Designing Rules: Heuristics of Invention in Design

by
Josep Maria Fargas i Texidó

Arquitecte, E.T.S.A.Barcelona
February, 1987

Submitted to the Department of Urban Studies and Planning
in Partial Fulfillment of the Requirements for the Degrees of

Master in City Planning
and
Master of Science in Architecture Studies

at the
Massachusetts Institute of Technology
June, 1991

Signature of Author _____
Josep Maria Fargas i Texidó
Department of Urban Studies and Planning and
Department of Architecture, May 20, 1991

Certified by _____
William L. Porter
Muriel and Norman Leventhal Professor of Architecture and Planning and
Head, Department of Architecture. Thesis Supervisor

Accepted by _____
Phillip L. Clay
Chairman, M.C.P. Committee

Accepted by _____
Julian Beinart
Chairman, Departmental Committee for Graduate Students, Course IV

Designing Rules: Heuristics of Invention in Design

by
Josep Maria Fargas i Texidó

Submitted to the Department of Urban Studies and Planning on
May 20, 1991, in partial fulfillment of the requirements for the Degrees
of Master in City Planning and Master of Science in Architecture Studies

Abstract

This thesis studies how designers interact with designs and with other designers, how they share ideas about design qualities, and how they ultimately perceive reality. It also shows how metaphors play a decisive role in the understanding of a given representation reducing the complexity of design, and how they constitute an autonomous ready-made kit of rules more or less internally consistent. This thesis also studies the different levels of design knowledge and its representation in a computer through an experiment about design replication and another one that implements Boston's Midtown Cultural District Zoning Plan.

Is it possible to synthesize and capture some architectural knowledge through a set of concrete rules ? The thesis approaches the example of a set of urban regulations as a design of a compendium of some partial but specific knowledge about architecture. The necessary objectivity and precision of its rules often goes against their needed fundamental character of generality. Thus, it is important to differentiate the rule itself from the spirit in which it has been created. Will it be possible, then, to design a valid set of regulations preserving this level of abstraction and generality or universality ?

As the dualistic character of the title shows, this thesis is about the design of the rules themselves and also about the designs the rules produce. Finally this thesis generalizes the concepts presented by the different experiments and exposes the main challenges towards using computers as design methodology.

Thesis Supervisor: William L. Porter
Title: Muriel and Norman Leventhal Professor of Architecture and Planning and
Head, Department of Architecture

Acknowledgements

I would like to thank my advisor William Porter for his ideas and acute comments, and my two readers: Donald Schön for his rigor and attentiveness and Gary Hack for his enthusiasm and professionalism. I am indebted to Brian DeLorey and Eric Schmidt of Boston Redevelopment Authority for their help and useful information.

I would like also to thank the Ministerio de Educación y Ciencia of Spain for giving me a Fulbright Scholarship without which my post-graduate studies at M.I.T. would not have been possible. I am also in debt to the C.I.R.I.T. of the Generalitat de Catalunya for a scholarship that helped me specially the last semester.

Finally I would like to thank my friend and colleague Pegor Papazian with whom I developed the first two experiments framework of this thesis, for his brightness and support. I am also in debt with my parents for their devotion and sacrifice.

But most of all I would like to thank Susanna, my wife and colleague, for her love and passion, and for bringing me back to reality.

I dedicate this thesis to Susanna and my parents.

Designing Rules: Heuristics of Invention in Design

Contents

"It is this chain of discoveries, as well as each individual discovery, that give rise to the emotion [...] this emotion which invariably follows closely the phases of the creative process." Poetics of Music, Igor Stravinsky.

Introduction

Trained first as an architect and later as a planner, my interests in design and specially in design methodologies have always been crucial to the way I look at the world. I think the way designers can improve their skills is by understanding how and what they design and under what kind of worlds they design. The following thesis is the result of a thinking process I have developed and synthesized at M.I.T., for the last three years under the very different courses I have been attending, in architecture, planning and computer science.

The thesis is divided in three parts in which I expose my thesis about new design methodologies or how to design in companionship with computers. To illustrate my thinking I expose different design experiments where I expose the kind of design knowledge involved in each experiment and the representation of this knowledge.

In the first part of the thesis I expose the two design experiments that constitute the framework of the experiment exposed in the second part that constitutes the main body of the thesis. The third part of the thesis generalizes the concepts exposed in the two previous parts.

The first design experiment, "Disposable Metaphors: An Experiment with a Frame, Two Rectangles and Two Lines", studies how designers interact with designs and with other designers, how they share ideas about design qualities, and how they ultimately perceive reality. It also shows how metaphors play a decisive role in the understanding of a given representation reducing the complexity of design, and how they constitute an autonomous ready-made kit of rules more or less internally consistent.

The second design experiment, "EstheR: An Esthetics Replicant", is about design replication, and it is a continuation of the first experiment about designers' actions on specific simple pictures. EstheR replicates the design moves of the subjects we studied in the Disposable Metaphors. EstheR shows how it is possible to design several metaphor-packages and write some procedural rules which constitute a more-or-less complete equivalent to the subject's metaphor, building them from their own specific "vocabulary".

The thesis also analyzes the kind of problems encountered in designing EstheR about design knowledge and its representation, and how they were solved or simplified. At the end of the first part, the thesis proposes a revised model for EstheR to better capture the real design process where designers are generally influenced by several metaphors simultaneously for every design move, a kind of metaphor competitiveness.

The second part of the thesis is about designing rules. As the dualistic character of the title shows, the thesis is about the design of the rules themselves and also about the designs the rules produce. Designing Rules, Boston (DeaRB) is the design experiment I developed for the thesis that implements Boston's Midtown Cultural District zoning regulations. DeaRB studies how regulations as a set of rules influence design and how to design the rules or the set of regulations to produce better designs, that is, designs that "better" express the purpose of the regulations, if we consider regulations as a specific set of knowledge about

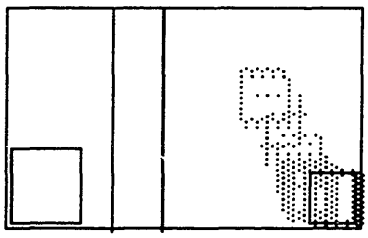
design. The challenge when designing rules to regulate design consists on the possibility of creating more general but at the same time more accurate rules, rules that will solve more satisfactorily a bigger number of special cases in which the traditional regulations fail in giving appropriate design solutions. To do this the thesis sustains a design approach to regulations over the "legal" approach, using heuristics based in design concepts expressed as set of designing rules. It proposes to recover and implement the knowledge from which the regulations were originally built and express it as a sets of rules or metaphor packages. The thesis proposes also how under different paradigms or sets of metaphors different characters or "hats" might be implemented acting and interacting under these different worlds.

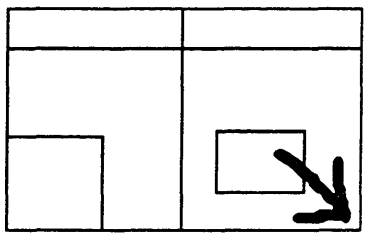
The thesis also shows how to use a CAD platform to implement designing rules, the kind of knowledge implemented and its internal and external representation. The specific implementation of DeaRB is explained with care showing the different strategies and programming techniques used. At the end of the second part, DeaRB is put in the context of EstheR showing the future possibilities of a "regulator" program to be extended to other contexts and plays a little bit of science fiction. It will be possible to model design negotiations where different "hats" will interact with each other regulating designs, and an internally designed arbiter will decide what regulations to apply.

The last part of the thesis generalizes the concepts developed in the first two parts and exposes several concerns about design methodologies, specially about regulations and Expert Systems where the expert will be the designer of the regulations and where the program would regulate helping to visualize instantiations of the regulations by capturing the knowledge of the expert. Issues about different kinds and levels of knowledge are pointed out, as well as representation and conception strategies, that is the means and meanings of designs.

Finally the thesis exposes some ideas about future CAD tools where the user will be able to define or tailor his/her design world in the computer, and the computer will in turn remember it, for each different user, so that every time the user will make use of a command this will behave as the user will expect, under his/her world or metaphor.

Part 1. EstheR: An Esthetics Replicant





1. Background

Disposable Metaphors: An Experiment with a Frame, Two Rectangles and Two Lines

The following thesis is the continuation of two experiments ("Disposable Metaphors: An Experiment with a Frame, Two Rectangles and Two Lines", and "EstheR: An Esthetics Replicant") that I accomplished jointly with Pegor Papazian and which I will describe and analyze in this Part since they constitute the fundamental framework of the thesis.

Disposable Metaphors or Gestaltatron was an experiment about design methods carried out in Donald Schön's Design Research Seminar at M.I.T., and in which we studied how designers interact with designs and with other designers, how they share ideas about design qualities, and how they ultimately perceive reality.

We showed how metaphors play a decisive role in the understanding of a given representation. The meanings a designer assigns to any ambiguous or abstract elements of a representation are crucial for defining the subsequent evolution of a design. We investigated some of the ways in which metaphors are used, combined, brought into competition and disposed of. We distinguished between analogies drawn by a designer between a representation and things in the world, and the analogies drawn by an observer between the same representation and other things. Finally we introduced the idea of Equivalent metaphors as a tool for replicating designers' moves. It is from here that we developed a knowledge-based system in which the expert was a designer. That system was EstheR.

For the purpose of the thesis, I want to point out that on the one hand there are metaphors and on the other the use individuals make of them. Individuals tend to use self-imposed constraints to create or to understand a design and they use metaphors as long as they are useful to act on a picture or a design. Metaphors reduce the complexity of the design task and facilitate communication between the designer's world and the world of others. A metaphor is a kind of world, a ready-made kit of rules more or less internally consistent, with its own autonomy. In our experiment for example, "stable" means "in equilibrium" in a statics or gravity metaphor. But what is very remarkable is that metaphors are used, combined and disposed of in different ways by designers more or less

successfully, depending on the robustness and completeness of the metaphors themselves and also depending on the consistency of the designer that uses them.
Some of the metaphors we derived from the experiment were the Coordinate-Axis or Centering, Statics, Snapshot, Extrusion, Landscape, Perspective-Scene and Mondrian-Painting metaphors. The more relevant uses of metaphors we found were Switching (designers switch metaphors according to their convenience), Overlaping (sometimes two or more metaphors might be used at the same time by a designer) and Sleeping metaphors (we noticed that metaphors were brought up again into play after being previously rejected or switched with other metaphors).

In designing EstheR, our second experiment about Knowledge-Based Systems, we only dealt with Statics and Centering metaphors for only one subject. We developed the concept of Equivalent metaphors that results from the fact that different metaphors might be used to produce similar pictures. This allows different designers to understand the same design through the different "meanings" they associate with these, or from their different "worlds". Thus, this is the basis we used to write a program that replicates different subjects' moves.

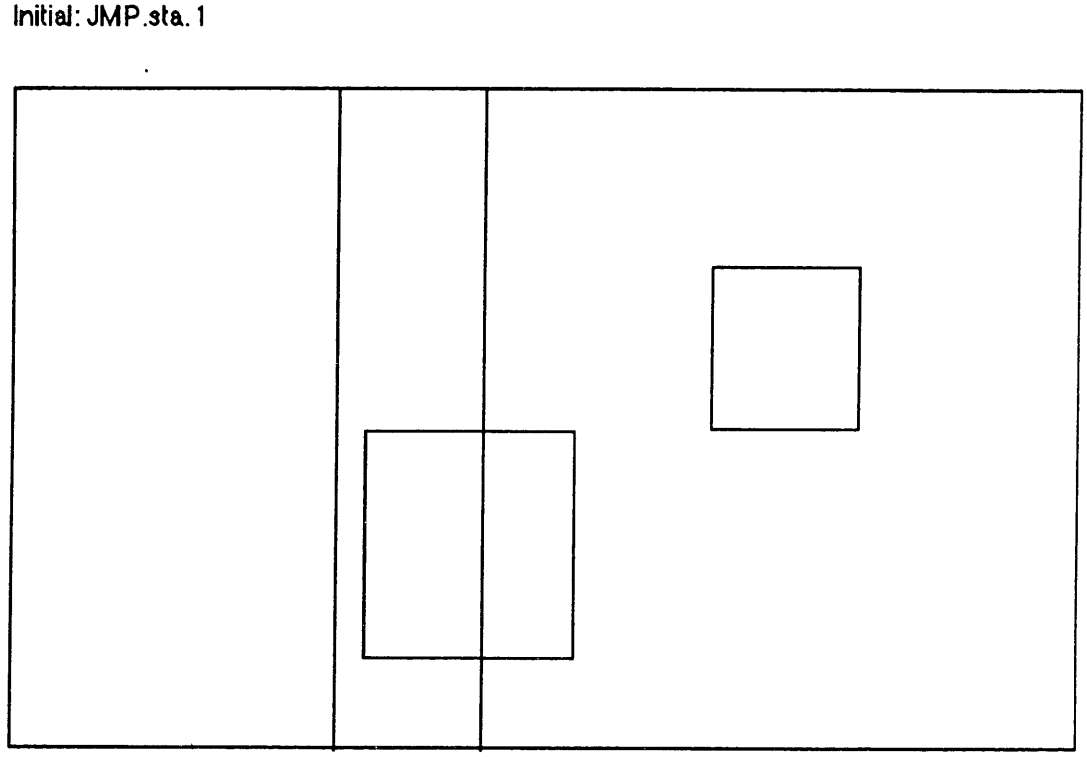
We enriched the scope of what we thought was going to be our next step, EstheR. As a result of Gestaltatron we thought of the possibility of building an analogous expression of a designer's understanding of a given quality. Even if we only studied one subject in EstheR, because we wanted to keep it simple, we claim that to a certain extent, it is possible to design several metaphor-packages and write some procedural rules which constitute a more-or-less complete equivalent to the subject's metaphor (limited to our own understanding and knowledge, and maybe our own biased metaphor?), building them from their own specific "vocabulary" (that up to a certain extent, we claim is shared by designers for that specific metaphor). Then, depending on the subject, certain strategies might be used to get "accurate" replicated actions as I will explain later.

Through Equivalent metaphor-packages EstheR was given a picture internally represented by a list containing the frame, the two lines and the two rectangles (like the one in Figure 1. generated by Gestaltatron) that it modifies to produce

all or some of the results attributable to different subjects' metaphors. The output is then a modified image also represented by a list that can be graphically displayed like each one of the subject's experiments (like Figure 2.). Because the system keeps track of the picture modifications history the process can be decomposed into different frames (see Figure 4., and notice how the last frame h resembles the frame in Figure 2. which was the final modified picture by subject JMP) that resulted from the different "moves" or actions proposed by each rule through the whole cyclical process.

The medium we used for both experiments was a computer program (running on a Macintosh computer in Allegro Common Lisp) which displays an initial picture (Figure 1.) on the screen and allows the subject to move the rectangles by pointing to them and dragging them with a mouse; or allows EstheR to derive a list of the actions to modify this same picture. In Gestaltatron, the first experiment, our subjects or designers were asked to produce a picture more representative of a given quality (in the example below the task was to "make the picture more stable as opposed to more dynamic" without explaining what we meant by stable or dynamic), and all moves were recorded and could be played back in real time or displayed (and printed out) as a collection of frames (see Figure 3.). After a replay is watched, it can be revised just like an initial picture, and a history of that revision is also recorded. Only during the replay and for each picture, the subject was asked to give some explanations about his/her previous actions. The complete history of moves is recorded as a list and can be analyzed in more detail than the sequence of frames would allow. The history contains a description of the initial picture, and a list of moves as follows: (rectangle-#-[1 or 2] time-elapsed-before-this-move new-coordinate-x new-coordinate-y). In EstheR, a similar list of moves is generated so as to output the process which EstheR went through when replicating a specific subject's design (Figure 4.).

