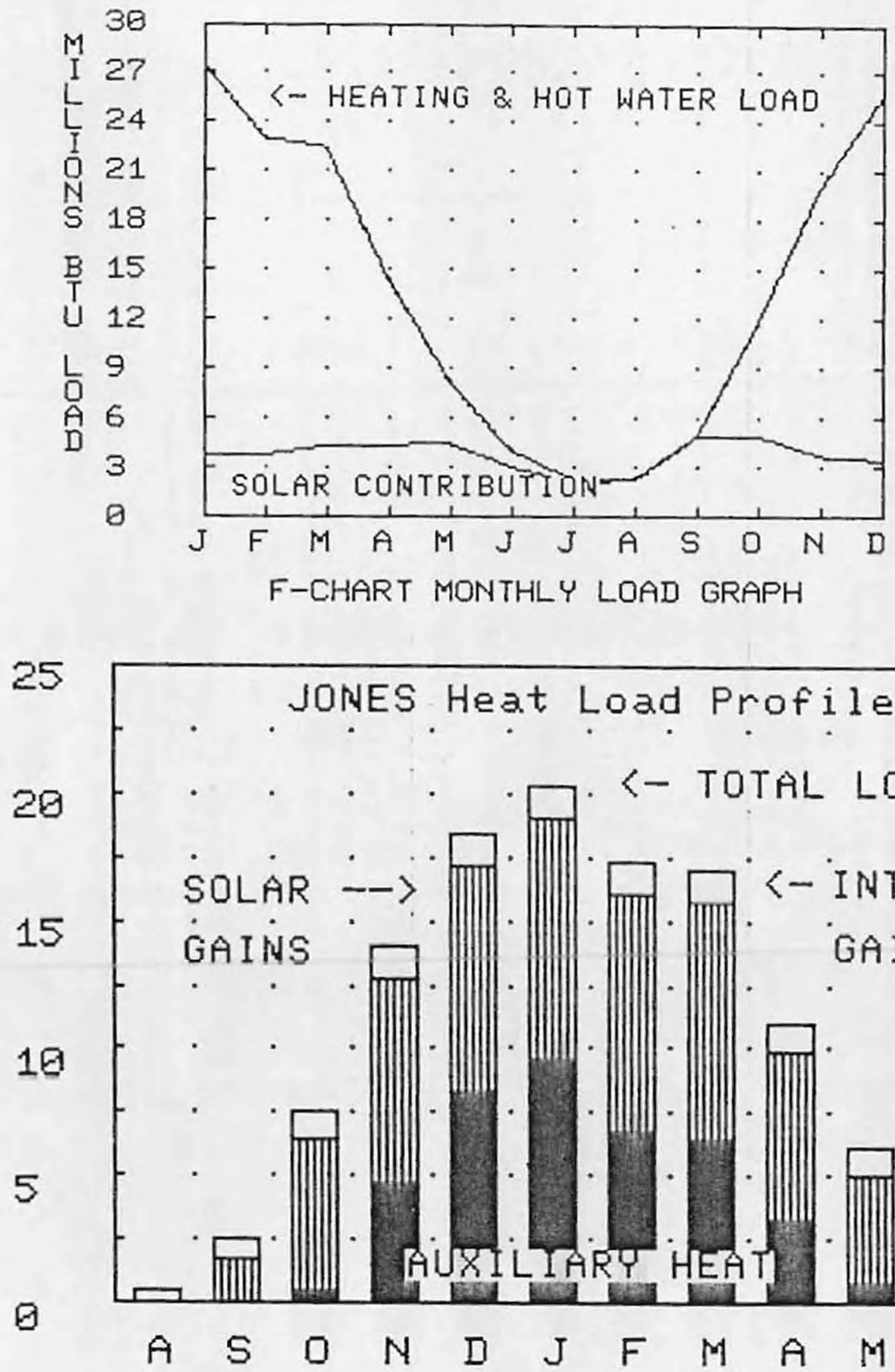
The results can be displayed in a variety of ways, depending upon who is using the information. For example, the architect wants to see costs broken down by system; roof, structure, and the exterior skin. The construction manager is interested in contract by contract costs; electrical, plumbing, heating and so forth. The owner wants costs separated by functional or zoned areas of the building, such as retail, office or parking.

4.1.2.3. Thermal Performance and Mechanical Analysis:

The raise in the fuel costs and energy conservation, as well as the decrease in the cost of computer hardware and software, led to the widespread use of computers to predict and evaluate a building's thermal performance. Energy evaluations of a design in the early stages must be fast, simple and accurate, and the computer is the ideal tool to provide this capability.

Many programs are available that allow the user to define a building to the computer, and receive in return an estimate of energy consumption due to heating and cooling loads, lighting and building services. Some of the programs run interactively , and have the ability to show the outcome in graphic form, as shown in figure (4.12). Methods of analysis vary, from interactive steady state calculations to a more stochastic day by day simulation. Special routines might include shading and other passive solar issues, or allow the user to compare the effects of various openings for a wall, a window, or a door.

The USA government has sponsored the development of several very large, comprehensive programs (DOE2, BLAST), Other groups offer energy related programs that are considerod to be very valuable tools for architects to use in all the phases of the design process (Leighton, 1984).



Energy performance analysis output in graphic form. (top) Sunpas Program; (bottom) F-Chart Program. (*Courtesy of SOLARSOFT, Inc.*)

Fig. (4.12): Energy performance analysis output in a graphical form.

In the mechanical engineering field, computer programs are available which assist in the HVAC processes, they aid in the selection of duct systems, piping systems, fans, cooling coils and so forth. Interactive programs allow the user to accept or reject what the computer has selected, and adjust certain variables to produce different results. As in the structural programs mentioned in the previous section, the database must contain current tables of available systems and their relevant properties, from which the program can select the optimal choice. Once the selections are made, other routines can estimate the cost for the chosen system. Other programs simulate an appropriate mechanical system for a building, and run evaluation tests of building performance with that particular system. Mechanical analysis programs are often combined with thermal analysis programs in one complete package.

4.1.2.4. Lighting and Acoustics:

A variety of programs are available that manipulate data concerning natural and artificial lighting, for use in both, indoors and outdoors. A typical program that is useful for architects predicts the amount of natural and artificial light which will enter a room. Output is usually in numerical form, giving the value of light intensity at any particular point in the room, or on any surface. Recent programs, give graphical outputs as well, represented by different grades of grey colors on the floor plan, as well as in the cross sections.

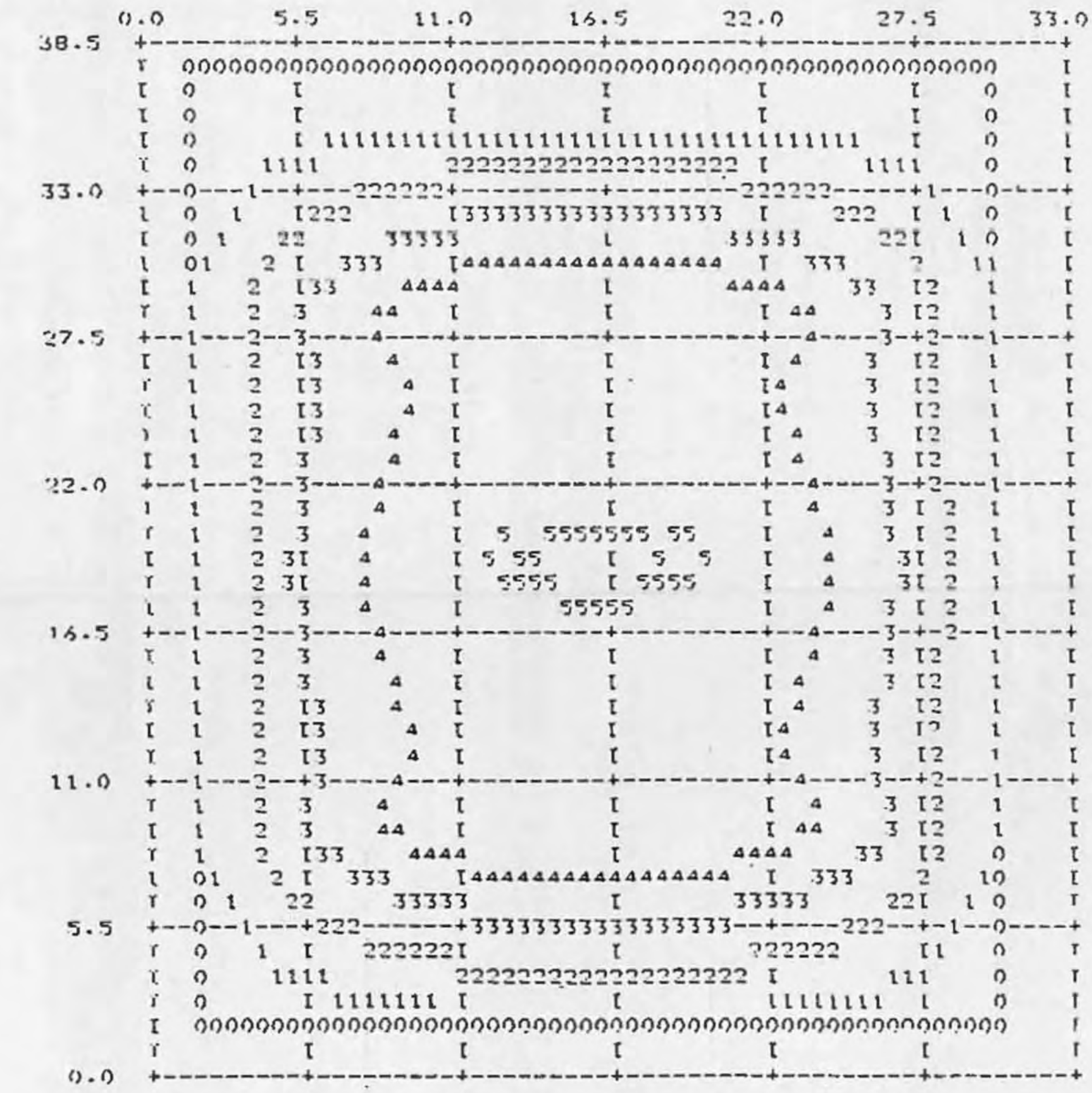
Sophisticated programs will also consider atmospheric conditions, nearby buildings, group luminance, and inter-room reflection from objects found inside the room. If desired, this output can be used as input to routine mapping programs, and a "contour map" of lighting levels can be drawn, as in figure (4.13).

CONTOUR PLOT

ROOM SURFACE LUMINANCES - FLOOR VIEWING

```
QUANTITY PLOTTED:   LUMINANCE IN FL
SCALE: 1 INCH=   5.5FEET
VALUES:     SYMBOL:
    1.526    0
    3.540    1
    5.555    2
    7.569    3
    9.584    4
   11.599    5

HORIZONTAL:  16 OUT OF  19 HARMONICS USED.
VERTICAL:  16 OUT OF  19 HARMONICS USED.
```



Contour plot (plan view) of room surface luminance values. (*Courtesy of Computer Sharing Services.*)

Fig. (4.13): Contour plot (plan view) of room surface luminance values.

For artificial lighting problems, a data library of typical lighting fixtures is useful. The architect can choose a particular fixture, specify how many, and where they will be used, and the output can predict the level of illumination produced. Conversely, if a certain footcandle level is desired, the computer can determine the optimum number and location of fixtures needed to produce that level. To make such programs even more useful, the inclusion of initial and operating costs in the database will allow an economic comparison between various systems.

Simple acoustic analysis can also be easily implemented as a computer program. One typical calculation gives reverberation times in a room. By varying ceiling height and the size of a room, a comparison of values can be made. Output can be numeric, or if desired, plotted as a graph showing the change in value as room size is altered.

Different types of spaces may require special methods of analysis. For example, the techniques used for auditorium design are different from those used for office space or open theaters. Programs can be tailored for these special room types as needed.

4.1.3. Production Applications:

The two major categories of architectural production or design documentation that can be automated are working drawings and specifications. Working drawings show complete detailed locations and arrangements of spaces, structural items, as well as other items of HVAC, plumbing and electrical equipment, and their different relations with each other.

Specifications, on the other hand, record the different selection and assignment decisions. Both of which have been tested and implemented in architect's offices since the late

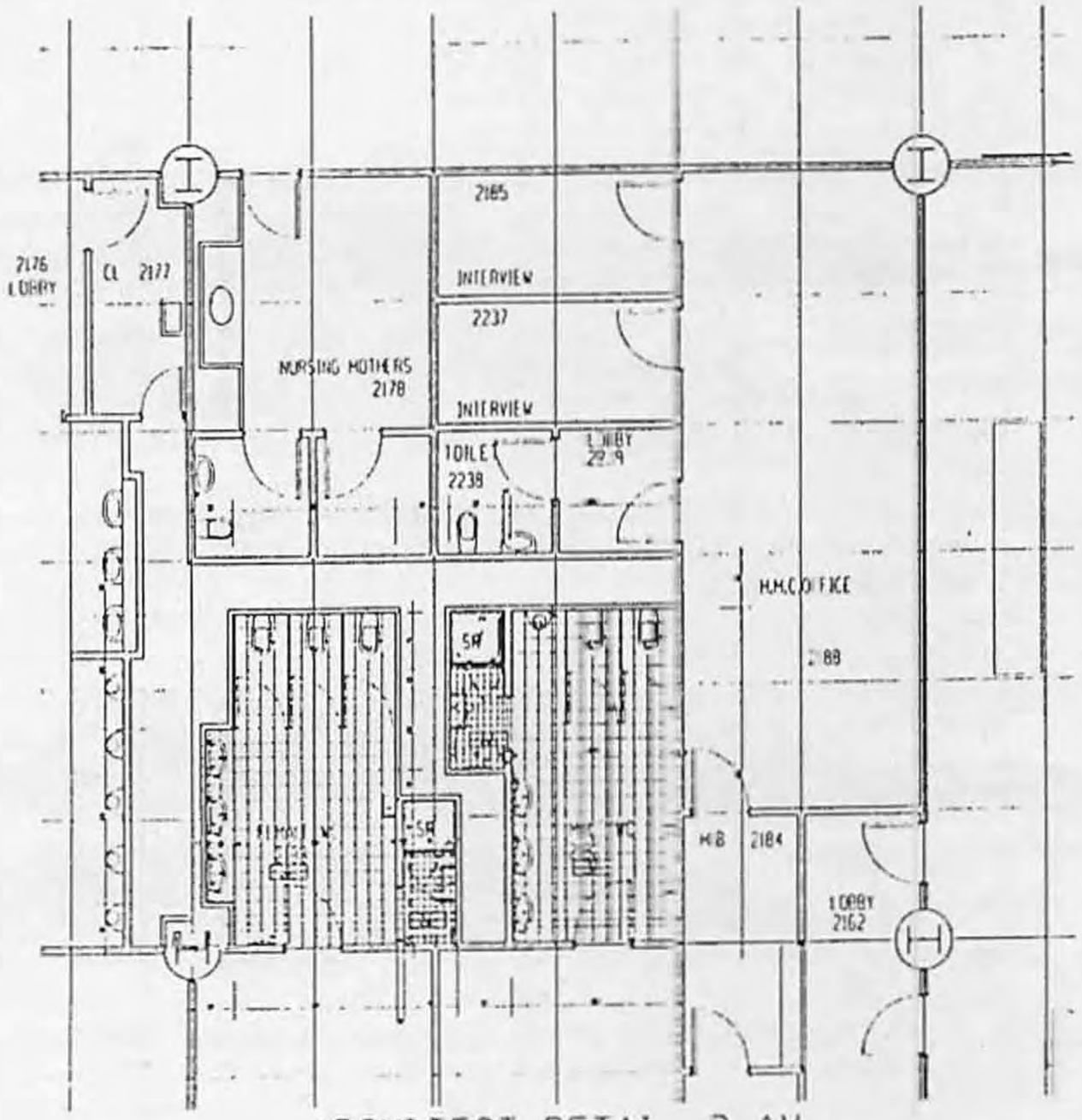
sixties, and now, can be generated at high speed and low cost by the computer.

4.1.3.1. Working Drawings:

The technology of automated drafting is now highly developed and has been widely employed in many fields. The rapidly decreasing cost of computer hardware and software, in addition to the ease of data entry methods, combined with the increasing cost of skilled labor, has made the use of computers in the architect's office a must.

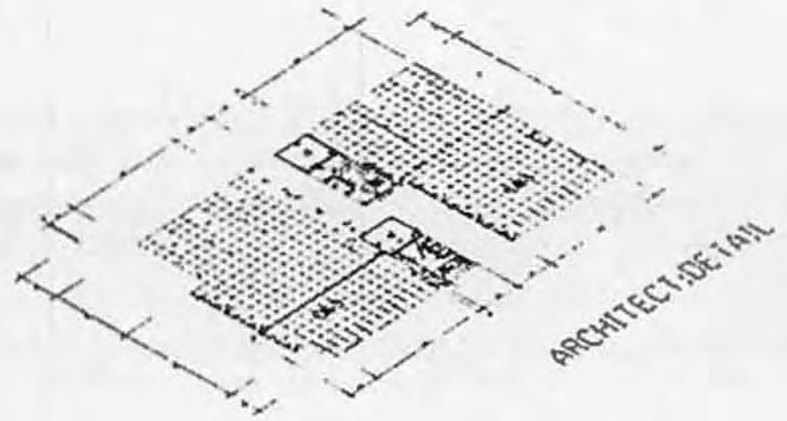
One of the most useful features of computer drafting is its ability to exploit repetition. It is possible to draw figures and symbols that are used repeatedly in the architectural design drawings, and store them in the computer memory to access them whenever needed. This is called a library of symbols. These symbols might include doors, windows, wall panels, bath room fixtures, and even complex items as typical details. When drawings are created, it will then be necessary only to indicate to the computer where these symbols should be placed, rather than having to redraw them each time.

The concept of layers permits a higher form of repetition. All drawing elements may be placed on any of many layers. Each system allows a different number of layers, some allow as high as several hundreds. Computer layers may be considered similar to layers of transparent paper that, when placed on top of each other, create the whole drawing. This is similar to the overlay technique used by many architectural firms today. For example, the architect may draw a floor plan by putting all exterior and interior walls on one layer, with dimensions and text on a different layer. Electrical symbols and lighting layouts may be drawn on a third layer, and mechanical equipment and symbols on fourth.

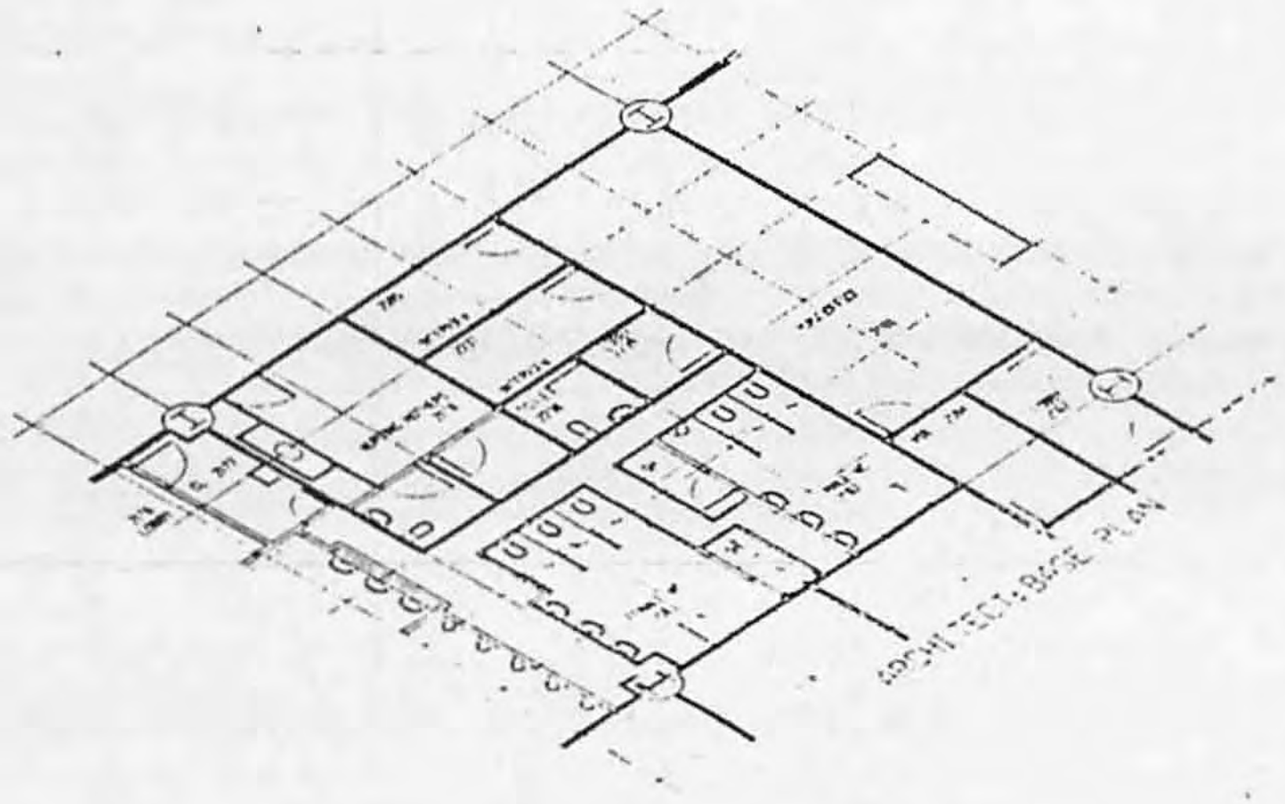


ARCHITECT: DETAIL PLAN

ARCHITECT BASE PLAN + ARCHITECT DETAIL

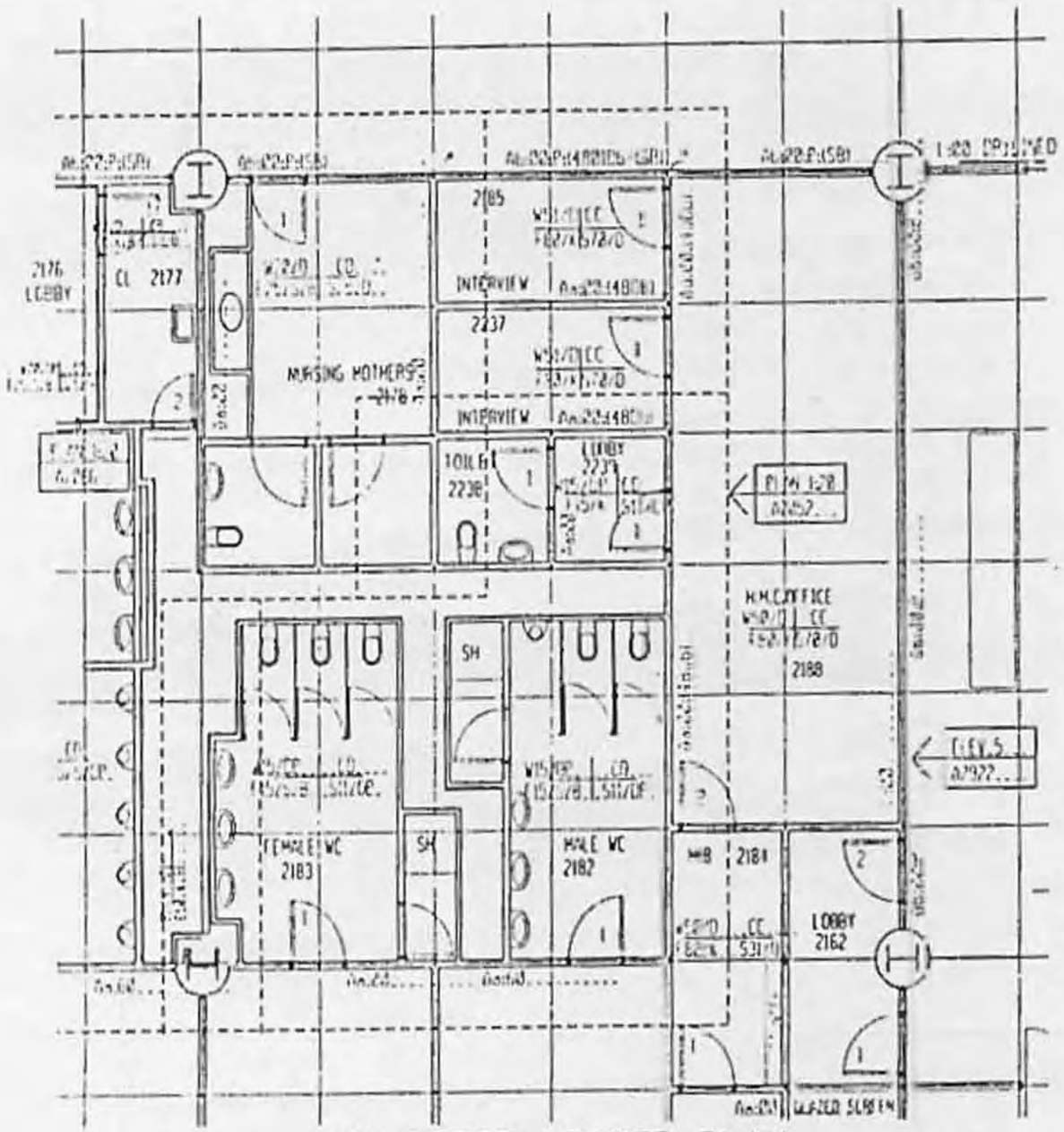


ARCHITECT DETAIL



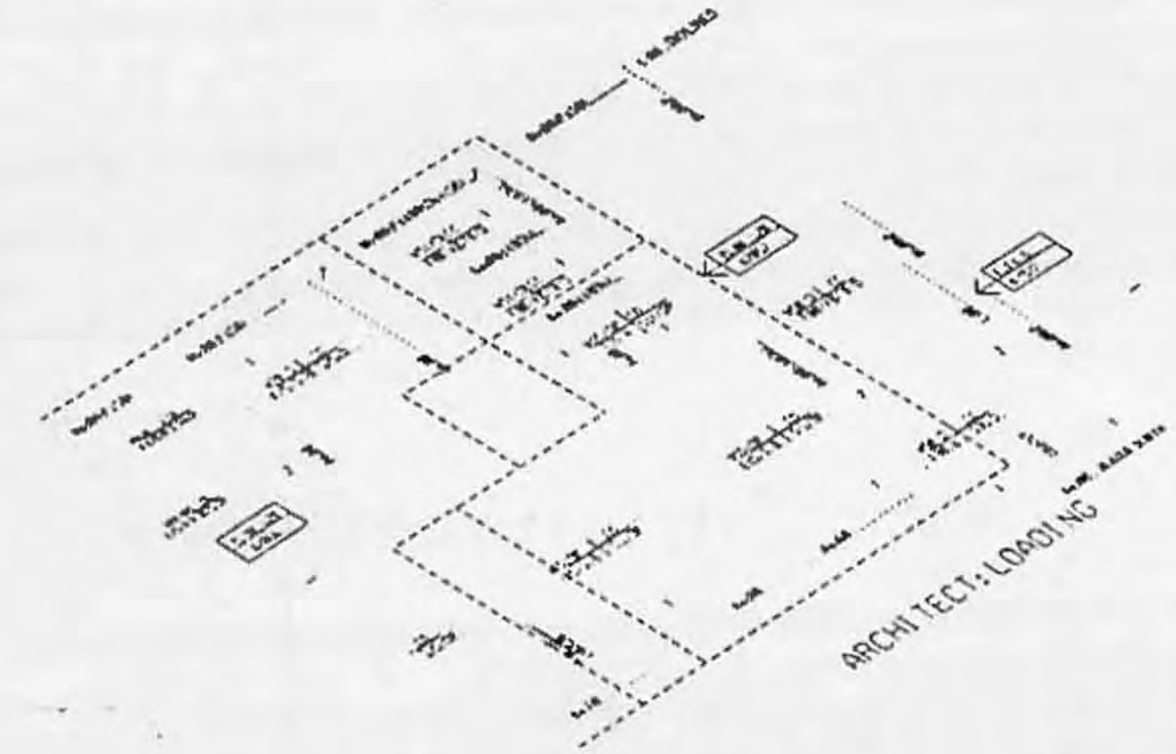
ARCHITECT BASE PLAN

Fig. (4.14a): Each layer in the drawing can be accessed, and plotted alone by itself.



ARCHITECT: LOADED PLAN

ARCHITECT BASE PLAN + ARCHITECT LOADING



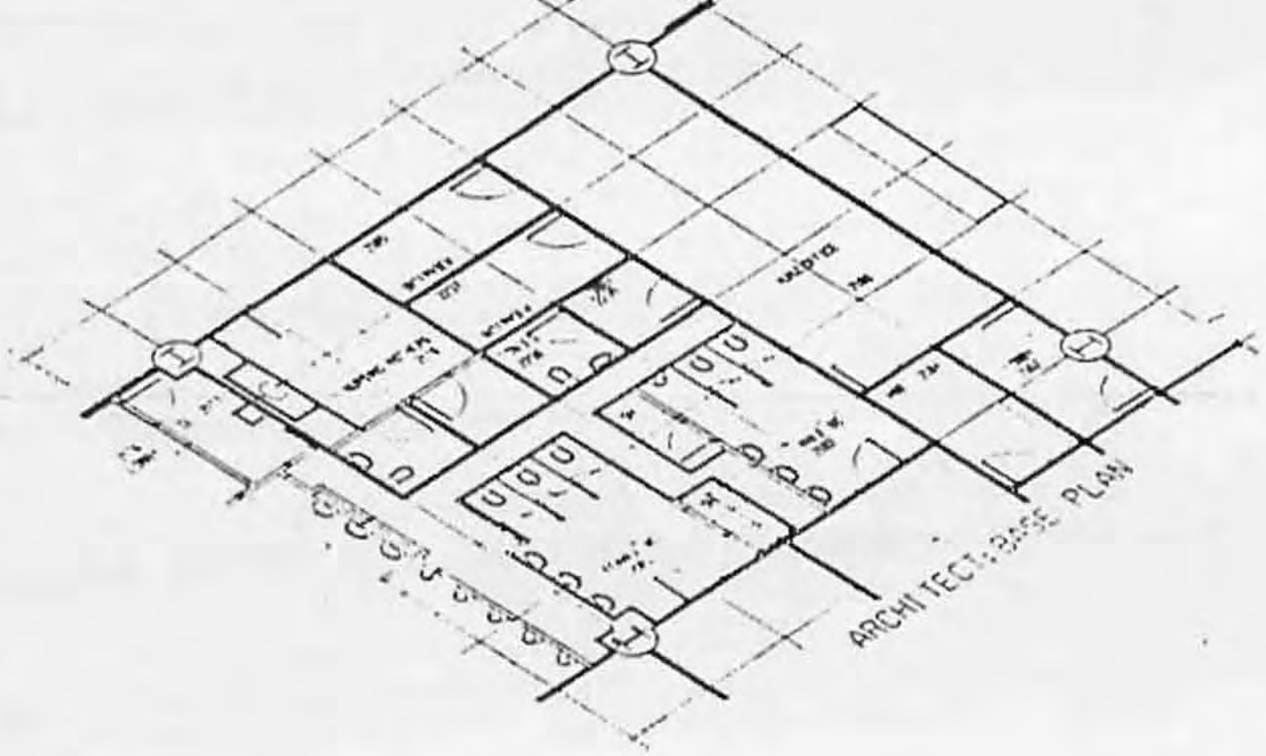


Fig. (4.14b): Use of multiple layers.

In this way, one drawing becomes more than one, in the sense that each layer can be accessed and plotted alone by itself, as illustrated in figures (4.14a & 4.14b).

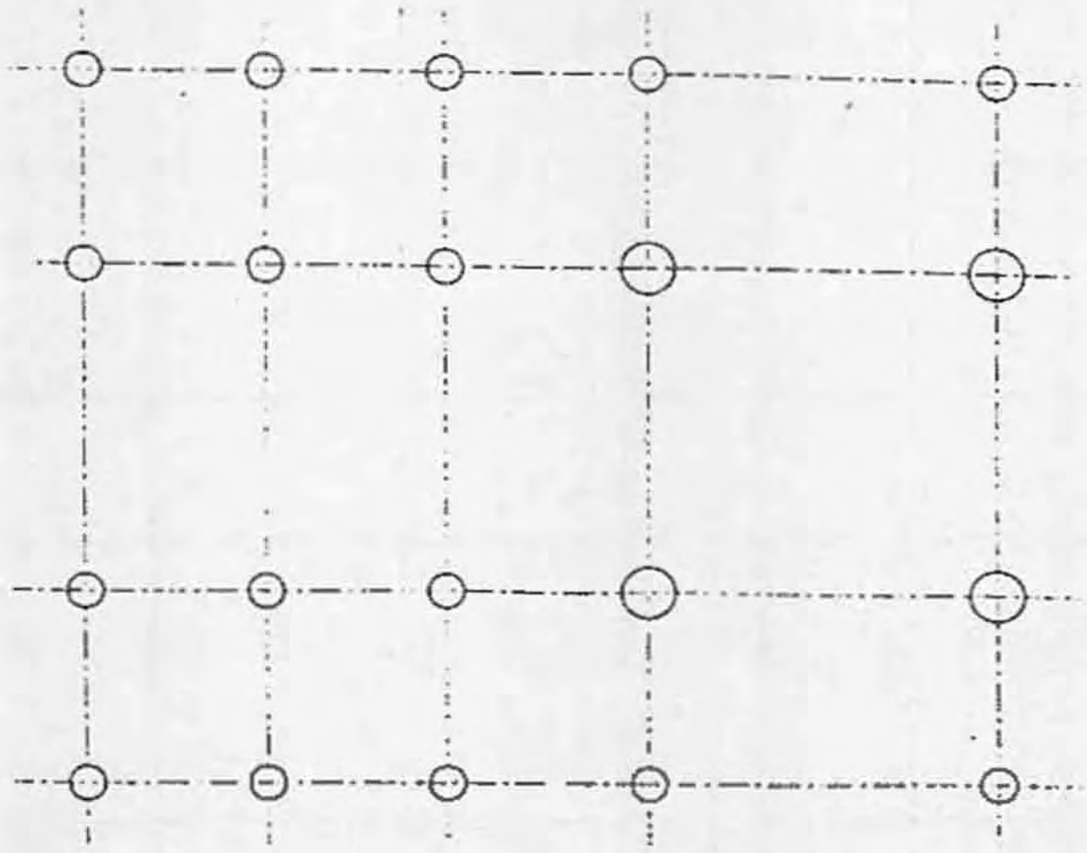
A firm that is specialized in designing the same building type quite frequently, or has a client who builds many similar buildings, such as a motel or a restaurant chain for example, may find the repetition feature very useful.

For example, if the set of drawings for one building is very similar to those needed for another one, changes can be made to the drawings in order to produce those needed for the new design very quickly and easily. The first set of drawings will be copied in the computer and then the originals will be removed. Changes can be made to the copies so as to prevent the destruction of the originals.

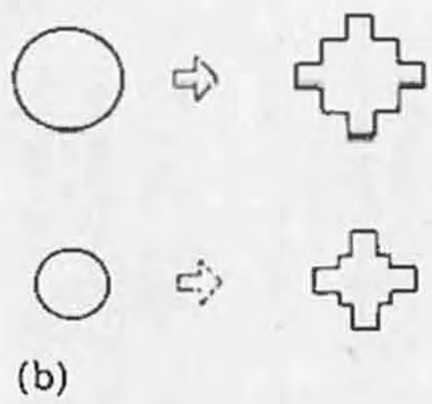
With only slightly more effort, two sets of drawings can be produced out of one. Accordingly, if several floors in the same building are very similar, the same process can be used. One floor can be created, then several copies of that floor can be altered to create other floors.

Another use of layers is illustrated in figure (4.15), which shows the possibility of visually comparing different design alternatives.

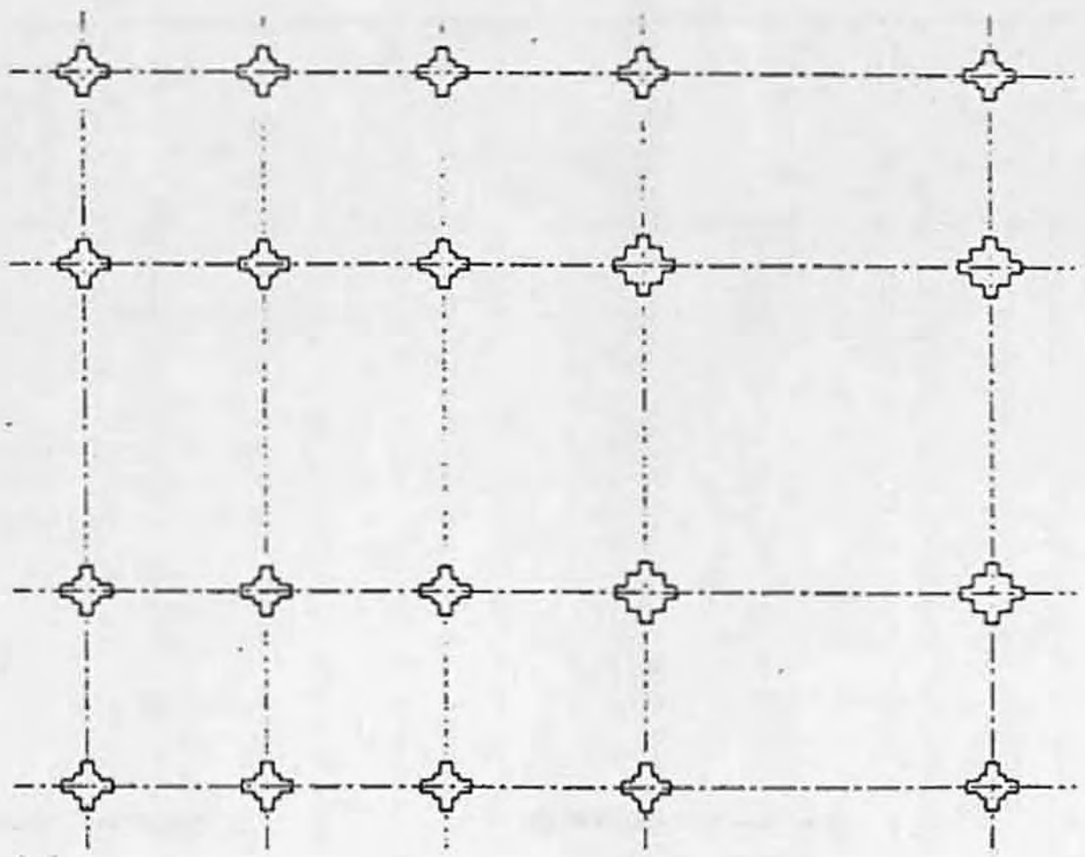
Another advantage of computer drafting is allowing a consistent drafting style. A firm can design standards for arrow heads, section and detail markers, borders and title blocks, material symbols, dimension style, and so on, and include all these features in the database. Every working drawing can then have a consistent style, including the lettering.



(a)



(b)



(c)

Fig. (4.15): Different shapes for columns could be evaluated.

Standard graphic symbols can be stored and easily retrieved and inserted into a drawing by the use of a "menu". A menu can also hold common commands that can be executed by choosing them from the menu instead of typing them on a keyboard.

In addition to all that has been mentioned so far under the advantages of computer drafting is the ease by which changes can be made on a drawing. On-going changes can be taken care of easily, whether as minor as moving the position of a door, or as major as the re-designing of a building's plan. The information in the database may be used as input to a variety of other programs, such as a piping or structural analysis, or the generation of a bill of materials. This feature points out towards the importance of developing a totally integrated system, which is the main goal of this thesis.

4.1.3.2. Specifications:

The architectural firm interested in automating and managing its information for specification writing can do that through the use of "database management system softwares". Specifications can be produced using the report generation facilities of database management system. Elimination of hand typing in this way can cut many days from the completion time of a project, and this is extremely important when a tight deadline must be met. Further more, sorting and formatting facilities allow a wide variety of different specification and ordering documents to be generated economically from the same database, and enable cost analysis to be broken down in numerous different ways with little effort.

Additionally, the computer can conserve the specification writer's time by assembling packages of text into semi-finished form for final editing and approval (Gero, 1977).

4.1.4. Business and Management Applications:

The applications grouped in this category are not uniquely architecturally oriented. That is, the issues covered here are viable in other business and commercial fields as well. Another common feature of these application programs is their avoidance of any design issues. These programs do not attempt to help the architect with his design work, nor provide any evaluation of his design. Instead, they aid him in managing both, the architectural office, and the projects in that office.

4.1.4.1. Office Accounting:

Computer programs to automate office accounting have been available for many years, and are used successfully by many different businesses and professions. Separate application programs aid in controlling and managing payrolls, check writing, journal entries, invoice preparation, and payable and receivable accounts.

4.1.4.2. Comprehensive Financial Management:

This category of programs expands upon the office accounting programs mentioned above, by facilitating many types of decision making faced by the firm's managers. An executive's judgment is often based on intuition as well as an educational background. Computer application programs have been developed which use the financial database of the accounting programs to alleviate the guesswork, and provide data to back up certain decisions. This database is, in essence, a record of the firm's years of production experience. With appropriate programs, this information can be organized to show past performance, and act as a guide for managing current projects and planning future ones.

Programs today are expanding to become more valuable tools than just the automated book-keeping systems available in the past. Comprehensive programs will help control the financial position of the firm, as well as forecast and plan budgeting and project management decisions. Besides performing standard accounting functions, these comprehensive financial packages organize and present information to help answer the following types of questions: the profitability of proposed projects, determining if any types of projects are consistently profitable while others signal a loss, on what size projects can the firm make the most money, on which clients is the firm consistently losing or making money, and other similar long-range planning questions.

Typical weekly input to a program of this type might include all employee hours spent on each project, percentage completed of each project, cash disbursements and receipts, and journal entries and invoices. The database must continually be updated to include new projects, salary changes, employee turnover, and so forth. With this information, a wide range of reports can be produced, including payrolls, project details, progress and summary reports, summary of direct expenses, income expenses, office earnings reports, expense analysis, time analysis, and many more.

In general, reports fall into one of three categories: for accounting purposes, to provide details on each project, or to give project progress.

There are many advantages to using a computerized financial management system. The budget for each project is input when a new job is added to the database. In this way, the project manager is able to monitor the direct job expenses against the proposed budget, and have a solid basis for making management decisions. The bookkeeper is freed from laborious

journal keeping, and concentrate on improving the cash flow by closely monitoring bills sent out and bills paid.

The amount of information available, and the types of reports produced by a computerized system are much more extensive than are generally available with manual accounting methods. The firm's new problem then, might be to effectively make use of all this new information.

4.1.4.3. Project and Personal Scheduling, Construction Management:

Most computerized project management technology is based on CPM (Critical Path Method), and PERT (Project Evaluation and Review Technique), two techniques formulated in the late 1950's. The basis of both methods is a network representing each task that must be done to complete a project, and the length of time needed to complete the task. Computerized versions of these techniques have been widely available for many years (Leighton, 1984).

CPM can be used in the architectural office to effectively schedule activities on many levels, beginning with one project's design development and production sequence, upto the scheduling of as many as a hundred projects that the office may be working on simultaneously. Personnel within a firm can also be charted by the CPM, to help allocate employees to varying projects, and avoid personnel overflows and/or understaffing. Another important use is, timing material, and parts orders correctly during construction to ensure that the equipment and material are on the site when it is needed.

Several integrated computer systems are able to monitor and control the design process. In these systems, CPM or PERT is only one link in a package of programs which augment the

network technique. Programs which run before the CPM to assure quality input, are necessary to obtain the best results from the CPM. The output obtained is then plotted, graphed and analyzed in a variety of formats, to make it more worth while for managers to use and adhere to. In short, their approach is to enhance basic CPM techniques, to make them more accurate, workable, and feasible for architectural firms to utilize.

4.2. Different Computer-Aided Design Tools:

Existing computer-aided design tools can be distinguished into four categories:

Representation: or computer-aided drafting is firmly restricted to the documentation or the detailed design phases as a representation medium of design decisions that have already been made. The exception, sometimes is the occasional computer generated perspective or axonometric drawing of design proposals, to communicate ideas or to confirm expectations.

Simulation: The computer is used to predict the consequences of a decision proposal in some performance area. In this category, the decision making lies outside the computer model.

Generation: The computer is used to explore the consequences of applying an ordered set of design rules. There is no separate performance; all designs that conform to these rules are equally acceptable.

Optimization: Is a tool whereby the generation mechanism is accompanied by an ability to simulate performance in a specified criterion in order to identify solutions that could offer the best solution in that criterion.

4.2.1. Representation or Computer-Aided Drafting:

The introduction of the computer as a representation medium, started in the 1960's with the seminal work of Ivan Sutherland on the sketchpad program (Sutherland, 1963).

In parallel with the early work of Sutherland and others at MIT, IBM developed an elaborate system known as DAC-1 (Design Augmented by Computer), for use by General Motors in automobile design. The system was made public at the 1964 Fall Joint Computer Conference, and was to be the forerunner of many interactive computer aided design systems installed by automobile and aerospace firms by the end of the 1960's (Prince, 1971). A third important early system was developed at around the same time by Itek Laboratories for lens design work. Since then, computer aided design has been increasingly and widely employed in various aspects of mechanical, civil, electrical and industrial engineering (GOTT, 1973).

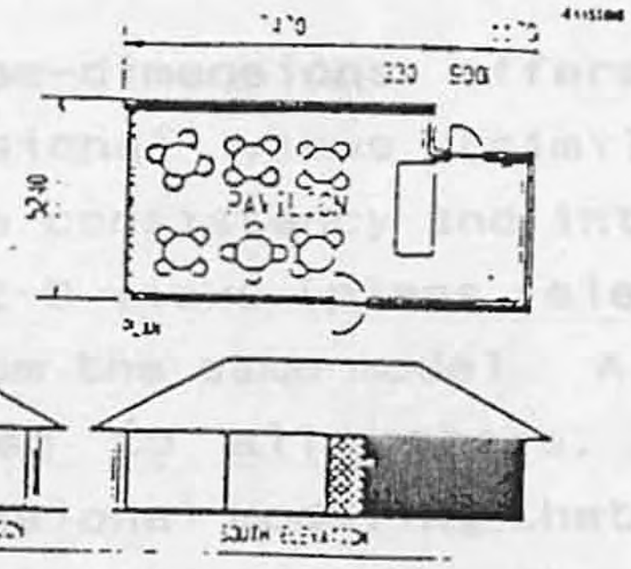
Some very ambitious experimental interactive graphics computer-aided design systems were implemented during the 1960's, for example Souder and Clark's COPLANNER (Souder and Clark, 1964), and Negroponte's URBAN5 (Negroponte and Groisser, 1970). In 1968, the Yale school of architecture sponsored a major conference on computer graphics in architecture and design (Milne, 1969). However, the first practical applications, which concentrated mostly on the areas of structural and mechanical calculations, cost estimation and economic analysis, and specification production, were rather mundane in character and did not involve any elaborate graphics (Harper, 1968, and Negroponte, 1975). The first installations of interactive computer graphics systems in architectural offices began to appear in the early 1970's.

Recently, most of what is called computer-aided design in architectural practice refers to the representational use of computer through computer graphic implementing the fundamentals established in the early 1970's. Elaborate or simple systems are simply used to document drawings. A drawing is a "model" of the artifact or scene it depicts. In normal drafting, the model physically consists of particles of ink or graphite on paper, we interpret the particles as paths of lines, polygons, and surfaces, but there is no internal structure that establishes such entities.

In computer graphics, there has to be an internal structure, if only to relate a pattern of bits stored in the computer to an image on the computer screen. Most graphics packages today use structure only to accommodate computer storage and performance concerns. The structure does not reflect relations of the elements to themselves and to other drawings from the design point of view. In architecture, the act of drawing is intimately linked with the art of design. The process of designing takes place in parallel with the process of drawing, erasing, modifying, and redrawing (Akin, 1979, and Lawson, 1982).

Computer graphics applications in architecture can be distinguished into three main categories: 2-dimensional, 2.5-dimensional modeling, as well as 3-dimensional or wireframe, solids, and shaded area modeling, as illustrated in figure (4.16).

Two-dimensional drawings can be modeled as: i) a set of points or pixles mapped onto a screen, ii) a set of lines defined in the coordinate system (continuous, dashed, dotted) as well as attributes (properties like color, thickness or type, iii) as a set of objects such as elements, components, or geometric primitives. Objects can be assigned attributes at a higher level of detail.



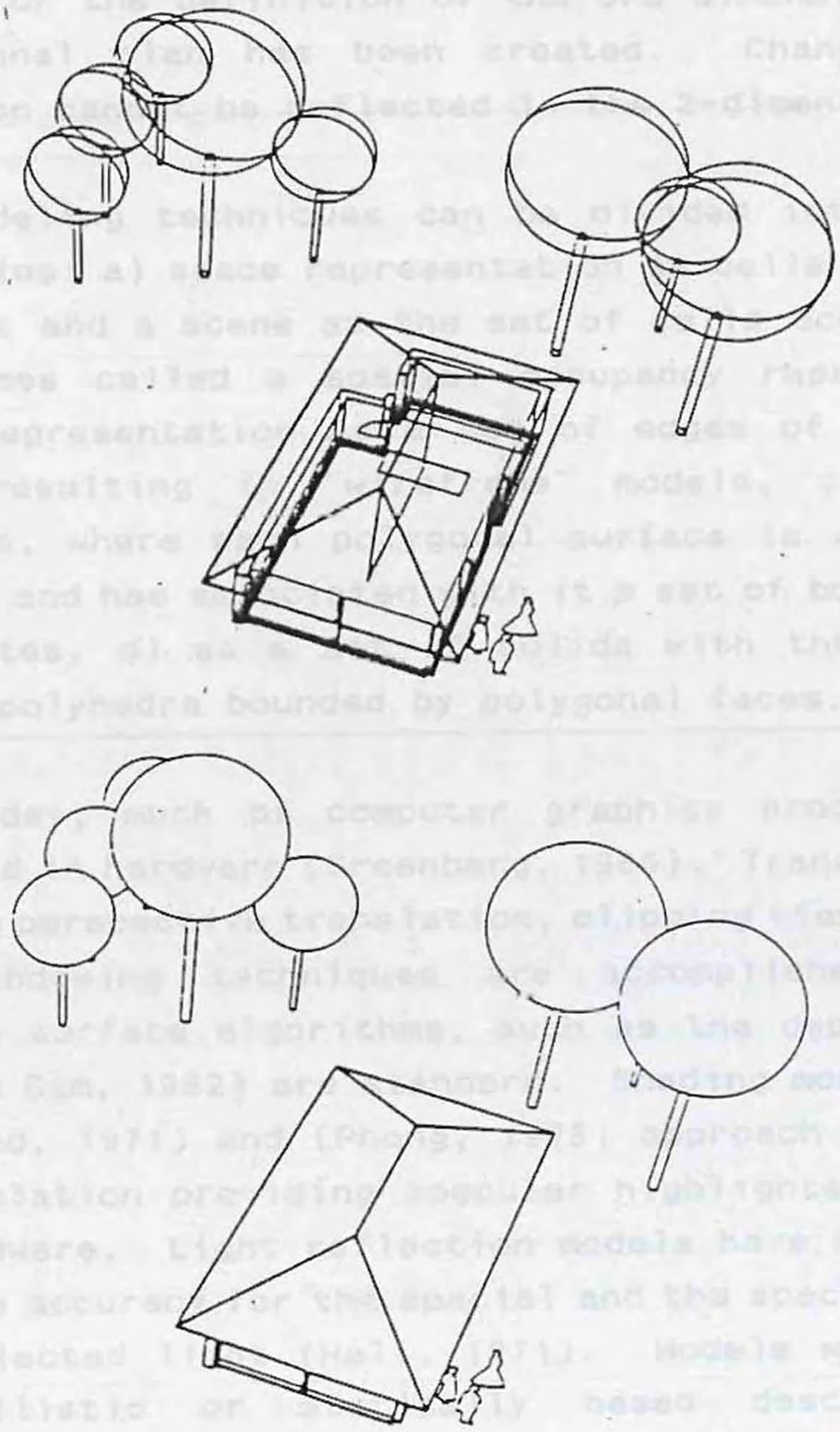


Fig. (4.16): A 2-D, wireframe, and solid modeling representation.

Modeling in three-dimensions offers the ability to generate three dimensional views (similar to the final design), as well as the consistency and integrity that can be obtained when all the 2-D views (plans, elevations, sections, etc.) are generated from the same model. A change in one view can then be propagated to all others. A hybrid of 2-dimensional and 3-dimensional modeling that has been generated allows for the definition of the 3rd dimension after the 2-dimensional plan has been created. Changes in the 3rd dimension cannot be reflected in the 2-dimensional model.

Modeling techniques can be divided into the following categories: a) space representation as cells (voxels) in a 3-D matrix and a scene as the set of cells occupied by matter (sometimes called a spatial occupancy representation), b) solid representation as a set of edges of their enclosing plans resulting in "wireframe" models, c) as a set of surfaces, where each polygonal surface is recognized as an entity, and has associated with it a set of bounding edges and attributes, d) as a set of solids with the recognition of closed polyhedra bounded by polygonal faces.

Today, much of computer graphics processing has been embedded in hardware (Greenberg, 1985). Transformations which perform perspective translation, clipping viewpoints, mapping, and windowing techniques are accomplished in hardware. Visible surface algorithms, such as the depth-buffer (Foley and Van Dam, 1982) are standard. Shading models, such as the (Gouraud, 1971) and (Phong, 1975) approach which use linear interpolation providing specular highlights, are also built in hardware. Light reflection models have been developed to provide accuracy for the spacial and the spectral distribution of reflected light (Hall, 1971). Models which depend on a probabilistic or statically based description of the microfacets of a surface, Blinn model (Blinn, 1977), and Cook model (Cook and Torrance, 1982) provide reasonable results in

modeling surface descriptions such as roughness, size, and orientation of the microfacets.

In 1980, Whitted introduced a technique called "ray tracing" that accounts for the effect of intra-environment on the models image. Another method was developed under the Program of Computer Graphics at Cornell University, Ithaca, called the radiosity method (Goral et al, 1984).

A detailed analysis of computer graphics modeling can be found in Mitchell's; Computer Aided Architectural Design (1977), and Radford's and Stevens'; CAAD Made Easy (1987).

4.2.2. Simulation:

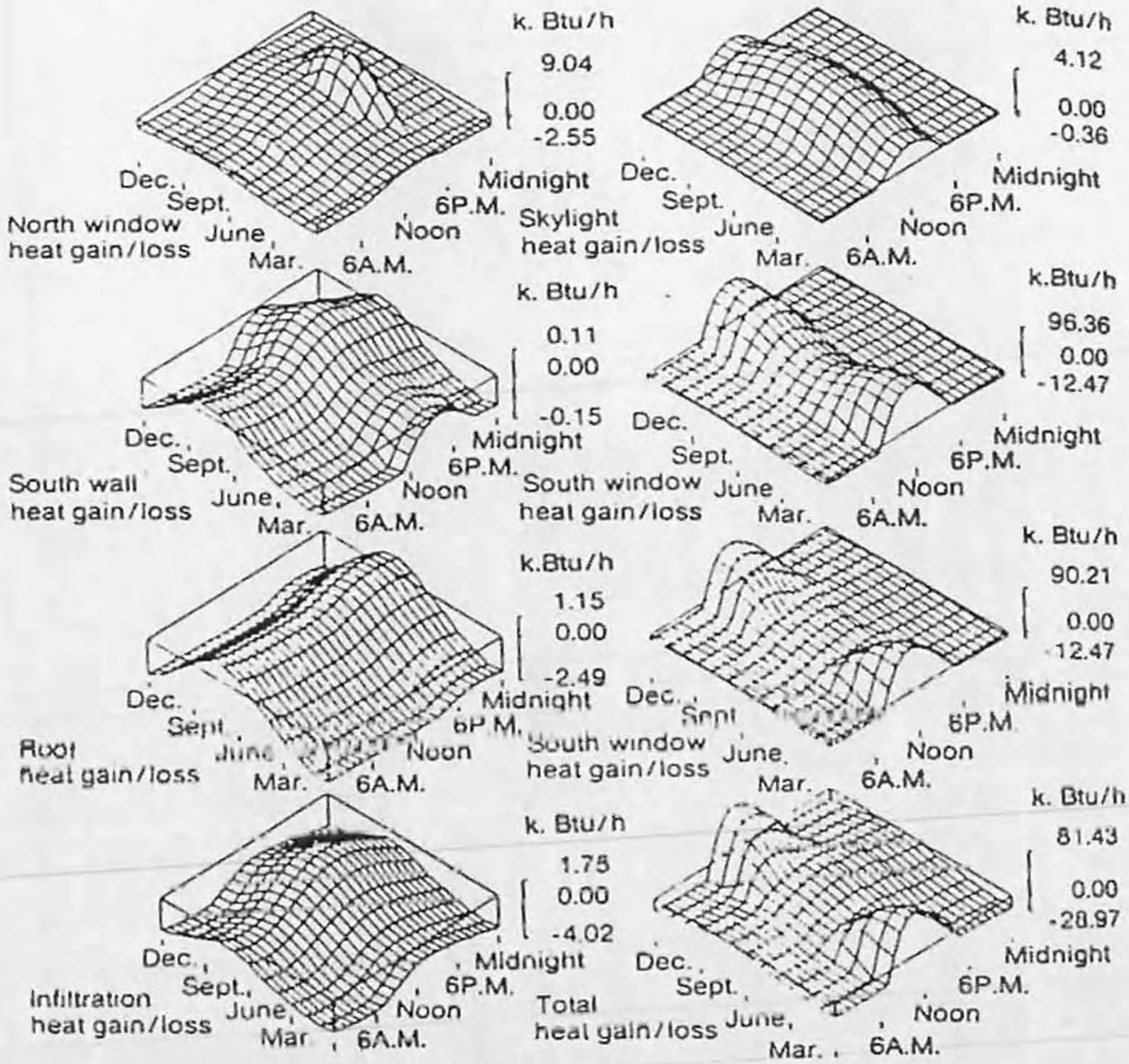
Most computer-aided design outside the realm of computer-aided drafting are based on "simulation models", given a description of the relevant characteristics of a design proposal they will simulate the behavior of the design artifact in order to predict its performance.

Simulation models can be either "static" or "dynamic", based on whether time is a variable in the simulation. Most simulation models are based on the static behavior of buildings. Although available dynamic models can simulate reality in such a way that would have been impossible to do manually, available simulation models are based predominantly on the engineering subtasks present in the design process that have been quantified outside the research world of architecture (Radford and Stevens, 1987). Much of the early work with computers in architecture dealt with the exploration of the quantitative aspects of design; the investigation of the structural, economic, and environmental performance of a building (Kalisperis, 1988).

The results from a simulation program, shown in figure (4.17), are usually presented in multi-valued form (internal temperatures at different times, or heat flows through different parts of the building's fabric for a thermal program) with no direct indication of the goodness or badness of the results, which is left completely to the users own evaluation criteria. In order to improve or develop solutions, we need to understand how to improve performance by modifying one or more of the design variables.

In the process of "appraisal" (Maver, 1977; Markus et al., 1972), a first approximation is nominated by the designer, and its performance is predicted. It is up to the architect then, whether modification of the solution is necessary, and if so, what form that modification should take. The success of this process depends in the first place, on the designer's ability to formulate a useful hypothesis by which he can modify the solution, and that demands either a theoretical knowledge of the field, or some clear trend emerging from a series of simulations, or both. If a modified solution is very similar to the original (perhaps changing only one item at a time in a parametric study), it is easy to understand the reason for changes in performance, but the range of design options explored is small. On the other hand, if the modified solution is radically different than the original, it becomes impossible to attribute variations in performance to individual design changes. Recently, certain simulation models have been improved, and evaluative criteria for results can be defined as part of the model's knowledge base.

The majority of simulation programs concern the building's thermal performance and structural analysis. Many different energy analysis programs are available, the best known including DOE-2 (Hunn, 1984), BLAST (Hittle, 1979), ESP (Clark, 1978) and NBSLD (Kusuda, 1974).



