Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

User Interface And Data Structure Design For Architectural Expert System

by
Yonghua Ma

Computers are currently used to generate drawings in the architectural design process. This thesis anticipates the development of an Architectural Expert System, which could provide assistance in the decision-making processes in architectural design. The focus is on the design of the user interface of such a system. Architectural object-oriented data hierarchies are created. Also, a set of drawing tools, which assist the user in the creation of architectural objects such as walls, windows, and doors, are provided.

USER INTERFACE AND DATA STRUCTURE DESIGN FOR ARCHITECTURAL EXPERT SYSTEM

by
Yonghua Ma

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements
for the Degree of Master of Architecture
School of Architecture
October, 1992

APPROVAL PAGE

User Interface And Data Structure Design
For Architectural Expert System

by
Yonghua Ma

Dr. Filiz Ozel, Thesis Adviser
Assistant Professor of Architecture, NJIT

Barry Jackson, Committee Member
Associate Professor of Architecture, NJIT

Dr. David Hawk, Committee Member
Professor of Architecture and Co-Director of the Center
for Building Engineering and Architectural Research, NJIT

BIOGRAPHICAL SKETCH

Author: Yonghua Ma

Degree: Master of Architecture

Date: October, 1992

Undergraduate and Graduate Education:

. Master of Architecture, New Jersey Institute of Technology, Newark, NJ, 1992

. Master of Architecture, Southeast University, Nanjing, P.R.China, 1988

. Bachelor of Architecture, Southeast University, Nanjing, P.R.China, 1985

Major: Architecture

ACKNOWLEDGEMENTS

I would like to express my utmost gratitude to my adviser, Dr. Filiz Ozel, for her advise, criticism, and support. Without her guidance, this thesis would not have been possible.

I also would like to express my thanks to professor Barry Jackson, and Dr. David Hawk, for their very helpful comments.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1
INTRODUCTION

Architectural design needs to meet a wide range of aesthetic, functional and technical requirements. It requires a wide range of knowledge from many disciplines including: structures, construction, social sciences, engineering, psychology, etc. Architectural design objects are traditionally represented by drawings. Computer graphics is becoming accepted by architects as a tool to help generate drawings. Computers, plotters, and printers are used in place of pencils, triangles, and T-squares.

Is it possible that CAD systems can also be used to assist architects in their more fundamental decision-making processes, such as in design knowledge representation. To find out what computers can do in the wider field of architecture, we have to first understand what architects do when they design a building.

1.1 Architectural Design Process

A building can be conceived as a container of activities. The original goal of architectural design is to provide a controlled environment so that certain human activities can be carried out conveniently, comfortably and safely. To design a project, architects interpret its objectives,

articulate its goals, and find the problems that need to be solved in each phase. Thereafter they work out the solutions and evaluate them.

This should not be a one-way process. After evaluation of a solution, it may be found not to be a good one. Therefore, designers try to get another solution. Sometimes, a previous problems will return as a later problem is solved. In this case, designers have to go back and work on that problem again. This circle may repeat several times before a satisfactory solution is reached (figure 1-1).

The solution to any problem often results in new problems to be solved. For example, in a certain phase in the design of an office building, the ceiling height of a office room is set to be 8'-6" for economical reasons. It could be a comfortable room if well designed. With the design taken further, the building air conditioning system needs to be designed. The ceiling height of the room will soon be a problem if the ducts are to be located in the ceiling. The air that comes out of the ducts may be 15 F-degree or more warmer than the room temperature. People will thus feel uncomfortable when resulting hot air blows directly onto them. It is not a good solution to use air with lower temperature. Low air temperature reduces the delta T between the intake air

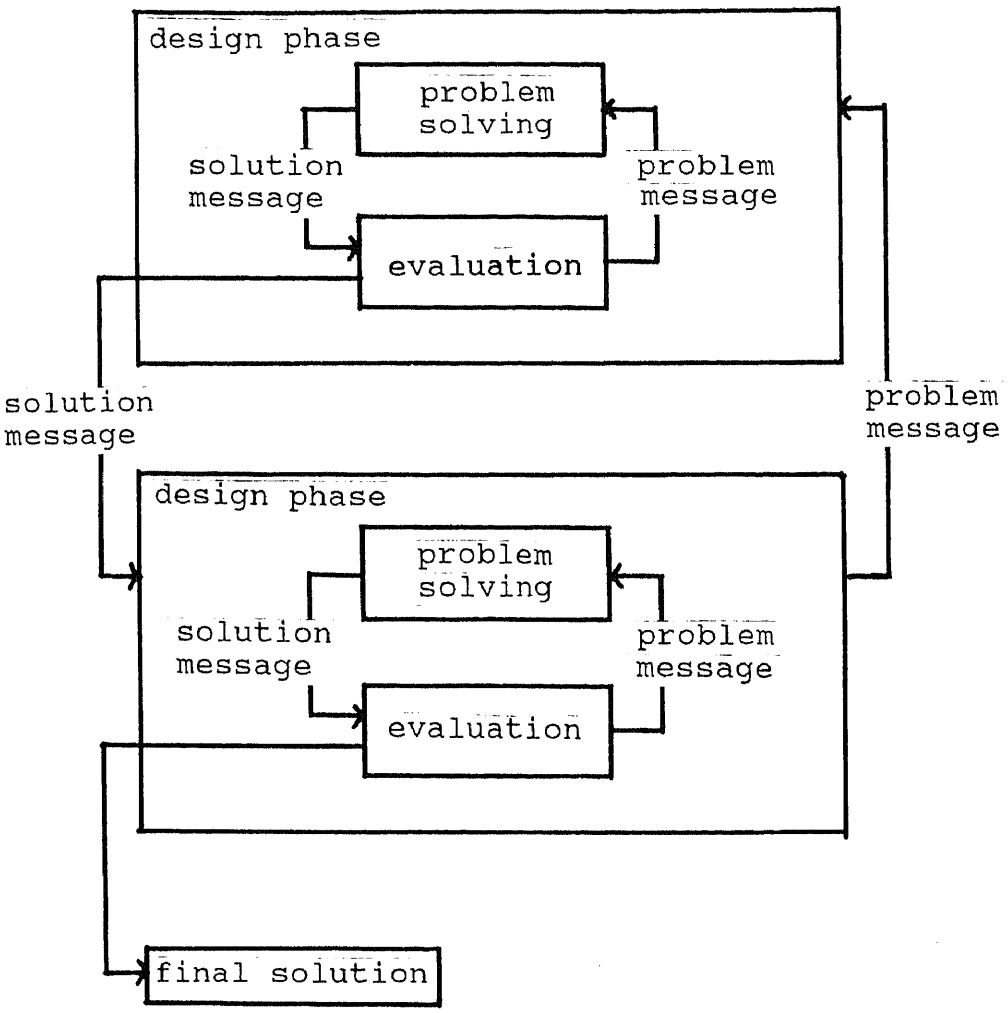


Figure 1-1 Architectural Design Process

and the room. It will lower heating efficiency, and cause the waste of energy and money. To solve it, the room height has to be raised. (figure 1-2).

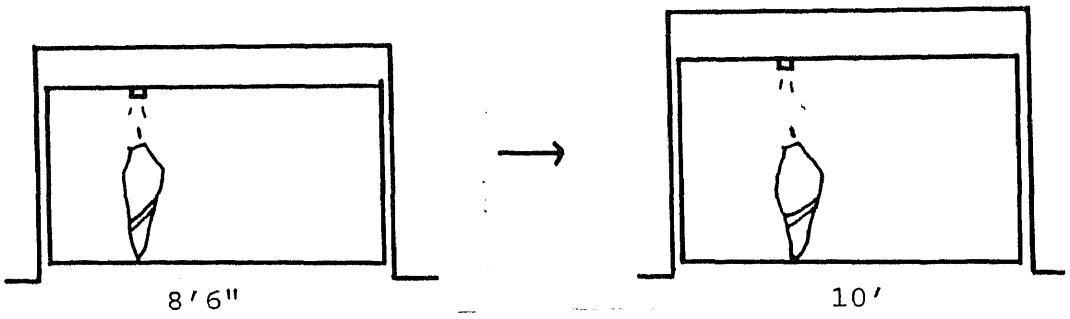


Figure 1-2 Room Height and Air Conditioning System

Meeting this situation, we must return to the previous phase to re-decide room height and re-adjust many relations to different parts of the building.

1.2 Uncertainties In Architectural Design

To interpret design objectives and set goal in the different phases, architects need to use much personal judgement from the study of human needs. This opens up many different choices. Even if an objective is interpreted in the same way, each designer can choose different methods to reach it (figure 1-3).

This is because of different psychological responses, experiences and so on. As stated before,

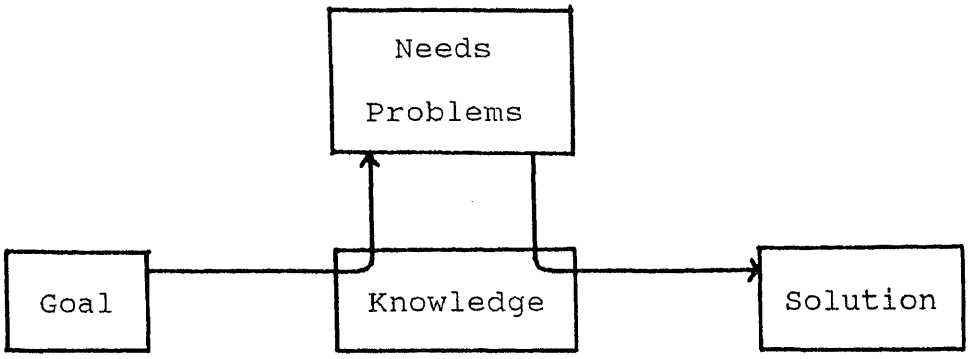


Figure 1-3 Design Model

buildings should be designed to meet the requirements of convenience, comfort, and safety (figure 1-4).

It should be kept in mind that it is impossible for every user to feel the same level of comfort in a building. Many factors such as temperature, color, lighting, humidity, noise, scale, etc. affect people's feeling of environmental comfort in a building. Perception of a building and psychological responses to an environment vary greatly from person to person. It is quite difficult, if not impossible, to set up standards in all these matters. A room may be warm enough or not, sufficiently cheerful or not, etc., according to professional experiences and knowledge in relative disciplines. A designer reaches his own decision on how users will perceive his building by their senses, their

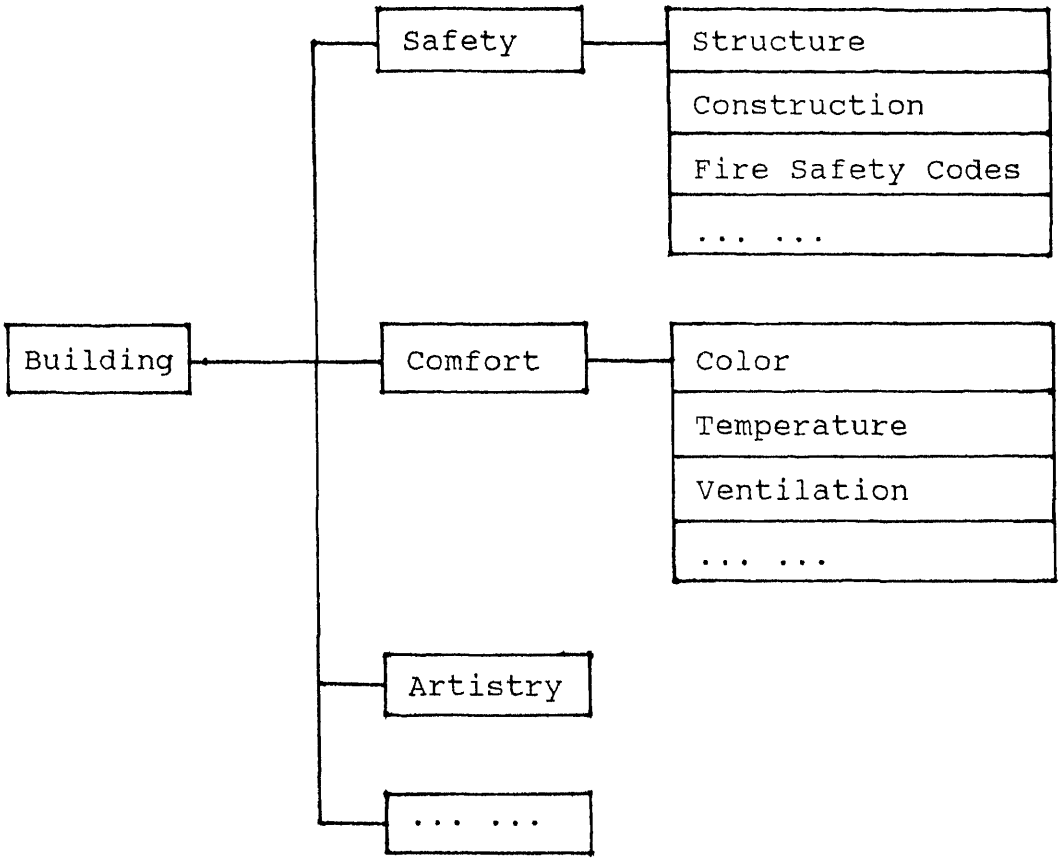


Figure 1-4 Building Requirements

understanding of psychological responses, and ideas about comfort.

1.3 Computer Applications

In architectural design, solutions are usually presented by drawings. Most current CAD systems are used to this.

This is called Computer Aided Drafting. To assist design, to assist the decision-making processes in architectural design, computers should also be used in the process of problem solving and solution evaluation.

Does this mean that computers could give out the final design automatically when they are used in all aspects of architectural design, such as problem solving, solution evaluation, and design representation? This is clearly not the case. The potential of computers in the field of architecture should not be exaggerated. Looking at the inherent characters of architectural design, we can say that it can never be a completely automated process.

We discussed the uncertainties in architectural design above. Computers are not good in such jobs. CAD systems will deal in certainties, or high probabilities which are changed from uncertainties by architects. As shown in figure 1.2-2, safety requirements of a building include compliance with some safety codes, which are set as rules. These codes are minimum requirements for all buildings, or for certain types of buildings. They are criteria. If they are interpreted to a computer, the system will be able to do code-checking. That is what we want a CAD system to do, and leave the jobs that need soft-edged skills up to architects.

1.4 Architectural Expert System and User Interface

An Architecture Expert System is such a CAD system that assists not only the drawing-generating but also the decision-making processes in architectural design. This study anticipates the development of a user interface for such a system.

One of the focuses of this study will be the design of object hierarchies, which can be linked to an interpreted architectural knowledge base. These hierarchies should be consistent and logical, and enable one to analyze and evaluate buildings for compliance with building codes and other requirements.

A user interface can be envisaged as an extension of the short-term memory of the user. So it is very important to understand the mechanisms by which the user processes information. As the users of Architectural Expert Systems are designers, architects, etc., we should consider the specific needs and limitations of the architectural design process. The other focus of this study will be to provide drawing tools, object graphic representation methods familiar to architects. The design of these tools and representations will be based on the study of architectural design process and traditional architectural representation. All the functions should be useful and easy for architects to learn.

The interface will be based on AutoCAD (Autodesk inc.). The reason for this selection is that AutoCAD has an open structure. By using its programming language AutoLISP, one can create a new menu, a command language, as well as a data structure.

CHAPTER 2
OBJECT REPRESENTATION

2.1 Knowledge And Object-Oriented Data Base

As we discussed before, architectural design is a goal-directed activity. To design a project, architects interpret the goal, set objectives, find out the solutions for problems, and then evaluate the solutions. During this process, knowledge about architecture, as well as many other disciplines, is required.

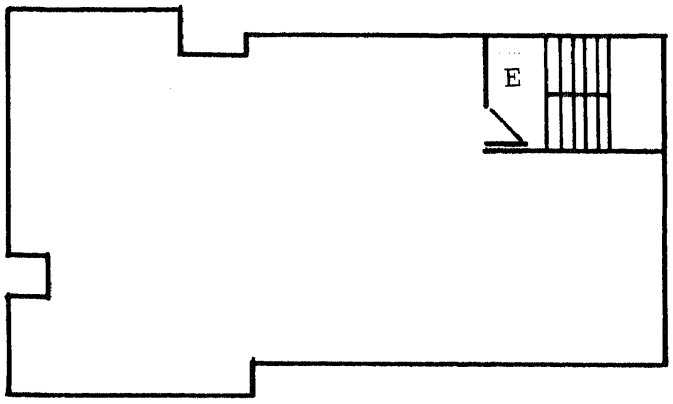


Figure 2-1 Exit Space in a Building

Figure 2-1 is a building plan. E is an exit-space of the building. According to architectural knowledge, it must be enclosed. To use a CAD system to process such a checking, we use the function as below:

```
Function: exit-space checking
          .check if space E is an exit
          .if NO
                ..end the function
          .else (YES)
                ..check if the space is enclosed
                          &......
                          ......
                ..prompt the result of
                          the checking
          .end the function
```

However, it is impossible to use this function on general CAD systems, such as Auto CAD. Because although space E needs to be closed, it could have door(s). The two plans in figure 2-2 are different.

In figure 2-2(a), space E is not enclosed. Wall W stops at point p2. In fig.2-2(b), D is a door in wall W. Space E is enclosed, although wall W seems to stop at point p2 too on the drawing. Plan (b) meets the requirement while plan (a) does not. If both plans are .DWG files created by Auto CAD, they do not have essential structure to represent the difference.