Figure 1.
Example of one of the initial pictures randomly generated by Gestaltatron that was given to subject JMP to make it more stable, as opposed to more dynamic

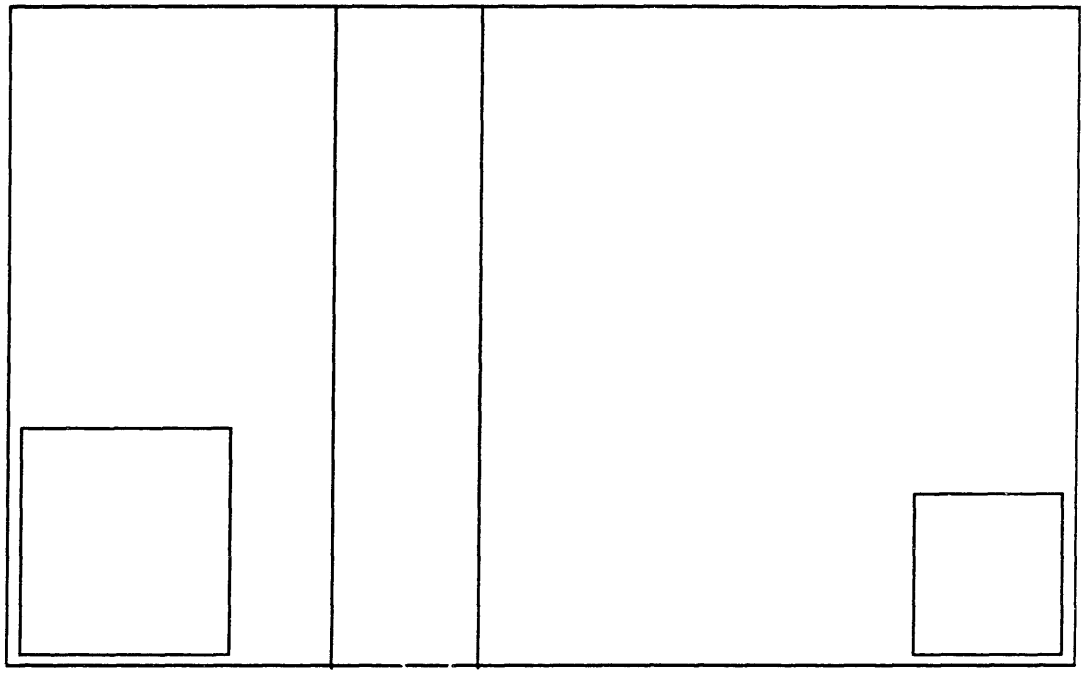


This is the list of the initial picture:

((55 55 455 305) (317 122 373 184) (187 184 266 271) (231 55 231 305) (176 55 176 305))

Figure 2.
This is the "stable" final picture that subject JMP produced during the experiment

JMP.sta.1

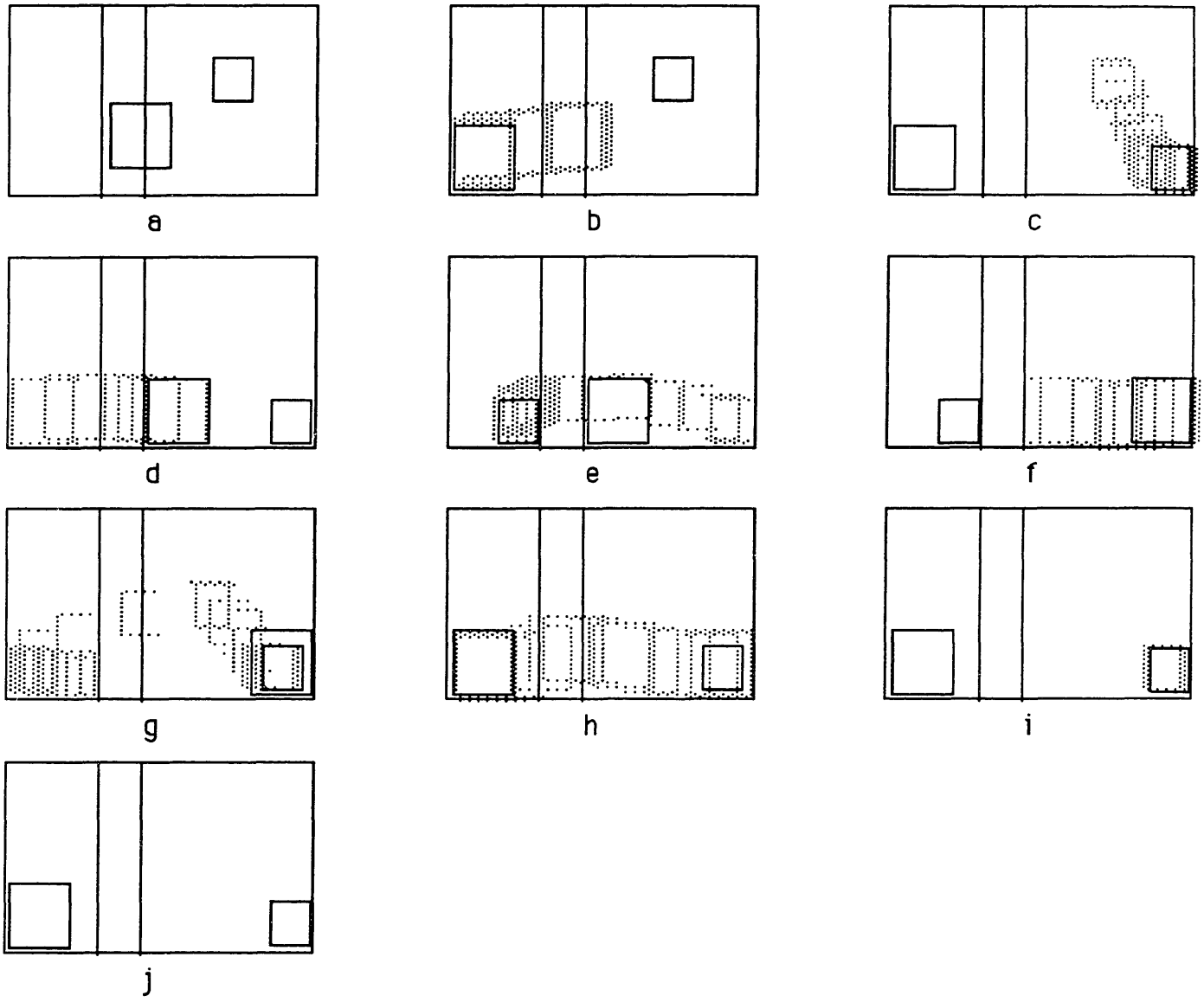


This is the list of the final picture generated by subject JMP through the experiment:

((55 55 455 305) (394 239 450 301) (60 214 139 301) (231 55 231 305) (176 55 176 305))

Processes of initial moves and possible revision moves are printed out as a series of frames, where each frame marks the state of the picture when the subject paused for more than a certain number of seconds, or switched form moving one rectangle to moving the other one. In each frame all intermediate movements leading to the state depicted in that frame are shown in gray.

Figure 3.
Process or collection of frames in which subject JMP went through to modify the initial picture so as to make it more stable



This is the list of the whole process in which subject JMP went through:

(Note that the list of the initial picture is appended to the list of JMP's actions: the rectangle moved, the time elapsed before reaching that position and the two coordinates of the increment or decrement of the new position)

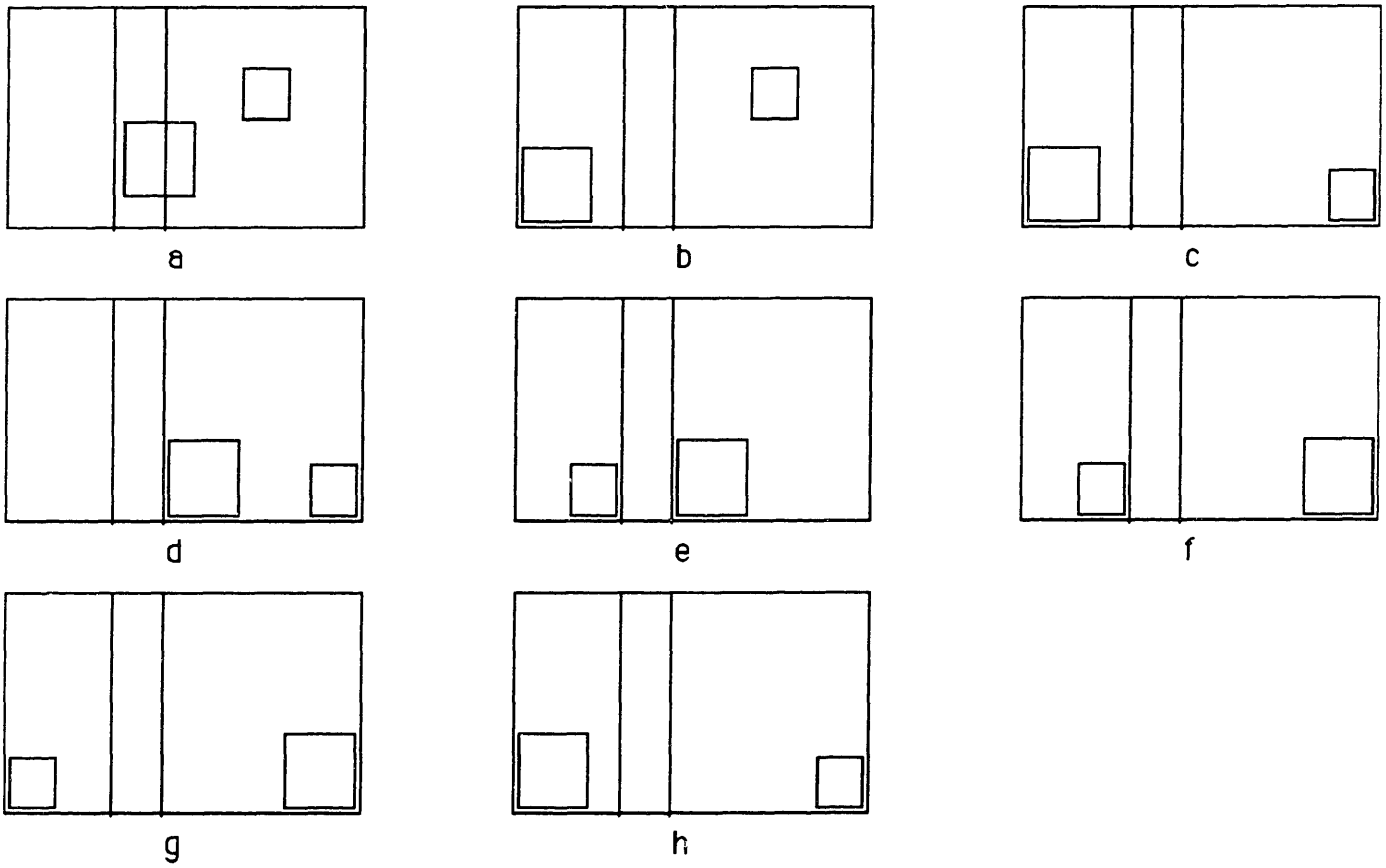
((55 55 455 305) (317 122 373 184) (187 184 266 271) (231 55 231 305) (176 55 176 305) (2 36 -1 0) (2 0 -2 1) (2 0 -4 0) (2 0 -5 1) (2 0 -22 3) (2 0 -30 5) (2 0 -24 2) (2 0 -24 0) (2 0 -5 4) (2 0 -11 5) (2 0 -4 1) (2 0 0 1) (2 0 0 1) (2 0 0 2) (2 0 0 1) (2 0 1 0) (2 0 0 1) (2 0 0 1) (2 1 1 0) (2 1 0 1) (2 2 0 1) (2 2 1 0) (2 1 1 0) (1 28 0 1) (1 0 2 7) (1 0 14 22) (1 0 14 20) (1 0 10 25) (1 0 1 8) (1 0 3 12) (1 0 2 6) (1 0 3 4) (1 0 1 2) (1 0 1 0) (1 0 4 3) (1 0 1 1) (1 0 1 1) (1 0 0 1) (1 1 1 0) (1 0 3 0) (1 0 12 1) (1 0 1 0) (1 0 11 3) (1 0 1 0) (1 0 0 1) (1 0 -2 0) (1 0 -3 0) (1 0 -1 -1) (1 0 -

1 0) (1 0 -1 0) (1 1 1 0) (1 1 0 -1) (1 0 -1 0) (1 2 0 1) (1 2 -1 0) (1 4 -1 0) (1 1 1 0) (2 54 1 0) (2 0 8 -1) (2 0 32 -4) (2 0 36 0) (2 0 28 -2) (2 0 33 4) (2 0 26 2) (2 0 9 2) (2 0 2 0) (2 0 1 0) (2 2 0 -1) (2 3 0 -1) (1 60 0 -1) (1 0 -15 -4) (1 0 -26 -11) (1 0 -46 -6) (1 0 -42 -10) (1 0 -38 4) (1 0 -32 0) (1 0 -40 0) (1 0 -12 3) (1 0 -3 1) (1 0 -4 1) (1 0 -4 1) (1 0 -5 2) (1 0 -4 2) (1 0 -4 1) (1 0 -1 3) (1 0 -2 3) (1 0 -2 4) (1 0 -2 4) (1 0 3 1) (1 0 1 0) (1 1 0 1) (1 0 0 1) (1 0 1 0) (1 0 0 1) (2 64 1 0) (2 0 13 0) (2 0 44 2) (2 0 30 6) (2 0 24 -4) (2 0 34 -4) (2 0 1 0) (2 1 0 1) (2 0 -5 0) (2 1 -1 -1) (2 0 -3 -1) (2 1 -1 0) (2 1 0 1) (2 0 -1 0) (1 14 -1 0) (1 0 -17 1) (1 0 -2 0) (1 0 -4 1) (1 0 -6 -1) (1 0 -7 -1) (1 0 -5 0) (1 0 -4 -2) (1 0 -3 0) (1 0 -1 0) (1 0 -1 0) (1 0 -2 0) (1 0 -1 0) (1 0 -1 0) (1 0 -1 0) (1 1 -1 0) (1 0 0 1) (1 0 -1 0) (1 30 0 -4) (1 0 11 -20) (1 0 48 -22) (1 0 84 -29) (1 0 88 -14) (1 0 7 3) (1 0 18 22) (1 0 16 12) (1 0 16 24) (1 0 12 20) (1 0 5 3) (1 0 5 1) (1 0 1 0) (1 0 3 -1) (1 0 4 0) (1 0 2 0) (1 0 3 1) (1 0 2 0) (2 9 0 1) (2 0 -4 3) (2 0 -15 1) (2 0 -34 -4) (2 0 -40 -5) (2 0 -32 -5) (2 0 -40 -6) (2 0 -26 -3) (2 0 -20 1) (2 0 -24 11) (2 0 -26 9) (2 0 -24 5) (2 0 -20 0) (2 0 -4 -1) (2 0 -1 0) (2 0 1 0) (2 0 0 -1) (2 0 0 -1) (2 0 0 -1) (2 0 0 -2) (2 0 0 -1) (2 1 0 -1) (2 0 0 -1) (1 13 1 0) (1 0 2 1) (1 0 2 1) (1 0 1 0) (1 0 2 1) (1 1 0 1) (1 0 1 0) (1 1 1 0) (2 27 0 1) (2 1 0 1) (2 0 -1 0) (2 0 -1 0) (2 0 0 -1) (2 68 0 0))

2. An example of EstheR's work

The following is an example of the output display of the process and the tracing of the program when it is asked to replicate the same picture that has been given before to a subject. This example is the replication of the JMP's anterior one.