Qualitative and quantitative information on heat gain and loss through elements of a northern hemisphere building at the early stages of design is provided by these three-dimensional graphs from the SOLAR5 simulation program. The left-hand column shows heat transfer through a north window, a south wall (notice the 4-hour time lag), the roof (notice the damping effect of high mass construction), and infiltration. The right-hand column shows the results for a skylight (its "heat mountain" shape shows its poor passive solar performance), a south window (its "saddle" shape demonstrates good passive solar performance because it gains more in winter than in summer), a south window with overhang (eliminating much of the summer overheating at the cost of only a little winter gain), and finally the total heat gain and loss (internal thermal storage will prevent winter passive gain from temporarily overheating occupied spaces). *(Milne, 1982; reprinted by permission from the Proceedings of the CAD 82 Conference, published by Butterworth and Co. Ltd. and sponsored by the journal Computer-Aided Design.)*

Fig. (4.17): An output from the SOLAR5 simulation program.

Specialized energy analysis simulation models address specific areas, such as passive solar energy design, SOLAR5 (Milne, 1982, 1984). Models are available for calculation of the distribution of daylight as well as distribution from an artificial lighting scheme, and present the results as numbers, or as shades of color of greyness on a drawing of the room (Miller, Ngai, and Miller, 1984).

In the early 1960's, a significant development in the computerization of structural computations was the emergence of large scale, comprehensive structural computation systems based upon the "problem-oriented language" concept. The pioneering system of this type, STRESS (Structural Engineering Systems Solver) was developed in the early 1960's (Fenves, 1965). A direct successor to STRESS was STRUDL (Structural Design Language) which was developed as a subsystem of ICES (Integrated Civil Engineering System) (Logcher, 1970). A version utilizing an interactive graphics interface has also been implemented (Biggs, 1973).

Other areas of building performance simulation have been explored as well. Applied knowledge and quantification of acoustics allowed for the development of computer programs that predict sound level propagation, as well as acoustical characteristics of the indoor environment or the room acoustics. Basic formulas which are suitable for performing acoustic analysis of proposed designs, have been implemented in computer programs (Boyer and Degelman, 1966; Degelman, 1968; Simon and Rien, 1968; Campion, 1970; Hawkes and Stibbs, 1969, and Powell, 1973).

In addition, simulation programs have been developed and used for the generation of cast data (Paterson, 1974), bidding (Winslow, 1972), quantity surveying and cost estimation (Britch, 1963; Dent, 1964; Smart, 1966; Monk and Dunstone, 1968), and project management (Krawczyck, 1984).

Land use information models were introduced by Paul and Landini (1975) with the LUMIS systems. A number of general computer mapping programs which are suitable for land form manipulation have been reported in the literature at an urban and regional scale (Porter, 1970; and Kamnitzer & Hoffman, 1970) such as SYMAP and SYMVU, shown in figure (4.18) (Dougenik and Sheehan, 1976), CASAT (Ward et al, 1970), and OVERLAY (Dudnik, 1971). All of the engineering simulation models were developed independently.

The development approach was one of discrete operation rather than as a part of the overall process. Models were created outside of a holistic view of architectural design. The computer programs that were developed did not accommodate task tendencies and their interrelations as subtasks of the overall design process (Kalisperis, 1988).

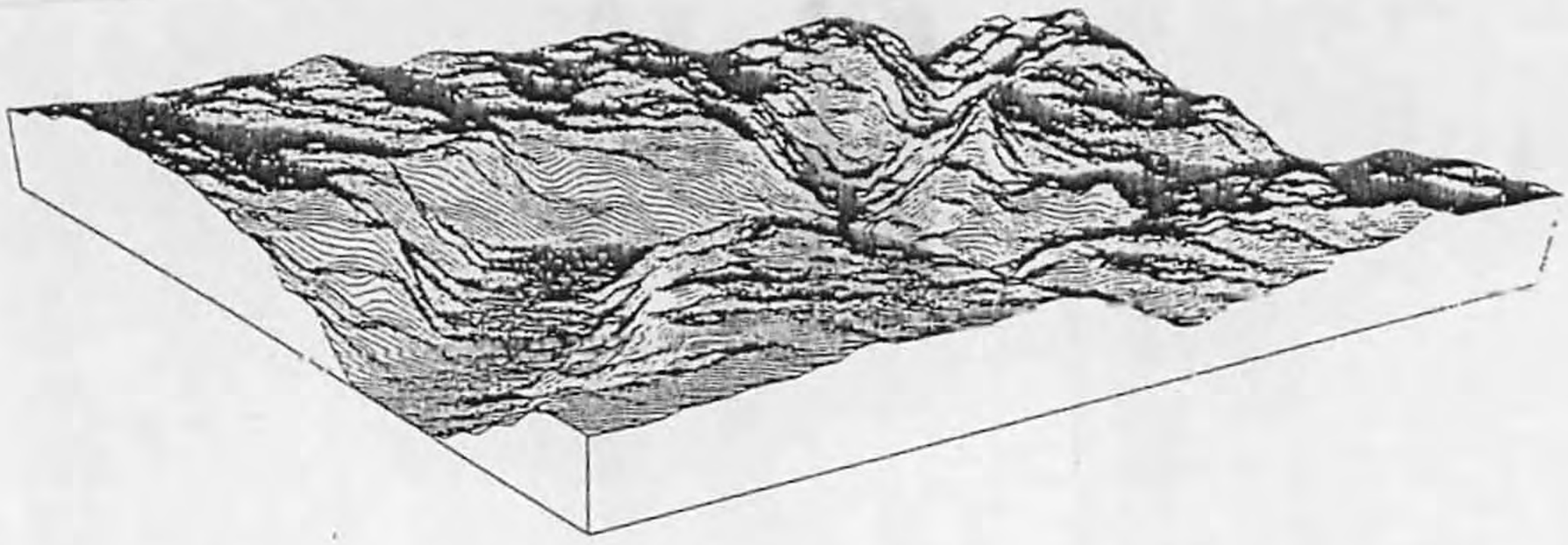


Fig. (4.18): The output from SYMAP and SYMVU programs.

4.2.3. Generation:

In a "generative model", the model itself generates a design solution according to some prescribed rules. Much of the early research on using computers in architecture sought the automatic synthesis of a plan for a building from some basis of desired functions or facilities and the relationship between them. This process is known as "layout planning" or "facilities planning". The designer specifies the areas required by each item, and the degree of importance of the relationship on an ordinal scale between each pair of items. The facility areas are then fitted directly into available areas or zones within a given outline or building parameter. The resulting diagrammatic layout is then linked to a wider appraisal (evaluation) package that measures cost and environmental performance as well as circulation.

The two basic approaches to facilities planning are: a) the familiar "association matrix and cluster diagram", and b) the use of the location/allocation optimization algorithm to assign facilities to spaces according to a "bestfit" measure of association.

The output from the simplest generative model is a cluster diagram, which is based on association (or proximity) values and allows those clusters to be translated into a rectilinear layout. Then the layout can be tested according to criteria for adjacency, orientation, and access. SPACES2 is a typical application of such a model (Radford et al, 1987). Certain models fit into the optimization category since they optimize (by a heuristic search method) a layout of clusters to minimize a mathematical objective function.

Optimization terminates with a cluster diagram, not with a plan, as will be discussed later under the optimization

category, deriving the plan from the diagram depends entirely on the architect's skill.

The earliest and most common mechanism for laying out spaces directly, is an algorithm that allocates a list of required facilities to a set of available locations in a way that minimizes the total "circulation cost", by using what is known as a "quadratic assignment formulation" of the problem (Mitchell, 1977). A facilities planning program typically operates in either; section or plan, assigning facilities to floors of a building, "stacking", or assigning facilities to zones or areas on a floor, "blocking".

Application programs which are based on this method, have been introduced in the literature, and include TOPAZ (Crawford, Mitchell and Booth, 1980): figure (4.19) shows a TOPAZ output, IMAGE (Weinzapfel and Handel, 1975), CRAFT (Willoughby, 1970), and General Space Planning (Eastman, 1972) are other examples of such programs.

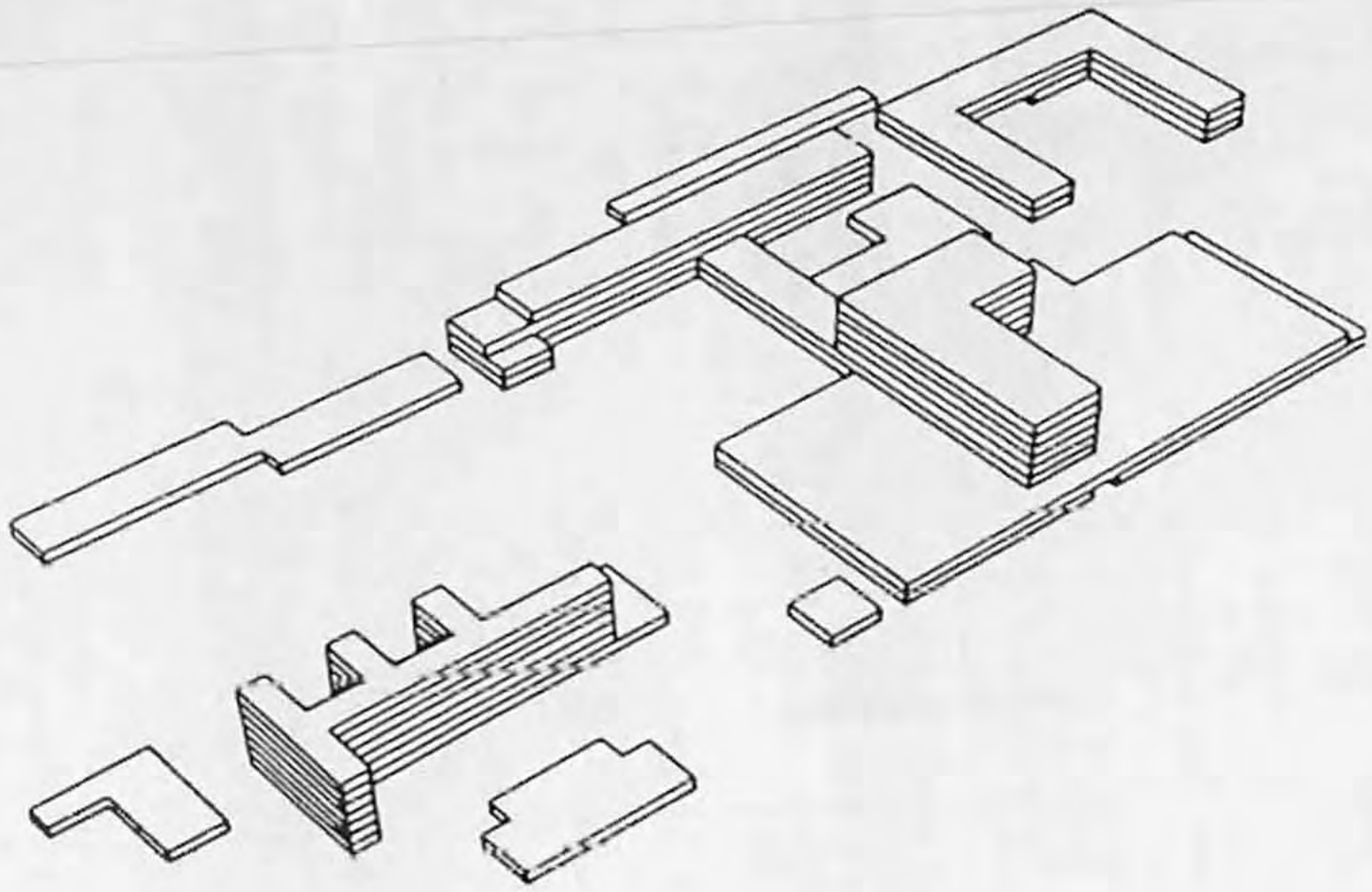
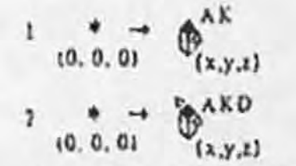


Fig. (4.19): An isometric output from the program TOPAZ.

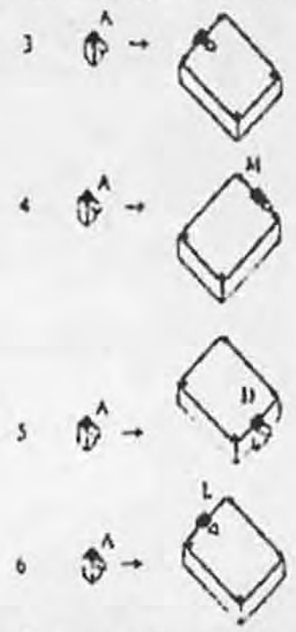
Shape grammars are concerned with the generation of form through the application of defined rules which constitute a grammar (analogous to a grammar in human language) and which can be used to express the way that the elements of a design are composed in a certain style. George Stiny, James Gips, Ullrich Flemming, and William Mitchell are among the first who have investigated the role of "shape Grammars" in architectural design. They proved that it is possible to identify grammars that appear to define the design of certain Palladian villas (Stiny, and Mitchell, 1978), Frank Lloyd Wright's Prairie Houses (fig. 4.20) (Koning and Eizenberg, 1981), Miesian architecture as in figure (4.21) (Schmitt, 1988), and numerous other examples.

A shape grammar formulation allows for algorithms to be defined directly in terms of shapes, and is one of the most fascinating areas of recent research in architecture. Stiny, in 1980, pointed out that there are four components to a shape grammar that apply sequentially: i) a finite set of shapes, ii) a finite set of symbols, which are used to label the shapes, iii) a finite set of shape rules of the form "shape A becomes shape B", and iv) an initial labeled shape. In most applications and reasoning for the shape grammars, the language has been drawn from an existing and highly regarded body of work vocabulary.

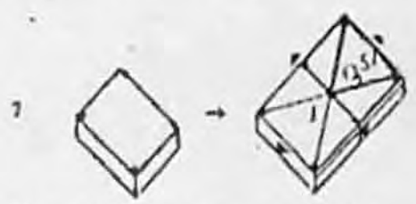
Existing applications render shape grammars as a tool of architectural criticism, enabling an understanding of something about the history of architecture. In order to develop designs based on the concept of "shape grammars", it is necessary to develop an architectural language.
Analogies between linguistic paradigms and the architectural design process, are an emerging field of architectural research (Coyne, 1986, and Goel, 1986). Laseau proposes a graphic language that has grammatical rules comparable to those of verbal language (Laseau, 1980).



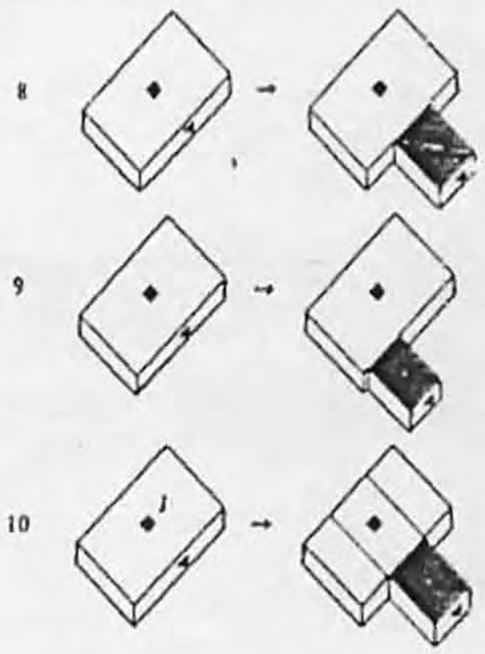
Locating the fireplace

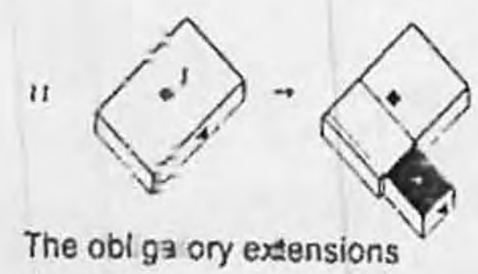


Adding a living zone

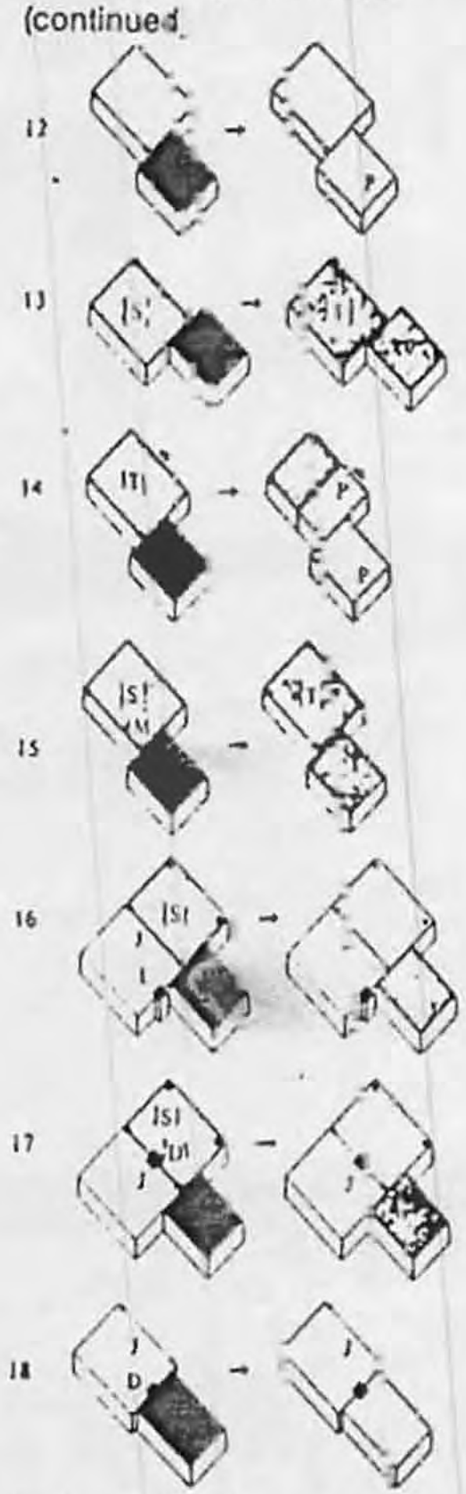


Completing the core unit





The obligatory extensions (continued)



Assigning function zones

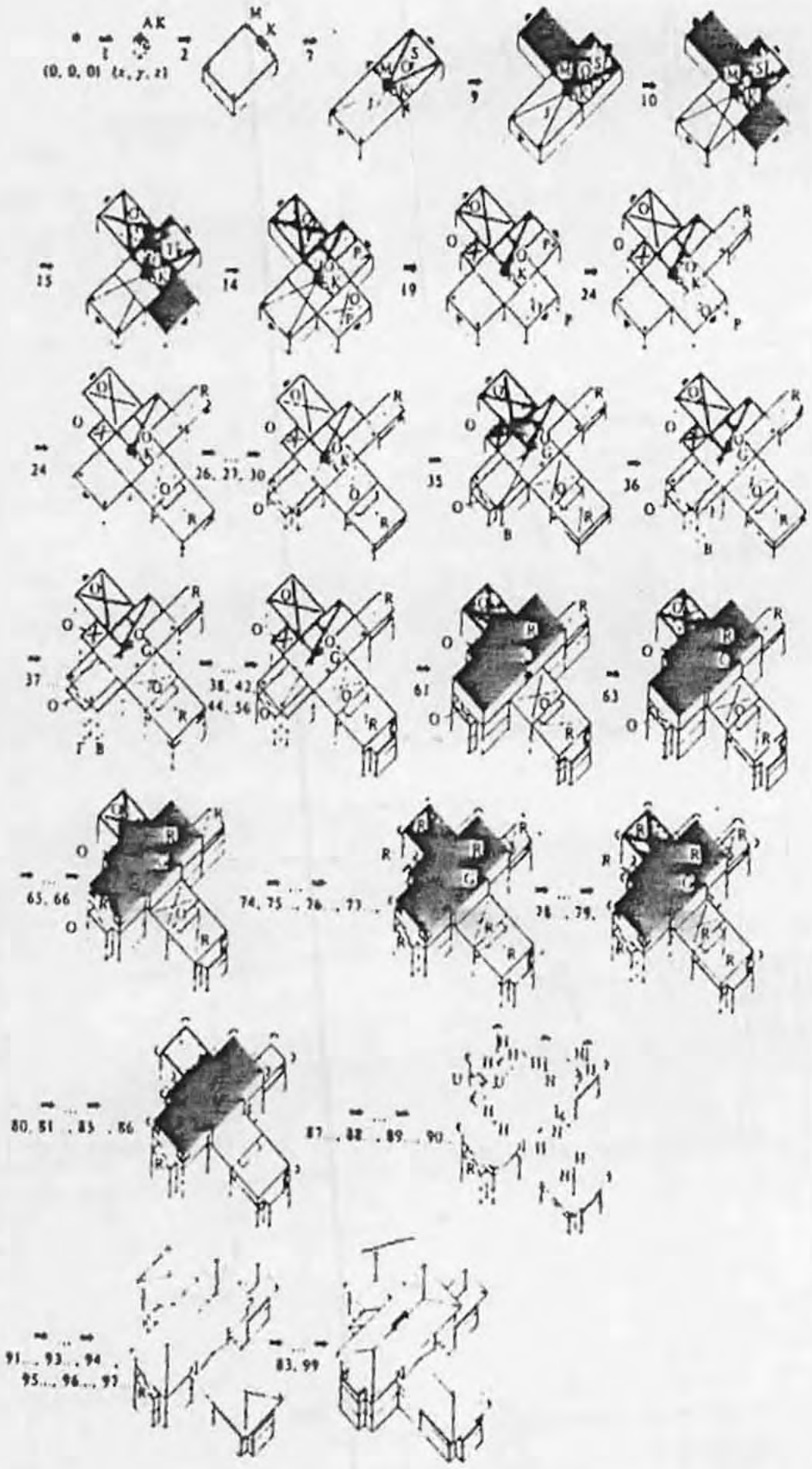
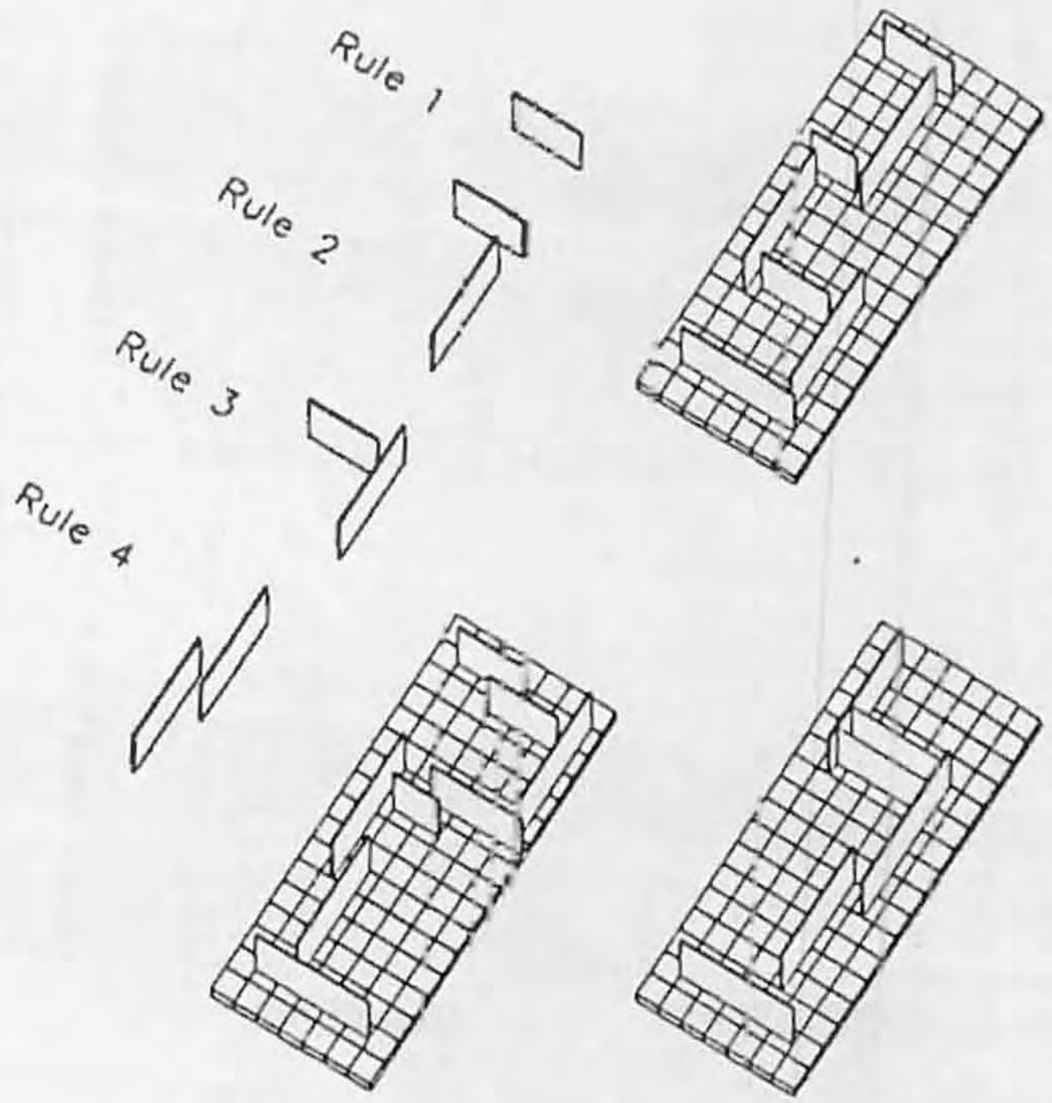
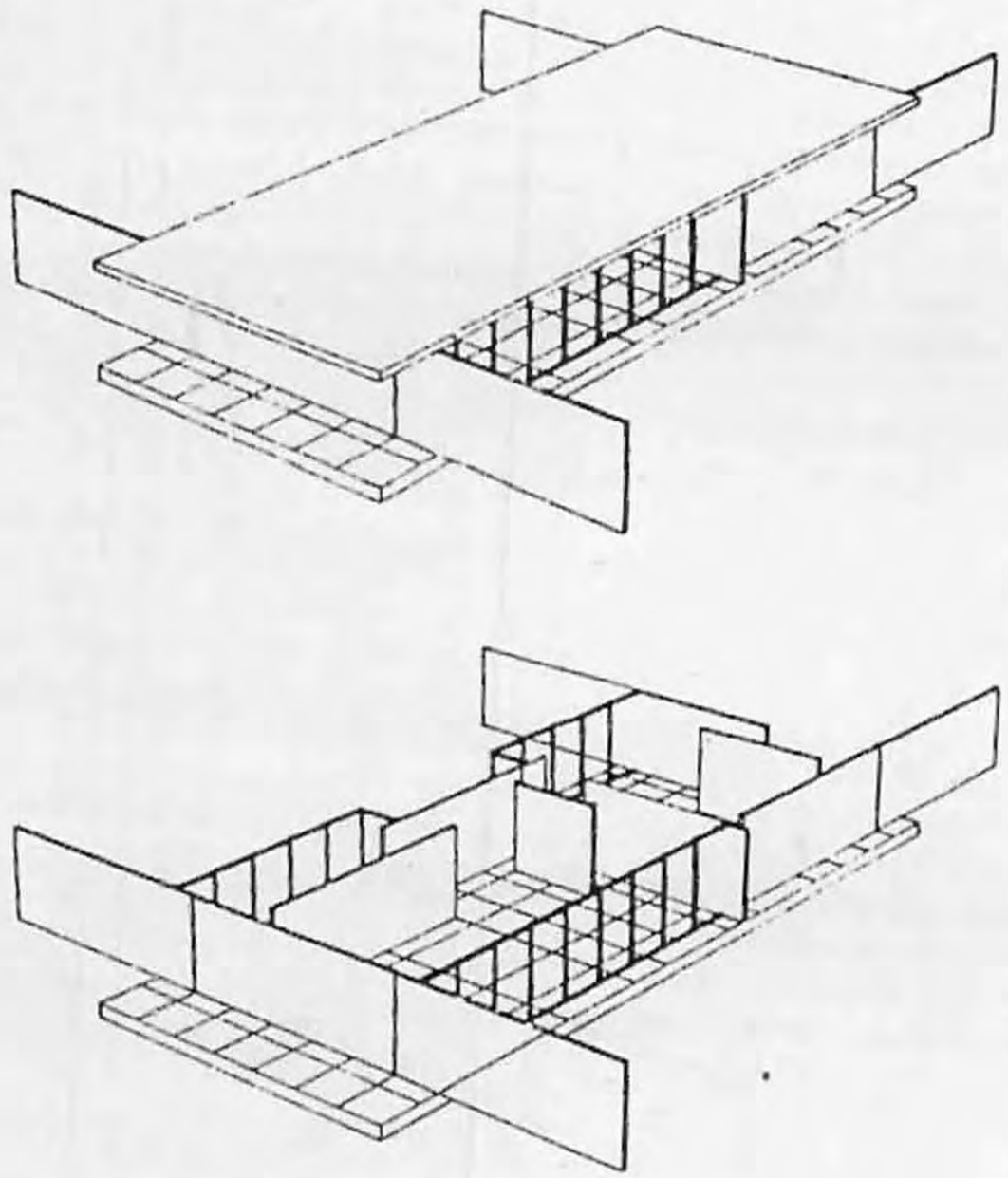


Fig. (4.20): An extract from the transformation rules developed for the composition of designs in the style of Frank Lloyd Wright's Prairie houses.





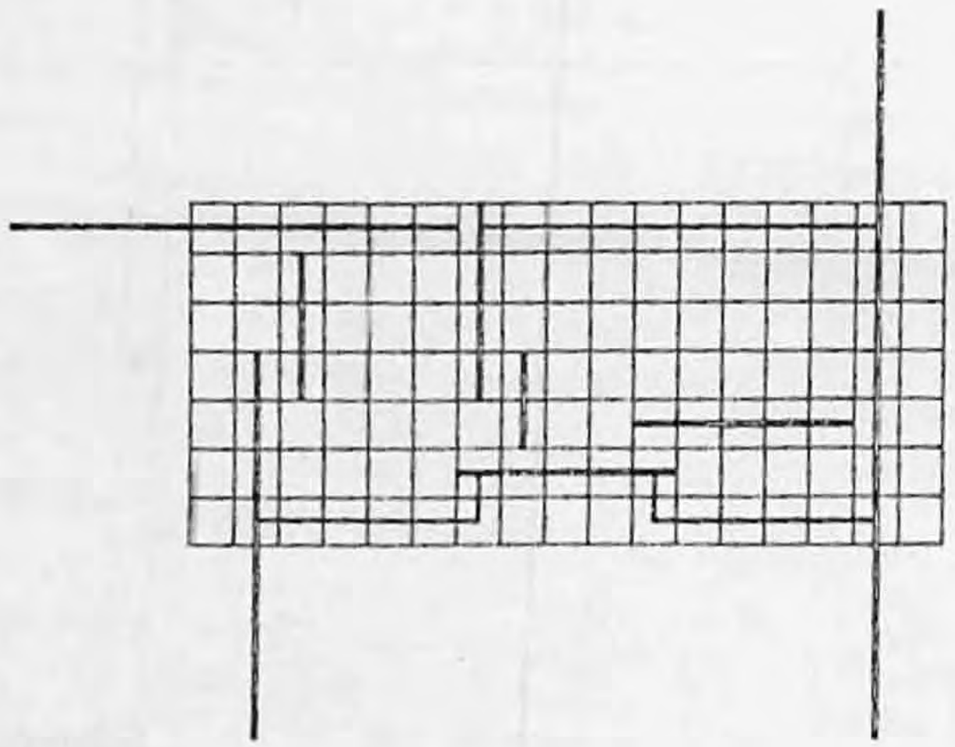


Fig. (4.21): Designing according to panel architecture rules. The resulting building has Miesian character.

The nouns, verbs and modifiers in verbal language represent identities, relationships between nouns, and qualifications and quantifications of identities respectively. Graphic language being inherently explicit, needs a grammar, vocabulary, elements and operations that establish identities, relationships and modifiers.

Graphic language represents vocabulary as symbols composed of point and line primitives, which constitute its elements. It represents relations with proximity and the thickness of connecting lines. It represents modifiers by varying symbol size and line weight. Coyne's and Goel's work focuses on the formalization of relations between the linguistic paradigm and the design process, Thus establishing a more general and fundamental approach which is particularly interesting for computer based applications.

It is important to point out here, that the graphic language differs from the verbal language in the sense that it is both; sequential and simultaneous. Though the proposed graphic language paradigm can benefit from parallel structure in verbal language, it must not obscure the fact that it does not sufficiently represent all complexities of the design process.

Bonta argues about the need for a changing graphical language that suits the different design phases:
"The successive stages of the process are usually registered by some kind of graphic model. In the final stages of the design process, designers use highly formalized graphic languages such as those provided by descriptive geometry. But this type of representation is hardly suitable for the first stages, when designers use quick sketches and diagrams of abstracted ideas which are handled at the beginning of the design process, they must be expressed necessarily by means of rather ambiguous, loose graphic language. Such a language

would register the information exactly at the level of abstraction it has, and it would facilitate communication and cooperation among designers." (Bonta, 1972, p.6).

The need for architectural language has been pointed out by Sir John Summerson thirty three years ago. In his essay entitled "The Case for a theory of Modern Architecture", Summerson wrote:
"... one may even be justified in speaking of a missing architectural language." (Summerson, 1957).

The complexity of the architectural problem, and the difficulties in defining its problem space and its task environment, make the definition of an architectural a language improbable. Even if it were possible to define a universal architectural language, the complexity of the language render it worthless. If we take Arabic grammar as a parallel, correct application of grammatical rules ensures legal arabic prose, but does not guarantee great literature. Shape grammars may allow designing of component and consistent buildings, rather than creating great architecture.

4.2.4. Optimization:

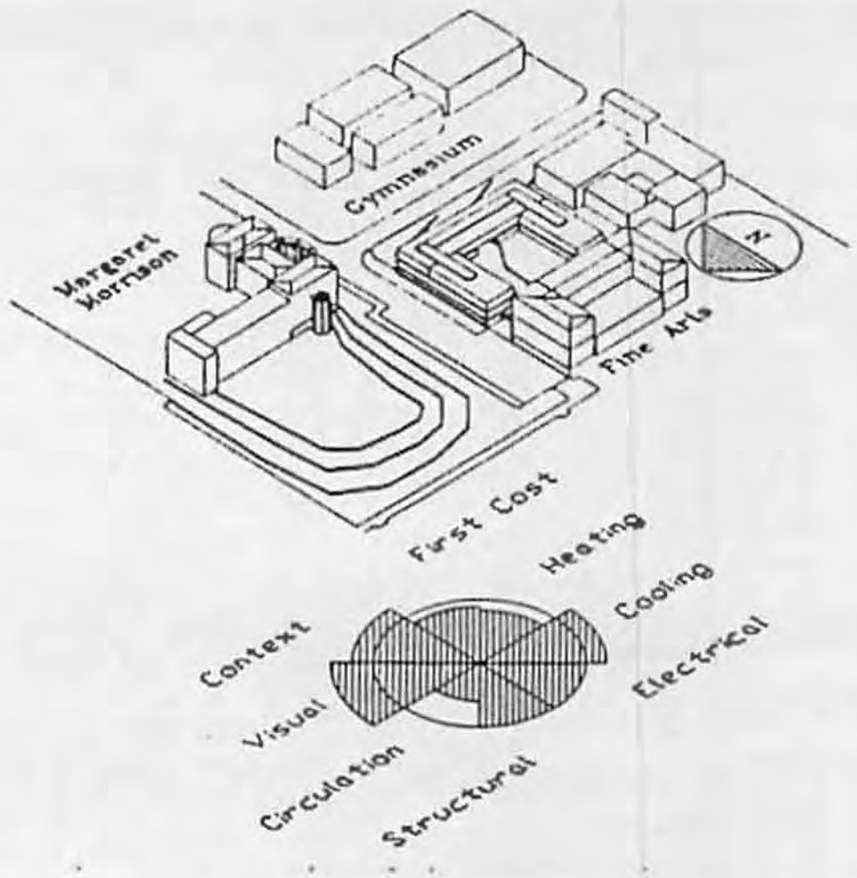
In a rigorous mathematical sense, optimization means achieving the very best solution to a certain problem. The simplest optimization method, is to generate every possible solution within the problem constraints, apply some rule for measuring their relative performance in a desired objective, and then select the best solution. This exhaustive enumerative approach is simple, but not very practical: there are over three and a half million ways (10!) of arranging just ten blocks in a row, and architectural design problems are much more complex, and have many more variables than the ten blocks.

In mathematics, there are some well established algorithms for finding an optimal solution for a certain problem, which can be expressed in a suitable mathematical form. Classical calculus, linear programming and various heuristic methods have been developed for particular problems. Although, a heuristic method does not guarantee an optimum solution, but appears to approach it, or at least to do better than the unaided human. Examples of different applications of all these techniques in architecture are found in "Computer Aided Architectural Design", by Mitchell (Mitchell, 1977).

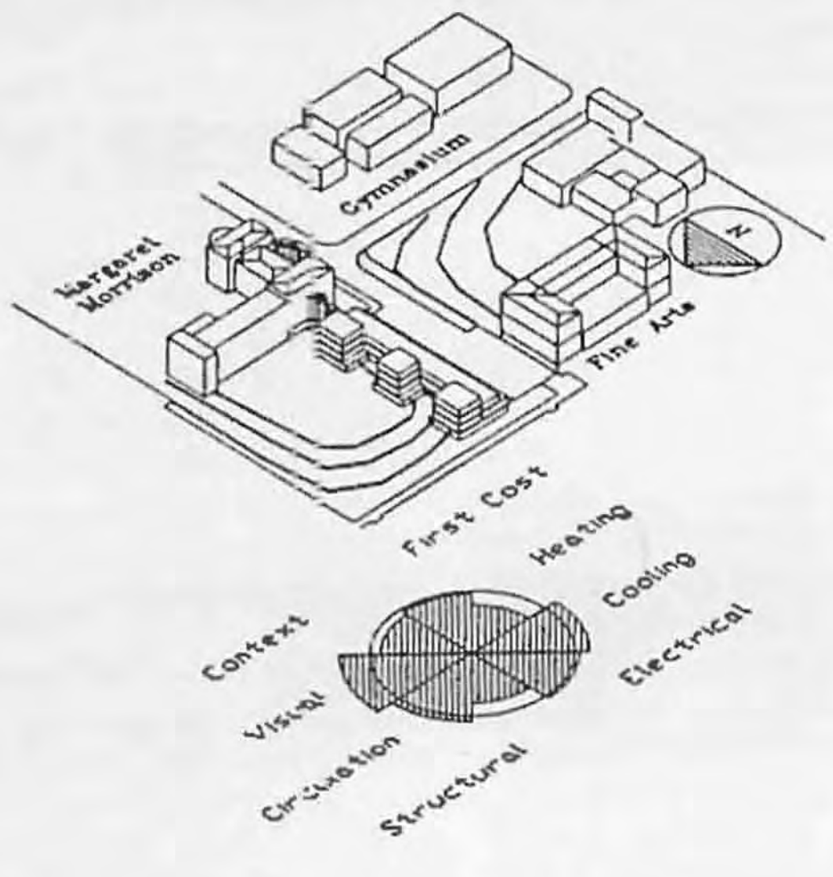
Most attempts to use optimization techniques in architecture, have been made in research studies rather than in practice (Radford and Gero, 1980), and is a reflection of the state of development in this area, and the difficulty of formulating relevant architectural design problems in the necessary mathematical form to bring to bear these optimization techniques. The major exception is in facilities planning, where algorithms not only exist, but really do get used in practice.

There are a number of examples where this has been tried, and extended to environmental design site development, building services, structures and other aspects of buildings as well. Radford and Gero (1988) present a number of different techniques for optimization such as linear programming, differential calculus, dynamic programming and pareto-optimization, based on the Pareto method (Pareto, 1971), in multi-criteria optimization, as shown in figure (4.22).

Optimization, however, is a "pretty weak model of design" (Radford, 1987, p. 209). Optimization is applicable only where the index of performance is measurable, which s the case if the purpose of building is limited to minimizing circulation, energy use, or cost.



Design alternatives for a given site and parallel display of the performance of alternatives. A U-shaped solution, forming a courtyard with the existing building (type 1, top), and an L-shaped building, using the existing College of Fine Arts as a backdrop (type 2, bottom).



Design alternatives for a given site and parallel display of the performance of the alternatives. An E-shaped proposal (type 3, top), and a linear building (type 4, bottom). Trade-offs between the different schemes become obvious.

Fig. (4.22): Multi criteria optimization for a school design problem based on simulation.

Unfortunately, optimization techniques cannot be employed in architectural design when nonquantifiables (in respect to their measure of performance) such as aesthetics, human behavior, and the building/user interface are present.

In addition, most applications of optimization have considered just one criterion in isolation, minimizing circulation, cost or energy use, but not all three. Multi-criteria optimization applications simply supply tradeoff information on the conflicts between designing for different objectives, and are restricted to the quantifiable or measurable aspects of architecture.

Architectural design is much more concerned with the tradeoffs between different design aims and goals, besides, it is concerned with finding an acceptable compromise, or a satisficing solution (Simon, 1970) rather than optimizing an objective.

In this chapter, different computer applications in architecture were discussed. Different areas of architectural computing were addressed, and these included: design applications, technical applications, production applications, and business and management applications. Different computer-aided design tools were also addressed, and these included: representation, simulation, generation, and optimization.

So far, different design methods, as well as different problem solving techniques , and different computer applications in architecture have been presented and discussed.

In the following chapter, a framework for an integrated computer-aided architectural design environment will be introduced. At first, the different components of that environment are specified, then each one of them is presented

in detail. This environment is achieved through the introduction of a certain model developed for the study, namely "ICAAD.DSS", which is an acronym for Integrated Computer-Aided Architectural Design Decision Support System.

REFERENCES FOR CHAPTER 4:

Akin, O.:

Models of Architectural Knowledge: An Information Processing Model of Design. A Ph.D. Dissertation, Carnegie Mellon University, 1979.

Beheshti, M.; and Monroy, M.:
"Requirements For Developing An Information System For Architecture." In CAAD Futures '87. Proceedings of the Second International Conference on Computer_Aided Architectural Design Futures, Eindhoven, The Netherlands, 20 22 May, 1987. Edited by Tom Maver, and Harry Wagter. New York: Elsevier, 1988.

Biggs, J. M.; Logcher, R. D.; Schumacher, B.; and Sowards, R. D.:

Interactive STRUDL Graphics. Cambridge, Massachusetts: Massachusetts Institute of Technology, Department of Civil Engineering, 1973.

Blinn, J.:

"Models of Light Reflection for Computer Synthesized Pictures." SIGGRAPH 1977 Proceedings, in Computer Graphics, vol. 2, no. 2, 1977, pp. 192-198.

Bonta, J.:

"Notes on the Semiotic Theory of Graphics Languages." In International Conference on Semiotics Proceedings. Ulm, West Germany, 1972.

Boyer, L. L.; and Degelman , L. O.:
Acoustical Design for Optimum Reverberation Time. The Architectural Engineering Department, The Pennsylvania State University, 1966.

Burden, E.:
Design Simulation. New York: McGraw-Hill, 1985.

Britch, A. L.:
"Quantity Surveying by Electronic Computers." In Architect's Journal, March 27, 1963.

Broadbent, G.:
Design in Architecture. Architecture and the Human Sciences. David Fulton Publishers, London, 1988. First published in 1973.

Campion, D.:
"Computer-Aided Acoustical Analysis." In Building, May, 1970, pp. 111-116.

Clark, J. A.:
"A Design Oriented Thermal Simulation Model." In CAD 78, edited by A. Pipes. Guildford, England: IPC Science and Technology Press, 1978.

Cook, L.; and Torrance, K. E.:
"A Reflection Model for Computer Graphics." In ACM Transactions on Graphics, vol. 1, no. 1, 1982, pp. 7-24.

Coyne, R.; and Gero, J.:
"Semantics and the Organization of Knowledge in Design." In Design Computing, New York: John Wiley & Sons, Ltd., 1986.

Crawford, J. R.; Mitchell, W.; and Booth, I.:
"Application of a Computer Model for the Redevelopment of Royal Canberra Hospital." In CAD/CAM in the Eighties, Association for Computer-Aided Design, Melbourne, Australia, 1980.

Degelman, L. O.:
Acoustical Design for Optimum Reverberation Time Including Air Absorption. Department of Architectural Engineering, The Pennsylvania State University, 1968.

Dent, C.:
Quantity Surveying by Computer. Oxford, England: Oxford University Press, 1964.

Dougenik, J.; and Sheehan, D.:
SYMAP User's Reference Manual. Cambridge, Massachusetts: Laboratory for Computer Graphics and Spatial Analysis, Harvard University, Graduate School of Design, 1976.

Dudnik, E. E.:
"Overlay." In Red Book, edited by C. Steinitz. Cambridge, Massachusetts: Laboratory for Computer Graphics and Spatial Analysis, Harvard University, Graduate School of Design, 1971.

Eastman, C. M.:
"Preliminary Report on a System for General Space Planning." Communications of the ACM, vol. 15, no. 2, February, 1972.

Fenves, S. J.:
STRESS, A Reference Manual. Cambridge, Massachusetts: MIT Press, 1965.

Foley, J. D.; and Van Dam, A.:
Fundamentals of Computer Graphics. Reading, Massachusetts: Addison-Wesley, 1982.

Gamboa, T. P.:

Understanding the Significance of Computers in Architecture. An M.Sc. Dissertation. The Architectural Department. The Pennsylvania State University, 1987.

Gero, J.S.:

Computer Applications in Architecture. Applied Science Publishers Ltd. London, 1977.

Goel, V.:

Assigning Locative Prepositions to the Spatial Relations Implicit in 3-D Geometrical Models of Objects and Schemes. An M.Sc. Thesis, York University, 1986.

Goral, M.; Torrance, K. E.; Greenberg, D. P.; and Battaile, B.:

"Modeling the Interaction of Light Between Diffuse Surfaces." In ACM Computer Graphics Proceedings, 1984, pp. 213-222.

Gott, B.:

"The Scope of Computer-Aided Design." In Computer-Aided Design, edited by J. Vleitstra and R. Weilinga. Amsterdam: North-Holland, 1973.

Gouraud, H.:

Computer Display of Curved Surfaces. A Ph.D. Dissertation, University of Utah, 1971.

Greenberg, D. P.:

"Computer Graphics for Architecture: Techniques in Search of Problems," In Architectural Record, mid-August, 1977, pp. 98-105.

"Computer Graphics and Visualization." In Computer-Aided Architectural Design Futures, edited by Alan Pipes. London: Butterworths, 1985.

Hall, R. A.:

A Methodology for Realistic Image Synthesis. An M.Sc. Thesis, Cornell University, 1971.

Harper, G. N.:

Computer Applications in Architecture and Engineering. New York: McGraw-Hill, 1968.

Hawkes, D.; and stibbo, R.:

The Environmental Evaluation of Buildings. Cambridge, England: Cambridge University, Center for Land Use and Built Form Studies, Working Papers no. 15, 27, 28, 29, 30, 31, 1969.

Hittle, D. C.:

Building Loads Analysis and Systems Thermodynamic (BLAST). Champaign, Illinois: U.S. Army Construction Engineering Research Laboratory (CERL), 1979.

Hunn, B. D.:

"The DOE-@ Computer Program for BUilding Energy Analysis." Presented at the Conservation/Energy Management by Design Conference, El Paso, Texas, 1984.

Kalisperis, L. N.:

A Conceptual Framework for Computing in Architectural Design. A Ph.D. Dissertation. The Architectural Department. The Pennsylvania State University, 1988.

Kamnitzer, P.; and Hoffman, S.:
"Intuval: An Interactive Computer Graphics Aid for Design and Decision Making in Urban Planning." In EDRA 2, Proceedings of the EDRA Conference, edited by J. Archea and C. Eastman. Pennsylvania, Carnegie Mellon University, 1970.

Koning, H.; and Eizenberg, J.:
"The Language of the Prairie: Frank Lloyd Wright's Prairie Houses." In Environment and Planning B, vol. 8, 1981, p. 295.

Krawczyck, R. J.:
"Computer Assisted Scheduling for Architects." In Architectural Technology, vol. 2, no. 1, Spring, 1984.

Kusuda, T.:
NBSLD Computer Program for Heating and Cooling Loads in Buildings. NBS Report NBSIR 74-574, 1974.

Laseau, P.:
Graphic Thinking for Architects and Designers. New York: Van Nostrand Reinhold Co., 1980.

Lawson, B.:
How Designers Think. London: Architectural Press, 1982.

Leighton, N. L.:
Computers in the Architectural Office. Van Nostrand Reinhold Company Inc., New York, 1984.

Lindhult, M. S.:
"Towards an Intuitive Computer-Aided Design Process." In Proceedings of the 1987 Conference on

Planning and Design in Architecture, held in Boston, Massachusetts, August, 17-20, 1987. Edited by J. P. Protzen.

Logcher, R. D.:
"ICES STRUDL: An Integrated Approach to a Computer System for Structural Engineering." In Emerging Methods in Environmental Design and Planning, edited by G. Moore. Cambridge, Massachusetts: MIT Press, 1970.

Markus, T. A.; Whyman, P.; Morgan, J.; Maver, T.; Canter, D.; and Flemming, J.:
Building Performance. London: Applied Science, 1972.

Maver, T. W.:
"Building Appraisal." In Computer Applications in Architecture, edited by J. Gero. London: Applied Science, 1977.

"Social Impacts of Computer-Aided Architectural Design." In Advancing Building Technology. Proceedings of the 10th triennial Congress of the International Council for Building Research, Studies and Documentation, vol. 1, Washington, D. C.: 1986, pp. 176-196.

Miller, N.; Ngai, P.; and Miller, D.:
"Computer Graphics in Lighting Design." In International Lighting Review, vol. 4, 1984.

Milne, M.:
Computer Graphics in Architecture and Design. New Haven, Connecticut: Yale University Press, 1969.

"SOLAR5, A User Friendly Computer-Aided Energy Conserving Design Tool." In CAD 82, edited by A. Pipes. Guildford, England: Butterworths, 1982.

"A Building Energy Design Tool That Draws Pictures of Thermal Performance." In the Proceedings of CAMP 84, AMK, Berlin, Section C4.3, 1984.

Mitchell, W. J.:
Computer-Aided Architectural Design. Van Nostrand Reinhold Company, Inc. New York, 1977.

Monk, K. W.; and Dunstone, P. H.:
"Quantity Surveying by Computer: 1961-68." In Building, May 31, 1968, pp. 67-68.

Moore, G.:
Emerging Methods in Environmental Design and Planning. Cambridge, Massachusetts: MIT Press, 1970.

Negroponte, N., ed.:
Computer Aids to Design and Architecture. New York: Petrocelli/Charter, 1975.

Negroponte, N.; and Groisser, L. B.:
"URBAN 5: A Machine that Discusses Urban Design." In Emerging Methods in Environmental Design and Planning, edited by G. Moore. Cambridge, Massachusetts: MIT Press, 1970. 1970.

Pareto, Vilfredo:
Manual of Political Economy. New York: A. M. Kelly Publishers, 1971.

Paterson, J. W.:
"An Integrated CAD System for an Architect's Department." In Computer-Aided Design, vol. 6, no. 1, January, 1974, pp. 25-31.

Paul, C. K.; and Landini, A. J.:
Final Report: LUMIS. Pasadena, California: Jet Propulsion Laboratory, 1975.

Phong, B. T.:
"Illumination for Computer-Generated Pictures." In Communications of the ACM, vol. 18, no. 6, June, 1975, pp. 311-317.

Pipes, A., ed.:
Computer-Aided Architectural Design Futures. International Conference on Computer-Aided Architectural Design, Department of Architecture, Technical University of Delft, The Netherlands, 18 and 19 September, Butterworths, London, 1985.

Porter, W.; Lloyd, K.; and Fleischer, A.:
"Discourse: A Language and System for Computer-Aided City Design." In Emerging Methods in Environmental Design and Planning, edited by G. Moore. Cambridge, Massachusetts: MIT Press, 1970.

Powel, J. A.:
"Use of a Simple Methodology in the Compilation of a Computerized Acoustic Design Package for Architects and Builders." In the Bulletin of Computer-Aided Architectural Design, no. 12, July, 1973.

Prince, M. D.:
Interactive Graphics for Computer-Aided Design.
London: Addison-Wesley, 1971.

Iyengar, E.:
"Computers: The Profession Adjust." In Progressive Architecture, no. 5, 1985, pp. 150-151.

Radford, A.; and Gero, J.:
Design by Optimization in Architecture, Building and Construction. New York: Van Nostrand Reinhold Co., 1988.

Radford, A.; and Stevens, G.:
CADD Made Easy, a Comprehensive Guide for Architects and Designers. McGraw-Hill Book Company: New York, 1987.

Reynolds, R. A.:
Computer Methods for Architects. Butterworths Publication, London, 1980.

Schmitt, G.:
Microcomputer Aided Design. For Architects and Designers. John Wiley & Sons Ltd., Company. New York, 1988.

Simon, H.:
"Style in Design." In the Proceedings of the Environmental Design Research Association Conference, Pittsburgh: Carnegie Mellon University, Department of Architecture, 1970.

Simon, J. A.; and Rein, R. E.:
Acoustical Design and Analysis for Speech Privacy. The Department of Architectural Engineering, The Pennsylvania State University, 1968.

Smart, D. A.:
"The Application of a Computer to Quantity Surveying." In Computer Bulletin, September, 10, 1966, pp. 24-30.

Souder, J. J.; and Clark, W. E.:
Planning for Hospitals: A System Approach Using Computer-Aided Techniques. Chicago: American Hospital Association, 1964.

Stiny, G.; and Mitchell, W.:
"The Palladian Grammar." In Environment and Planning B, vol. 5, 1978, pp. 5-18.

Summers, L. H.:
Algorithms and Computational Methods in Building Design. A Report Submitted to the National Science Foundation (NSF), and The Computer Integration in Construction (CIC). Pennsylvania, 1988.

Summerson, J.:
"The Case for a Theory of Modern Architecture." In the RIBA Journal, vol. 64, 1957.

Sutherland, I. E.:
"Sketchpad: A man-Machine Graphical Communication System." In Proceedings of the 1963 Spring Joint Computer Conference. Baltimore, Maryland: Spartan Books, 1963.

Ward, W. S.; Grant, D. P.; and Chapman, A. J.:
"A PL/1 Program for Architectural Space Allocation." In the Proceedings of the 5th Urban Symposium. New York: Association for Computing Machinery, 1970.

Weinzapfel, G. E.; and Handel, S.:
"IMAGE: Computer Assistant for Architectural Design." In Spatial Synthesis in Computer-Aided Building Design, edited by C. Eastman. New York: John Wiley & Sons, Ltd., 1975.

Willoughby, T. M.:
"Understanding Building Plans with Computer Aids." In Models and Systems in Architecture and Building, edited by D. Hawkes. Hornby, Lancaster, England: The Construction Press, 1970.

Winslow, W. F.:
"Translating Individual Construction Project Information into the Resources Required for Construction." In Environmental Design: Research and Practice, edited by W. Mitchell. Los Angeles, California: UCLA, 1972.

CHAPTER 5

A FRAMEWORK FOR AN INTEGRATED COMPUTER-AIDED ARCHITECTURAL DESIGN DECISION SUPPORT SYSTEM

CHAPTER 5

5. A Framework for an Integrated Computer-Aided Architectural Design Decision Support System.

5.1. The Architectural Design Process.

5.2. The "BASED" Model; An Information Processing Model for the Architectural Design Process.

5.3. The Proposed Approach.

5.3.1. The Suggested Framework.

5.3.1.1. The Multi-disciplinary Users.
5.3.1.2. The Decision Support System.
5.3.1.3. The Task Environment of Architecture.

5.4. A Conceptual Model for the Decision Support System, The (ICAAD.DSS) Model.

5.4.1. The Communicator, or the dialog Generation and Management System (DGMS).
5.4.2. The Database or the Databank (DB).
5.4.3. The Database Management System (DBMS).
5.4.4. The Program Bank (PB).
5.4.5. The Expert System Component.

CHAPTER 5

A FRAMEWORK FOR AN INTEGRATED COMPUTER-AIDED ARCHITECTURAL DESIGN DECISION SUPPORT SYSTEM

This chapter presents the "ICAAD.DSS" conceptual model, which provides a framework for an integrated computer-aided architectural design (CAAD) decision support system. The model is based on a unified approach to computing in architecture, which in turn is based on a holistic view of the architectural design process. The proposed model shifts the focus from product to process, and views the design problem as a goal-oriented, problem-solving activity that allows a design team to identify strategies and methodologies in the search for design solutions. This chapter introduces a new environment for the use and integration of computers in the architectural design process.

Architectural design has long been considered a uniquely human process that relies on the intuition, experience, and judgment of the designer. Design methods, as discussed earlier in chapter two of this thesis, is a specific branch of design theory which addresses the design processes in order to increase the quality of the design activities, by explicating the knowledge necessary for the execution of the design process (Kalisperis, 1988).

Computer-aided design developments have paralleled developments in design methodologies. Computer-aided architectural design (CAAD), is design supported by structured data and adequate computer programs to elaborate these data. In its most developed form, CAAD is a decision support system

to be employed in the architectural design process (Bax, 1986).

In order to develop these systems, one has to study the nature of the design process, and know what is possible in the field of information and computer technology. It was important then to develop, an information processing model for the design process, then proceed with the development of the ICAAD.DSS model. The information processing model which is presented by the "BASED" model has its basis in design methodology, the BASED model explores the entire design process, and allows the architect to "enter" it at any desired phase or mode. However, the goal of this chapter is to introduce a new environment for the use and the integration of computers in the architectural design process, which is achieved through the introduction of the (ICAAD.DSS) model for an integrated computer-aided architectural design decision support system.

5.1. The Architectural Design Process:

An outgrowth of the first generation design models, as was discussed earlier, was the realization that architectural design problems are ill-defined problems (Rittel and Webber, 1973). Emerging social problem-solving paradigms, which seek to construct a cognitive psychology of problem-solving have direct relevance to architectural design. In this context, architectural design problems have been presented as both; problem-solving and puzzle-making. For some time, it has been argued that much of what architects do has the character of puzzle making (Archea, 1985, 1987). Puzzle making predicates that the objective is known, that there is a clear answer and that the information needed to solve the puzzle is available (Heath,'1984). However, in the case of architectural design, the objective may not always be known, and the problem space might not contain a solution to the problem.

Design problems, of which architectural design problems are a subclass, are not simply puzzles. Design problems are distinguished from other problems by the fact that their goal state cannot be predefined. They are concerned with "closure of the terminal state" (Wade 1977) or establishing the "most satisficing solution" (Simon 1970). Actually, architectural design includes structured, semi-structured and ill-structured problems; all of which are included to varying degrees in each situation across a continuum (Heath, 1984). The goal state of architectural design problems is not predefined. Additionally, since architectural problems are ill-defined, certain assumptions have to be made which in turn are influenced by the degree of problem severity and the conceptual tools the designer uses.

The architectural design problems can be characterized by the following descriptive phrases:

a. The problems associated with the system are multi-disciplinary.
b. There are multiple levels of merit.
c. The measures of merit may not be equally important to the final decision.
d. All of the information required for making a decision leading to an adequate design may not be available.
e. Some information may be "hard" (that is, based on scientific principles) and some information may be "soft" (that is, based on the perception and judgment of the designer).
f. They occupy a continuum along structured and unstructured problem types.
g. They involve quantitative as well as highly qualitative subtasks.

h. They do not follow a descriptive or a normative decision making approach, but rather an oscillation between both approaches.

Architectural design problems are open and only partial structures at best. Adequate solutions to such problems can best be negotiated by a team effort which is integrated via computer based design support systems (Kalisperis, 1988).

Architectural design problem-solving entails the determination of objectives and whether or not it is possible to accomplish them. It is also reinforced by the concepts of "task environment" and "problem space." Task environment is defined as "situation", which refers to an environment combined with a goal or task and is extremely complex as it includes the wider segments of society in relation to architectural design, refer to chapter two in this thesis. On the other hand, problem space is the problem solver's internal representation of the task environment; the mode of conceptualization of the problem space has a significant impact on design synthesis.

During the search for solutions in the "solution space", there is always an iterating or cyclical relation between different steps taken throughout the process which range from "intuitive leaps" to complex technical calculations. Within each step, information is obtained, analyzed, synthesized, evaluated and a decision is taken about the next step, whether to proceed forward or to back-track to a certain step in the process, i.e. that there is always a certain chain between forward and backward tracking throughout the whole process.