All entities in Auto CAD are stored in terms of

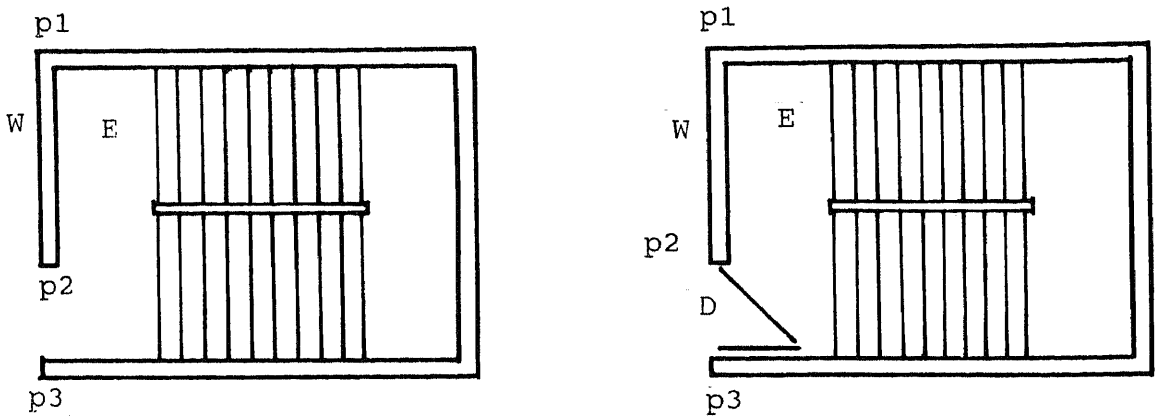
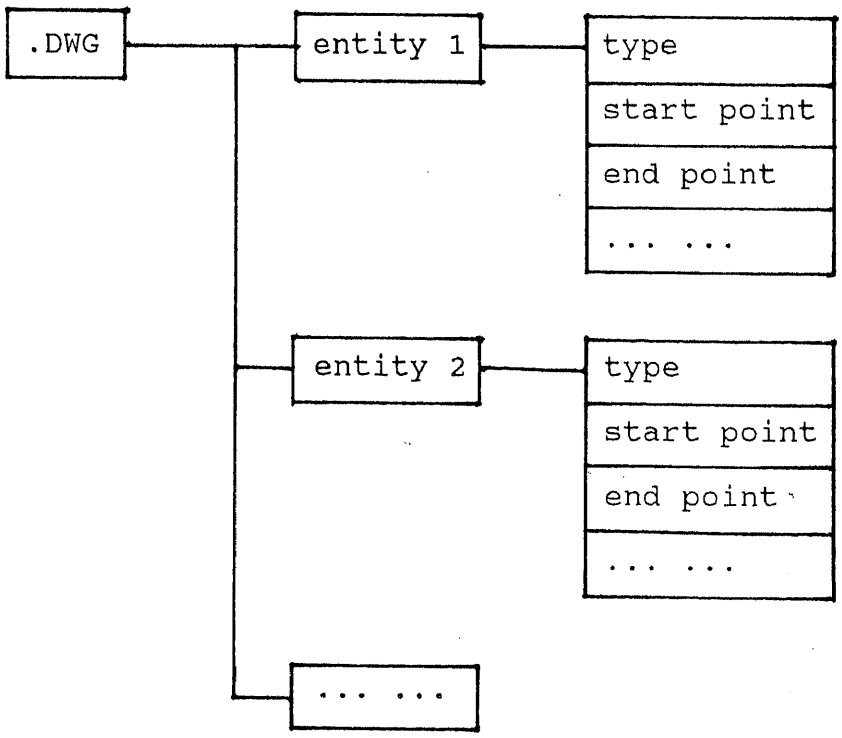
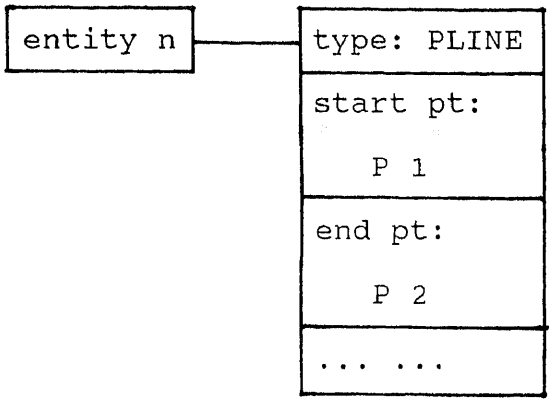


Figure 2-2 Closed and Unclosed Spaces

points, lines, plines etc. Entities exist independently. There is no relation among them:

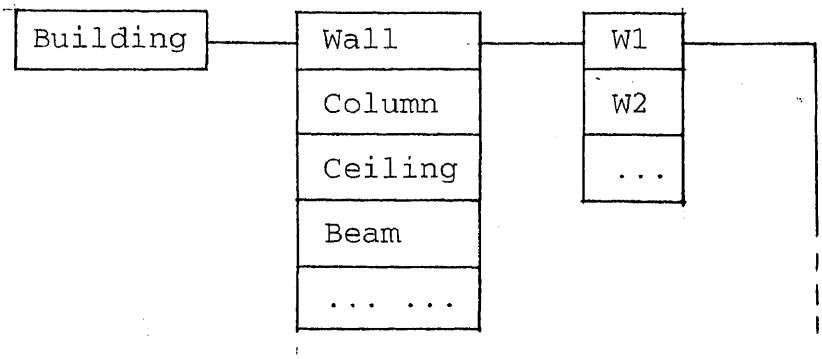


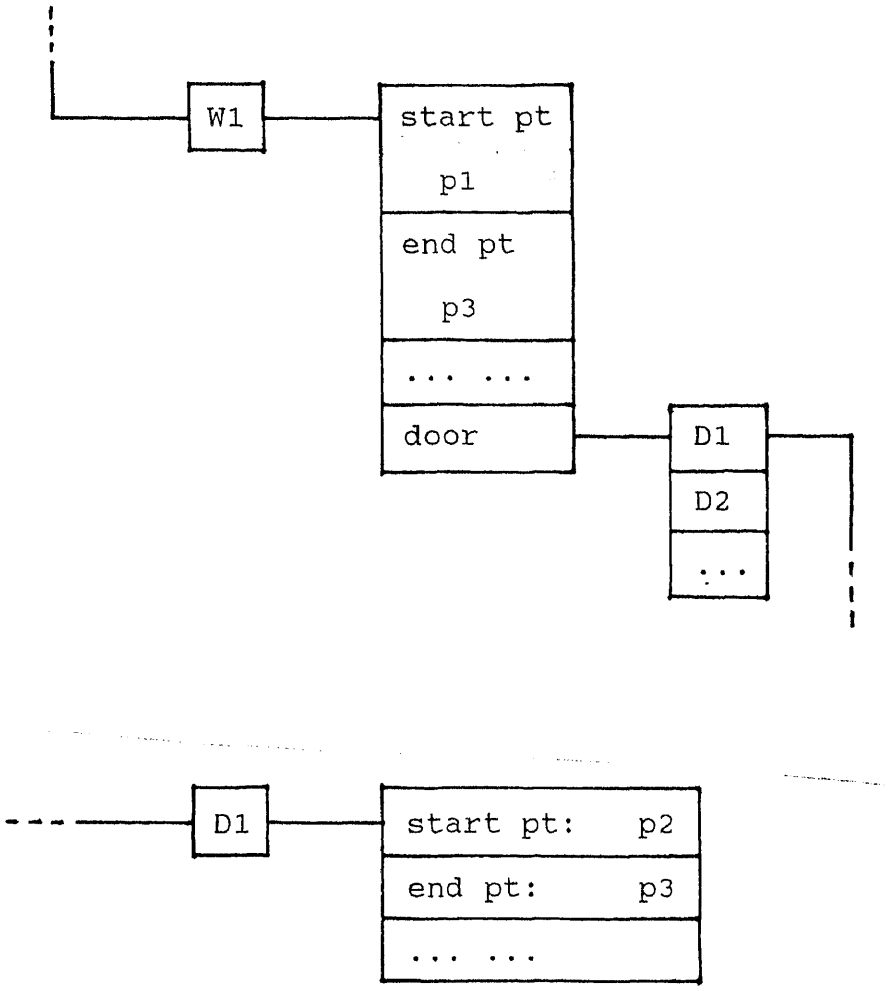
Wall W in either plan (a) or plan (b) is:



You may even erase wall W in fig.2-2(b), while door D is still there, which becomes meaningless.

In .DWG file, entity D and W are not door and wall. They are lines same as steps, etc. To check if space E is enclosed, computer needs to check its walls, floors. We can see, the enclosing checking can not be done in such a case. Knowledge is based on objects. It can not be used without certain objects. We hope the plan to be stored in terms of objects as following:





While wall W stops at point p2 (figure 2-2 (a)), space E is not enclosed. In figure 2-2 (b), wall W stops at point p3, separates space E from other parts of the building. D is only a door in wall W. If wall W is erased, door D will not exist any more, because they are stored in the computer in an "object-oriented" structure. In such a data base, entities exist more similar as they do in real word. Then many architectural knowledge can be used based on such a data base.

2.2 Building Components, Spaces And Data Base

Now, we know we need an object oriented architectural data base for design evaluation, code checking etc. In this data base, architectural elements should exist just as they are in real world.

What are those elements? What kind of object hierarchy this data base should have? Let's have a look at a building in real world.

Usually, one would say that a building consists of many rooms, which are enclosed by wall(s), ceiling(s), and floor(s). And there are some openings, such as doors and windows, in those walls (figure 2-3).

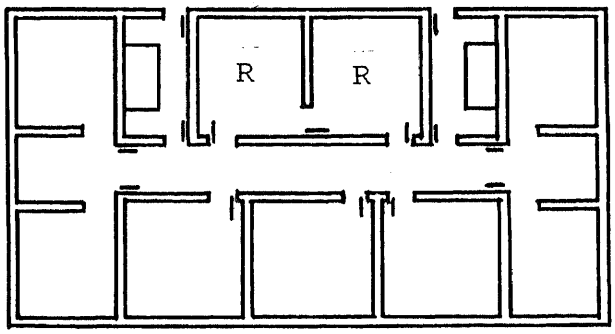
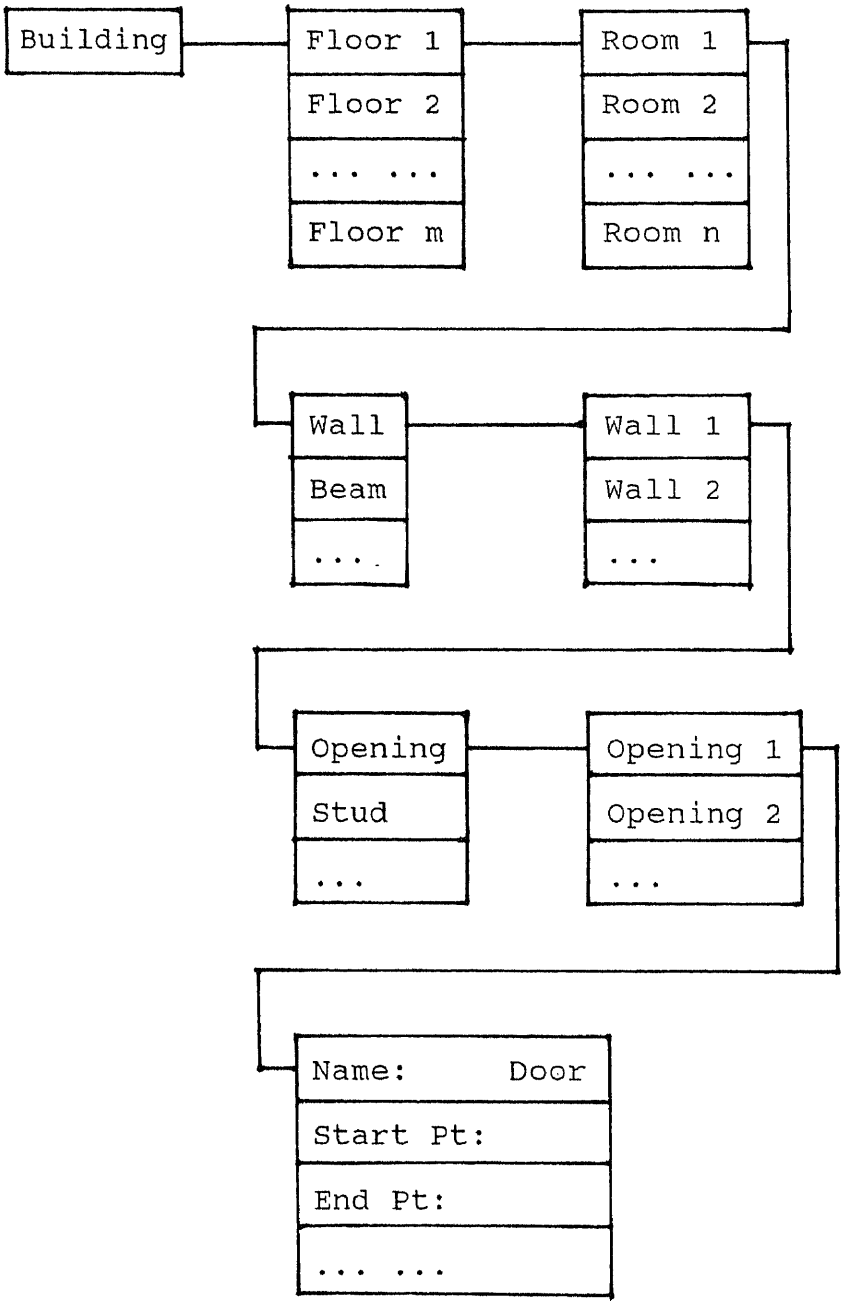


Figure 2-3 A Building Plan

It is natural that we would assume the data base should be as shown in below:



This hierarchy reflects architects' concepts of buildings.

As we discussed before, a building is actually a container of human activities. To be a container, it provides a set of rooms (spaces) to shelter human activities. In their design, architects are concerned about spaces where people will live or work.

Because rooms (spaces) are considered as objects in the hierarchy of the data base, such applications as design evaluation, code compliance checking are possible. However, it also results in some problems to put rooms as objects in the same hierarchy with other building components such as walls, openings, etc. Let's use the plan in figure 2-3 as an example (figure 2-4).

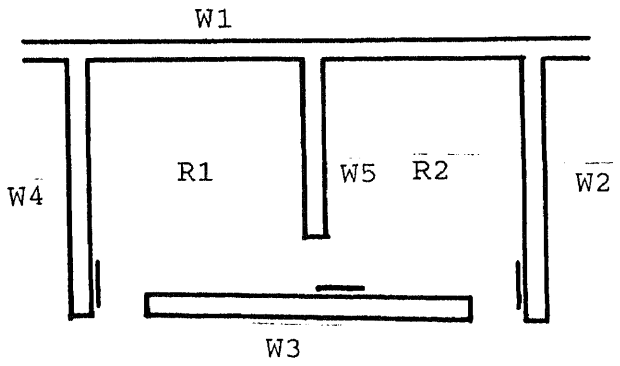


Figure 2-4 Rooms and Walls

Actually, room R1 and R2 have a total of five walls. In the data base, each room has four walls. R1 and R2 are composed of eight walls (figure 2-5).

Obviously, they are not the same as they are in the real world.

Actually, wall W4 and W8, W2 and W6 in figure 2-5 are the segments of wall W1 and W3 in figure 2-4. Wall W1 and W7 in figure 2-5 are the same wall. They are

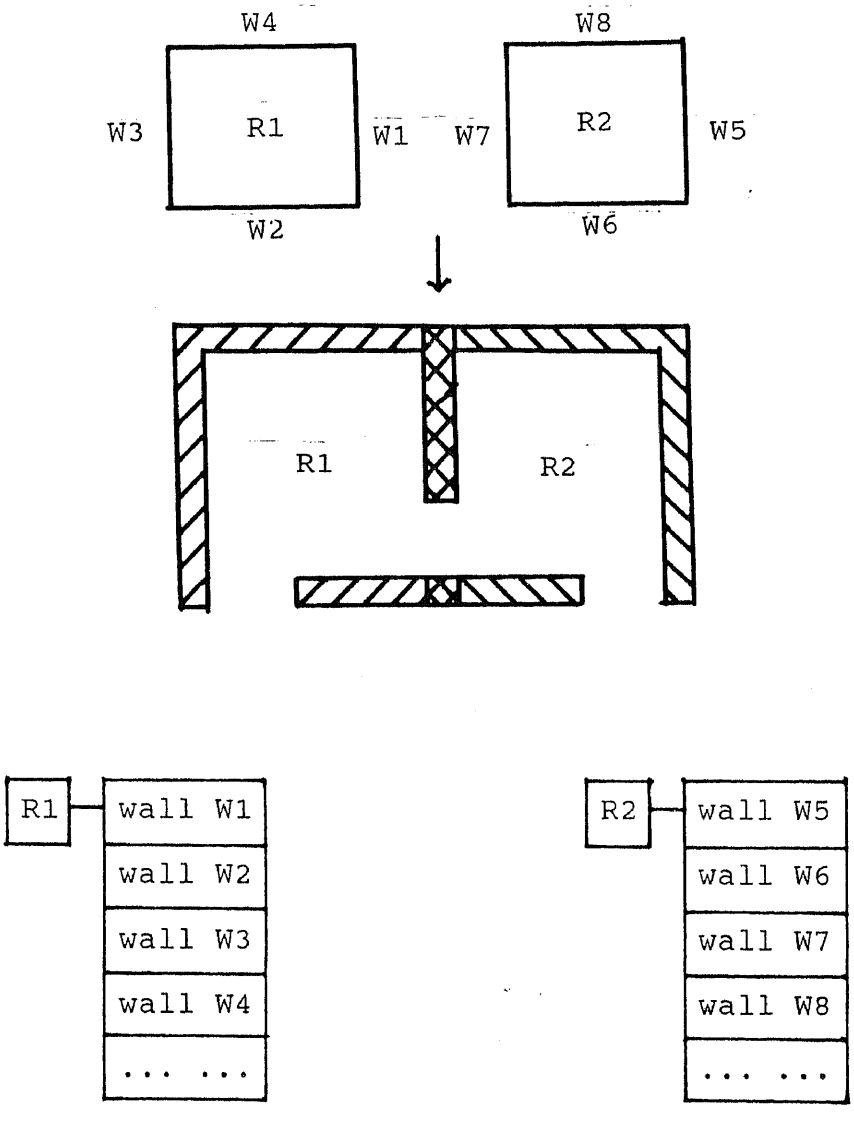


Figure 2-5 Duplicated Walls Between Spaces

duplicated. There should be only one wall separating room R1 from room R2 (W5 in figure 2-4). In the data base, they are stored as two different walls. While using this system, user has to draw those walls separately. All the subcomponents, such as doors, windows, and studs, have to be drawn twice too. It is not difficult to anticipate how much more drawing work one has to do while drawing a large project which has a lot of adjacent rooms (spaces) that share the same walls. Of course it is not convenient for users. Meanwhile, it takes more time for computers to generate drawings. The processing speed of computers will be much slower.

Besides the drawing problem, we will also get problems during some evaluation and/or code checking processes, due to the duplication of the walls shared by two rooms. Those problems may be more serious.

For example, we want to estimate the cost of a building. The cost of materials are needed. Those materials are used to build walls, floors, etc. So the total volume of walls has to be calculated.

Budget = Cost of Materials (COM) + ...

COM = Volume of Walls * .....

To get the right result, each wall should only be calculated once. If some walls are shared partially or totally by different rooms, they are stored as different walls in data base. There may be several walls in the data base to represent one wall in the real world. It is necessary for the system to find out which walls are duplicated, so that they will not be calculated twice.

As we know, one important function of a wall is to divide spaces. Therefore most walls of a building belong to more than one space. To remove all duplicated walls, a lot of calculation work has to be done. This also reduce the prompt speed of the system.

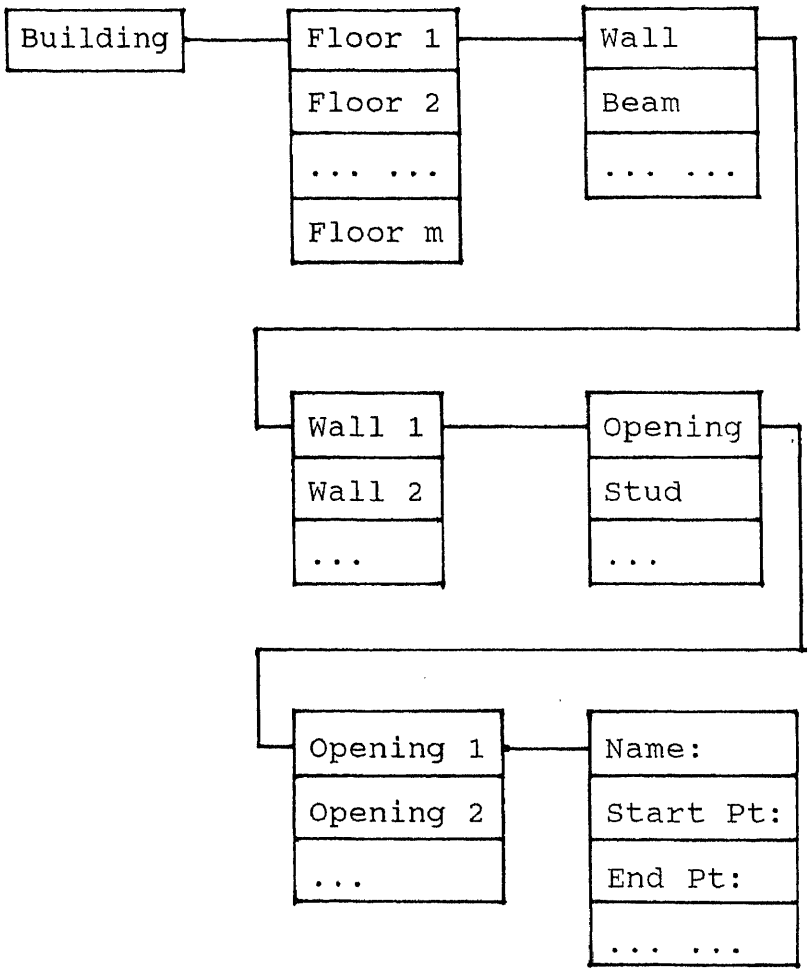
The reason of why we have so many problems is that rooms are used together with walls, floors in the hierarchy of data base. Rooms are regarded as a type of building component. However, rooms are different from walls, floors and other building components.

2.3 Building Component Hierarchy

Building components, such as walls, floors, are used to build a building. We never use rooms to build a building. Rooms are what we will get when a building is built. They are not physical objects, although people can perceive them. They are only the spaces enclosed by certain architectural elements. There are not the objects named

as rooms in the real building hierarchy. In this data base, we add rooms to the building component hierarchy, where they do not exist.