Figure 4.
Process in which Replicant Pris went through to replicate JMP's actions to make the picture more "stable"



The following is the trace of the program in which the first picture "fp" was given to subject "JMP" so as to make it "more stable". It has been designed to be self explanatory. First it shows the triggered rules, then the metaphor chosen and finally the action undertaken. After a move the program starts the same process again.

```
? (replicate replicant (read-pict 'fp "jmp-sa-1"))

- triggered:
("A) LINES DO NOT INTERSECT -> CONSIDER STATICS METAPHOR"
 "B) NO INTERSECTION OF LINES CLOSE TO THE CENTER -> CONSIDER STATICS
METAPHOR"
 "C) BOTH BLOCKS FREE-FLOATING -> MOVE LOWER TO MOST-CONVENIENT-CORNER
LOWER THAN ITSELF")
- fired:
"C) BOTH BLOCKS FREE-FLOATING -> MOVE LOWER TO MOST-CONVENIENT-CORNER LOWER
THAN ITSELF"
- active metaphor: STATICS
- moving BLOCK-2 to UR corner (55 305)
+
+
- triggered:
("A) LINES DO NOT INTERSECT -> CONSIDER STATICS METAPHOR"
 "B) NO INTERSECTION OF LINES CLOSE TO THE CENTER -> CONSIDER STATICS
METAPHOR"
 "D) ONLY ONE BLOCK FIXED -> MOVE FREE ONE TO MOST-CONVENIENT-CORNER LOWER
THAN ITSELF")
- fired:
"D) ONLY ONE BLOCK FIXED -> MOVE FREE ONE TO MOST-CONVENIENT-CORNER LOWER
THAN ITSELF"
- active metaphor: STATICS
- moving BLOCK-1 to UL corner (455 305)
+
+
- triggered:
("A) LINES DO NOT INTERSECT -> CONSIDER STATICS METAPHOR"
 "B) NO INTERSECTION OF LINES CLOSE TO THE CENTER -> CONSIDER STATICS
METAPHOR"
 "E) BOTH BLOCKS CORNER-FIXED -> MOVE MOST-CONVENIENT-BLOCK TO MOST-
CONVENIENT-CORNER"
 "F) BOTH BLOCKS CORNER-FIXED -> SWAP BLOCKS
"
 "H) BOTH BLOCKS FIXED -> MOVE FLATTER BLOCK TO CEILING")
- fired:
"E) BOTH BLOCKS CORNER-FIXED -> MOVE MOST-CONVENIENT-BLOCK TO MOST-
CONVENIENT-CORNER"
- active metaphor: STATICS
- moving BLOCK-2 to UR corner (231 305)
+
+
- triggered:
("A) LINES DO NOT INTERSECT -> CONSIDER STATICS METAPHOR"
 "B) NO INTERSECTION OF LINES CLOSE TO THE CENTER -> CONSIDER STATICS
METAPHOR"
```

```
 "E) BOTH BLOCKS CORNER-FIXED -> MOVE MOST-CONVENIENT-BLOCK TO MOST-
CONVENIENT-CORNER"
 "F) BOTH BLOCKS CORNER-FIXED -> SWAP BLOCKS
"
 "H) BOTH BLOCKS FIXED -> MOVE FLATTER BLOCK TO CEILING")
- fired:
"E) BOTH BLOCKS CORNER-FIXED -> MOVE MOST-CONVENIENT-BLOCK TO MOST-
CONVENIENT-CORNER"
- active metaphor: STATICS
- moving BLOCK-1 to UL corner (176 305)
+
+
- triggered:
("A) LINES DO NOT INTERSECT -> CONSIDER STATICS METAPHOR"
 "B) NO INTERSECTION OF LINES CLOSE TO THE CENTER -> CONSIDER STATICS
METAPHOR"
 "E) BOTH BLOCKS CORNER-FIXED -> MOVE MOST-CONVENIENT-BLOCK TO MOST-
CONVENIENT-CORNER"
 "F) BOTH BLOCKS CORNER-FIXED -> SWAP BLOCKS
"
 "H) BOTH BLOCKS FIXED -> MOVE FLATTER BLOCK TO CEILING")
- fired:
"E) BOTH BLOCKS CORNER-FIXED -> MOVE MOST-CONVENIENT-BLOCK TO MOST-
CONVENIENT-CORNER"
- active metaphor: STATICS
- moving BLOCK-2 to UL corner (455 305)
+
+
- triggered:
("A) LINES DO NOT INTERSECT -> CONSIDER STATICS METAPHOR"
 "B) NO INTERSECTION OF LINES CLOSE TO THE CENTER -> CONSIDER STATICS
METAPHOR"
 "E) BOTH BLOCKS CORNER-FIXED -> MOVE MOST-CONVENIENT-BLOCK TO MOST-
CONVENIENT-CORNER"
 "F) BOTH BLOCKS CORNER-FIXED -> SWAP BLOCKS
"
 "H) BOTH BLOCKS FIXED -> MOVE FLATTER BLOCK TO CEILING")
- fired:
"E) BOTH BLOCKS CORNER-FIXED -> MOVE MOST-CONVENIENT-BLOCK TO MOST-
CONVENIENT-CORNER"
- active metaphor: STATICS
- moving BLOCK-2 to UR corner (55 305)
+
+
- triggered:
("A) LINES DO NOT INTERSECT -> CONSIDER STATICS METAPHOR"
 "B) NO INTERSECTION OF LINES CLOSE TO THE CENTER -> CONSIDER STATICS
METAPHOR"
 "E) BOTH BLOCKS CORNER-FIXED -> MOVE MOST-CONVENIENT-BLOCK TO MOST-
CONVENIENT-CORNER"
 "F) BOTH BLOCKS CORNER-FIXED -> SWAP BLOCKS
"
 "H) BOTH BLOCKS FIXED -> MOVE FLATTER BLOCK TO CEILING")
- fired:
"E) BOTH BLOCKS CORNER-FIXED -> MOVE MOST-CONVENIENT-BLOCK TO MOST-
CONVENIENT-CORNER"
- active metaphor: STATICS
- moving BLOCK-2 to UR corner (55 305)
```

```
+
+
- triggered:
("A) LINES DO NOT INTERSECT -> CONSIDER STATICS METAPHOR"
 "B) NO INTERSECTION OF LINES CLOSE TO THE CENTER -> CONSIDER STATICS
METAPHOR"
 "F) BOTH BLOCKS CORNER-FIXED -> SWAP BLOCKS
"
 "H) BOTH BLOCKS FIXED -> MOVE FLATTER BLOCK TO CEILING")
- fired:
 "F) BOTH BLOCKS CORNER-FIXED -> SWAP BLOCKS
"
- active metaphor: STATICS
- moving BLOCK-2 to UL corner (455 305)
- moving BLOCK-1 to UR corner (55 305)
+
+
nil
```

3. Range of problems handled by EstheR

EstheR could only handle problems based on the experiment itself. Until now, it can replicate a subject's actions (as referred to JMP) to make a picture more stable using statics or centering metaphors. There are a lot of details that EstheR cannot handle, for example style. Through the first experiment we saw that subjects had their own style to move rectangles inside the frame (i. e., they were more or less accurate when placing them next to the lines or the frame). A formula might be derived so as to place rectangles considering style, i. e., when moving a rectangle to a corner, this could specify at what distance of the frame exactly, now the program does it at 6 pixels in each direction.

We did not have enough time to play with all the factors that lead the program to decide for an action, and it might be interesting to devote more time to revise the Immediacy formula that derives how "immediate" for a subject a move is.

We also found that there was a kind of "seeing" like "A is to B as C is to D" that we embedded in the rules themselves that might be made explicit very easily as a rule.

We have been thinking in the possibility of inheriting metaphor frames a sort of (is-a gravity statics), but this will produce problems at the level of rules. Common Lisp allows the possibility of building structures from others already

defined which represents a kind of inheritance since the properties of the new structures are defined by the ones of the old structures plus the specific of the new ones. This will give a lot of flexibility to the program and will naturally capture the levels of abstraction of the different metaphors.

Both systems are very subjective, not only because we have used our knowledge to derive rules about other designer's moves, but also because of the degree of consistency of an individual doing the experiment. We could not reflect this either in EstheR, but I do not think it is an important issue in relation to this thesis given that the whole design of the model will reflect my own knowledge and consistency about design, and its character of individuality.

In the first experiment, Disposable Metaphors or Gestaltatron, we saw that subjects developed skills along the process of modifying pictures and learned how to make a picture more expressive of the given quality. For example the last sets of modified pictures tended to be more consistent than the first. This means that same group of answers might be difficult to obtain because answers of the same individual might vary through time. The scores that EstheR uses to decide what rule to fire might, in the end, account for this.

It would be very useful to redo another experiment using Gestaltatron again in which pictures produced by EstheR would be shown to JMP to judge their stability and even allow him to modify them to make them even more stable. If the subject finds a large percentage of them stable or more stable than their initial picture we can consider that we know what stability means for JMP given that specific world. To check the consistency of JMP's judgements about stability of the pictures produced by EstheR, old pictures produced by him in the first experiment might be mixed with EstheR ones. This was something that we did not do because of lack of time, but might be very helpful for testing our findings and building a more comprehensive basis for the thesis.

4. Knowledge representation

In fact we think that EstheR knew a lot of things that are sometimes scattered and reflect our own ability and knowledge about capturing what a subject was intending to do when modifying a picture. We also had to trust what our subjects told us in the second round when they were allowed to modify again the picture that they had already manipulated. Sometimes designers' explanations about their work are very different from what designers really do, and this is very often unintentional, because they try to rationalize what they do, what might be very difficult to do as we hope to have shown.

In general, EstheR knew what a specific subject will do given a specific picture or family of pictures. It also knew how to represent internally a given picture for each metaphor, this is what <metaphor>-s-description does (see Appendix 1 for an illustration of metaphors, and seeing-as and move rules). It knew what to "look for" in a picture, which is embedded in the "seeing-as" rules. Sometimes picking a metaphor implies deciding what action to do. Finally, according to what it saw for each metaphor it had the knowledge to propose a "move" or action. The system also had other more specific knowledge, for example in the statics metaphor it knew the picture corners priority for a specific subject. Most-convenient-corner chose the corner considering non-intersecting positions, proximity, same handedness, opposite up-downness, non-repetitiveness and corner-type priority.

The following is the procedure corner-type priority illustrated in Appendix 1.

```
(defun cornertype-priority (corner)
  (let ((corner-score-list
         '((8 UL LL) (8 UR LL)
           (7 DL LL) (7 DR LL)
           (6 UL FF) (6 UR FF)
           (5 UL FB) (5 UR FB)
           (4 UL FL) (4 UR FL)
           (3 UL BF) (3 UR BF)
           (2 UL FL) (2 UR FL)
           (1 DL FL) (1 DR FL)))
        (score 0))
    (progn
      (map nil #'(lambda (x)
                   (if (and (eq (second x) (handedness-updownness corner))
                            (eq (third x) (intersection-type corner)))
```

EstheR also knew how persistent a subject was with a specific metaphor and how this persistency influenced other factors. It also knew how to stop! In a real world, designers tend to say that a design stops when the client wants to stop it... In our case we found that our subjects stopped because they were very satisfied with a picture or simply because they were tired. EstheR also knew this, and metaphors said "how much they wanted to accomplish a move" and an arbiter decided which move to do.

As the theoretical model we designed shows (see Figure 5. below), we have conceived independent metaphor-packages that have two sub-packages, the "seer" and the "rules". The seer saw the picture and produced a <metaphor>-s-description that was processed by the Left Hand Side (LHS) of the rules sub-package. The LHS then interacted with the Right Hand Side (RHS) that proposed a move or action. The proposed RHS's move altered the picture through again the <metaphor>-s-description.

The computer model implemented (see Figure 6. below), seeing-as rules and move rules produced scores that altered either immediacy or move-scores, and an "arbiter" decided which rule to fire. The arbiter is were I think the main changes will occur for future work. EstheR's arbiter only compared scores and fired the move rule that had the best score. In fact, the arbiter did not have design knowledge at all, and more important, metaphor packages were independent, they did not interact with each other so as to know the other move candidates proposed by the rest of the metaphor packages. It was in the next round of the cyclical process -after a rule is fired and thus a specific design move is done- when metaphors acted on the new picture and thus got to know that move.

Figure 5. Theoretical Model

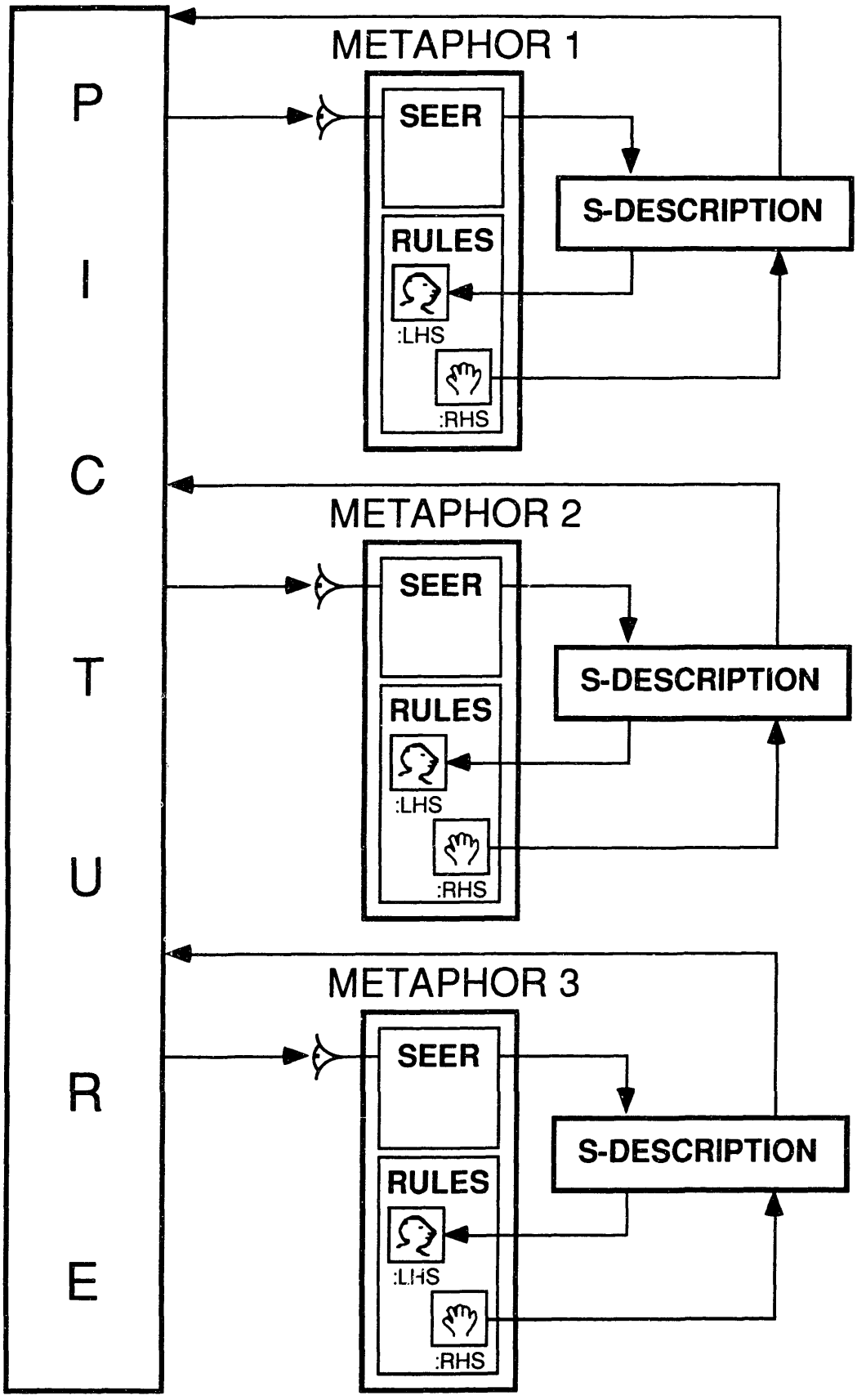
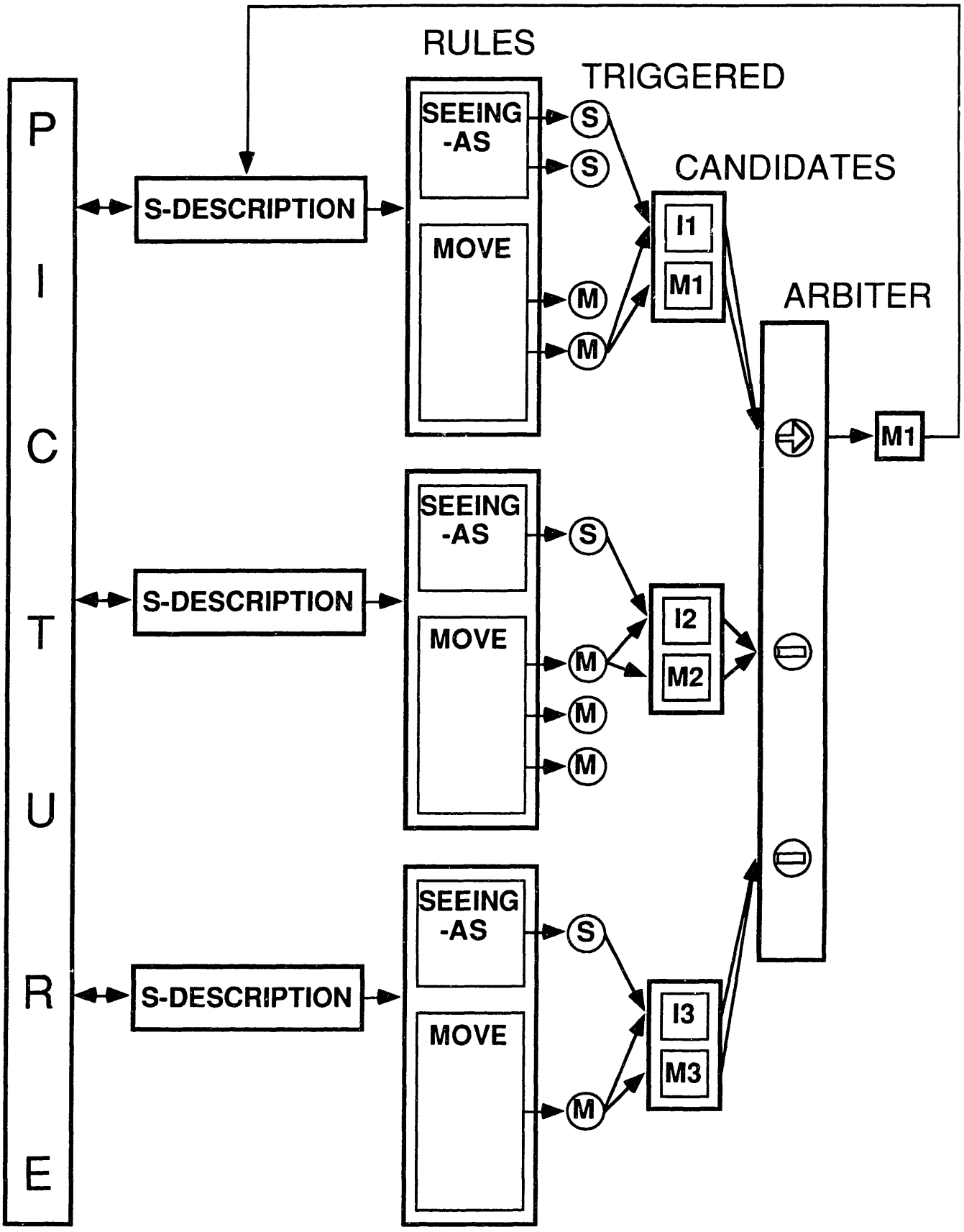
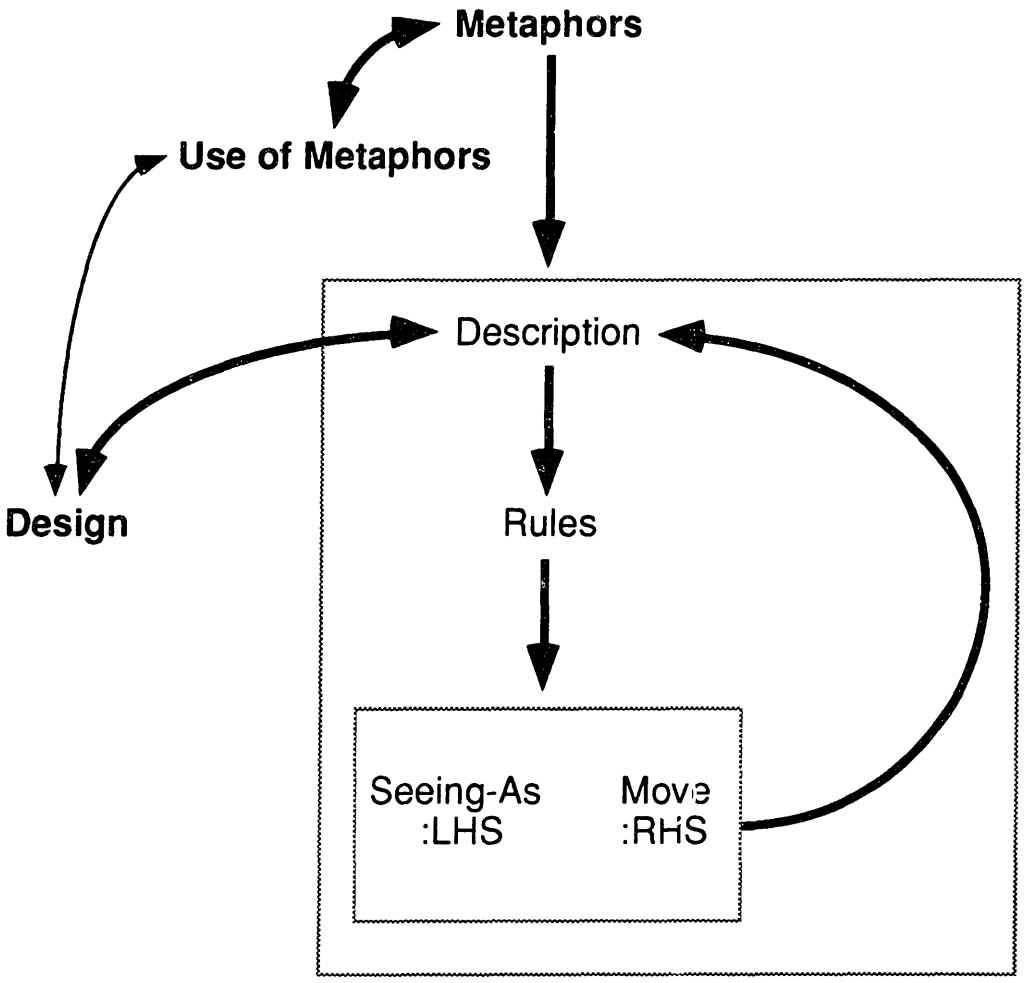