5.2. THE "BASED" MODEL:

The "BASED" model (Hosny et al, 1990), or the information processing model for the architectural design process, will

elaborate on the ADIS model (Beheshti and Monroy, 1986). The most essential characteristic of the ADIS model is the continuous feedback between its different steps. Consequently, the proposed model of the design process will constitute five steps:

i) Briefing or problem definition; this refers to the listing and classification of all the design requirements, regarding the gathering of the needed information, confronting it with the intended outcome, the ranking of the factors or the problems, their inter-relations and the formulation of the different courses of action. In other words this is the definition of the problem space along with the solution space.

ii) Analysis; this is defined by Jones as a "listing of all design requirements and the reduction of these to a complete set of logically related performance specifications," then information context is added to each specification.

iii) Synthesis; in Jones's words is "finding possible solutions for each individual performance specification and building up complete designs from these with least possible compromise." In other words it is the search for solutions in the solution space.

iv) Evaluation; which refers to the testing of the accuracy with which alternative designs fulfil the different performance requirements, and finally

v) Decision; which refers to the logical selection of one of the alternatives in the context of the task environment and the intended solution space, then communicating this decision with another step in the process whether it be a forthcoming or a previous step.

Later on in this chapter, this process will be referred to by the acronym: BASED. It is important to point out here that this iterating process; "BASED", exists in all the different stages of design, but with different values of importance regarding each step. In other words, in the synthesis step, for example, there are the different steps of the BASED process, except that the emphasis is given much more to the synthesis part of the BASED, than it is given to the rest of the parts, as shown in fig. (5.1).

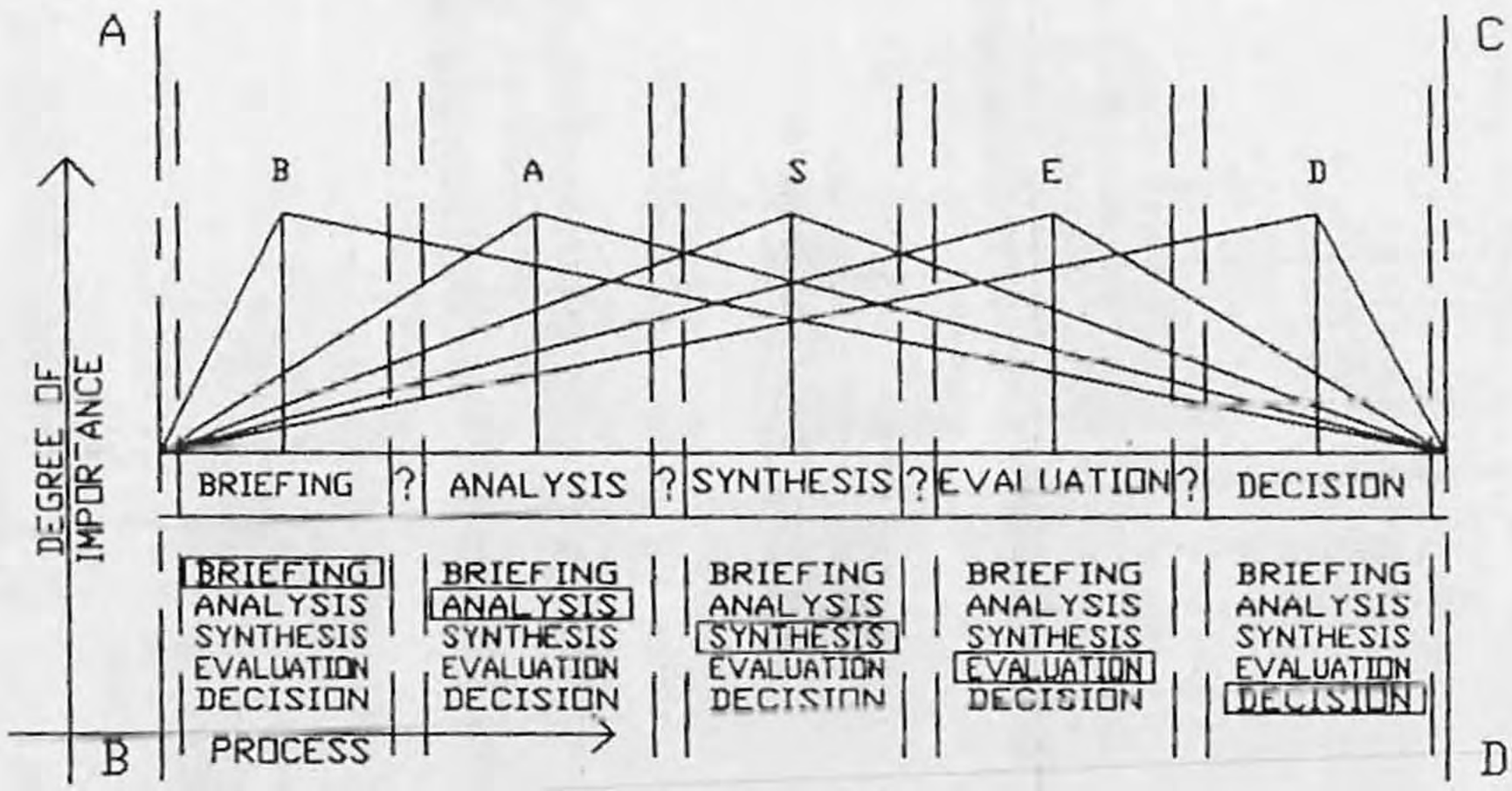


Fig. (5.1): The BASED process

As discussed earlier, architectural design problems occupy a continuum along structured (well-defined) and unstructured (ill-defined) problem types, besides, their organizational or functional boundaries are not the same and sometimes they are not definable; in effect, there is no sharp line separating the different steps of the BASED process, instead, there is a continuum, or a fuzzy area in which lies the designer's "Weltanschauung" or his own philosophy, intuition and creativity, along with his own personal view of regarding and evaluating the different alternatives according to his own perception and cognition style.

The different constituents of the BASED process are all happening at the same time. The architect is briefing, analyzing, synthesizing, evaluating and making some decisions concurrently, depending on the different information and experience he has. The correct way to perceive this diagram is to imagine that it is not a linear one, instead, it is seen as a cylinder by putting the left line AB on top of the right line CD. This perception gives an idea that the design process is considered to be cyclical, iterating and continuous. Figure (5.2), shows the cyclical relation between the different BASED steps, and the database in the middle of the circle to be accessible from any step at any time.

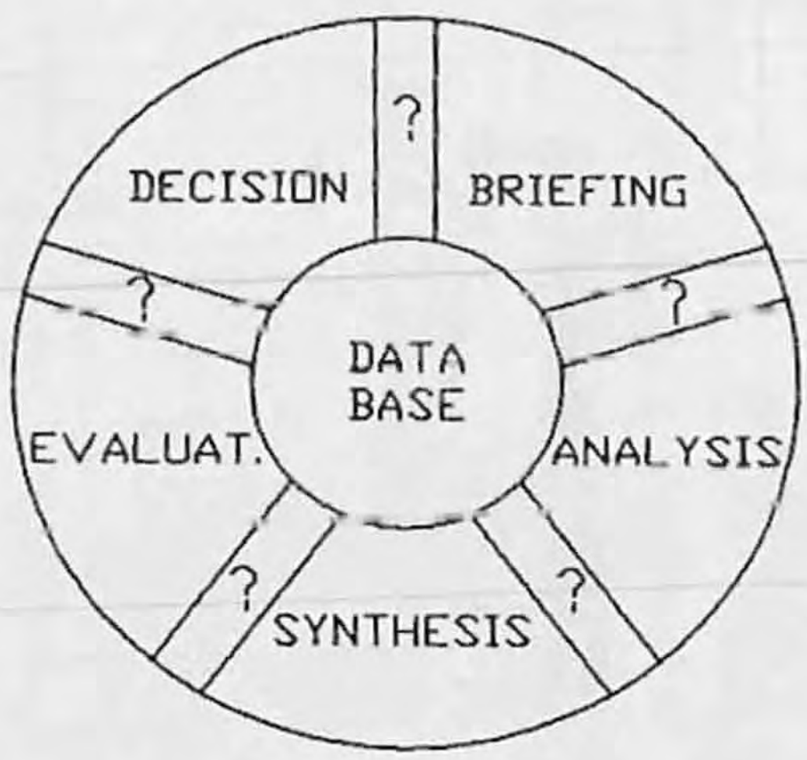
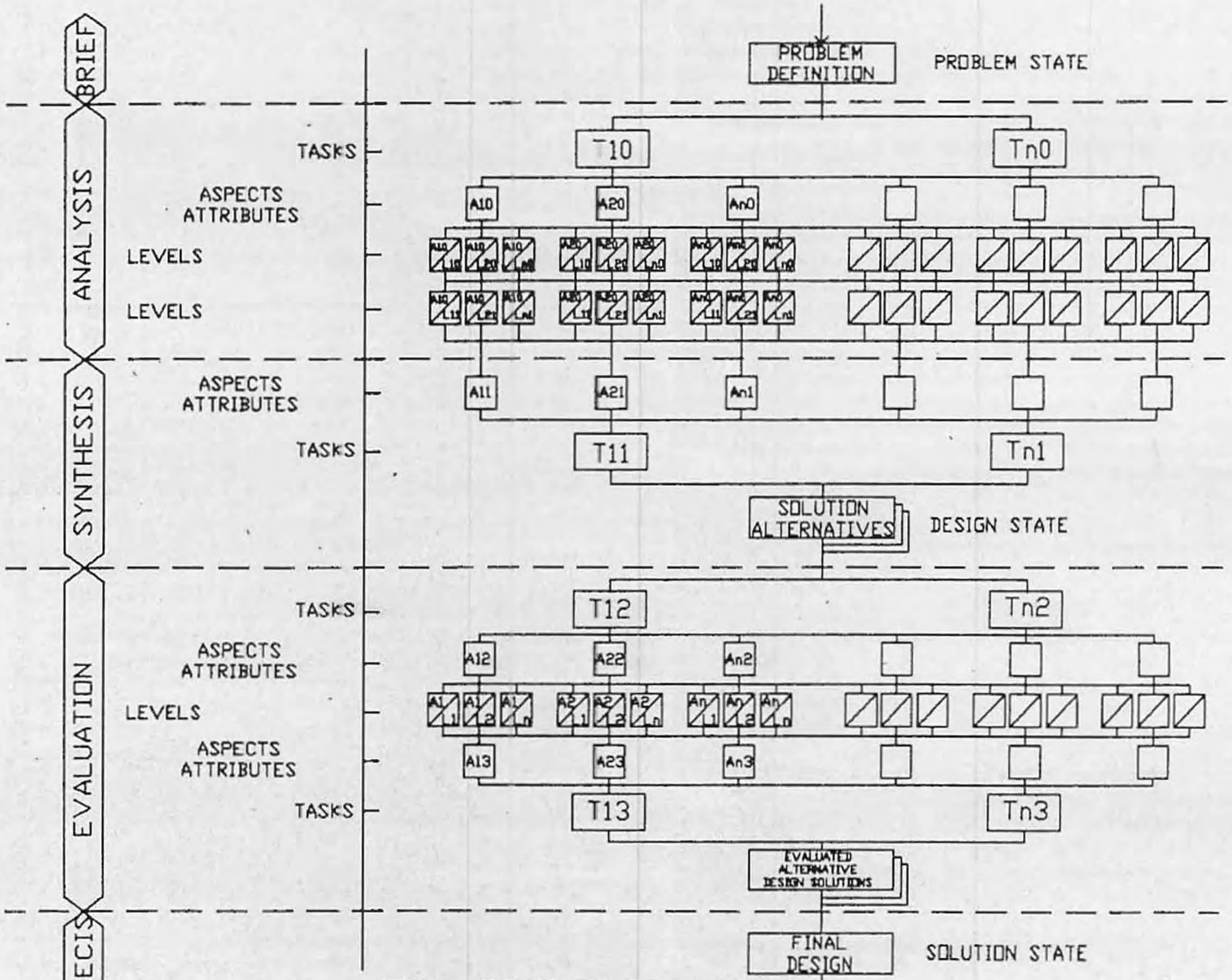


Fig. (5.2): The cyclical (iterative) design process.

This view of the design activity will be the basis for the description of each step in the BASED model, regardless of the different phases, which slightly differ from one practice to another. Different views of the phases could be seen in the American Institute of Architects (AIA), or the Royal Institute of British Architects (RIBA) plans of work. The end result, or the model, is a generic map (fig. 5.3) that charts the entire process to be applicable to any one of the different multi-disciplinary users.



THE PROPOSED MODEL↓
FOR A SINGLE PHASE IN THE ARCHITECTURAL DESIGN PROCESS

Fig. (5.3): The BASED model of the design process for one phase.

As mentioned earlier, the architectural design process (ADP) is not seen as a linear process, instead it is seen as a cyclical, iterating one. The reflection of this view on the model is presented in the freedom of choice for the designer in moving from one task to another or from one subtask to another, and in the possibility of moving from one step in the BASED process to another without quitting the main task on hand, and without having to go through a special pre-defined sequence. Instead, the designer is able to access the process and start his design at any point in the map, he is able to show his own cognitive style, and to adopt his own approach for searching the problem space. He has the ability to adopt either "a breadth first" or " a depth first" search approach or even a combination between both approaches, in a way similar to a "hill climbing" approach.

Figure (5.4) illustrates that the nodes represent states; either a problem state, a design state or a solution state, and the links represent transitions, or certain processes applied to the nodes to change their current state to another future state.

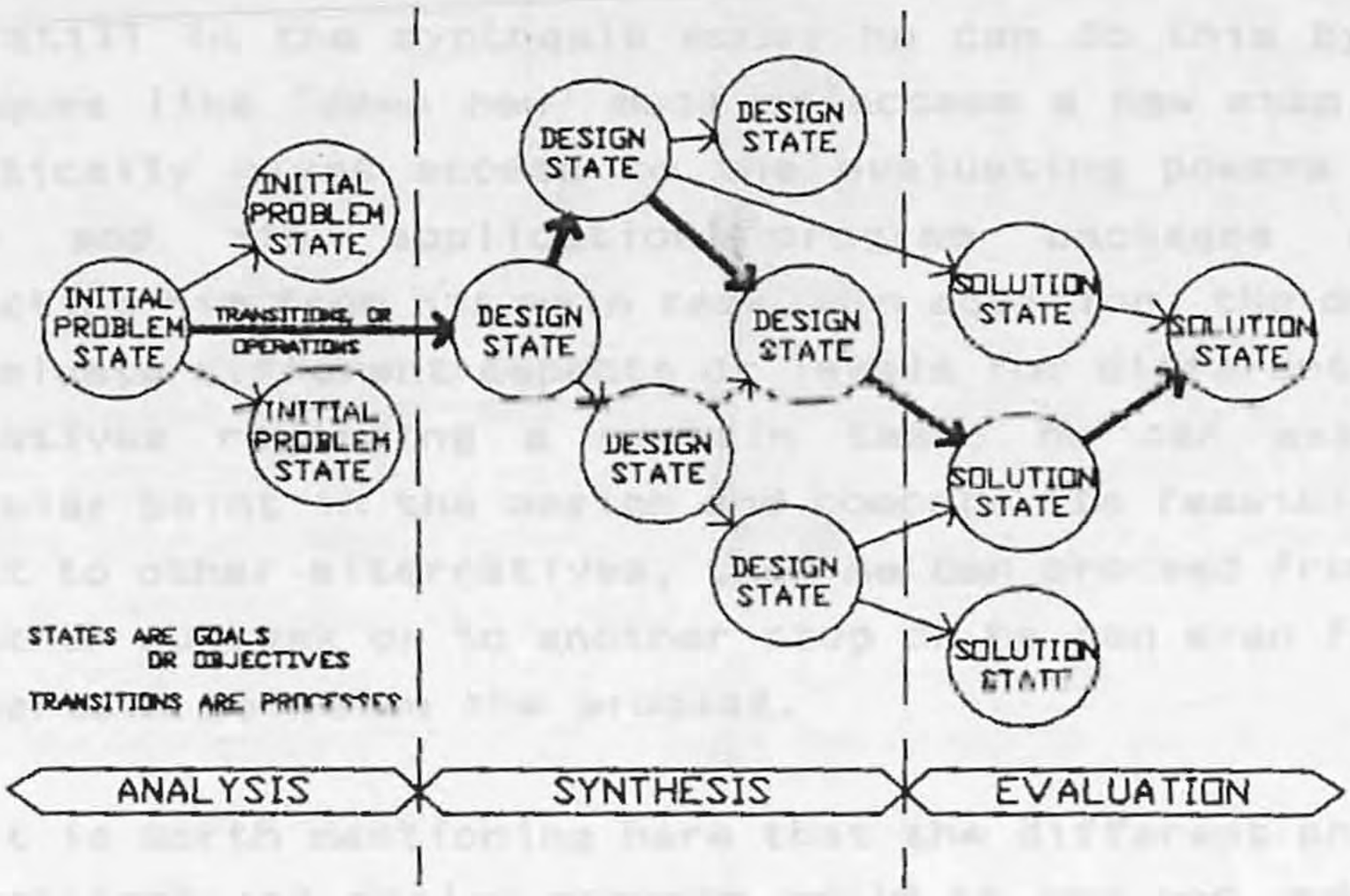


Fig. (5.4): Design states and transitions.

It is possible for the designer, for example, to examine a certain problem in respect to more than one aspect under different tasks concurrently. He can move from one node (state) to another through certain links (processes). Referring to the BASED model in figure (5.2), the designer can move from one aspect A_{10} under task T_{10} to another aspect A_{20} under the same task T_{10} without having to go "in depth" till completely processing aspect A_{10}. He may pick a certain level A_{10}/L_{10} under a specific aspect A_{10} and cross reference it with another level A_{20}/L_{10} under another aspect A_{20}, or A_{n0}/L_{10} under aspect A_{n0} for example. Then from there, he (the designer) can proceed to cross examine the same level for the same aspect under another task T_{n0} and so on.

In addition, the designer has the ability to move from one step in the process to another without having to go "in depth" regarding a certain step, i.e. he can adopt "a breadth first" approach and go across the chart. As an example, it is possible for the designer to evaluate any selected alternative in respect to the different given criteria while he is still in the synthesis mode; he can do this by using techniques like "open new" mode or access a new step, which automatically gives access to the evaluating powers of the system and its application program packages without distracting him from his main task. In addition, the designer can evaluate different aspects or levels for different design alternatives regarding a certain task; he can examine a particular point in the design and compare its feasibility in respect to other alternatives. then he can proceed from there to another subtask or to another step or he can even feedback to a certain point in the process.

It is worth mentioning here that the different phases in the architectural design process could be grouped under five main categories, as in figure (5.5), viz.:

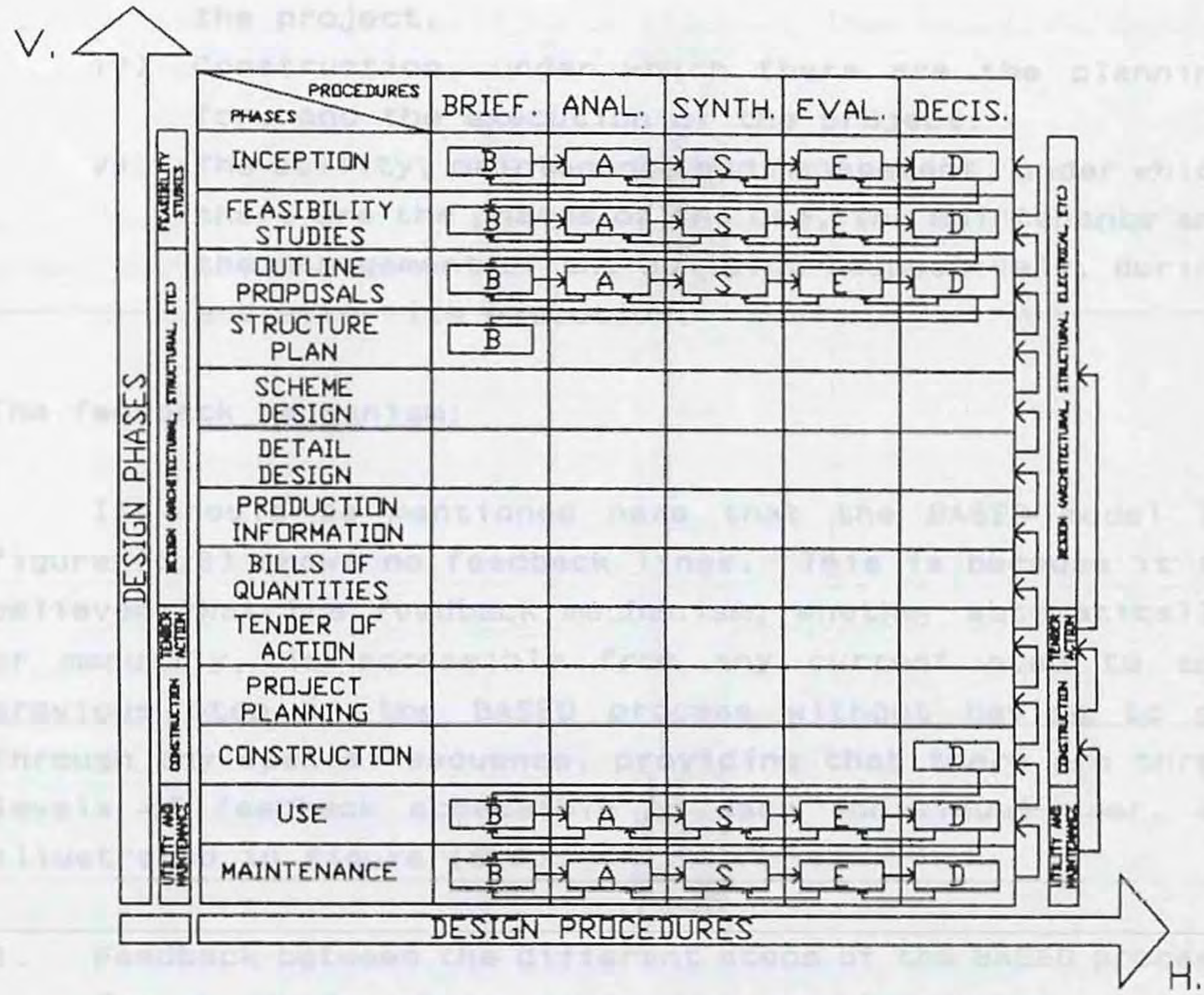


Fig. (5.5): The different design phases and procedures in the ADP.

i) Feasibility studies, under which there are the phases of: recommendations for the client, and determining the general approaches for the layout, design and construction.

ii) Design activities, encompassing the architectural, structural, electrical, mechanical and HVAC design activities, under which there are the preliminary, the design development and the detailed design phases.

iii) Bid preparation or the tender of action, under which there are the phases of: preparing bills of quantities, and the different specifications for the project.

iv) Construction, under which there are the planning for, and the execution of the project.

v) The utility, maintenance and management, under which there are the phases of the use, the maintenance and the management of the building respectively, during and after its execution.

The feedback mechanism:

It should be mentioned here that the BASED model in figure (5.3) shows no feedback lines. This is because it is believed that the feedback mechanism, whether automatically or manually, is accessible from any current step to any previous step in the BASED process without having to go through any special sequence, providing that there are three levels of feedback accessible to each individual user, as illustrated in figure (5.5):

1. Feedback between the different steps of the BASED process for a single phase under the same category, e.g. a feedback from an evaluation step to a synthesis step in the detailed design phase under the design category, weather it is architectural, structural, electrical or mechanical.

2. Feedback between the different steps of the BASED process for more than one phase under the same category, e.g. a feedback from an evaluation step in the detailed design phase, to a synthesis step in the design development phase under the design category.

3. Feedback between the different steps of the BASED process for more than one phase under different categories, e.g. a feedback from an evaluation step in the bid operation category, to a synthesis step in the detailed design phase under the design category.

The strategy for the automatic feedback mechanism, and the redesign process depends on a special search algorithm named Heuristic-Depth-First (HDF) Lee and Kwon (1989), which will be explained later under the expert system component in the framework.

5.3. The Proposed Approach:

The understanding that the architectural design process (ADP) is necessarily based on an assembly and interpretation of a substantial amount of knowledge, is believed to be of crucial importance for the concept of a CAAD system. The processes and the information which drives them cannot be separated in a CAAD system. The nature of the data flow is dynamic or bi-directional and includes a wide range of different information resources. Architects usually retrieve information, change it to serve their needs, decide upon what to do, then return it as modified information to the information pool, (or the blackboard in the proposed framework) so that other users can start building upon it.

In order to support this decision-making process, the study has to develop a flexible decision support system to aid the architect in his decision making. A unified approach is needed to the employment of computers in the creative and decision functions of the ADP.

The model proposed here, will attempt to provide such a framework. The model will consider the architectural design process (ADP) as a goal-oriented, problem-solving activity,

where each state can be obtained by the application of an operation to the previous state, to allow the architect to identify the different strategies and methodologies in the search for architectural design solutions, combining knowledge of diverse areas towards a common purpose within a flexible decision support system. Decision makers must be provided with an environment within which a specific decision support system can be built through the selection of appropriate data and models (Kalisperis, 1988). A conceptual diagram for the needed environment is shown in figure (5.6).

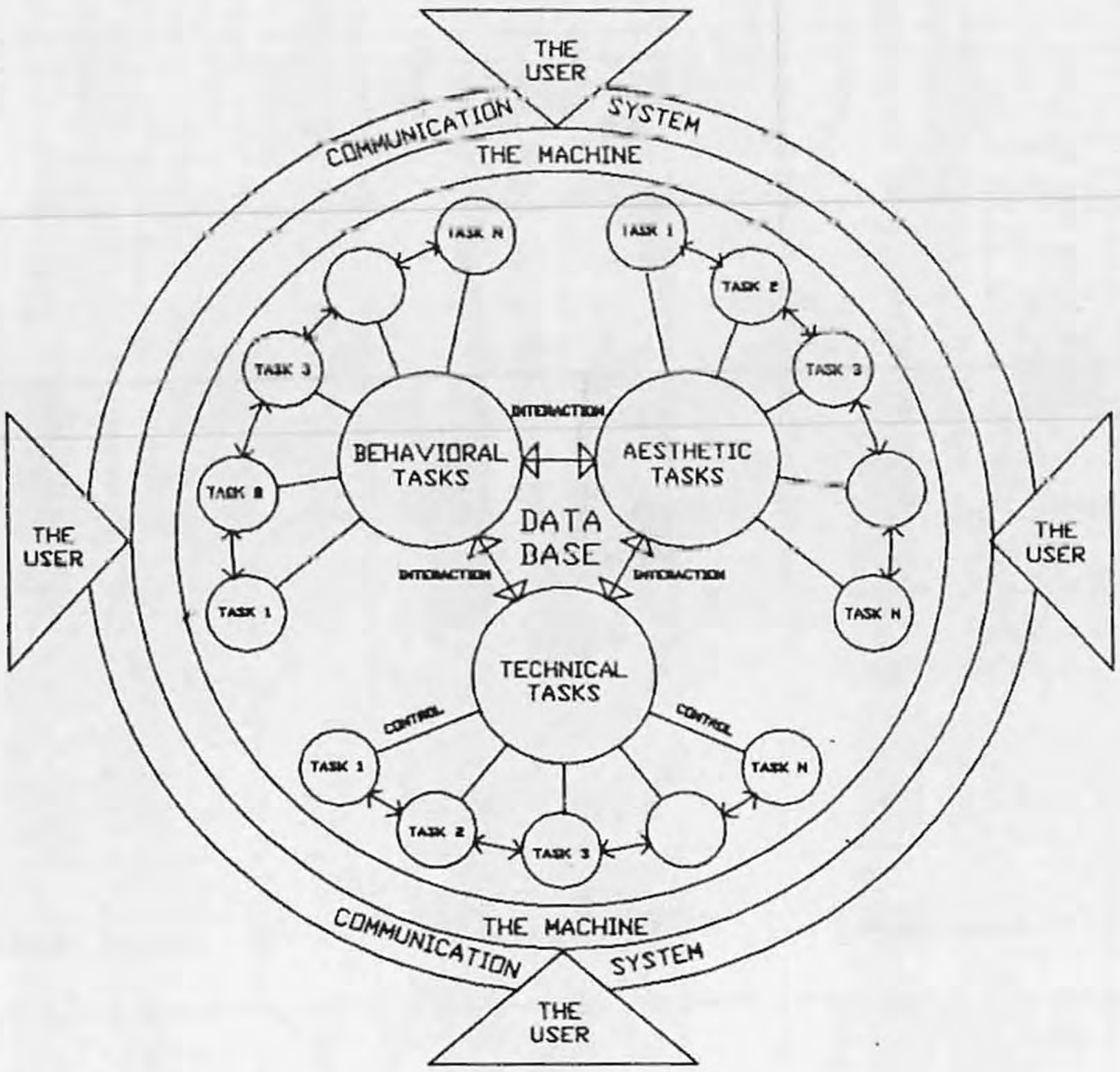


Fig. (5.6): The computer-aided architectural design process (For a single phase).

The diagram represents one phase of the design process. In the figure, the different users are shown gathered around the system, communicating with each other by means of their computers which are shown embracing the different task environments of the ADP, viz. the behavioral, the technical and the aesthetic tasks, which will be discussed later under the "Task Environment of Architecture". They are drawn as circles having the same diameter, to indicate that they all have equal importance in the ADP. They are located inside the inner rings, to indicate that they are directly coordinated and handled simultaneously by the design team, through the central database. The database is accessible for all design subtasks, it is not duplicated and is updated as more information becomes available. The updating process is discussed later under the different levels of the external environment.

5.3.1. The Suggested Framework:

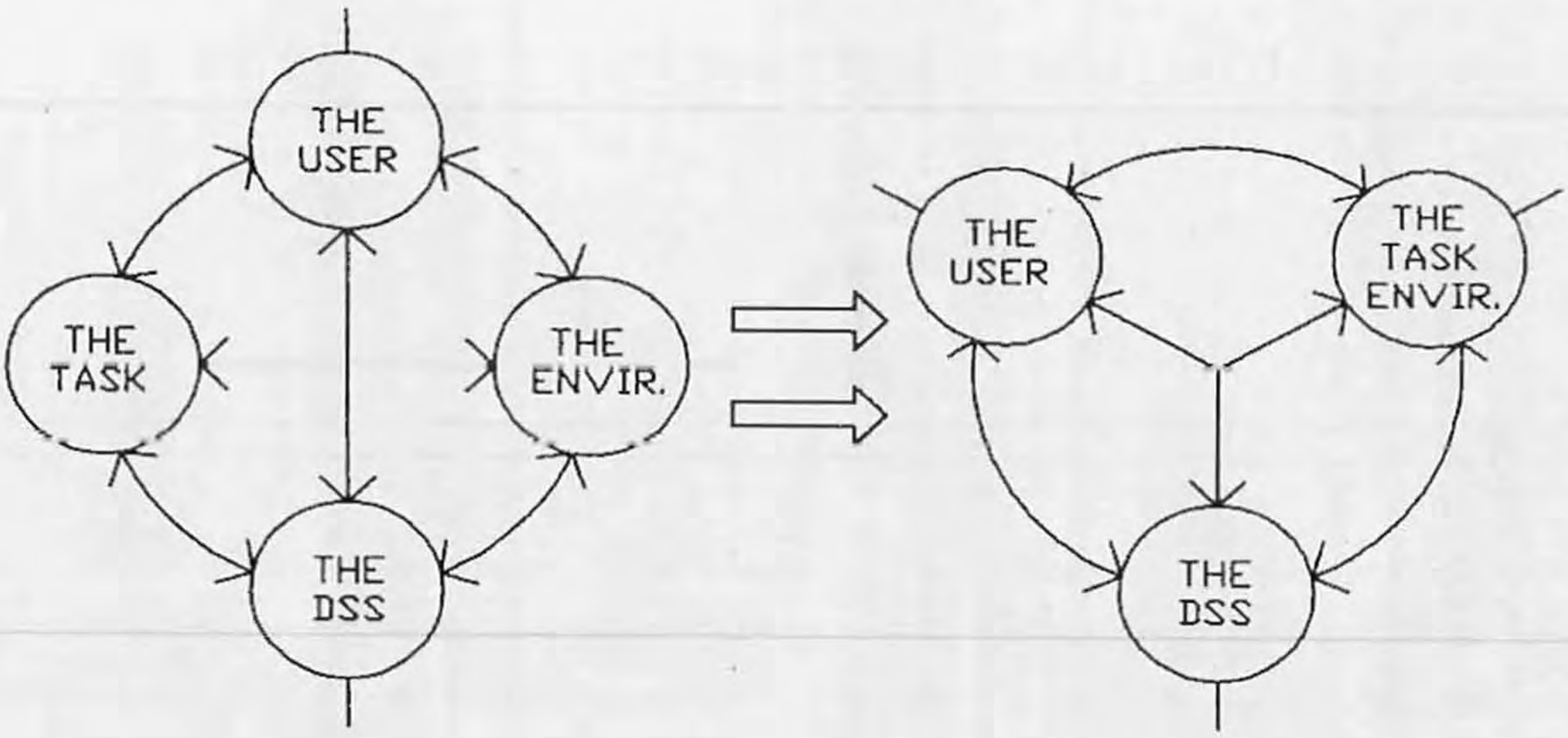
Adopting this previous view of the design process, and given the methods to simulate it in the computer, it is possible now to define the different components of the suggested framework as in figure (5.7).
The framework will include:

1. The multi-disciplinary users or the human factor.
2. The decision support system.
3. The task environment of architecture.

5.3.1.1. The Multi-disciplinary users:

It is important to involve all the different parties who exercise influence on the development of the architectural object or the artifact, as participants, like actors in a play, all take part in and contribute to the results of the process in a well balanced and controlled manner. The users

might be architects, landscape architects, interior designers, or architectural engineers, or engineers whether they be structural, mechanical, or electrical engineers.



a: The theoretical approach b: The practical approach

Fig. (5.7): The suggested framework for the decision making system.

Among the whole organization, there are three levels of control over the design process; these are:

i) The personal or local level, in which the designer himself makes his own decisions and controls his own process.
ii) The project's level, in which the designer's decisions are given to another designer to accomplish his own part of the design under the control of the project manager.
iii) The system's or the organization's level in which different decisions made by the different designers inside the organization are evaluated by the system's managers, or the organization's director.

Design problems are team problems, based on a number of different and contradicting disciplines. Designing is considered to be a social activity in which various parties seek consensus about what to do (Bucciarelli et al, 1987; Habraken, 1987; and Robbins, 1987). Among a particular design team, it is very well possible that the results of the contribution of different architects/designers will come up with many conflicts, unless their is a certain control from the architect/designer who is responsible for the higher design level to re-specify the requirements and constraints for each of the sub-tasks assigned to the different users as described before.

Within a group, a designer may have a set of activities open simultaneously (e.g. through windowing techniques). Each activity can interact with others by watching intermediate results on the screen. However, the database management system must provide support for data protection or security through the use of the "read-only" mechaniom which gives access to some design data areas in a way similar to that suggested by Wheeler (1985) in his "unified model for building". The architect produces a root drawing or model, the "Architect's base plan", to which the other consultants have "read only" access and on top of whiuh they can add their own "write protected" files. Every time they access the model to write in the outcome of their work on the project, they are prompted with the current version of the "Architect's base plan" on the screen, and can thus respond promptly to recent changes and avoid wasting time on redundant work. The architect meanwhile adds uniquely architectural material in his own overlaid files and maintains the root model as everybody's work requires (Wheeler, 1985). In this way, each user will have the ability to see other user's work, but only make changes to those parts for which he is responsible. This means that individuals will be able to control, to a certain allowable degree, what other group members are able to see,

and hierarchies of transactions can be defined (DeSanctis and Gallupe, 1986).

5.3.1.2. The Decision Support System:

In order to facilitate computing in the architectural design process, a flexible decision support system environment needs to be developed (Kalisperis, 1988). A unified approach to design embracing the behavioral, the scientific and the artistic modes of design problem solving is required. This would ultimately lead to a better understanding of the goals and the means in architectural design, in other words, the task environment of architecture (Heath, 1984 and Haider, 1986). Such a unified approach, incorporating several vast fields of knowledge (Watson, 1984), demands a viable conceptual framework.

It is likely that no specific environment will be required to satisfy all the needed requirements, in fact, it is important to recall that the performance criteria for any specific decision support system, like the one presented here, will be entirely dependent on the design problem itself, the task environment and the cognitive style of the designer involved. All of which are subject to change. A major value of decision support systems is their ability to accommodate change. Thus, it is impossible to identify a typical specific decision support system. However, the following objectives represent a set of capabilities that characterize the full value of the system's concept from the designer's point of view:

1. It should support decision making in the first place.
2. It should be able to support different users at all levels, using different kinds of programs in which there is a variety of representational schemes or symbol systems. It should support the different user's views

of the object. Every designer has his own view of the design that reflects his own culture, knowledge and experience. The system should assist in integration between the levels whenever possible. Moreover, a major need articulated by decision makers is the need for integration and coordination of decisions made by different designers or specialists dealing with related parts of a larger problem, as well as accommodating group decisions where decisions must result from negotiation and interaction among several decision makers. This could be achieved by using a common high level language within which different types of the needed programs can be accommodated. Integration between the different users can be achieved through the use of a single integration point; the data dictionary (Kalisperis, 1988).

3. It should be possible for all the design team members to work on the system simultaneously. In accordance, it should provide a certain kind of protection to the data against corruption. The system must provide access with control and locking to prevent multiple users from corrupting the data. This could be achieved by having a protection function in the database management system. Once a model or an object is created in the system, it could only be accessible to the other parties of the design team through "reading" only. For the other consultants to add their own write protected files to that model or object (e.g., the architect's base model), they'll have to add on top of it, by using the concept of layers. Every time they access the architect's base model to start their own work, they should be prompted with the current version of the model, and can thus respond immediately to recent changes and avoid wasting time on redundant work.
4. It should support all phases of the decision making process. In 1960, Simon introduced a popular model of

decision making (Simon, 1960). He envisaged three main steps in the process:

i. Intelligence: Searching the environment for conditions calling for decisions. Raw data are obtained, processed and examined for clues that may identify problems.

ii. Design: Inventing, developing and analyzing possible courses of action. This involves processes to understand the problem, to generate solutions and to test their feasibility.

iii. Choice: Selecting a particular course of action from those available. A choice is made and then implemented.

5. Any user should be able to find a needed piece of information easily and quickly through the database. This could be achieved by means of using the concept of the moving, or bidirectional database (Thierauf, 1988).

6. The system should keep track and record of all the different decisions taken by the users throughout the different phases of the architectural design process. This could be achieved by using the retrieval function in the database management system, through the use of a blackboard architecture. The blackboard architecture is layered so that the development of different versions of a design can be traced, and different refinements of partial designs can be compared (Sprague and Carlson, 1982).

7. All users would be able to use, create and add new programs to the system. New programs should be named, and made available to other users. Conflicts or collision between different names should be minimized. The means to avoid names collision is to require that every name includes the path to it.

8. The system should be user friendly or easy to use, in the sense that one does not need to have any programming skills to make use of it. This could be achieved through

the dialog generation and management system. It should prompt the user with the different alternatives and give him the freedom of choice between them. Suggesting solutions and alternatives could be done by having an expert system component in the system.

9. The system should be transparent, in the sense that its tools should not be of any kind of obstacle to the designer. The tools should not distract the designer from his main task, towards some tedious process for managing the tools or operating the system, they should be used without conscious effort in controlling them. In addition the tools should give the user multiple choices between different ways of information forms or representation, e.g., the user can work either in a graphical, or textual form. The system's tools should become transparent (not noticeable) more like the way one speaks his mother language without having any difficulty.

It is important to point out here, that it is a temporary specific decision support system that aids the design team in solving a particular problem within its specific context based on the design team's problem perception and formulation. The flexible decision support system environment allows the creation of a temporary system which will accommodate the solution of an architectural design problem. "Such a specific system is only capable of solving the particular problem that it was created for" (Kalisperis, 1988), it could be either discarded upon achievement of a satisfactory solution, while the overall environment is retained, or, it could be saved for later use on similar problems, like a chain of restaurants for example.

In addition, it should be quite clear that there is no one correct solution for any design problem. This theoretical limitation is applicable in the design of a CAAD decision support system as well.

5.3.1.3. The Task Environment of Architecture:

As discussed earlier, the architectural design problem-solving entails the determination of objectives and whether or not it is possible to accomplish them. It is also reinforced by the concepts of "task environment" and "problem space". The task environment is defined as a "situation", which refers to an environment combined with a goal or task, it is extremely complex as it includes the wider segments of the total society in relation to architectural design. Newell and Simon (1972) argue that it is impossible to describe any task environment completely, Heath (1984) agrees that individuals and tasks differ, and a description that is apt for one situation may not be apt for another. On the other hand, problem space is the problem solvers' internal representation of the task environment; the mode of conceptualization of the problem space has a significant impact on design synthesis.

The External Environment:

As discussed earlier in this chapter, the database is accessible for all design subtasks, it is not duplicated, and is updated as more information becomes available.

The updating occurs in the three levels of the external environment shown in figure (5.8). In the first level, or the local level, where there are different architects/designers working in the same project, then, the individual architect is responsible for the updating. In the second level, that is the project level, where there are different designers like architects, architectural engineers, structural engineers, mechanical engineers, etc. all working in the same project under the supervision of the project manager, then the project manager is responsible for updating the data. In the third level, or the system level the system manager or the

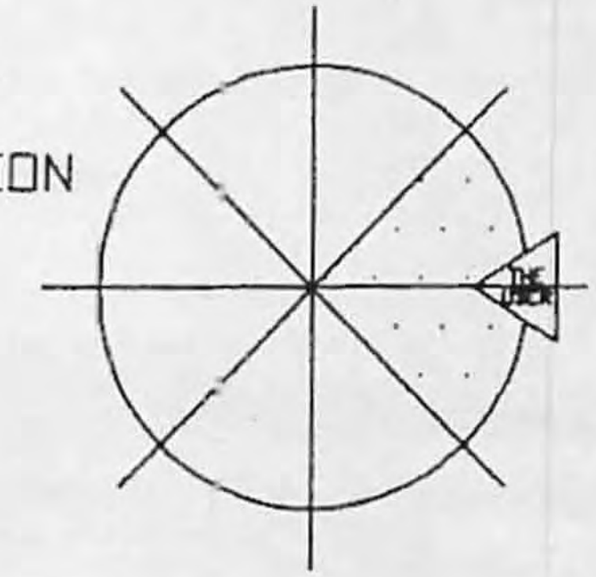
organization's director is the one responsible for the updating of data.

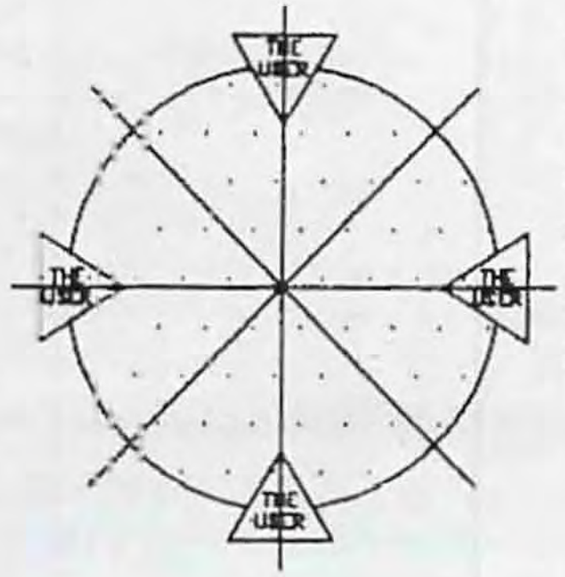
The three main tasks or modes of the architectural problem solving, according to Watson (1984) could be divided into the following individual subtasks as examples:

1. The behavioral tasks, (the building as a social tool):
   a. anthropology,
   b. environmental psychology,
   c. policy studies,
   d. architectural programming,
   e. decision-making methods and
   f. user participation.
2. The scientific or technical tasks, (the building as a product):
   a. economics,
   b. ecological impact studies,
   c. building science,
   d. construction technology,
   e. environmental controls and
   f. product design.
3. The artistic or aesthetic tasks, (the building as an art object):
   a. aesthetics,
   b. architectural history,
   c. cognitive psychology,
   d. theory and criticism,
   e. visual methodology and
   f. architectural design.

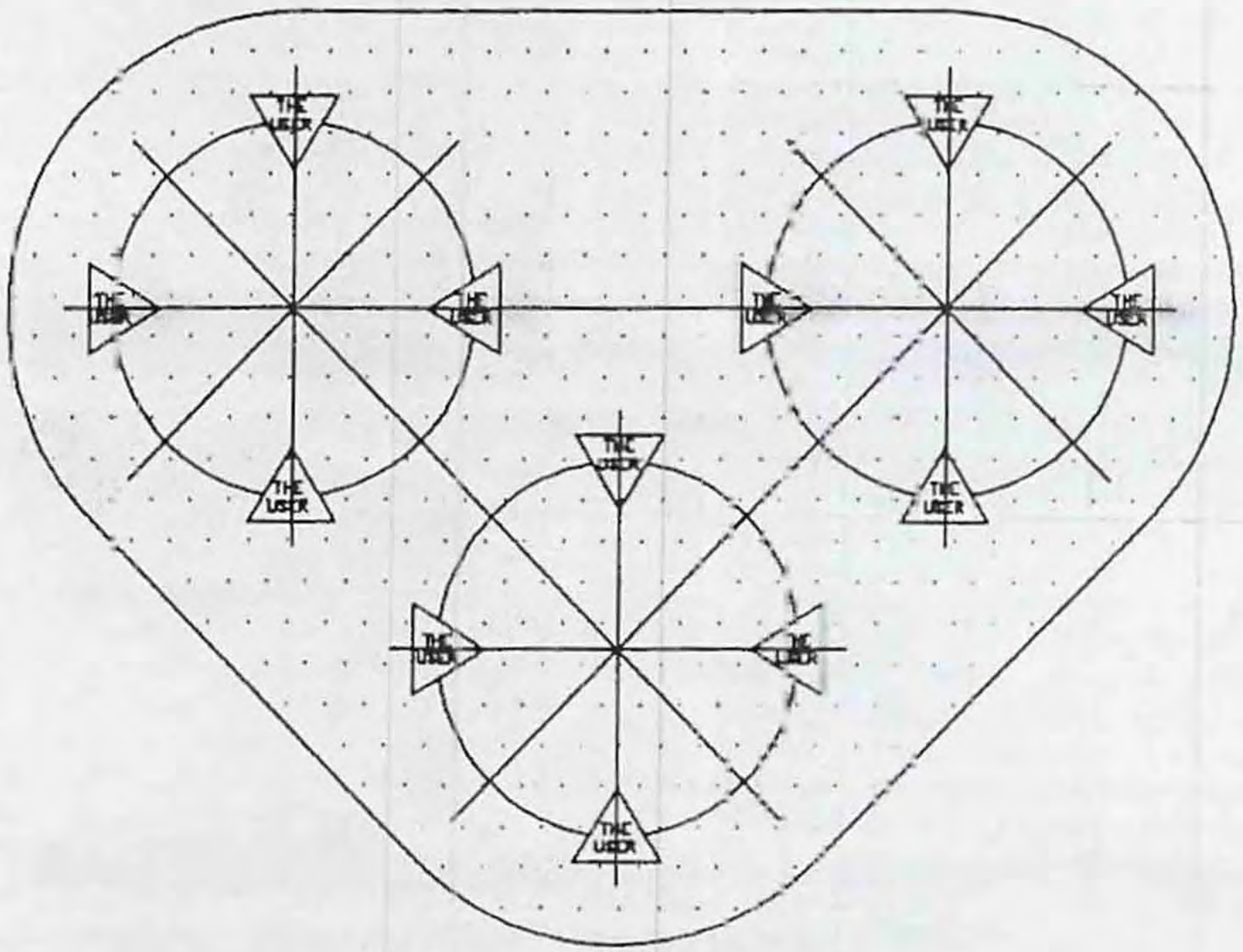
The individual subtasks are integrated and interrelated as well, and they are not limited to a specific number. Each individual user has his own computer and tools (workstation), and communicates with the rest of the design team through the blackboard. This communication is controlled by the controller under the dialog generation and management system, as will be described later on in this chapter.

THE FIRST LEVEL
'LOCAL LEVEL'
* ONE ORGANIZATION
* ONE PROJECT
* ONE USER
* ONE TASK





THE SECOND LEVEL
'THE PROJECT'S LEVEL'
* ONE ORGANIZATION
* ONE PROJECT
* DIFFERENT USERS
* DIFFERENT TASKS



THE THIRD LEVEL
'SYSTEM'S LEVEL'
* ONE ORGANIZATION
* DIFFERENT PROJECTS
* DIFFERENT USERS
* DIFFERENT TASKS

Fig. (5.8): The three levels of the external environment.

5.4. A Conceptual Model for The Decision Support System, The (ICAAD.DSS) Model:

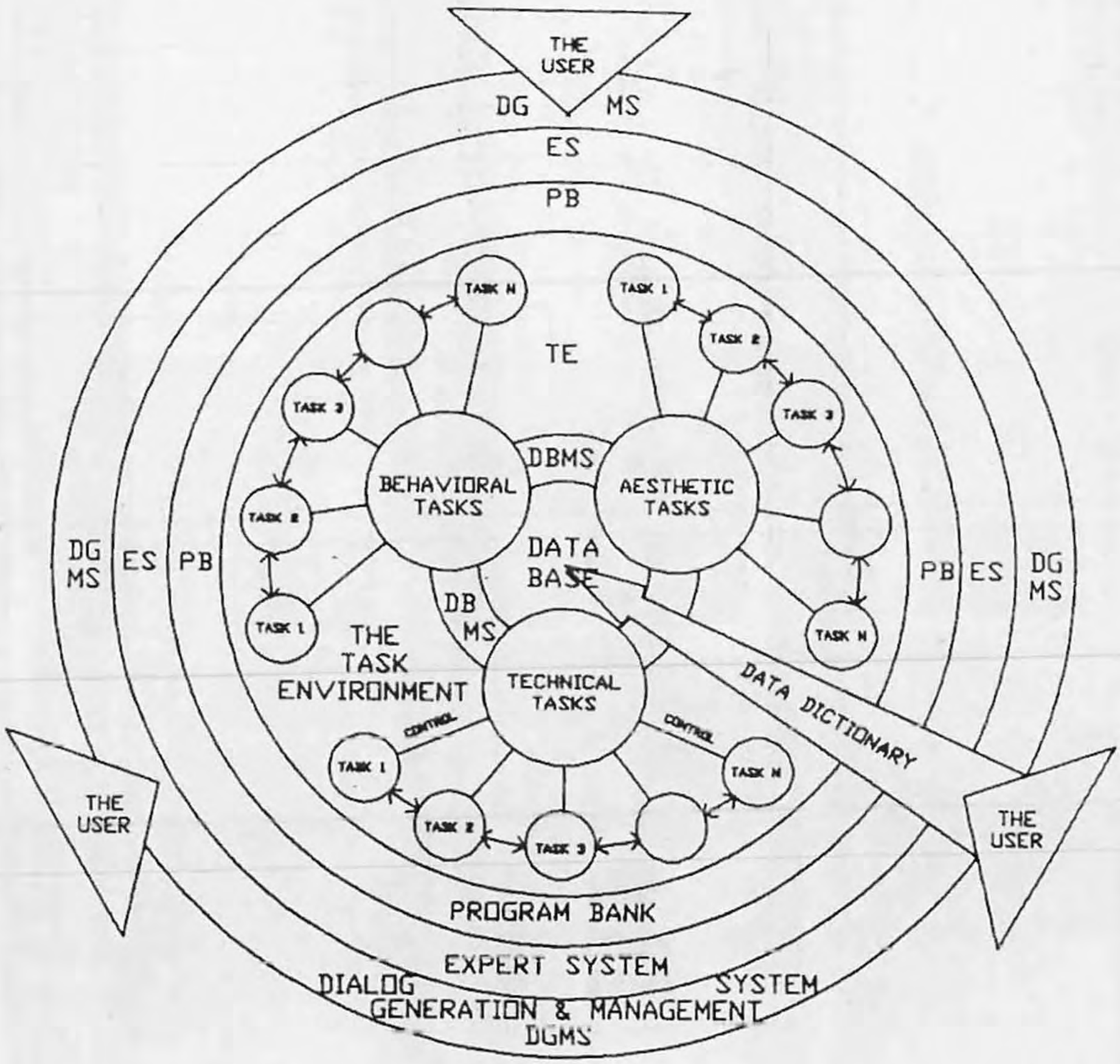
The decision-making system will constitute the three factors mentioned early in this chapter, that are involved in the decision making process. The architects/designers (users) will use a decision support system (the DSS) within a certain (task environment).

The framework is intended to be a complete intelligent architectural design assistant. The framework will give the architect a clear picture of every phase in the ADP, he will be able to examine all possible design alternatives. At the same time, consequences of various aspects of every design alternative should be easily available to him. Farther more, the framework will assist the architect in making accurate decisions with regards to these alternatives.

The conceptual diagram for the needed flexible decision support system for design can now be represented as shown in figure (5.9). It shows the different building blocks presented from within out; at the center there is the database and the database management system, then there is the task environment, the program bank, the expert system and the dialog generation and management system for managing the interface between the different users and the system.

5.4.1. The Communicator or the Dialog Generation and Management System (DGMS):

Much of the power, flexibility and usability characteristics of a decision support system derive from capabilities in the interaction between the user and the system, or the user interface.



DB= Data Bank PB= Program Bank

DBMS= Database Management System ES= Expert System

TE= Task Environment DGMS= Dialog Generation and Management System

Fig.(5.9):The proposed model for the decision making system.

The "ICAAD.DSS" Model.

The tool for managing the man-man and man-machine will be a Dialog Generation and Management System (from now on will be referred to as DGMS). It will transform, structure and retrieve the needed data for use by the different users at the different phases of the ADP. The system will offer all the needed information and tools for the design team to proceed with the design in a smooth way. All the system's capabilities are articulated and implemented through this component; the DGMS.

The desirable capabilities of this component will include: i) the ability to handle a variety of styles, with the ability to shift among them at the user's choice, ii) the ability to accommodate user actions with a variety of input devices, ii) the ability to represent data with a variety of formats and output devices, iv) the ability to provide flexible support for the user's knowledge base. The main goal here is to allow a more human dialog or communication to take place between the user and the system.

This communication is achieved and controlled by the Dialog Generation and Management System or the Communicator through using the concept of a "blackboard architecture."

The Blackboard Architecture:

The communicator controls the entire system . There are many levels to the system as described before. The architecture that best fulfills the requirements of monitoring and controlling a multi-level system is that of a blackboard. The blackboard architecture is layered, so that the development of different versions of a design can be traced and different refinements of partial design can be compared.

This architecture is described by Hayes-Roth (1983) from whom the following description is quoted:

> "The blackboard architecture has four definitive elements: a) entries, which are intermediate results generated during problem solving: b) knowledge sources, which are independent, event-driven processes that produce entries; c) the blackboard, which is a structured, global database that mediates knowledge source interactions and organizes entries; and d) an intelligent control mechanism, which decides if and when particular knowledge sources should generate entries and record them on the blackboard. New "solution islands" appear and existing islands grow wherever the opportunities are most promising. Eventually, mutually supportive partial solutions merge to form a complete solution."
>
> (Hayes-Roth 1983)

"Entries" on the blackboard are design units (blocks, components, etc.) suggested as refinements of a particular level of representation. The "knowledge sources" are sets of production rules which govern the search for or design of design units, and their combination into partial designs. Examples of such rules might be relevant to one level of representation or concern design data from more than one level.

> "...knowledge sources frequently transform entries at one level of abstraction into entries at another level, some knowledge sources...operate bottom-up. They aggregate several lower level entries into a smaller number of higher level entries. Other

knowledge sources operate top-down...(Yet others) operate within a single blackboard level or between different blackboard "panels" (subdivisions of a single level of representation). Thus, the blackboard architecture can combine knowledge sources embodying different inference mechanisms in a single problem-solving system."

(Hayes-Roth 1983)

The "blackboard"itself serves in separating the different kinds of knowledge source from each other, while allowing them to communicate and influence each other via blackboard entries. In addition, the blackboard will serve in maintaining data consistency and data integrity, by considering only the information entered on the blackboard. The blackboard will also help organize all partial and complete solutions generated for a particular problem by marking the relationships of the different entries to one another. The "control mechanism" or the controller is responsible for governing the blackboard activities. It is basically a scheduler which looks at a list of "knowledge source activation records" to decide which action to trigger next. The controller will examine the range of possibilities for action and take a truly global view of the activities of the system (Begg,1984).

Amongst the points that the controller will be watching for are: Where are the big searches? What program knows the heuristics to control these searches? Where are the big computations? How to reduce them or shift them for other less time-consuming methods? These are meta-rules, they are rules about using other rules, reasoning about appropriate computations. The blackboard control will enable the knowledge bases pertaining to each level of representation to

be called into action when that level is accessed by a designer.

The blackboard architecture provides a meta-level control for the system, monitoring the use of the design tools and ensuring that the relevant information is being provided from an appropriate source to each one of the different users. It also provides unlimited access to design data through its multi-dimensional representation scheme. In addition, it keeps track of the different versions of partial designs, and coordinates the activation of various sources of knowledge through a meta-level control mechanism (Begg,1984).

5.4.2. The Database or the Databank (DB):

Data or information needed for decision making comes from different sources. Decision making is dependent on external data sources such as economic or demographic data or even some special needs dictated as data by the client, as well as internal data sources, such as the basic information needed for the design to be accomplished, and these are material libraries, building codes and regulations, catalogs of parts, building products data, etc.

The database structure must be integrated to serve all functions of the design process, as well as to serve all levels of information needs at different design phases. Data must be classified according to their use. Operational distributed data for the three phases of design viz. the preliminary design, the design development and the design production must be extrapolated and summarized with respect to their level of application.

A three leveled database:

Integration is achieved through a single integration point, the data dictionary. The definitions of all the

components are entered into a single dictionary. Creating a specific system out of the environment (models, records, messages and screens) the dictionary is updated with all the relationships established between them. Once defined in the dictionary, any entity by declaration is known to any and all other components. Since the dictionary is active, all the relationships between entities are dynamically maintained. The communication between the different components is achieved by a single definition point. Redefinition then, is not required, and the associated opportunities for inconsistency errors can be eliminated (Kalisperis, 1988).

As mentioned above, data or information must be classified according to their use in the different phases of design, viz. preliminary design, design development and design production; as top level, middle level and bottom level respectively. A distributed database that is capable of handling the requirements for these three levels of data, could be built on a moving file concept. As operating data are received, appropriate adjustments will be made automatically through the decision support system to the suitable higher levels (Thierauf, 1988); the moving database will reflect not only the small detailed operating activities at the bottom level for the data, but also the strategic or planned performance of these same activities at the other two levels as well.

The lower-level databases (needed for the design production phase), feed the middle-level, where data are summarized to reflect a more aggregated state (in the design development phase), and are in turn fed to the top-level databases (preliminary design phase) for producing conceptual information, where the need for data abstraction is high (Thierauf, 1988 and Kalisperis, 1988).

The idea here, is that the process can repeat itself in the opposite direction, i.e. it can go either ways: from top to bottom, or from bottom to top. This orderly arrangement of the database allows the user at the top-level to ask "what-if" kinds of questions, in order to determine if the whole concept of the design or part of it can be improved, or if strategic or major changes should be made.

To operate the distributed moving databases in an efficient manner, it is necessary to develop an organization-wide (environment-wide) data element dictionary/directory that includes standard definitions, an index system, mathematical routines for deriving higher levels of data from the stored database elements, and security rules for files' access, protection and data abstraction. The dictionary/directory should permit only critical data to be referenced in memory or auxiliary storage devices, while less critical and more voluminous files are retrievable from magnetic tape or some other storage medium (Thierauf 1988). The data dictionary is a facility for defining all system entities, their attributes and their relationships (Kalisperis, 1988).

As illustrated in fig. (5.10), the lower-level of the database for the detailed design phase can be referenced by the data directory as well as the top-level of data for the preliminary or conceptual design phase. This approach permits large on-line files to be handled on an interactive basis from the lowest to the highest level, where data items can be stored, manipulated and chosen according to many criteria (Thierauf, 1988). Meanwhile, thought must be given to growth that can be anticipated in volume of records. In general, the insertion of additional or new records in the databases should not upset the file organization to the point that it will be necessary to reorganize them at frequent intervals.