The solution is to create another hierarchy for the data base reflecting the building hierarchy in the real world:



In this hierarchy, objects reflect their positions in the real world. All these elements are those people can see and touch in the real world. Other elements that can only be conceived by human, such spaces, are removed.

A building is divided into different floors. On each floor, there are walls, steps. In some walls, there may be some doors and/or windows.

Compared with the previous object hierarchy of the data base, this one is more clear and simple. Not only the duplicated walls are excluded from the data base, but also a wall could be stored as a complete one in the data base, instead of being stored as several different walls (figure 2-6).

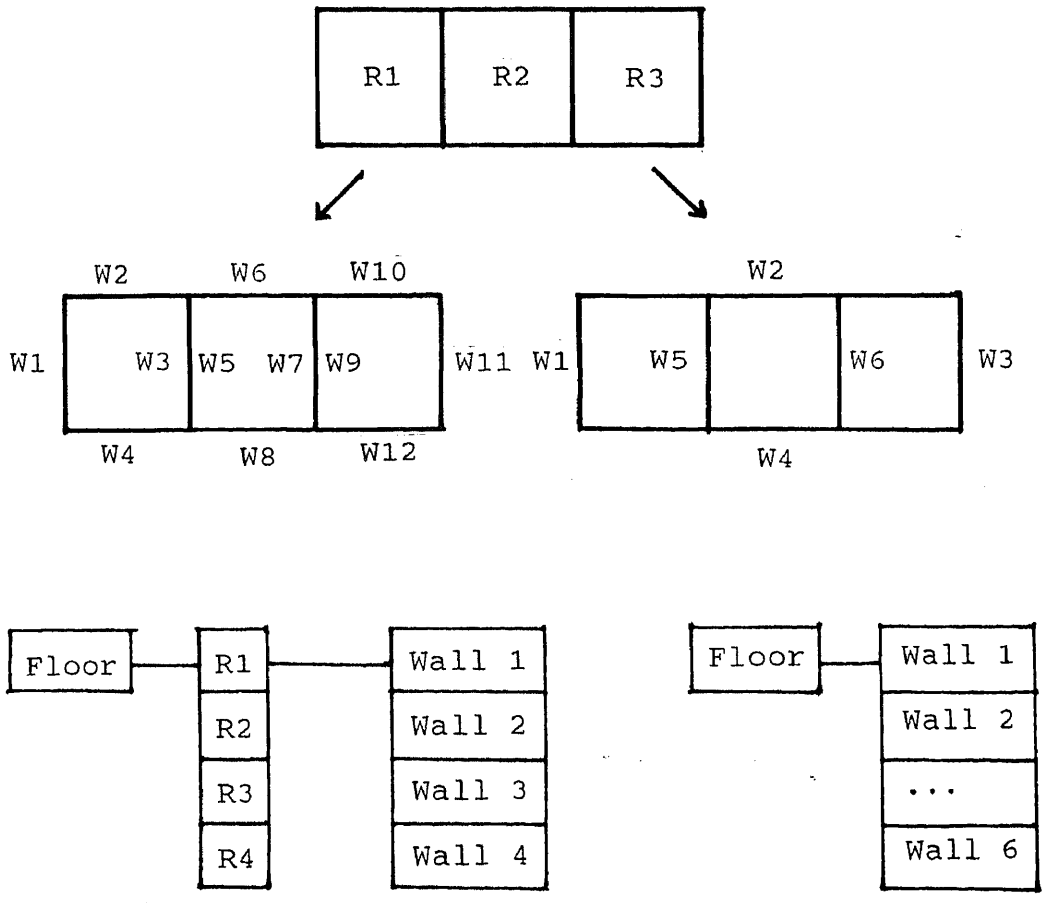


Figure 2-6 Building Objects in Different Hierarchies

Computer memory for storing data and running time for checking the relationships among the building components are saved.

2.4 Space Hierarchy

As rooms are removed from the building component hierarchy, there are no duplicated objects in data base any more. This is good for certain evaluation as well as graphic representation. But "room" is such an important concept in architectural knowledge that we can not design any building without this concept. It is even impossible to evaluate designs without it. Actually, we should use "space" here instead of "room". Room is a specific name for certain kinds of spaces.

People use the spaces of a building. If we want to design a comfortable and safe building for people, we have to be concerned more about spaces. That is why many building codes are about spaces. There are some requirements for building components. However, they are mostly for the spaces of the building. Following is the 5-1.3.1c (pp.37) in Life Safety Code:

> Any opening (of an exit) shall be protected by an approved self-closing fire door.

It sets out the requirement for openings. The purpose is for the exit. The exit is defined (5-1.2.3 pp.31) as:

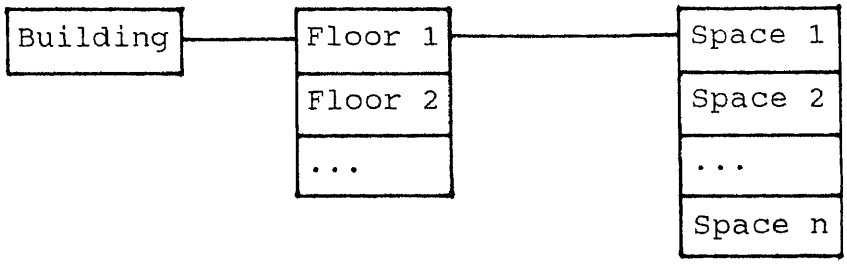
> Exit is that portion of a means of egress which is separated from all other spaces of the building or structure by construction or equipment ... to provide a protected way of travel to the exit discharge.

It is very clear that exit is a space. The requirement is for the openings of a specific kind of space. Therefore, to check if the openings of a space is in compliance with Life Safety Code, the space has to be checked first. Only when it is an exit, its openings should meet the requirement.

In our data base, spaces are removed. The system does not know what a space is. It is not important to the system where an opening is. With all objects and attributes of any object in the data base, we are not able to let the computer know what a space is. We have to use spaces as a kind of basic element in data base.

We have discussed before, spaces are not the same type of elements as other building components. It will result in a lot of problems if we add spaces in building component hierarchy. So, we need to create another

functional element hierarchy in the data base, which includes spaces:



This hierarchy is parallel to the component hierarchy.

When a set of walls are drawn, it becomes very clear to humans where spaces are (figure 2-7).

But computers can not know. Defining a space to a system is actually same as defining a polygon. A space is stored as a polygon in data base, It is 2D. It can be used for 3D calculation. Because the position of this polygon is known to the system, the elevation of the floor and the ceiling at that position can be found too. So the height of the space can be calculated.

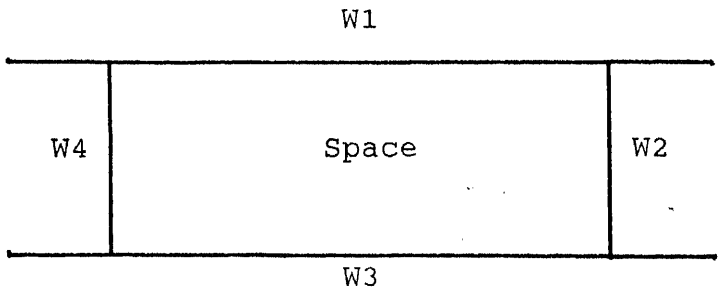


Figure 2-7 Space Defining

To define a polygon, we only need to know the end points (vertices) and the types of all edges (figure 2-8).

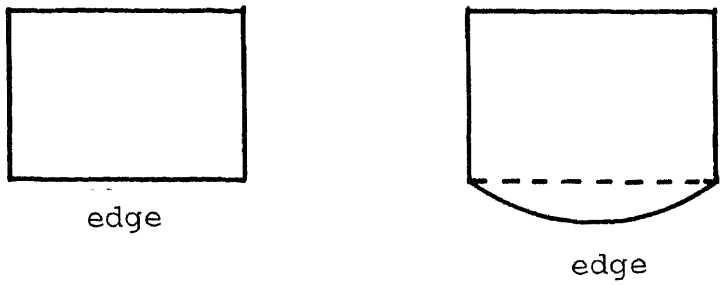


Figure 2-8 Vertexes and Edge Types Determine a Polygon

However, the polygon is used to represent a space. The type of a polygon edge should include the information of the building component(s) on this side of the space. Is it formed by a wall, a set of columns, or nothing as an open space?

Also, the attributes of spaces are determined by the enclosing walls, floors and other building components. It is meaningless if only the position and form of a space is known. The attributes of all enclosing components should be accessible. The relationships between spaces and building components have to be reflected in the hierarchy (figure 2-10).

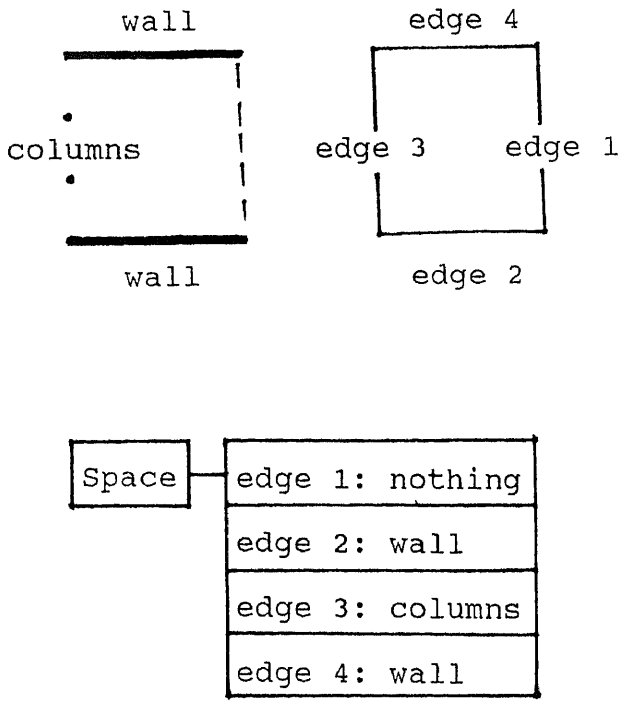


Figure 2-9 Spaces in Hierarchy

We can find some parts of this hierarchy are same as those of building component hierarchy. All the walls and openings are already stored in the data base. It is not necessary to store them repeatedly. The functional element hierarchy only needs to provide access to those data stored in building component hierarchy. That means in functional element hierarchy, each edge of the polygon only needs some pointers which point to those building components (figure 2-11).

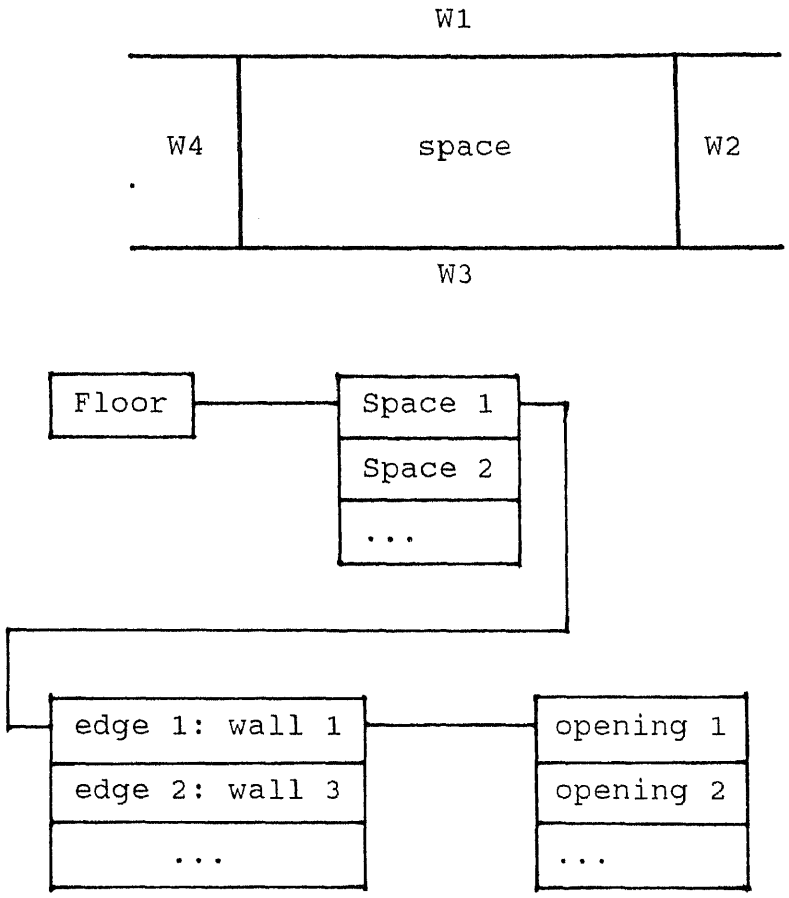


Figure 2-10 Space and Building Components

As the figure above, space S1 is a polygon which has four edges, ED1, ED2, ED3 and ED4. In functional element hierarchy, the pointers point to the enclosing walls in building component hierarchy. And start points, end points define the specific segments of those walls which form the space.

Now we may check what kind space it is. If this

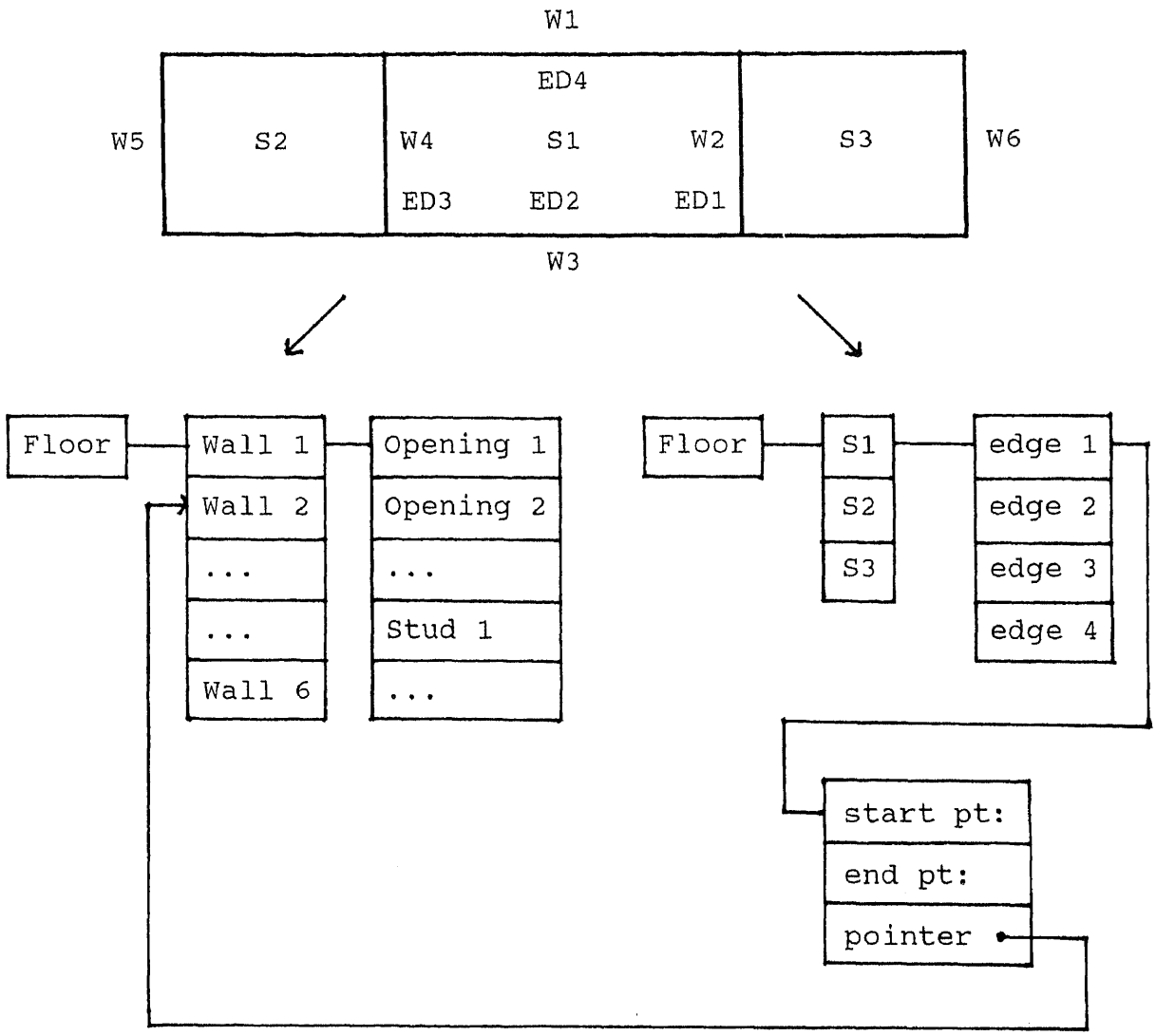


Figure 2-11 Space Hierarchy

space is an exit. We want to check its openings in wall ED1 for compliance with the requirement mentioned before. From the given data, we know the edge ED1 is a part of wall W1 in building component hierarchy. Then, we can

check all the openings in edge ED1 to find out which are between the start and end points of edge ED1.

```
Wall W1: op1, op2, op3, ..... op n.
Edge ED1: start point p1, end point p2.
i = 1
while i <= n do
(if start point or end point of opening(i) is
     between p1 and p2
     (put opening(i) into opening list OP-LST)
 i = i +1
)
```

After this process, we get the opening list OP-LST. All the openings in OP-LST are in the wall ED1, and should meet the requirement.

CHAPTER 3
ABOUT AUTOCAD

As our user interface is based on AutoCAD, it is necessary to introduce some basic concepts of entities, and data structure of AutoCAD.

AutoCAD provides a set of entities for use in constructing a drawing. An entity is a drawing element such as a line, polyline, or circle.

The following table is given by AutoCAD Reference Manual.

Table 1 AutoCAD Entities

Entity type	Description
Lines	Lines can be drawn with various dot-dash linetypes. When drawing a line segment, you can provide either 2D *(x,y)* coordinates or 3D *(x,y,z)* coordinates.
Arcs and Circles	Arcs and Circles can be drawn with various dot-dash linetypes. Several methods are provided for drawing arcs and circles.
Points	Points can appear as dots, squares, circles, X's, or any combination of these. You can locate Point entities using either 2D or 3D coordinates.
Blocks	Blocks are compound entities formed from groups of other entities.
Attributes	Attributes attach constant or variable text information to each instance of a Block. You can choose whether or not they are visible.
Dimensions	Dimensions (generated when associative dimensioning is enabled) are compound entities similar to Blocks, containing all the lines, arcs, arrows, and text comprising a dimensioning annotation.

Entity type	Description
Polylines	Polylines are 2D connected line and arc segments, with optional dot-dash linetypes, width, and taper. Commands are provided to construct ellipses, regular polygons, filled circles, and "doughnuts" using Polylines.
3D Polylines	3D Polylines are fully general three-dimensional objects composed of straight line segments (but no arcs, width, taper, or linetypes).
3D Faces	3D Faces are three-dimensional triangular or quadrilateral plane sections.
3D Meshes	3D Meshes are three-dimensional polygon meshes. You can specify the size of the mesh and the location of its vertices. Commands are provided to construct ruled surfaces, surfaces of revolution, and tabulated cylinders using 3D meshes.
Text	Text can appear in a variety of fonts, with any size and orientation you wish. In addition, you can create text *styles* to apply mirroring, obliquing, or a horizontal expansion or compression factor to the text characters.
Traces	Traces are two-dimensional, solid-filled lines of any width you specify.
Solids	Solids are two-dimensional, solid-filled triangular or quadrilateral objects.
Shapes	Shapes are small objects you can define outside AutoCAD and place in the drawing with a specified scale and rotation.

AutoCAD entities can be drawn at a specified coordinate location. Many of them can be given a thickness. All these data are stored in entity data lists, which can be read by a programming language -- AutoLISP's entity access functions. For example, we draw a line:

```
Command: line
From point: 1,0
```

```
To point: 1,1
To point: <Return>
```

The data list might be:

```
((-1 . <Entity name: 6000003C>)
 (0 . LINE)
 (8 . A)
 (62 . 3)
 (38 . 1.000000)
 (39 . 2.000000)
 (10 1.000000 0.000000)
 (11 1.000000 1.000000)
)
```

In entity data list, "entity name" is actually a pointer. An entity can be accessed if its name is known.