Figure 6. Computer Model



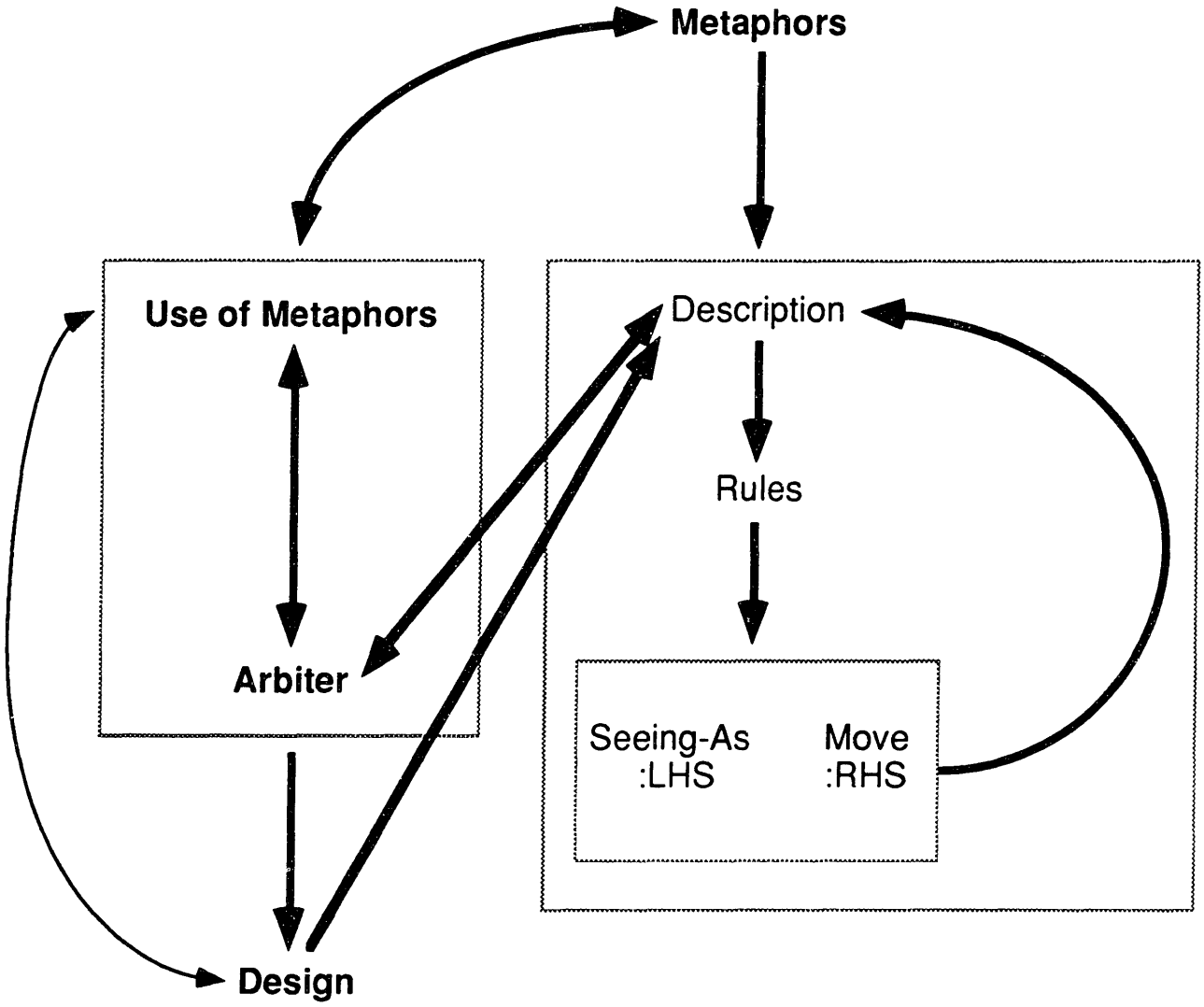
It is interesting to notice how explicit the simplified path diagram of our model is (see Figure 7. below) in showing how metaphors and the use designers make of metaphors directly influence design (note that I will use design meaning EstheR's picture to make a more general statement). For the purpose of this research and for simplicity, metaphors might be thought as independent variables, although the first experiment Gestaltatron was specifically about the tools -metaphors- designers use to "see" and to interact with the world). The use designers make of metaphors might be understood as intervening variables, and the description and the rules as dependent variables. Depending on what part of the design process cycle we are looking at, the design itself can be either an independent variable (in a replicating process) or a dependent variable (in a creative process).

Figure 7. EstheR Path Diagram



The main difference between EstheR's path diagram (Figure 7.) and the proposed one for further research (Figure 8.) is that metaphors are given a chance to interact with each other before a design move is done. The arbiter would then play a decisive role along with how metaphors are used; they will be implemented with certain independence from the metaphor package and with a very strong relationship as a heuristics structure parallel to the metaphor packages (in EstheR the use of metaphors was embedded in each metaphor package, since the rules themselves proposed their score and immediacy). There might be a metaphor for which the arbiter might behave as intervening variable although in general it will be a dependent variable.

Figure 8. Proposed Path Diagram



It is interesting to note the new model's cycle or loop between description, arbiter and design as opposed to EstheR's linear relationship between design and description. The "double arrow" shown between arbiter and description represents the "loop" that will take place just before a design move will be accomplished by the arbiter. As I pointed before, each metaphor move candidate will be evaluated by the rest of metaphors giving the arbiter a chance to make a more complete design move. Also, note that the bi-directional arrow or relationship between design and description of EstheR's path diagram has been substituted in the proposed model by a unidirectional arrow from design to description, since description is the "medium" between the program and the design or picture, and it is now the arbiter that will control the flow between metaphors' moves and design. I think this is very important to make a more realistic and interesting model since designers are generally influenced by more than one of the worlds or metaphors they use at each design move. This will imply some substantial changes in the conflict resolution strategy of EstheR.

5. Limits of the model

In our experiments what has always been crucial has been the conflict resolution strategy. We needed EstheR to know about when to allow repetitions of old moves, the persistency of a subject in favoring a metaphor, how immediate was a move for a given subject, etc. The program also decided what to do according to the priority of rules that might be opposite to their immediacy.

In EstheR we defined Immediacy as a function of the scores given by the seeing-as rules, the score given by the move-rules, the sensitivity to history and the tendency to favor a given metaphor. We defined the sensitivity to history as function of the number of times the metaphor had been active and a measure of persistency or reluctancy of the subject to switch metaphors. We also limited the number of times any rule would be used consecutively to 5 according to the example where a single rule was the most repeated from the experiments we carried out. To find the right scores for the subject we studied took a very long trial and error process, but helped us to modify our formula and realize the relationships between all the factors. (See the Appendix 1 for the

implementation of metaphors, seeing-as and move rules, their representation, and part of the formula we designed for Immediacy.)

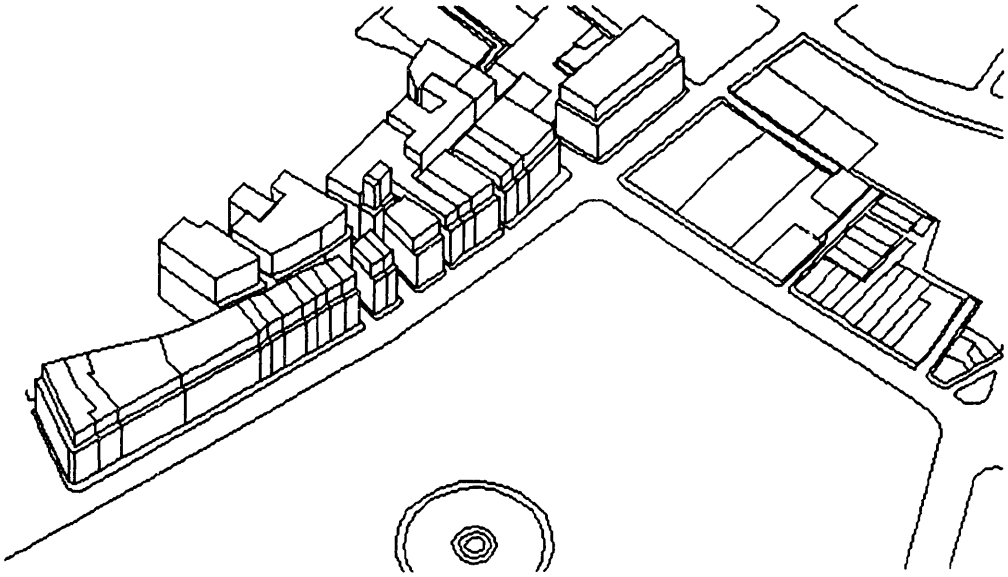
In my opinion we managed to implement in EstheR the concept of Switching metaphors by the persistency factor, and the concept of Sleeping metaphors by allowing each metaphor a second round after a number of actions done by a metaphor. It was harder to deal with the concept of Overlapping metaphors which means that two or more metaphors might happen literally at the same time, as I pointed out earlier. It is clear that in EstheR all the metaphors propose actions, but only one action is taken at each time. In fact if two metaphors propose the same kind of move, this gets a higher score which makes it more probable, and if this might be seen as a kind of overlap it does not really simulate a real overlap or interaction. This is one of the main changes I think would have to be introduced in future versions of EstheR.

Dealing with design knowledge implies a great subjectivity and thus, the character of generality and completeness of our project might be questionable. We have always been very conscious and deliberate about this, although I think we have made a very big effort to find a good representation of the knowledge involved in each metaphor. I think the revision of EstheR's conflict resolution strategy so as to make it more detailed and accurate and maybe to make it function of the individual to replicate might bring significant insights about the design process. This point is particularly addressed by DeaRB (Designing Rules, Boston, exposed in Part 2) with the concept of "hats" developed in this thesis.

It is also very difficult to derive rules about design and sometimes we were forced to be very simplistic so as to make things work. It was also very difficult to find a good structure for the knowledge base. Our models are not really structured as trees they might be closer to nets. We have tried very deliberately and literally to implement the conceptual idea of "seeing-as" or the intentionality of designers when looking at the world. Evidently our models imply a lot of assumptions that might be very biased (by our own "more or less" Equivalent metaphor), but at least we think they allow to better understand design and the limitations of knowledge-based computer systems in trying to produce knowledge-based CAD systems which might be one of the immediate applications of our research. I hope that the inherent bias might be reduced in a

more design oriented or personal approach for the proposed thesis experiment DeaRB. Paradoxically I think the indispensable final character of generality that a thesis requires will be better reached than with EstheR. Also I think that the concept of Equivalent metaphors that we revealed in Disposable Metaphors or Gestaltatron will have to be better exploited in future work for better completeness.

Part 2. Designing Rules, Boston (DeaRB)



1. Metaphors, rules and regulations

Before entering into the experiment Designing Rules, Boston (DeaRB), I would like to state some definitional questions about metaphors, rules and regulations that are important to relate DeaRB to the two anterior experiments: Disposable Metaphors and Esthetics Replicant (exposed in Part 1). As I will explain later, Designing Rules is an experiment dealing with real design problems and design regulations. When dealing with real designs and regulations as set of rules acting upon or modifying designs, the relationship between regulations and metaphors the way they were defined in Disposable Metaphors or Gestaltatron, might be unclear. The following states the framework of DeaRB in relation to Gestaltatron and EstheR, and also it further develops the concept of metaphor as defined in Gestaltatron in the context of designing regulations.