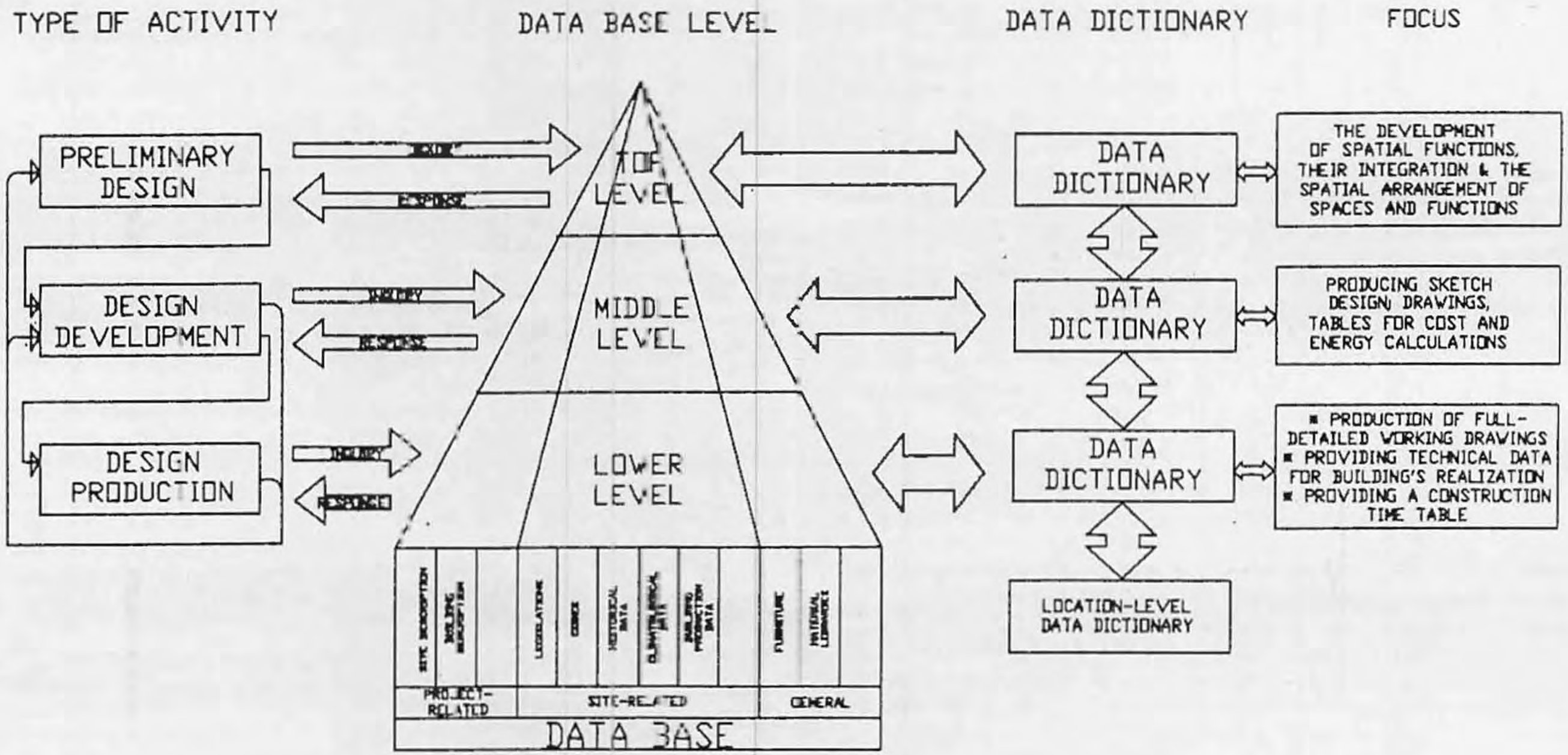


Fig. 15.10): The Dictionary/Directory System

The Moving/Bidirectional Database

Different types of needed databases:

It is important to know that the database has to support not only a model of a design solution, but also the communication between the different designers, the client, and the environment, not to mention obvious connections with the manufacturing and construction site as well.

According to (Marvin, 1985) the two principal influences on architectural design decision making are; published information and experience derived from education and practice.

Published information could be divided into three main parts (Hosny et al, 1990):

1. Project-related databases, or internal databases, these are the data specific to a certain project, like: site and building descriptions.

2. Site-related databases, or external databases containing information relevant, but not specifically related to the project, but rather specific to the site of the project, like: microclimate, design legislations and codes, historical data, building performance data and catalogs of standard parts, cost and time values, etc.

3. General or generic databases, these are data generic to all kinds of projects, like: data associated with furniture, or even material libraries. Or data generic to all similar kinds of projects, like room layouts in hotels, hospitals or dormitories. Or data dictated by site-neighbor relations regarding a specific style or character.

5.4.3. The Database Management System (DBMS):

A great deal of time is taken up in the architecture activity in looking for, and collecting information or data. The term database is applied to any structured set of information, generally stored in a computer and amenable to computer processing. The database management system (DBMS) is a self-contained program that manipulates databases to extract information. It will allow the storage of large quantities of data in a manageable format. Instead of sifting through the different databases, the DBMS has the ability to search them by combinations of different items to find all instances of records in the database that contain all the nominated descriptions. DBMS are useful for large amounts of data and handling difficult queries which could be scattered through several works, which is the case in the architectural design practice.

DBMS are used to create, maintain and sort different files, perform computations upon different fields, process queries, request information and produce reports in a generalized fashion without requiring changes to the programs in the system. In other words, the DBMS is not only effective for generating and maintaining a wide variety of routine management and operating reports, but also adaptable to meeting the new and emerging requirements to answer a myriad of "what if" questions. This capability means that the database management systems are important aids to users seeking to explore and understand new relations among various data elements, besides they minimize redundancy of data elements, by using only one data file to serve the different users. As a result of this single-record concept, a transaction need only to be entered once to be accessible for all the users (Thierauf, 1988).

Thê design of the DBMS for the ICAAD.DSS, should include the following functions: 1) creation and deletion, 2) dictionary, 3) updating, 4) query or retrieval, 5) sharing, 6) protection, and 7) recovery.

5.4.3.1. The creation and deletion operations support addition and subtraction of different objects in the database.

5.4.3.2. The dictionary operations are used to create the data in the database. Like a library's card catalog, the database dictionary supports adding new entries, deleting entries, retrieving information on the entries, and maintaining multiple indices. The dictionary operations are integrated with the other functions, for example, deleting an item from the dictionary should result in deleting it from the database too.

5.4.3.3. The update operations permit values to be replaced in the database with the entry of a single transaction. Such operations are important for data accuracy. Data updating is done according to the three data levels discussed earlier in this chapter.

5.4.3.4. The query or retrieval operations are used to select and manipulate records and fields from the database. They form the basis for the creation, deletion, and the updating operations. These operations are usually what distinguish a certain DBMS from another.

5.4.3.5. The sharing operations of the DBMS allow different interdisciplinary users to access the system at the same time. The DBMS also provides locking functions to prevent users from accessing inconsistent data and preventing "deadlock" (preventing each other from proceeding). Different locking levels are discussed in Blasgen et al., (1981). Level 1 locking allows the user to retrieve but not update the data

that is currently being updated by another user. Level 2 locking does not allow access to a record unless no other user is updating that record, this assures that the current data values are the most recent ones. Level 3 locking ensures that every retrieval of a record during a user-defined transaction (period of the lock) yields the same value, i.e., no other user can update the record during the transaction.

5.4.3.6. The protection operations are used to prevent data corruption, whether intentionally or not. A user will have access to other user's files in the database, as well as other user's models or objects through reading only. If he is to add his own files to that model or object, he will have to add on top of it, by using different layers.

5.4.3.7. The recovery operations are used to provide the ability to recover any hardware or software failures. This is done by using the check-points concept, and that is the DBMS frequently saves the data at different points or intervals on separate files. In case of any failure, these files are automatically accessed and the database can be recovered to the most recent checkpoint.

5.4.4. The Program Bank (PB):

An assortment of already existing computer programs and/or program packages are available in the system to aid, automate and accelerate a wide variety of tasks in the architectural practice. Many of these applications have been available for decades, others are relatively new. Most are continually being refined and improved as computer technology improves.

It is important to point out here that these existing application programs were developed independently, outside a comprehensive framework, they were presented individually as

computational aids for architectural design and they did not exist in an integrated environment. Integration of these different program packages will be achieved through a single integration point, and that is the data dictionary, as will be discussed later on in this thesis.

The program bank is seen to encompass different computer programs to help in the various activities encountered in an architectural design office, starting from all that happens before a project starts till it is fully completed. These programs are classified under the following main categories: general management and project control, accounting and cost control, site planning and land use studies, environmental control and building services, plan and layout analysis, graphics or drafting packages and programs for checking the completeness and the consistency of the information.

5.4.5. The Expert System Component:

According to Marvin (1985), the second principal influence on design decision making is the "experience" derived from education and practice. In addressing this issue, the proposed system is seen to be an intelligent one, in the sense that it is expected to provide expert advice to the user during the evolution of the design solution. Several expert systems may operate concurrently supporting different design functions. Accordingly, the databases must provide support for the integration of the knowledge bases into its operations.

As already mentioned before, the dialog component along with its management system, supports the use of the system by the different decision makers, and the data component along with its management system, provides access to the raw material for the decision making.

The data component supports descriptive operations or activities which support primarily the intelligence phase of decision making.

It is the expert system component that will provide the analytical powers of the system and will assist in the prescriptive activities of decision making, by suggesting different options as solutions, whether to continue, or to feedback to a certain step in the BASED process, this is done by triggering the feedback mechanism in the system to return to the appropriate step. This process could be done automatically by the system, or could be done manually by the user. It is up to the user to either accept the system's default and feedback to a certain step in the process, or reject it and build upon his own perception and evaluation applying his own cognitive style.

The automatic feedback mechanism is achieved by using the (HDF) or the heuristic-depth-first search algorithm (Lee and Kwon, 1989). This searching algorithm regards the feedback and the redesign activity as a graph-searching problem, it is basically similar to depth-first strategy but is augmented by being guided by heuristic rules. Heuristics are rules of thumb, they are criteria, methods or principles for deciding which among several different courses of action promises to be the most effective in achieving a certain goal. In addition, heuristics indicate a way to reduce the complexity of the design problem by reducing the number of evaluations in order to obtain solutions within a reasonable time frame. The suggested expert system here utilizes these heuristics, couples symbolic reasoning capabilities with numerical analysis approaches, and works either automatically or as an assistant to the designer. Its objective is to find satisfactory designs, which satisfy certain design criteria, i.e. to find the most "satisficing" solution, rather than to look for an optimum design.

In the feedback process, based on the evaluation and decision results, the expert system collects and sorts all the possible improving suggestions (rules), stores them into a stack, and always pops out the first suggestion from the stack to improve the current design. If the suggestion should not give a satisfactory solution, which might be caused either by the constraint violation or by finding out that the design is not improved, the system can backtrack to any previous design and try the next suggestion popped out from the stack (Lee and Kwon, 1989).

For an unsatisfactory design, there are always several actions for its possible improvement. The priority of each action is stored as a "priority factor" associated with its rule, this priority factor provides important information for triggering the rule. In other words, the information stored in the rule base must include what actions are possible to improve the design, as well as which action might have the most prominent effect on improving that design. For this reason, it is not necessary to try out and evaluate all the possible suggestions through the different analysis before the system can pick out the most "satisficing" one as the redesign step to feed it back, which will help in lowering the high cost of executing an extensive computation for the evaluation of each individual design solution (Lee and Kwon, 1989).

It is also important to point out here that the feed back mechanism of the system does not have to follow a certain sequential pattern. This will enable the architect to access the BASED process at any desired step, he can return to any part in this process at any time. In addition, he can freely explore the whole process and bring about any desired changes in the design or even review any previous decisions or activities. The designer's tool for doing this is the browser, as will be discussed later.

One of the important parts in the expert system component which is useful for supporting many of the activities of the users; is the model subsystem or the modeling component.

The Modeling Component:

Models are simulations of the real world. They are either; static models, simulating the real world at a given point in time (e.g. an architectural plan), or dynamic models, simulating the real world seen over a period of time and allowing a study of the consequences of actions (Beheshti and Monroy, 1987).

A very important aspect of the decision support system is its ability to integrate data access and decision models. It does this by embedding the decision models in an information system that uses the data base as the integration and communication mechanism between the different models.

The modeling component in the system consists of permanent models, ad hoc models, user-built models, models for operational, tactical, and strategic decision support, and models to support a variety of tasks and analysis approaches. These range in size from very small to very large, with some of the smaller ones acting as model " building blocks" to support the construction of other models (Sprague and Carlson, 1982).

The modeling component in the system has the ability to: create new models quickly and easily, access and integrate the model's building blocks, maintain a wide range of models supporting the different users, interrelate these models with appropriate linkages through the database, and manage the models with management functions analogous to database management (e.g., mechanisms for storing, cataloging, linking and accessing models). In short, the modeling component

supports activities that emphasize the design and choice phases. In addition, it allows the different users and decision makers to support these kinds of activities directly. The support for these activities depends on feedback and interaction between the user and the expert system which allows the examination of intermediate results, accommodation of subjective judgment during the problem-solving process, and modification of the objectives if the user's perception of the problem changes.

Integration of this component with the other components is seen to be of crucial importance. Its direct integration with the dialog generation and management system gives the user direct control over the different operations, manipulations and the use of models. It is seen that, it is this integration that gives the user the ability to do true interactive modeling, instead of just running a model in an interactive environment. The decision maker can interrupt the model, run model segments in a variety of sequences, explore, test, probe the nature of a problem and its solution, change parameters and even change objective functions in response to intermediate results if necessary, using the full range of data formatting and display capabilities.

In addition, the direct linkage and integration of the expert system component with the database and the database management system will update the models (through the modeling component in the expert system) as the data values are updated, modified or restructured. With the model output feeding-back to the database, accuracy can be achieved, because the database values are validated and checked by the DBMS. Farther more, consistency is also achieved by having all the models draw on the same database. If more than one model uses the same data item, it will be the same for all the users, output of a certain model can be used as an input for another without having the trouble to compile it. It also

facilitates the updating of the models, as data are updated, all models based on these data are updated too. On the other hand, it becomes so easy to display, format or present any output from a model.

In this chapter, the "ICAAD.DSS" conceptual model has been presented. The concepts of the moving, bi-directional database, as well as the data dictionary/directory system as integrating tools, are seen to be of great importance. The following chapter will discuss the data structure for the ICAAD.DSS model, and farther address these two concepts.

REFERENCES FOR CHAPTER 5:

Archea, John.:

"Architecture's Unique Position Among The Disciplines, Puzzle Making vs. Problem Solving." In The Design Process, pp. 20-22, September,1985.

Bax, M. F.:

"Domain Theory: Applications for CAAD." In Open House International, Vol.11, no.2, 1986.

Begg, Vivienne.:

Developing Expert CAD Systems. New York: Unipub, 1984.

Behesht1, M. and Monroy, M.:

"Requirements For Developing An Information System For Architecture." In CAAD Futures '87. Proceedings of the Second International Conference on Computer-Aided Architectural Design Futures, Eindhoven, The Netherlands, 20-22 May 1987. Edited By Tom Maver and Harry Wagter. New York: Elsevier, 1988.

Blasgen, M. W., et al.:

"System/R: An Architectural Overview," in IBM systems Journal, Vol. 20, No. 1, 1981, pp. 41-62.

Bucciarelli, L.; Goldschmidt, G.; and Schon, D. A.:

"Generic Design Process in Architecture and Engineering." In Proceedings of The 1987 Conference On Planning And Design In Architecture. Boston, Mass., August 17-20, 1987. Edited by J. P. Protzen. New York, The American Society Of Mechanical Engineers, 1987.

Cross, Nigel.:

Developments in Design Methodology. New York: John Wiley & Sons, Ltd., 1984.

DeSanctis, G. and Gallupe, B.:

"Group Decision Support System: A New Frontier." In Decision Support Systems, Putting Theory into Practice, edited by Sprague and H. Watson, pp. 190-201, Englewood Cliffs: Prentice-Hall, Inc., 1986.

Habraken, N. J.:

"Control Hierarchies in Complex Artifacts." In Proceedings of The 1987 Conference on Planning and Design in Architecture. Boston, Massachusetts, August 17-20, 1987. Edited by J. P. Protzen. New York, The American Society Of Mechanical Engineers, 1987.

Haider, J.:

A conceptual framework for Communication-Instruction in Architectural Design. Ph.D. Dissertation, The Pennsylvania State University, 1986.

Hayes-Roth, B.:

The Blackboard Architecture: A General Framework for Problem-Solving? Technical Report, HPP-83-30, Computer Science Department, Stanford University, 1983.

Heath, T.:

Method in Architecture. New York: John Wiley & Sons, Ltd., 1984.

Hosny, S.; Sanvido, V.; and Kalisperis, L.:
A Framework for an Integrated Computer-Aided Architectural Design Decision Support System. Unpublished paper, The Pennsylvania State University. January, 1990.

Kalisperis, Loukas, N.:
A Conceptual Framework for Computing in Architectural Design. Ph.D. Dissertation, The Pennsylvania State University, 1988.

Lee, H. and Kwon, T.:
"Heuristic Redesign and Rule Formulation for Complex Engineering Designs." In Preprints of the NSF Engineering Design Research Conference. University of Massachusetts, Amherst, June 11-14, 1989.

Marvin, Heather.:
Information and Experience in Architectural Design. Research Paper no. 23, Institute of Advanced Architectural Studies, University of York, The King's Manor, York, Great Britain, 1985.

Newell, A., and Simon, H.:
Human Problem Solving. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1972.

Rittel, H., and Webber, M.:
"Dilemmas in a general theory of planning". In Systems and Management Annual, pp. 219-233, edited by R. Ackoff. New York: Petrocelli, 1974.

Robbins, E.:
"An Anthropology of Architecture: Some Preliminary Suggestions." In Proceedings of the 1987 Conference on Planning and Design in Architecture. Boston,

Massachusetts, August 17-20, 1987. Edited by J. P. Protzen. New York, The American Society Of Mechanical Engineers, 1987.

Simon, H.:
"Style in Design." Proceedings of the Environmental Design Research Association Conference, Pittsburgh: Carnegie-Mellon University, Department of Architecture, 1970.

Sprague, R. H. and Carlson, E.D.:
Building Effective Decision Support Systems. Englewood Cliffs, New Jersey: Prentice-Hall, 1982.

Thierauf, R. J.:
User-Oriented Decision Support Systems. Englewood Cliffs, New Jersey: Prentice-Hall, 1988.

Wade, John, W.:
Architecture, Problems and Purposes. New York: Wiley-Interscience, 1977.

Watson, Donald.:
"Model, Metaphor and Paradigm." Journal of Architectural Education, vol. 37, no. 3-4, Spring and Summer, 1984.

Wheeler, B. J. Q.:
"A Unified Model for Building," In Computer-Aided Architectural Design Futures, edited by A. Pipes, London: Butterworths, 1986.

CHAPTER 6

THE DATA STRUCTURE FOR THE "ICAAD.DSS" MODEL

CHAPTER 6

THE DATA STRUCTURE FOR THE ICAAD.DSS MODEL

6.1. Information Levels for the Architectural Design Process.

6.1.1. Information System Abstraction.

6.1.2. Data Abstraction.

6.1.3. The Concept of Metadata.

6.1.3.1. The Data Dictionary/Directory System.

6.1.3.2. Relationship Between Dictionary and Directory Metadata.

6.1.4. Active versus Passive DD/DS.

6.1.4.1. Active DD/DS.

6.1.4.2. Passive DD/DS.

6.1.5. Why a Data/Dictionary Directory System?

6.2. Data Models.

6.2.1. Record-Based Logical Models.

6.2.1.1. The Hierarchical Model.

6.2.1.2. The Network Model.

6.2.1.3. The Relational Model.

6.2.2. Object-Based Logical Models.

6.2.2.1. The Entity- Relationship (E-R) Data Model.

6.3. Which Data Model Best Represents the Architectural Design Problems?

6.3.1. Object-Oriented Databases.

6.3.2. Advantages of Object-Oriented Programming for the Architectural Design Process.

6.4. The Moving/Bidirectional Database Concept.

6.4.1. The Definition of the Moving/Bidirectional Database Concept.

6.5. The Internal Representation of a Design Abstraction.

6.5.1. The Preliminary Design Phase.
6.5.2. The Design Development Phase.
6.5.3. The design Production Phase.

CHAPTER 6

THE DATA STRUCTURE FOR THE ICAAD.DSS MODEL

6.1. Information Levels for the Architectural Design Process:

The information needed for the architectural design process can be described as belonging to three levels of abstraction; top, middle and lower levels, analogous to the three phases in the design process; viz. the preliminary, the design development and the design production phases (Hosny et al, 1990).

6.1.1. Information System Abstraction:

In the top level of abstraction or the preliminary design phase, the needed information is heuristic, in order to reduce the solution space into a more manageable space so that a small number of design solutions could be developed and evaluated in the next level. This reduction or "satisficing" usually concerns the adopted concept, or the "a priory" solution, and whether or not the different design alternatives meet the specific design criteria, and design constraints, dictated by the program. Models that are used in this design phase, rely greatly on heuristics, grammars, and rules, to help in the generation of design alternatives. Evaluation programs such as energy calculation, cost estimation, and simulation packages are useful in this design phase as well.

In the middle level, which is the design development stage, there is another level of data or information needed, which is more detailed, and is helpful in the development of the different design solutions established in the previous design phase. Models used in this design phase may include; energy analysis, cost/function analysis, simulation packages, and drafting packages including wire frame, perspective

generation, and solid modeling capabilities, these will help the architect in his decision making, as well as visualizing his ideas in a much realistic form.

On the lower level or the design production phase, the needed information is repetitive, frequently produced, and frequently accessed. This type of information (such as product types, product standards, performance standards, product details and catalogs) is the most amenable to automation in an organization.

In this stage of design, the designer considers the appearance as well as the function of every element in more detail, and begins to select specific materials, models and manufacturers. He starts giving exact dimensions, studies the behavior and the fixing method of each item to determine whether it will fit into the overall scheme. The availability and supply of products, their maintenance and replaceability, are considered in this design phase as well. Drafting packages are considered to be the best application programs that are used in this design phase.

6.1.2. Data Abstraction:

Analogous to the three information levels needed for the three design phases in the architectural design process, the definition of three levels of abstraction for the database follows:

The view level:

This is the highest level of abstraction at which one describes only a part of the entire database. Despite the use of simpler structures at other data levels, there remains a form of complexity resulting from the large size of the database (Silberschatz, 1986). In other words, by reducing

the complexity of a single record; its fields or attributes; this will increase the number of the records themselves in the database, because the reduced attributes have to be assigned to other records in other data levels in order to give a complete description of the entire database.

It is obvious, that many users will not be concerned with all of the information available in the system. Instead, they will only need a part of the database. This is also true in the case of a single user. For a particular phase in the design process, the user is only concerned with a specific portion of the data (his own view of the data) that is available for this specific design level. Later on, the designer may wish to explore more of the information as he proceeds to another phase of design.

The ability of having more than one view provided by the system for the same database (multiple representations), is helpful for the exploration of more design alternatives.

The conceptual level:

This is the middle level of abstraction at which it is possible to describe "WHAT" data are actually stored in the database, and the relationships that exist among data. This level describes the whole database in terms of a small number of relatively simple structures. Although the implementation of the simple structures of the conceptual level may involve complex lower level structures, the user of the conceptual level need not be aware of this.

The physical level:

This is the lowest level of data abstraction at which it is possible to describe "HOW" the data is actually stored, and

retrieved. At this level, complex low level data structures are described in detail.

The interrelationship among these three levels of data abstraction could be illustrated as in figure (6.1).

An overall view of the different phases in the design process, along with the corresponding views of the data abstraction levels could be seen as in figure (6.2).

6.1.3. The Concept of Metadata:

In order to manage the data that is shared by different users at all levels in the organization, it is essential that data about data be clearly specified, easily accessed and well controlled. This is known as "metadata" (Leong-Hong and Plagman, 1982), also known as "metaknowledge".

The first step in this process is to identify and describe the data objects that exist in a single project, such as the rooms, the walls, the floors, the ceilings, the doors, the windows and so on. These objects are called "entities". Descriptions of these objects or entities are identified and converted to computer readable forms, so that they can be stored, processed and made available to various user groups in the organization. The description or the definition of these data objects is achieved through the use of a data definition language (DDL). The processing is achieved through a data manipulation language (DML), and the access of the data is made available through the use of a data query language (DQL).

The second step in the process, is to identify certain data about these data objects in order to address issues like, what are the kinds of data available, where is it located, and how to retrieve it?

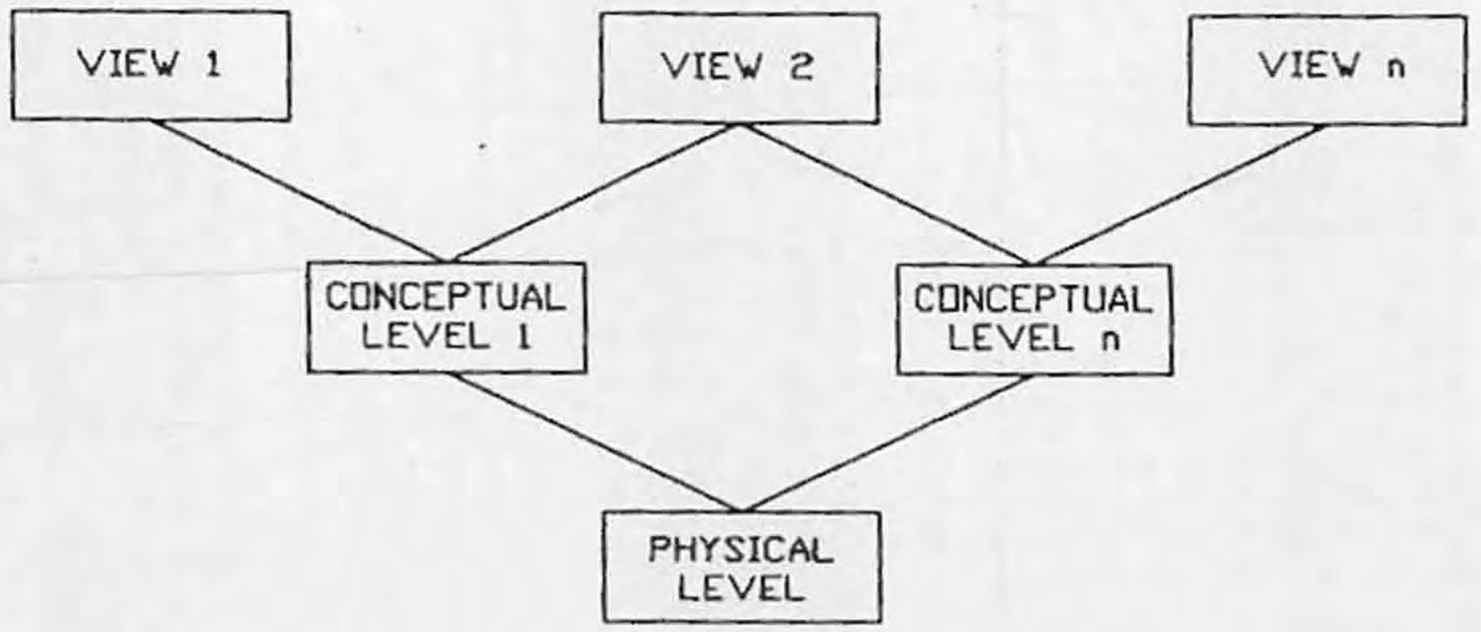


Fig. (6.1): The three levels of data abstraction.

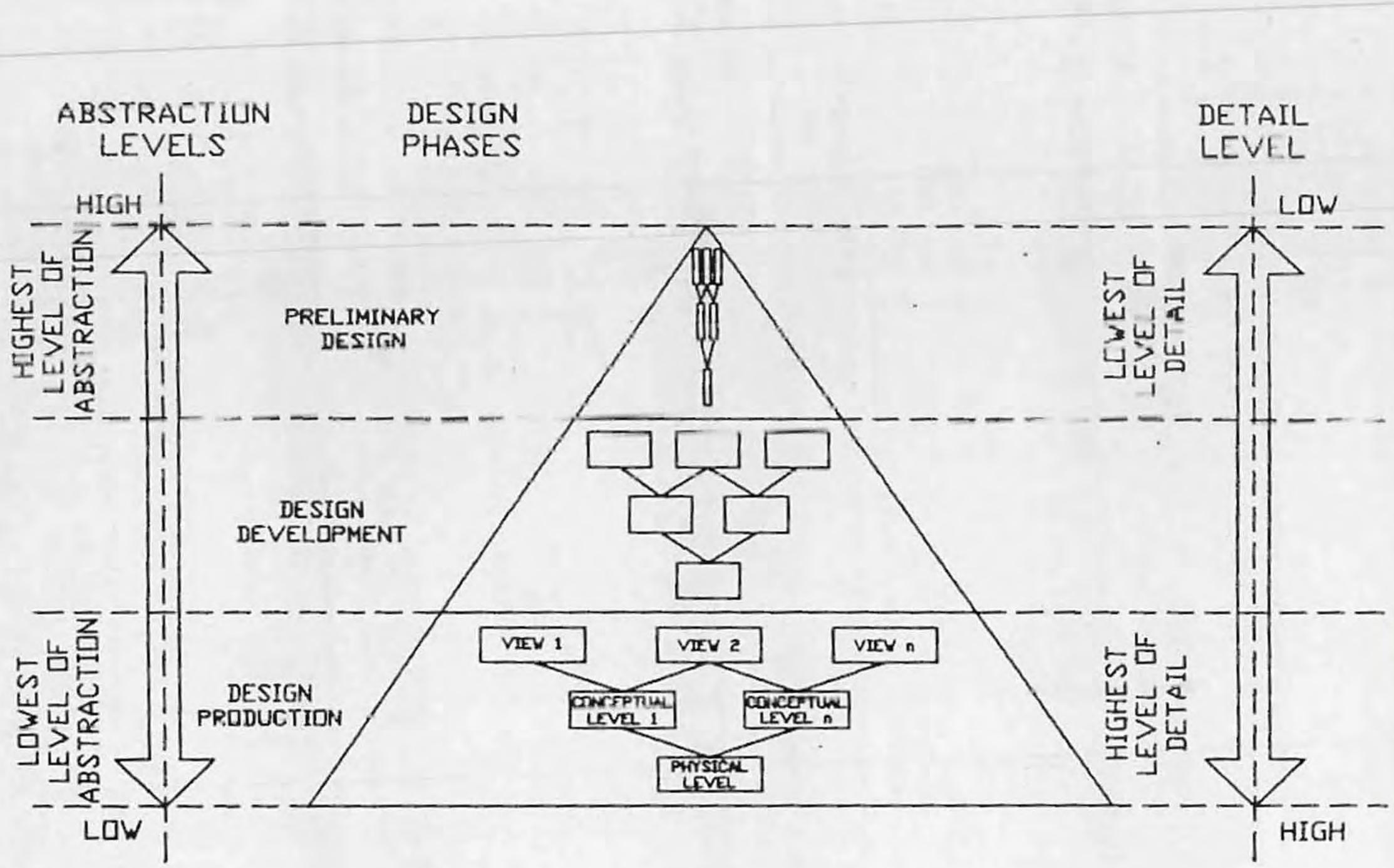


Fig. (6.2): The three levels of design abstraction.

In the database environment, these entities described above, are represented in the form of data objects such as data elements, records, or files. These data objects are called "metadata entities".

The data used to describe metadata entities is called "metadata", that is, data about the data. A collection of related metadata, when managed and controlled as a unit, is called a metadata database. It is referred to as "metadatabase".

6.1.3.1. The Data Dictionary/Directory System:

Some procedures are required for collecting the metadata at its source in order to help co-ordinate and control the metadata that is to be stored in the metadatabase. These procedures insure completeness by describing in detail: the type of metadata that is required, and the sources from which the metadata should be collected. In accordance, metadata falls into two general categories:

1) what the data is, or what does it mean, and
2) where the data can be found, and how it can be accessed.

The system that is needed to support the management and control of the metadata is called the Data Dictionary/ Directory system, or the DD/DS. The two categories listed above, are the main differences between the dictionary (what), and the directory content of the DD/DS (where and how) respectively.

The kinds of metadata that reside in the DD/DS are descriptive data. For example, they would include size, value range, type of characters, and relationship to other data entities. In other words, the Data Dictionary/Directory

System is a system that is designed to support comprehensively the logical centralization of data about data (metadata), as in figure (6.3).

Logical centralization means that the user's perception of the DD/DS is that it provides a centralized repository for the metadata. However, this does not prevent the ability of physical distribution of the metadata. Logical centralization also suggests that there is an organization wide coordination and control of metadata, which is a key requirement for the integration of the metadatabase management.

The DD/DS's data definition language can be used as the central mechanism for data description and definition throughout the organization. This would ensure that the metadata collected will be clearly, concisely and consistently defined. The DD/DS's maintenance facilities can be used to insure that the metadata will be updated consistently for applications using the metadata, and that only proper and authorized changes will be incorporated.

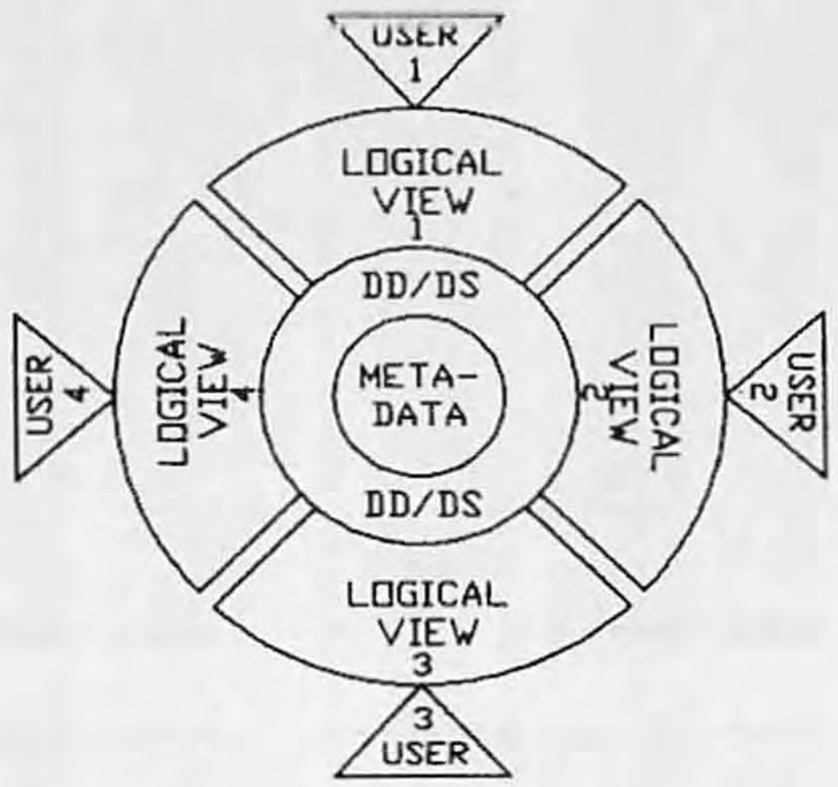


Fig. (6.3): The DD/DS Philosophy.

It supports the logical centralization of metadata.

6.1.3.2. Relationship Between Dictionary and Directory Metadata:

The relationship between the dictionary and directory metadata fall into one of these categories (Leong-Hong and Plagman, 1982):

1) Directory metadata is a subset of dictionary metadata.
2) Directory and dictionary metadata are mutually exclusive.
3) Directory and dictionary metadata are mutually interdependent.

In the first instance, the directory metadata actually becomes a subset of the dictionary metadata. This means that the physical view of the metadata is a subset of the logical view of the metadata.

By contrast, when there is mutual exclusivity, the dictionary and the directory metadata are completely independent of each other. In this case, the physical view is totally separate from the logical view, which is virtually nonexistent in the current technology.

In reality, dictionary and directory metadata are mutually interdependent, resulting in partial independence (or partial dependence), with some metadata falling into both categories. The metadata in the dictionary addresses features like; identification, source, description, etc. On the other hand, the metadata in the directory addresses features like; physical location, compaction rules, access modes, etc. The overlapping metadata could be used both; physically and logically. Among this shared metadata is the description of features like; length, value, security, etc.

With this definition of the DD/DS, it is neither practical nor desirable to separate the dictionary and the directory metadata in the context of metadatabase implementation, for various reasons. According to Leong-Hong and Plagman (1982), these include:

i) The introduction of redundancy, since it may be necessary to duplicate part or all of the metadata from the dictionary or the directory;
ii) with the duplication of metadata, the potential for introducing inconsistent metadata is a high risk; and,
iii) there is a need to coordinate the database design to support physical and logical mappings of metadata.

6.1.4. Active Versus Passive DD/DS:

The underlying reason for implementing a DD/DS is to gain greater control over the organization's data resources. To achieve this, a proper implementation strategy must be selected for integrating the DD/DS into the operating environment. An important aspect of this implementation strategy is the selection of a system's hardware design, or architecture framework under which the DD/DS can function effectively. The two basic system's architectural alternatives are : an active DD/DS and a passive DD/DS.

6.1.4.1. Active DD/DS:

The DD/DS can be used to actively control the use of the metadata, and by extension, can also actively control the database processing environment.

According to Plagman (1978): "A DD/DS is said to be active with respect to a program or a process if and only if that program or process is fully dependent upon the DD/DS for its metadata". Therefore a DD/DS is active with respect to a DBMS if the DBMS is solely dependent on the DD/DS for its schema definition.

Likewise, a DD/DS is only active in respect to an application program, if that application program is solely dependent upon the DD/DS for its data description.

6.1.4.2. Passive DD/DS:

By contrast, the definition of a passive DD/DS is based upon the relative lack of active control that is exercised through the metadata management. A passive DD/DS does not require that the process or the system components depend on the DD/DS for their metadata.

As a result, an active DD/DS can better serve the goals of managing data than can a passive DD/DS, and accordingly, it will serve in managing the data for our suggested framework for a CAAD decision support system. The reasons for this, according to Plagman (1982), could be summarized as follows:

i) Eliminating redundant metadata definition, by not having to redefine the data separately for each user, the active DD/DS will reduce tedious data definition work.

ii) Insuring metadata consistency, since the metadata can only be obtained from the DD/DS, all the users and the DD/DS will have consistent metadata, assuming that the DD/DS's content is itself consistent.

iii) Establishing control over metadata usage, since the users are dependent on the DD/DS for their metadata, the usages of the metadata are controlled. Non-authorized users can be blocked by withholding the metadata.

iv) Establishing control over metadata changes, by centralizing the source of metadata, all changes to existing data descriptions can be reflected consistently throughout the environment. Non-authorized changes can be controlled. (Stonebreaker, 1974).

6.1.5. Why a Data Dictionary/ Directory System?

In general, the data dictionary/directory system is considered to be a primary tool for metadatabase management, as well as being a useful tool for defining all system entities, their attributes and their relationships. In other words it is important for achieving an integrated environment for computing in architecture.

By entering the definitions of all the components into this DD/DS, it is possible to create a single database environment that could support the different views of the multidisciplinary users, refer to figure (6.3). Once defined in the DD/DS, any entity by declaration is known to any and all other components. Since the DD/DS is active, all the relationships between entities are dynamically maintained.

The DD/DS will act as a single definition point, thus redefinition is not required and the associated opportunities for inconsistency errors can be eliminated (Kalisperis, 1988). The database structure of such a flexible environment must be integrated to serve all functions of the design process, as well as to serve the different levels of information needs at

the different design phases as mentioned previously; viz. the preliminary, the development and the production phases of design. A distributed database that is capable of handling the requirements for these three levels of data abstraction could be built on a moving file concept, or a bidirectional (dynamic) database concept.

6.2. Different Data Models:

In order to describe the structure of the database, it is important to define the concept of a data model.

A data model is a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints. There are two kinds of data models that have been in use, those are the logical data model, and the physical data model. Under the logical data model, there are the record-based and the object-based logical data models. The logical data model illustrates the structure of the different building components or database entities, as well as the relationships between them, and the methods that could be operated on them. On the other hand, the physical data model is the level of implementation of the logical data model. The scope of this work will only cover the logical data models.

6.2.1. Record-Based Logical Models:

Record-based logical models are used in describing data that is pertinent for the phases of design development, as well as design production to support several views. These models are used to specify both; the overall logical structure of the database and a higher-level description of the implementation. However, they do not provide facilities for specifying data constraints explicitly. There are three data models that are most widely accepted, these are:

6.2.1.1. The Hierarchical Model:

Data in the hierarchical model is represented by a collection of records, and relationships among data are represented by links. In this model, the records are organized as collections of trees. A sample hierarchical database is shown in figure (6.4).

6.2.1.2. The Network Model:

The network model is similar to the hierarchical model in the sense that data and relationships among data are represented by records and links which can be viewed as pointers. The records in the database are organized as collections of arbitrary graphs. A sample network database is shown in figure (6.5).

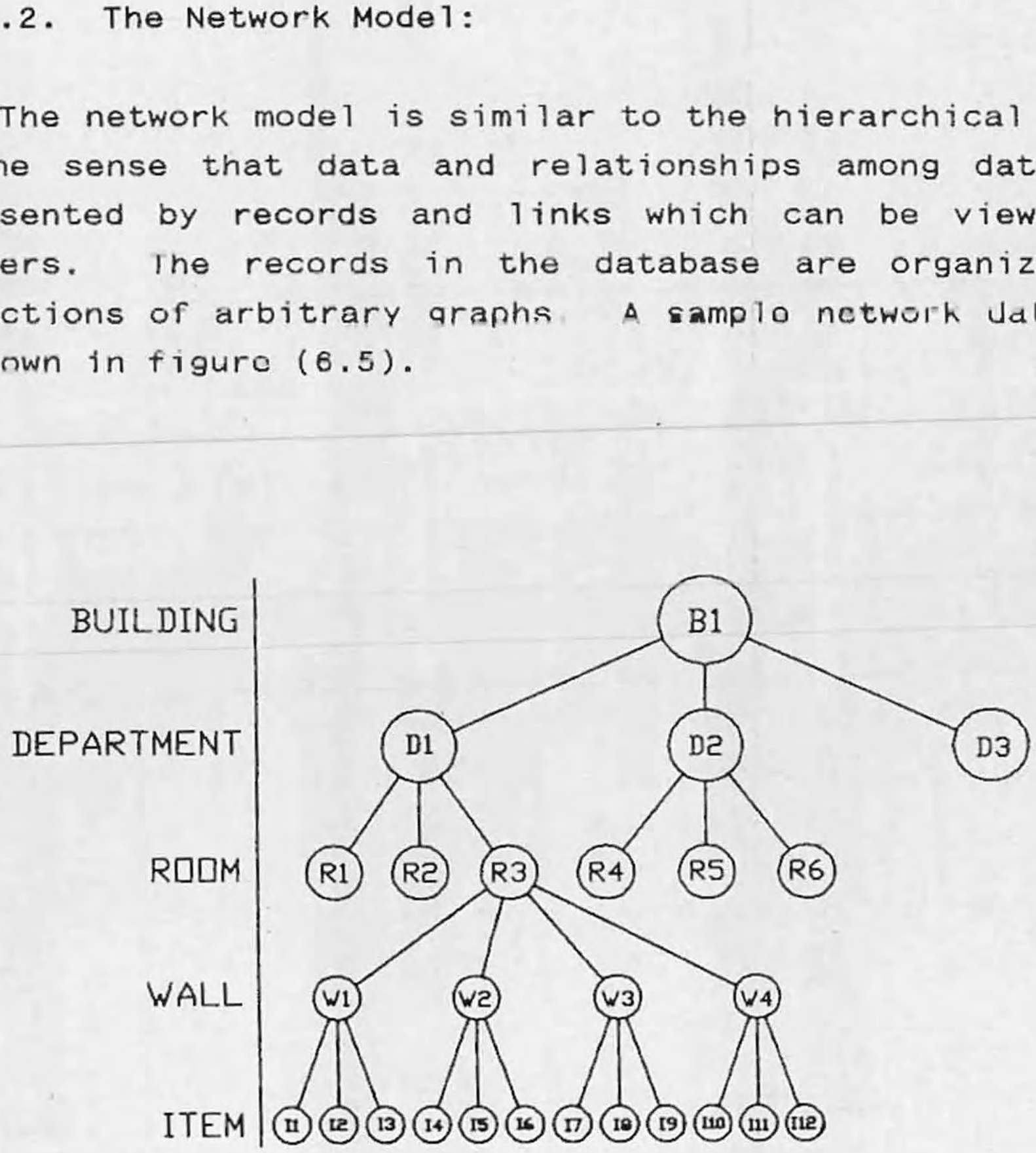


Fig. (6.4): A sample hierarchical database.
Each sibling node has a single parent node.

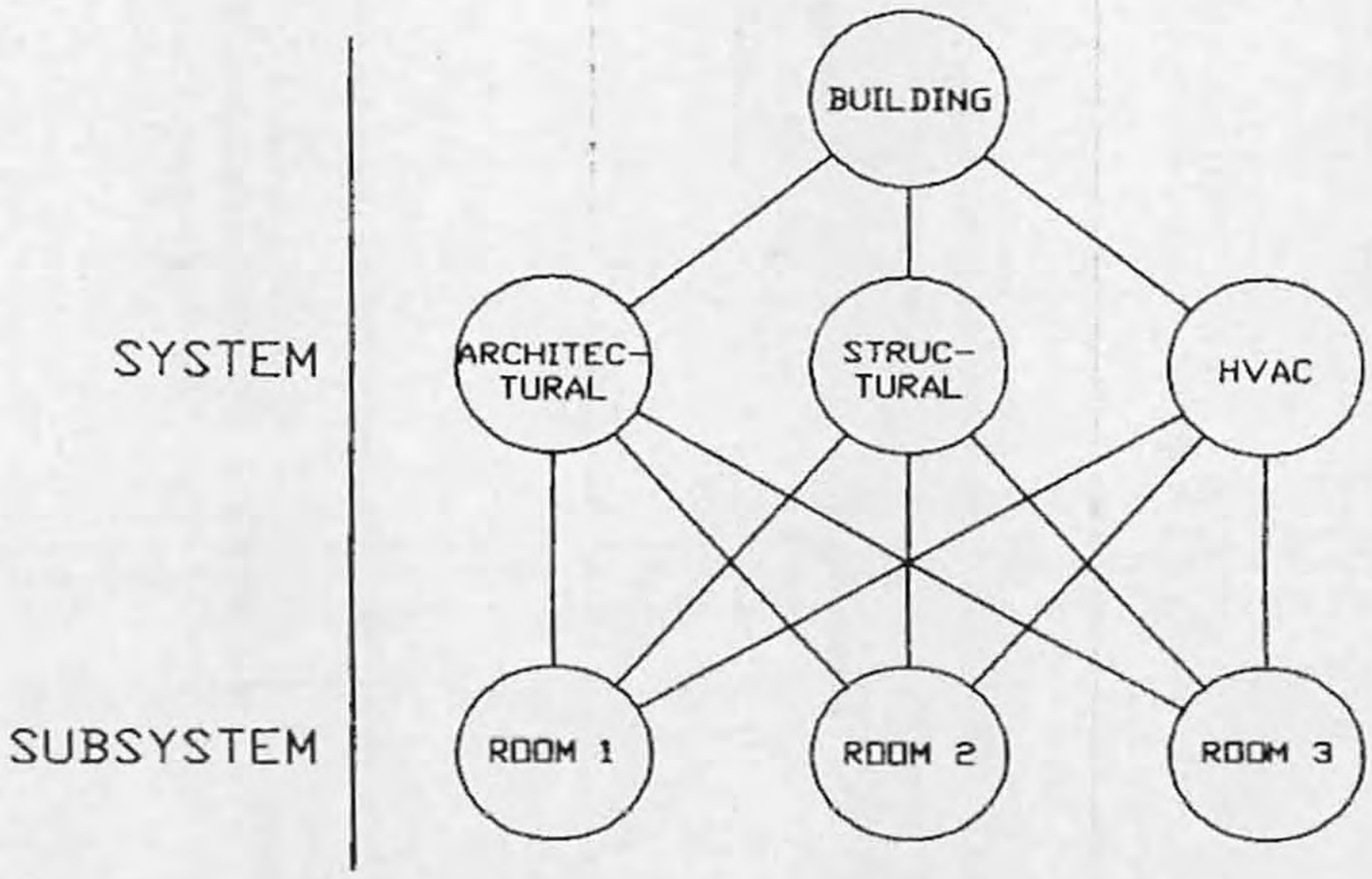


Fig. (6.5): A sample network database.

A sibling node has more than one parent node.

6.2.1.3. The Relational Model:

In this model, the data and the relationships among data are represented by a collection of tables, each of which has a number of columns with unique names. To illustrate this, an example of a relational database follows in figure (6.6):

SURFACES-ROOMS	
surf.	room
LP1	020
SP1	021
KP1	022
LP2	023

SURFACES				
id	type	code	explan.	area
LP1	floor	M1	Plastic	15
SP1	wall	T2	wallpap.	18
KP1	ceil.	AK1	met.	15
LP2	floor	LK1	nat.	55

ROOMS			
id	name	vol.	area
020	office	45	15
021	office	45	15
022	W.C.	12	04
023	corr.	150	55

Fig. (6.6): A room database from the relational database model.

(After Bjork and Penttila, 1989).

6.2.2. Object-Based Logical Models:

Object-based logical models are used in describing data at the design development phase as well as at the preliminary design phase. They are characterized by the fact that they provide fairly flexible structuring capabilities and allow the user to specify data constraints explicitly. According to Silberschatz (1986), there are at least thirty such different models, and more are likely to come. Some of the more widely known ones are: the entity-relationship model, and the semantic data model. Among these models, the entity-relationship (E-R) data model has gained acceptance as an appropriate data model for database design and it is widely used in practice (Silberschatz, 1986).

6.2.2.1. The Entity-Relationship (E-R) Data Model:

The (E-R) data model is based on a perception of a real world which consists of a collection of basic objects called entities, and relationships among these objects. An entity is an object that exists and is distinguishable from other objects. This distinction is accomplished by associating with each entity a set of attributes which describes the objects. For example, the attributes id, width, and height, describe one particular door. A relationship is an association among several entities. For example, a Door-Wall relationship associates a certain door in a particular wall. The set of all entities of the same type and relationships of the same type are termed an entity set and relationship set, respectively.

In addition to entities and relationships, the E-R model has the ability, over other data models discussed earlier, to represent certain constraints to which the contents of a database must conform. One such important constraint is

mapping cardinalities which express the number of entities to which another entity can be associated via a relationship set. These are one-to-one relationship, one-to-many, many-to-one, and many-to-many relationships (Silberschatz, 1986).

The overall logical structure of a database can be expressed generically in a graphical way by the E-R diagram as in figure (6.7).

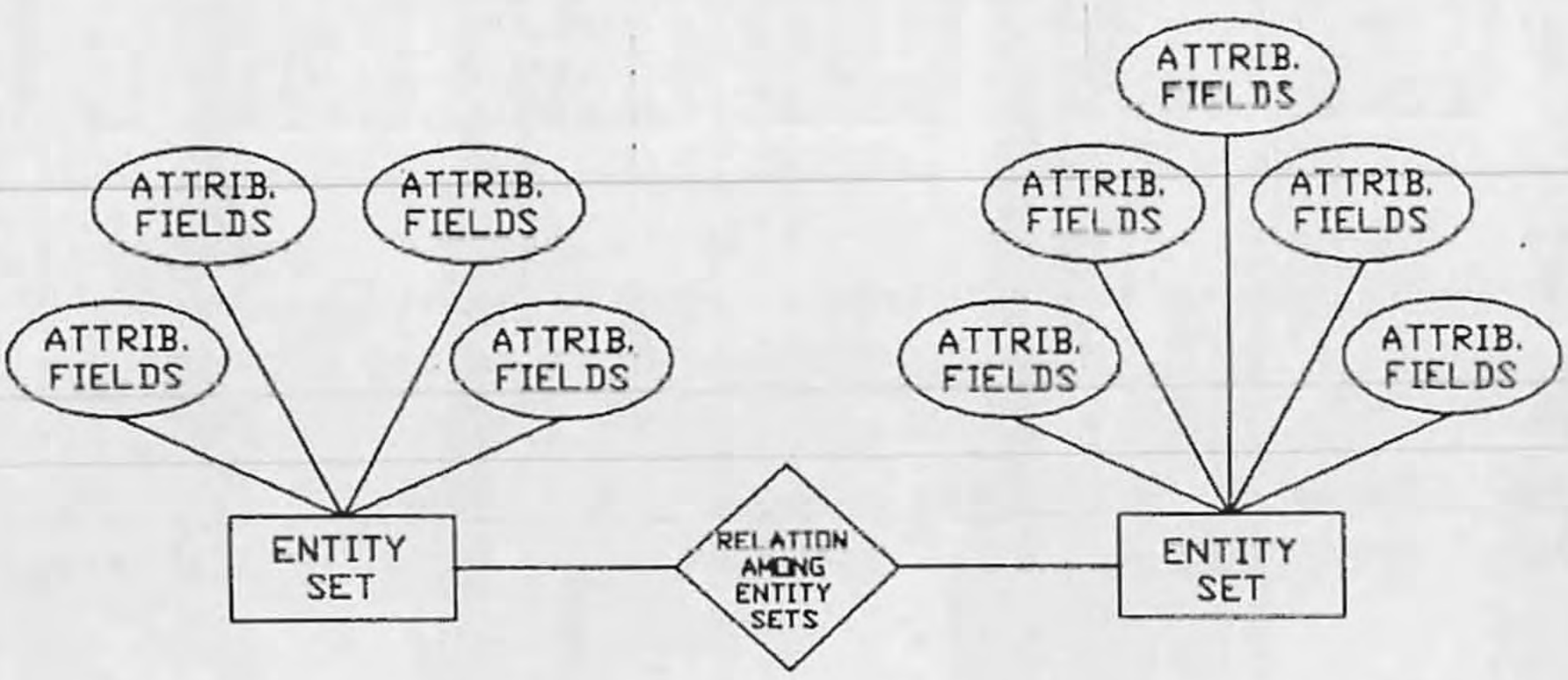


Fig. (6.7): A sample generic E-R diagram.

To illustrate this, the E-R diagram corresponding to this scheme as a part of a database consisting of rooms and the surfaces that they include is shown in figure (6.8).

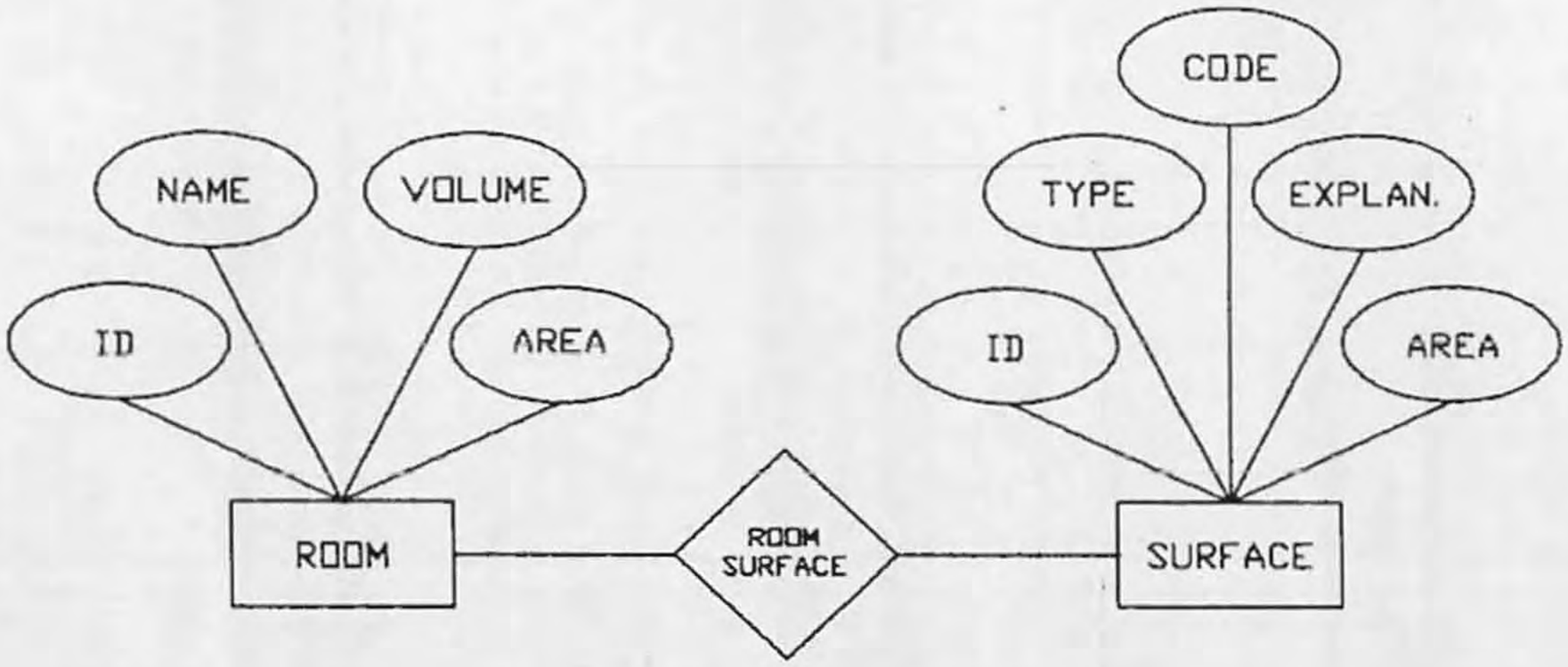


Fig. (6.8): An E-R diagram showing a room-surface scheme.

6.3. Which data model best represents the architectural design problems?

From the above mentioned data models, it is obvious that the data provided by these models is for the most part very regular, which means that data items and records have well-defined formats and simple interactions, which is not usually the case with architectural design problems.

The hierarchical approach to data organization is easy to implement. However, it is not flexible, and is unable to easily model the complexity of architectural design problems in which different hierarchies may exist, and in which more general relationships must be represented.

On the other hand, the network approach for organizing data relieves the major restrictions of the hierarchical data model. As a result, they are more likely to represent the real world than the hierarchical data model.

However, both data models, the hierarchical and the network data models need a program to read or interpret their records. Thus if the data organization is changed, the programs referencing the data may also change. In contrast, one of the major features of the relational data model is its ability to hide how data is organized (transparency of data organization). The relational data model provides an especially simple interface to interactive users whose queries cannot be anticipated.

However, according to Peterson (1987), and Schmitt (1988), none of the above mentioned data models alone, is ideal for describing architecture and the architectural design process. In addition to a rich set of data types with complex interactions, architectural applications are frequently interactive and demand very fast retrieval of small amounts of data (Peterson, 1987). The design and the artifact being designed rely greatly on "soft" information, that is information given by the designer through his own perception, experience, culture, as well as his own design scenario.

For example, an architect using the system to design a house, may wish to view the design as a floor plan, an elevation, a section, a perspective view, or even a list of components such as windows, doors or built in pieces of furniture. The architect may wish to shift from one view to another very quickly comparing certain aspects of his design, making modifications interactively and immediately reviewing the impact of these modifications from another perspective. The relations among the data items and records, as well as the objects themselves, are very complex. Even more demanding on

conventional data models, the architect may wish to develop several slightly different designs or alternatives, all evolving from a single basic starting point in the ADP, and then he may wish to merge selected variations back into the basic design. Such complexities and requirements in the architectural design process create a need for another type of data model beside what is already existing. A data model that is more flexible and less restrictive in its abstract representation of the ADP and the design artifact.

6.3.1. Object-Oriented Databases:

Object-oriented databases have emerged to support such kinds of complex applications as the architectural design (Kalay et al, 1985), (Lapre and Hudson, 1987). Developed from the concept of object oriented programming (OOP), object oriented databases among other types of databases, are the most promising to decrease the gap between the tool and the real world, this is referred to as the "semantic gap"; according to Peterson (1987), one of the measures of a data processing tool's goodness is how easily the tool can be made to model the real world. The distance between the tool and the world is frequently referred to as the "semantic gap". Tools with a smaller gap are easier to use, resulting in better processing efficiency (Peterson, 1987).