CHAPTER 4
DESIGN DATA STRUCTURE

4.1 Hierarchical Files

Any computer has limited space for storing variables and running programs. We can not predict how many components a building will have. It may be a two-story house or a eighty-story skyscraper. What we do know is that we usually work on a certain floor at a time. It is not necessary to know the information of all other floors. If we create one file for the data of each floor respectively, we can get a set of data files for a project:

```
----------
File name: HOUSE.BAC
(Basic information of the project)
Floor created: 1, 3
The last floor worked on:  3


----------
File name: HOUSE.C1
(Components data of the first floor)
Total walls:  23
WALL[1][1]:
```

```
WALL[1][2]:
... ...
Total columns: 10
COLM[1][1]:
COLM[1][2]:
... ...

----------
File name: HOUSE.S1
(Space data of the first floor)
Total spaces:  8
SPC[1][1]:
SPC[1][2]:
... ...


----------
File name: HOUSE.C3
(Components data of the third floor)
Total walls:  18
WALL[3][1]:
WALL[3][2]:
... ...
Total columns:  4
COLM[3][1]:
```

```
COLM[3][2]:
... ...

----------
File name: HOUSE.S3
(Space data of the third floor)
Total spaces:  4
SPC[3][1]:
SPC[3][2]:
... ...
```

When a user starts the system, and gives the project name HOUSE, the computer reads the file HOUSE.BAC to find out that the third floor is the one on which the user worked during the last edition. Therefore, only the file HOUSE.C3 and HOUSE.S3 are read into the computer space. After the third floor is finished, all the variables are written back to the files, and removed from the computer space. Then the user can work on another floor.

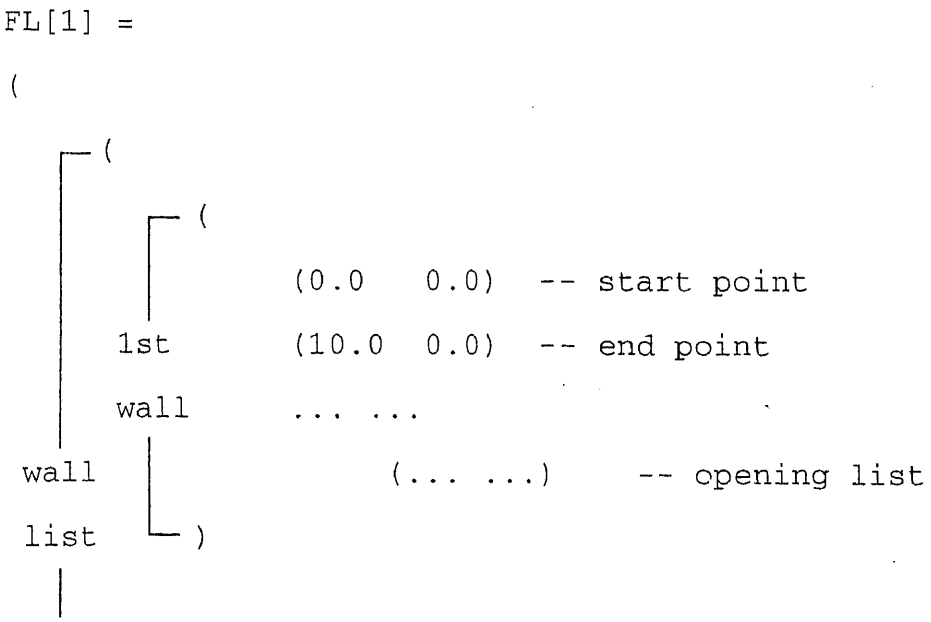
By this way, the number of variable is reduced respectably. Of course, the data for two or more floors should be able to be read at one time if necessary. In any case, variable number is always kept to the minimum.

4.2 Object Data List

4.2.1 Basics Of LIST

AutoCAD and AutoLISP support several basic data types, such REAL, INTEGER, STRING, and LIST. The LIST is similar as the STRUCTURE in C. It is a collection of one or more entities, possibly of different types, grouped together under a single variable name for convenient handling. LISTs help to organize complicated data, particularly in hierarchical data structure, because they permit a group of related entities to be treated as a unit instead of as separate ones. And those entities can be lower level LISTs.

For a project, we use a set of LISTs, name FL[1], FL[2], etc., to store the data of each floor.



```
    |   2nd (
    |   wall      ... ...
    |        )
 wall
 list       ... ...
   |
   |
   |_ )


      (
   column ... ...
   list          ... ...
      )


      ... ...
)

FL[2] =
(
      (... ...)
      (... ...)
       ... ...
)
```

LIST FL[1] has a clear hierarchical structure. However, it is too large to handle. It is necessary to reduce its size.

4.2.2 Hierarchical Lists

The entities of a list are not necessarily lists. They can be pointers which point to lower level lists. A list name is actually such a pointer. By using pointers instead of lists wherever possible, we can separate a large list into many small ones.

```
FL[1] =
(
     (SUB            -- lower level lists
          (
            WALL[1][1]
            WALL[1][2]
            ... ...
            CLUM[1][1]
            CLUM[1][2]
            ... ...
          )
          ... ...
     )
     ... ...
```

```
)

Each variable is a list:

WALL[1][1] =
(
     (AutoCAD entities)
     (NAME   wall)
     (STAR PT  (0.0   0.0))
     (END PT   (10.0  0.0))
     ... ...
     (SUP   FL[1])        -- higher level list
     (SUB                 -- lower level list
          (
            OPNG[1][1]
            OPNG[1][6]
            ... ...
          )
     )
     ... ...
)

OPNG[1][1] =
(
     ... ...
```

)

In such a structure, the data are stored in many lists, which can be accessed from either higher or lower level list. List (SUP ...), and (SUB ...) are pointer lists linking the different levels in the hierarchy.

4.2.3 AutoCAD Entity Pointer

AutoCAD entities are basic elements composing building objects. In building component hierarchy, each object may have a set of points, which point to its composing entities.

For example, a wall is drawn with four lines. In the data list of this wall, there should be four pointers pointing to the four line-type entities in AutoCAD drawing database. Whenever this wall needs to be modified, the four entities can be called and modified. Entity names are the pointers which make entities accessible.

However, an entity may have different names during different editing phases. That means, entity name can not be used to link the drawing database and building component hierarchy.

Fortunately, AutoCAD provides another unique identifier, or entity handle to every entity in a

drawing. It is displayed in hexadecimal notation. An entity handle is an identifier permanently assigned to an entity throughout its lifetime. It can be used instead of entity name as entity pointer in the hierarchy. By using certain AutoLISP function, the entity name can be got only if its handle is known.

CHAPTER 5
GRAPHIC REPRESENTATION

5.1 General Issues

Graphic representations play important roles in the architectural design process. Most design messages are transmitted in term of drawings in architectural design. No matter how they are organized in the data base, building components have to be represented graphically.

There is no command in AutoCAD to open a hole in an object, while this object still exists. To draw an opening, therefore, is actually not to draw it, just to draw the objects enclosing it. In figure 5-1, the wall is divided into several segments. When those segments are created, the window will be there.

We can see, to draw a wall with a window, the system needs to know the coordinates of the start points and end points of the wall and the window. Also, it needs the width and some other properties of the wall and the window. By processing those data, the system gets the vertexes of each segment of the wall. Then it will draw them one by one. During this process, all the data, such as coordinates, width, and so on, should be given. Any change of the data will lead to re-doing the whole process and regenerating the whole drawing (figure 5-2). It will take a lot of time.

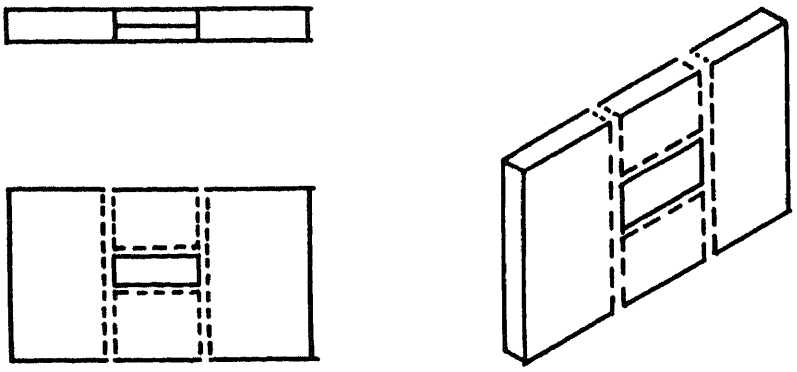


Figure 5-1 Draw an Opening With AutoCAD

Many changes have to be made to the design during architectural design process. All those changes should be responded quickly by the system.

It may be a good solution to provide different graphic representations at different design phases. Let's take a look at the way that architects design a project.

Usually, architects begin a design with very schematic sketch, then refine it, in step by step fashion. In early design phases, refined drawings are not necessary. Sketches, wireframe drawings are acceptable to architects (figure 5-3).

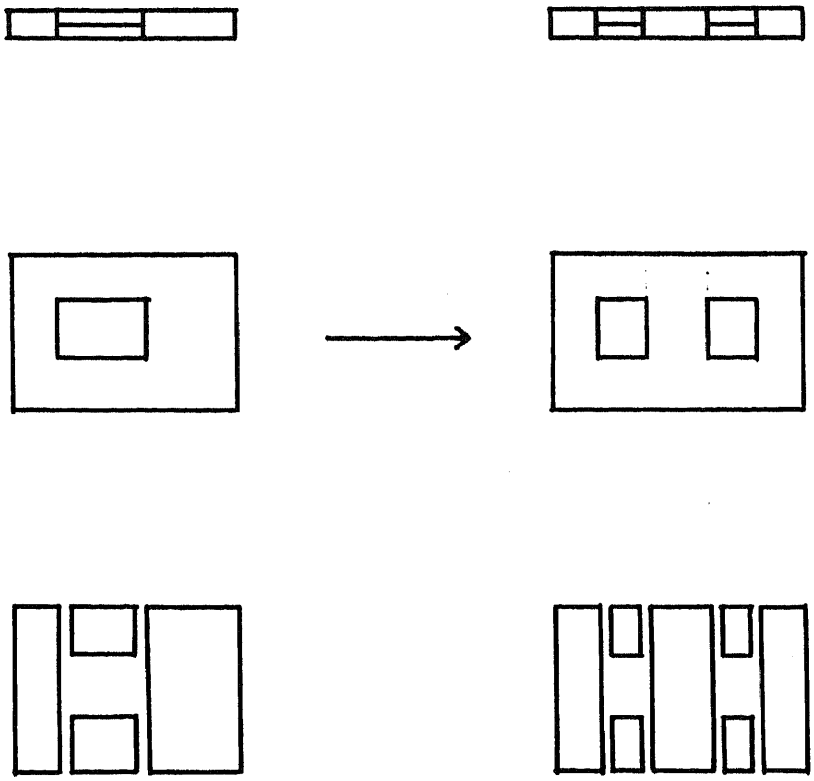


Figure 5-2 Openings in a Wall

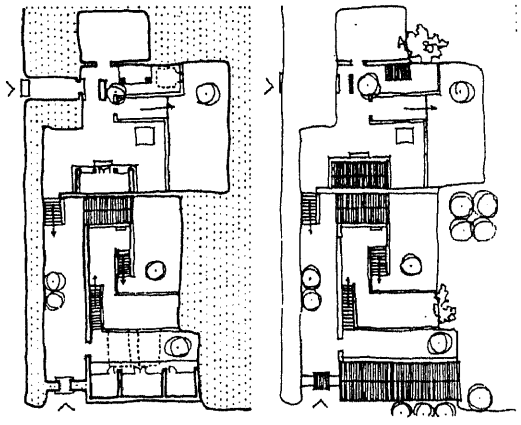


Figure 5-3 Representations in Early Design Phases

So, the following tools may be provided by the system:

DRAWING

- WALL
- OPENING
- COLUMN
- ... ...

3D-VIEWING

RENDERING

- 2D
- 3D

The tools under DRAWING generate 2D representations, such as plans. This kind of representations should be able to be generated and regenerated quickly. They should also reflect the object hierarchy of the data base, so that they can present what are in the data base. And they should allow the processing that needs architectural knowledge, such as checking the design for compliance with certain building code.

It is not necessary for such representations to be well rendered, because they are for the architects who

design the project, and are quite familiar with the project. Those architects could understand the representations very well even though they are sketches or wireframe drawings. What we are interested in is how to reflect objects in data base by those representations.

3D-VIEWING generates wireframe 3D images. It provides a time-saving way for users to visualize their design.

When a design is completed, or it comes to the end of a certain design phase, a set of well represented drawings are needed. Those drawings should be in the form of traditional drafting conventions, so that the information provided by the drawings can be understood by those who are not familiar with the project, such as clients, other architects, etc. RENDERING is the function to generate such drawings for final representation. Whenever it is called, it generates 2D or 3D drawings.

All those drawings generated for final representation are not related with the data base. They are only the 2D or 3D pictures of the objects. To change the design, the user has to go back to the modifying mode. Any change made to the drawings generated for final representation will not be written to the data base.

In this Chapter, we will discuss mainly 2D representations for the drawing/modifying phases.

5.2 Components Representations

In architectural drawings, especially in plans, floors, walls, and openings are the most important and basic items.

5.2.1 Floors

We need to draw floors only if there are changes in elevation on a floor. So floors are closely related with steps and stairs.

On AutoCAD, we may set the variable THICKNESS according to the changes in elevation, then use LINE to draw a polygon as a floor or a step. Latter, whatever we draw, check its position first, set the AutoCAD variable ELEVATION equal to the thickness of the floor on which this component is. In this case, user does not need to calculate the height of a component from the ground before drawing it. The height of a component is from the floor on which it is located. This is the way architects work. When we draw a wall, for example, we usually care for how high it is from the floor. We may even not know how high it is from ground.

5.2.2 Walls

In AutoCAD, if we draw a wall with LINE, we can only draw the outline of the wall. By using PLINE, we can draw the wall body at one time (figure 5-4).

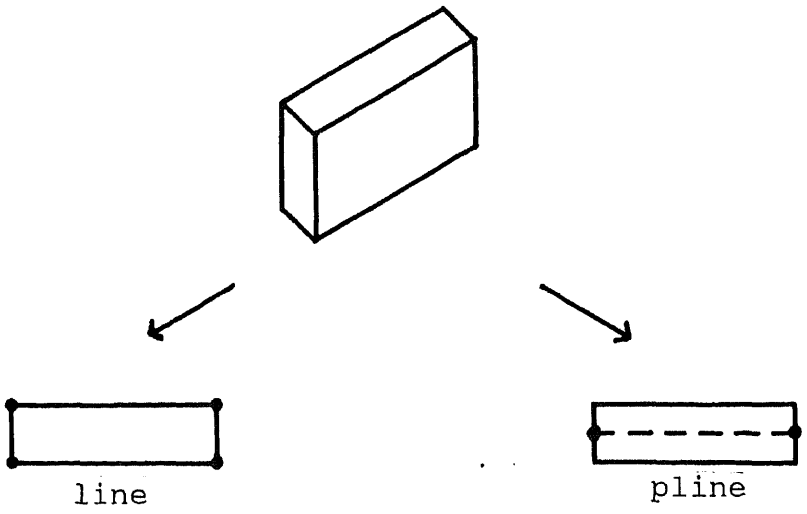


Figure 5-4 Difference Between LINE and PLINE

Because plines have width, they look like walls in both 2D and 3D. We select the pline as graphic representation of wall.

By using pline, we can draw continuous walls connected smoothly. However, those walls must be drawn at one time. If not, walls can not connect smoothly (figure 5-5).

It is unpractical to ask users to draw all continuous walls at one time. On the other hand, PLINE

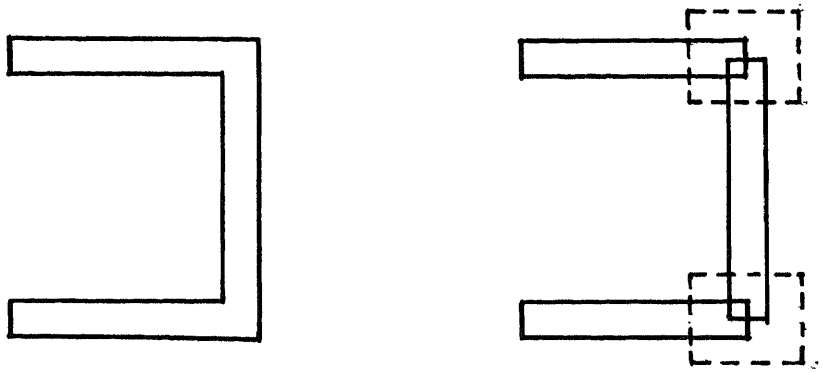


Figure 5-5 Connections of PLINEs

only generates one entity at a time. In figure 5-5(a), three walls are actually one entity. They only have one entity name. If one of the wall need to be deleted, we have to break this entity first, then delete it (figure 5-6).

In this case, the original entity does not exists any more. Instead, two new entities are generated. They have their own new entity names. It makes a lot of difficulties for entity tracing, because after each change, many data have to be corrected.

Avoiding such situation, we should only draw one segment each time with PLINE. Then, the system checks all

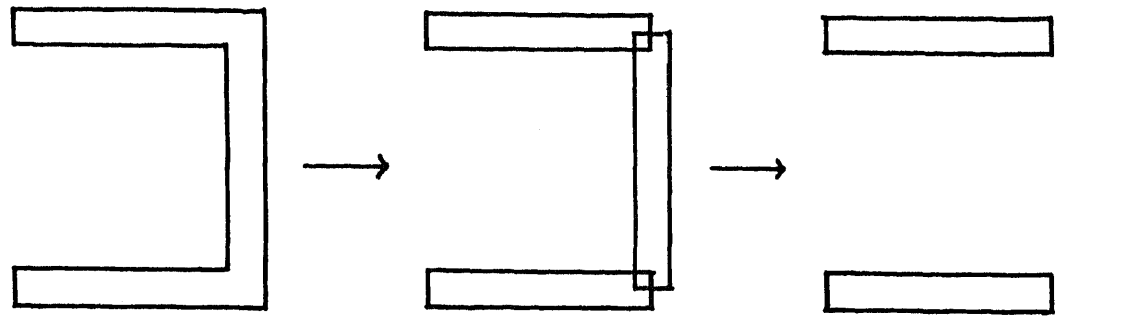


Figure 5-6 Break a PLINE

the walls, to find out those end to end connections, and extend the ends to make them connect smoothly.

However, such connections, as well as some others, are still not what we want. We may set FILL to "ON" to fill the interiors (figure 5-7).

5.2.3 Openings

Usually, a wall with a door is represented as figure 5-8. The wall is from p1 to p4, and the door is from p2 to p3.