An important quality of Gestaltatron and EstheR was that as theoretical experiments design problems were kept as simple as possible with a higher degree of abstraction than real design problems. Thus, metaphors were clear. In reality and as noted in Gestaltatron, the way designers "see-as"[1] designs does not tend to be as clear as in the experiments because reality is more complex and because usually different metaphors tend to "co-exist" with each other "polluting", up to a certain extent, the clarity of designs. In my opinion, this co-existence and "competition" of metaphors might either produce bad designs or exceptional ones, and it really depends on the designer to decide the degree of complexity of the design "moves" in relation to the metaphors that produce them. Gestaltatron proved that the degree of completeness of a metaphor is directly related to the success of the design, at least in the experiment and under the designer point of view. I would like to generalize this to any design, though I think exceptional designs are characterized by the existence of limited intentional and internal design contradictions, as I will point out in the conclusion of the thesis and which might be the subject of another thesis.

According to David Cooper's view concerning "the primacy of metaphor"[2], metaphorical talk is temporally and logically prior to literal talk. In this sense,

---

1 D. Schön and G. Wiggins, Kinds of Seeing and Their Function in Designing

2 David Cooper, Metaphor

language would be fundamentally metaphorical in character. As Nietzsche said, "Truths are illusions of which one has forgotten that this is what they are - metaphors that have become worn out and without sensuous force; coins that have lost their face and are considered, no longer as coins, but as pure metal". It is in this sense that I will argue that design, as language, and a set of urban regulations, as an extension of design, would be fundamentally metaphorical.

On the other hand, the concept of "dead metaphor", that is metaphor that has lost its meaning, that has been absorbed by the common practice of language and that has become a standard and a conventional way to refer to something, has a strong relevance to regulations. For example the concept of "street wall"[3] of Boston's Zoning code is basically a metaphor to refer to façade alignment and it might be a dead metaphor. Considering limit heights, one could think about a good city form as one which is limited to a certain height or has a kind of "roof". There might be several dead or just forgotten metaphors that might have been the basis from which creating these regulations. For example the idea of a good city as a healthy and equitable city, a city where the sun is able to reach the ground and to bathe every body's house might be another metaphor from which zoning set backs from street wall might have been created. The approach of the thesis to regulations is precisely to recover the meanings of this dead or "amnesiac"[4] metaphors and build sets of rules directly from them in a heuristic manner. In my opinion, what I call the purpose or the "spirit" of the regulations can be better reflected in the rules if these are derived from the original metaphor and according to the "primacy of metaphor" thesis.

I should now go one step further and argue that, to a certain extent, a set of "constitutive"[5] rules, designed with the same goal, might be considered a metaphor. Talking about speech acts, Mac Cormac criticizes how John Searle's "constitutive rules" relate to metaphors in the sense that every metaphor can be paraphrased exactly in a literal expression. I do not want to go as far as Searle, but I think the idea of constitutive rules as metaphor might very well extend the definition of metaphor as was defined in the first experiment Disposable

---

3 B.R.A., Downtown Zoning, Midtown Cultural District Plan

4 David Cooper, Metaphor

5 Earl Mac Cormac, A Cognitive Theory of Metaphor

Metaphors or Gestaltatron. I would like to think about a specific set or sub-set of regulations as a metaphor.

Finally, I would like to point out an issue about sets of metaphors and how these sets might represent different design paradigms. There are metaphors that might be grouped according to several criteria or according to different "worlds"[6]. In this way a designer might use different metaphors than a lawyer to "see" the world and act upon it, just as different designers might use different metaphors. I would argue that this metaphors can be grouped into sets of metaphors or metaphor packages that will represent the different paradigms. Gestaltatron introduced the concept of Equivalent metaphors and I would like to extend it in the sense that there might be more or less Equivalent paradigms with more or less completeness in their equivalence. In relation to regulations, paradigms and their equivalence becomes a fundamental question since "like all disciplines, architecture and law each have distinctive values, methodologies, and semantic practices peculiar to themselves"[7]. Richard Lai quotes Ada Huxtable who says: "The problem with law and [...] the quality of the design involved is that such judgments cannot be quantified -they are unavoidably subjective, although responsible judgment rests on a very specific set of standards and their interpretation...". Part of this thesis will show how designing regulations requires a strong fidelity to the paradigm from which they are thought. I mean for example, the specific zoning rules that DeaRB, the following design experiment, will implement respond basically to design considerations and I will show how the way the Zoning code is written, that is, using an equivalent world or paradigm -legal paradigm- implies a loss of any design knowledge and consequently is unable to solve certain cases or instantiations of the regulations. I will show how a design heuristic approach to regulations will be able to better capture the "spirit" or purpose of the regulations than the traditional legal set of rules.

---

6 Donald Schön, Designing: Rules, Types and Worlds

7 Richard Lai, Law in Urban Design and Planning. The Invisible Web

2. Purpose of Designing Rules, Boston (DeaRB)

Designing Rules, Boston (DeaRB) implements the Boston Redevelopment Authority (BRA) Downtown Zoning Code for the Midtown Cultural District (MCD) of Boston. My purpose when designing DeaRB has not been to provide a fully complete zoning tool because I wanted to keep the program as simple as possible to exemplify clearly the way I think computers have to be used in design under the frame of metaphors that has already been exposed earlier in the thesis. In any case the possibility of designing a more or less complete zoning tool exists, as I hope DeaRB illustrates, and the considerations about how to do it will be discussed later when talking about further steps that DeaRB or similar projects to DeaRB might take.

DeaRB also studies how regulations as a set of rules influence design and how to design the rules or the set of regulations to produce better designs, that is, designs that "better" express the purpose of the regulations, if we consider regulations as a specific set of knowledge about design. The thesis studies the kind of knowledge that allows to design and decide what is a "good" environment or the kind of knowledge involved in regulating a design to make it more expressive of this given quality, to put it in terms of Gestaltatron (the first experiment).

On the one hand, the character of "objectivity" and precision of computer tools allows to study the design paradigms themselves testing their completeness and validity of their principles -metaphors or sets of regulations- when trying to act upon the physical environment. On the other hand and in relation to the implementation and design of a set of regulations, the computer model allows an exhaustive control of what it is being proposed and thus it allows to assess the impact and accuracy of the rules in relation to the "purpose" or "spirit" of regulations. It is very useful to detect the cases in which the rules do not apply, and the study of such cases might bring revealing insights about the proposed rules and design ideals.

One of the most relevant points of the thesis and specially of designing rules to regulate design is the challenge consisting on the possibility of creating more general but at the same time more accurate rules, rules that will solve more

satisfactorily a bigger number of special cases in which the traditional regulations fail in giving appropriate design solutions. As I explained earlier in Part 1, this can be done by preserving the design approach to regulations over the "legal" approach, using heuristics based in design concepts expressed as set of designing rules.

Finally it is also important to approach the idea of knowledge representation in design. Very often the success of computer Expert Systems lies precisely in finding the right representation of the knowledge they try to encompass. In my opinion, knowledge representation is a special fundamental issue in design and it has not been very much addressed in computer science. The future of CAD systems and specifically the difference between CAD systems as drafting tools and CAD systems as designing tools mainly resides in this issue. In the field of urban regulations the representation of knowledge is especially important since I think good regulations should be expressed through drawings with higher or lower degrees of abstraction and iconicity, but they have to be ultimately drawn. In the section about how DeaRB works I illustrate how I understand this point and I deliberately use diagrams and drawings produced by DeaRB.

3. Inside DeaRB

31. Strategy of DeaRB in relation to the CAD platform

The current programming languages or platforms like Common Lisp and C Language are very flexible in terms of programming techniques, but they still do not provide the adequate graphic primitives to design a program capable to handle real design situations. On the other hand, when there is the need of a designer's interaction with the program and the design, then "open architecture" CAD systems are indispensable to allow create specific commands and procedures on top of the ones provided by the system. This was an important factor to decide what platform to use, since my purpose was not to reinvent how to draw or display a line in the screen or to design basic drawing primitives. I considered three basic platforms, AutoCAD by Autodesk, Arris by Sigma Design, and the Personal Iris graphics environment by Silicon Graphics.

The latter has the advantage that the designer can control and design all the graphic environment tailoring it to his/her needs. This flexibility is at the same time its disadvantage with respect to the other two platforms, since everything has to be programmed before a line can be drawn in the screen, like screen refreshment, control of the stack, menus, pointing devices, etc.

The Arris platform is a very powerful CAD environment, but the syntax of the programming language, Sigmac, based in C language, is sometimes obscure. I needed a clear programming language, under a syntactic point of view, so as to be able to express design knowledge clearly.

I decided to implement DeaRB on top of AutoCAD by Autodesk using the AutoLisp, dialect of the Lisp language. Although AutoCAD is quite limited to build a sophisticated design package, not only in terms of processing speed but also in terms of data management, flexibility of internal commands and efficiency, it has turned to be an adequate platform to develop a program like DeaRB or the first steps towards a more complete regulating/designing tool. As I will point out later, it might be necessary to change the platform for future extensions of the program. Also, the fact that AutoCAD is one of the most extended CAD systems and also one of the most affordable, my contribution might turn to be significant to make the point that there is no more the need of sophisticated and expensive systems and computers to really change the traditional way designers think and work.

The principal issue about data management of DeaRB or any, I might say, design Expert System is to decide how the different information is going to be handled, stored, changed and accessed. AutoCAD does not provide a lot of possibilities to solve this issue. On the other hand, I really think it is important that the graphic results of a regulator or DeaRB, in my case, would be able to "stand alone" or behave like regular computer designs that can be manipulated by all graphics primitives of the system, AutoCAD in my case. It is in this way that the need of thinking separate drawing and knowledge and information databases becomes very strong. The solution I adopted was to use AutoCAD blocks to store alphanumeric information about the lots that were represented in AutoCAD by polylines (that also have the advantage that they can be exported to other CAD and GIS -Geographical Information Systems- platforms). I could

have written the information to another file, but it is always slower to read the disk than the drawing file itself. I defined the block containing an AutoCAD attribute for each information field like ward, lot number, street name, street number, etc, allowing to retrieve at any time the information about a specific lot. This is not really the way blocks and attributes have been designed to be used in AutoCAD, since they allow repetitive instantiations of graphic elements like chairs and then retrieve statistics about the number of specific graphic elements (chairs in this example) and their attributes like price, quality, location, type, etc, which values might vary (black chair as opposed to red chair). In DeaRB blocks exclusively contain alphanumeric data.

The important difference between how blocks are designed in AutoCAD and how DeaRB uses them is that, in principle, once a block is inserted in a drawing, there is no drawing entity that will need to know about the block or vice versa, since the block already contains the drawing entity itself with all its attributes. The consequence is that usually AutoCAD blocks are used as rigid drawing entities, difficult to change and designed to be reproduced almost infinitely throughout the drawing document, only allowing to change the existing attribute values of each instantiation. I think the nice aspect of DeaRB is that drawing entities are kept absolutely independent from their attribute blocks and both of them can be changed independently of each other. As I pointed before there was no sense to define a lot as a block since all blocks are different, but attributes are always the same, only their values change for each lot. In order to achieve this independence, blocks need to know what lot is the one they are qualifying. For this purpose I used AutoCAD *handles* that are unique identifiers for each AutoCAD entity inside the drawing database. Then the block itself has an attribute field in which it stores the lot polyline ID to be able to refer to it when regulating the lot. In this way, there are fields or attributes that a block uses for its own identification purposes (lot's polyline handle or façade point handle), fields that the program knows and fills automatically, as soon as the user sets a specific parameter (like the zoning code by which DeaRB derives the zoning parameters), and other fields that the user needs to fill in him/herself and are specific to each lot (like the zoning code). I think this is a very flexible and innovative way to use blocks, since it allows the kind of flexibility and efficiency that designing programs need.

Another interesting aspect of DeaRB's strategy is the possibility it offers to convert a CAD system (AutoCAD in this case) into a GIS (Geographical Information Systems). Typically Geographical Information Systems are characterized by their ability to geo-reference any sort of data, they are a "powerful set of tools for collecting, storing, retrieving at will, transforming, and displaying spatial data from the real world for a particular set of purposes"[8]. Although by far DeaRB is not a complete and efficient GIS, at least it opens a way of how to use a CAD system as a basic or single purpose GIS. The way DeaRB has been designed also opens the possibility, in the future, to implement other features typical of GIS, or at least to link CAD to GIS, since I implemented typical attribute fields of such systems like ward number, lot number (used by the Assessors Department of the BRA), street number, street name, lot use, etc. GIS usually use these kind of attributes to geo-reference any data, like census data, cadastral data, historical data, etc. This also offers the possibility to define different blocks of attributes for each lot so as to be able to keep different databases for each lot. I will expand this part later in the thesis when talking about how to implement the different metaphors to regulate the lots. Now I would like just to point out that this strategy offers the possibility of regulating a lot under different scenarios independent of each other or even in such a way that some relationships could be established between these.

Finally, the "knowledge" of the system is contained in a Lisp code artificer of data interaction. The code is invoked by the user to accomplish the specific tasks of binding data to lots (defining lot attributes), modifying and consulting attributes, regulating the lots, changing specific parameters, etc, either by typing the specific commands at the AutoCAD prompt or by using the pull down menus I defined for the purpose. The issues related to the field of Artificial Intelligence and specifically to knowledge representation will be discussed in the next section illustrating them with DeaRB itself.

---

8 P.A. Burrough, Principles of Geographical Information Systems for Land Resources Assessment

32. What DeaRB does and knows

To implement regulations or rules about design, several assumptions and scenarios have to be stated so as to be able to built from them. Regulations have to be abstract enough to encompass different situations under which the real solutions or instantiations of the rules might vary substantially. Since regulations are very often designed based on specific design problems and generalized a posteriori, there is always a margin of "error" or inaccuracy related to very specific real situations in which rules do not apply properly. In these real life cases negotiation is very often the way out of the problem. In the case of designing a program that will implement regulations one has to state very clearly the frame of action and the limitations of the applicability of the rules, and the designer can always design the "regulator" so as to ask the user what does he/she wants to do in a "not computable" situation.

I will develop next, the two levels of decisions that I think should be made when designing a "regulator". The first is about the set of rules to replicate or to implement, that is the kind of designs the program is going to produce and according to what criteria. The second level of decisions is more intrinsically related to the implementation of the rules themselves. The regulated "object(s)" have to be studied so as to define what will be the set of topological and geometrical relationships that the program will have to know so as to be able to relate objects and their parts to take the appropriate design decisions. This second set of decisions is decisive so as to reach a more "complete" system. When designing the program itself it is very difficult to encompass all the possible exceptional cases that one wishes to make the program responsible for and it is especially important to design the program with modularity or using object oriented techniques, that will be explained later, so as to be able to modify it easily when these cases appear.