The object oriented approach has two fundamental advantages; encapsulation (modularity), and inheritance. For more details about the object oriented data base design, see Peterson (1987).

In contrast to traditional programming where the user applies operators to operands, and must assure that they are of the same type or compatible with each other, encapsulation enables the user to request an action from an object, then the object chooses the correct operator by itself. In other

words, the object oriented approach encapsulates into a single structure data and the procedures or methods which manage that data. The external interface of each object is defined by the object's methods. The exact form of each method, as well as any other data items contained in the object, are not accessible from outside the object. This object abstraction results in programs constructed of objects that interact by calling each other's methods.

The second property, inheritance, greatly improves the reusability of codes, as opposed to traditional programming where new functionalities often means extensive re-coding (Cox, 1986).

An object may inherit attributes from more than one object definition, besides it may mix, match and even substitute inherited variables and methods.

The basic idea of an object oriented database is to represent an item in the real world being modeled with a corresponding item in the database. This includes modeling the behavior of each object as well as the object's structure.

The architect's design for a high rise office building for example, would be considered as a single object in the database. In addition to modeling the building's structure, the design object might also model the building's behavior in high winds, or its heating and cooling requirements under varying conditions. The building's plan would contain many objects as well, each of which is also considered as a single object in the database. This one-to-one mapping reduces the semantic gap between the real world and the database modeling of that world.

6.3.2. Advantages of Object-Oriented Programming for the Architectural Design Process:

The object-oriented approach seems to fit well with the way architects think about their design and their artifacts. It is interesting from the architectural standpoint because of the following reasons:

6.3.2.1. Objects represent data as well as the operations or the methods needed to be performed on these data. The representation of parts of a building as objects, for example, may allow the exploration of more concepts in an early phase of design, through rotating or moving that object along one of the three axis to explore other design alternatives.

6.3.2.2. Objects represent physical objects, ideas, building functions, relations between building functions and other real world entities. The functional diagram (or the bubble diagram) of a building, for example, is very important in the preliminary design phase. This diagram can be implemented as an object by itself. Such an object has the advantage that it can be related to other objects, to improve the error prone in the process of translating the meaning of one representation (adjacency matrix) to another (floor plan). This approach has been implemented in the functional module of ARCHPLAN by Schmitt (1988).

6.3.2.3. Objects can inherit knowledge from other objects. Class inheritance allows the establishment of hierarchical and other forms of order between building elements and functional relations. In architectural design, this feature is very important because useful spatial or functional elements are

constructed and defined only once, then they are inherited completely or partially by other elements on a different level of abstraction. An example, is the removal of an entire wall, the wall is removed with all the doors and windows it includes.

For the above mentioned reasons, it is obvious that all types of data models could be useful in the architectural design process. Some of them are heavily used and some are not, but neither of them alone is sufficient to represent the complex nature of the architectural design as a whole. However, adopting an object-oriented approach for the data structure of the proposed system seems quite promising regarding architectural design information needs.

6.4. The Moving Database Concept:

It is important that the user be able to easily access, retrieve and modify the data according to his own view needs. Recognizing this, the DD/DS should be provided with **extensibility.** This means that the data can more or less change itself, to best suit the needed design phase at hand.

The extensibility capability enhances the power and the flexibility of the DD/DS as an effective tool for managing metadata. It could potentially provide a three-dimensional enhancement to the dictionary, because the user can define additional metadata entities, and store information corresponding to these entities, beside being able to use the reporting capabilities of the DD/DS to have queries answered.

In order to create a new entity, it is important to define the meta-entity, define its relationships to other meta-entities in the metadatabase, and create new attributes that can be used to describe the new entity, or use existing

attributes (if applicable) for this purpose. The insertion of additional or new data records in the database does not disturb the file organization, so that it will not be necessary to reorganize the data at frequent intervals.

This three-dimensional view could be illustrated as in fig. (6.9). Along with the different metadata that is needed to define different kinds of data like: project related, site-related and generic data base, the user may define additional metadata concerning the three modes of inquiry in the ADP, viz. the technical, the aesthetic, and the behavioral modes (refer to the ICAAD.DSS model in chapter 5 in this thesis).

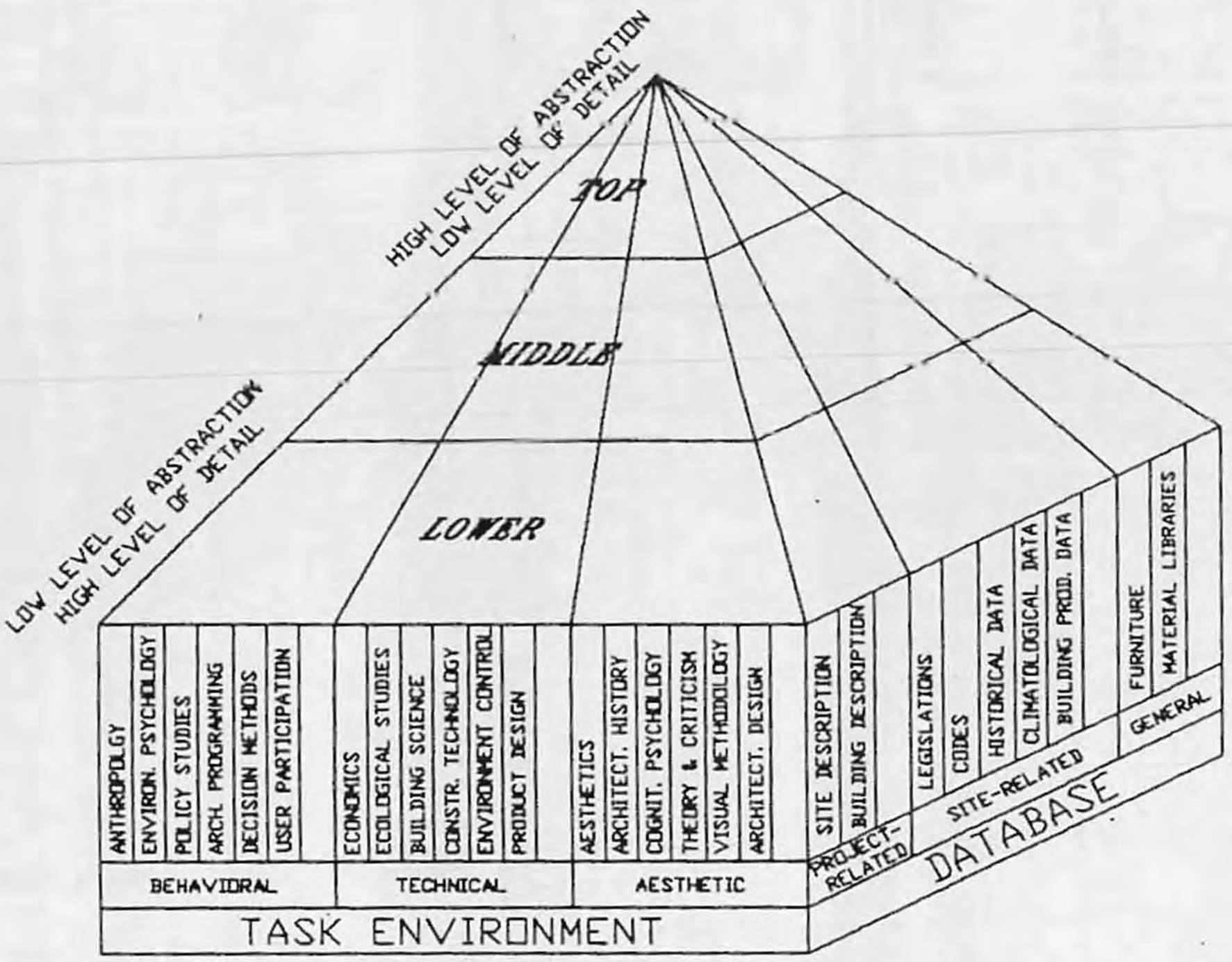


Fig. (6.9): The three dimensional view for the moving/bidirectional database.

This concept got its name "moving" because of the ability of the user, through the user interface to move, or slide along the different data series. The user is able to jump from one data level to another, he can browse through the entire database for information he needs according to the design phase in hand (Hosny et al. 1990).

By means of utilizing a specific file format, which will be able to operate with all the application programs as well as the DBMS and the DGMS softwares in the system, this file could be maintained as a master file for all operations in the system. As a result, changing the data in any of the three levels will result in an appropriate change in the database in the other two levels.

The idea is that, in order to maintain data integrity and consistency, if any user changes his database at a certain level, then appropriate changes are automatically made in the other two levels as well. This integration is achieved according to a number of procedures:

1. The user changes his data at a certain level, e.g. he is changing his own view of the database in the preliminary design phase.
2. This change in data is automatically reflected in the master database files found in the DD/DS location in the system.
3. Whenever another user tries to access a certain piece of information, the database for his own view in the design development phase for example, which is provided through the DD/DS will reflect this data change done earlier in the preliminary design phase.

This is also true for the opposite case, in the sense that, if a user in a lower level (e.g. the design development phase), had to change his data or add new constraints, this

is automatically reflected in the database accessed by another user in the upper level (e.g. the preliminary design phase), in order to achieve data consistency, and accordingly, to achieve design consistency. The effect is a bi-directional impact, that is both interactive and transparent.

6.4.1. The Definition of the Moving Database Concept:

It is the transfer of a whole body of data from one location of storage, feeding a particular design phase in the ADP, into another location of storage, feeding a different phase in the ADP in such a way that, if a certain object is created, or accessed, it will have three different types of data related to it, regarding the three phases of design in the ADP (Hosny et al. 1990).

The idea is, as data in the system is retrieved through the DD/DS for a specific design phase, only a portion of the data in the DD/DS is accessible through that design phase. That portion of the data is what will be needed for that design phase in particular. This means that the quality, or the kind or type of data accessed by the DD/DS is changed too in order to accommodate this specific design phase.

Through the suggested user interface, the user is able to choose from a menu, the specific data level that he needs for making his design decisions at a certain time, then he is opening another range, or scope of data appropriate to the specific design phase at hand, as will be discussed in the next chapter.

6.5. The Internal Representation of a Design Abstraction:

It is important to point out here that the internal representation of the data structure for a single design phase, is seen to be a composite of all three levels of data

abstraction described earlier in figure (6.1), i.e., that each single design phase has three different levels to accomplish it.

In order to define a design abstraction, some important issues should be made clear, and these are:

1. As the design proceeds from a higher level of abstraction, to another lower level of abstraction, the degree of details increases. The opposite is also true, in the sense that as the designer backtracks from a lower level of abstraction, to another higher level of abstraction, the degree of details decreases, refer to figure (6.2).

2. Another important issue, is the notion that, in order to define a certain level of abstraction, it is important to define the needed data, as well as the different methods or operations associated with this data.

In a single design phase, the process starts with a certain view of the data, and accordingly, a certain view for a part of a solution or "a priory solution" (Akin, 1986); for example, in considering an architectural floor plan, the needed information or the data will be regarding the basic entities or objects that constitute the whole plan, like walls, doors, windows, floors and ceilings. Then certain methods or operations are applied to these data to transform the solution to another solution state (another abstraction form or another view).

It is important that the operations include "mappings" from previous abstractions, i.e. that operations include ways to transform data from one abstraction to another (e.g., from a single line drawing for a floor plan to a double line drawing for that plan showing wall connections), or even to

allow previous abstractions to be maintained into the current one.

The methods or operations are needed to make changes to the initial view; for example, at this view level in the preliminary design phase, the methods will be; applying some rules in the system regarding the use of the above mentioned objects. These rules have a condition/action form. They automatically execute a certain action whenever a certain condition is met (which will be explained later). They are needed to create, and modify the plan, like creating a partition or a wall, moving that wall, rotating it, creating spaces, doors and windows, etc. The advantage of having several views is to examine as many alternatives as feasible.

Then the designer moves to the second level of data abstraction, and that is the conceptual level. At this level, he (the designer) is able to describe "WHAT" data are actually there in the database, as well as "WHAT" are the relations that exist among this data. Again, at this level, there may be more than one conceptual level for each single view, which also helps exploring more design alternatives.

3. The third important issue, is the definition of a set of relations that is always maintained in order to achieve consistency among the data sets. Examples for these are; the relations between walls and spaces, regarding that walls usually enclose spaces, as well as the relations between walls, doors and windows, regarding that doors and windows are always located within walls. A relation may take one of many forms, and these are governed by relation statements like:

* (wall --> door <-> window), this is a sequential relation which implies that the sequence for achieving a wall must precede that of achieving either the door or the window. The " > " indicates that what is at the left must precede

what is at the right. The "<->" indicates that the door or the window may be achieved in any order.

* (a_k_o), which means that what precedes it is "a kind of" what comes after it. In other words, this kind of relation connects a more specialized entity or an object with a more generalized one. An example would be, a master bedroom is a_k_o a bedroom; a bedroom is a_k_o a room; a room is a_k_o an enclosure; an enclosure is a_k_o a space, and so forth.

* (a_p_o), which means that what precedes it is "a part of" what comes after it. In other words, this kind of relation establishes hierarchies between different objects in a building. For example, a door is considered to be "a part of" a wall; and the wall is considered to be 'a part of" a room, and so on. In architectural terms, this means that whenever the wall is removed from the database, the door is removed accordingly.

* (c_t), which means that both objects engaged in this relation are connected to each other, and it is required that they maintain the same relation if one of them should change. An example would be: two walls that are perpendicular to each other. If a rotation operation was to be made to one of them, the other will automatically be rotated so as to maintain the needed relation; (perpendicularity) in this case.

4. The fourth point is the set of tests and performance conditions that can be evaluated from the data. These are important for determining how well the design alternatives meet the goals and the constraints in the program or the problem's brief. As well as to give an idea about the degree of violation regarding a certain constraint. An example would be; calculating the room areas, and the total space, and

comparing them with the program, then listing the differences, as for cost estimation purposes. Or, checking accessibility relations between inner and outer spaces, as for fire requirements.

It should be made clear that the above mentioned internal representation of an abstraction regarding data sets, operations, relations, and the tests and evaluators are always there in all different levels of abstraction; a different level of abstraction would use the same internal data representation (the same four items), but would provide different data sets, different operators, different relations, different evaluation tests and different modes of interaction.

To illustrate the distinction among the different levels of abstraction, we draw an analogy to the concept of data types in programming languages. Most high level programming languages support the notion of a record type. For example, in a pascal-like language, we may declare a record as follows:

```
type door = record
                id : string;
                number : string;
                type : string;
                width : string;
                height : string;
                material : string;
                finishing : string;
                color : string;
            end;
```

This defines a new record called "door" with eight fields. Each field has a name and a type associated with it.

At the design production phase, a door, room, or column record can be described as a block of consecutive storage locations (e.g. words or bytes). At the design development

phase, each such record is described by a type definition, as illustrated above, and the interrelation among these record types is defined. Finally, at the preliminary design phase, several views of the database are defined according to the several users using the system.

For example, a mechanical engineer who is working on the ducts plan, can only access the part of the database that has information about the ducts and the pipes. He need not access information, for example, about electrical installation, besides he only needs to access that part of the database that is relevant to the specific design phase he is dealing with. For example, if he is working in the preliminary design phase, he does not have to deal with information regarding the details for the joints, besides, he does not have to know at that point in the design process, what are the actual needed areas for the ducts.

To make this clear, we will take an architectural object like "a window" in a room, as an example, and reference the above mentioned four points to each one of the different levels of abstraction, viz. the preliminary design phase, as in figure (6.10.a), the design development phase, as in figure (6.10.b), and the detailed design or the design production phase, as in figure (6.10.c) respectively.

We'll show what kind of data is related to the window for each level of abstraction. We'll show as well what are the allowable methods or operations assigned to that window, along with the different sets of relations (geometrical and functional relations) that could be related to it, as well as the different tests and evaluators that could be applied to that window to check for its data consistency, and integrity within the design context.

The PRELIMINARY DESIGN PHASE:

DATA:

IF_ natural light is needed and/or
_ visual relations to the site are of interest and/or
_ natural ventilation is needed
THEN_ use a window

IF_ privacy is needed
THEN_use either reflective glass, or drapes, or roller blinds

IF_ sun protection is needed
THEN_use either venetian blinds, or shutters or exterior louvers

OPERATIONS (METHODS):

- Scaling : only allowed in plane of window.
- Moving : only allowed within wall.
- Rotating : only allowed within wall.

RELATIONS:

a sliding window SW1 is (a_k_o) window
window W1 is (a_k_o) window
a window is (a_p_o) wall
a louver is (c_t) window
a shade is (c_t) window

TESTS and EVALUATORS to check:

- whether window is within a wall
- that window does not exceed wall limits
- that privacy has been met
- that sun protection has been met

Fig. (6.10.a)
Defining the Preliminary Design Phase

THE DESIGN DEVELOPMENT PHASE:

DATA:

window id,
type,
shape,
materials,
number of panes,
type of glazing,
approx. dimensions, (width, height),
approx. sill's height,
type of shading device,
type of privacy assigned.

OPERATIONS:

- Scaling : only allowed in plane of window.
- Moving : only allowed within wall.
- Rotating : only allowed within wall.
- 3-D viewing : isometrics, perspectives and shading.

RELATIONS:

a sliding window SW1 is (a_k_o) window
window W1 is (a_k_o) window
a window is (a_p_o) wall
a louver is (c_t) window
a shade is (c_t) window

TESTS and EVALUATORS to:

- calculate area of window
- check performance criteria, like:
  - heat loss
  - day-light coefficient
  - view
  - appearance from outside
  - appearance from inside
- calculating cost

Fig. (6.10.b)
Defining the Design Development Phase

THE DESIGN PRODUCTION PHASE:

DATA:

window id,
type,
shape,
materials,
number of panes,
type of glazing,
U-values,
construction material,
final shape,
exact dimensions, (width, height),
exact sill's height,
different kinds of hardware,
type of shading device,
type of privacy assigned.
bills of materials

OPERATIONS:

- Scaling : only allowed in plane of window.
- Moving : only allowed within wall.
- Rotating : only allowed within wall.
- 3-D viewing : isometrics, perspectives and shading.
- hatching.
- dimensioning
- zooming for more details.

RELATIONS:

Emphasis is given to window details;
relations like (a_p_o) are often used:
a mullion is (a_p_o) a window
the glazing is (a_p_o) a window
the handle is (a_p_o) a window

TESTS and EVALUATORS to :

- calculate exact area of window
- calculate exact amounts of constructing materials
- calculate heat loss
- calculate exact cost
- check against program's initial budget
- list differences in budget and estimated cost

Fig. (6.10.c)
Defining the Design Production Phase

Fig. (6.10): Defining the different phases of design.

6.5.1. The Preliminary Design Phase:

As the program's brief indicated that natural light is required for that room, and that visual relations to the site are of interest, then the object "window" is created within a single line drawing of the room plan. The program's brief might as well indicate that, privacy is an important issue to be achieved, as well as it is important to prevent the direct sun rays from entering into the room.

Rules as well as grammars, mostly govern the design generation in this phase. For the above mentioned criteria, the rules may take the form:

```
IF_ natural light is needed
THEN_ put a window within the exterior wall
```

another rule would have a different required condition, but the action needed would be the same, like:

```
IF_ visual relations to the site are of interest
THEN_ put a window within the exterior wall
```

Rules for governing privacy issues for a window would take the form:

```
IF_ privacy is needed
THEN_ use either reflective glass or drapes or
      roller blinds
```

For sun protection, the rule may be:

```
IF_ sun protection is needed
THEN_ use either venetian blinds or shutters or
      exterior louvers
```

Another rule may be:

IF_ Louvers should be used
AND_ Facade is South oriented
THEN_ Make louvers horizontal

Or,

IF_ Louvers should be used
AND_ Facade is East or West oriented
THEN_ Make louvers vertical

Operations for creating and editing the window like scaling, moving, and rotating are allowed at that point under certain conditions, this is because the object "window" inherits properties and information from its object class, it is not allowed, for example to enlarge the window to the extent that it exceeds the wall limits that it is embedded within. Moving the object "window" for example is not allowed unless it is put within the exterior wall of the room to satisfy the "visual relation" goal with the site. Rotating the window, on the other hand, is only allowed in the plane of the exterior wall.

Sequential Relations like:

wall --> window

Implies that the sequence for achieving a wall must precede that of achieving the window. Other relations like (a_p_o) are assigned to the window at this design phase. The relation takes the form:

window is (a_p_o) wall

This relation includes that the window could only be placed within a wall, and if that wall was to be removed or deleted, then the window is removed or deleted automatically with the wall.

At that point in the design, the designer has decided to put a window within the exterior wall of the room, and he decided as well to put a horizontal louver above the window in the south exterior wall.

The tests at that point in the ADP would check for features like: the window does not exceed the wall limits, it should not be rotated around the z axis of the wall, and it should not be moved so that it is half in the wall and half outside the wall. Tests would also check that the initial goals are met, regarding the need for natural light, the interest in an external view, privacy, and sun protection. An option would be to calculate a rough cost estimate, if the kind of material was indicated at the beginning of this phase.

So far, some of the initial goals in the program are already met, and some are not. The privacy constraint has been violated.

If the designer wishes to access the next design phase, he may do so by choosing the appropriate phase from the menu in the browser's screen. But then, the system will prompt a message on the screen, informing the designer that there is a goal that has not been achieved, or a constraint that has not been met, stating what is this constraint and what is the appropriate step in the ADP to be accessed in order to resolve the conflict. Then it is up to the designer, whether to accept the system's default, and "backchange", or to access the next step, break the violation, examine new constraints, and consider resolving this violation at a later stage in the process. Refer to fig. (6.10.a).

6.5.2. The Design Development Phase:

After completing the preliminary design phase, the architect decides to access the next phase in the ADP, and

that is the design development phase. He may do that by choosing the appropriate phase from the browser's menu on the screen. As soon as the architect chooses to quit a certain phase and access another phase in the process, the testing and evaluation programs in the system automatically prompt him with suggestions to go to a previous step in the process, if there was a violation for any constraint. If not, the architect is then prompted with another scope or range of data relevant to the newly accessed design phase.

According to the moving data base concept, additional information is available at that point in the ADP that is relevant to the user's needs to more develop his design. Data regarding the shape of the window, its type, its materials, the number of panes, approximate dimensions, approximate height of the sill, and shading devices are all assigned to the object "window" at this time. Translating these to architectural decisions, the architect may decide to have a sliding window, made of aluminum and transparent glass, with four panes, approximate dimensions are 1.20 m x 2.40 m, approximate height of sill is .90 m, and finally he decides to put a horizontal reinforced concrete louver above the window in the external southern wall, in order to fulfill the goal of "sun protection".

The operations for this phase (regarding our particular example), are almost the same as in the previous phase, except for the addition of 3-dimensional visualization techniques in the form of isometrics, axonometrics, or even perspectives.

The relations in this phase of the ADP include all pre-existing relations. In addition they may include (a_k_o) type of relations. Since the architect decided that the window would be a sliding window, then the relation would take the form:

sliding window is (a_k_o) window

Having the object "sliding window" inherit the properties of its object class "window" would maintain the relation:

window is (a_p_o) wall

This means that the object "window" has to be embedded within the wall, and if any operations are performed on the wall, these will be reflected on the window.

Another kind of relation is the sequential relation regarding the achievement of the different objects. This could take the form:

wall --> window

This means that the wall must be achieved before the window; if there is no wall, then there could be no window, unless the whole wall is considered as a big window, which will then be represented by the "a kind of" relation:

wall is (a_k_o) window

Another kind of relations is the relation of the window to the horizontal louver. This could be written in the form:

louver is (c_t) window

This means that the louver "is connected" to the window, so whenever the window's location in the exterior wall is changed, the louver's location has to be changed accordingly.

A rule might guard the relation between the window and the louver, and could be written in the form:

IF_ a horizontal louver is to be used

THEN_ its location should always be exactly above the window

The tests will check for the area of the window and whether it is sufficient for the room's space. An energy simulation test would check for the performance criteria of the window. A rough cost estimate (yet more detailed than the one applied in the preliminary design phase) could be calculated according to the area of the window, the cost of the horizontal louver is also calculated, and all could be checked against the initial budget, and differences may be listed.

The tests may show that the initial budget has been exceeded, then the system will advice the designer to go back and resolve this conflict. Refer to fig. (6.10.b).

6.5.3. The Design Production Phase:

After developing the design of the object "window", the designer decides to shift to the next phase in the ADP, and that is the design production phase. As mentioned earlier, the process of accessing a new design phase is accompanied by suggestions from the evaluating programs in the system whether to go to a previous design phase if a design constraint was violated, or to proceed to another design phase, break the constraint and examine new variables, according to the designers will.

Now the designer has a new scope of information regarding the window, he may access detailed information like what are the different kinds of glass that could be used, what is the U-value for each, what kinds of materials are suitable for constructing the window, how to preserve the maximum inside heat, and what kind of hardware could be used. In this design phase, the designer makes decisions as to what material is he

going to use, what kind of glass will be suitable, which panes will be movable and which will be fixed, what kind of hardware is he going to use, what exactly is the final shape, what are the dimensions of the window jambs and mullions, what is the thickness of the glass, and accordingly, what is the actual area of the window. Deciding upon these issues, the designer may pick a double glazed window, with 4 mm transparent glass, with four panes, the outer two are fixed, and the inner two are sliding. In addition, he may decide not to have the horizontal louver, but to use venetian blinds instead, in order to fulfill both, the privacy goal as well as the sun protection goal, and not to have the cost exceed the initial budget.

The operations regarding this phase of design, are concentrated on applying more details for the window. These may be using hatching patterns, using zooming techniques, and using dimensions. The designer may wish to visualize his artifact in a 3-dimensional view from different points; from the outside of the room, as well as from the inside of the room, and he may wish to see how will the proportions of the mullions and the jambs will look like, he may wish to examine the relation between the void and the solid in the window as a whole.

The relations in this phase of design still include all existing relations from previous design phases. They may include relations like:

```
        a mullion is (a_p_o) window
or      glazing is (a_p_o) window
or      handle is (a_p_o) window
```

These relations mean that if ever the window's location is changed or the window is even completely removed or discarded, then all these objects being related to the window, are removed accordingly.

Different kinds of tests are applied in this phase of design. An exact estimation of the area, as well as the lengths of the different mullions could be calculated to determine the needed amounts of construction materials. An exact calculation of the heat loss could be calculated. An accurate cost estimate could be calculated at that point regarding the different materials used, as well as their quantities, and their costs, which could be checked against the program, and differences could be listed, if any. Finally, a calculation of bills of materials could be supported in this phase of design. Refer to figure (6.10.c).

Fig. (6.11) shows a graphical analysis as to what is happening to the data during the different design phases, regarding the data, operations, relations , and the tests needed to define each level of abstraction. An empirical value is assigned to both; the abstraction, as well as the detailed levels, which ranges from 0 to 10.

In this chapter, different information levels for the ADP have been discussed. Important questions regarding the data structure of the ICAAD.DSS model have been answered in detail to allow computer implementation; questions like: why do we need a DD/DS?, and which data model best represents the architectural design problems? have been answered as well. The moving/bidirectional database as the central organizing concept for the integration of computers in architecture has been defined, and knowledge about the internal representation of architectural design abstraction has been discussed and defined as well.

Now it is important to identify the user interface, which is the main theme of the next chapter in this study.

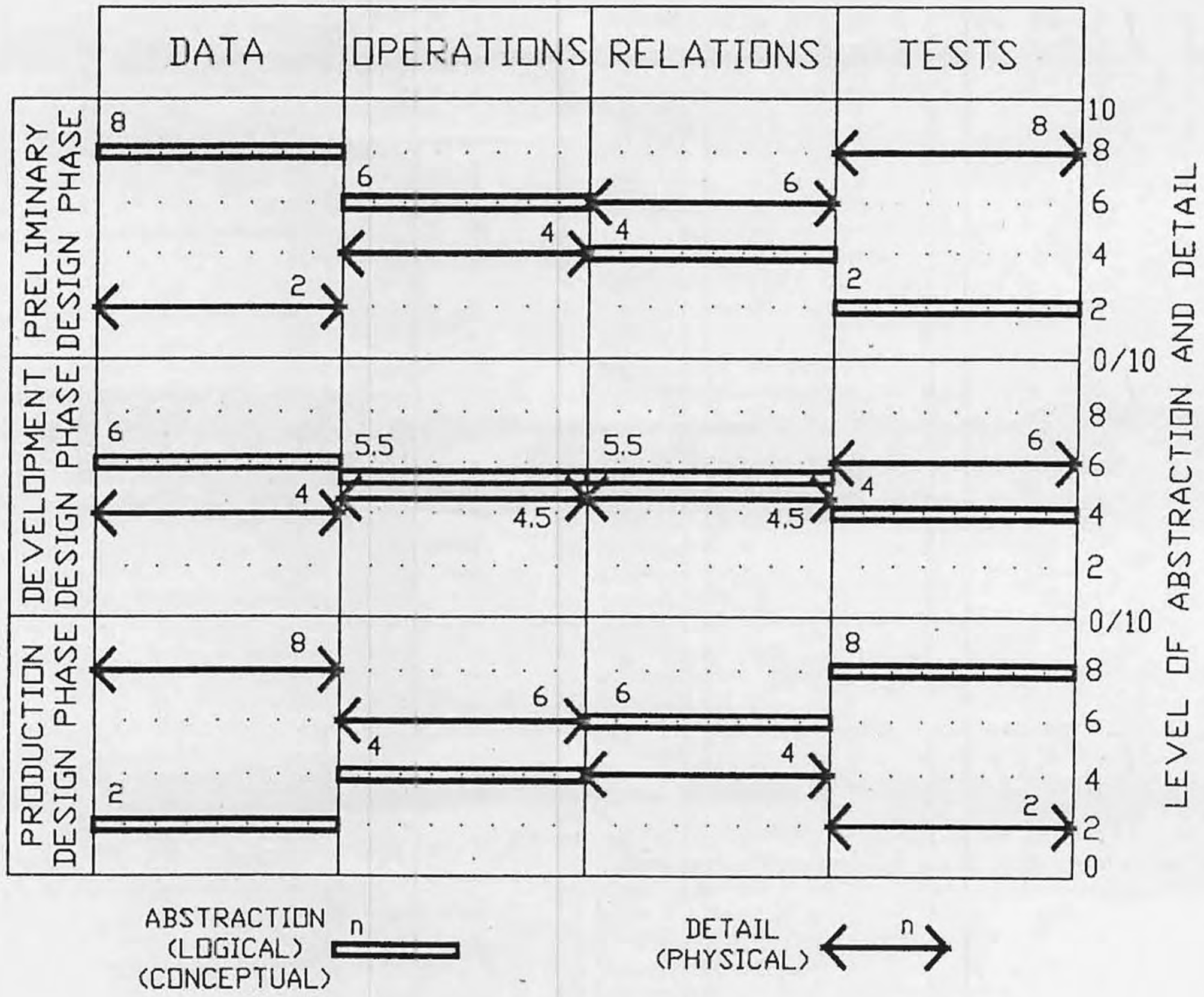


Fig. (6.11): Different levels of abstraction and detail in the architectural design process.

REFERENCES FOR CHAPTER 6:

Adeli, H.:
Expert Systems in Construction and Structural Engineering. Chapman and Hall, New York, 1988.

Akin, O.:
Psychology of Architectural Design. Published by Pion Limited, London, Great Britain, 1986.

Bamford, C., and Curran, P.:
Data Structures, Files and Databases. Macmillan Education Ltd., London, 1987.

Bjork, B., and Penttila, H.:
A Scenario for the Development and Implementation of a Building Product Model Standard. Technical Research Center of Finland, March, 1989.

Brooner, E. G.:
Microcomputer Data-Base Management. Blacksburg, continuing education series, Howard W. Sams & Co., Inc., Indianapolis, 1982.

Brown, T. B.:
The Development of an Architectural Information Classification System and Computerized Data Storage and Retrieval System. An M.Sc. Thesis, The Pennsylvania State University, 1968.

Chandrasekaran, B.:
"A Framework for Design Problem-Solving." In Research and Engineering Design, Vol. 1, no. 2, 1989. Springer International, New York.

Conklin, J.:
"Hypertext: An Introduction and Survey". In Computers Vol. 20 No. 9, 1987, pp. 17-41.

Cox, B. J.:
Object-Oriented Programming. Addison Wesley Publishing Company, Reading Massachusetts, 1986.

Eastman, C. M.:
"Abstractions: A Conceptual Approach for Structuring Interaction with Integrated CAD Systems." In Computers and Graphics. Vol. 9, no. 2, 1985.

Hosny, S.; Kalisperis, L.; and Sanvido, V.:
The Data Structure for the ICAAD.DSS Model. Unpublished Paper. The Pennsylvania State University. April, 1990.

Hutt, A.T.F.:
A Relational Data Base Management System. A Wiley Interscience Publication, John Wiley & Sons, New York, 1979.

Jones, J. A.:
Databases in Theory and Practice. TAB Professional and Reference Books, a division of TAB Books Inc. PA. 1987.

Kalay, Y., Swerdloff, L., and Harfmann, A.:
"ALEX: A Knowledge-Based Architectural Design System." In Proceedings of ACADIA Workshop '85, edited by P. McIntosh, 1985.

Kalisperis, L. N.:

A Conceptual Framework for Computing In Architectural Design. Ph.D. Thesis, The pennsylvania State University, 1988.

Kruglinski, D.:

Data Base Management Systems, A guide to Microcomputer Software. McGraw-Hill, Berkley, California, 1983.

Lapre, L., and Hudson, P.:

"Talking About Design: Supporting the Design Process With Different Goals." In CAAD Futures, 1987. Proceedings of the 2nd International Conference on CAAD Futures, Eindhoven, The Netherlands, 20-22 May, 1987, pp.127-136.

Leong-Hong, B. W., and Plagman, B. K.:

Data Dictionary/Directory Systems, Administration, Implementation and Usage. A Wiley-Interscience Publication, John Wiley & Sons, New York, 1982.

Medland, A. J.:

The Computer Based Design Process. Springer-Verlag, New York, 1986.

Peterson, R. W.:

"Object-Oriented Database Design." In AI Expert, March, 1987. PP. 27-31.

Rasdorf, W. P., and Salley, G. C.:

"Generative Engineering Databases - Toward Expert Systems." In Computers and Structures, Pergamon Press, Vol. 20, Numbers 1-3, pp 11-15, 1985.

Rasdorf, W. J., and Watson, B. R.:
"A knowledge-Based Approach to Engineering Information retrieval and Management." In Expert Systems in Construction and Structural Engineering, by H. Adeli, editor, Chapman and Hall Ltd. London, U.K., pp. 267-295, 1987.

Schmitt, G.:
"Archplan: An Architectural Planning Front End to Engineering Design Expert Systems." In Expert Systems for Engineering Design, edited by Michael D. Rychener. Academic Press Inc. CA, USA, 1988.

Silberschatz, A.:
Database System Concepts. New York: McGraw-Hill, 1986.

Stonebreaker, M.:
"A Functional View of Data Independence." In Proceedings of SIGMOD Conf., 1974, ACM, New York, 1974, pp. 63-81.

Teory, T. J., Yang, D., and Fry, J. P.:
"A Logical design Methodology for Relational Databases Using The Extended Entity-Relationship Model." In Computing Surveys, Vol. 18, No. 2, June, 1986.

Thierauf, R.J.:
User Oriented Decision Support Systems, Accent on Problem Finding. Prentice Hall, Englewood Cliffs, New Jersey, 1988.

Van Duyn, J.:
Developing a Data Dictionary System. Prentice Hall, Inc., Englewood Cliffs, NJ, 1982.

Wertz, C. J.:
The Data Dictionary: Concepts and Uses. North Holland, a division of Elsevier Science Publishers. The Netherlands. 1986.

CHAPTER 7

THE VISUAL REPRESENTATION OF THE DESIGN STRUCTURE (THE BROWSER), AND THE USER INTERFACE

CHAPTER 7

7. The Visual Representation of the Design Structure (the Browser), and the User Interface.

7.1. The Browser.

7.2. The Suggested User Interface.

7.2.1. Hypertext.

7.2.2. Hypermedia.

7.2.2.1. The Conceptual Window.

7.2.2.2. The Graphical Window.

7.2.2.3. The Textual Window.

CHAPTER 7

THE VISUAL REPRESENTATION OF THE DESIGN STRUCTURE (THE BROWSER), AND THE USER INTERFACE

7.1. The Browser:

The visual representation of the design structure will be achieved by using a browser. The nodes and their interconnecting links of the model will be displayed on a canvas of virtually unlimited size. The browser's screen is seen to be divided into several windows (fig. 7.1). Most of it will be a view of that particular point of interest in the design process the architect is dealing with; the "current window". This current window should show full details of the nodes and their links. Another part of the browser's screen is dedicated to showing the whole model of the design process; the "model's map" or the BASED model, including the previous node which the architect has already dealt with, as a point in a highlighted series of nodes and links. Any changes made in the "current window" are automatically updated and displayed in the "model's map" (Hosny et al. 1990b).

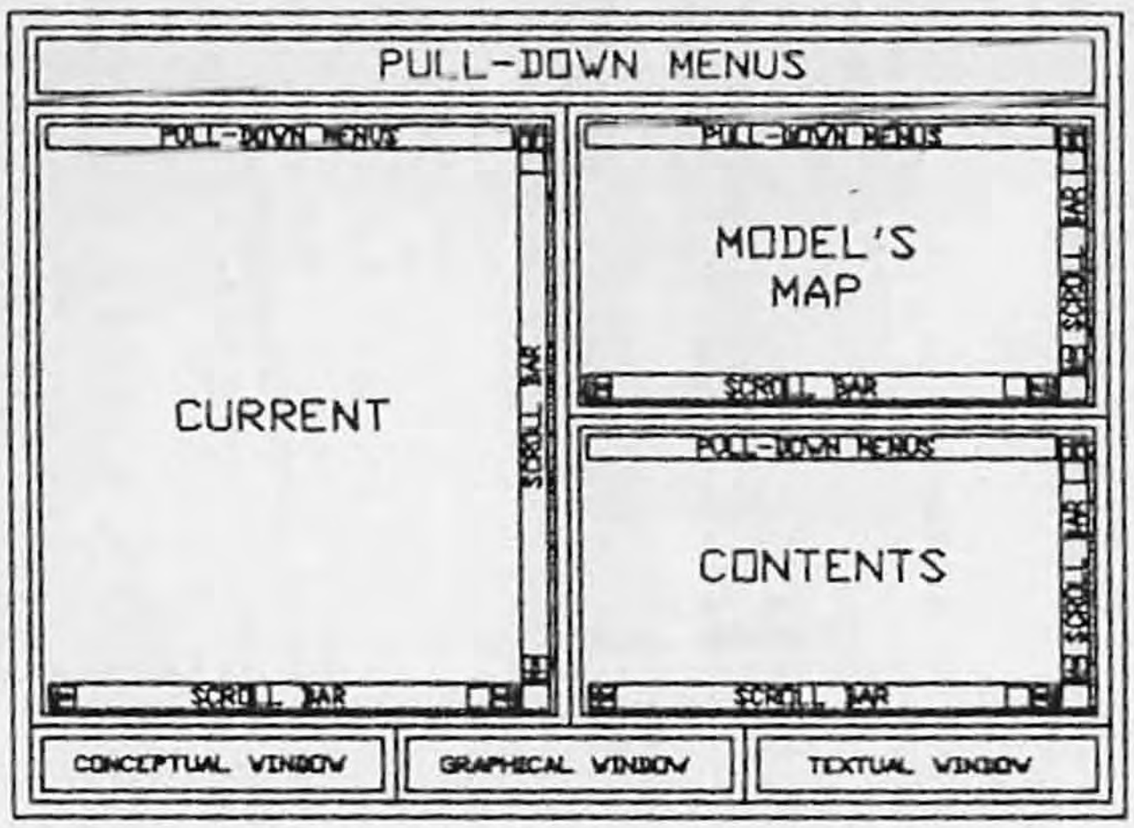


Fig. (7.1): The browser's screen, showing the division into several windows.

In addition, the system keeps a record of all changes in decisions and their sequences for the user's review in respect to the specific nodes or links that caused them. These nodes or links could be accessed immediately through the browser, by simply picking the required node or link by using a pointing device, such as a mouse, to display the contents of that node or link in another window; which is called the "contents" window. The contents of the nodes or links are displayed in that particular form in which they were used, that form which caused the decision to be made, they can be textual in the form of notes or comments, they can be graphical in the form of plan or elevation drawings, or they can be pictorial in the form of site topography and site-neighbor relations. In addition, if the designer wants to access a previous node that is not seen in the "current window", he can simply go to the "menu" window at the top of the screen, and select a "zoom" command to include that particular node and its neighboring nodes and links in a window from the "model's map", this will directly prompt the designer with that node and its surroundings in a suitable manageable scale in the "current window", or still, he can use the scroll bars along the sides of the "current window" until he gets the desired node or link. After examining that node, he can select a "go to" command from the "menu" window to access the previous node (which will be highlighted), or any other node he desires, by simply selecting it by the mouse from the "model's map" (Hosny et al. 1990).

7.2. The Suggested User Interface (hypermedia):

During the last three years (as of this writing in 1990), there has been an increased interest in the use of hypertext and hypermedia techniques in database systems. These techniques seem to be quite interesting tools for presenting design descriptions, especially in architectural design, where

the process of information retrieval relies extensively on the user's perception.

In order to define hypermedia, it is useful to define hypertext first.

7.2.1. Hypertext:

Hypertext is defined as non-linear text, i.e. writing and reading in a "more human way", which means jumping from one thought to another, connecting objects with appropriate links (Conklin, 1987). The main idea is to allow the user to easily, quickly and associatively build these connections.

7.2.2. Hypermedia:

Hypermedia combines different information storage media, not only textual information, but also graphical, and pictorial, as well as animated videos and digitized speech. Links and connections between different data items, whether text, graphics pictures or anything else, can be built just as with hypertext within an integrated computer-controlled environment.

A basic concept in hypermedia's user interface is a document window which shows a logical collection of information, such as a text document or a graphical file. Links, relations or "jumpers" from one window to another allow flexible and free combination of any data items.

Exploring the information in the data structure may take one of the following ways:

* following the links between the data windows one at a time, or

* browsing through the data structure, and choosing among the different data nodes in the data structure's overall map.

The information the user gets when browsing through the hypermedia depends solely on the user's actions or paths at the moment of browsing. The hypermedia's data structure requires the user to participate actively in the processes of gathering, creating, browsing through the data structure, as well as analyzing the information.

An example of data browsing could be a library of references in textual form as well as image libraries in graphical or pictorial forms, to which the user can freely refer to at any needed point in the ADP. An example of a hypermedia's data structure is shown in figure (7.2), after (Conklin, 1987).

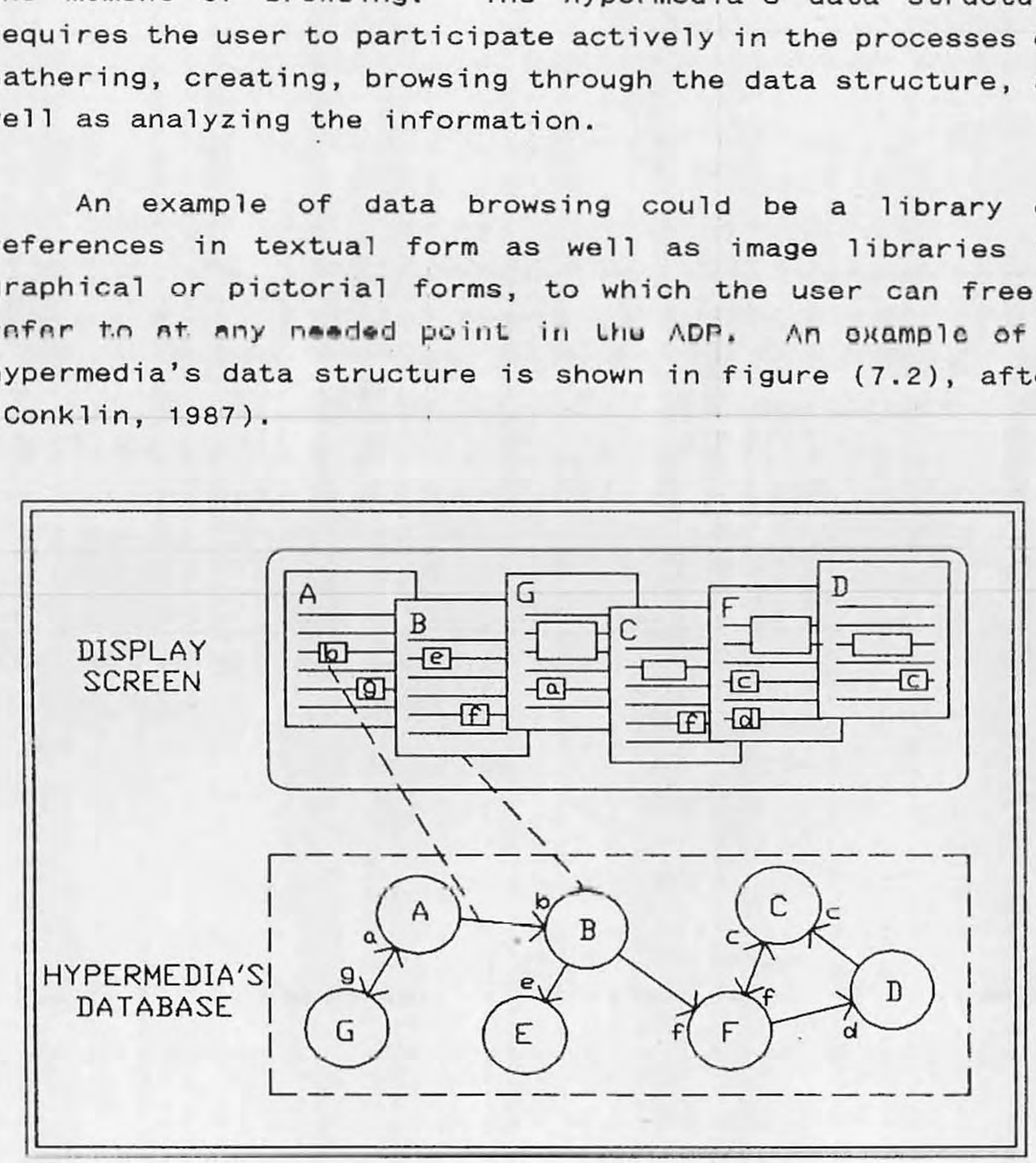


Fig. (7.2): An example of Hypermedia's data structure.

This kind of interface seems most suitable for architects who prefer dealing with drawings or graphical material rather than textual material, and clicking the buttons of a mouse rather than typing commands. The tools for browsing through the design and data structure are the "mouse", and the browser's module as suggested in the ICAAD.DSS paper (Hosny et al. 1990).

The tool for describing building objects such as rooms, walls, windows, and doors through the hypermedia's user interface, is "on screen windows", with conceptual, graphical and textual context (Conklin, 1987). Using several different windows simultaneously gives a good overall view of the whole structure of the design model. A description of each one of the above mentioned windows follows.

It is important to point out here that the three windows are accessible through any of the three levels of abstraction in the ADP regarding the preliminary, the design development and the design production phases. But the context of each single window varies according to the level of abstraction or to the required design phase. In each level of abstraction, some design variables will be ignored by not being represented in the hypermedia's windows. These variables will be represented in other levels of abstraction according to the user's progress in the ADP. Each abstraction level provides a means to investigate just a portion of the design, as well as to make decisions about it.

7.2.2.1. The Conceptual Window:

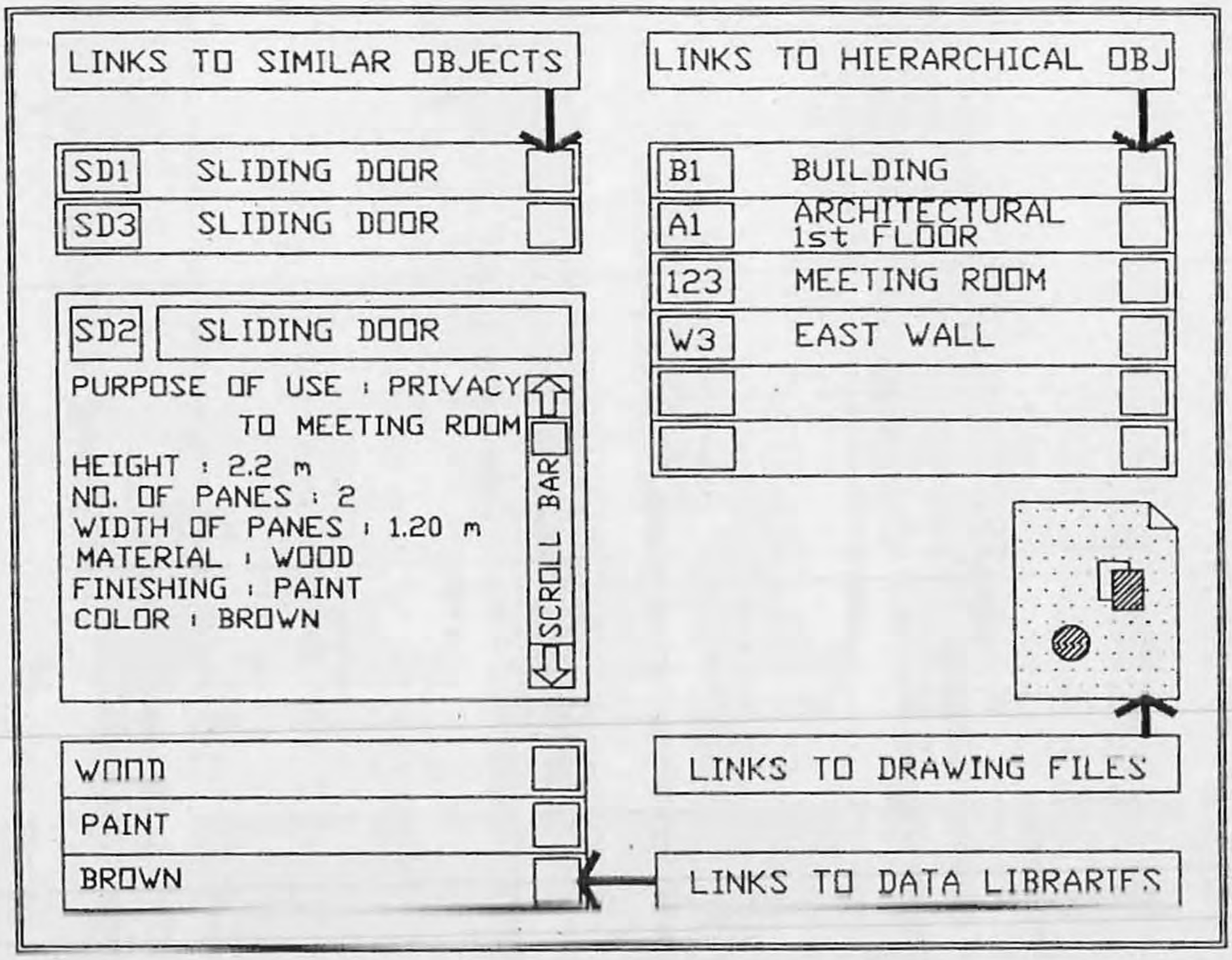
A conceptual window will show all the data describing one object, as well as the relations that this object has to other objects. The user may jump from an object to all the related objects, in other words, he may "move" within the network of building objects. This is achieved by accessing

certain links in the "conceptual window" which will give access to other objects related to the one currently in use.

As an example, if the architect is in the design production phase, dealing with an object that is a "sliding door"; he will find the attribute data related to the sliding door; namely its id. or its number, the height, the breadth, the material of the door, the number of panes, the method of opening, finishings and colors, etc. He will find as well, some links that will relate the sliding door to other objects through an (a_kind_of) relation in the form [sliding door is a_k_o door], this will have the sliding door inherit properties from the class "door", these properties include data about that specific object class, allowable operations to that class, set of relations to other objects or classes, as well as a set of tests and evaluators to check its data integrity and consistency.

The user will also find other links like (a_part_of) relations that will connect the sliding door to other objects like walls in the form [sliding door is a_p_o wall], which will allow certain methods to be operated on the door, like no rotation is allowed along the vertical axis of the door, and other operations like, if the wall is removed; then the door is automatically removed accordingly.

Other links to the other two kinds of windows also exist in the conceptual window, like links to a textual, or a graphical window, so that the user may choose the most suitable form of data that best fulfills his needs. Refer to figure (7.3).



The "conceptual window" shows the different attributes for the mentioned object. It shows as well, different links to other data sets in the data structure. There are links to other similar objects, such as other sliding doors in the project, and these are governed by (same_as) type of relations. Other links to hierarchical objects in the "sliding door"'s hierarchical tree also exist, such objects are the wall in which the door is located, and the room in which this wall is located, and these are governed by (a_part_of) type of relations. In addition, other links to data libraries and drawing files also exist.

Fig. (7.3): A conceptual window of a "sliding door" object.

7.2.2.2. The Graphical Window:

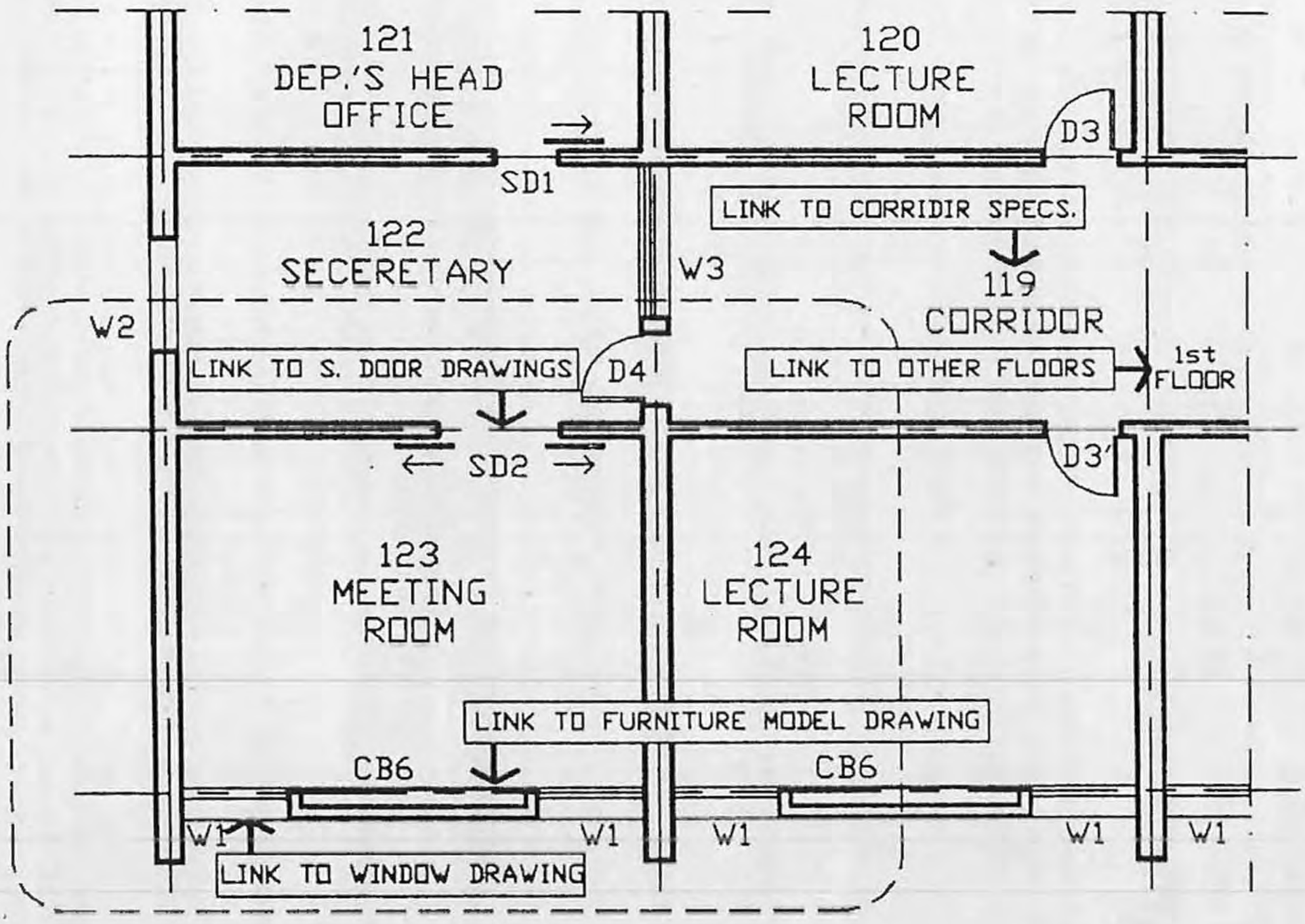
A graphical window shows drawings or graphical data related to the object. Several drawings can be linked to one object. For example the plan for the previous sliding door may have links to other drawings connected to it like elevations, sections, detailed or even perspective drawings. Refer to figure (7.4). Included in this window, are links to the other two hypermedia's windows; the graphical and the textual windows.

7.2.2.3. The Textual Window:

The textual window shows textual documents regarding the selected object. This window allows access to all written specifications. Refer to figure (7.5). Links to the other two hypermedia's windows are also included in this window.

The question as of how and when is it possible to change or shift from one level of abstraction of data (a particular design phase) to another could now be answered. In order to satisfy the requirements of an abstraction level to be able to shift to another level, it is important to identify several issues, like:

* The selection of a data structure that supports the representation requirements.
* The definition of a set of operations or methods that will help generate all possible design actions.
* The application of some relations into the data structure and the operations so that they are executed or maintained.
* The application of some tests and evaluators to determine how well design solutions meet the goals, constraints, and performance requirements suggested by the program, before allowing the shift from one phase to another.



The graphical window shows different links to: detailed drawings for objects, such as the needed sliding door (SD2), or other doors that appear in the plan, or even other sliding doors in the same project but do not appear in the plan (could be accessed by clicking the mouse twice on the needed object's category). In addition, there are links to objects like movable furniture or fixed furniture, as well as links to other floor plans, sections and elevations. The window shows also, links to the other two hypermedia's windows, viz. the conceptual, and the textual windows.

Fig. (7.4): A graphical window of a "sliding door" object.

Doors : D
Page 1 of n
Sliding Door : SD
ID : SD1

This Sliding Door is a kind of Doors that slide along the sides of the Wall. Its ID number is SD1, its Height is 2.20 m (masonry) and 2.18 m (carpentry). Its Width is 2.40 m (masonry) and 2.38 (carpentry). The total Area is 5.19 m^2. The Material used for constructing the door is Wood. The Finishing surface is Paint. The Color is Brown. The Number of Panes is 2, the Way of Opening is that both of them slide along hanging bars at the top of the door opening. The Hardware used for the handles is made of brass. There are no Vision Panels in this door.

The Floor Number where this door is located is the first floor in the architectural floor plans (A1). It is located within the Wall Number (W3). This wall is part of two Rooms, their Numbers are 123 (meeting room) and 122 (secretary).

SD1 : is a wooden sliding door, painted brown, separating the meeting room and the secretary.

Next page <N>;
Return to Conceptual Window <C>;
Return to Graphical Window <G>.

The textual window enables the user to browse through the database of articles, drawings and pictures by selecting from the highlighted items in the text of the articles, or accessing a graphical window, or a conceptual window. The textual window shows textual information with link terms highlighted. The selection is done by using a pointing device (a mouse or a light pen).

Fig. (7.5): A textual window of a "sliding door" object.

After identifying these issues, the shift from a design phase to another will proceed according to the five steps of the BASED process discussed in chapter 5.

1. Define the design goals, design constraints, and the performance requirements (Briefing),
2. Analyze the goals and constraints as to which is to be achieved first, and give weights and priorities to activities (Analysis),
3. Generate design concepts (Synthesis),
4. Evaluate design concepts (Evaluation), and then
5. Decide whether to shift to another level or not (Decision).

As discussed earlier, only a portion of the database that is appropriate for a specific design phase could be accessed at a certain time. If the user attempted to access another level of data abstraction in another design phase at a certain time, then the system will activate the evaluation programs to check whether the goals are achieved and certain constraints are met. If these are not met, then the system will advice the user to resolve this conflict, which in itself could be done manually, or by accepting the defaults of the system. Resolving the conflict could be done on the spot, at the time of its detection, or later on in the ADP.