In the plan, the wall is displayed as two segments, p1p2, p3p4. If we draw p1p2, p3p4 with PLINE

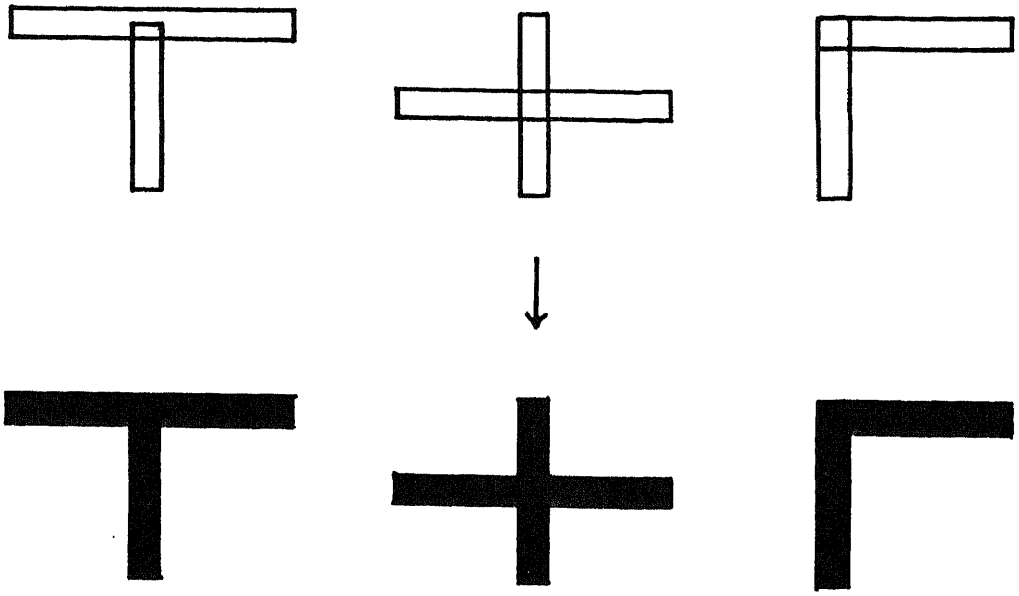


Figure 5-7 FILL and Connections

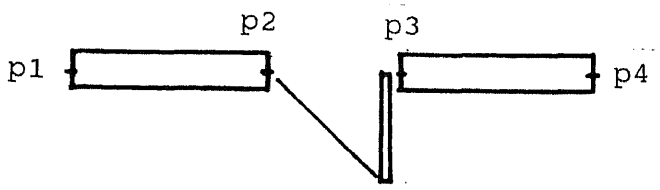
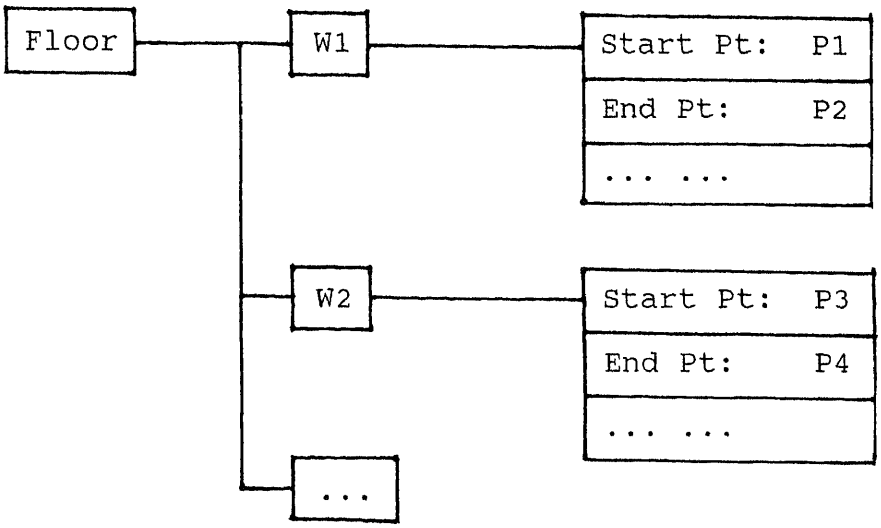


Figure 5-8 A Wall With a Door

respectively, they will be treated as two walls in data base:



What we want is that the wall from point P1 to point P4 is one entity, but looks broken from point P2 to point P3.

To draw such a wall, we can draw a solid wall p1p4 with PLINE first (figure 5-9 a), then draw the door p2p3 still with PLINE (figure 5-9 b). So far, we have two entities for wall p1p4 and door p2p3. Next, we set the redraw mode to 2, redraw door p2p3 with AutoLISP function REDRAW. In this mode, an entity is invisible although it still exists (figure 5-9 c). Door p2p3 looks erased. And wall p1p4 looks broken while it is still a single entity. At last, draw the leaf of the door (figure 5-9 d).

It is similar to draw a window.

Doors and windows may need to be represented in details. We can create a block for each type of

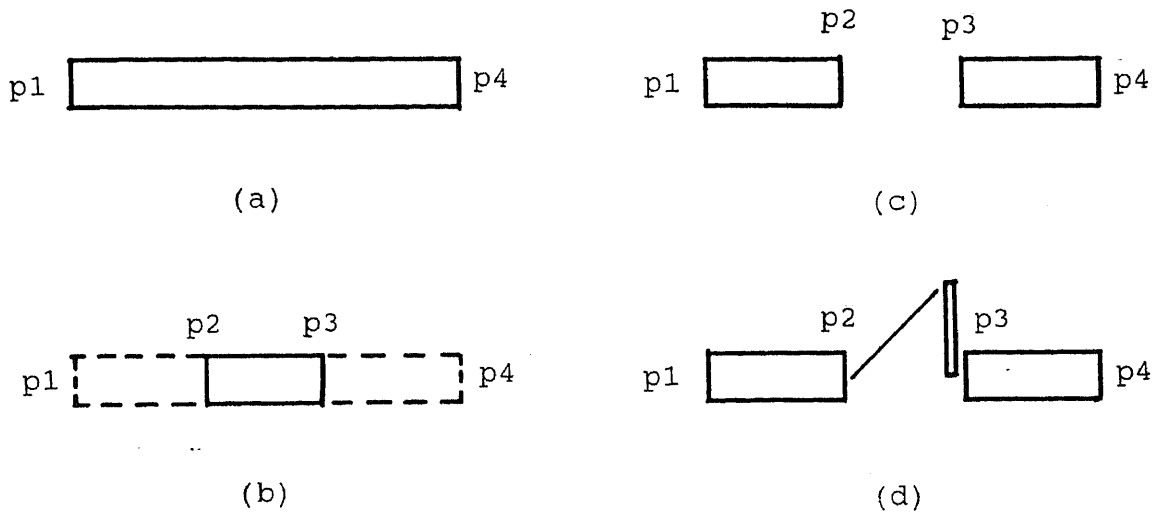


Figure 5-9 Door Representation

subcomponents in details, such as windows, doors, and columns. Users only need to input the position and the name of type, the scaled block will be inserted automatically.

To draw a subcomponent, it is necessary to define under which component it is. For example, to add a window or stud, we have to select a wall first. So the component could be put to the correct position in the hierarchy in the data base. If we erase the wall latter, the window or stud will be found out by the system, and removed from the data base.

5.2.4 Beams, Ceilings

Beams, ceilings are not always presented in plan. We can set a layer for them. Whenever asked by user, they can be displayed with dash lines.

5.3 Multi-Level Representation

Details of architectural drawings vary with the scale. The system should be able to choose different representations according to the scale of the drawing.

Because the objects in the data base are hierarchical, we can use a variable LEVL to indicate the position of the level in hierarchy. With given scale, the system calculates at which level the representation should be. If a drawing is too small, it may only have walls (figure 5-10).

For multi-level representation, we can draw those components on separate layers. By controlling the layers, present different level (figure 5-11).

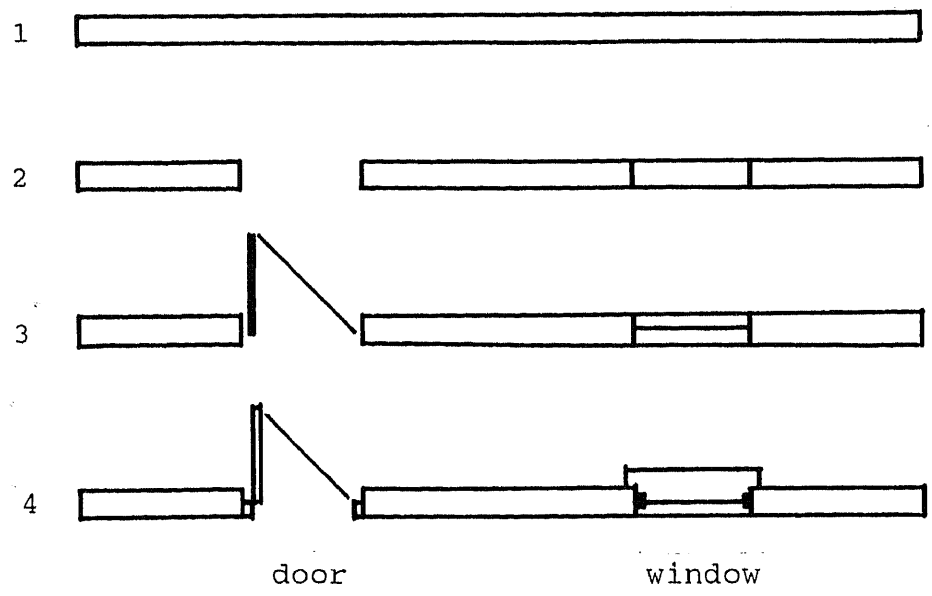


Figure 5-10 Multi-level Representations

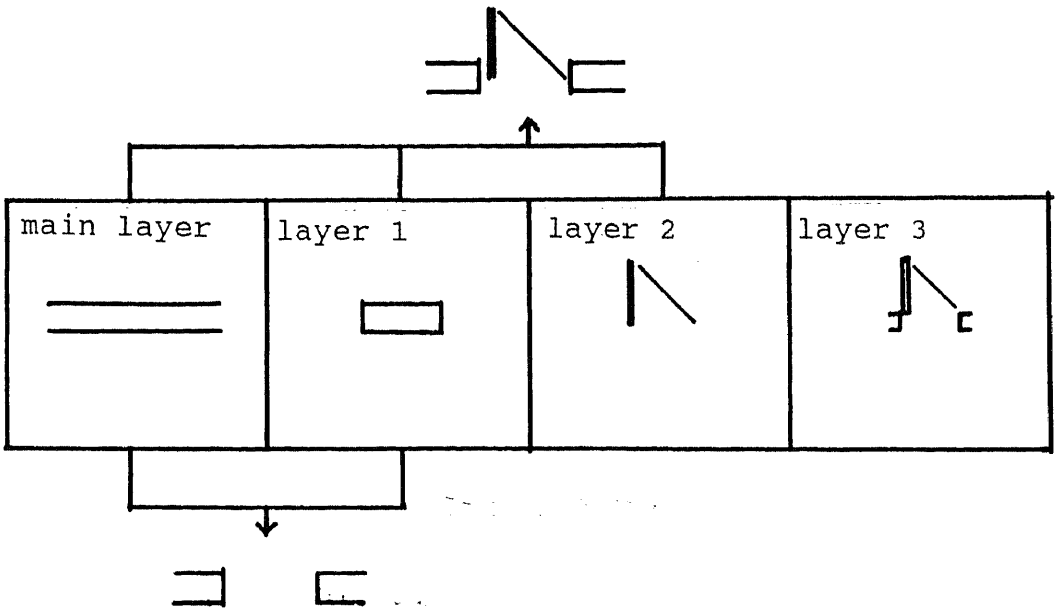


Figure 5-11 Layers and Representations

CHAPTER 6
DEFINING SPACES

Before design evaluation or any code checking, the functional element hierarchy has to be created. To define a space, we may select a set of walls as the boundaries of the space. After calculation, the system can get a polygon as the space (figure 6-1).

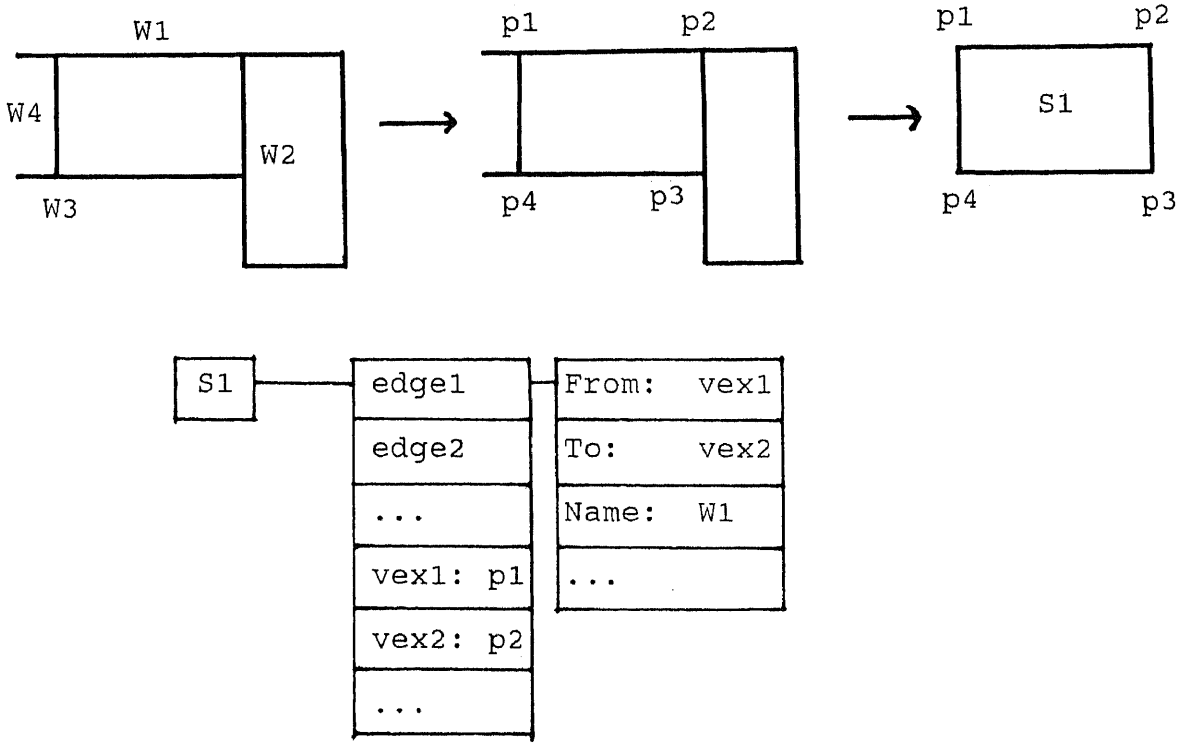


Figure 6-1 Spaces Defining

At the same time, we input the attributes of the space, such as what the occupant load is, if it is an exit or not.

In some cases, spaces are not divided by walls. We need to use assistant lines. Therefore, it is possible to define subspaces within a large room, or an area which includes several rooms (figure 6-2).

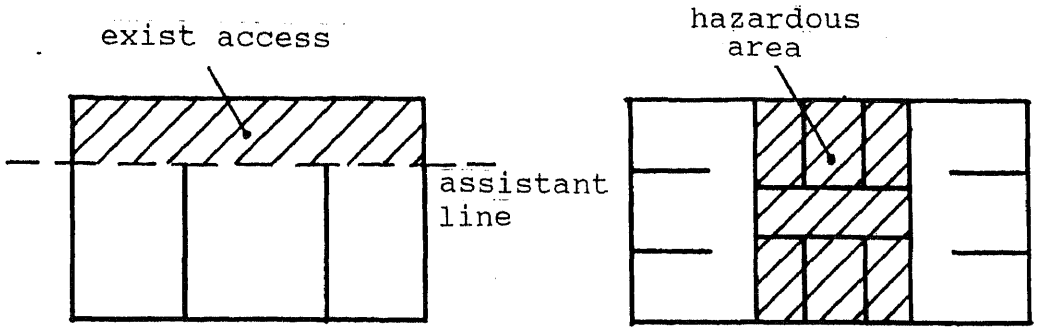


Figure 6-2 Special Spaces Defining

Because we do not have duplicated boundaries in the data base, it is impossible to generate spaces automatically by the system. The user has to select boundaries for each space.

CHAPTER 7
CONCLUSION

Architectural knowledge is needed in decision-making processes in architectural design. The object-oriented hierarchical data structure proposed in this study can be linked with an interpreted architectural knowledge base, to make the use of computers in theses processes possible. All design objects are stored as they are in the real world. The knowledge based on those design objects can be used.

Usually, building components, instead of lines and points, are used for representations in architectural design. An Architectural Expert System enables its user to use these components in design. It provides object representations for many building components, such as walls, openings, columns, etc., rather than for lines, points, and surfaces. However, object representations designed in this study are limited. It is impossible to provide representation for every building component. With the development of industries, there will be new building materials and components. Therefore, the next step of this study should provide tools for users to compose their own objects and/or graphic representations. The current system allows users to use their own graphic representations instead of the symbolical

representations, and to change the position of any object in the hierarchies. These functions are useful not only for creating new object, but also for some unusual designs. For example, a side of a space may be a big window, from floor to ceiling. There is no wall on this side. In the data hierarchy, an opening is under a wall. We can use "DETACH" and "ATTACH" functions to change its position to the same level as walls. Such functions, with new-object-composing function, will enable the system to be more flexible.

APPENDIX A
MENU FILE

The blank lines at the end of a menu serve to erase items of longer submenu.

```
*** screen
[ArchCAD]$S=SCREEN
[--------]
[HELP]

[SET FLR]$S=FLOOR FLOOR
[DRAW]$S=DRAW
[MODIFY]$S=MODIFY
[TOOLS]
[DISPLAY]$S=DISPLAY
[OPTION]$S=OPTION OPTION
[DATA LST]$S=LK LK

[SPACE]$S=SPACE

[RENDER]$S=RENDER
[PLOT]

[SAVE]SAVE
[END]END
[GIVE UP]QUIT


**FLOOR 3
[SET FLR]FLOOR

[NEW]NEW
[CHANGE]

[ON]
[OFF]

[  LAST]$S=


**DISPLAY 3
[DISPLAY]

[RM ALINE]CLEAN
[REDRAW]REDRAW
[ZOOM]

[  LAST]$S=
```

```
**DRAW 3
[DRAW]

[WALL]$S=WALL WALL
[OPENING]$S=OPENING OPENING
[COLUMN]$S=COLUMN COLUMN
[STUD]
[STAIRS]
[BEAM]
[CEILING]

[AST-LINE]$S=ALINE ALINE

[OTHER]

[  LAST]$S=


**WALL 3
[WALL]WALL

[WIDTH]W
[HEIGHT]H
[CLOSE]C


**OPENING 3
[OPENING]OPENING

[SEL WALL]S
[DOOR]D $S=OPN2
[WINDOW]W $S=OPN2


**OPN2 4

[2ND PT]S
[HEIGHT]H
[WIDTH]


**COLUMN 3
[COLUMN]COLUMN

[CIRCLE]$S=CIRCLE C
[RECTANG]R
[SQUARE]S


**CIRCLE 4
```

```
[DIAMETER]D
[RADIUS]


**ALINE 3
[ALINE]ALINE

[END]END
[NEAR]NEA
[INTER]INT
[PERPEN]PER
[TANGENT]TAN


**SPACE 3
[SPACE]

[CREATE]SCREATE
[MODIFY]

[  LAST]$S=


**RENDER 3
[RENDER]

[2D]
[3D]

[  LAST]^C$S=


**MODIFY 3
[MODIFY]

[ATTACH]$S=TACH ATTACH
[DETACH]$S=TACH DETACH

[COPY]
[MOVE]$S=MOVE MV
[ROTATE]
[DELETE]DELETE

[  LAST]$S=


**TACH 3

[REMOVE]R
[ADD]A
```

```
[DONE]


**MOVE 3
[MOVE]MV

[BASE PT]B
[XY DIST]XY
[2ND PT]


**OPTION 3
[OPTION]OPTION

[1]1
[2]2
[3]3
[4]4

[  LAST]$S=


**LK 3
[DATA LST]LK

[SELECT]S
[NAME]N

[WALL]W
[OPENING]O
[COLUMN]C
[ASST-L]A

[  LAST]$S=


*** pop1
[Draw]

*** pop2
[Modify]

*** pop3
[Display]

*** pop4
[Tools]

*** pop5
[Options]
```

```
*** pop6
[Floors]

*** pop7
[Files]

*** pop8
[Render]

*** pop9
[Help]
```