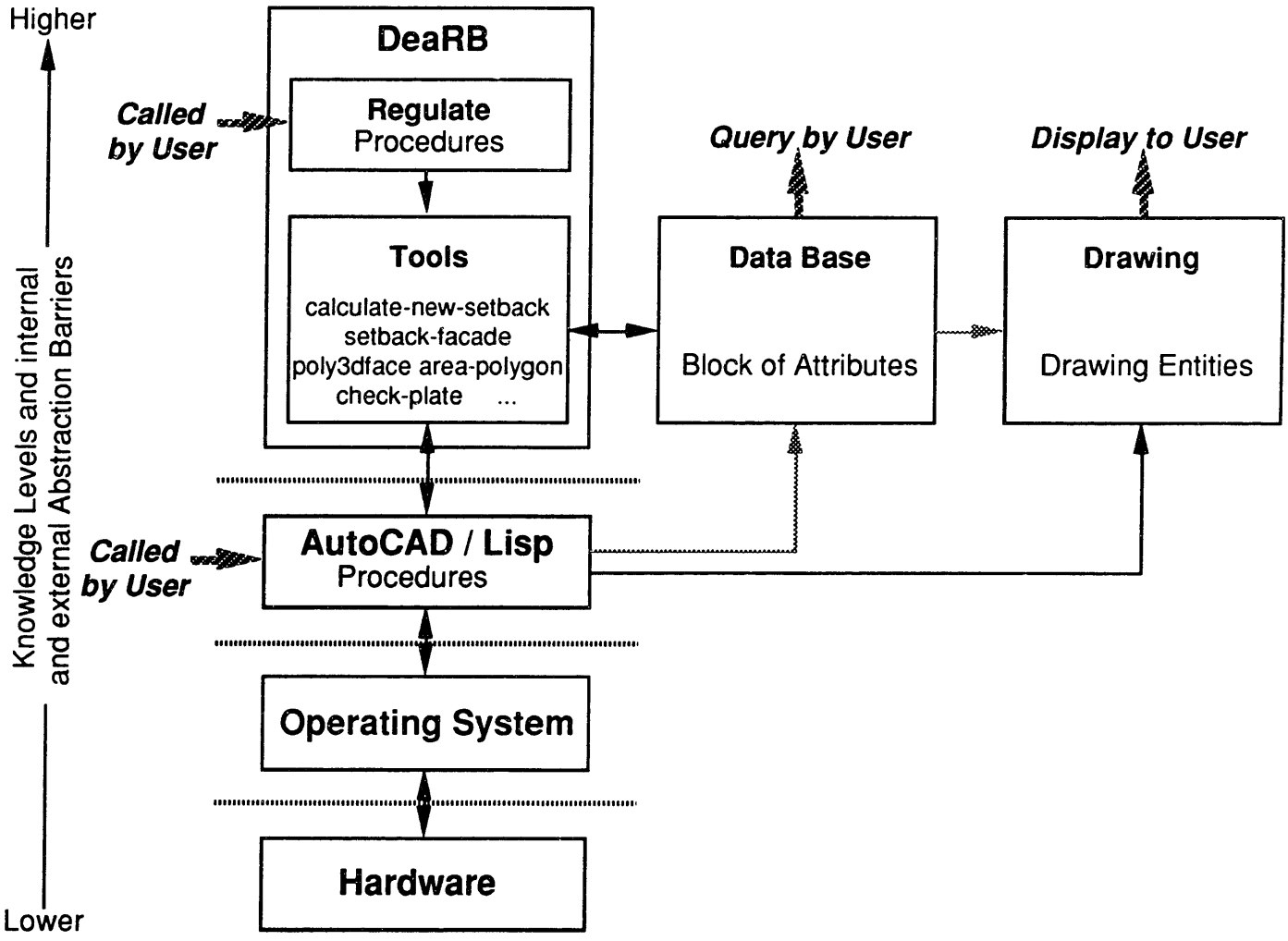
The program knows things related to different "levels of abstraction" or according to different "abstraction barriers"[9]. I mean that the same way the user does not need to know how the program works internally to be able to use it, the programmer does not need to know specific things about the computer or the

---

9 H. Abelson, P. Sussman, Structure and Interpretation of Computer Programs

programming language itself, like how the processor internally binds values to specific parameters, etc, so as to be able to design a program to do specific tasks. In my opinion, the theme about abstraction barriers is a key issue when thinking about further steps that CAD systems will take in the future and I will expand it later in the thesis. Now it is important to bring out the concept since the two anterior sets of decisions that should be made when designing a "regulator" and, I claim, any program related to design are directly related to the concept of abstraction barriers and to the specific knowledge embedded in the program. Figure 9. below illustrates the different levels of knowledge associated with abstraction barriers between platforms and, inside DeaRB, between procedures; it also shows DeaRB's interface strategy.

Figure 9.
Diagram of DeaRB's relationship with the different platforms and levels of knowledge



In relation to the first set of decisions, DeaRB knows the MCD Downtown Zoning -in the case of BRA's regulations- which is expressed as a set of conditional rules (see procedure *zoning-parameters* in Appendix 2) applied before regulating any lot, and used by the system to derive the building form. The program also knows how the zoning parameters of each lot (as-of-right height limit, maximum height limit, set back from street wall, as-of-right floor-to-area ratio -F.A.R.-, and maximum F.A.R.) relate to each other (see Figure 10. in section 33 of this Part). These zoning parameters depend on the metaphor or rules set implemented that is the BRA in this case, as I pointed out at the beginning. Finally, DeaRB also knows how to build according to these parameters. Zoning parameters by themselves are not enough to derive the building envelope; specific decisions need to be taken for instance in relation to floor heights, use and mix of uses, criteria about maximizing the built area, etc. At this point I would like to introduce the concept of "hats" or the "kind of characters", in a theatrical sense, that the program would behave like. When designing DeaRB I ran into the need of defining a frame from which I would take design decisions inside each metaphor set. This is a very fundamental aspect of design which I think is very nicely introduced into DeaRB and up to a certain extent was indirectly present in EstheR, although "hats" were not explicit and were embedded into the metaphor packages themselves. Specially in planning hats are a very useful and classical metaphor to classify, generalize and dramatize certain attitudes towards decision-making. For example to better understand a specific situation one can analyze it under different perspectives, the one of a rational planner, a traditional designer-planner, an advocacy planner, an economist-planner, etc. In DeaRB I implemented the hat of what I called "unscrupulous developer" because it tries to optimize the number of office floors in relation to residential floors, but without loosing any F.A.R., the specific details of the implementation will be explained in the next section.

All of these considerations are proper to a specific professional domain, the planning discipline, and these domains usually build from either "lower level" knowledge domains or complementary or basic domains like social science, economics, mathematics, geometry, etc. It is in this sense that when writing a program for designing it is not enough to just build in the specific domain of interest, but one has to build also in lower level domains, for instance the geometrical, topological and logical domains, and the more design oriented

future CAD packages will be, the less designers will have to build in these "lower level" domains, that is more abstraction barriers will be built into the future CAD packages. The question will then be what the appropriate abstractions are to build a CAD system. It is in this sense that the second set of decisions embedded in DeaRB becomes important.

In relation to the second set of decisions related to the implementation of the rules themselves or the regulations, DeaRB knows how to associate a specific drawing entity (or AutoCAD polyline) to its specific information database (or AutoCAD block of attributes) and vice versa, as I explained before in the strategy of the program. It also knows how to triangulate a polygon so as to be able to build surfaces (AutoCAD 3D faces) indispensable to produce rendered images, how to set back the façade of a lot in both cases where lots have neighboring walls or when they are isolated, how to find the center of gravity of a polygon and how and where to draw the entities (polylines and 3D faces).

DeaRB also assumes that the concept of street wall is higher priority, consequently, it will always build up to the as-of-right height limit, but if a set back plate is to be built, then it knows that it can start the plate lower than the as-of-right height limit, as if there was a cornice higher than the last floor of the lot from which last floor the new set back plate will start (see Figure 15. in section 34 of this Part). This kind of knowledge might remain a function of the "hat" that in this case wants to optimize the number of floors built while maintaining the "cornice" or street wall height. It is interesting to note the compatibility or equivalence of metaphors and hats in this case since in a heuristic approach it will also make sense to preserve the height of the last floor (if there is no need to increase the first floor height or the other floors height) and build the set back plate on top of it putting the cornice of the building at the mandatory height independently of the set back plate.

Specifically in relation to BRA's regulations, DeaRB knows that a given façade of a lot might not be a strait line, thus it knows how to recalculate the new set back of the lot to match the desired area for the upper set back floors. This last kind of knowledge is somehow higher level than the rest since it builds from it and has a higher degree of sophistication (see procedure calculate-new-setback in Appendix 2).

Thus, designing programs have to be built upon internal (low level) and external (high level) abstraction barriers so as to be able to change them with the minimum modifications and to better express and relate to the design quality of the problem. For instance, when calculating the new polygon for the set back lot at the specified height, if the set back is bigger than a possible step back or indentation of the contour of the lot, then negative areas might appear in the new polygon. To solve this, I could have modified the procedure calculate-new-setback itself, but I designed an independent procedure, check-plate that checks if the new building plate has bad or negative areas before drawing it. This has the advantage to speed up computation time because it does not interfere with the rest of calculations to derive the new set back which are recursive and thus require a lot of processing time. The other advantage of this strategy is that in different cases than the ones existing in the MCD (i. e. when trying to implement a set of regulations for a different city) it might be necessary to check other polygon conditions (i. e. to reduce façade step backs or indentations, to check for minimum distance between contiguous towers in neighboring lots, etc) which can be just added to this procedure without having to introduce major programming changes. This is one example of the strategic power of what is called object oriented programming techniques that I think need to be strongly related to the design world, specially in future CAD systems, as I will point out later in the thesis.

Finally, there are other kinds of things that DeaRB knows that are even lower level than the ones exposed before. For example, it knows the heights of the different floors (ground floor, office floor and residential floor) that can be modified for each lot according to its use needs. DeaRB also knows for each lot where is the façade and how is the façade, if it is just a line or a succession of segments or if the lot is isolated. This last part demonstrates how important low level knowledge might be since just by allowing lots to have a façade with more than one segment and not being necessarily isolated complicated enormously the algorithms as procedure *setback-facade* and *calculate-new-setback* denote. Another low level knowledge implemented in DeaRB is how to "order" the list of vertices of the polygon lot and its façade so as to give the right list to the other procedures since they expect a clockwise list of points. DeaRB verifies the list and changes it if it is not clockwise ordered.

In relation to the database, DeaRB knows how much area is built for each lot and this number can be later managed independently of the lots. This allows to do some statistics about built area in the MCD and the possibility of transfers of air rights. For example for historical buildings it calculates how much F.A.R. is not built so as to be able to sell it or use it in other lots by the same owner or the B.R.A. itself. The underlying metaphor would be to keep the city with a "deformable" ceiling rather than a fixed one, preserving built volume over the MCD. It might be even possible to "export" air rights to non-equivalent areas establishing compensation factors like price of land to make a fair export.

33. Range of problems handled by DeaRB

DeaRB handles only five zoning parameters of the MCD Downtown Zoning Plan and it knows how they interact (as-of-right height limit, maximum height limit, set back from street wall, as-of-right F.A.R., and maximum F.A.R.). These are the most significant zoning parameters and other specifications were not considered to keep the design problem simple. For instance, DeaRB does not know the regulations about shadows and their impact over Boston Common and Public Gardens.

In relation to the zoning parameters handled, DeaRB only allows for one set back although regulations allow several set backs in function of height and for specific cases. This was not implemented for simplicity questions and because it can only happen in very few lots. The program can only handle two heights associated to two floor-to-area ratios, as-of-right and maximum, although there might be some cases in which depending on negotiations, a higher height might be allowed as well as a higher F.A.R than the maximum ones. Also, as I will point out in section 5 of this Part, there are no use considerations which means that there is no possibility to specify bonuses or limitations of zoning parameters according to use, as the Zoning code foresees.

DeaRB also handles the maximum floor plates defined in the Zoning code (25,000 square feet for the MCD, section 38-19) and minimum floor plates (5,000 square feet) were introduced to avoid impossible plates. These can also be changed depending on specific considerations as use of the office or residential

floors. In the picture below some lots do not have set back plates since their area is less than the minimum. Note that an "unscrupulous" developer could buy adjacent lots so as to have a higher lot area and to be able to build a set back plate.

The way DeaRB is designed, it only knows what each specific hat will do under each of the implemented metaphors. That is, DeaRB builds as an "unscrupulous developer" under B.R.A.'s metaphor or set of rules. When building a lot, DeaRB optimizes the number of office floors in relation to residential floors. It tries to build as much offices as possible but using all the possible F.A.R., as explained in the next section. This is the kind of design problem that DeaRB can handle given a lot.

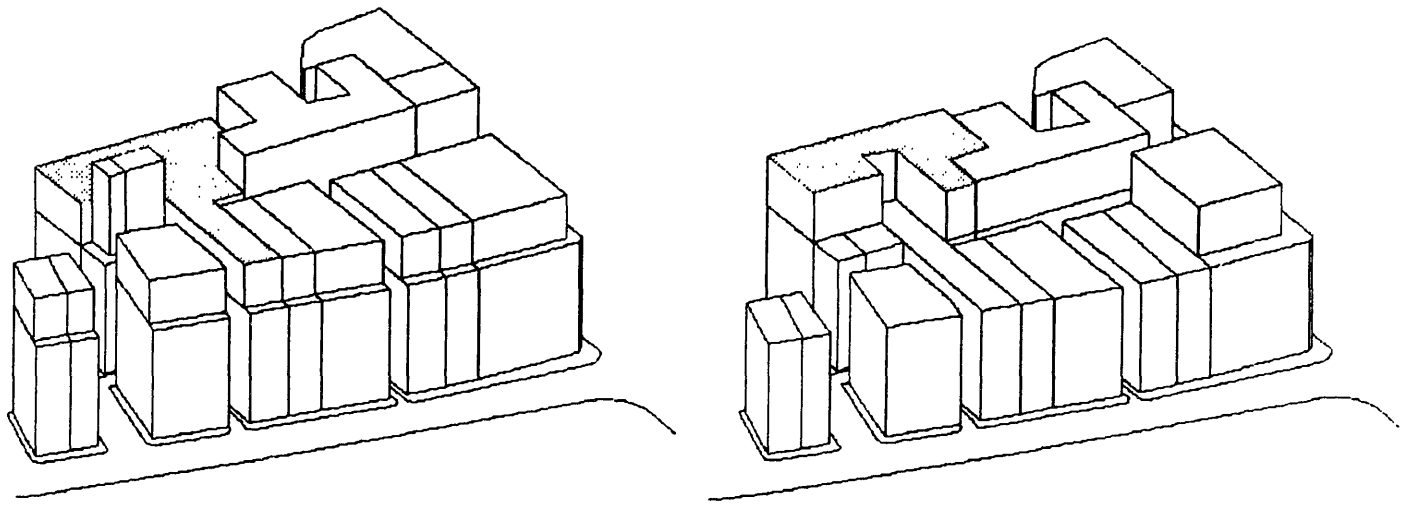
In relation to the implementation , DeaRB can handle almost all lots. As I already mentioned before, lots might have different kinds of façades (a strait line in neighboring lots, several segments step back or with indentations) or they might be isolated blocks. What DeaRB does not handle is the few cases in which a lot has two different façades, for example facing a street on one side and having a back street on the opposite, or special cases in which a lot has access rights to a principal street through a kind of "corridor" but where the main area of the lot is in the back of such block, like Astor Theater in Tremont Street. In this cases when DeaRB tries to regulate the lot, it sets back only the portion of the façade to the street and tries to recalculate the new set back in function of the new area of the plate only moving the façade segment of the lot giving a non-satisfactory solution as shown in the pictures below (see right picture of Figure 10. in which the plate has a pronounced indentation due to the façade).

Note also that if the lot has two non-contiguous façades then the more restrictive regulations are applied and this is why Astor Theater has a height lower than its neighboring lots in the back part of the lot, corresponding to the rear façade. As I mentioned before DeaRB does not handle this situations satisfactorily.

Figure 10.
Screen bit maps of DeaRB's regulatory work on Astor Theater block

"rough envelope"
Only heights limit applied

"maximum envelope"
All zoning parameters applied

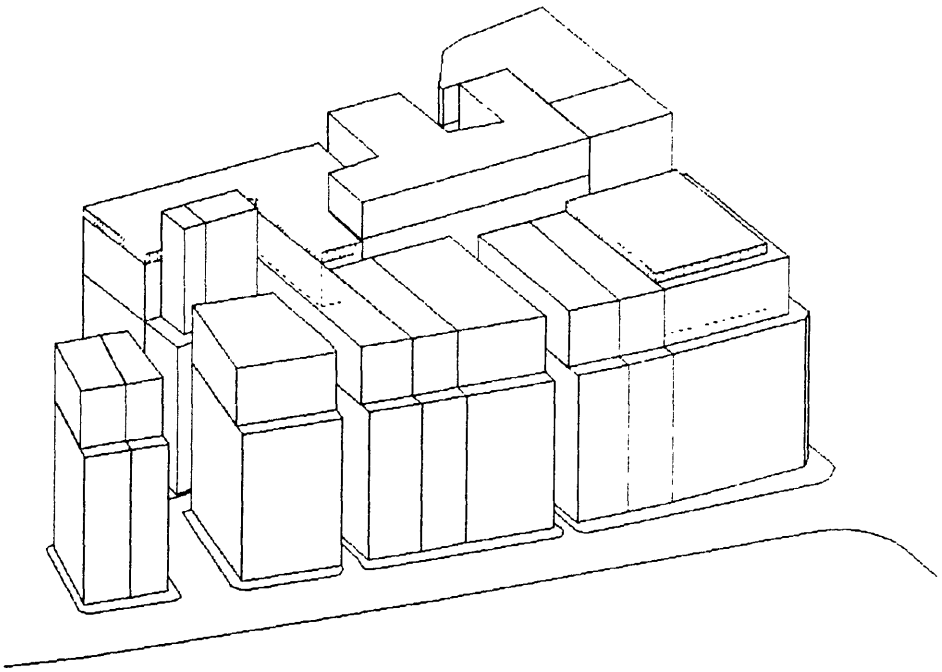


DeaRB can also build the new plate according to the specified area when possible. That is, if the maximum height allows to have a higher building when using the maximum plate allowed by the regulations (25,000 square feet), it is possible to tell DeaRB to build a smaller plate just by reducing the maximum area of the new plate. This is a nice feature since it is possible to compare the solutions and decide which one to use. What DeaRB does not allow is to build several slender towers on top of the block and this might be included in other versions of DeaRB, as I will explain later in section 4 of this Part. Note that in the above example (Figure 10.), the picture in the right as referred to maximum regulations the regulated buildings are not built up to the maximum height because the plates are smaller than the minimum plate defined as 5,000 square feet.