According to Eastman (1985), the more preferable is to break the constraint derived from an earlier abstraction, make the desired changes, and "backchange" the earlier representation, which may backchange in a more previous abstraction, possibly back to the initial one. The less preferred means of iteration is to make the revisions directly to the previous abstraction, then update that change into the current one.

An example could be drawn to illustrate this through the design of a window. The designer has to specify at the beginning that one of his goals e.g. is utilizing natural light. At the preliminary design phase, the designer is using heuristic programs that are found in the system, these apply certain rules in the form of IF_THEN statement. For example a rule may be expressed generically as follows:

```
IF_ goal is achieved
THEN_ accept shift to another level
```

In contrast,

```
IF_ goal is not achieved
THEN_ prompt the user with the message:
      "Goal is not achieved.
      Want to reconsider? [Y or N]"
```

For our particular example, the rule may be expressed as follows:

```
IF_ natural light is important
THEN_ put a window
```

At the design development phase, the designer accesses the hypermedia's menu and retrieves certain data about that window and its created environment. At this level, the designer is able to assign certain attributes for the window, regarding its location, its size, shape, kind of glass to be used, U-values and dimensions etc. His tools for doing this are the different application programs in the program bank, as well as the different information provided through the databank in the system.

As he settles on these items, the designer may want to access the third phase of design and that's the design production phase. The designer may do so by just picking that phase from the menu. If for example he attempted to produce a report on that window from hypermedia's textual window, the

system will activate the evaluation programs to check for the goals' achievement and the integrity of the constraints. If any of these were violated, the system will keep a note of that until a certain point in the process when the designer wishes to move to another design abstraction, then the system will prompt the user with a message advising him to go to a previous step in the process to resolve the conflict which will be identified by the system. Then the user may take the system's default regarding the needed adjustments, or he may go on his own and resolve the conflict according to his own perception. His way for doing this, as was discussed earlier, is either to ignore the conflict derived from the early abstraction temporarily and go on with the desired modifications, then backchanges the earlier representation. Or he may revise directly the previous abstraction, then update the change into the current one.

In the case that all goals and constraints were met, the system will accept the designer's attempt to access the design production phase and automatically will search the database of the window, and will generate the needed report.

The strategy for the automatic feedback mechanism, and the redesign process by the system depends on a special search algorithm named (HDF) Heuristic-Depth-First (Lee and Kwon, 1989), which has been mentioned earlier in chapter 5 in this thesis.

In this chapter, the Browser and the Hypermedia's user interface have been presented. It is believed that the suggested user interface will provide flexibility as well as freedom of choice, because of the different alternatives available for the retrieval and the representation of data presented in the three different windows of the hypermedia.

Moreover, this user interface seems most suitable for architects who prefer dealing with drawings, graphics or even text, through clicking the buttons of a mouse, or using a light pen, rather than issuing commands by typing them using the keyboard.

REFERENCES FOR CHAPTER 7:

Conklin, J.:
"Hypertext: An Introduction and Survey." In Computers. Vol. 20, no. 9, 1987, pp. 17-41.

Eastman, C. M.:
"Abstractions: A Conceptual Approach for Structuring Interaction with Integrated CAD Systems." In Computers and Graphics. Vol. 9, no. 2, 1985.

Hosny, S.; Sanvido, V.; and Kalisperis, L.:
A Framework for an Integrated Computer-Aided Architectural Design Decision Support System. Unpublished Paper. The Pennsylvania State University, 1990.

Hosny, S.; Kalisperis, L; and Sanvido, V.:
The Data Structure for the ICAAD.DSS Model. Unpublished Paper. The Pennsylvania State University, 1990b.

Lee, H.; and Kwon, T.:
"Heuristic Redesign and Rule Formulation for Complex Engineering Designs." In Preprints of the NSF Engineering Design Research Conference. University of Massachusetts, Amherst, June 11-14, 1989.

CHAPTER 8

CONCLUSIONS, SUMMARY AND RECOMMENDATIONS

CHAPTER 8

Conclusions, Summary and Recommendations

Conclusions.

Summary.

Recommendations.

CHAPTER 8

CONCLUSIONS

It has been argued that the different application programs have not been integrated as a whole into the basic architectural design process. Recent packages have presented some compatible programs, but they are narrow in scope, they address only a limited number of tasks within small projects. The true integration of computers into the architectural design process is not progressing well, and the architect is not yet using the full capabilities of the computer as a true design tool. As a result, a unified approach towards computing in architecture based on a holistic view of the architectural design process is imperative.

Although some integrated design systems are available in the market, yet, they could not deal with the architectural design problem taken as a whole. However, the principal contrast between what is suggested in this study and these other approaches lies not only in the ICAAD.DSS's knowledge based components vs. other systems' algorithmic process components, but also in the integration issues of different building systems of design, construction, and management processes, regarding the different users and disciplines involved.

The study has introduced a conceptual model for the full integration of the CAAD process. Different characteristics, as well as the different building blocks of the system has been defined and presented. The ICAAD.DSS model presented in the study will attempt to integrate the whole CAAD process through a single definition point, and that is the Data Dictionary/Directory System, through using the concept of a moving/bidirectional database to support multidisciplinary users working with different phases, abstractions and task environments in the architectural design process.

The data dictionary/directory system along with the moving/bidirectional database concepts working together through a blackboard architecture are seen to be the focal point for increasing data capabilities, as well as achieving a fully integrated CAAD process.

The study has argued as well that design decisions and assumptions occur all along the entire architectural design process, but they are not documented, thus, careful deliberation and much of the domain learning that went into resolving key design issues is wasted. The information processing model, or the BASED model presented in the study will guide in determining the data flow during the different processes, as well as exploring and capturing the entire design process.

The documentation and the capturing of the design process featured in the BASED model are seen to be very important in the externalization of the different thought processes, in addition, they are useful through the use of the expert system component in learning and educational purposes.

The study has also pointed out the complexity of the architectural design problems regarding the various amounts and kinds of data needed to accomplish a certain goal. Such complexities and requirements create a need for another type of data model that is more flexible and less restrictive in its abstract representation of the design artifact.

Object-oriented databases are the most promising to support such kinds of complex applications as the architectural design. They decrease the semantic gap between the tool and the real world.

The study has presented as well, the Browser and a conceptual design for a special user friendly interface which offers a variety of data forms to suit different users' styles. In addition, it has pointed out the importance of hypermedia as a design tool. Hypermedia is a database method

that provides a novel way of directly and easily accessing data. This method is quite different from the traditional use of queries. At the same time, it is a representation scheme, a kind of a semantic network that mixes informal material (in graphical, textual or any other form) with more formal and mechanized operations and processes. In addition, it is a tool that features link icons which can be arbitrarily embedded within the content material by the user.

When designing a user interface for a computer-aided architectural design system, it is important to consider dealing with graphical windows with optional access to other kinds of textual information through the use of pointing devices, rather than dealing with textual data through typing commands on a keyboard. Hypermedia is seen to be one of the most suitable database methods that provides such features, in addition, it suits the type of thinking practiced during the architectural design process. It eases the restrictions on the designer, it does not force a strict decision about whether any given idea is either within the flow of a design thought or concept or outside of it.

SUMMARY:

The integration of computers into architecture is seen to be of crucial importance. Chapter one of this thesis presents the problem; the lack of computer integration in the architectural design process prevents the architect from using the full capabilities of computers in his design process. The goal of the study is developing a framework for an integrated computer-aided architectural design process. This chapter introduces the methodology for achieving the goal of the study.

As an approach towards introducing computers in the architectural design process, a study of the different design methods for architecture was presented in chapter two of this thesis. As a result, the second generation design method's approach was selected as the basis for the "ICAAD.DSS" model presented in chapter 5 of this thesis.

Based on the analysis of the different architectural design methods, the third chapter proposed problem finding and problem solving as a model of architectural design, introducing the different problem solving and search techniques used in the architectural design process.

The fourth chapter dealt with an analysis of the uses of computing in architecture until the present decade, including the integration of computers into architecture, based on the first and second generation architectural design methods.

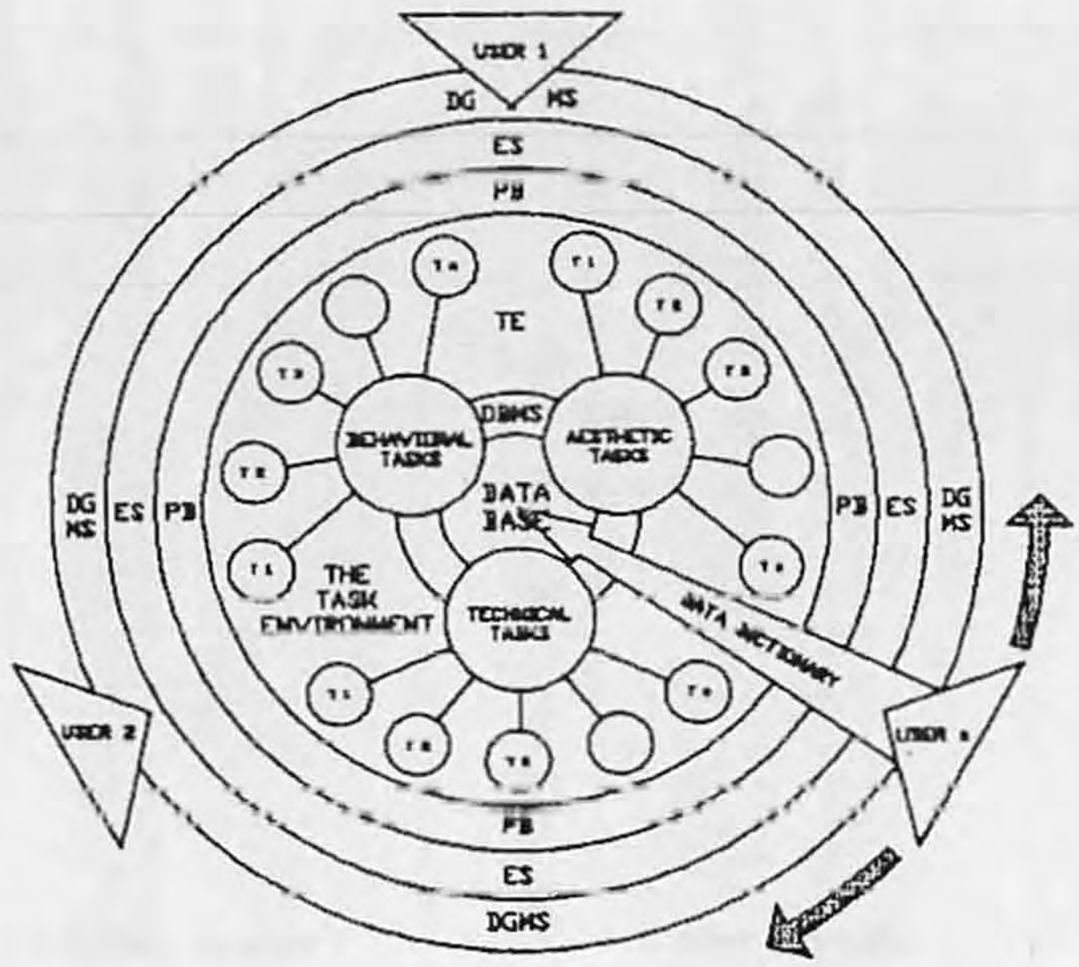
Based on the acceptance of problem solving as a model of architectural design, and the importance of integrating computers into the architectural design process, chapter five introduced a conceptual framework for an integrated architectural design computing environment. This framework is based on a unified approach to computing in architecture which embraces the scientific, the artistic, and the behavioral modes of architectural design problem solving.

The "ICAAD.DSS" model presented in chapter 5, provides such a framework. This framework shifts the focus from product to process, and views the design problem as a goal-oriented, problem-solving activity. Based on the principles of the second generation methods, the framework allows multi-disciplinary users in a design team to identify strategies and methodologies in the quest for design solutions.

The proposed framework allows for an intuitive dialog between the architect and the computer. It is believed that this framework will expand the role of computers from their already established ability in representing and evaluating design solutions, to the ability of assisting in the creative and judgmental functions of the architectural design process as well.

The framework embraces three major components; the users, the task environment, and the decision support system. The different building stones for the framework are addressed; viz. the communicator or the dialog generation and management system, the databank, the database management system, the program bank and the expert system.

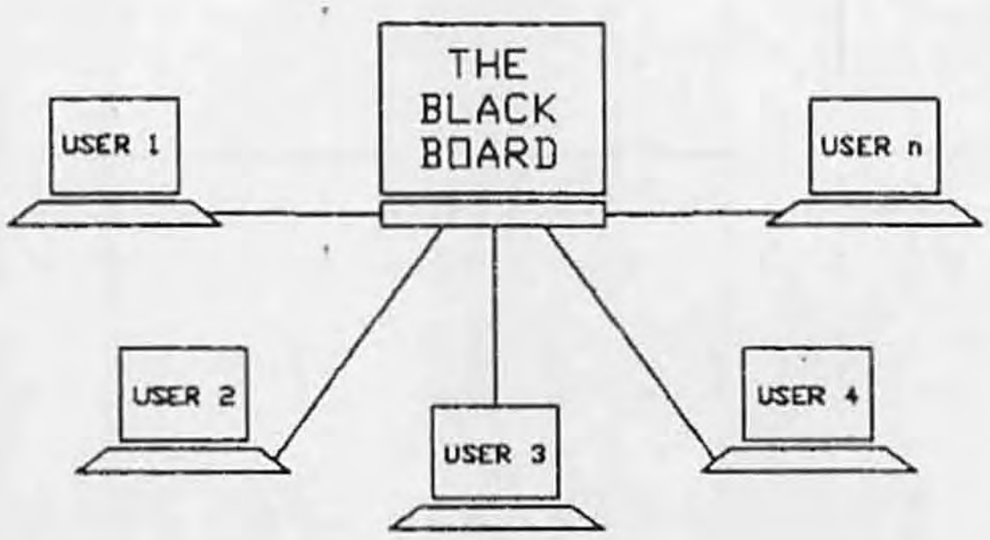
What is important to be pointed out, is the realization that there is no final solution to an architectural design problem, there is no definite stopping rule, there is no correct solution only good or bad, and the selection of an appropriate solution depends mainly on the designer's "weltanschauung", his own philosophy, perception and cognitive style. This theoretical limitation is applicable in the design of a CAAD decision support system as well.



THE ICAAD.DSS MODEL

The methodology to achieve the communication between the different components of the framework, as well as the control

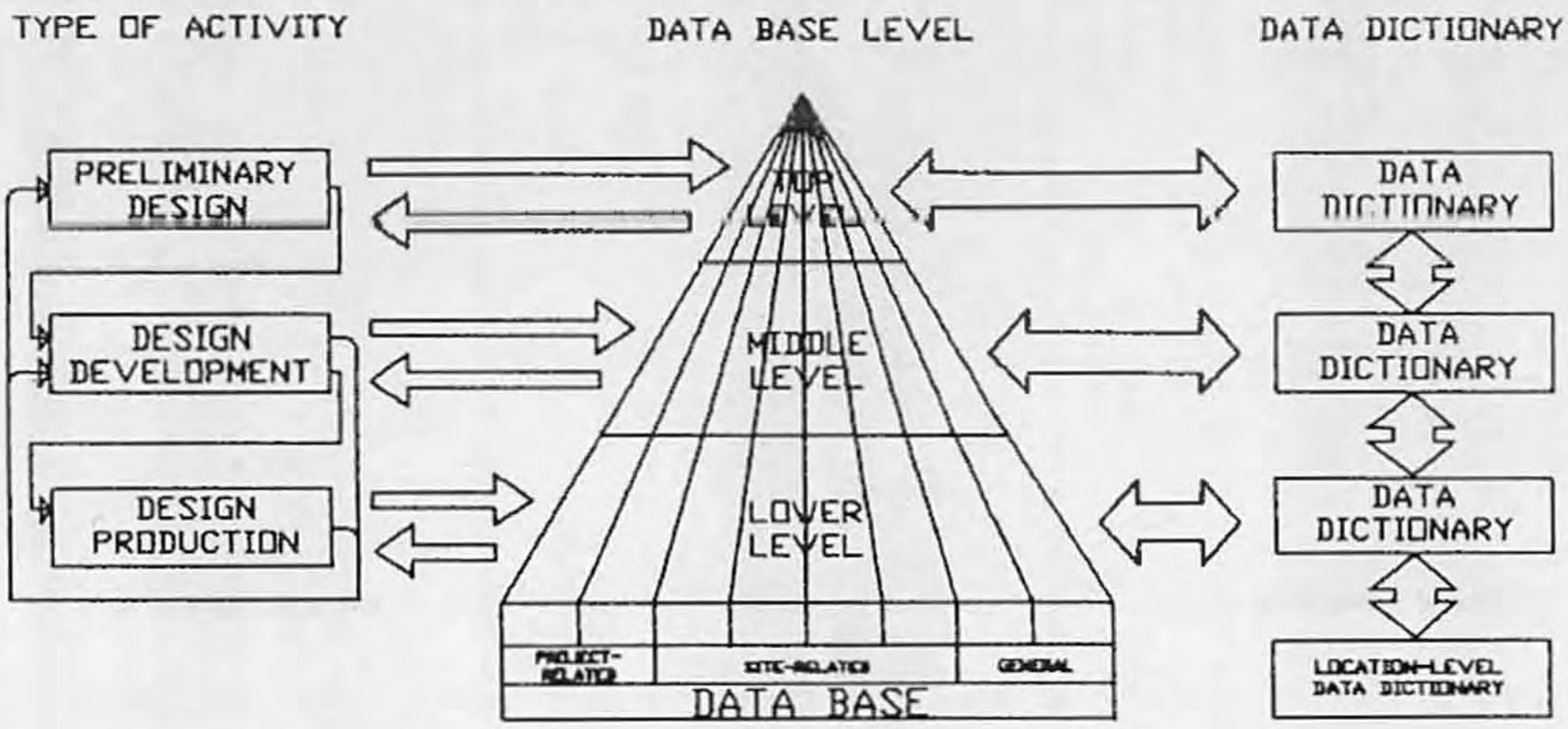
over them, is introduced through the use of a dialog generation and management system working through a blackboard architecture, in which the blackboard serves as a global representation of the design solution.



THE BLACKBOARD AS A GLOBAL REPRESENTATION

OF THE DESIGN SOLUTION

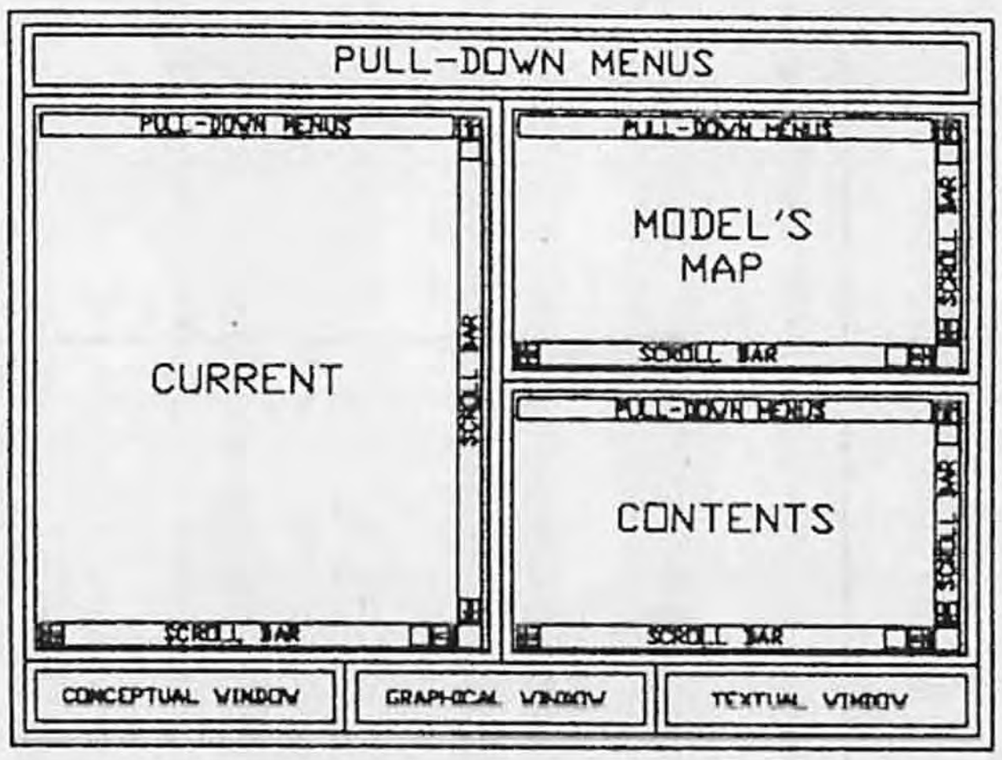
Chapter six introduces the importance of the moving, bi-directional database, as well as the data dictionary concepts, for achieving consistency and data integrity.



THE DATA DICTIONARY/DIRECTORY SYSTEM

THE MOVING-BIDIRECTIONAL DATABASE

The user interface, and the browser are addressed in chapter seven of the thesis.



THE USER INTERFACE

AND THE BROWSER

Chapter 8 presents the conclusions, a summary and a number of future tasks that need to be fulfilled in order to implement the proposed computing environment in a complete practical way.

This study provides a theoretical model which defines the general guidelines for the strategy and methodology needed for achieving a successful integration of computers within the architectural design process.

The study has addressed the structure of the proposed computing environment, and has defined the concept of the moving, bi-directional database for the decision support system, which is seen to be an important step towards the enlargement of data capabilities, as well as achieving a fully integrated environment for computing in architectural design.

RECOMMENDATIONS:

It is hoped that future Computer-Aided Architectural Design packages will automatically coordinate all organization's inputs, outputs, processing methods and procedures, to accomplish the organization's objectives. Even though, this tends to be utopian when carried out to its highest level of achievement, it is still an ultimate goal for a CAAD system designer and/or user.

It is also hoped that further studies, along with the results from this study will address the implementation of the proposed computing environment in a complete practical way. The first procedure would then be; choosing a specific design problem, then building a specific decision support system for that problem. For the implementation to be successful, some future research tasks have to be accomplished according to the general guidelines presented in this study:

Task 1. Defining and standardizing the minimum permissible amount of design variables present in a typical architectural design problem.

Task 2. Enumerating the standard data elements present within this architectural design problem.

Task 3. Defining linkages between the information within the standard databases and the degree of flexibility and integrity as a part of the data dictionary schema.

Task 4. Building the program bank, the expert system component, and the data bank; according to the needed task environment.

Task 5. Writing programs for the database management system (DBMS), the dialog generation and management system (DGMS), and the hypermedia's user interface.

Task 6. Building the active data dictionary/directory system (DD/DS), using the moving/bidirectional database concept.

It is assumed that various groups combining different disciplines, should simultaneously participate in, and contribute to the developments achieved so far, in order to make computing in architecture a reality in its proper context.

APPENDIX

THE BASED PROCESS AT ONE LEVEL FOR DIFFERENT TASKS

	SITE	PLANNING	STRUCTURE	ENVIRONMENT	SERVICES	MANAGEMENT	ECONOMICS
USER / BASED	Landscape architect	Architect	Structural Engineer, Architect	Architectural Acoustician, Lighting Engineer, Mechanical Eng.	Mechanical and Electrical Eng.	Project Manager, Design Manager, Quantity Surveyor Construction Manager, etc.	
BRIEFING	Function and Activities Social Organization Design Qualities: 'Image', 'Atmosphere', etc.		Codes Fire Standards	Environmental Standards Codes	Energy Standards	Contracting Strategy Economic Factors	Cost Limits, Budget Tax Factors Accounting
	Program for SITE	Program for PLANNING	Program for CONSTRUCTION SYSTEMS and MATERIALS	Program for ENVIRONMENTAL STUDIES	Program for SERVICES	Program for PROJECT MANAGEMENT	Program for COST CONTROL
ANALYSIS	Analyze separate programs to ensure consistency, completeness, adequacy of objectives, requirements and standards.						
SYNTHESIS	Develop range of alternative site treatment building locations	Develop range of alternative of plan forms building forms, etc.	Develop range of alternative construction concepts: frames, space truss, materials, etc.	Develop range of alternative energy systems	Develop range of alternative servicing concepts	Develop range of alternative management approaches	Develop range of alternative cost implications
EVALUATION	Review alternative design concepts, ensure compatibility between susystems. Discard design concepts not compatible with at least one alternative in each subsystem						
	Evaluate remaining alternatives against program	Evaluate remaining alternatives against program	Evaluate remaining alternatives against program	Evaluate remaining alternatives against program	Evaluate remaining alternatives against program	Evaluate remaining alternatives against program	Evaluate remaining alternatives against program
	Form permutations of design concepts: each permutation comprising one alternative from each design subsystem Evaluate each permutation for performance against program objectives, and program criteria						
DECISION	Select permutation that best satisfies program objectives and criteria						

FORMS SELECTED	DESIGN CONCEPT BUILDING SYSTEM CONCEPTS	MANAGEMENT STRATEGY
REPEAT AT OTHER LEVELS AND DEVELOP FURTHER		

Fig. (A.1): The BASED Process at One Level for Different Tasks

	SITE	PLANNING	CONSTRUCTION	SERVICES
PRELIMINARY DESIGN	Identify alternative possible sites. Locate the building.	Decide general character and kind of space and environment. Determine plan type and overall building form. Determine principal separations of planning subdivisions of the building. Plan major spacial units. Establish general massing.	Select constructional system.	Select building services systems. Select energy/fuel types to be used.
DESIGN DEVELOPMENT	Modify the site and landscape; earthmoving, planting, water, site construction, etc.	Determine building volumes, shapes, orientation, number and height of floors. Determine main circulation routes for people and others. Locate building's entrances, exits for people, material, goods, etc. Determine general arrangement of fenestration (solids and voids).	Select main constructional materials. Determine structural dimensions. Determine enclosure system, including walls, roofs and fenestration.	Select major mechanical, electrical, etc., plant and determine its location. Determine main circulation routes for services. Plan general distribution of mechanical, electrical, etc., services. Plan routes for all environmental services from mains to rooms and other spaces. Determine dimensions of service mains.
DETAILED DESIGN	Design plantings, and site construction in detail.	Design individual room spaces in detail.	Design other constructional elements: partitions, floors, ceilings, windows, sunshades, doors, etc. Determine constructional details: window and door openings, windows, doors, etc.	Design organization of services within ducts. Determine environmental services control systems. Select secondary environmental services equipment and determine its location. Select and locate switches, controls, etc. Determine detailed layouts of servicing routes from point of entry to rooms to terminals, outlets, etc.

Fig. (A.2): A Summary of Design Decisions Undertaken by the Design Sub-System

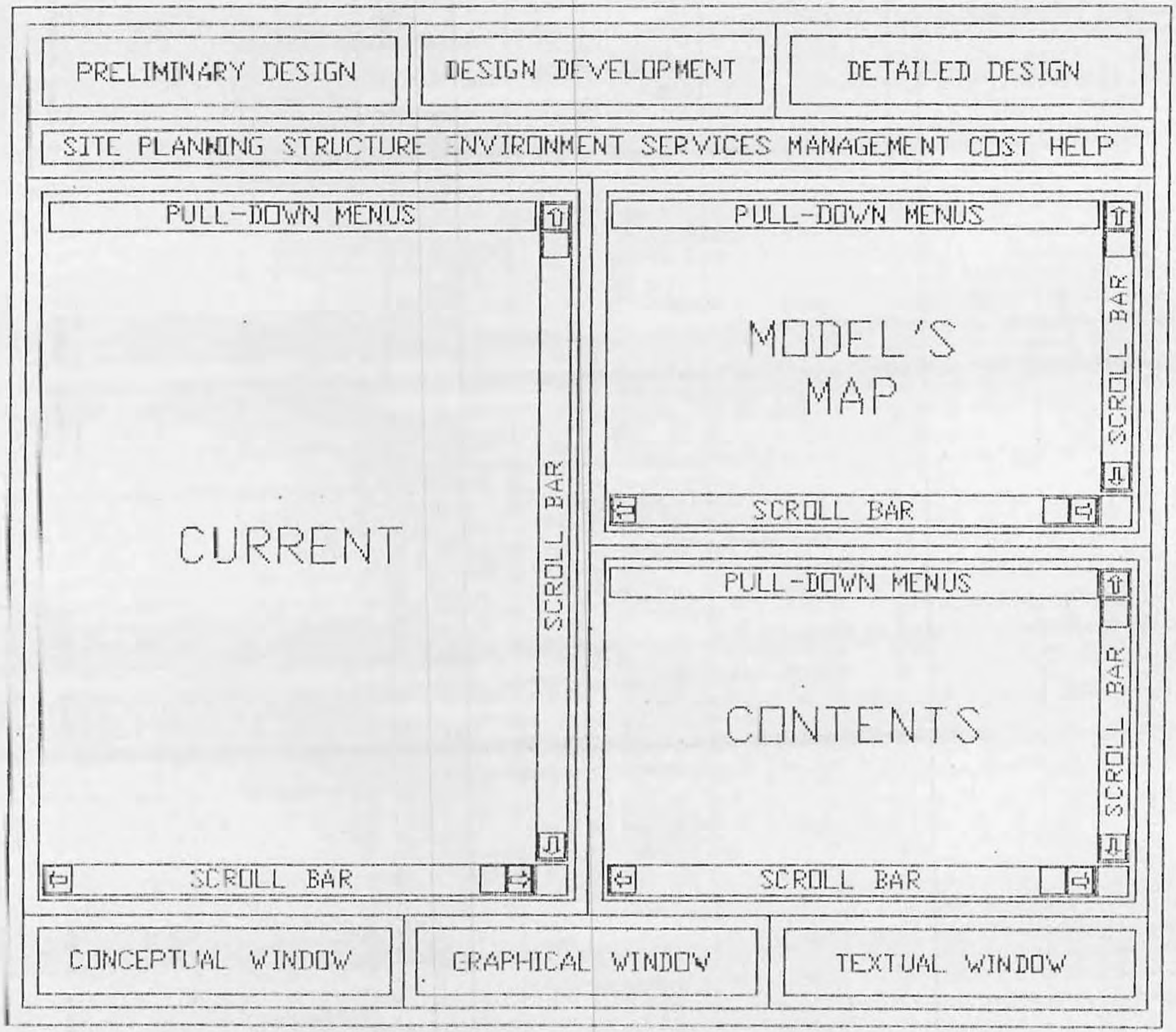


Fig. (A.3): The User Interface, anc the Browser.

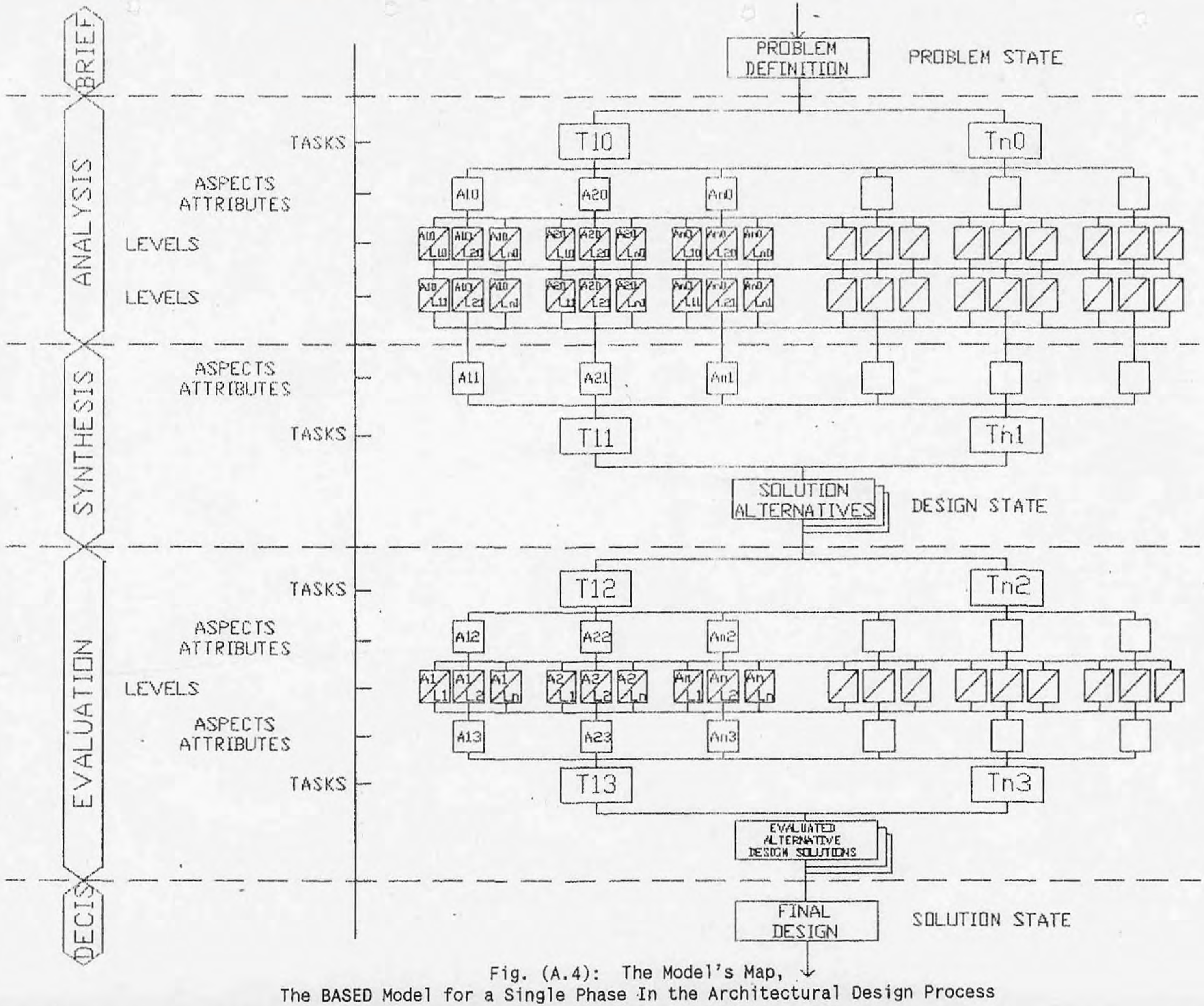


Fig. (A.4): The Model's Map,
The BASED Model for a Single Phase In the Architectural Design Process

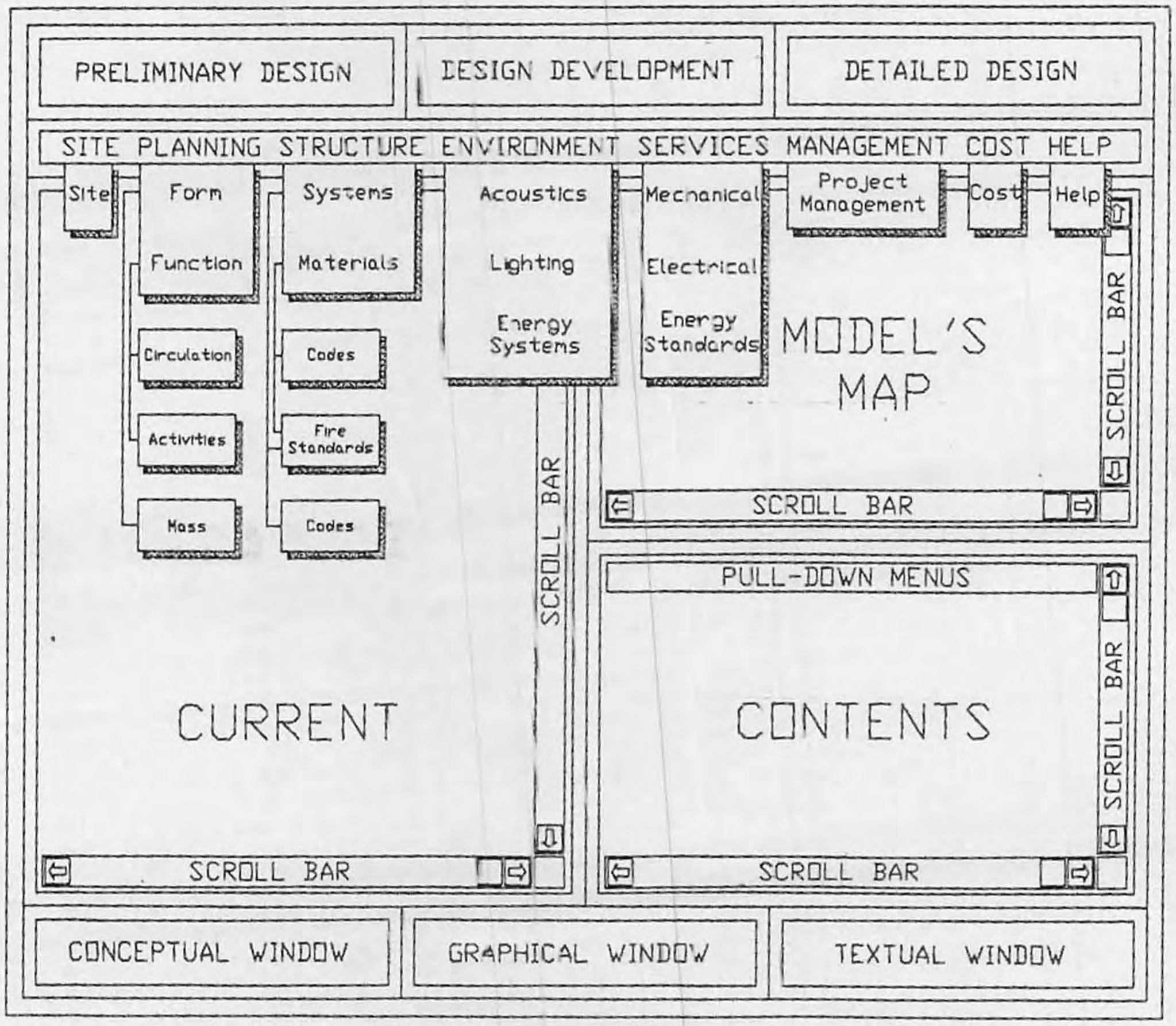


Fig. (A.5): The User Interface with the Screen Showing the Pull-Down Menus

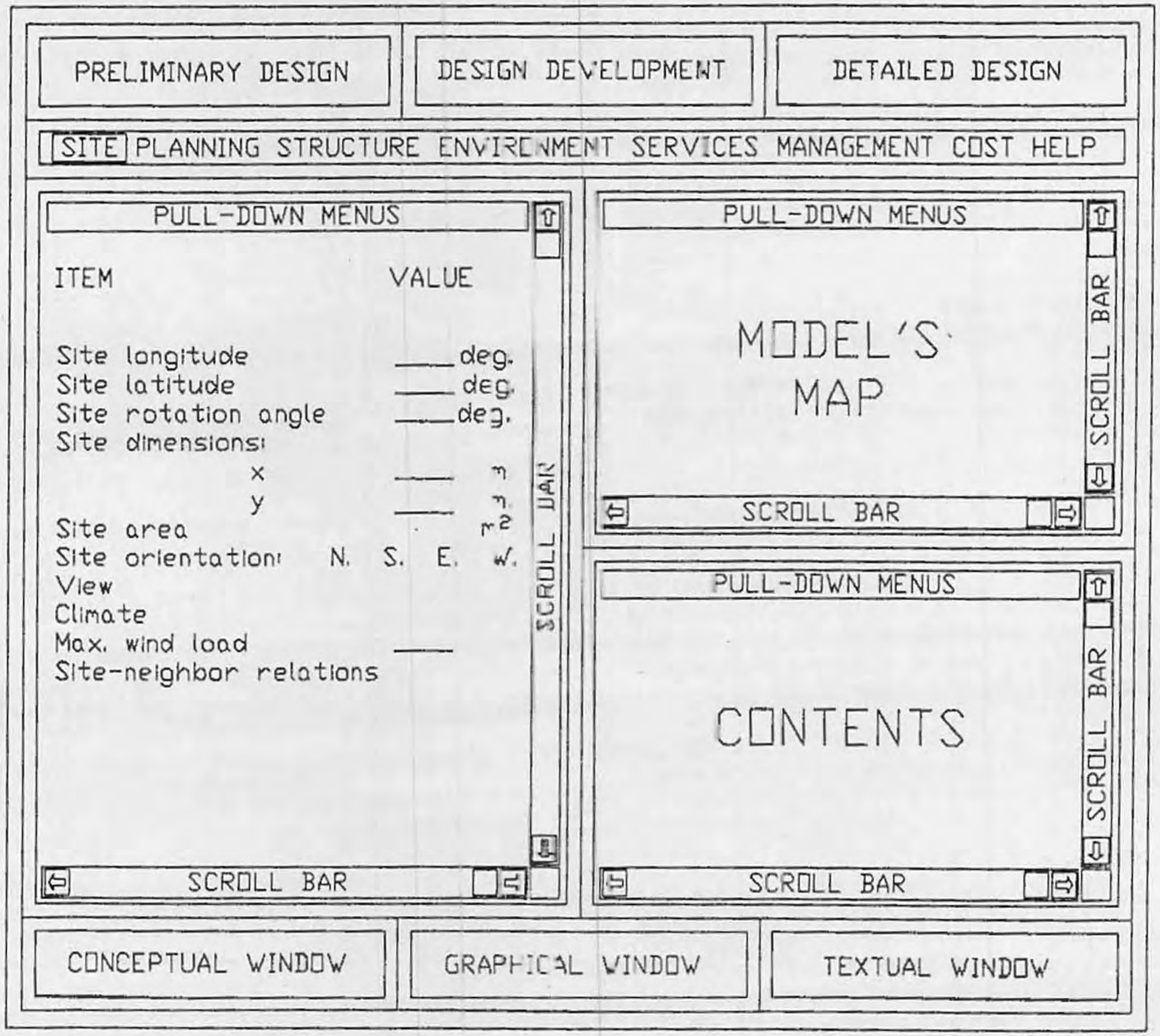


Fig. (A.6): The Screen Showing the "SITE" Item Highlighted

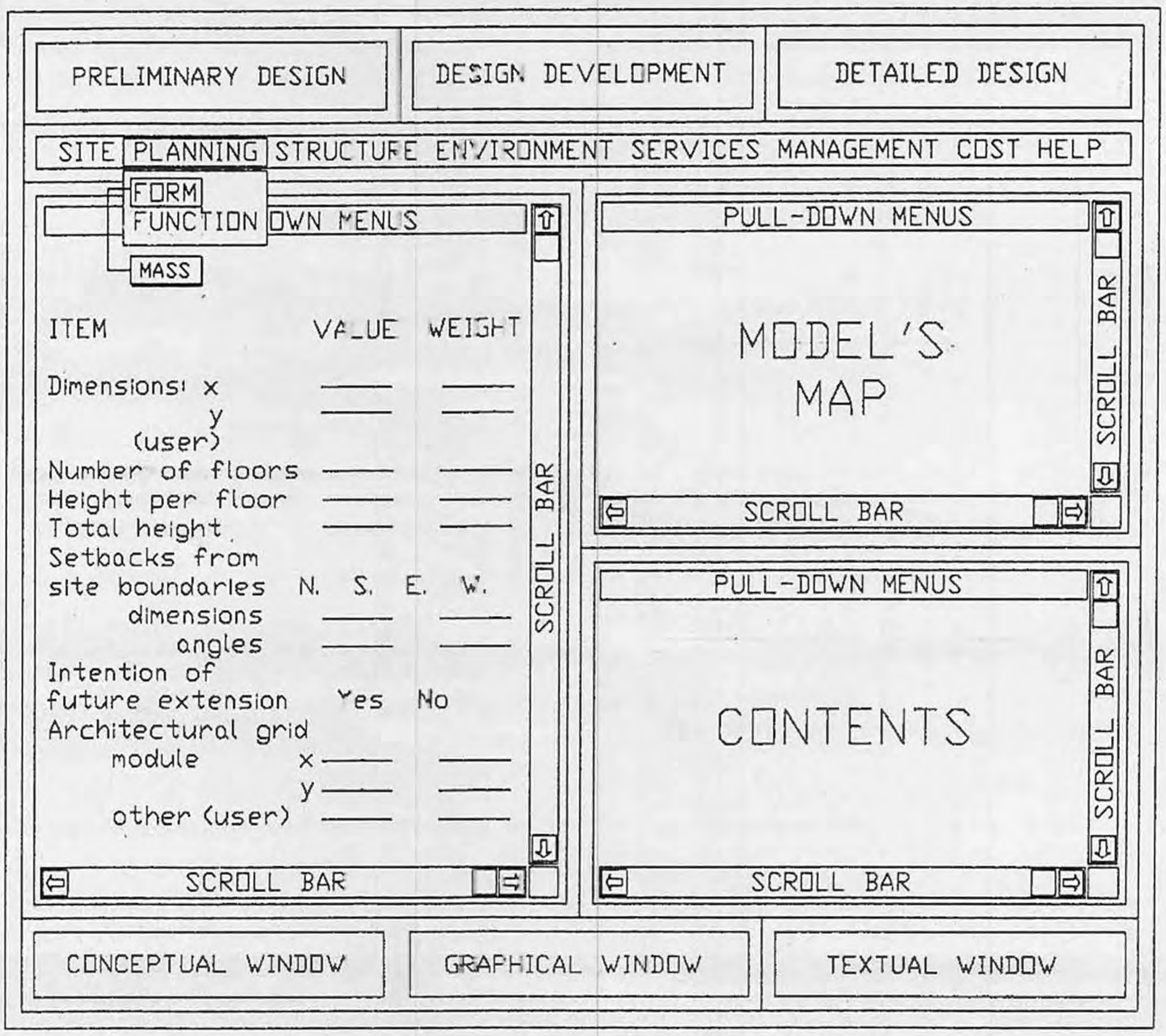


Fig. (A.7): The Screen Showing the "MASS" Item Highlighted
Under the "FORM" In the "PLANNING" Menu

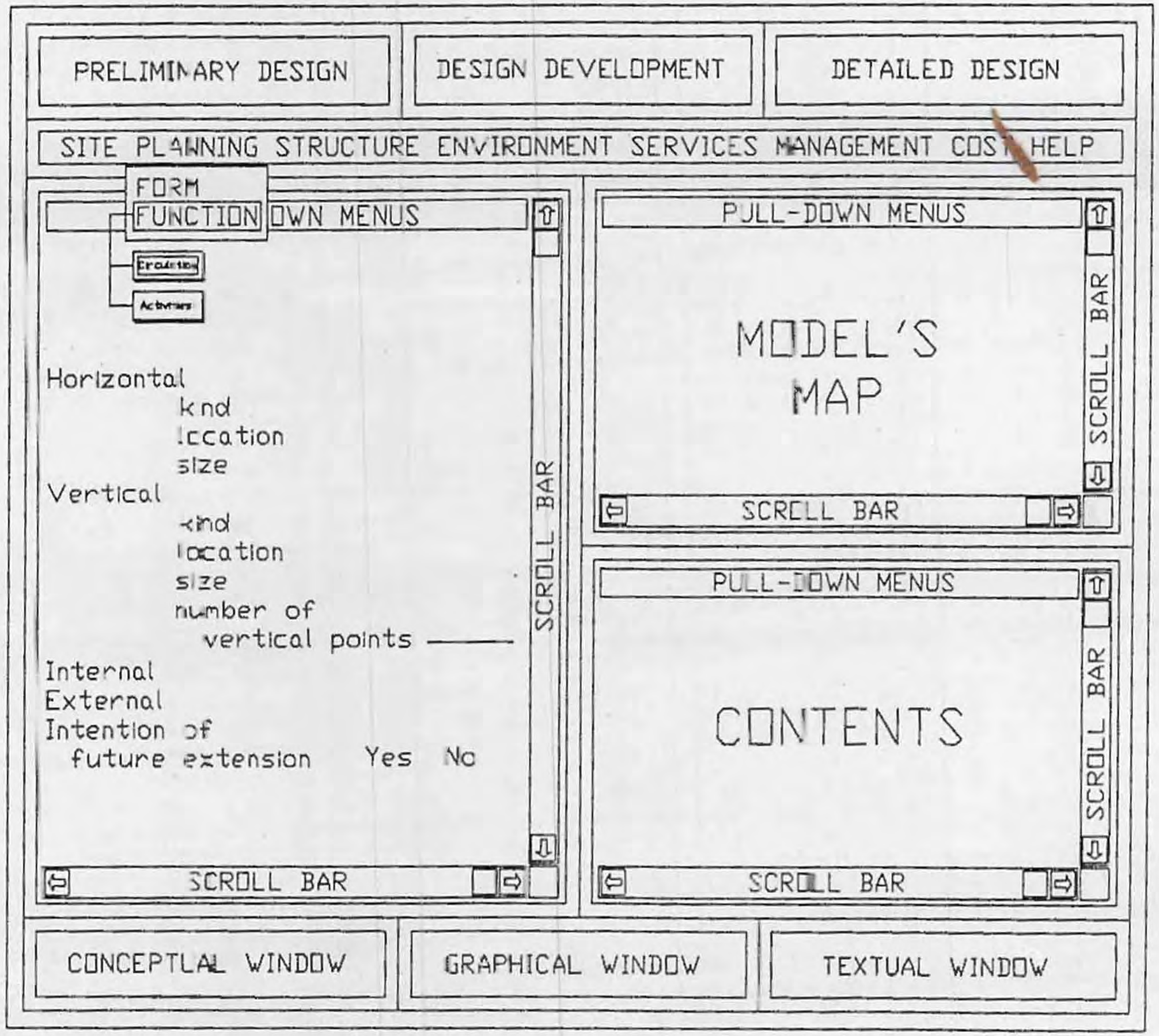


Fig. (A.8): The Screen Showing the "CIRCULATION" Item Highlighted Under the "FUNCTION" In the "PLANNING" Menu

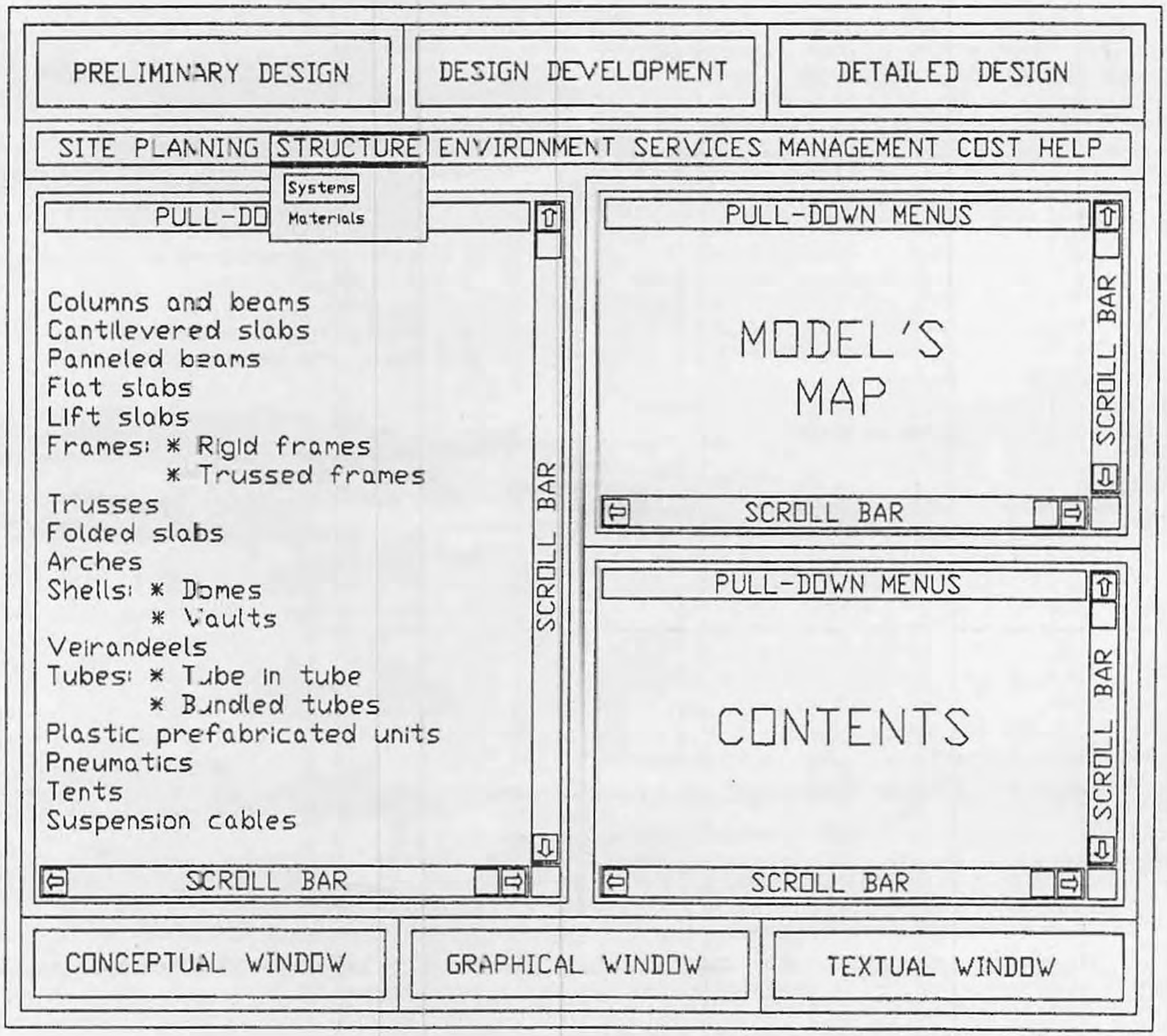


Fig. (A.9): The Screen Showing the "SYSTEM" of Construction Highlighted Under the "STRUCTURE" Menu

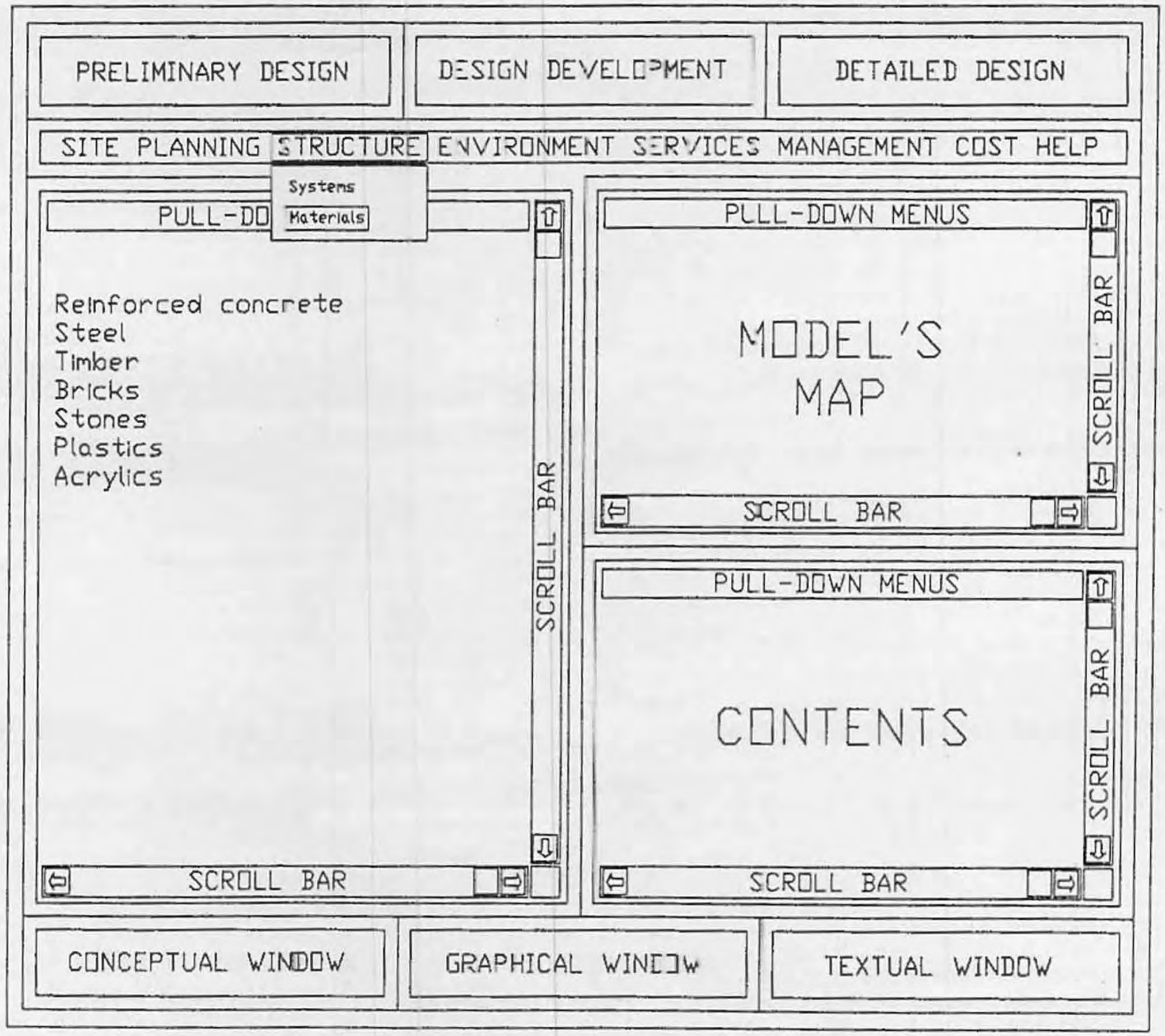


Fig. (A.10): The Screen Showing the "MATERIALS" of Construction Highlighted Under the "STRUCTURE" Menu