APPENDIX B
PROGRAM

```
;START UP
;============================================================
(defun S::STARTUP ( / tmpfile mlayer olayer copycmd)
;change environment variables, set new layers and
;variables
;this function will be executed automatically

   (setq oldbl (getvar "BLIPMODE")
         oldcm (getvar "CMDECHO")
         oldth (getvar "THICKNESS")
         oldfl (getvar "FILLMODE")
   )

   (setvar "CMDECHO"  0)
   (setvar "FILLMODE" 1)
   (setvar "BLIPMODE" 0)

;change AutoCAD commands
   (command "handles" "on"
            "undefine" "end"
            "undefine" "redraw"
            "undefine" "save"
            "undefine" "quit"
   )

   (setq realfile  (getvar "DWGNAME")
         file (strcat realfile "$")  ;temporary file
   )

   (if (not (equal (substr (getvar "CLAYER") 1 3) "FL-"))
;it is a new project
      (progn
        (setq fln 1
              waln[1] 0
              opngn[1] 0
              clumn[1] 0
              spcn[1] 0
              FLLST (list fln)

              dlevl 3
              wlevl 3
        )
        (command "layer" "m" "FL-1"
                 "new" "WDTL-1-1,WDTL-1-2,WDTL-1-3,
                  DDTL-1-1,DDTL-1-2" ""
        )
      )
```

```
      (progn
;it is an existing project
        (setq copycmd (strcat "copy " realfile ".* "
                              file ".*"
                      )
        )
;copy all files to temporary files
        (command "shell" copycmd)

        (setq tmpfile (strcat file ".BAC"))
        (READFILE tmpfile)

        (setq tmpfile (strcat file ".C" (itoa fln)))
        (READFILE tmpfile)

        (setq tmpfile (strcat file ".S" (itoa fln)))
        (READFILE tmpfile)
      )
   )
   (setq waln  (VALU "waln[fln]")
         opngn (VALU "opngn[fln]")
         clumn (VALU "clumn[fln]")
         spcn (VALU "spcn[fln]")

         dtl     (strcat "DTL-" (itoa fln))
         wdtl[1] (strcat "W" dtl "-1")
         wdtl[2] (strcat "W" dtl "-2")
         wdtl[3] (strcat "W" dtl "-3")
         wdtl[4] (strcat "W" dtl "-4")

         ddtl[1] (strcat "D" dtl "-1")
         ddtl[2] (strcat "D" dtl "-2")
         ddtl[3] (strcat "D" dtl "-3")
   )
   (setq mlayer (strcat "FL-" (itoa fln))
         olayer (getvar "CLAYER")
   )
   (if (not (equal mlayer olayer))
       (command "layer" "t" mlayer "s" mlayer "f" olayer
"")
   )
   (CHLEVEL)
   (princ)
);end of STARTUP


;WALLS
;============================================================
(defun c:WALL ( / spt ept sptx higt)
;draw walls
```

```
   (setq spt (getpoint "\nEnter start point: ")
         sptx spt
         ept 0
   )

   (if (not (equal spt nil))
     (progn
        (if (equal wdth nil)
            (setq wdth 0)
        )
        (if (equal higt nil)
            (setq higt 0)
        )

        (while (and (not (equal ept nil))
                    (not (equal ept sptx))
               )
               (WALMOR)
        )
     )
   )
   (princ)
) ;end of WALL
;---------------------------------------

(defun WALMOR ()
;continue to draw walls

   (initget "Width Height Close-wall")
   (setq ept (getpoint spt
               "\nWidth/Height/Close-wall/<To point:>"))
   (if (equal ept "Width")
       (WALWDTH)
   )
   (if (equal ept "Height")
       (WALHIGT)
   )
   (if (equal ept "Close-wall")
       (setq ept sptx)
    )
   (if (and (not (equal ept "Width"))
            (not (equal ept "Height"))
            (not (equal ept nil))
       )
       (progn
        (setvar "THICKNESS" higt)
        (setq spt (polar spt (angle ept spt) (/ wdth 2)))
        (command "pline" spt "w" wdth wdth ept "")
        (WALSAVE)
        (setq spt ept)
```

```
        )
    )
);end of WALMOR
;---------------------------------------

(defun WALSAVE (/ cwal hdl ele1 ele2 ele3 ele4 ele5 ele6
                  ele7 ele8 tmp oldwal newwal)
;save wall data

;get a name for the new wall
    (if (null (valu "WALEMP[fln]"))
        (setq waln (1+ waln)
              cwal (var "WAL[fln][waln]")
        )
        (progn
           (setq cwal (car (valu "WALEMP[fln]")))
           (set (var "WALEMP[fln]")
                (rmx  cwal (valu "WALEMP[fln]"))
           )
        )
    )
    (setq hdl  (cdr (assoc 5 (entget (entlast))))
          ele1 (list 'DWG hdl)                ;AutoCAD
entity
          ele2 (list 'NAME "Wall")            ;type name
          ele3 (list 'ROOT "WAL" fln)         ;variable name
          ele4 (list 'SUP (var "FL[fln]"))    ;parent object
          ele5 (cons 'SPT spt)                ;start point
          ele6 (cons 'EPT ept)                ;end point
          ele7 (list 'WIDTH wdth)             ;width
          ele8 (list 'HEIGHT higt)            ;height

          tmp  (list (list hdl cwal))

          oldwal (assoc 'SUB (valu "FL[fln]"))
    )
    (set  cwal (list ele1 ele2 ele3 ele4
                     ele5 ele6 ele7 ele8
               )
    )
;put its name in index which is used to find an object
;from its handle
    (set (var "WALLST[fln]")
         (append (valu "WALLST[fln]") tmp)
    )
    (if (equal oldwal nil)
        (progn
           (setq newwal (list (list 'SUB cwal)))
           (set  (var "FL[fln]")
                 (append (valu "FL[fln]") newwal)
           )
```

```
        )
        (progn
           (setq newwal (append oldwal (list cwal)))
           (set  (var "FL[fln]")
                 (subst newwal oldwal (valu "FL[fln]"))
           )
        )
    )
);end of WALSAVE
;---------------------------------------

(defun WALWDTH ( / init)
;get wall width

    (setq init (getdist (strcat "\nWall width <"
                                (rtos wdth) ">: "
                        )
               )
    )
    (if (not (equal init nil))
        (setq wdth init)
    )
);end of WALFST
;---------------------------------------

(defun WALHIGT ( / init)
;get wall height

    (setq init (getdist (strcat "\nWall height <"
                                (rtos higt) ">: "
                        )
               )
    )
    (if (not (equal init nil))
        (setq higt init)
    )
);end of WALHIGT


;OPENINGS
;============================================================
(defun c:OPENING ( / copng cmp cmplst wdspt wdept wdist
                     whigt1 whigt2 cwal wdth ang dw spt ept)
;draw openings

    (setq dw 0)
    (GETWAL)

    (while (and (boundp 'dw)
                (boundp 'cwal)
        )
```

```
        (initget "Select-wall Door Window")
        (setq dw (getpoint "\nSelect-wall/Door/Window: "))
        (if (equal dw "Select-wall")
            (GETWAL)
        )
        (if (or (equal dw "Door")
                (equal dw "Window")
            )
            (OPNGDRW)
        )
    )
    (princ)
); end of OPENING
;---------------------------------------

(defun OPNGSAVE (/ oldopng ele1 ele2 ele3 ele4
                   ele5 ele6 ele7 ele8 tmp newopng)
;save openings into data list

    (setq oldopng (assoc 'SUB (eval cwal))
          ele1 (cons 'DWG cmp)           ;AutoCAD entity
          ele2 (list 'NAME dw)           ;type name
          ele3 (list 'ROOT "OPNG" fln)   ;varible name
          ele4 (list 'SUP cwal)          ;higher level
          ele5 (cons 'SPT wdspt)         ;start point
          ele6 (cons 'EPT wdept)         ;end point
          ele7 (list 'HEIGHT1 whigt1)    ;height
          ele8 (list 'HEIGHT2 whigt2)    ;height
          tmp  (list ele1 ele2 ele3 ele4
                     ele5 ele6 ele7 ele8
               )
    )
    (set  copng tmp)
    (set  (var "OPNGLST[fln]")   ;put into opening index
          (append (valu "OPNGLST[fln]") cmplst)
    )
;process higher level object
    (if   (equal oldopng nil)
          (set   cwal  (append (eval cwal)
                               (list (list 'SUB copng))
                       )
          )
          (progn
             (setq newopng (append oldopng (list copng)))
             (set  cwal
                   (subst newopng oldopng (eval cwal))
             )
          )
    )
);end of OPNGSAVE
;---------------------------------------
```

```
(defun GETWAL (/ lst name en1 en2 tmp)
;select a wall in which to draw a opening

   (setq lst (OBJSEL "\nSelect a wall: ")
         cwal (car lst)
         name (cadr lst)
   )
   (while (or (not (equal name "Wall"))
              (equal cwal nil)
          );check if the selected obj is a wall
       (setq lst (OBJSEL "\nNo wall selected.")
             cwal (car lst)
             name (cadr lst)
       )
   )

   (setq spt   (cdr (assoc 'SPT (eval cwal)))
         ept   (cdr (assoc 'EPT (eval cwal)))
         wdth  (cadr (assoc 'WIDTH (eval cwal)))
         ang   (min (angle spt ept)
                    (angle ept spt)
               )
   );get information useful for opening
   (if (> (angle spt ept) (angle ept spt))
       (setq tmp spt
             spt ept
             ept tmp
       )
   )
);end of GETWAL
;---------------------------------------

(defun OPNGDRW (/ wp cmp1 cmp2 cmp3 cmp4)
;draw openings

   (setvar "BLIPMODE" oldbl)
   (initget 1)
   (setq wdspt (getpoint spt "\nStart point: ")
         wdspt (inters spt ept wdspt
                       (polar wdspt (+ ang (/ pi 2)) 2) nil
               )
         wdist  "Height"
   )
   (while (equal wdist "Height")
       (initget "Second-point Height")
       (if (equal dw "Window")
           (setq  wdist (getdist wdspt "\nSecond-point
                                 /Height<Window width:>"))
           (setq  wdist (getdist wdspt "\nSecond-point
                                 /Height<Door width:>"))
```

```
        )
        (if (equal wdist "Height")
            (progn
             (initget 1)
             (if (equal dw "Window")
                 (progn
                    (setq whigt1
                         (getdist "\nSill height: ")
                    )
                    (initget 1)
                    (setq whigt2
                         (getdist "\nWindow height: ")
                    )
                 )
                 (progn
                    (setq whigt1 0)
                    (setq whigt2
                         (getdist "\nDoor height: ")
                    )
                 )
             )
            )
        )
    )

    (if (and
           (not (equal wdist "Second-point"))
           (not (equal wdist nil))
        )
        (setq wdept (polar wdspt ang wdist))
    )

    (if (equal wdist "Second-point")
        (progn
           (initget 1)
           (setq wdept (getpoint wdspt "\nSecond-point: ")
                 wdept (inters spt ept wdept
                          (polar wdept (+ ang (/ pi 2)) 2)
                           nil
                       )
           )
        )
;check which one is closer to spt
           (if (< (distance spt wdept)
                  (distance spt wdspt)
               )
               (setq wp wdspt
                     wdspt wdept
                     wdept wp
               )
            )
            (setq    wdist (distance wdspt wdept))
```

```
        )
    );end of if

    (if (boundp 'wdist)
      (progn
        (if (null (valu "OPNGEMP[fln]"))
            (setq opngn (1+ opngn)
                  copng (var "OPNG[fln][opngn]")
            )
            (progn
                (setq copng (car (valu "OPNGEMP[fln]")))
                (set (var "OPNGEMP[fln]")
                     (rmx copng (valu "OPNGEMP[fln]"))
                )
            )
        )

        (setvar "BLIPMODE" 0)
        (command "pline" wdspt "w" wdth wdth wdept "")
        (setq cmp1 (entlast))

        (if (equal dw "Window")
         (progn
             (command "change" cmp1 "" "p"
                      "la" wdtl[1] ""
             )
             (redraw (entlast) 2)
             (command "line" (polar wdspt (+ ang (/ pi 2))
                                    (/ wdth 2)
                             )
                             (polar wdept (+ ang (/ pi 2))
                                    (/ wdth 2)
                             )
             "")
             (setq cmp2 (entlast))
             (command "line" (polar wdspt (- ang (/ pi 2))
                                    (/ wdth 2)
                             )
                             (polar wdept (- ang (/ pi 2))
                                    (/ wdth 2)
                             )
             "")
             (setq cmp3 (entlast))
             (command "line" wdspt wdept "")
             (setq cmp4 (entlast))
;change the entities'layer for multi-level representation
             (command "change" cmp2 cmp3 ""
                               "p" "la" wdtl[2] ""
                      "change" cmp4 "" "p" "la" wdtl[3] ""
             )
;get index data
```

```
            (setq cmp1 (cdr (assoc 5 (entget cmp1)))
                  cmp2 (cdr (assoc 5 (entget cmp2)))
                  cmp3 (cdr (assoc 5 (entget cmp3)))
                  cmp4 (cdr (assoc 5 (entget cmp4)))
                  cmp  (list cmp1 cmp2 cmp3 cmp4)
                  cmplst (list (list cmp1 copng)
                               (list cmp2 copng)
                               (list cmp3 copng)
                               (list cmp4 copng)
                         )
            )
            (OPNGSAVE)
         )
        )
        (if (equal dw "Door")
         (progn
            (command "change" cmp1 "" "p" "la" ddtl[1]
            "")
            (redraw (entlast) 2)
            (LEAFDRW)

            (setq cmp1 (cdr (assoc 5 (entget cmp1)))
                  cmp  (list cmp1)
                  cmplst (list (list cmp1 copng))
            )
            (if (boundp 'cmp2)
                (setq cmp (append cmp (list cmp2))
                      cmplst (append cmplst
                                     (list (list cmp2 copng)))
                )
            )
            (OPNGSAVE)
         )
        )
      )
   )
);end of OPNGDRW
;---------------------------------------

(defun LEAFDRW (/ lf basept tmppt blk xx yy)
;draw a door leaf

   (setq lf (GMNUSEL "_door" 8)) ;select a leaf

;Set base point and xx -- x scal factor
   (if (or (equal lf 1)
           (equal lf 6)
       )
       (progn
           (setq tmppt (polar wdspt ang (/ wdist 2)))
           (while (null basept)
```

```
                    (setq basept (getpoint tmppt
                          "Select the side the leaf is: "))
               )
               (if (< (distance basept wdspt)
                      (distance basept wdept)
                   )
                   (setq basept wdspt
                         xx     1
                   )
                   (setq basept wdept
                         xx     -1
                   )
               )
          )
          (setq basept wdspt
                xx 1
          )
     )

;Set yy -- y scale factor
     (if (and (numberp lf)
              (< lf 4)
          )
          (setq yy (YYGET "\nDirection the leaf swings: ")))
     (if (or (equal lf 4)
             (equal lf 5)
             (equal lf 8)
          )
          (setq yy 1)
     )
     (if (or (equal lf 6)
             (equal lf 7)
          )
          (setq yy (YYGET "\nSelect the side: "))
     )

;Insert block
     (if (numberp lf)
          (progn
             (setq blk (strcat "graphmnu/_door_" (itoa lf)))
             (command "insert" blk basept  (* xx wdist)
                      (* yy wdist) (* (/ ang pi) 180)
             )
             (setq cmp2 (entlast))
             (command "change" cmp2 "" "p" "la" ddtl[2] "")
             (setq cmp2 (cdr (assoc 5 (entget cmp2))))
          )
     )
); end of LEAFDRW
;---------------------------------------
```

```
(defun YYGET ( msg / blkang tt)
;calculate y scale factor

   (while (null blkang)
          (setq blkang (getangle basept msg))
   )
   (setq tt (- blkang ang))
   (if (or (and (>= tt 0)
                (<= tt pi)
           )
           (< tt (* -1 pi))
       )
       (setq yy 1)
       (setq yy -1)
   )
);end of YYGET


;COLUMNS
;============================================================
(defun c:COLUMN (/ basept tmp tmpx tmpy tmpang cmp)
;draw columns

   (setq basept (getpoint "\nEnter the center point: "))
   (if (boundp 'basept)
       (progn
       (princ "\nCircle/Rectangular/Square/<")
;prompt the data of the last column
       (if (equal clumlast "Circle")
           (progn
              (princ "Radius ")
              (princ clumrad)
           )
           (if (boundp 'clumlast)
               (progn
                  (princ "Side ")
                  (princ clumx)
                  (princ " ")
                  (princ clumy)
                  (princ "; Angle ")
                  (princ (angtos clumang))
               )
           )
       )
       (princ ">: ")

       (initget "Circle Rectangular Square")
       (setq tmp (getkword))

       (cond
          ( (equal tmp "Circle")
```

```
;draw circle columns
            (initget "Diameter")
            (setq clumrad (getdist basept
                    "\nDiameter/<Radius>: "))
            (if (equal clumrad "Diameter")
              (progn
                (setq clumrad (getdist "\nDiameter: "))
                (if (boundp 'clumrad)
                    (setq clumrad (/ clumrad 2))
                )
              )
            )
            (if (boundp 'clumrad)
;if the user hit <RETURN>, cancel the function
                (CCLUM)
            )
          )
;draw rectangular columns
          ( (equal tmp "Rectangular")
            (setq tmpx (getdist "\nX-side length: ")
                  tmpy (getdist "\nY-side length: ")
            )
            (if (and (boundp 'tmpx)
                     (boundp 'tmpy)
                )
              (progn
                (setq clumx tmpx
                      clumy tmpy
                )
                (setq tmpang (getangle "\nAngle <0>: "))
                (if (null tmpang)
                    (setq clumang 0)
                    (setq clumang tmpang)
                )
                (RCLUM)
              )
            )
          )
;draw square columns
          ( (equal tmp "Square")
            (setq tmpx (getdist "\nSide length: "))
            (if (boundp 'tmpx)
              (progn
                (setq clumx tmpx
                      clumy tmpx
                )
                (setq tmpang (getangle "\nAngle <0>: "))
                (if (null tmpang)
                    (setq clumang 0)
                    (setq clumang tmpang)
                )
```

```
                (RCLUM)
              )
            )
          )
;draw columns as the last one
          ( (boundp 'clumlast)
            (cond ((equal clumlast "Circle")
                   (CCLUM)
                  )
                  (T (RCLUM))
            )
          )
        );end of cond
        );end of progn
   )
   (princ)
);end of COLUMN
;---------------------------------------

(defun RCLUM (/ spt cmp1 cmp2 cmp3)
;draw rectangular or square columns

   (setq spt (polar basept (+ clumang pi) (/ clumx 2)))

   (command "pline" spt "w" clumy ""
            (polar spt clumang clumx) "")
   (setq cmp1 (list 'X-SIDE clumx)
         cmp2 (list 'Y-SIDE clumy)
         cmp3 (list 'ANGLE clumang)
         cmp (list cmp1 cmp2 cmp3)
   )

   (CLUMSAVE)
);end of RCLUM
;---------------------------------------

(defun CCLUM (/ spt)
;draw circle columns

   (setq spt (polar basept 0 (/ clumrad 2)))

   (command "pline" spt "w" clumrad ""  "a" "ce"
            basept (polar spt 180 clumrad) spt ""
   )
   (setq cmp (list (list 'RADIUS clumrad)))

   (CLUMSAVE)
);end of CCLUM
;---------------------------------------

(defun CLUMSAVE (/ tmpclum hdl ele1 ele2 ele3 ele4 ele5
```

```
                    indx oldclum newclum)
;save the column data

;save this column as the new "last column", so that the
;user could draw the same column without reinputting the
;same data
   (if (boundp 'tmp)
       (setq clumlast tmp)
   )

;check if there is any deleted column, if yes, use its
;name for the new column, if not, give a new name
   (if (null (valu "CLUMEMP[fln]"))
      (setq clumn (1+ clumn)
            tmpclum (var "CLUM[fln][clumn]")
      )
      (progn
         (setq tmpclum (car (valu "CLUMEMP[fln]")))
         (set (var "CLUMEMP[fln]")
              (rmx  tmpclum (valu "CLUMEMP[fln]"))
         )
      )
   )
   (setq hdl  (cdr (assoc 5 (entget (entlast))))
         ele1 (list 'DWG hdl)
         ele2 (list 'NAME "Column")
         ele3 (list 'ROOT "CLUM" fln)
         ele4 (list 'SUP (var "FL[fln]"))
         ele5 (cons 'SPT basept)

         indx  (list (list hdl tmpclum))  ;for index

         oldclum (assoc 'SUB (valu "FL[fln]"))
   )
   (set  tmpclum (append (list ele1 ele2 ele3 ele4 ele5)
                         cmp
                 )
   )
   (set (var "CLUMLST[fln]")
        (append (valu "CLUMLST[fln]") indx)
   )
;put a pointer in its higher level object
   (if (null oldclum)
      (progn
         (setq newclum (list (list 'SUB tmpclum)))
         (set (var "FL[fln]")
              (append (valu "FL[fln]") newclum)
         )
      )
      (progn
         (setq newclum (append oldclum (list tmpclum)))
```

```
        (set   (var "FL[fln]")
               (subst newclum oldclum (valu "FL[fln]"))
        )
      )
   )
) ;end of CLUMSAVE


;ASSISTANT LINES
;============================================================
(defun c:ALINE (/ tmplin cly cclr spt ept)
;draw assistant lines

   (setq cly (getvar "clayer")
         cclr (getvar "cecolor")
   );change layer and color

   (if (null aln)
       (setq aln 0)
   )
   (command "layer" "m" "0" "")
   (command "color" "blue")

   (princ "\nDraw assistant line(s)
               for open boundary(s).")
   (setq spt
         (getpoint "\nEND/NEAr/INTer/PERpen/TANgent
                    <Start point>: ")
   )
   (while (boundp 'spt)
       (setq ept
             (getpoint spt "\nEND/NEAr/INTer
                       /PERpen/TANgent <To point>: ")
       )
       (if (boundp 'ept)
           (progn
               (command "line" spt ept "")
               (setq aln (1+ aln)
                     tmplin (VAR "ALIN[aln]")
               )
               (set tmplin (list (list 'NAME "Asst-line")
                                 (cons 'SPT spt)
                                 (cons 'EPT ept)
                           )
               )
               (setq hdl
                    (cdr (assoc 5 (entget (entlast))))
                     ALINLST (append ALINLST
                                (list (list hdl tmplin))
                             )
               )
```

```
            )
        )
        (setq spt ept)
    )
;change the layer and color back
    (command "layer" "m" cly "")
    (command "color" cclr)
    (princ "\n(To remove all asst-lines, type \"CLEAN\")")
    (princ)
);end of ALINE


;SPACES
;============================================================
(defun c:SCREATE (/ vn en mesg1 mesg2 stop lst obj1 obj2
                  objx name1 name2 namex spt1 spt2 sptx
                  ept1 ept2 eptx done i v tmp e du obj name)
;create spaces

    (setq vn 0
       en 0
       mesg1 "\nSelect boundary: "
       mesg2 "\nSelect boundary: "
    )
    (while (and (null stop)
                (null EDGE[1])
           )
        (setq lst (OBJSEL mesg1))
        (if (null lst)
            (setq stop "y")
            (progn
              (setq obj1 (car lst)
                    du nil
              )
              (DUCHECK obj1)
              (if (boundp 'du)
                (setq mesg1 "\nDuplicated.
                                Select again: ")
                (progn
                   (setq name1 (cadr lst)
                         spt1
                         (cdr (assoc 'SPT (eval obj1)))
                         ept1
                         (cdr (assoc 'EPT (eval obj1)))
                   )
                   (if (null ept1)
                       (VSAVE1)
                       (ESAVE1)
                   )
                )
              )
```

```
        )
    )
)
(while (and (null stop)
            (null done)
       )
    (setq lst (OBJSEL mesg2)
          du  nil
    )
    (if (null lst)
        (setq done "Y"
              obj2 objx
              name2 namex
              spt2 sptx
              ept2 eptx
        )
        (progn
           (setq obj2 (car lst))
           (DUCHECK obj2)
           (if (boundp 'du)
            (setq mesg2 "\nDuplicated.
                         Select again: ")
            (progn
                (setq name2 (cadr lst)
                      spt2
                      (cdr (assoc 'SPT (eval obj2)))
                      ept2
                      (cdr (assoc 'EPT (eval obj2)))
                )
            )
           )
        )
    )
    (if (null du)
        (if (null ept2)
            (VSAVE2)
            (ESAVE2)
        )
    )
)

(if (or (and (boundp 'done)
             (equal en 3)
             (equal EDGE[1] EDGE[3])
        )
        (and (boundp 'done)
             (< en 3)
        )
    )
  (progn
    (setq mesg2
```

```
          "\nToo few boundaries to form a space.")
    (princ mesg2)
    (setq stop "Y")
  )
)

(if (equal mesg2
           "\nUnclosed boundary. Select again: ")
  (progn
    (princ "\nUnclosed boundary. No space formed.")
    (setq stop "Y")
  )
)

(if (and (null stop)
         (boundp 'eptx)
    )
  (progn
    (set (var "EDGE[en]") nil)
    (setq en (1- en)
       VEX[0] (valu "VEX[vn]")
    )
  )
)

(if (null stop)
  (progn
    (setq tmp (list (list 'VEX vn)
                    (list 'EDGE en)
              )
          i   0
    )
    (while (<= i vn)
        (setq v
              (cons (var "VEX[i]") (valu "VEX[i]"))
              tmp (append tmp (list v))
              i   (1+ i)
        )
    )
    (setq i 1)
    (while (<= i en)
        (setq obj  (valu "EDGE[i]")
              name (cadr (assoc 'NAME (eval obj)))
        )
        (if (equal name "Asst-line")
            (setq obj "NONE")
        )
        (setq e   (list (var "EDGE[i]") obj)
              tmp (append tmp (list e))
              i   (1+ i)
        )
```

```
        )
        (setq spcn (1+ spcn))
        (set (var "SPC[fln][spcn]") tmp)
        (princ "\nSpace ")
        (princ (var "SPC[fln][spcn]"))
        (princ " formed.")
      )
    )

    (setq i 0)
    (while (<= i en)
        (set (var "VEX[i]") nil)
        (set (var "EDGE[i]") nil)
        (setq i (1+ i))
    )
    (princ)
);end of SCREATE
;---------------------------------------

(defun DUCHECK ( obj / i tmpobj1 tmpobj2)
;check if a boundary is selected again

    (setq i 0)
    (while (and (null du)
                (<= i en)
           )
      (setq tmpobj1 (cadr (valu "VEX[i]"))
            tmpobj2 (valu "EDGE[i]")
      )
      (if (or (equal obj tmpobj1)
              (equal obj tmpobj2)
          )
          (setq du "Y")
          (setq i (1+ i))
      )
    )
);end of DUCHECK
;---------------------------------------

(defun VSAVE1 ()
;save the first vertex

    (if (boundp 'VEX[0])
        (setq mesg1 "\nUnclosed boundary. Select again: ")
        (setq VEX[0] (list spt1 obj1)
              sptx   spt1
              eptx   ept1
              objx   obj1
        )
    )
);end of VSAVE1
```

```
;---------------------------------------

(defun ESAVE1 ()
;save the first edge

   (if (null VEX[0])
       (setq en (1+ en)
             EDGE[1] obj1
             sptx spt1
             eptx ept1
             objx obj1
       )
       (if (or (equal sptx spt1)
               (equal sptx ept1)
               (equal (angle spt1 sptx)
                      (angle sptx ept1) 0.01
               )
           )
         (setq en 1
               EDGE[1] obj1
         )
         (setq mesg1
               "\nUnclosed boundary. Select again: ")
       )
   )
);end of ESVAE1
;---------------------------------------

(defun VSAVE2 ()
;save vertexes

   (if (equal vn en)
       (setq mesg2 "\nUnclosed boundary. Select again: ")
       (progn
         (if (or (equal spt2 spt1)
                 (equal spt2 ept1)
                 (equal (angle spt1 spt2)
                        (angle spt2 ept1) 0.01
                 )
             )
           (progn
             (setq vn (1+ vn)
                   mesg2 "\nSelect boundary: "
             )
             (set (var "VEX[vn]") (list spt2 obj2))
           )
           (setq mesg2
                 "\nUnclosed boundary. Select again: ")
         )
       )
   )
```

```
);end of VSAVE2
;---------------------------------------

(defun ESAVE2 ()
;save edges

   (if (boundp (var "VEX[en]"))
     (progn
       (setq pt (car (valu "VEX[en"")))
       (if (or (equal pt spt2)
               (equal pt ept2)
               (equal (angle spt2 pt)
                      (angle pt ept2) 0.01
               )
           )
         (progn
            (setq spt1 spt2
                  ept1 ept2
                  en (1+ en)
                  mesg2 "\nSelect boundary: "
            )
            (set (var "EDGE[en]") obj2)
         )
         (setq mesg2
               "\nUnclosed boundary. Select again: ")
       )
     )
     (progn