Another nice feature of DeaRB is that when calculating the new F.A.R. for the set back plate, if there is still enough height, it rounds the F.A.R. to the immediate superior integer and considers it as the number of floors for the set back plate or "tower" so as to build all of the floors with the same area. This means that sometimes DeaRB will propose plates higher than the limit height by an amount always lower than the height of the lower floor (residential floor). It considers that since the set back would be bigger than the minimum zoning set

back there would be the possibility of negotiation with the B.R.A. specially in the case of an unscrupulous developer (see Figure 11. below and the marked lots). Since this is applicable to all situations, in the case of isolated blocks the amount of façade is so big in relation to the block that the different set back is very small to get a "bonus" height. It seems unfair to allow such set backs in this cases and DeaRB allows them. The solution might be to implement some rules similar to the ones proposed in the heuristics metaphor, that is considering street ratios or proportions, as I will explain later.

Figure 11.
Super-imposition of "rough envelope" and "maximum envelope" applied to Astor Theater block. Note the areas marked in which heights are bigger than the allowed by the regulations only by an amount lower than the lower floor height (residential floor).

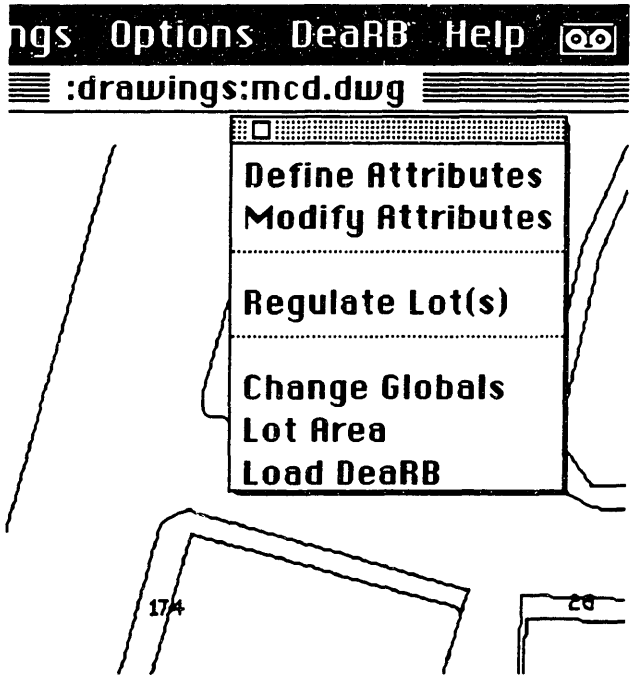


It is also possible to derive regulations by first designing manually a specific lot and afterwards through a process of trial and error it is possible to derive the zoning parameters to regulate the lot. The way DeaRB handles this is by changing the values of the block of attributes which might not be the best way since it is not very intuitive. As I will show later in the thesis, the interaction with the drawing entities might be very easily implemented the way DeaRB works.

I should point out that B.R.A.'s Zoning code states that the minimum set back for the MCD has to be at least 10 feet and the as-of-right height 90 feet when these are not specified. This means that there might be cases in which set backs do not make much sense, specially in lots with fronts in back streets and with façades with considerable indentations. This is particularly important for the lot next to Astor Theater shown in the above pictures. This is a vagueness of the regulations. Note also that this lot has again two façades so this case is particularly conflictive.

34. How does DeaRB work?

Figure 12. Functions of DeaRB's pull-down and detachable menu under item "DeaRB" of AutoCAD menu bar



DeaRB is fully menu driven which makes it very easy to use. The first thing to do is drawing a lot, a simple AutoCAD polyline. The following functions of DeaRB I will describe are the ones shown in the above bit mapped image of AutoCAD's pull-down and detachable menus under "DeaRB" item of AutoCAD's menu bar (see Figure 12.). Once identified the lots to regulate, then DeaRB needs to be loaded if it has not been loaded automatically at startup (by clicking "Load DeaRB"). The next step is to associate a block of attributes to each lot (by

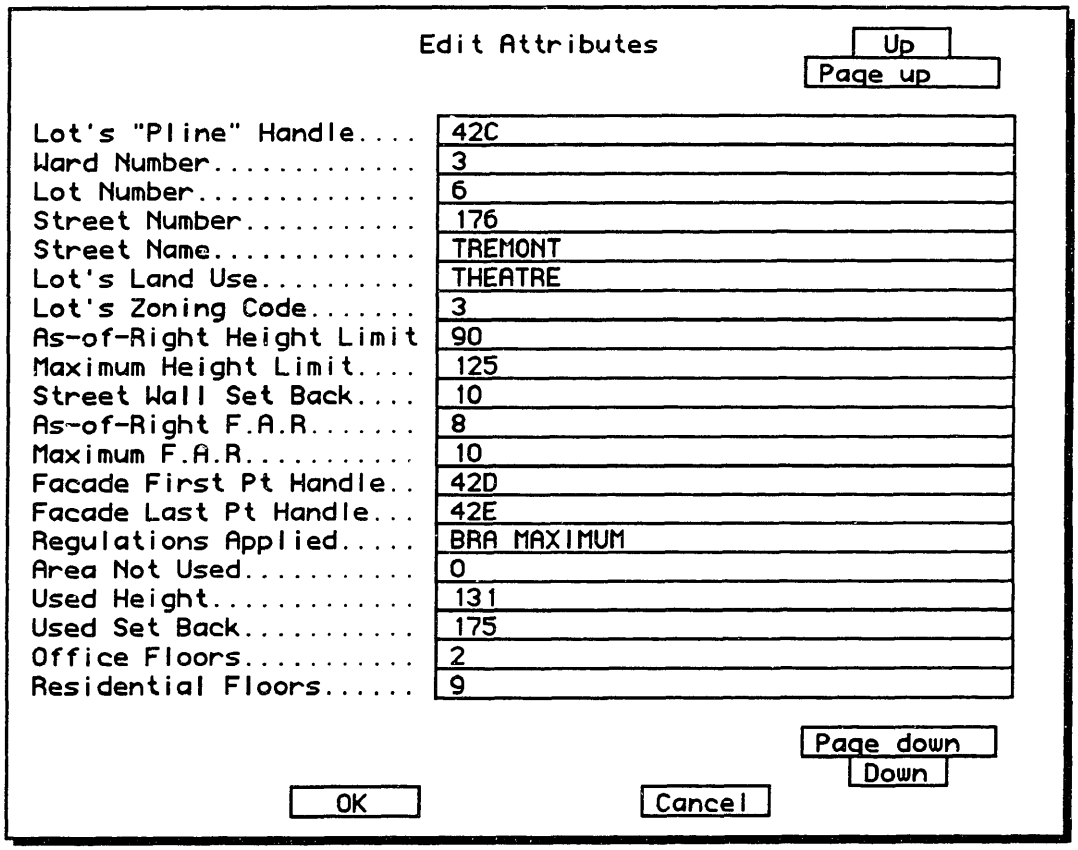
clicking "Define Attributes"), and finally the lot(s) can be regulated (by clicking "Regulate Lot(s)"). In the following explanation I will call "the user" any person who might work with DeaRB since it will need some external input to regulate and there are different ways to get back the new regulated parameters in function of the needs. The programming code of DeaRB's procedures I will describe next are included in Appendix 2.

As I said, before regulating a lot its list of attributes, or AutoCAD block of attributes needs to be defined. The procedure LOTATT binds drawing entities and blocks through AutoCAD handles, as I explained earlier, and it is invoked by clicking "Define Attributes" under "DeaRB" item of the menu bar (see Figure 12.). DeaRB only asks or prompts the user to input the parameters that define the lot, since it already knows the zoning parameters according to the zoning code (see procedure *zoning-parameters* in Appendix 2). Once the parameters are fixed, there is always the possibility to edit them using procedure MODATT and change them (the procedure is invoked by clicking "Modify Attributes" under "DeaRB" item of the menu bar illustrated in Figure 12.). This procedure is specially useful to get back the values proposed by DeaRB after regulating a lot, since there are fields specially designed to store the parameters DeaRB calculates (used height, potentially built area not used, number of office and residential floors, and the type of regulation set applied -i.e. B.R.A-) and these are only filled in after a regulation set is applied.

DeaRB's pull-down menu also includes two useful functions. "Change Globals" changes the global parameters (first floor and office and residential floors height, and maximum and minimum plates -although in the case of B.R.A. regulations maximum plates are fixed). "Lot Area" returns the area of the lot picked by the user. (see Figure 12.)

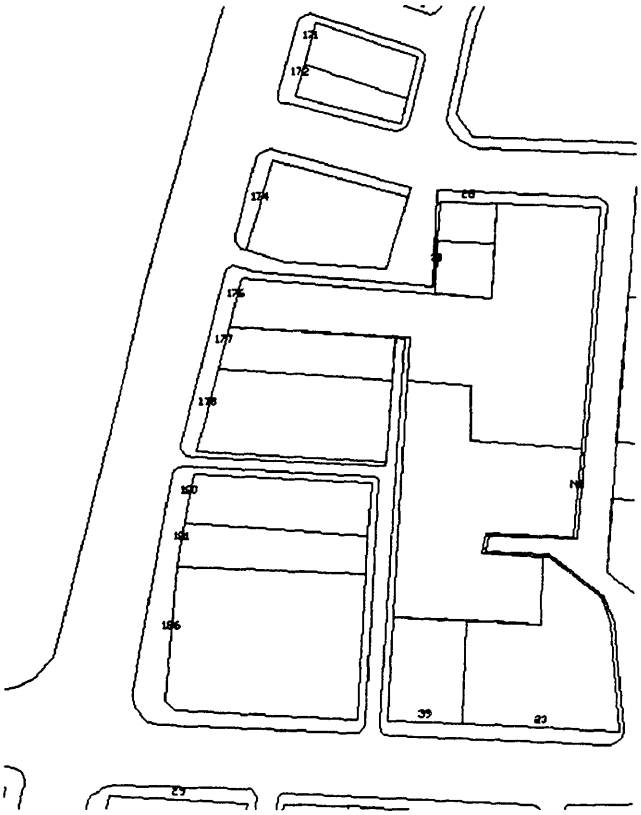
The screen bit map below (Figure 13.) is an example of the pop-up menu of DeaRB's AutoCAD block of attributes (LOTATTS) once these are associated to a lot by clicking "Define Attributes" as I described above. The pop-up menu is brought up when the user selects "Modify Attributes" item of the pull-down menu which internally calls procedure MODATT.

Figure 13.
AutoCAD's representation of the block of attributes for Astor Theater's lot



The next image (Figure 14.) is part of the MCD computer model done in AutoCAD. It corresponds to Astor Theater block that is being used as an example to illustrate how DeaRB works. Note that from the above list of attributes for Astor Theater block, only the street number of the lot is "visible" and DeaRB uses it when it needs the user to refer to a specific lot.

Figure 14.
Astor Theater block of the MCD computer model



DeaRB has a general procedure, REGULATE, that when called (by clicking "Regulate Lot(s)" in the pull-down menu) prompts to the user how does he/she want to regulate the lot(s) -the metaphor package to use- and wearing what hat. I will use the B.R.A.'s set of regulations (MCD Downtown Zoning Plan) and the "unscrupulous developer" hat to show how DeaRB works and how it manages the assumptions.

As I said before, DeaRB always considers the ground floor with special uses (commercial, cultural, leisure, etc) and only its height impacts the regulated design. Thus, DeaRB only knows about office and residential floors use (although one could fake the program and make it regulate other uses without DeaRB realizing it, since the knowledge part of the program only knows about relationships between offices and residencies).

The basic assumption of the unscrupulous developer hat is that it tries to build as much offices as possible (in relation to residencies), but without loosing any potentially built area (F.A.R.). This is just an assumption that might turn to be false in reality, but my purpose when designing the hat was only to state some assumptions to be able to propose a design. This need of assumptions before designing anything turns out to be an important issue in CAD systems since some CAD tools might be meaningless if there is no purpose, as I will discuss later.

The following is the flow of DeaRB when trying to regulate a lot under the unscrupulous developer hat using what I called B.R.A.'s metaphor package. I will only expose how the higher level knowledge is implemented and not the basic procedural calls and algorithms implementation that can be seen in the programming code included in Appendix 2. Basically I will explain how procedures *rough-envelope*, *as-of-right-regs*, and *maximum-regs* work (they are all called by procedure *regulate-as-BRA-ud* for BRA's metaphor and unscrupulous developer hat). Before this, I only want to say that any future metaphors and hats can be implemented autonomously of the rest of the metaphors and only REGULATE needs to know about their existence since it directs procedural calls in function of the user selections. Also, low level procedures can be shared by metaphors since they are more or less independent of them. This independence of low level procedures is function of the relationship between their level of abstraction and the metaphor itself. For instance, *check-plate* makes sure that the new plate is geometrically correct and embeds low level knowledge, but *calculate-new-setback* only works if there is a set back and if the set back has to be re-calculated optimizing the plate area (preserving the same plate area for all floors higher than the street wall or as-of-right height limit) what might or might not be the case depending on the metaphor.

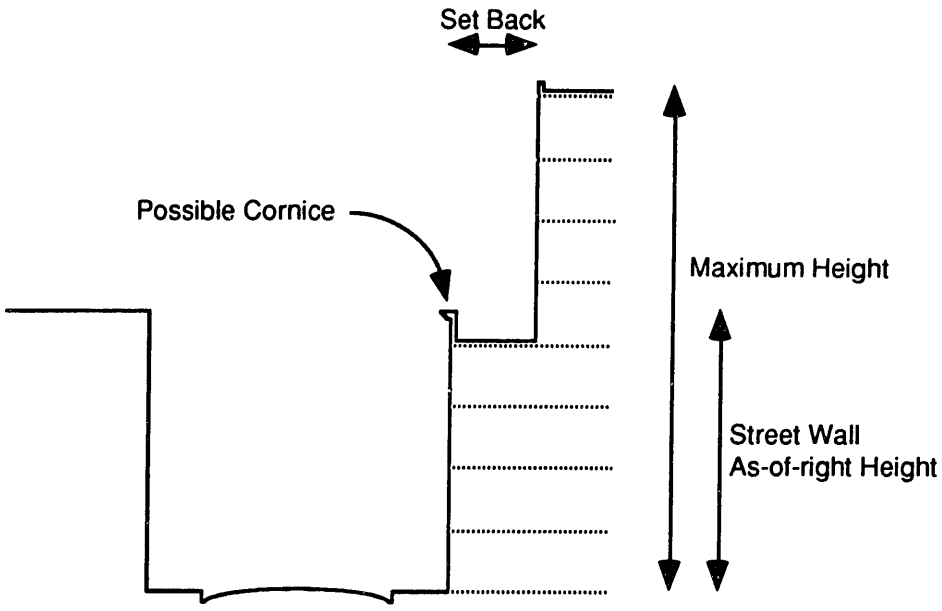
DeaRB's "rough envelope" is characterized by the use of only three zoning parameters: as-of-right height limit, maximum height limit and set back (see Figure 10. left). The program considers that these parameters are the more decisive to define the envelope that the building will have after all parameters are applied. If the maximum F.A.R. is more restrictive than the maximum height, then the final building envelope will be under the rough envelope since

the set back is always defined as the minimum set back from street wall. If the maximum height is more restrictive than the maximum F.A.R. then the building envelope will be the rough envelope. Thus, rough envelope calculation does not need either F.A.R. interaction to derive the envelope, or any assumption related to a specific hat, that is what use is preferential for the building.

What DeaRB understands as "as-of-right envelope" is the calculation of an envelope considering only as-of-right zoning parameters (as-of-right height and as-of-right F.A.R.) and this is handled by procedure *as-of-right-regs*. The way the MCD Zoning is written, maximum height is always associated to set back since in general a street wall of 90 feet height has to be respected when it is not specified in the specific zoning code of the lot. To calculate the as-of-right building envelope, DeaRB needs assumptions about preferences in use for building floors. Since the unscrupulous developer tries to fill as many offices as possible, the program tries first to build only offices. If there is any F.A.R. not used, it then calculates the minimum number of residential floors needed to be mixed with office floors to build all the F.A.R. The way DeaRB handles this is by calculating first how much height is left after supposing all lot is built with residencies and checking if this height left is multiple of the difference in height between offices and residencies. This very simple algorithm will also be used to calculate the "maximum envelope" exposed below.