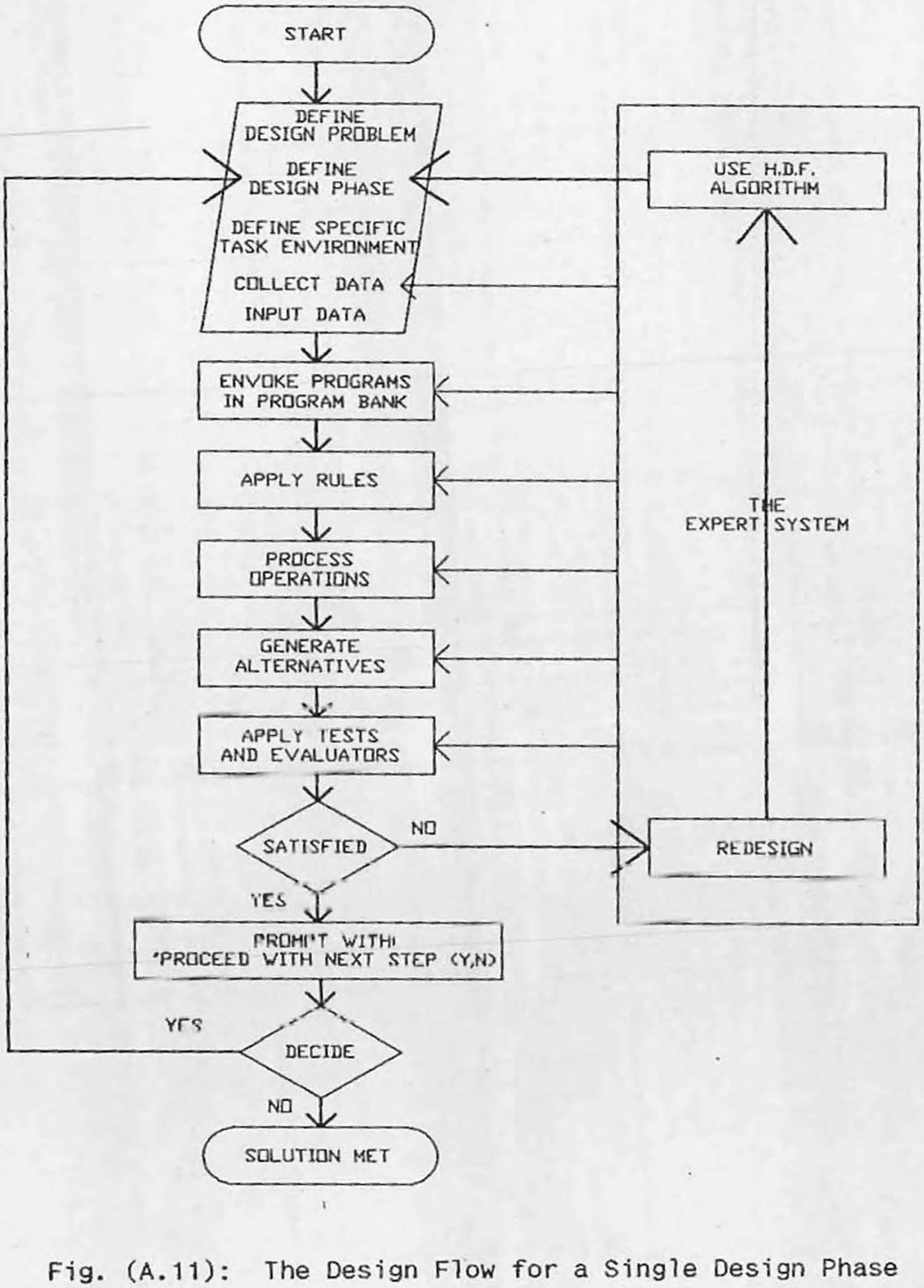


Fig. (A.11): The Design Flow for a Single Design Phase

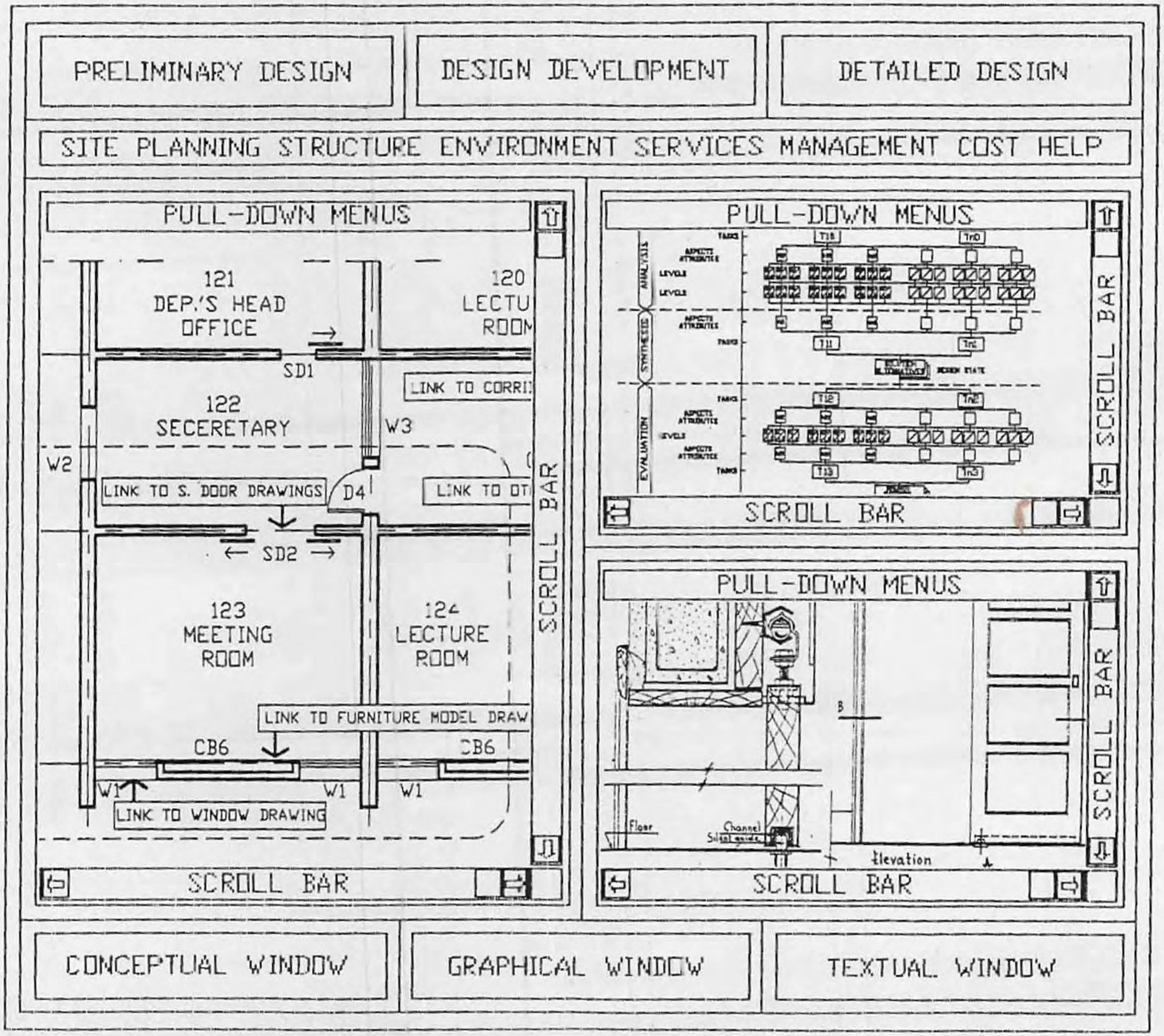


Fig. (A.12): The Screen Showing Different Information In the Different Windows

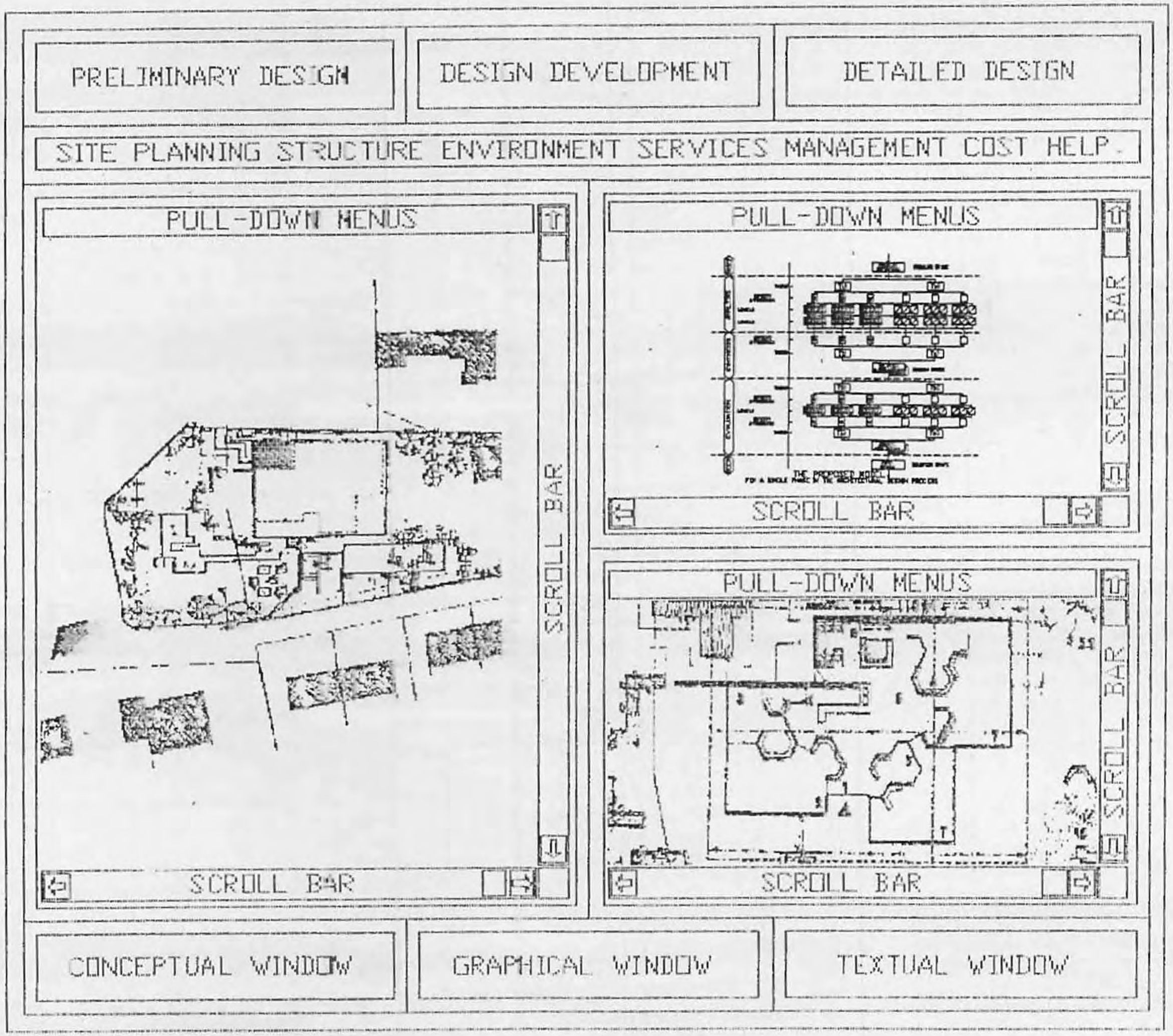


Fig. (A.13): The Screen Showing Different Information In the Different Windows

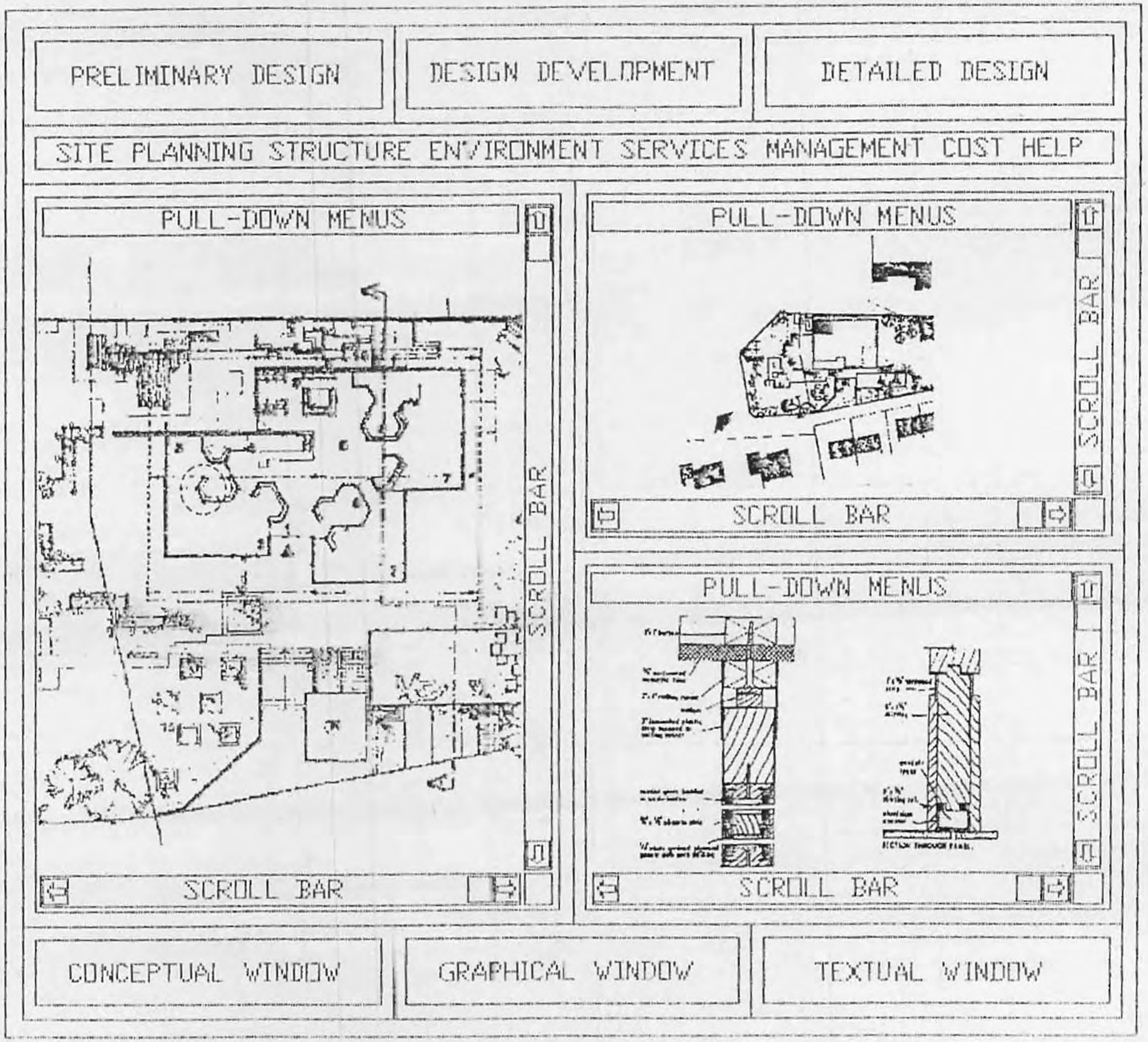


Fig. (A.14): The Screen Showing Different Information In the Different Windows

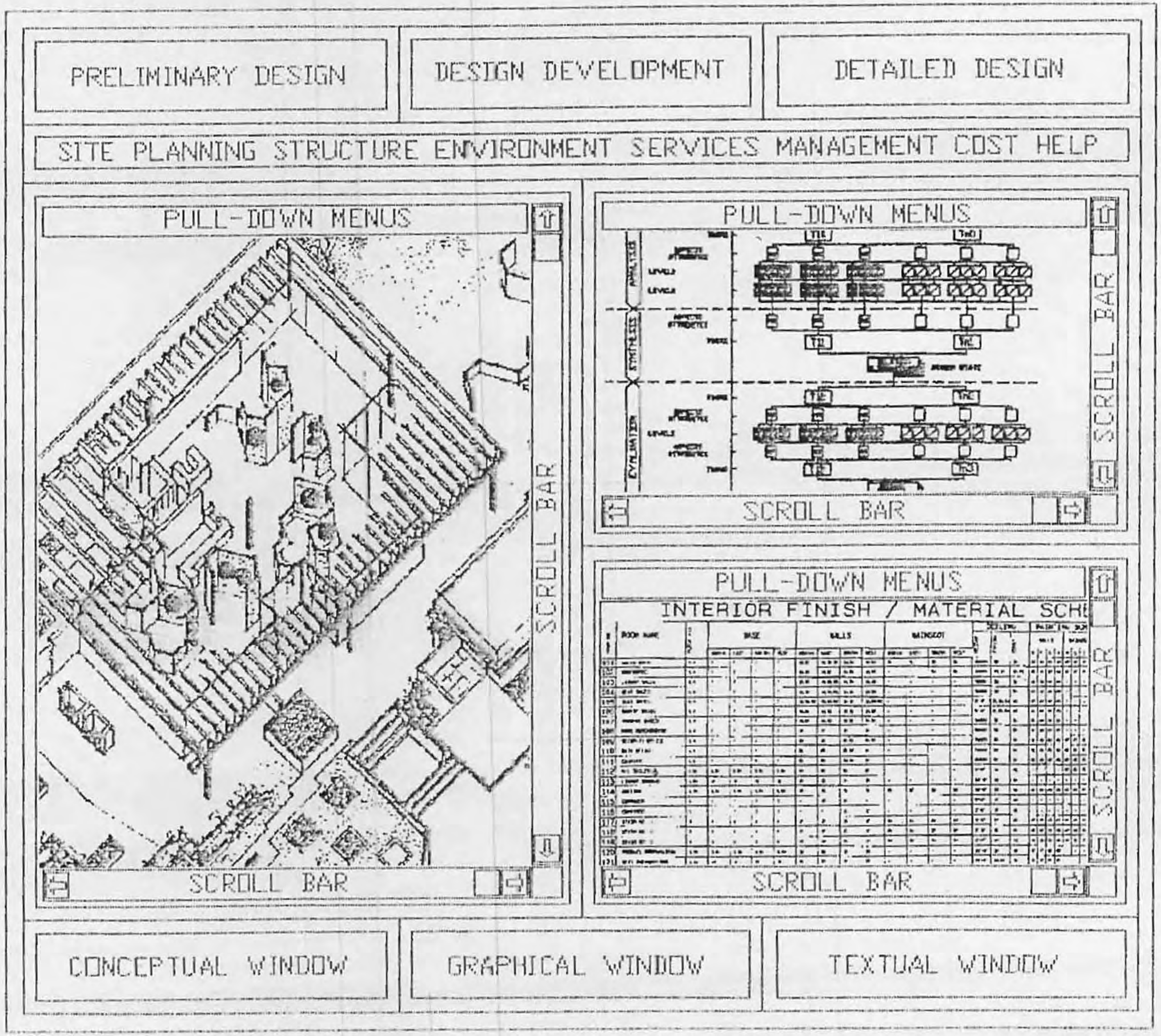


Fig. (A.15): The Screen Showing Different Information In the Different Windows

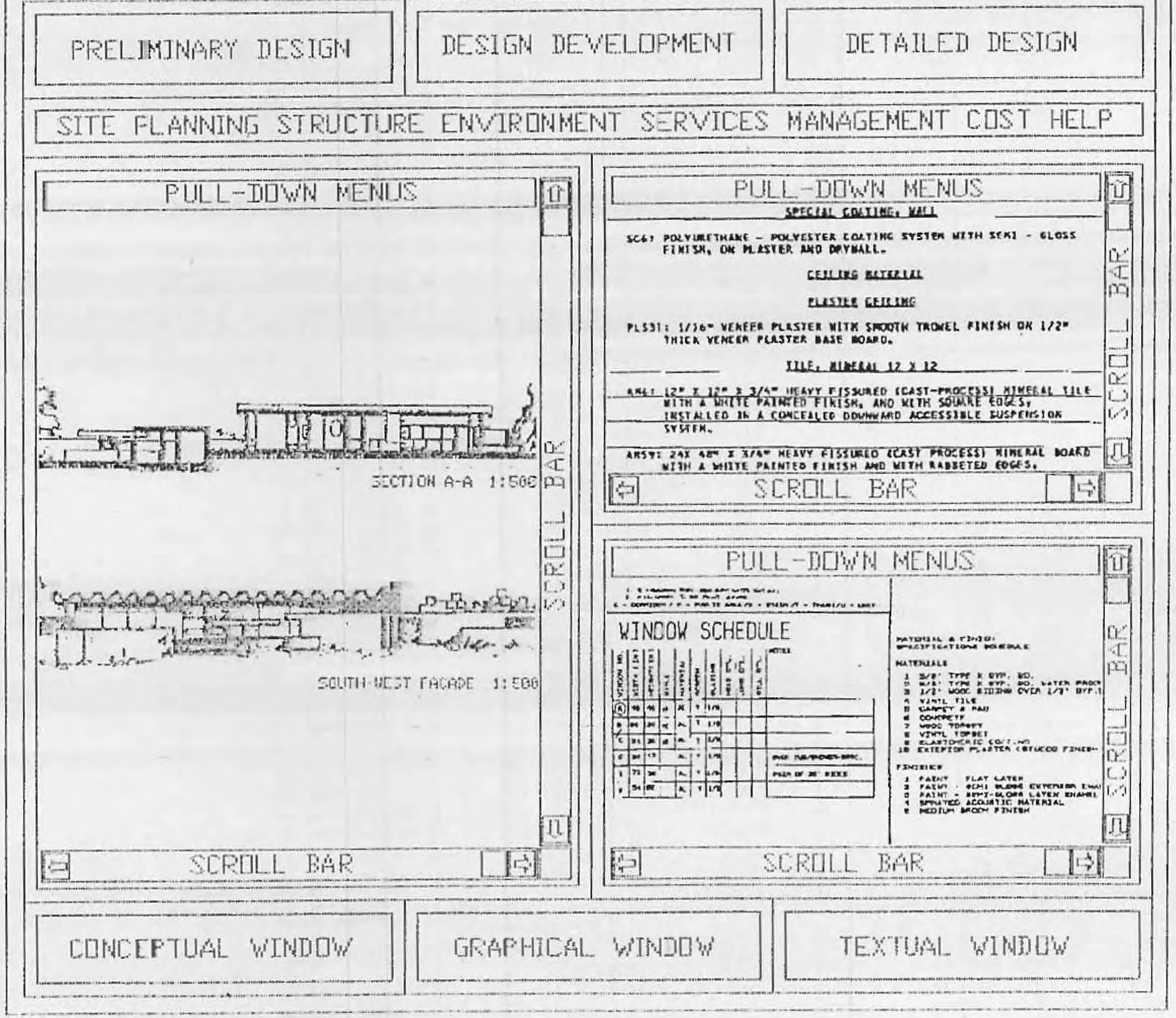
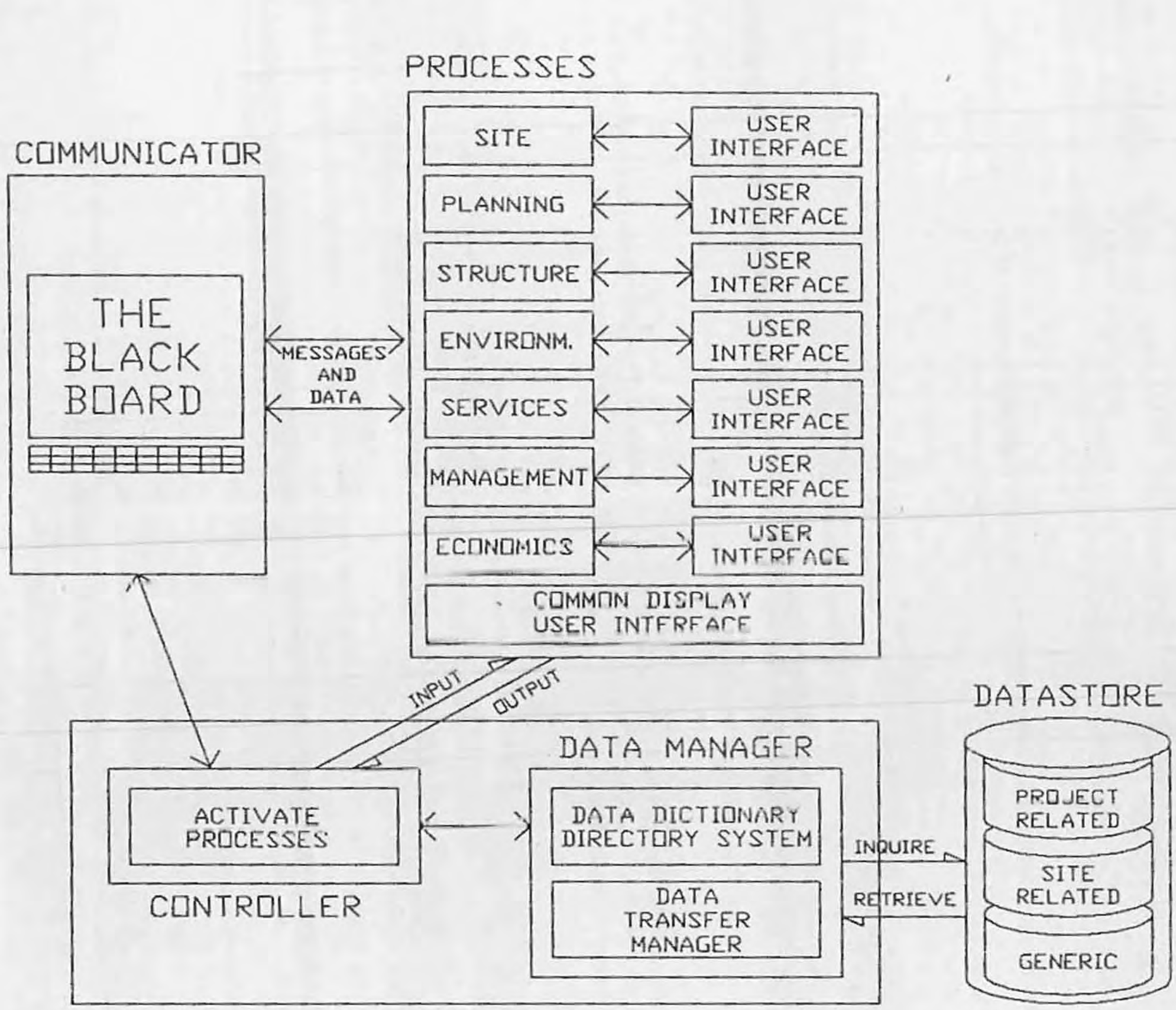


Fig. (A.16): The Screen Showing Different Information In the Different Windows



THE CONTROLLER AND THE COMMUNICATOR

الملخص

طرق التصميم كمدخل لعملية التصميم المعمارى بالإستعانة بالكمبيوتر

يقدم نموذج ال (ICAAD.DSS) الذى تم استعراضه فى الرساله ، اطارا جديدا لنظام تدعيم اتخاذ القرارات التصميميه فى العمليه التصميميه المعماريه . ويعتمد هذا الإطار على اتخاذ مدخل موحد للحوسبه فى التصميم المعمارى، الذى بدوره يعتمد على نظرة كلية لعملية التصميم المعمارى. هذا النموذج يهتم ب "الطريقه" اكثر من "النتيجه"، و يعتبر المشكلة التصميمية هى مشكلة بحث عن هدف و بحث عن طريقة مناسبة لتحقيق هذا الهدف بشكل يسمح للمهندس المصمم ان يحدد استراتيجية و طرق البحث عن حل مناسب لمشكلته التصميمه بالإسلوب الذى يراه .

و يشمل الإطار المقترح المستخدمين "USERS" باختلاف تخصصاتهم: مثل المهندس المعمارى و الإنشائى والصحى و الكهرباء و غيرهم، ثم حالات متعددة من مختلف المهام المقترنة ببيئة معينة "TASK ENVIRONMENT" مثل المهام الفنية أو الجمالية أو الإجتماعية ، ثم نظام تدعيم اتخاذ القرارات "DSS"، أو "DECISION SUPPORT SYSTEM". و لقد تعرض البحث بالتفصيل لمختلف مكونات نظام تدعيم اتخاذ القرارات، و هى:
مسهل الإتصال بين الكمبيوتر و المستخدم، بنك المعلومات، بنك البرامج التطبيقية ، نظام ادارة قاعدة البيانات والمعلومات، ثم وحدة النظم الخبيرة .

كذلك تم اقتراح وجود "الكشاف"، "BROWSER" و مهمته الرئيسية هى اظهار و تسجيل مختلف مراحل العملية التصميمية بصريا، حيث تظهر نقاط التجميع المختلفة للنموذج المقترح و كذلك مختلف الوصلات التى تربط بينها على الشاشة كما لو كانت نسيج لا نهائى غير محدد الإطراف.

و يتحقق الإتصال بين المستخدم و الكمبيوتر عن طريق وحدة اقامة و ادارة الحوار "DGMS" أو "DIALOG GENERATION AND MANAGEMENT SYSTEM" و قد أقترح لتدعيمها الفيزيائى نوع "السبورة"، "BLACKBOARD" حيث تستخدم السبورة كاداة اظهار شاملة للحل التصميمى.

اما بنك المعلومات، و بنك البرامج التطبيقية فسوف يكون التعامل معه على اساس "القراءة فقط" ، "READ ONLY" حتى لا يسمح بتغيير المعلومات و البيانات عن عمد أو عن غير عمد.

و تكون مهمة نظام ادارة قواعد البيانات هى خلق الملفات الجديدة و تصنيفها و ترتيبها، بالإضافة الى طلب المعلومات الجديدة و اصدار التقارير بطريقة عامة بدون طلب أى تغييرات فى البرامج الإصلية .

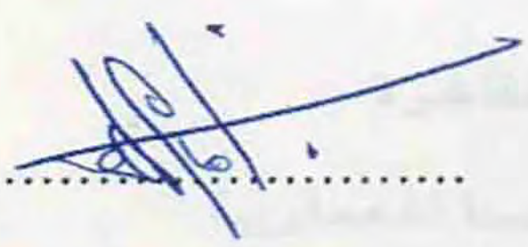
اما انسجام و توافق و تكامل المعلومات و صيانتها فسوف يتحقق عن طريق استخدام فكرة قاعدة البيانات تنائية الحركة .

هذا البحث يقدم مدخلا جديدا لإستخدام و تكامل الكمبيوتر فى عملية التصميم المعمارى.

لجنة المتحنين

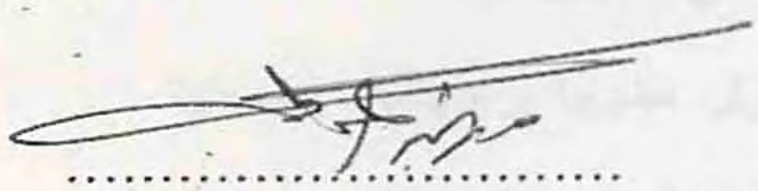
١ - أ . د . امام محمد شلبى

استاذ ورئيس قسم الهندسة المعمارية

كلية الهندسة - جامعة عين شمس ..... 

٢ - أ . د . محمد توفيق عبد الجواد

استاذ ورئيس قسم العمارة

كلية الفنون الجميلة - جامعة حلوان ..... 

٣ - أ . د . محمد زكى خواس

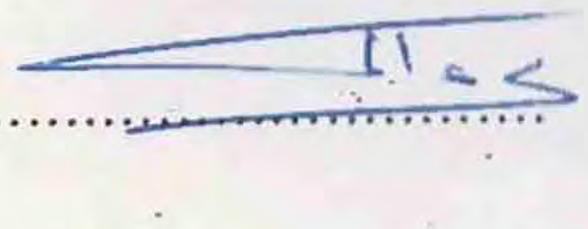
استاذ العمارة بقسم الهندسة المعمارية

كلية الهندسة - جامعة عين شمس ..... 

٤ - أ . د . محمد عبد الحميد شعيره

استاذ التحكم الآلى بقسم هندسة

الالكترونيات والحاسبات

كلية الهندسة - جامعة عين شمس ..... 

التاريخ ١٩٩١/٣/٢

تعريف بالباحث

الاسم : سمير صادق حسنى

تاريخ الميلاد : ١٤ ديسمبر ١٩٥٤

محل الميلاد : مصر الجديدة - القاهرة

الدرجة الجامعية الأولى : بكالوريوس الهندسة المعمارية

الجهة المانحة للدرجة الجامعية الأولى : قسم الهندسة المعمارية - كلية الهندسة - جامعة عين شمس تاريخ المنح : يونيو ١٩٧٧

الشهادات الأخرى الحاصل عليها وتواريخ الحصول عليها وجهات منحها :

ماجستير الهندسة المعمارية - قسم الهندسة المعمارية .

كلية الهندسة - جامعة عين شمس سنة ١٩٨٧

الوظيفة الحالية : مدرس مساعد بقسم الهندسة المعمارية

كلية الهندسة - جامعة عين شمس

التوقيع :

التاريخ :

جامعة عين شمس
كلية الهندسة

أساليب التصميم المعمارى بالاستعانة بالكمبيوتر

رسالة مقدمة من

مهندس / **سمير صادق حسنى**
المدرس المساعد بقسم الهندسة المعمارية
جامعة عين شمس

للحصول على درجة دكتوراه الفلسفة

فى العمارة

إشراف

أ . د . **محمد زكى حواس**
استاذ العمارة بقسم الهندسة المعمارية
كلية الهندسة – جامعة عين شمس

أ . د . **محمد عبد الحميد شعيره**
استاذ التحكم الآلى بقسم هندسة الالكترونيات والحاسبات
كلية الهندسة – جامعة عين شمس

القاهرة

١٩٩٠

بسم الله الرحمن الرحيم

The impact of the new methods on design was limited to the resolution of puzzles: such as, given a set of rental terms, a range of cost/unit, and certain regulations as to the minimum number of units of each type, what is the best mix of flats in a new development?, in terms of return on investment, where a "best" answer can be obtained, by using linear programming methods, or the more complex site-feasibility analysis using iterative methods reported by Gero, and James (1972).

Another example would be: given a number of sets of elements for roof constructions, some of which are compatible, what is the most economic practicable combination?, a puzzle that can be solved by using the AIDA method, or Analysis of Interconnected Decision Areas (Luckman, 1969). Such methods provide quicker and more reliable solutions to puzzles which would previously have been solved by trial and error, or by guesswork. It is important to point out here that these methods did not deal with the "outline and scheme stages of architectural design", but confined themselves to the better constrained tasks of detailed design (Heath, 1984). The reason for this is that the "sequence of activities was unique to each project and depended mainly on the constraint that was the most important, whether this constraint was decided to be so by the client or the designer himself." (Rittel, 1972). In fact "first generation methods seem to start once all the really difficult questions have been dealt with already." (Rittel, 1972).

First generation methods were suited to the solution of well-constrained, or "well-behaved" (Rittel, 1972), or "well-structured" (Simon, 1973) problems. On the other hand, the intention of "second generation" methods was to extend the scope of method to ill-constrained, ill-behaved, or "wicked" problems.

According to Simon (1972), it is impossible to construct a formal definition of "well-structured" problems. However, it is possible to set forth a partial list of requirements or characteristics that have been proposed at one time as criteria that a problem must satisfy in order to be regarded as well-structured. A further element of indefiniteness and relativity arises from the fact that the criteria are not absolute, but generally express the relation between characteristics of a problem domain, on the one hand, and the characteristics and power of an implicit or explicit problem-solving mechanism, on the other. With this approach, it is possible to say that a problem may be regarded as well-structured to the extent that it has some or all of the following characteristics:

1. There is a definite criterion for testing any proposed solution, and a mechanizable process for applying the criterion.
2. There is at least one problem space in which can be represented the initial problem state, the goal state, and all the other states that can be reached, or considered, in the course of attempting a solution to the problem.
3. Attainable state changes can be represented in a problem state, as transitions from given states to the states directly attainable from them.
4. Any knowledge that the problem solver can acquire about the problem can be represented in one or more problem spaces.

As stated earlier, these criteria are not entirely definite. But this vagueness and relativity, simply reflect the continuum of degrees of definiteness between the well-structured and the ill-structured ends of the problem spectrum, and the dependence of definiteness upon the power of the problem solving techniques that are available.

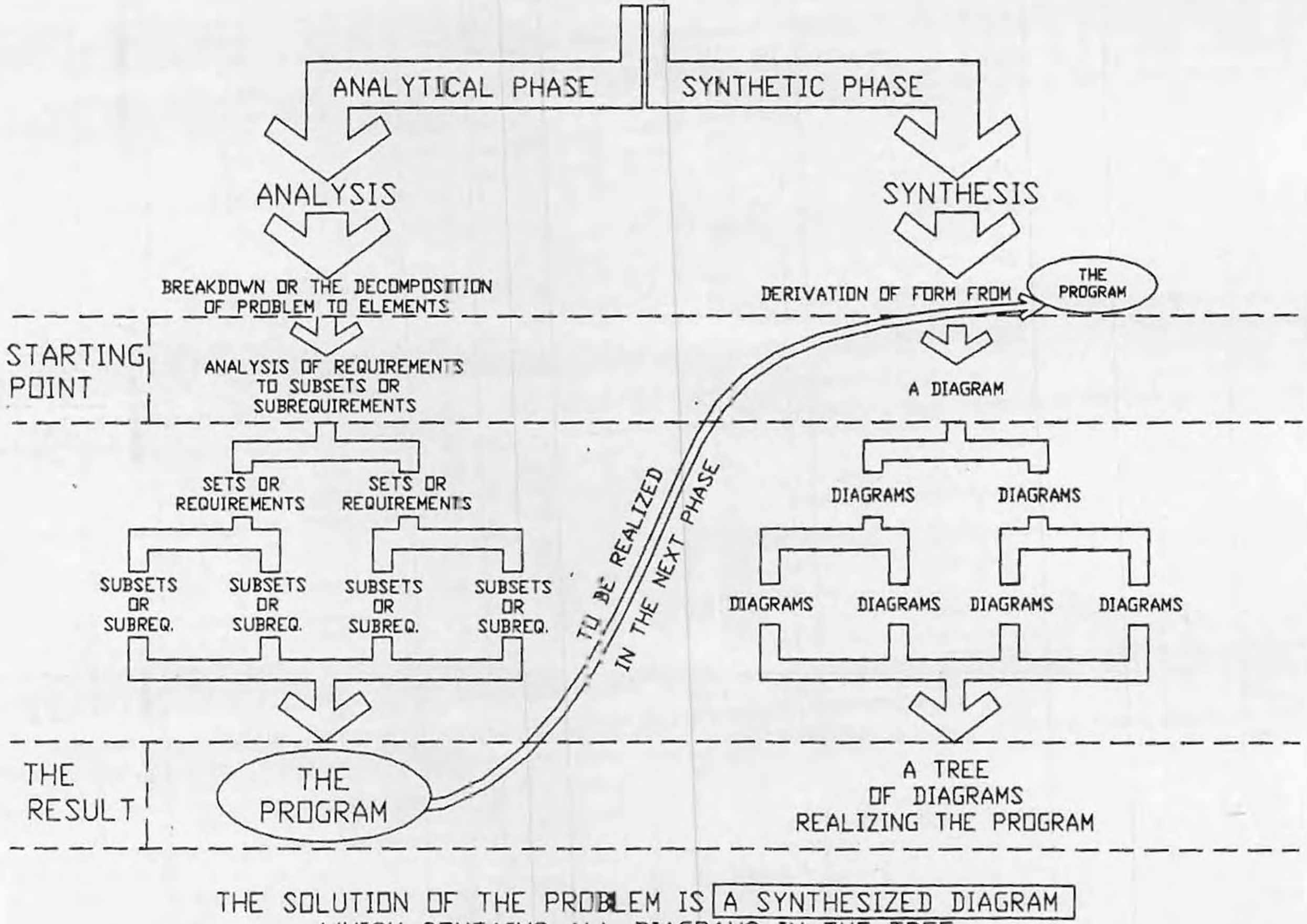
2.3. Early Design Models:

2.3.1. Alexander's Model; The Synthesis of Form, The Decomposition Theory:

Perhaps the most well known model of the design process, is the one which was described by Christopher Alexander (1971), who claimed that there was a very important underlying structural correspondence between the pattern of a problem and the process of designing a physical form which answers the problem. Alexander's thinking was based on the premise that "it is not only the result which is important, but the process too" (Alexander, 1971, p. 133). The form of the path which leads to a solution is as important as the solution itself.

Alexander's definition of the architectural design process consists of two phases. The first phase is analysis, and the second phase is synthesis. Every problem, can be "decomposed" in a way unique to the problem. The analytical phase of the design process consists of the decomposition of the problem into elements and subsets, and the establishment of a hierarchy among them. (Each element of the decomposition is a subset {sibling node} of the subsets above it {parent node} in the hierarchy of the problem). Figure (2.1) illustrates that the analytical phase begins with an analysis of problem solving requirements which results in a "program" that is realized in the synthesis phase, during which a form is derived from the program. The starting point of this phase is a "diagram", and the result is a tree of diagrams, which represents the "realization" of the program.

The critical activity of this phase is the matching of the requirements of the program with their corresponding diagrams. The solution of the problems is a synthesized diagram, that contains <u>all</u> diagrams in the tree.



THE SOLUTION OF THE PROBLEM IS A SYNTHESIZED DIAGRAM
WHICH CONTAINS ALL DIAGRAMS IN THE TREE
BESIDES, IT SHOULD SEARCH FOR A HARMONY BETWEEN FORM AND CONTEXT

Fig. (2.1): Alexander's Model.

The essence of Alexander's model of the design process can best be explained by describing the process a designer would follow when employing the model in the design of a family residence. The residence is supposed to provide a shelter for the family. It is also expected to provide privacy, offer comfort, and match the lifestyle of its inhabitants. Each of these general requirements must be broken down into different sub-requirements.

According to Alexander's model, the designer would first organize a complete list of requirements, enumerate the elements in the list and then cross-relate all of these elements. This permits him (the designer) to establish the elements which form sets and a hierarchy for these sets. For example, the requirement for comfort might mean that views should be provided by some windows, or that bathrooms should be placed on every floor. On the other hand it might mean that seating should be arranged in such a way that the residents sit comfortably when looking at the views through the windows, or that each bathroom should have two sinks. Each of these requirements would be an element in Alexander's "program" and would all form a hierarchical structure: the existence of a view through a window obviously comes before seating arrangements that take advantage of the view, while the existence of a bathroom comes before the requirement to provide two sinks.

Once the "program" has been established, the designer can proceed with finding diagrams that represent each element, and each set of elements substituting a list of verbally depicted requirements with an equivalent set of diagrams organized as a tree. The only remaining step would be the translation of these diagrams into floor plans, sections, and elevations. The tree of diagrams will dictate the actual form of the house.

Alexander's definition of the design process represents an attempt to provide a conceptual framework for arriving at "explicit maps of the problem structure". Essential to the process of designing is the search for fitness between form and context: a search for harmony between these two intangibles, with varying degrees of success. A "good" fit may eventually be achieved through trial and error. According to Alexander, the process of trial and error is too slow, too expensive, and results in too many highly undesirable misfits. He attempted to improve the process of design in order to reduce the number of misfits through the definition of a formal method.

In his book Notes on the Synthesis of Form, Alexander attempted to find out "what's actually going on" in good design. The problem of design is to fit the form to its context. Form, as Alexander defines it (Alexander, 1971), is that part of the environment over which the designer has control; context is that part of the environment which places demands on the form. A good design is defined as a good fit, one in which form and context are in frictionless coexistence. "A well-designed house not only fits its context well but illuminates the problem of just what context is" (Alexander, 1971). Through the method proposed by Alexander in this book, the design problem becomes one in which the designer's purpose is to achieve "fitness" between the two entities: the form in question, and its context. In combination, these two entities form the ensemble (Studer, 1965). The validity of the design can be measured only within the ensemble and only by a condition of fit. The design problem is presented as a task of avoiding a number of specific potential misfits in the ensemble. These misfit variables form a set which is called 'M' in this method. The interaction among the variables are denoted as intervariable (causal) links, and are represented by associating 'M' with a second set defined as 'L'. 'L' consists of non-directed, signed, one-dimensional elements

called links. Each link joins two elements of 'M', and contains one other element of 'M'. The two sets define a linear graph G (M,L).

When these interactions become numerically dense, they form discrete, hierarchical, sub-systems. Since the designer lacks the cognitive capacity to manipulate such complexity, the solution to the problem is to construct a symbolic representation of the problem's structure. This symbolic representation is the decomposition of the problem into smaller manageable subsets such as a hierarchical tree of interacting sets (Summers, 1988).

The decomposed problem is the program. It is a "reorganization of the way the designer thinks about a problem...it gives us a series of sub-problems and tells us in what order to solve them" (Alexander, 1971,p. 93).

The synthesized process program is accomplished by interpreting the decomposed sub-systems into a series of "constructive diagrams", which are subsets of 'M', because they contain fewer requirements than 'M' itself, and less interaction between them, is simpler to diagram than 'M'.

It is therefore natural to begin by constructing diagrams for the smallest sets prescribed by the program. If we build up compound diagrams from these simplest diagrams according to the program's structure, and build up further compound diagrams from these in turn, we get a tree of diagrams. This tree of diagrams contains just one diagram for each set of requirements in the program's tree. This is called the realization of the program, as in fig. (2.2).

It is possible to bring out the contrast between the analytical nature of the program and the synthetic nature of its realization. From fig. (2.2), the diagram on the top

shows that the tree of sets is obtained by successive division and partition. The tree of diagrams, on the bottom is made by successive composition and fusion. At its apex, is the last diagram, which captures the full implications of the whole problem, and is therefore the complete diagram for the form required.

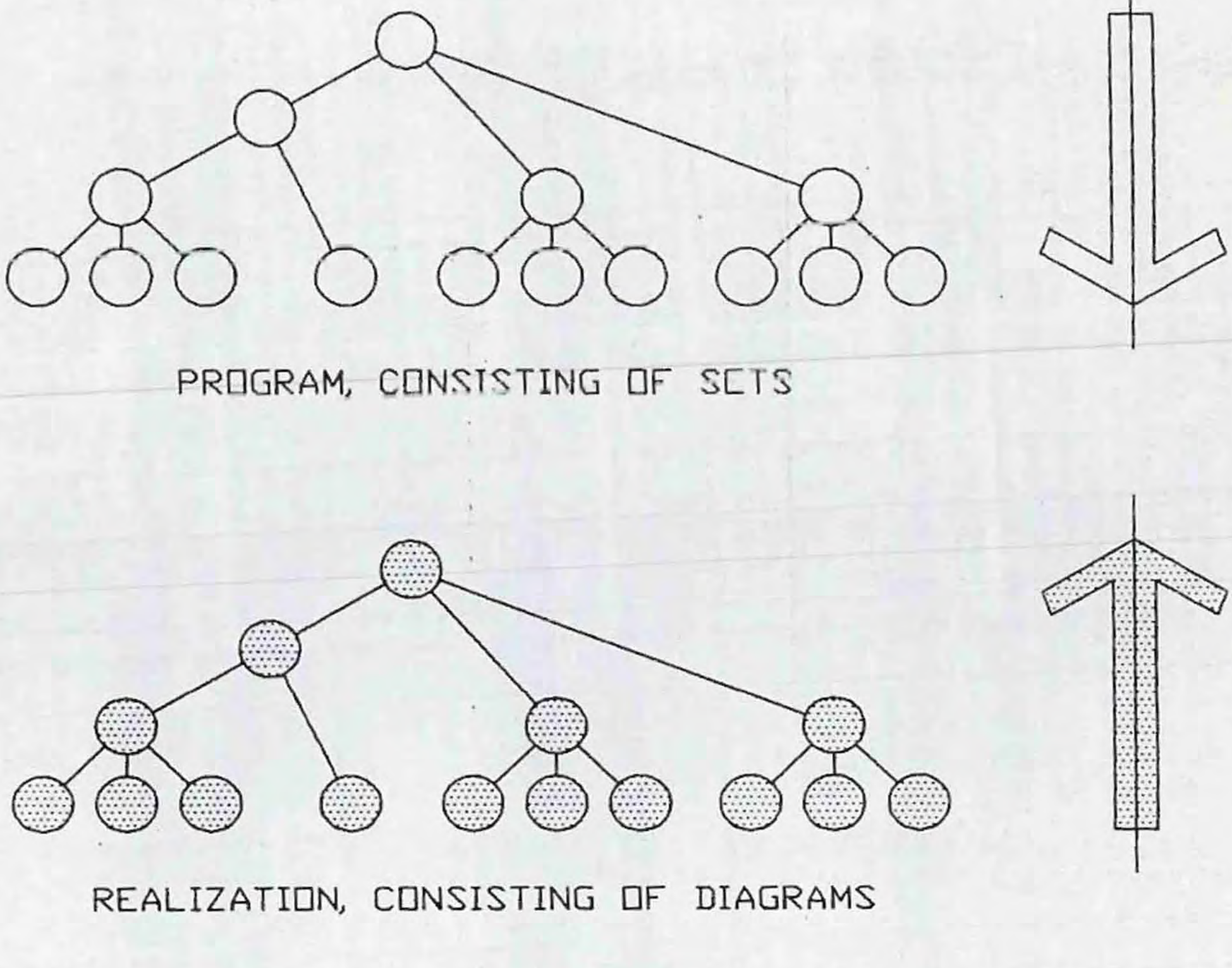


Fig. (2.2): The program consisting of sets, and the realization, consisting of diagrams.

2.3.1.1. Criticism of The Decomposition Theory:

In general, there are five criticisms of the hierarchical decomposition theory as stated by Heath (1984). First, it is difficult to be sure that all "misfits" have been found. Second, information about what "fits" may be just as important as what "misfits" because if what is right about a situation is ignored then, correcting a misfit may produce another, different misfit. Third, the world is not divided into simply good and bad, fit and misfit. Fourth, real environment behavior systems are too complex to be represented in a tree pattern. Fifth, the assumption that is always possible to find a solution at any level of recombination that will not compromise a solution at another level cannot be justified.

2.3.2. Broadbent's Model:

Broadbent's approach to design methods could be described as common sense (Heath, 1984). He does not believe in any one ideal design process, he defines design goals as "client motivation", and "user requirement". User needs provide a superior guide to how one should design than the client's initial motivation (Broadbent, 1973). The process which Broadbent describes in detail, stems from a consideration of user needs, though he is careful to specify that there are other processes, and he provides a chart of the three essential systems; environmental, building, and human, and their different elements, as in figs. (2.3).

According to Broadbent, design can commence at any point in the chart; for example, the designer may start by "knowing" the system of construction.

He suggests that simple design investigation can be guided by precedents; the building type, the client's own building if he already has one, and so on.

ENVIRONMENT SYSTEM		BUILDING SYSTEM		HUMAN SYSTEM	
CULTURAL CONTEXT	PHYSICAL CONTEXT	BUILDING TECHNOLOGY	INTERNAL AMBIENCE	USER REQUIREMENTS	CLIENT OBJECTIVES
	The site as given in terms of:	Modifications of external environment to provide suitable ambience for specified activities by means of:	Provision of physical conditions for performance of activities in terms of:	Provide for specified activities in terms of the following needs:	Return for investment in terms of:
Social Political Economic Scientific Technological Historical Aesthetic Religious	*Physical characteristics:* climatic geological topographical *Other constraints:* land use existing built forms traffic patterns legal	*Available resources in terms of:* cash materials labour/equipment *Structural systems:* mass planar frame *Space separating system:* mass planar frame *Services system:* environmental information transportation *Fitting system:* furnishing equipment	*Structural mass:* visible surfaces space enclosed *Sensory environment:* lighting sound control heating/vent	*Organic:* hunger and thirst respiration elimination activity rest *Spatial:* functional (inc. fittings) territorial *Locational:* static dynamic *Sensory:* sight hearing heat and cold smell kinaesthetic equilibrium *Social:* privacy contact	Security Prestige Profit Expansion or other provision for change Housing of particular activities so as to encourage user well-being, motivation, etc.

(G. H. Broadbent, adapted from T. L. Markus: Building-Environment-Activity-Objectives model)

Fig. (2.3); Broadbent's Model.

Heath (1984) concludes that user needs may indicate the necessity of a detailed investigation which would lead to an accurate "fit". In this case, a list of activities would be prepared in consultation with the client, and through observation. As a result, a planning program is defined which documents environmental conditions, required physical space, structural effect, and relationship to other activities. During this stage, relationships to other activities are treated systematically with respect to the grouping and interconnection of activities resulting in a flow chart, or what Broadbent calls "strings of beads".

Site considerations generate a separate line of reasoning during design development. Environmental, legal, and physical characteristics of the given or selected site are then established. The considerations yield precise constraints. All these site characteristics and constraints are then represented in an "environmental matrix", which may be a literal physical model. Having completed both of these phases, the designer "backs the strings of beads into the environmental matrix, locating the most important beads in the environmentally best position for them". Trade-offs and compromises are necessary at this stage, as a result of internal conflict (Broadbent, 1980).

At this point, the building exists as a sort of diagram in space, then it is necessary to give it physical form. Broadbent proposes four pure types of design that allow for the transformation from diagram to physical form. These four types are pragmatic design, iconic design, analogical design, and canonic design.

Pragmatic design is the use of available materials and construction methods, and establishes a building form by trial and error. Iconic design is the literal repetition of a tried and accepted architectonic form, and allows for form

extrapolation as well as form combination. Analogical design is "the central mechanism of creativity"; it is the transfer of ideas from one context to another, or the displacement of concepts. Canonic design involves the use of a geometric grid, or proportional system.

Broadbent considers these four types of methods to be of uneven usefulness in present-day design. His approach to architectural design is evolutionary rather than revolutionary. Fundamental to his method is the definition of a problem space based on precedents.

In his book; "Method in Architecture", Heath (1984) states that:
"...Broadbent really considers only the constrained, well-behaved problem; it is significant that he chooses a small office building as his worked example. He sees very well that lack of constraint is a difficulty, ...and he also sees that this kind of sequence is by no means the only possible one. But he does not really provide any alternative."

In the "systems" building, the problem is just to establish the constraints, and to rank them. In the symbolic building, the problem is so lightly constrained that this kind of approach is not feasible.

It is really a method suited only for the "commodity" building, and for that it is, as Broadbent acknowledges in part at the beginning of his discussion, generally over-elaborate. It represents an important step forward in method, in that it summarizes first-generation methods and extends them by relating them to architectural history and the culture or subculture of architecture generally. But it is not complete", (Heath, 1984, pp. 140-141).

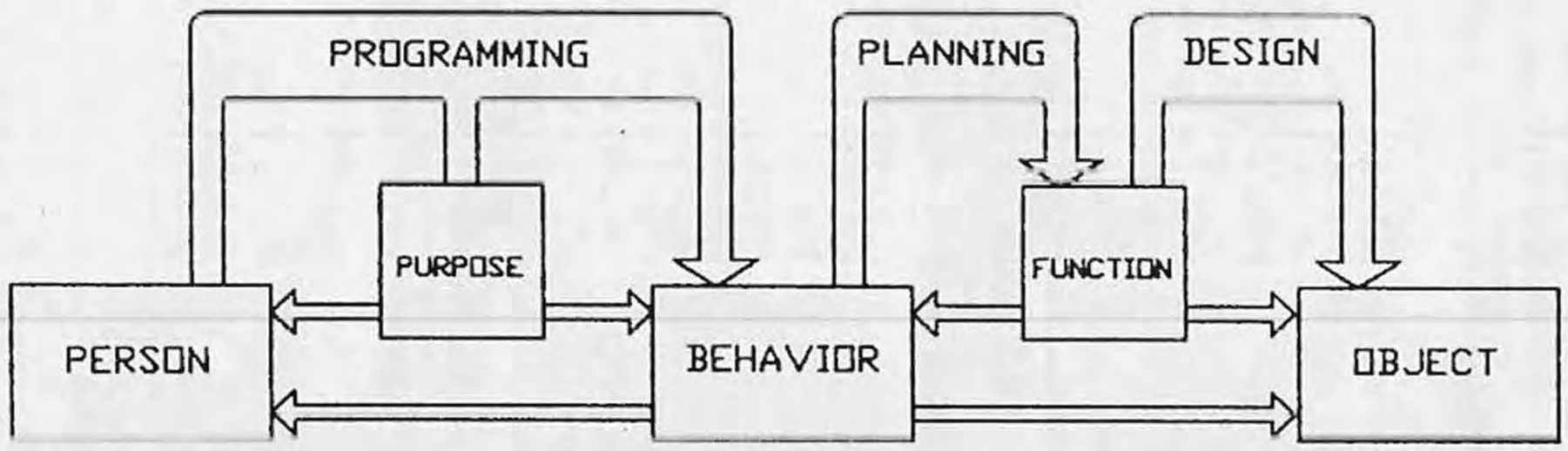
2.3.3. Wade's Model:

Wade's approach to method (1977), is more abstract and comprehensive, and has its basis in academia. It is based on a logical framework drawing on a wide variety of systems theory, psychological, and sociological sources. He identifies architecture as a problem-solving process with the following general problem form: A -> B. The initial state (A), the transformation process (->), and the final state (B) in this problem formulation can exist in: known, range, or unknown condition. In order to achieve a solution, at least two terms must be in the known state.

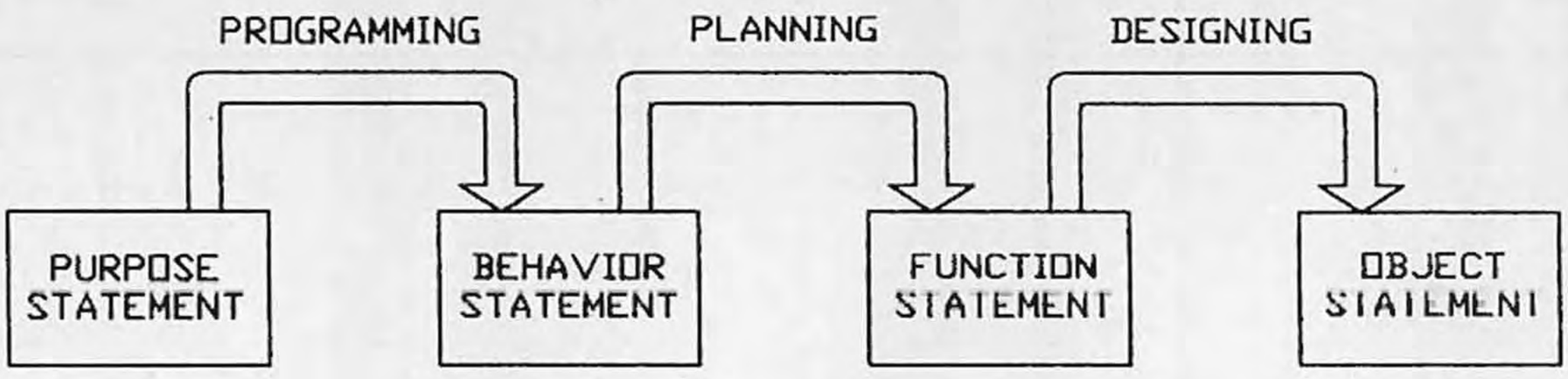
According to Wade (1977), if two or three terms of a certain problem have the condition: range and/or unknown, then the problem is ill-defined, and "closure" is required to bring at least two items into the known condition. Design problems characteristically require closure of the terminal state or goal. Goal closure is achieved by considering what is feasible to the designer and the client, based on the present state of the external environment. The designer proceeds to solve the problem by "imagining", which is presented in the form of drawings, trial solutions that allow critical discussion of each solution, and the re-formulation of goals in a "generate-and-test" solution environment. As goals may conflict, negotiations are allowed through an argumentative process (between the designer and the client), until closure is achieved. When the goal closure has been formulated, the transformation process is achieved by abduction.

The abduction process involves decomposition or breaking down of the two known terms in such a way that some fit can be discerned between the corresponding parts. The second phase of abduction is means-end-analysis (Wade, 1977), as will be explained later under problem finding in chapter 4, in this thesis. The scheme for the analysis is the purpose-behavior-

function-object spectrum. Each succeeding term is a means with respect to the preceding term, and an end with respect to the following term. "Programming converts purpose into behavior information; planning converts behavior into function information; design converts function into object information," as in fig. (2.4).



DESIGN PHASES AND THE INFORMATION SPECTRUM
ACCORDING TO WADE (1977).



STEPS IN THE DESIGN PROCESS
ACCORDING TO WADE (1977).

Fig. (2.4): Wade's Model.

Each move requires choice or decision, "since the demand statements in a means-end-analysis are typically at a higher level of abstractness than the proposed element to supply that demand, there is no way to ensure a determinate selection." The required choice or decision is then referred to "the problem authority (usually the client)," at each step. The process is then one of analysis-proposal-analysis-fit, culminating in the final phases of the abduction which are "specification, generation of proposals, and testing of proposals against specifications."

The testing criteria can be classified under headings summarized in the acronym: SOFA; symbolic, ordering, functional, and affective aspects. The final process in detailed design, or the fitting of function to object, is carried out either by selection or building up, or by some combination of the two.

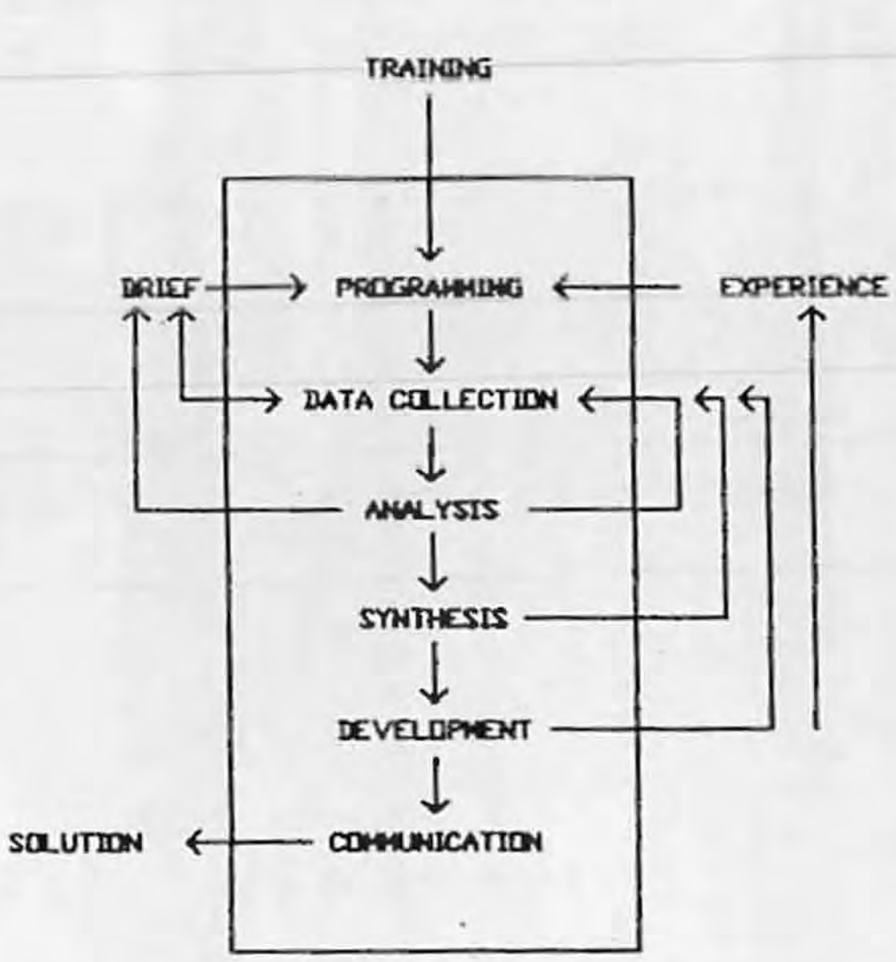
Wade does not clearly define the conflicts that can be generated by incompatible groups of clients and users, and pays little attention to the client/user distinction. "It is not clear how such conflicts are to be resolved, nor is it clear how 'identification' of client (and user?) with the final design is to be achieved. The implementation problem is not treated" (Heath, 1984, p. 145).