       (setq pt (inters spt1 ept1 spt2 ept2))
       (if (or (equal spt1 spt2)
               (equal spt1 ept2)
           )
           (setq pt spt1)
       )
       (if (or (equal ept1 spt2)
               (equal ept1 ept2)
           )
           (setq pt ept1)
       )
       (if (null pt)
           (setq mesg2
                 "\nUnclosed boundary. Select again: ")
           (progn
             (setq spt1 spt2
                   ept1 ept2
                   vn (1+ vn)
                   en (1+ en)
                   mesg2 "\nSelect boundary: "
             )
             (set (var "VEX[vn]") (list pt))
             (set (var "EDGE[en]") obj2)
```

```
            )
        )
      )
    )
);end of ESAVE2


;FLOORS
;============================================================
(defun c:FLOOR (/ tmpfln file1 file2)
;set new floor or change to another floor

   (initget "New")
   (setq tmpfln (getint "\nNew/<Change to>: "))
   (if (equal tmpfln fln)
       (progn
          (princ "\nIt is the current floor.")
          (setq tmpfln nil)
       )
   )
   (if (boundp 'tmpfln)
       (if (equal tmpfln "New")
        (progn
;set new floor
           (setq tmpfln (getint "\nNew floor number: "))
           (if (equal tmpfln fln)
              (progn
                (princ "\nIt is the current floor.")
                (setq tmpfln nil)
              )
           )
           (if (numberp tmpfln)
               (NEWFLOOR tmpfln)
           )
        )
        (progn
;change to another floor
           (if (member tmpfln FLLST)
            (progn
;save the data of the current floor
               (WRITEDAT)
               (setq file1
                     (strcat file ".C" (itoa tmpfln))
                     file2
                     (strcat file ".S" (itoa tmpfln))
               )
;read the data of the selected floor
               (READFILE file1)
               (READFILE file2)
               (CHFLOOR tmpfln)
            )
```

```
            (progn
               (princ "\nFloor-")
               (princ tmpfln)
               (princ " not found.
                        \nSelect \"New\" to create it.")
            )
           )
         )
        )
    )
    (princ)
);end of FLOOR
;---------------------------------------

(defun NEWFLOOR ( newfln / comfm oldfln onfl clsfl)
;execute a new floor setting

    (if (member newfln FLLST)
      (progn
        (princ "\nFloor ")
        (princ newfln)
        (princ " already exists. Delete it <N>? ")
        (initget "Yes")
        (setq comfm (getkword))
        (if (equal comfm "Yes")
          (progn
            (WRITEDAT)
            (DELFLOOR newfln)
            (CHFLOOR newfln)
          )
        )
      )
      (progn
         (WRITEDAT)
         (setq oldfln fln
               fln newfln
               FLLST (append FLLST (list fln))
         )
         (set (VAR "waln[fln]") 0)
         (set (VAR "opngn[fln]") 0)
         (set (VAR "clumn[fln]") 0)
         (set (VAR "spcn[fln]") 0)

         (CH-VAR)
         (setq onfl  (strcat "FL-" (itoa fln))
               clsfl (strcat "*DTL-" (itoa oldfln)
                             "*,FL-" (itoa oldfln))
         )
         (command "layer" "m" onfl "n" wdtl[1] "n" wdtl[2]
                          "n" wdtl[3] "n" ddtl[1] "n"
                               ddtl[2] "f" clsfl ""
```

```
        )
        (CHLEVEL)
      )
   )
);end of NEWFLOOR
;---------------------------------------

(defun CHFLOOR (newfln / oldfln onfl clsfl)
;execute floor changing

   (setq oldfln fln
         fln newfln
   )

   (CH-VAR)

   (setq onfl (strcat "FL-" (itoa fln))
         clsfl (strcat "*DTL-" (itoa oldfln)
                       "*,FL-" (itoa oldfln)
               )
   )
   (command "layer" "t" onfl "m" onfl "f" clsfl "")
   (CHLEVEL)
);end CHFLOOR
;---------------------------------------

(defun CH-VAR ( / dtl)
;change glable variables

   (setq dtl (strcat "DTL-" (itoa fln))
         wdtl[1] (strcat "W" dtl "-1")
         wdtl[2] (strcat "W" dtl "-2")
         wdtl[3] (strcat "W" dtl "-3")
         wdtl[4] (strcat "W" dtl "-4")

         ddtl[1] (strcat "D" dtl "-1")
         ddtl[2] (strcat "D" dtl "-2")
         ddtl[3] (strcat "D" dtl "-3")

         waln (VALU "waln[fln]")
         opngn (VALU "opngn[fln]")
         clumn (VALU "clumn[fln]")
         spcn (VALU "spcn[fln]")
   )
);end of CH-VAR
;---------------------------------------

(defun DELFLOOR (n / dtl8 dtl dtl1 dtl2 dtl3 dtl4 dtl5
                     dtl6 dtl7 i ii len entlst ent)
;delete an existing floor to set it as a new one
```

```
    (set (VAR "WALLST[n]") nil)
    (set (VAR "WALEMP[n]") nil)
    (set (VAR "OPNGLST[n]") nil)
    (set (VAR "OPNGEMP[n]") nil)
    (set (VAR "CLUMLST[n]") nil)
    (set (VAR "CLUMEMP[n]") nil)
    (set (VAR "waln[n]") 0)
    (set (VAR "opngn[n]") 0)
    (set (VAR "clumn[n]") 0)
    (set (VAR "spcn[n]") 0)

    (setq dtl8 (strcat "FL-" (itoa n))
          dtl  (strcat "DTL-" (itoa n))
          dtl1 (strcat "W" dtl "-1")
          dtl2 (strcat "W" dtl "-2")
          dtl3 (strcat "W" dtl "-3")
          dtl4 (strcat "W" dtl "-4")
          dtl5 (strcat "D" dtl "-1")
          dtl6 (strcat "D" dtl "-2")
          dtl7 (strcat "D" dtl "-3")

          i    1
    )

    (while (<= i 8)
      (setq dtl (read (strcat "dtl" (itoa i))))
      (setq entlst (ssget "X" (list (cons 8 (eval dtl)))))
      (if (boundp 'entlst)
          (progn
             (setq len (sslength entlst)
                   ii  0
             )
             (while (< ii len)
                 (setq ent (ssname entlst ii))
                 (entdel ent)
                 (setq ii (1+ ii))
             )
          )
      )
      (setq i (1+ i))
    )
);end of DELFLOOR


;MOVE
;=============================================================
(defun c:MV (/ lst obj name basept newpt tmp tmpx tmpy)
;move an object and its subobjects

    (setq lst (OBJSEL "\nSelect object: "))
    (if (boundp 'lst)
```

```
        (progn
        (setq obj (car lst)
              name (cadr lst)
              basept (caddr lst)
              newpt "Base-point"
        )
        (princ "\nThis is ")
        (princ name)
        (princ " ")
        (princ obj)
        (princ ".")
;reset base point
        (while (equal newpt "Base-point")
           (initget "Base-point XY")
           (setq newpt (getpoint basept
                       "\nBase-point/XY/<Second-point>: ")
           )
           (if (equal newpt "Base-point")
            (progn
               (setq tmp (getpoint "\nBase point: "))
               (if (boundp 'tmp)
                   (setq basept tmp)
               )
            )
           )
        );end of while

;get XY distance
        (if (equal newpt "XY")
            (progn
               (setq tmpx (getdist "\nX distance: ")
                     tmpy (getdist "\nY distance: ")
               )
               (if (and (boundp 'tmpx)
                        (boundp 'tmpy)
                   )
                   (setq newpt (polar basept 0 tmpx)
                         newpt (polar newpt (/ pi 2) tmpy)
                   )
                   (setq newpt nil)
               )
            )
        )

        (if (boundp 'newpt)
            (progn
               (setq dist (distance basept newpt)
                     ang (angle basept newpt)
               )
               (MOVOBJ obj ang dist)
            )
```

```
        )
        );end of progn
    )
    (princ)
);end of MV
;---------------------------------------

(defun MOVOBJ (_obj _ang _dist / lst objdwg objsub spt
                ept newspt newept hdl ent subent)
;execute object moving

    (setq lst    (eval _obj)
          objdwg (cdr (assoc 'DWG lst))
          objsub (cdr (assoc 'SUB lst))
          spt    (assoc 'SPT lst)
          ept    (assoc 'EPT lst)

          newspt (polar (cdr spt) _ang _dist)
          newspt (cons 'SPT newspt)
    )
    (set _obj (subst newspt spt lst))

    (if (boundp 'ept)
      (progn
        (setq newept (polar (cdr ept) _ang _dist)
              newept (cons 'EPT newept)
        )
        (set _obj (subst newept ept (eval _obj)))
      )
    )

    (if (boundp 'objdwg)
        (foreach hdl objdwg
          (setq ent (handent hdl))
          (command "move" ent "" basept newpt )
        )
    )

;move subobjects
    (if (boundp 'objsub)
        (foreach subent objsub
          (MOVOBJ subent _ang _dist)
        )
    )
);end of MOVOBJ


;REMOVE ASST-LINES
;============================================================
(defun c:CLEAN (/ entlst ent len i ly)
;remove all assistant lines
```

```
   (setq entlst (ssget "X" (list (cons 8 "0"))))
   (if (boundp 'entlst)
       (progn
          (setq aln     0
                ALINLST nil
                len     (sslength entlst)
                i       0
          )
          (while (< i len)
               (setq ent (ssname entlst i))
               (entdel ent)
               (setq i (1+ i))
          )
       )
   )
;change the layer back
   (setq ly (strcat "FL-" (itoa fln)))
   (command "layer" "s" ly "")
   (princ)
);end of CLEAN


;DELETE
;============================================================
(defun c:DELETE (/ lst comfm type obj
                   tmpsup oldsub newsub)
;delete an object

   (setq lst (OBJSEL "\nSelect Object: "))
   (if (boundp 'lst)
     (progn
       (setq obj  (car lst)
             name (cadr lst)
       )
       (princ "\nThis is ")
       (princ name)
       (princ " ")
       (princ obj)
       (princ ". Delete it <N>?")
       (initget "Yes No")
       (setq comfm (getkword))
       (if (equal comfm "Yes")
         (progn
            (DELOBJ obj)
            (setq tmpsup (cadr (assoc 'SUP (eval obj)))
                  oldsub (assoc 'SUB (eval tmpsup))
                  newsub (RMX obj oldsub)
            )
            (set  tmpsup
                  (subst newsub oldsub (eval tmpsup))
```

```
            )
            (set obj nil)
          )
        )
      )
    )
    (princ)
);end of DELETE
;---------------------------------------

(defun DELOBJ (cobj / tmplst objroot objflr objlst objemp
                 objsub objdwg hdl ent lst subent)
;execte object deleting


    (setq tmplst  (eval cobj)
          objroot (cadr (assoc 'ROOT tmplst))
          objflr  (caddr (assoc 'ROOT tmplst))

          objlst  (VAR (strcat objroot "LST[objflr]"))
          objemp  (VAR (strcat objroot "EMP[objflr]"))
          objdwg  (cdr (assoc 'DWG tmplst))
          objsub  (cdr (assoc 'SUB tmplst))
    )

    (foreach hdl objdwg
      (setq ent (handent hdl)
            lst (list hdl cobj)
      )
      (entdel ent)
      (set objlst (RMX lst (eval objlst)))
    )

    (set objemp (append (eval objemp) (list cobj)))
;delete its subobjects
    (if (boundp 'objsub)
      (foreach subent objsub
          (DELOBJ subent)
          (set subent nil)
      )
    )
);end of DELOBJ


;ATTACH AND DETACH
;=======================================================
(defun c:ATTACH (/ subobj pobj)
;change parent object for selected object(s)

    (setq subobj (MOBJSEL))
    (if (boundp 'subobj)
```

```
        (setq pobj (car (OBJSEL "\nNew parent-object: ")))
    )

    (if (boundp 'pobj)
        (XXTACH subobj pobj)
    )
    (princ)
);end of ATTACH
;---------------------------------------

(defun c:DETACH (/ subobj pobj)
;release selected object(s) from its parent object

    (setq subobj (MOBJSEL))
    (if (boundp 'subobj)
        (progn
         (setq pobj (VAR "FL[fln]"))
         (XXTACH subobj pobj)
        )
    )
    (princ)
);end of DETACH
;---------------------------------------

(defun XXTACH (lst pob / obj oldsub
                  newsub newsup oldsup oldind newind)
;execute attaching some objects to another object

;change the pointer for a new higher level object
    (foreach obj lst
      (setq oldsup (assoc 'SUP (eval obj))
            supobj (cadr oldsup)
      )
      (if (or (equal supobj pob)
              (equal obj pob)
          )
          (setq lst (RMX obj lst))
          (progn
            (setq newsup (list 'SUP pob))
            (set obj (subst newsup oldsup (eval obj)))

;remove its pointer from original parent object
            (setq oldind (assoc 'SUB (eval supobj))
                  newind (RMX obj oldind)
            )
            (set supobj
                 (subst newind oldind (eval supobj))
            )
          )
      )
    )
```

```
;put its pointer into tne new parent object
   (setq oldsub (assoc 'SUB (eval pob)))
   (if (null oldsub)
       (set pob (append (eval pob)
                        (list (cons 'SUB lst))
                )
       )
       (progn
          (setq newsub (append oldsub lst))
          (set pob (subst newsub oldsub (eval pob)))
       )
   )
);end of XXTACH


;REDRAW
;============================================================
(defun c:REDRAW (/ coo oolst ln i n ent)
;redraw all objects

   (command ".redraw")    ;call AutoCAD command
   (setq i 1)

;redraw all openings to keep them invisible
   (while (<= i opngn)
       (setq coo (VAR "OPNG[fln][i]"))
       (if (not (equal (eval coo) nil))
        (progn
          (setq oolst (cdr (assoc 'DWG (eval coo)))
                ln    (length oolst)

                ent   (nth 0 oolst)
                ent   (handent ent)
          )          ;get entity name from its handle
          (redraw ent 2)
          (setq n 1)
          (while (< n ln)
          (setq ent (nth n oolst)
                ent (handent ent)
          )
          (redraw ent)
          (setq n (1+ n))
          )
        )
       )
       (setq i (1+ i))
   )
   (princ)
);end of REDRAW
```

```
;DETAIL LEVEL
;============================================================
(defun c:OPTION (/ optn init1 init2 )
;change the level for opening representation
;return the level numbers for windows and doors

   (princ "\nCurrent detail level for Doors is <")
   (princ dlevl)
   (princ ">. ")
   (setq init1 (GMNUSEL "_ddtl" 4))     ;call graphic menu
   (if (boundp 'init1)
       (setq dlevl init1)
   )

   (princ "\n\nCurrent detail level for Windows is <")
   (princ wlevl)
   (princ ">. ")
   (setq init2 (GMNUSEL "_wdtl" 4))
   (if (boundp 'init2)
       (setq wlevl init2)
   )

   (if (or (boundp 'init1)
           (boundp 'init2)
       )
       (CHLEVEL)
   )
   (princ)
);end of OPTION
;---------------------------------------

(defun CHLEVEL (/ wonlayers donlayers)
;execute the level change

   (if (> wlevl 1)
       (setq wonlayers wdtl[1])
   )
   (if (equal wlevl 2)
       (setq wonlayers (strcat wonlayers "," wdtl[2]))
   )
   (if (equal wlevl 3)
       (setq wonlayers
             (strcat wonlayers "," wdtl[2] "," wdtl[3])
       )
   )
   (if (equal wlevl 4)
       (setq wonlayers (strcat wonlayers "," wdtl[4]))
   )

   (if (> dlevl 1)
```

```
        (setq donlayers ddtl[1])
    )
    (if (equal dlevl 3)
        (setq donlayers (strcat donlayers "," ddtl[2]))
    )
    (if (equal dlevl 4)
        (setq donlayers (strcat donlayers "," ddtl[3]))
    )

    (command "layer" "f" "w*" "f" "d*"
          "t" wonlayers "t" donlayers ""
    )

    (c:REDRAW)        ;keep opening invisible
);end of CHLEVEL


;DATA LISTS
;============================================================
(defun c:LK ( / tmp mesg1 mesg2 lst maxm i n totl)
;get data lists of building objects

    (initget "Select Name Wall Opening Column Asst-line")
    (setq tmp (getkword "\nSelect/Name/Wall
                         /Opening/Column/Asst-line: ")
    )
;list data for named object
    (cond ( (equal tmp "Name")
            (setq tmp (read (getstring "\nName: ")))
          )
;select an object to list its data
          ( (equal tmp "Select")
            (setq tmp
                 (car (OBJSEL "\nSelect Object: \n"))
                 )
          )
          ( T
;list data of all walls
            (if (equal tmp "Wall")
                (setq maxm  waln
                      tmp "WAL[fln][i]"
                )
            )
;list data of all openings
            (if (equal tmp "Opening")
                (setq maxm  opngn
                      tmp "OPNG[fln][i]"
                )
            )
;list data of all columns
            (if (equal tmp "Column")
```

```
                (setq maxm  clumn
                      tmp "CLUM[fln][i]"
                )
            )
;list data of all assistant lines
            (if (equal tmp "Asst-line")
                (setq maxm aln
                      tmp "ALIN[i]"
                )
            )
          )
    )

    (princ "\n")
    (if (equal (type tmp) 'str)
        (progn
;list all objects of a type, such as walls, openings,
;etc.
            (textscr)
            (setq i    1
                  totl 0
            )
;print the data on the screen
            (while (<= i maxm)
                   (princ (VAR tmp))
                   (princ "\n")
                   (if (boundp (VAR tmp))
                       (progn
                          (foreach n (VALU tmp)
                                (princ n)
                                (princ "\n")
                          )
                          (setq totl (1+ totl))
                       )
                       (princ "No value\n")
                   )
                   (setq i (1+ i))
                   (princ "\n")
            )
            (princ "\nTotal Number: ")
            (princ totl)
        )

;list a named or selected object
        (if (boundp 'tmp)
            (progn
               (textscr)
               (princ tmp)
               (princ "\n")
               (if (boundp tmp)
                   (foreach n (eval tmp)
```

```
                        (princ n)
                        (princ "\n")
                    )
                    (princ "No value\n")
                )
            )
        )
    )
    (princ)
); end of LK


;SAVE
;============================================================
;save changes, stay in modifying model

(defun c:SAVE ( / tmpf newf comfm cmd)
   (while (not (equal comfm "Yes"))
      (princ "\nEnter the file name <")
      (princ file)
      (princ "> ")
      (setq tmpf (getstring))

      (if (and (not (equal tmpf ""))
               (findfile (strcat tmpf ".dwg"))
          )
          (progn
             (princ "\n")
             (princ tmpf)
             (initget "Yes No")
             (setq comfm (getkword " already exists.
                                   Overwrite it? <N>"))
          )
          (setq comfm "Yes")
      )
   )
   (setq cmd "cmd") ;cmd will be checked in WRITEDAT
;save temporary files
   (WRITEBAC)
   (WRITEDAT)
   (command ".save" file "y")

   (if (not (equal tmpf ""))
       (setq newf tmp)
       (setq newf realfile)
   )
;copy all temporary files to the named files
   (setq cmd (strcat "copy " file ".* " newf ".*"))
   (command "shell" cmd)
   (c:redraw)
   (princ)
```

```
);end of SAVE

;QUIT
;============================================================
(defun c:QUIT (/ op delcmd)
;give up the changes and quit

   (initget "Yes")
   (setq op (getkword "\nReally want to
                       discard all changes to drawing?"))
   (if (equal op "Yes")
       (progn
;delete temporary files
          (setq delcmd (stracat "del " file ".*"))
          (command "shell" delcmd)
;change environmental variables back
          (setvar "CMDECHO"  oldcm)
          (setvar "FILLMODE" oldfl)
          (setvar "BLIPMODE" oldbl)
          (setvar "THICKNESS" oldth)
;recover AutoCAD commands
          (command "redefine" "end"
                   "redefine" "redraw"
                   "redefine" "save"
                   "redefine" "quit"
          )

          (command "quit" "y")
       )
   )
);end of QUIT


;END
;============================================================
(defun c:END ( / copycmd delcmd)
;save all changes, and quit

;change environmental variables back
   (setvar "CMDECHO"  oldcm)
   (setvar "FILLMODE" oldfl)
   (setvar "BLIPMODE" oldbl)
   (setvar "THICKNESS" oldth)
   (command "redefine" "end"
            "redefine" "redraw"
            "redefine" "save"
            "redefine" "quit"
   )

;save objects
   (WRITEBAC)
```

```
   (WRITEDAT)
;copy temporary files back and delete them
   (setq copycmd (strcat "copy " file ".* "
                         realfile ".*"
                 )
         delcmd  (strcat "del " file ".*")
   )
   (command "shell" copycmd)
   (command "shell" delcmd)

   (command "end")
   (princ)
);end of END


;WRITE FILES
;============================================================
(defun WRITEBAC (/ file1 fp)
;write data to main file

   (setq file1 (strcat file ".BAC")
         fp (open file1 "w")
   )
   (prin1 'fln fp)
   (print fln fp)
   (print 'FLLST fp)
   (print FLLST fp)
   (print 'dlevl fp)
   (print dlevl fp)
   (print 'wlevl fp)
   (print wlevl fp)
   (close fp)
);end of WRITEBAC
;---------------------------------------

(defun WRITEDAT (/ n i maxm fp file1 file2 fp1 fp2)
;write a floor data into its file
   (setq tt 0)
   (set (VAR "waln[fln]") waln)
   (set (VAR "opngn[fln]") opngn)
   (set (VAR "clumn[fln]") clumn)
   (set (VAR "spcn[fln]") spcn)

   (setq file1 (strcat file ".C" (itoa fln))
         file2 (strcat file ".S" (itoa fln))
   )

   (setq fp1 (open file1 "w")
         fp2 (open file2 "w")
         fp  fp1
   )
```

```
;as the first line should use function "prin1",
;others use "print", tt has no meaning, it is used
;for the first line, so that the rest of the file
;can use a same function
   (prin1 'tt fp)
   (print "tt"  fp)

;Save cmp-information.
   (WRITEVAR "FL[fln]")
   (WRITEVAR "WALLST[fln]")
   (WRITEVAR "WALEMP[fln]")
   (setq i 1
         maxm (VALU "waln[fln]")
   )
   (while (<= i maxm)
          (WRITEVAR "WAL[fln][i]")
          (setq i (1+ i))
   )
   (WRITEVAR "waln[fln]")

   (WRITEVAR "OPNGLST[fln]")
   (WRITEVAR "OPNGEMP[fln]")
   (setq i 1
         maxm (VALU "opngn[fln]")
   )
   (while (<= i maxm)
          (WRITEVAR "OPNG[fln][i]")
          (setq i (1+ i))
   )
   (WRITEVAR "opngn[fln]")

   (WRITEVAR "CLUMLST[fln]")
   (WRITEVAR "CLUMEMP[fln]")
   (setq i    1
         maxm (VALU "clumn[fln]")
   )
   (while (<= i maxm)
          (WRITEVAR "CLUM[fln][i]")
          (setq i (1+ i))
   )
   (WRITEVAR "clumn[fln]")

   (close fp)

;Save space-information.
   (setq fp fp2)
   (prin1 'tt fp)
   (print "tt"  fp)

   (setq i 1
         maxm (VALU "spcn[fln]")
```

```
   )
   (while (<= i maxm)
          (WRITEVAR "SPC[fln][i]")
          (setq i (1+ i))
   )
   (WRITEVAR "spcn[fln]")
   (close fp)
);end of WRITEDAT
;---------------------------------------

(defun WRITEVAR ( v / tmp)
;save a variable

   (setq tmp (VAR v))
   (if (boundp tmp)
     (progn
       (print tmp fp)
       (print (eval tmp) fp)
;delete this variable except for function SAVE
       (if (null cmd)
           (set tmp nil)
       )
     )
   )
);end of WRITEVAR


;READ FILES
;============================================================
(defun READFILE ( ff / tmp tmp1 fp)
;read a named file

   (if (findfile ff)
     (progn
       (setq fp  (open ff "r")
             tmp (read-line fp)
       )
       (while (not (equal tmp nil))
              (setq tmp (read tmp))        ;variable name
              (setq tmp1 (read-line fp))
              (set  tmp (read tmp1))       ;value
              (setq tmp (read-line fp))
       )
       (close fp)
       (setq waln  (VALU "waln[fln]")
             opngn (VALU "opngn[fln]")
             clumn (VALU "clumn[fln]")
             spcn  (VALU "spcn[fln]")
       )
     )
     (progn
```

```
        (princ "\007\007\007\n")
        (princ file1)
        (princ " not found.")
      )
    )
);end of READFILE


;SELECTE OBJECT
;=============================================================
(defun OBJSEL (_mesg / _ent _hdl _lst _type _basept _obj)
;select an object and return a list of variable name,
;object name, and the base point,
;for example: (WAL[1][1] Wall (3.0 3.0 0.0))
   (setq _ent (entsel _mesg))
   (if (boundp '_ent)
       (progn
          (setq _basept (cadr _ent)
                _ent (car _ent)
                _hdl (cdr (assoc 5 (entget _ent)))
          )  ;get the handle
;search for the object in data list
          (setq _obj
               (cadr (assoc _hdl (VALU "WALLST[fln]")))
          )
          (if (null _obj)
              (setq _obj
                (cadr (assoc _hdl (VALU "OPNGLST[fln]")))
              )
          )
          (if (null _obj)
              (setq _obj
                (cadr (assoc _hdl (VALU "CLUMLST[fln]")))
              )
          )

          (if (null _obj)
              (setq _obj (cadr (assoc _hdl ALINLST)))
          )

          (setq _type (cadr (assoc 'NAME (eval _obj))))
          (list _obj _type _basept) ;get the list
       )
   )
);end of OBJSEL
;---------------------------------------

(defun MOBJSEL (/ stat obj objlst)
;select several objects at on time
   (setq stat "Add")
   (while (or (equal stat "Remove")
```

```
            (equal stat "Add")
        )
        (setq obj (car (OBJSEL "\nSelect object: ")))
        (while (boundp 'obj)
            (if (equal stat "Add")
                (if (member obj objlst)
                    (princ "\nDuplicated.")
                    (setq objlst
                        (append objlst (list obj))
                    )
                )
                (if (member obj objlst)
                    (setq objlst (RMX obj objlst))
                    (princ "\nNot found in
                            the selected-object set.")
                )
            )
            (setq obj
                 (car (OBJSEL "\nSelect object: "))
            )
        )
        (initget "Remove Add Done")
        (setq stat (getkword "\nRemove/Add/<Done>: "))
   )
   (setq objlst objlst)
);end of MOBJSEL


;GRAPHIC MENU
;============================================================
(defun GMNUSEL ( _name _number / _base _higt _scale
                                                 _choice)
;call graphic menu, and return the selected number

   (setq _name (strcat "graphmnu/" _name ".dwg"))
   (if (findfile _name)
     (progn
      (setq _base (getvar "viewctr")
            _higt (getvar "viewsize")
            _scale (/ _higt 9)
            _choice 0
      )
;insert the graphic menu
      (command "insert" _name _base _scale "" "")

      (while (and (numberp _choice);input is not a number
                  (or (> _choice _number)
                      (<= _choice 0)    ;out of the range
                  )
             )
          (setq _choice (getint "\nType the number
```

```
                                    of your selection:")
            )
         )
         (entdel (entlast))      ;remove the graphic menu
         (setq _choice _choice) ;return the selection
       )
       (progn
         (princ "\nGraphic menu ")
         (princ "\"")
         (princ _name)
         (princ "\"")
         (princ " not found.")
         (princ)
       )
    )
);end of GMNUSEL


;OTHERS
;=============================================================
(defun VAR (_v / _i _ln _vname _char _m _n)
;this function makes set possible in AutoLISP
;it gives the name of a variable with subvariables,
;i.e. WAL[m][n] -- WAL[1][3]

    (setq _ln    (strlen _v)
          _i     1
          _vname ""
          _char  (substr _v _i 1)
    )
    (while (and (not (equal _char "["))
                (<= _i _ln)
           )
        (setq _vname (strcat _vname _char)
              _i     (1+ _i)
              _char  (substr _v _i 1)
        )
    )

    (if (equal _char "[")
        (RDN)
    )
    (if (< _i _ln)
      (progn
         (setq _m _n
               _i (1+ _i)
         )
         (RDN)
      )
    )
```

```
   (if (not (equal _m nil))
       (setq _vname
             (strcat _vname "[" (itoa (eval _m)) "]")
       )
   )
   (if (not (equal _n nil))
       (setq _vname
             (strcat _vname "[" (itoa (eval _n)) "]")
       )
   )

   (setq _vname (read _vname))
);end of VAR
;---------------------------------------

(defun RDN ()
;change the subvariable from string to integer

   (setq _i    (1+ _i)
         _char (substr _v _i 1)
         _n    ""
   )
   (while (and (not (equal _char "]"))
               (<= _i _ln)
          )
       (setq _n (strcat _n _char)
             _i (1+ _i)
             _char (substr _v _i 1)
       )
   )
   (setq _n (read _n))
);end of RDN
;---------------------------------------

(defun VALU (_v)
;give the value of a variable with subvariables

   (eval (VAR _v))
);end of VALU
;---------------------------------------

(defun RMX (_item _list / _half1 _half2 _tmplst _newlst)
;remove an item from a list, and return the new list

   (if (member _item _list)
       (setq _half2 (cdr (member _item _list))
             _tmplst (reverse _list)
             _tmplst (cdr (member _item _tmplst))
             _half1  (reverse _tmplst)
             _newlst (append _half1 _half2)
       )
```

```
        (setq _newlst _list)
    )
);end of RMX
```