In his criticism of Wade's model, Heath (1984) writes: "a more serious criticism concerns the assumed primacy of the person-purpose-behavior-function-object sequence, and the corresponding layout-functional system-detailed design and programming-planning-design sequences. Wade discusses the mixture of information that may make up the "problem statement"; he makes it clear that such information can range over the person-object spectrum. He also points out...that goal closure may occur only gradually through the design process. But it follows that what are presented as sequences

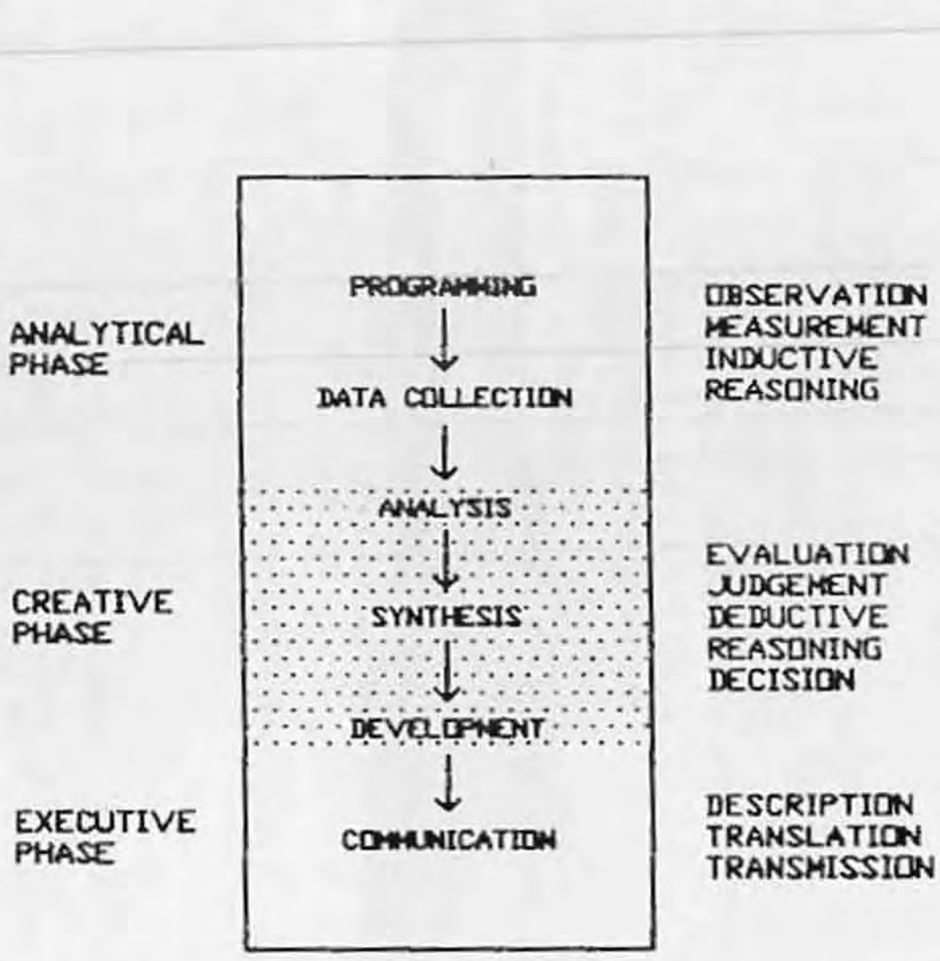
are not sequences, but structures, frameworks which ultimately have to be filled out, and a separate description of process is needed. The final design will be analyzable in terms of such sequences, but the actual order of decision, the design process will depend on the distribution of well- and ill-constrained elements throughout the structure. It will, therefore, be more like the process that Wade describes for detailed design, or 'build-up'. Heath (1984), p.144.

2.3.4. Archer's Model:

This model proposed in 1964, is considered to be one of the early models of the design process. In this model; fig. (2.5), the phase of communication was first introduced as an additional phase to the analysis-synthesis model.



A BREAKDOWN OF BASIC DESIGN PROCEDURE



THE MAIN PHASES OF DESIGN ACCORDING TO ARCHER'S MODEL

Fig. (2.5): The main phases of design according to Archer's model.

The first phase (analytical) consists of observation, measurement, and inductive reasoning, the second phase (creative) includes evaluation, judgement, deductive reasoning and decision making, as in fig. (2.5). Once the crucial decisions are made, the design process continues with the execution of working drawings, schedules, etc., again in an objective and descriptive mood, i.e., the last phase (executive) consists of description, translation and transmission. The most essential characteristic of Archer's model is the continuous feedback between these phases.

Archer, like Alexander, views the design process clearly as a step-by-step process. The steps in Archer's analytical phase consist of the definition of objectives, the definition of factors affecting the design, and the relationships among them, as well as the formulation of subproblems.

The result of this phase is similar to the result of Alexander's analytical phase. In Alexander's model, it is called the "program", while in Archer's model , it is called the "definition of the problem". Archer (1970) states that "the design problem is expressed as a rank ordered list of the attributes which the final solution is required to have."

The difference between Archer's and Alexander's models is made evident in the phase of synthesis. Archer thinks that form depends on the designer's values, and the incidence of original ideas during the design process. He assigns the primary importance in the phase of synthesis to the search for solutions to the problem.

Design ideas, plans and solutions to problems in architecture are usually carried out not by their designer, but by someone else. Archer recognizes the importance of the communication between the two parties; he includes this

communication as an essential part of the design process, and makes it the central issue of the executive phase.

In further elaboration of his "logical model of the design process", Archer (1970) resorts to the extensive use of mathematical notation, and the terminology of operations research in an attempt to lay the groundwork for a scientific approach for design, or a science of design that is compatible with operations research and management science. He did that in the hope of discovering some general laws of design. Design, according to Archer, is not limited to any specific discipline, he is striving for a general theory which would be applicable to any field or discipline.

2.3.5. Space Allocation Model (Tabor's Model):

In his paper "Analyzing Communication Patterns", Tabor (1976) characterizes two types of approaches for space allocation as "permutational" and "additive", and presents space allocation as a design method.

In the permutational method, initially a complete layout of the plan is produced either at random, or arbitrarily. Then the positions of activities, rooms or dimensional units, are rearranged in such a way as to progressively reduce the alternatives for a solution that satisfies a predefined set of criteria within the initial layout. It is in this sense that the method consists of a process of "improvement" of the starting layout.

On the other hand, in the additive method, the layout of the plan is achieved by starting with an empty site or an empty "floor plan", and adding each room or dimensional unit sequentially. An optimum layout based on a set of constraints can then be built up. The criteria upon which each successive unit is positioned are dependent on the measure of the

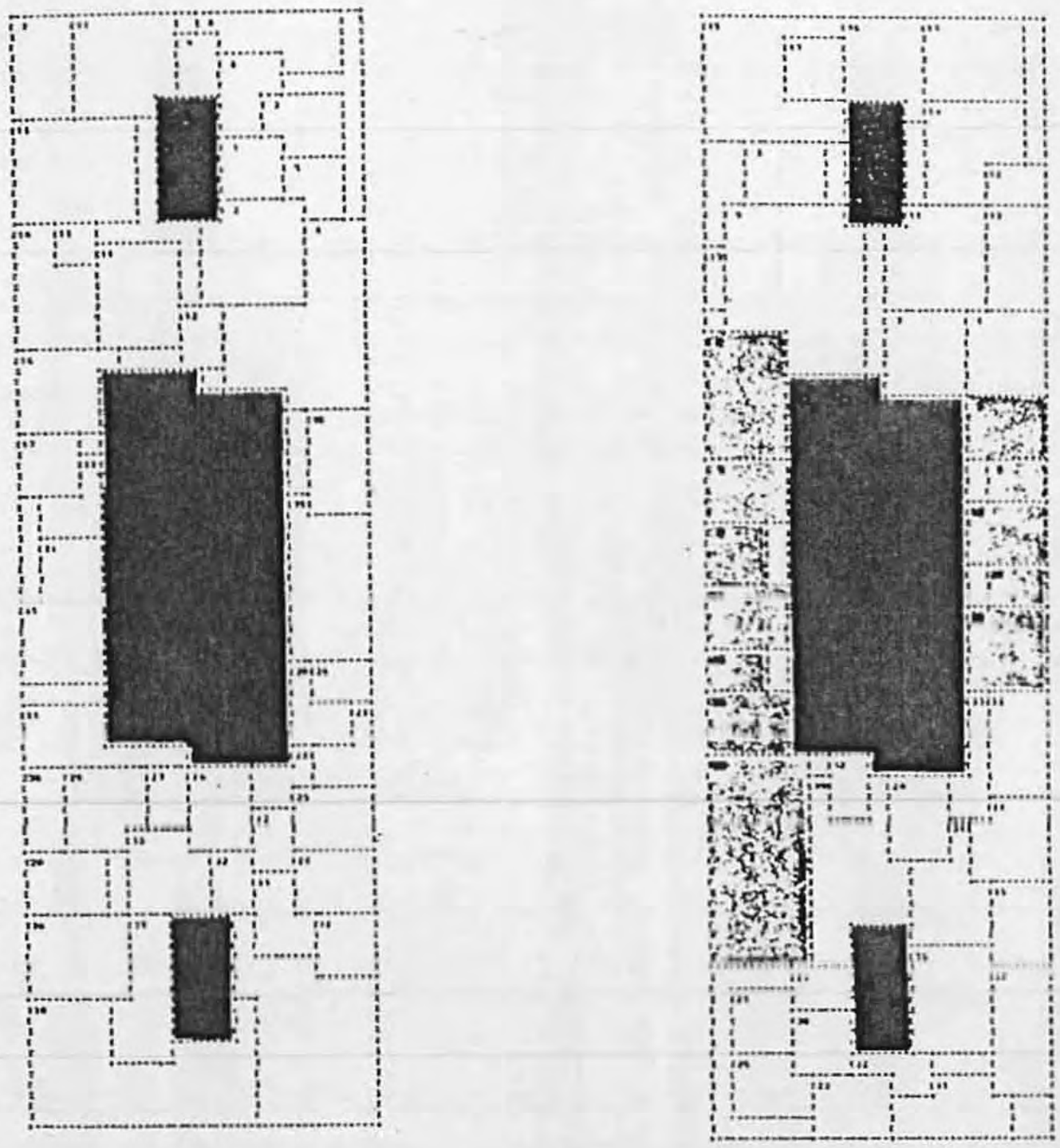
association of that unit with all other units already placed within the empty site. This differs from the permutational method which starts with a non-valid layout and proceeds in stages of rearrangement.

With the permutational technique, it is possible to arrive at an unmanageable set of solutions. The number of ways of arranging one activity in one location is factorial N (!N). In a layout with 10 rooms, the number of possible solutions is over 3.5 million. Even in cases where layout symmetry can be incorporated, the number of possible solutions can be overwhelming. If pairing is incorporated, the "optimum" layout that can be achieved becomes dependent on the original, arbitrarily selected plan with pairing indicating the rearrangement of pairs of units at each stage.

Every "additive" procedure has two essential requirements. The first is a kind of spatial framework within which the plan is assembled. The second requirement is the need for a criterion for the placement of the activities within the plan. In general, although the additive approach results in a smaller number of possible solutions, in its simplest application it often results in irregular and ragged layouts.

Different methods of application for either approach exist. The literature indicates programs for space allocation that incorporate, generate, and test procedures, heuristic search techniques, linear and non-linear programming, and analytical procedures.

The DOMINO developed by Mitchell and Dillon (1972) incorporates an additive approach with a heuristic search technique for explicit representation of spatial domains. The program was used to derive a solution of an office building floor plan (Mitchell, 1977).



ELEMENT NAME	NUMBER	AREA	ADJACENCY PREFERENCES							
410000	1	551	0	0	0	0	2 ADJ	5 ADJ	4	3 ADJ
DISTANCE BETWEEN DEPARTMENTS (FEET)			0	0	0	0	32	25	64	26
411000	2	1353	0	0	5 ADJ	0	4 ADJ	1 ADJ	0	0
DISTANCE BETWEEN DEPARTMENTS (FEET)			0	0	35	0	35	32	0	0
SU. CORE W C	54	1152	0	0	0	0	0	0	0	0
DISTANCE BETWEEN DEPARTMENTS (FEET)			0	0	0	0	0	0	0	0
REQUIRED			20	8	20	9	22	13	8	6
SATISFIED			17	7	14	5	10	3	4	2

Fig. (2.6): Layout generation using DOMINO.

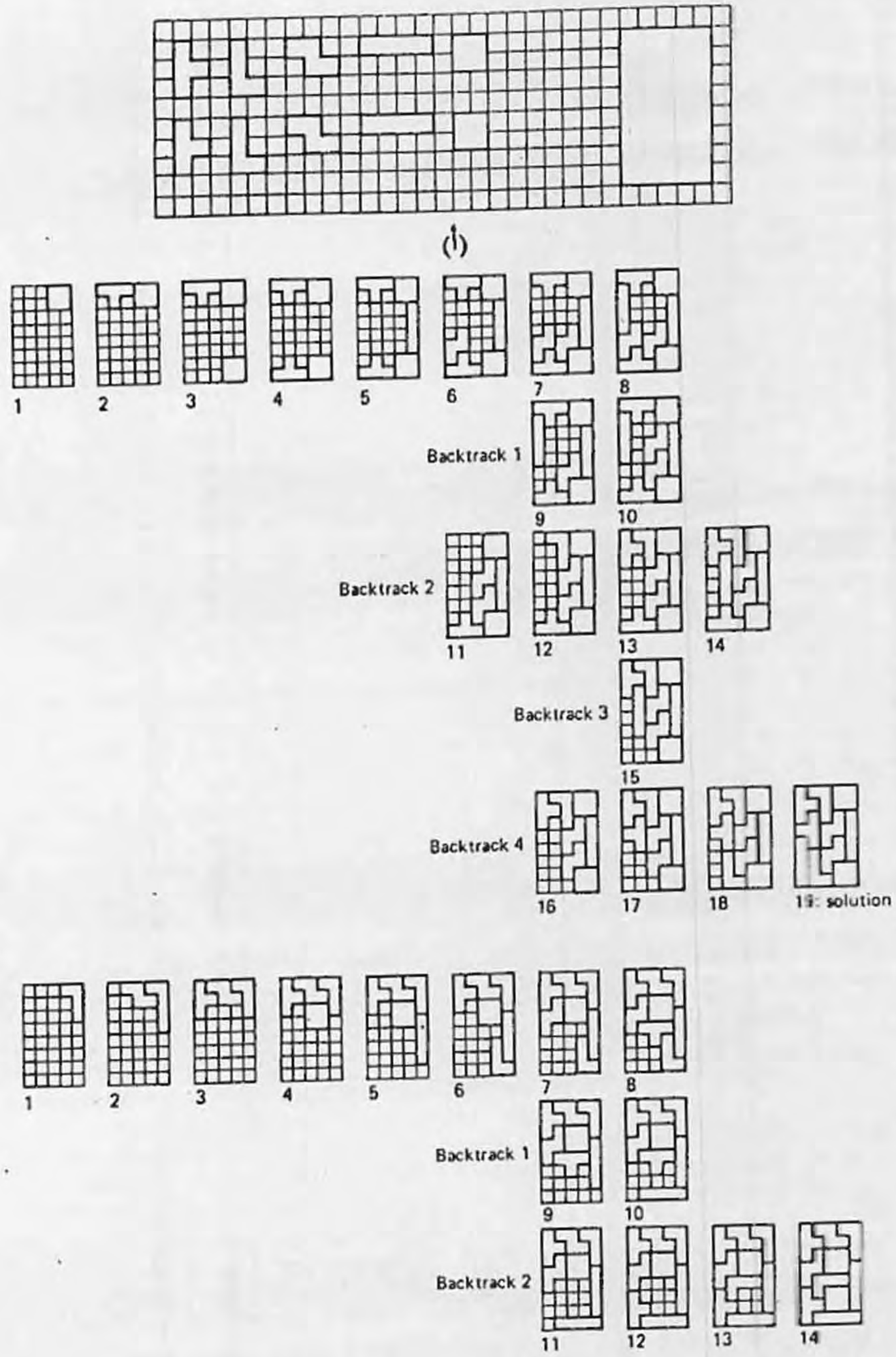
Figure (2.6) illustrates a worked example of layout generation using DOMINO. Selection constraints were defined as a set of heuristic rules and the floor plan was defined as a rectilinear grid with a minimum design unit size. The heuristic rules defined the adjacency parameters for the units, space requirements and dimensions. The generated layout incorporated all assigned spaces in an irregular form, or when the form was made rectilinear, unassigned spaces remained.

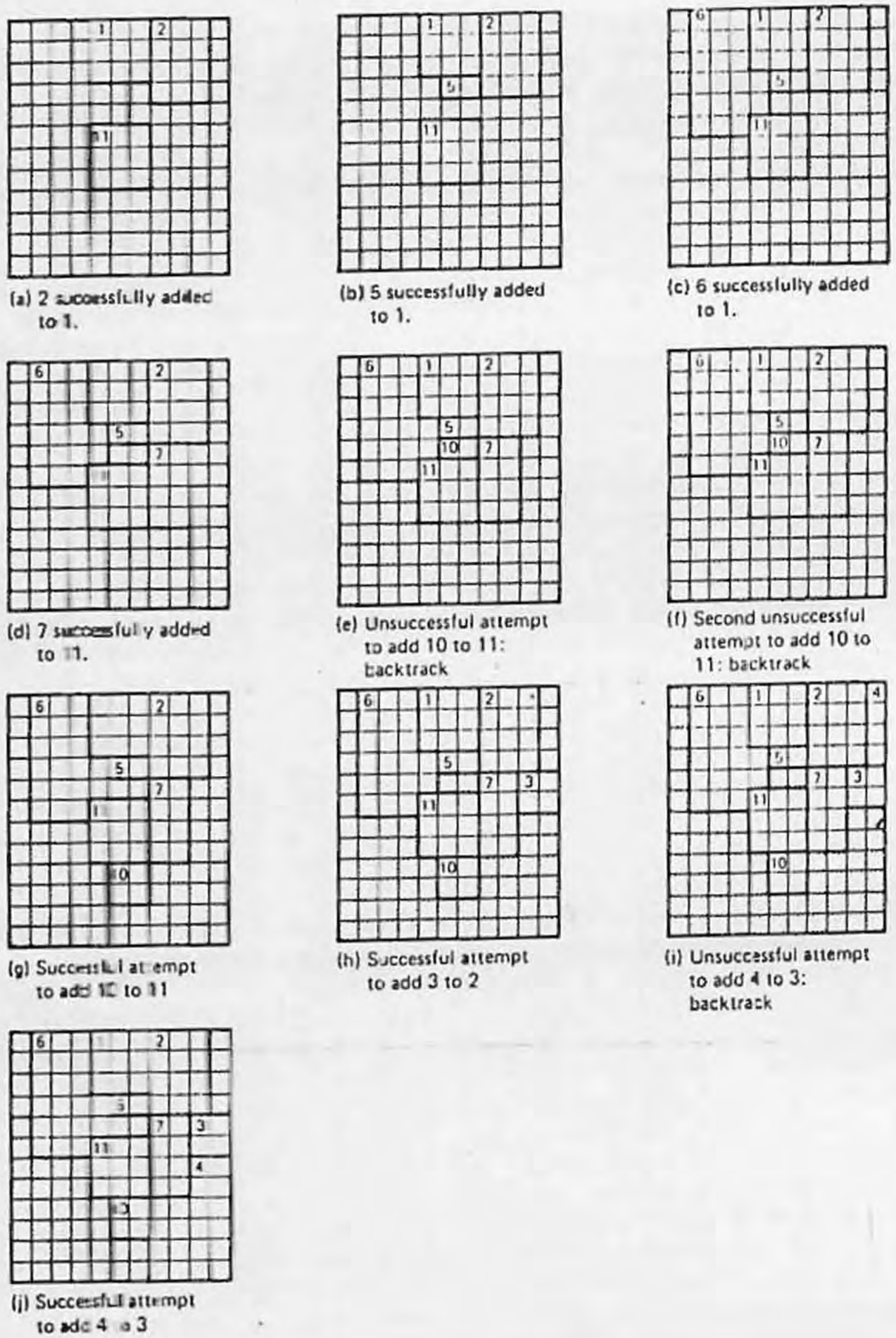
Although space allocation is considered to be an architectural design subtask method rather than an architectural design method, still it can assist the designer in space layout and planning. Major considerations are the limitations inherent in this method. The "optimal" solution is generated based on a bold conjecture, the assumption of an initial layout, floor plan and /or grid, and the adjacency constraints and considerations. Qualitative constraints, such as space perception, might have to be defined intuitively. Additionally, the initial layout and/or floor plan based on an intuitive decision by the designer.

Figure (2.7) illustrates the backtracking process by the DOMINO floor-plan layout program. The program attempts to add new spaces to the perimeters of located spaces to which there are high interactions. Backtracking is necessary if there is insufficient room for the added space at the chosen perimeter location. A new perimeter location is then selected and tried.

2.3.6. Jones's Model:

According to Jones, the method is primarily a means of resolving a conflict that exists between logical analysis, and creative thought. The difficulty is that the imagination does not work well unless it is free to alternate between all





(a) 2 successfully added to 1.

(b) 5 successfully added to 1.

(c) 6 successfully added to 1.

(d) 7 successfully added to 11.

(e) Unsuccessful attempt to add 10 to 11: backtrack

(f) Second unsuccessful attempt to add 10 to 11: backtrack

(g) Successful attempt to add 10 to 11

(h) Successful attempt to add 3 to 2

(i) Unsuccessful attempt to add 4 to 3: backtrack

(j) Successful attempt to add 4 to 3

Fig. (2.7): Backtracking by the DOMINO floor plan layout program.

aspects of the problem, in any order, and at any time. On the other hand, logical analysis breaks down if there is the least departure from a systematic step-by-step sequence (Jones, 1963). It follows that any design method must permit both kinds of thought to proceed together if any progress is to be made. Accordingly, Jones's method could be summarized into two main points:

1. Leaving the mind free to produce ideas, solutions, hunches, and guesswork, at any time without being inhibited by practical limitations, and without confusing the process of analysis.

2. Providing a system of notation which records <u>every</u> item of design information outside the memory, keeps design requirements and solutions completely separate from each other, and provides a systematic means of relating solutions to requirements with the least possible compromise. This means that while the mind moves from problem analysis to solution-seeking whenever it feels the need, the recording develops in three distinct stages of: <u>analysis, synthesis, and evaluation</u>.

   i. Analysis: The listing of all design requirements, and the reduction of these to a complete set of logically related performance specifications.
   ii. Synthesis: Finding possible solutions for each individual performance specification, and building up complete designs from these, with least possible compromise.
   iii. Evaluating the accuracy with which alternative designs fulfill performance requirements for operation, manufacture and sales <u>before</u> the final design is selected.

Later on, in 1969, Jones describes the three phases in a slightly different way, as <u>divergence, transformation, and convergence</u>.

i. Divergence is concerned with breaking the problem into pieces, and defining the boundaries of a space in which a fruitful search for a solution can take place.
ii. Transformation involves putting the pieces together in a new way.
iii. Convergence is testing to discover the consequences of putting the new arrangement into practice.

The design method that Jones presented, places importance on the generation and selection of alternative solutions during the design process. Central to this method, is the interaction between the client and designer during which the client presents tasks to the designer, who in turn responds by continuously building models of how these tasks can be completed. The central issue is the "prediction of performance." Selection of alternatives is based on the ability to deal with a number of small, discrete models all at one time and to derive a satisfactory prediction of the fitness of the tasks.

Jones proposes a form of decomposition of the problem into a number of different tasks, and then deals with each task independently, a concept largely attributed to the early work of Alexander and his decomposition theory (Coyne et al., 1990). Jones recognizes that the type and form of a particular model influence the solution of the problem. Although he is concerned with the availability of a variety of formal techniques of analysis, synthesis, and evaluation, he does not provide a means of identifying, understanding, and selecting them for the designer.

The task of cycling through the three phases of analysis, synthesis, and evaluation constitutes a kind of search procedure, these search techniques will be covered in detail in chapter three in this thesis. Implicit in the idea of search are some important notions: the existence of goals so that the search may have some direction; the existence of states or points along a search path that represent the position of the design in its development; and the existence of mechanisms for moving from one state to another (Coyne et al, 1990).

2.3.7. Other Models:

There have been other contributions to the early design methodologies. One of them, is Luckman's (1969) AIDA methodology for the decomposition of architectural problems based on the identification of "decision areas." AIDA which stands for the Analysis of Interconnected Decision Areas, is better suited for the solution of puzzles, and not applicable to architectural design problems (Kalisperis, 1988). Such a method provides quick and reliable solutions to puzzles which previously had been solved by trial and error methods or guesswork.

Luckman's model of the design process is based on the three stage process of analysis-synthesis-evaluation. However, his view is not that this is a simple, complete linear process, but that it recurs at different levels of design detail; the designer is continually cycling through analysis-synthesis-evaluation, proceeding from the more general problem levels, to the more specific.

Asimow (1962) presented his model of the design process, as encompassing the three phases of analysis, synthesis, and evaluation/decision, as shown in figure (2.8). Decision was further subdivided into optimization, revision and

implementation. Asimow's model was based on the sequential characteristic phases of engineering design, and as such not applicable to architectural design problems.

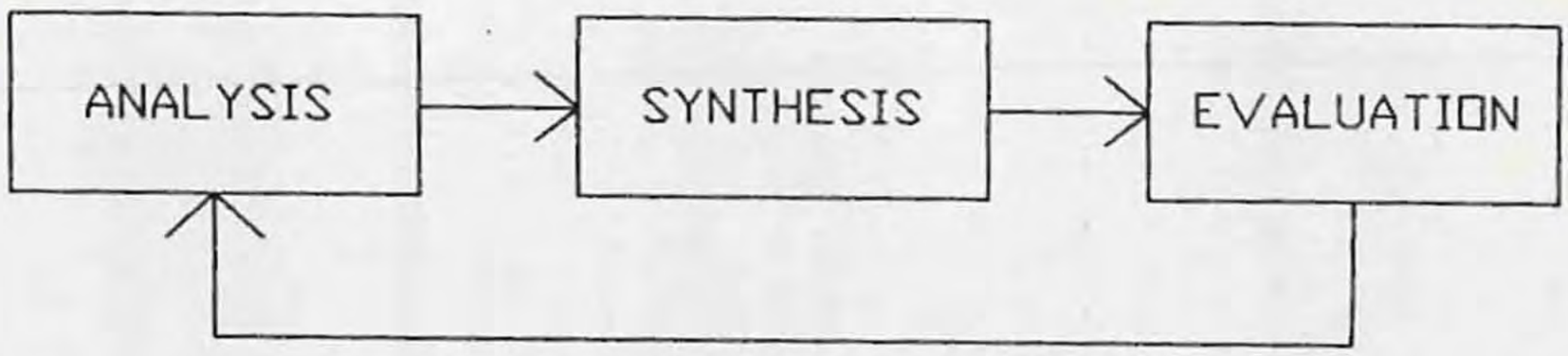


Fig. (2.8): The three-stepped process of design.

According to Asimow (1962).

2.4. Criticism of First Generation Design Methods:

First generation design methods were based on the assumption that the initial state, and goal state of design problems were known, or could be easily discovered. The primary emphasis was the development of better methods of organizing the transformation from the initial state to the goal state. The objective of these methods was in fact, to find an "algorithm," a logically rigorous set of rules for producing a satisfactory, or even optimum result (Kalisperis, 1988). This systems approach, as mentioned earlier, did not suit entirely the case of architectural design.

First generation methods were best suited to the solution of "well-structured", "well-behaved", or "well-constrained" problems. They were based on industrial design rather than architectural design. Although industrial design is concerned with objects that are only as complex as the physical systems

present in buildings, the role of these objects in systems of human activity is customarily much more closely defined. Initial states, as well as goal states, can be stated with "some precision", and the boundary between the physical systems and the activity systems are clear.

Industrial design products become obsolete within a relatively short period of time as a result of the design problem being defined within a rigid or nonadaptive context (Kalisperis,1988). On the other hand, architectural design artifacts are always perceived in their totality and are never fragmented.

First generation methods are concerned with objects whose role in the systems of human activity, is closely and clearly defined. The inputs and outputs required for the problem solving process can be stated with precision. Industrial design operates within a market that provides constant feedback and further precision to the definition of required inputs and outputs. The process of evaluating buildings is slow and laborious, providing delayed feedback for the redefinition of the architectural design problem.

First generation design methods were almost immediately criticized. In fact, some of the authors of those models (notably Alexander), soon modified there positions and pointed at the difficulties encountered in attempting to use the prescribed procedures (Alexander, 1965). The criticisms of the early design models or the first generation methods could be summarized as follows: a) design is not a strictly sequential process, b) design problems are "wicked" problems, and c) a linear, sequential procedure applied to architectural design problems cannot by itself yield any solutions.

2.5. Second Generation Design Methods:

In a discussion that appeared in the Fifth Anniversary Report of the Design Methods Group (Rittel, 1972), a new direction for design methods was introduced. These "second generation" design methods resulted from the difficulties experienced in the application of the "first generation" design methods (Grabow, 1983).

It was Rittel's definition (1974) of "wicked problems" which indicated the fundamental problem with the first generation methods (Rittel, 1971). In his article: "Some principles for the design of an educational system for design", which was published in the Journal of Architectural Education (1971), Rittel describes ill-structured problems; "...a class of social system problems which are ill-formulated, where the information is confusing, where there are many clients and decision makers with conflicting values, and where the ramifications in the whole system are thoroughly confusing".

Later, in a joint paper with Webber, Rittel states that: architectural problems, as well as "planning problems (societal problems), are inherently different from the problems that scientists and perhaps some classes of engineers deal with". Architectural and "planning problems are wicked problems" (Rittel, and Webber, 1973). As distinguished from problems in the natural sciences, which are definable, separable, and may have solutions that are findable, architectural problems are ill-defined, and they rely upon elusive personal judgement for resolution and not solution. "Social problems are never solved. At best they are only re-solved, over and over again" (Rittel, and Weber, 1973).

The problems that scientists and engineers have usually focused upon are mostly "tame" or "benign" ones. As an

example, consider a problem of mathematics, such as solving an equation. The mission is clear. In turn, it is clear whether or not the problem have been solved.

Rittel, and Webber then describe ten different properties of "wicked problems", which in turn apply to planning and architectural design problems:

1. Wicked problems have no definitive formulation. Any time a formulation is made, additional questions can be asked, and more information can be requested. The information needed to <u>understand</u> the problem depends upon one's idea for <u>solving</u> it. That is to say, in order to describe a wicked problem in sufficient detail, one has to develop an exhaustive inventory of all conceivable <u>solutions</u> ahead of time. The reason is that every question asking for additional information depends upon the understanding of the problem- and its resolution- at that time. Problem understanding, and problem resolution are concomitant to each other. Therefore, in order to anticipate all questions (to anticipate all information required for resolution ahead of time), knowledge of all conceivable solutions is required. In other words, whenever a wicked problem is formulated there already must be a solution in mind. For example, if the lack of privacy is defined as a deficiency of an architectural design, the solution has already been stated, and that is to provide the needed privacy.

2. Wicked problems have no stopping rule. Anytime a solution is formulated, it can be improved upon or developed further. The problem solver or the designer stops developing his solution not for reasons inherent in the 'logic' of the problem, but for considerations that are external to the problem: he has run out of time, or money, or patience, or the fee is exhausted, or the

building is finally built, or some other resource is exhausted. He finally says that this is the best solution within the limitations of the project, or that he likes this solution.

3. Solutions to wicked problems cannot be true or false, they can only be good or bad. For a building, there are no true or false solution, only a good or a bad solution. For different people judging a building's solution, their judgments are likely to differ widely to accord with their group or personal interests, their special value-sets, and their ideological predilections.

4. There is no immediate and no ultimate test of a solution to a wicked problem. For tame problems, one can determine on the spot how good a solution-attempt has been. With wicked problems, like architectural problems, any solution, after being implemented, will generate waves of consequences over the life cycle of the building. In other words, any time a successful solution is achieved, it is still possible for that same solution to fail in some other respect. An example would be: if visual relations to the site are of interest, then large glazed windows would be used, but in the same time, large glazed windows would cause another problem for privacy, or heating for example. In wicked problems, the full consequences cannot be appraised until the "waves of percussions" have completely run out, and there is no way tracing <u>all</u> the waves through <u>all</u> the affected lives ahead of time or within a limited time span.

6. Every solution to a wicked problem is a "one-shot operation. There is no opportunity to learn by trial-and-error, there is no possibility for experimentation, and every solution or attempt counts significantly. Once a building is designed and built, there is no way of

going back to redesign. Many people's lives will have been irreversibly influenced, and large amounts of money will have been spent. Whenever actions are effectively irreversible, every trial counts, and every attempt to reverse a decision or to correct for the undesired consequences poses another set of wicked problems, which are in turn are subject to the same dilemmas.

6. Every wicked problem can be considered to be a symptom of another higher level problem. As an example, if the maintenance of a residence is too expensive for its inhabitants, this indicates that there is a problem with the income of the inhabitants.

7. For every wicked problem there is always more than one possible explanation. The selection of the explanation depends on the employed perception. People choose those explanations which are most plausible to them. They pick the explanation of a discrepancy which best fits their intentions, and which conforms to the action-prospects that are available to them. The explanation also determines the solution to the problem. For example, the high cost of a building may be attributed to high materials' cost. In order to decrease the building's cost, it is necessary to choose other materials with less cost.

8. Wicked problems do not have an exhaustive list of potential solutions, nor a set of permissible operations. There are no criteria that enable one to prove that all solutions to a wicked problem have been identified and considered. In architectural design, a host of potential solutions arises, but another host is never thought up. It is then a matter of realistic judgment whether one should try to enlarge the available set or not, and it is a matter of judgment as well, which of these solutions

should be pursued and implemented. In such fields of ill-defined problems, the set of feasible plans of action or solutions that will lead to a conclusion do not only rely on judgment, but also on the capability and the experience of the architect to appraise or evaluate "exotic" ideas, as well as the amount of trust and credibility between both; the architect and his client.

9. Every wicked problem is essentially unique, it is a one-of-a-kind. No two problems are exactly alike, and no solutions or strategies leading to solutions can readily be copied for the next problem. Even if two residences are designed for the same family, under the same geographical conditions, they will never be identical.

10. The wicked problem solver, or the architect, has no right to be wrong, he is fully responsible for any actions. The architect's aim, in contrast to a scientist's aim for example, is not to find the truth, but to improve some characteristics of the world where people live. Architects "are liable for the consequences of the actions they generate, the effect can matter a great deal to those people that are touched by those actions" (Rittel, and Webber, 1973).

First generation methods of the design process, do not accommodate wicked problems. In such methods, the first step is to understand the problem, the problem and the solution are treated separately. On the other hand, when dealing with wicked problems, it is not possible to separate the solution from the problem. Solutions are generated as problems are further defined, and problems are defined better as further solutions are generated. The effort to define permissible operations is in direct contradiction to the properties of wicked problems.

In 1972, Rittel established that wicked problems cannot be subject first to rigorous analysis and then synthesis. According to Rittel; "design means thinking before acting". He argues that the design solution is a "plan for accomplishment of the solution, not the solution itself". His model of design defines three phases: the context, the object model, and the overall performance. The role of the designer is one of builder and operator of the object model; he manipulates parameters and assesses the measured performance. The designer also defines the alternating phases of the design process, and continuously goes through alternating sequences of generation and reduction. During generation, the designer searches for possibilities, while during reduction, possibilities are discarded based on their evaluation. The design process according to Rittel, is considered to be a problem solving, in which "cycles are not at all linear", but instead, they are constituted of networks.

Rittel's most important contribution is the outline of decision making in design problems. He describes how decisions are made: "...the designer is arguing toward a solution with himself, and with other parties involved in the project. He builds a case leading to a better understanding to what is to be accomplished. In its course, solution principles are developed, evaluated in view of their expected performance and decided upon. The parties commit themselves to specific courses of actions, and to the risks involved in them. In this way better formulations of the problem are being developed simultaneously with a clearer and clearer image of the solution. (Rittel, 1971, p.19).

2.5.1. The IBIS Model:

The IBIS method, which stands for "Issue Based Information System", was developed by Horst Rittel (Rittel, and Kunz, 1970), and is based on the principle that the design

process for wicked problems, is fundamentally a conversation among the stakeholders (e.g. designers, customers, implementers, etc.), in which they bring their respective expertise and viewpoints to the resolution of design issues. Any problem, concern, or question can be an issue, and may require discussion (if not agreement) in order for the design to proceed. In the IBIS model, it is this "argumentation" that constitutes the design process.

In the IBIS model, the chief element is the **Issue**, which states a question or problem. An Issue is responded to by **Positions**, which state possible resolutions of the Issue, and a Position can be either objected to, or supported by various **Arguments**. Connecting these three node types, there are different kinds of links. For example, a Position **Responds-to** an Issue, and this is the only place the Responds-to link can be used. Arguments must be linked to their Positions with either **Supports** or **Objects-to** links. Issues may **Generalize** or **Specialize** other Issues, and may also **Question** , **Be-suggested-by**, or **Replaces** other Issues, Positions, and Arguments. Fig. (2.9) shows a state transition diagram specifying all of the legal moves within the IBIS method.

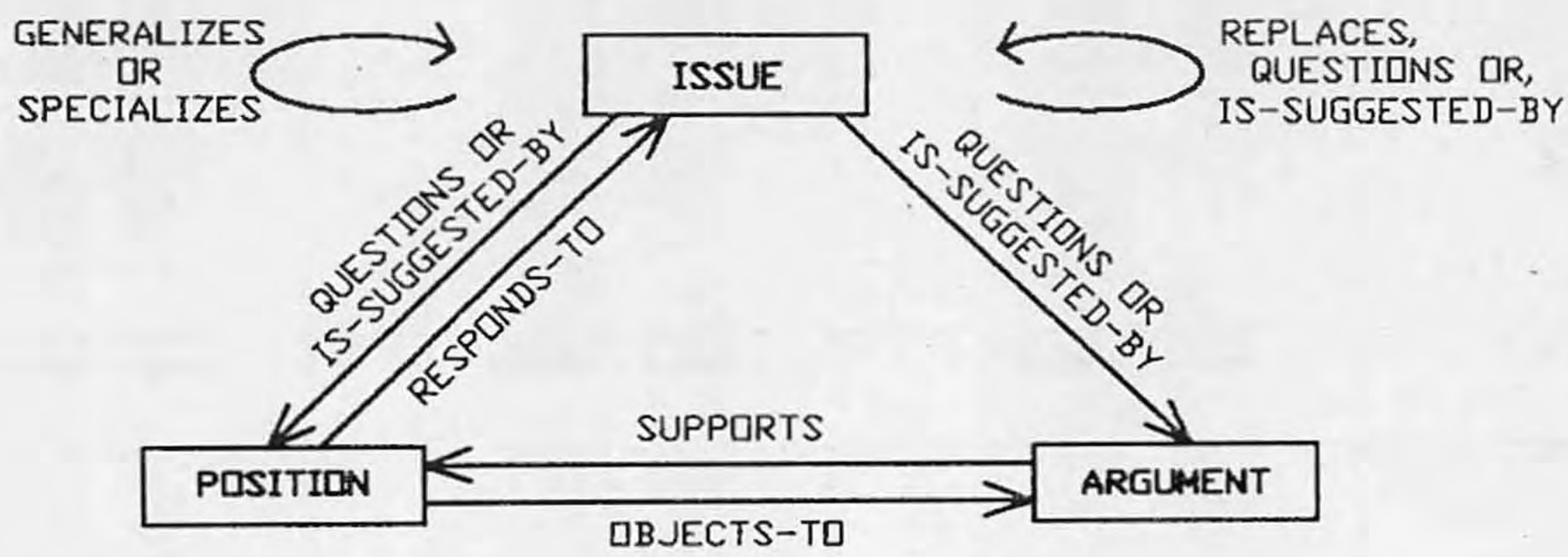


Fig. (2.9): A diagram for the IBIS method.

This process of argumentation or deliberation, does not necessarily require a group of people. In fact, every designer carries out this process while making his decisions. The designer considers issues, develops positions for these issues as well as against them, argues the merits of each internally, and then makes a decision.

An example of the argumentative nature of the design process can be illustrated through the design of a family residence. The designer summons the client with the members of the family and other parties who have a vital interest in the project, such as the structural and mechanical engineers, the banker, and so on. A list of significant issues is then presented to the members of the group. The designer then helps to define different positions along with the supporting, and counter arguments for each.

A typical IBIS discussion begins with someone posting an Issue node containing a question such as "How should we do X?". That person may also post a Position node proposing one way to do X, and may also post some Argument nodes which support that Position. Another user may post a competing Position responding to the Issue, and may support that with his own Arguments. Others may post other Positions, or Arguments which support or object to any of the Positions. In addition, new Issues which are raised by the discussion may be posted and linked into the nodes which most directly suggested them.

In the IBIS method, as other methods dealing with wicked problems, there is no stopping rule, nor is there a particular way of registering that an Issue has been resolved by agreement upon some Position. Rather, the goal of the discussion is for each one of the stakeholders to try to understand the specific elements of each other's proposals, and perhaps to persuade others of his own point of view.

2.5.2. The Pattern Language:

As a result of the implementation of his hierarchical decomposition method, Alexander in 1977 defined two problems in architectural design. First, that the actual substance out of which the environment is made consists of relations, or patterns rather than things; and second, that it is actually generated by the implicit, language-like system of rules which determines their structure. This work resulted in the publication of A Pattern Language (Alexander et al. 1977), which in addition to the other three volumes, The Timeless Way of Building (Alexander, 1978), The Oregon Experiment (Alexander, et al. 1975), and The Linz Cafe (Alexander, 1981), defined his "pattern Language" method or doctrine, as it has been called by others (Heath, 1984).

In the pattern language, the notion of ultimates is retained but the ultimates are no longer misfits. Instead they are systems of interaction between people and environments. Behavior is seen as forming a field of forces or tendencies-to-action, and the environment may interfere with, or facilitate these tendencies. A "good environment" is then taken to be one in which no two tendencies conflict, because they are not "allowed" to by the design. Certain systems of behavior and certain physical settings are in fact prescribed as ideal, or ultimate as "patterns."

In "A Pattern Language", the prescriptive intention is clear. Some of the patterns are "true invariants," they are properties to "all possible ways" of solving the stated problem. Such claims are very strong, they go far beyond the traditional use of pattern books in architecture as guides and devices for saving design time. Inevitably, they have been challenged (Heath, 1984, p. 135). In real life, there are so many conflicts, among the aims and interests for different groups as well as for any one person. The process of design

is largely one of moving sufficiently from conflict towards consensus to permit action; thus it could be treated as negotiation (Cuff, 1982). The pattern language simply legislates conflict out of existence.

Similar to the hierarchical decomposition, the pattern language assumes that there are no possible conflicts between "patterns". Protzen (1980) has drawn attention to the mandatory and arbitrary nature of patterns, and has argued that they are not empirically testable, and may be obstructive to inquiring design. Alexander's new model describes the design process essentially as the process of acquiring knowledge and then making decisions which reflect that knowledge. The "specialist" designs the patterns as a pictorial diagram describing one's own knowledge and perception of the task. The "object designer", defined as the client/owner in most cases, evaluates the pattern and either rejects it or accepts it.

An application of the pattern language is outlined in the book The Production of Houses (Alexander, 1985). When designing houses for ten different families in the Mexicali project, Alexander and the individual owners designed and built the houses based on 21 patterns. The architect/builder team generated individual houses based on a general type dependent on the 21 patterns. Alexander emphasizes this process in which the artificial schemes of architect and builder do not exist.

2.5.3. Squatters Technique:

According to Caudill (1971), the purpose of squatters technique "is to analyze the requirements, derive a program, and develop a schematic solution with the clients". The squatters technique was introduced to solve a communication problem. The office of William Caudill introduced the

technique into its practice after a series of observations that were made in the design process of a number of school buildings (Caudill, 1971). The fundamental intent of the technique, was to address the need for client approval of preliminary design drawings.

During the initial stages of design, the design team will derive a schematic solution through arguing and "squatting". Since the client is involved in the design process, approval of the preliminary drawings is automatic. Although this technique is not structured, does not exhibit any methodology, and does not indicate ways to resolve the conflict, Caudill's interest was in deriving a solution by taking advantage of the internal conflict present during squatting. The technique is very personal, and the outcome of the "squatter meeting" depends solely on the "squatter leader."

In general, the Squatters technique cannot be presented as a design method, but rather as a technique of involving the client/user in the design process at a very personal level.

2.5.4. Design as a Learning Process:

Bazjanac (1974) presented a model of the design process based on integration and learning. According to his view of the design process, the designer formulates the problem, and then proceeds with the search for the definition of the solution. The formulation of the problem is never final, and is dependent on the designer's understanding for the problem space at an instant of time. Definition of the problem is an evolutionary activity during which the designer increases his knowledge of the problem and its solution. "He gains new insights into the problem (and solution) which ultimately result in the formulation of a new view - the problem and the solution are defined" (Bazjanac, 1974, p. 15).

The process is terminated when one of the following conditions has been satisfied: a) the incremental gain in knowledge has become insignificant and the understanding of the problem as well as the solution cannot change enough to warrant further re-definition (i.e., the designer has reached his limits of understanding); b) the incremental gain in knowledge has become too costly; and c) the available resources (primarily time) have been exhausted.

2.5.5. Function-Analysis Techniques:

In the year 1988, Kirk and Spreckelmeyer defined a framework for decision making in design which had its roots in a number of design philosophies and associated professions, such as science, engineering, and business. Their model of decision making is based on the scientific method. The orderly progression proceeds from problem definition, to generation of alternatives that address the problem. Alternatives are tested against well-defined criteria, selecting the alternative that best solves the problem.

Their model is somewhat similar to William Pena's (1977) five-part process for architectural programming. Kirk and Spreckelmeyer argue that Pena's process has become a standard guide to defining architectural problems, and his use of the terms "goals", "facts", "concepts", "needs", and "problem statement" parallels the scientific method. Kirk and Spreckelmeyer's model of design, accepts Pena's approach as a predesign activity, one that should not be confused with the creative act of design itself. Their model is defined as a two-dimensional approach to problem solving that describes the methodology for decision making, and specifies its applications to architectural design problems.

The methodology involves three decision-making activities: a problem exists within a specific context of abstract ideas and human values, information, and economic, social, and cultural norms. A process of rational decision making is applied within this context to arrive at an understanding of the nature of the problem. A product or set of instructions to solve the problem is generated by this process in the form of strategies, plans, specifications, or buildings.

This methodology is used at each point in the architectural design cycle, as well as a repetitive sequence of decision making events. The design decision cycles proceed with the process of rational decision making within which four phases are identified, and these are: information analysis, speculation, evaluation, and synthesis.

2.5.5.1. Information Analysis: The first task for the design team, is to search out information concerning various building systems. The team needs to learn how the systems will go together, and which components will be necessary in order to construct the completed environment. The design team searches out the data from similar building projects, and pursues manufacturers' literature. The team eliminates extraneous information and concentrates only on the pertinent data. Their major task is to rank the value of available information.

A technique useful during this information phase is function analysis, focusing attention on the purposes of a particular building element or building activity. Function analysis takes a large building element, and divides it into discrete component parts. At the end of the analysis, the design team has a better idea of how each building system fits into a logical and hierarchical framework for the building as a whole.

2.5.5.2. Speculation: During this phase, the design team explores how information can be combined to yield physical solutions to the already defined functions and activities. The purpose is to generate design alternatives and to creatively seek out solutions from as many sources as possible. Whereas the information and analysis phase is concerned with precise definitions and data reduction, in the speculation phase the design team expands the range of design options and builds on the designer's integrative skills. The Delphi technique (Kirk 1980, Linstone 1975), which is a form of brainstorming (Clark, 1958), can be used to generate alternative designs. This technique encouraged open speculation and the uninhibited exchange of ideas among the team members.

2.5.5.3. Evaluation: Alternative solutions which are developed during the speculative phase, do not equally satisfy the project objectives. During the evaluation phase, the design team critically evaluates those solutions. An important criterion used to judge evaluation is the long-term cost implications of various building components. Life-cycle cost analysis measures the effectiveness of a range of building components in terms of costs that will be incurred during the useful life of a building.

2.5.5.4. Synthesis: Perhaps the most delicate aspect of decision making methodology is how the design team combines the separate pieces of a design problem to create a finished program, design, or building. In this process of synthesis, the design team must finally satisfy the original goals of the client and the building users. Those goals now become important measures for re-assembling and testing the components of the closure, structural, mechanical, and electrical subsystems.

After reviewing the client's design goals, the design team begins to measure the performance of individual components against these goals. The various alternatives are consolidated into design solutions that maximize the project objectives.

2.5.5.5. A final phase could be identified, and that is the Recommendations Phase: In the final phase of the methodology, the design team combines all the building component alternatives into a limited number of design development recommendations. In this instance, the recommendations take the form of graphic analysis. During the recommendation phase, concise communication techniques are used to relate the processes of information gathering, speculation, evaluation, and synthesis to the next level of design decision making. Detailed design options move the project from a context of general architectural concepts, to one in which specific proposals for building materials, construction details, and unit-price cost data are analyzed, and integrated into complete building assemblies.

Although the design model proposed by Kirk and Spreckelmeyer is based on a sequential view of the design process, it allows for both non-sequential activities within subphases and their mutual interaction. The model constitutes a hybrid of the first and second generation methods with distinct features of first-generation models during the problem formulation phase.

2.6. A Comparison Between First and Second Generation Design Methods:

There are a number of characteristics or principles of second-generation methods that distinguish them from those of the first generation (Heath, 1984). These characteristics could be summarized as follows:

1. The assumption of the "symmetry of ignorance." It is not a case of an 'expert' confronting a 'layman', which implies that knowledge as well as ignorance are shared among the expert (designer) and the client or user (layman). In reality, both sides have essential knowledge in rather similar amounts; and correspondingly, both sides are ignorant in some vital areas. This is the reason for using participatory and argumentative methods: the knowledge as well as the ignorance of all parties have to be brought out.

2. The second characteristic is the argumentative nature of the architectural design process, which is represented not as a discovery of facts, but as a process of detecting, defining, and deciding issues. This implies that design according to second-generation methods is a process of reaching consensus, through debate, making use of relatively well-established and tested knowledge and logical methods, including some of those developed in the first generation. The constraints on the design problem are decisions, and the making of these decisions is design.

3. It is important that the process of decision making, and the arguments used, should be as "transparent" as possible. This is analogous to Jones' distinction between "black-box" and "glass-box" design methods (Jones, 1969), to moving the whole process as far as possible into the "glass-box" realm. Two reasons for this approach could be detected; the first is that because the process is heuristic, each successive step depends on the total of the preceding steps. It is an incremental decision process that could fail at any time if the preceding step is not clearly understood. The second reason is that if the process seems to be leading in a wrong direction, that is there is no possible solution in terms of the decisions or constraints so far established, then it is necessary to neglect some constraints, undo some decisions, and back-track to a previous design state.

If the argument is transparent, then the process of back-tracking can be minimized, otherwise, the design process would be forced to start right from the beginning, or modifications may be attempted which result in contradictions and failure, either at the stage of implementation or in use.

4. Tied in with the principle of transparency, is the principle of objectification. The process of design must not only be transparent, but must also be recorded. Proper and explicit recording of the process reduces the risk that something critical might be overlooked. It also makes effective criticism and argument much easier. The process of back-tracking, if it becomes necessary, is much easier if agreement has been in the form of explicit consensus.

5. The delegation of judgement to the professional designer should be minimized. The more clearly any agent in the design process has to spell out his reasons for his decisions, the less opportunity he has to build in ideas about what ought to be the case, what the goals of the design are, which may not be agreed upon by the other parties. In practice, the scope of delegation should be clearly understood and agreed upon.

6. The need for the designers and users to bring about first agreement and then action. The outcome of this is less likely to fail than the more conventional process in which more decisions are taken by the designer, and the client or users are faced with a proposal that can only be accepted or rejected. In that sense, it is an interactive rather than a reactive model of the design process.

This is the paradigm change involved in the substitution of second for first generation methods. The new paradigm is based on urban planning and architectural design rather than on industrial design, which was the practical background from which first generation methods largely sprang.

While first generation methods were suited to the solution of "well-constrained" or "well-behaved" problems, the intention of second generation methods was to extend the scope of method to "ill-constrained", "wicked", or "ill-behaved" problems. In order to do so, the logical rigor characteristic of first generation methods had to be abandoned (Heath, 1984).

So far, we have discussed the evolution of different design methods, we have briefly presented their history, discussed the first generation models, as well as the second generation models, and pointed out the main characteristics that distinguish both of them.

In the following chapter, we will address the issues of problem solving and search in architecture. Different problem finding methods as well as search techniques will be addressed. A definition of "problem space", and "task environment" will be presented. The role of artificial intelligence, and expert systems in architecture will be addressed as well.

REFERENCES FOR CHAPTER 2:

Alexander, C.:
"A City is Not a Tree." In Architectural Forum, vol. 122, April/May, 1965, pp. 58-61.

Notes on the Synthesis of Form. Cambridge, Massachusetts: Harvard University Press, 1971. First Published in 1964.

The Oregon Experiment. New York: Oxford University Press, 1975.

The Timeless Way of Building. New York: Oxford University Press, 1978.

The Linz Cafe. New York: Oxford University Press, 1981.

The Production of Houses. New York: Oxford University Press, 1985.

Alexander, C.; Ishikawa, I.; and Silverstein, M.:
A pattern Language. New York: Oxford University Press, 1977.

Archer, L. Bruce:
"An Overview of the Structure of the Design Process." In Emerging Methods in Environmental Design and Planning, edited by G. Moore. Cambridge, Massachusetts: MIT Press, 1970.

Asimow, Moris:
Introduction to Design. New York: Prentice Hall, 1962.

Bazjanac, V.:
"Architectural Design Theory: Models of the Design Process." In Basic Questions of Design Theory, edited by W. Spillers. Amsterdam: North Holland, 1974.

Broadbent, G.:
A Semiotic Programme for Architectural Psychology." In Meaning and Behavior in the Built Environment, edited by G. Broadbent, R. Bunt, and L. Tomas. New York: John Wiley & Sons, Ltd., 1980.

Design in Architecture. New York: John Wiley & Sons, Ltd., 1973.

Broadbent, G., and Ward, A.:
Design Methods in Architecture, Architectural Association Paper No. 4. New York: George Wittenborn Inc., New York, 1969.

Caudill, William:
Architecture by Team: A new Concept for the Practice of Architecture. New York: Van Nostrand Reynolds, 1971.

Clark, T.:
"Needed: Graphics Standards." In Computerworld, April, 22, 1985.

Coyne, R. D.; Rosenman, M. A.; Radford, A. D.; Balachandran, M.; and Gero, J. S.:
Knowledge Based Design Systems. Sydney: Addison-Wesley Publishing Company, Inc., 1990.

Cross, N.:
Developments in Design Methodology. New York: John Wiley & Sons Ltd., 1984.

Cuff, D.:
"The Context for Design: Six Characteristics." In Knowledge of Design, Proceedings of the Thirteenth International Conference of the Environmental Design Research Association, pp. 38-47, EDRA Inc., 1982.

Gero, J.; and James, I.:
"An Experiment in Computer-Aided Constraint Oriented Approach to the Design of Home Units." In Environmental Design Research and Practice, A Joint Program of the Third Annual Environmental Design Research Association Conference (EDRA3) and the Eighth Annual AIA Architecture Researchers Conference (AR-8) (Ed. W. J. Mitchell), 1972. University of California, Los Angeles, pp. 20-1-1 to 20-1-9.

Grabow, S., and Alexander, C.:
The Search for a New Paradigm in Architecture. Boston: Oriel Press, 1983.

Gregory, S.:
The Design Method. London: Butterworths, 1966.

Heath, Tom:
Method in Architecture. New York: John Wiley & Sons Ltd., 1984.

Jones, J. C.:
"A Method of Systematic Design." In Developments in Design Methodology, edited by Nigel Cross. New

York: John Wiley & Sons Ltd., 1984. This paper was first published in 1963.

"The State of the Art in Design Methods." In Design Methods in Architecture, edited by G. Broadbent and A. Ward. George Wittenborn Inc., New York, 1969.

Design Methods: Seeds of Human Futures. New York: John Wiley & Sons Ltd., 1970.

Jones, J. C., and Thornley, D. G.:
Conference on Design Methods. London: Pergamon Press, 1963.

Kalisperis, L. N.:
A Conceptual Framework for Computing In Architectural Design. A Ph.D. Dissertation. The Architectural Department. The Pennsylvania State University, 1988.

Kirk, Stephen J.:
"Delphi: An Aid in Project Cost Control." In Architectural Record. December, 1980, pp. 51-55.

Kirk, Stephen J., and Spreckelmeyer, Kent:
Creative Design Decisions. New York: Van Nostrand Reinhold Co., 1988.

Kuhn, T.:
The Structure of Scientific Revolutions. Chicago: The University of Chicago Press, 1962.

Linstone, H. A., and Turoof, M., eds.:
The Delphi Method: Techniques and Applications. New York: Addison-Wesley, 1975.

Luckman, J.:
"An Approach to the Management of Design." In Design Methods in Architecture, edited by G. Broadbent, and A. Ward, pp. 128-135. George Wittenborn Inc., New York, 1969.

Mitchell, W. J.:
Computer-Aided Architectural Design. New York: Van Nostrand Reinhold Company Inc., 1977.

Mitchell, W. J., and Dillon, R.:
"A Polyomino Assembly Procedure for Architectural Floor Planning." In Environmental Design: Research and Practice, Proceedings of the EDRA3/AR8 Conference. Los Angeles, California: EDRA, 1972.

Moore, G.:
Emerging Methods in Environmental Design and Planning. Cambridge, Massachusetts: MIT Press, 1970.

Pena, W.; Parshall, S.; and Kelly, K.:
Problem Seeking: An Architectural Programming Primer. Washington, AIA Press, CRSS Publication 1987.

Popper, K.:
Conjectures and Refutations. London: Routledge and Kegan Paul, 1963.

Protzen, J.:
"The Poverty of The Pattern Language." In Design Studies, vol. 1, no. 5, 1980, pp.291-295.

Rittel, H.:
"Some Principles for the Design of an Educational System for Design." In Journal of Architectural Education, vol. XXVI, no. 1-2 Winter-Spring, 1971.

"On the State of the Art in Design Methods." DMG Occasional Paper no. 1: DMG fifth Anniversary Report, pp. 5-9. San Francisco: The Design Methods Group, 1972.

"Son of Rittelthink", an interview in the DMG Occasional Paper no. 1: DMG fifth Anniversary Report, San Francisco: The Design Methods Group, 1972b.

Rittel, H.; and Kunz, W.:
Issues as Elements of Information Systems. A Report submitted for the "Institut fur Grundlagen der Planung", Stuttgart University, West Germany, July, 1970.

Rittel, H.; and Webber, M.:
"Dilemmas in a General Theory of Planning." In Policy Sciences, no. 4, published by Elsevier Scientific Publishing Company, 1973.

Simon, H.:
"The Structure of Ill-Structured Problems." In Developments in Design Methodology, edited by Nigel Cross. New York: John Wiley & Sons Ltd., 1984. This Paper was Originally Published in Artificial Intelligence, no. 4, 1973.