BIBLIOGRAPHY

——."AutoCAD Release 10 — Reference Manual." Autodesk, Inc.(1989)

——."AutoLISP Programmer's Reference" Autodesk, Inc.(1989)

Boy, Guy A."Intelligent Assistant Systems." translated by Philippa H. Boy Moffett Field, CA: NASA-Ames Research Center (1989).

Broadbent, Geoffrey."Design in Architecture-Architecture and the Human Sciences."David Fulton Publishers Ltd. (1988)

Cleal, D. M.,and N. O. Heaton."Knowledge-based Systems." NY: Halsted Press (1988).

Huo, Xinming."AutoLISP Program Design & Applications." China: H. Computer Tech., Inc.(1989)

Jablonski, Allen D."Integrated Component-based Computer Design Modeling System." Newark, NJ: NJIT (1990).

Khoshafian, Setrag. "Object Orientation." NY: Wiley (1990).

Kim, Won.,and Frederick H. Lochovsky."Object-Oriented Concepts, Databases and Applications." NY: ACM Press (1989).

Laseau, Paul."Graphic Problem Solving — for Architects & Builders"

——."Life Safety Code Handbook" edited by James K. Lathrop. Quincy, MA: National Fire Protection Association (1981).

MacKellar, Bonnie K., and Filiz Ozel. "ArchObjects: Design Codes as Constraints in an Object-Oriented KBMS." Newark, NJ: NJIT (1990).

Omura, George. "The ABC's of AutoLISP." Alameda, CA: SYBEX, Inc.(1990)

Ozel, Filiz. "User Interface Issues in Object Oriented Architectural Data Models and Domain Specific Data Needs of Fire Safety in Buildings." Newark, NJ: NJIT (1991).

Rumbaugh, James, and Michael Blaha, and William Premerlani, and Frederick Eddy, and William Lorensen. "Object-Oriented Modeling and Design." Prentice-Hall, Inc.(1990)

Stanland Jr., Raymond E."The design Process." (1974)

Wade, John W. "Architecture, Problems, and Purposes — Architectural Design as a Basic Problem-Solving Process." John Wiley and Sons, Inc.(1977)