Note that there is no figure representing the as-of-right envelope since it is very simple, the lot is extruded up to the as-of-right height limit even if there is not enough F.A.R. to build up to it. DeaRB represents it like this because it thinks that the concept of street wall is mandatory and supposes that a kind of fake cornice might be built so as to preserve the 90 feet height of the street wall as stated in the zoning code. This is done for simplicity of the drawing, although the right number of office and residential floors are calculated and stored in the block attribute list. Figure 15. below shows the right representation. Note, though, that the figure represents all the zoning parameters in which the as-of-right representation will be the same but just up to the street wall height.

Figure 15.
DeaRB's internal representation of the street cross section using all zoning parameters



The algorithm to calculate the "maximum envelope" is the more complex DeaRB's algorithm. Its complication comes from the fact that several different cases might occur in a lot. Procedure *maximum-regs* calculates all the parameters to get the lot built using all zoning parameters (as-of-right height limit, maximum height limit, set back from street wall, as-of-right F.A.R., and maximum F.A.R.) constrained by floor heights (first floor and office and residential floors) and maximum and minimum plates. The strategy DeaRB follows will be exposed next and it might be useful to see DeaRB's programming code in Appendix 2 since it is well annotated, especially before any condition or case.

DeaRB first checks how many zoning parameters has the lot and it knows that if there are only as-of-right parameters it has to build the as-of-right envelope exactly as as-of-right procedure does. If there is also a maximum F.A.R. but no set back or maximum height limit then it does a similar calculation than the as-of-right one but using the maximum F.A.R. instead of the as-of-right F.A.R. maximizing always the number of offices. If there is a set back or maximum height limit then DeaRB checks different relationships between F.A.R.s and

heights that are referred as "general case" in procedure *maximum-regs* of DeaRB's code in Appendix 2. The basic strategy is to calculate a new F.A.R. for the set back plate by deducting from the total potentially built area the area already used for the as-of-right height and dividing the rest by the new area of the plate calculated from setting back the façade. This way DeaRB can use the same strategy as in the anterior cases but this time applied to the set back plate as if it was a different building after using it for the bottom part of the building (that is up to the as-of-right height limit) as in the other cases. The underlying idea is that when the area is the same over all the floors the F.A.R. equals the number of floors. Besides this, depending on the different cases which vary in function of what is more restrictive heights or F.A.R.s, DeaRB tries first to fill in all offices, if it cannot fill the lot with offices then it tries residencies on top of offices that is residences from the as-of-right height up to the maximum height, after this it tries the opposite case (residencies up to the as-of-right height limit and offices on top of these), and finally it tries to build all residences so as to use the maximum potentially built area possible. Thus, the first priority of an unscrupulous developer hat is to build as much as possible and then the maximum number of office floors without loosing potentially built area.

In relation to the lower level procedures I would like only to point out that for every interaction DeaRB has with AutoCAD commands, I designed a procedure that will handle this interaction. I did this to keep DeaRB as modular and independent as possible and for efficiency. As I explained earlier, the advantage of object oriented programming techniques and abstraction barriers allow to change this basic commands without having to change all the program code. To illustrate this for example, procedure *getparam* allows DeaRB to get any of the parameters stored in the block of attributes, LOTATTS, and the procedures that need to call LOTATTS to get any parameters do not need to know how to actually get the parameters. This has two advantages, first there is not duplication of code and thus less possibilities of errors, and second and more important, if LOTATTS is modified, or even the whole conception of interaction between DeaRB and AutoCAD is modified, then only this procedure or the ones that interact with AutoCAD will have to be modified while DeaRB will stay intact.

35. Successes, failures, and knowledge implementation

To analyze success or failure of a design experiment like DeaRB one has to think about the goals and purpose of such an experiment. As I said, the main purpose of DeaRB was to study design itself and how a set of rules can influence design, what kind of design knowledge might be implemented as set of procedural rules, and ultimately designing better regulations. Under this point of view, DeaRB is just an example of what can be done with very simple tools like AutoCAD in a real design problem. Since I only implemented MCD Downtown Zoning regulations and just one had (unscrupulous developer), it is difficult to realize what different sets of rules can produce with the same design problems. But I devoted my main effort to design the support structure of a program that can be very easily enlarged, and it is in this sense that I think DeaRB is useful. On the one hand, it demonstrates how to handle a zoning code that architects and planners have to deal with in their common practice; it very easily and simply shows the relationships between height, F.A.R. and set back which are not obvious or at least easy to visualize dynamically. On the other hand, it shows how to do it with very simple tools. In relation to these tools, I think DeaRB proposes a very nice way to convert AutoCAD into a low level Geographical Information System with the advantages of a CAD system.

One of my big deceptions after having used high end computers, is the fact that computer tools are still very primitive and there is still no good CAD platform from which tailoring one's design needs. In the future architects and planners will be able to "design" their own CAD or GIS platform to suit their specific needs. It is in this sense that the most challenging problems I encountered were not only related to the implementation of the regulations themselves, but also solving and designing algorithms related to euclidian or geometrical problems. In this sense, DeaRB has several procedures that might be very useful for the common practice of design like calculating centers of gravity, triangulating a polygon, automatically drawing 3D faces in a polygon, offsetting a polygon or part of a polygon a specific distance, etc. DeaRB also demonstrates the amount of work needed today to handle such "simple" design problems, although as I said before, once the platform is "tailored" or designed, other similar design problems are very easy to implement and other modules or metaphors can be "plugged in", as explained in section 4 of this Part.

In relation to the regulations themselves and specially in relation to the B.R.A.'s zoning code for the MCD, I realized the incompleteness of some of the rules, after using the program to regulate the lots. In the next sections I will expose some of the considerations and steps towards a more complete set of rules based more in heuristics rather than in a "legal" expression of design concepts and I will expose some other important zoning considerations that DeaRB does not handle. In this respect it is interesting to note the character of the algorithms and knowledge representation issues. DeaRB does not know design concepts or qualities like what is a good public space, or how tall is too tall, or what is an obstructive view, etc. DeaRB knows only how to manage specific parameters and how to maximize ones over the others which is in my opinion a low level of knowledge, or at least a very functional knowledge instead of qualitative knowledge. In the conclusion I will compare architectural and engineering design problems using another experiment in which I participated at M.I.T. to expose more clearly how the formal aspects of design are different from the functional aspects.

One of the main lacks of the regulations is that there is no considerations about the existing context which is taken care of by negotiations in a case by case basis. For example the concept of street wall defined by a fixed number does not seem appropriate and it should be defined contextually in function of the neighboring buildings and maybe the mean height of the block and ultimately the one of the district. In the heuristic section I will expose some criteria about how to implement this.

Thinking about set backs, it might seem inappropriate to set a minimum number that will not depend on the specific height used by a building, because neighboring buildings might have different heights, but since they will have the same set back the impact of the shadows over the street will be very different as well as the views from the street. If we now consider the idea of creating a set back street wall, that is an alignment of the set back building plates, the fixed set back might be very appropriate, although other considerations will have to be taken to limit heights. If slender but higher towers are allowed over a certain height, then the impact of these on the street wall is very small because these will be isolated towers and the different set backs of these might turn to be unnoticeable. The point is that it is difficult to judge regulations the way they

are written because the purpose or the "spirit" of the regulations does not appear clearly.

Since my purpose in designing DeaRB was not to provide a complete zoning tool I think I have met the majority of my objectives, although I had to simplify a lot of important scenarios as I will point out in the next section. When dealing with computers and specially doing research in design methodologies, it is important to know how to simplify a design problem without trivializing it, not only to avoid not useful calculations, but also to better state the design problem isolating it. In this sense, the first two experiments exposed in Part 1 are very illuminating, but DeaRB although it is not as clear and pristine, it allows to set similar design problems but in a real scenario. I think the five zoning parameters I studied with constraints on floor heights and plate areas are able to regulate accurately the big majority of the MCD lots according to the MCD Downtown Zoning Plan, if we leave apart considerations of shadow impacts over the Boston Common and Public Gardens.

4. The next step with DeaRB

In the anterior section I exposed some of the limitations of DeaRB in relation to the implementation of the MCD Zoning Plan. Next I will expose in more detail what might be interesting to include in a future version of the program and how this might be done. It will be also interesting to extend it so as to cover other sets of regulations like N.Y. City regulations. In the next section I will expose the benefits of combining the actual zoning code with a heuristic approach.

It would be interesting to implement in DeaRB some use criteria as complementary parameters. In fact MCD regulations are very explicit about what kind of uses are allowed and how the introduction of certain uses might allow for extra zoning bonuses. For instance, section 38-11 of the Downtown Zoning for the MCD says: "within PDA III, not more than 50% of the floor area above 155 feet may be devoted to Office uses". This might be very easy to implement and it will only effect a small part of the program, especially what I have referred to as the "general case", in relation to the programming code.

Another lack that DeaRB has is that it does not allow for successive set backs as I pointed out earlier in the thesis. This will give DeaRB more flexibility since it will be then possible to compare different building scenarios equally valid under the same set of rules or metaphor. Up to a certain extent it is possible to do this now but through manually changing the parameters in the attributes block. In relation to the implementation of successive set backs, DeaRB will only have to include at least two more fields in the block of attributes (another height limit and another set back) and when doing the calculations, if the new set back is not defined, it will have to call more than once procedure *calculate-new-setback*, as I explained in the previous section, to derive the set back from the new plate area.

In a more lower level knowledge implementation, it might be interesting to generalize procedure *check-plate* so as to handle more general cases in which the façade of a lot is very irregular and might give "negative" areas if the *new-setback* is bigger than the "indentations" of the street wall, as I pointed out before. Although DeaRB can handle lots in which corners have vertices very close together when calculating the set back plates, as I described in section 33 of this part, DeaRB cannot handle lots with very small indentations along the street wall plane (this by the way, does not happen in the MCD and since it complicates specially the algorithm *calculate-new-setback* it was not implemented). For completeness, this could be implemented at the same time that designing a rule to handle the concept of what I call street wall "continuity" defined in the MCD zoning code. As section 38-19(1) of the MCD Downtown Zoning says, "the Street Wall of any Proposed Project shall be built to be coextensive with at least 80% of the Existing Building Alignment of the block on which the Proposed Proposed Project fronts, or to a depth from the street line equal to that of at least 80% of the Existing Building Alignment of either block adjacent to the block on which the Proposed Project is located, if there is no Existing Building Alignment of such block".

As pointed out in the previous section and to complete some of the ideas exposed at the beginning of this section in relation to several set backs, a new version of DeaRB could allow to build several new plates from the as-of-right height instead of one big plate so as to have several but slender "towers" from the as-of-right height. This might result in very different built scenarios for lots which areas are bigger than at least two times the minimum plates. In this sense, if

the minimum plates are defined as 5,000 square feet, this new variable will not affect significantly the MCD since lots are relatively small. The way DeaRB might be able to handle this is by just considering the different resulting plates as different buildings which areas will have to add up to the maximum F.A.R. of the lot.

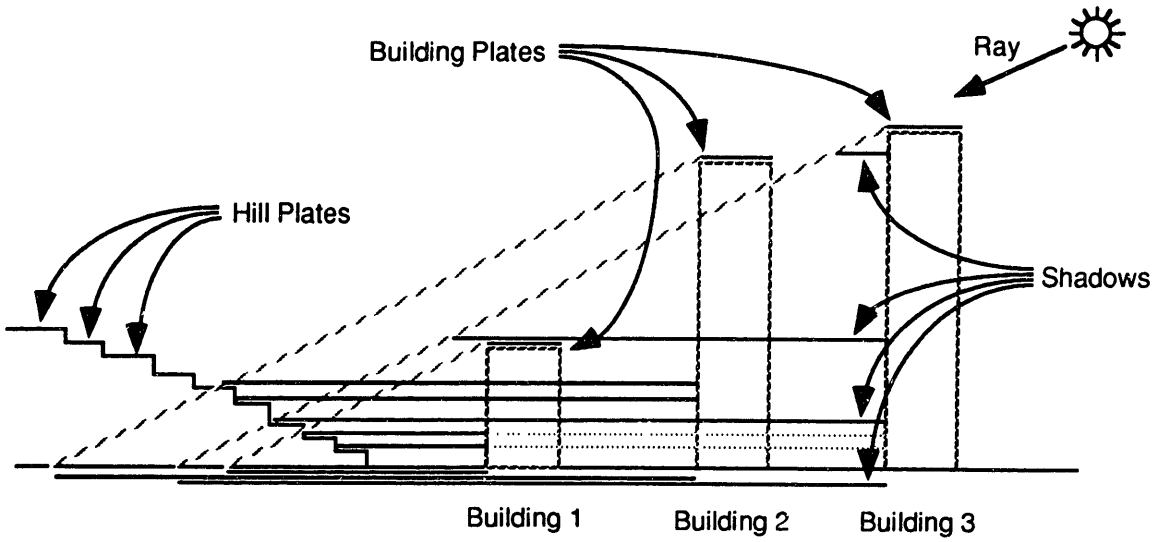
Finally I think there should be implement a module to consider maximum shadows over the Boston Common and Public Gardens. As the MCD Downtown Zoning Plan says in section 38-16, "Each Proposed Project shall be arranged and designed in a way to assure that it does not cast shadows for more than two hours from 8:00 a.m. through 2:30 p.m., on any day from March 21 through October 21, in any calendar year, on any single Shadow Impact Area". This last question will be more complex to implement since dealing with shadows usually needs complex algorithms and heuristics that might increase significantly computation time. In the next section about heuristics I will propose to build a separate module from DeaRB that will check shadow impacts and that will only be called by DeaRB or the user either for all or just for specific lots. There are two things that a shadow module will have to check: the first is the shadow itself and the second is the so called "ghost" effect. The "ghost" effect is that a building might benefit from another existent building by hiding its shadow on that of the existing building. In this sense, a shadow module might be interesting not only to check shadow impacts, but even to "transfer shadow rights" and also to allow derive a rough shape of the building from a given shadow. Very few of the existing CAD packages include satisfactory shadow modules, but none of them allow to study specifically the shadow of one single building and the ghost effect in relation to the rest of the shadows. It might be very complex to implement a complete module for shadow impact assessment, but if we reduce the problem to a plan view then it might turn to be simple enough to be handled even by AutoCAD, since the problem is reduced to calculate the distance of the shadow given a constant ray of light and the height of a building, or different heights and plates if there are set backs as shown in Figure 16. below. The module will have to find out the lots in the direction of the ray that might intersect with the shadow, but not the intersection itself. Then it will have to calculate and draw a shadow from the studied building to each of the lots that might intersect with this shadow at a height a little bit higher than the height of the last plate of these possible impacted buildings (so as not to intersect with them if the shadow

goes over them) and let AutoCAD render the image. Since AutoCAD renders in a way that higher faces always hide lower ones relatively to the view, the shadows that will go over plates will always hide these and the shadows that will not reach plates will not hide these plates. This method has the advantage that does not need to calculate intersections and it can be used to derive the ghost effect since the shadow of the studied building can be drawn in a different color and at the lowest height relatively to the rest of the shadows, these will hide parts of it getting what is called the ghost effect. The real shadow impact of the building will be then displayed on the specific color just showing the part of the shadow that is not ghosted. As Figure 16. below shows, shadows of buildings closer to the park might be drawn at a higher height than buildings farther from it so as to derive the ghost effect of these in the park.

Another important consideration in shadow studies is topography. In the case of the Common and Public Gardens topography is relevant enough so as to yield very different shadows (these will be shorter since the park has a hill). To use the same model I described above, hills might be considered as if they were buildings. Then, shadows of buildings would be drawn from these to the hill that can be treated as a building with successive set backs (successive shadows would be drawn at each different plate height) and the hills will have their own shadows that would ghost the shadows of the buildings studied.

Figure 16. below illustrates the algorithm to implement the shadow module above explained . The method is shown in cross section so as to appreciate plates and shadows which in reality will be surfaces rendered in a plan view. The profiles of the buildings are shown in dotted lines since there is no need to draw them in plan view (buildings will be represented by their highest plate. Note how shadow of building 3 do not affect building 2, but goes over building 1. Note also how the hill is represented as another building covering part of the building shadows at ground level, and how in turn, the building shadows hide those "hill plates" that would be shadowed by buildings.

Figure 16.
Cross section illustrating the implementation of the shadow module



5. A heuristic approach to regulations

This section will expose the kind of knowledge that can be implemented for what I have called "heuristics" metaphor. Again I do not pretend to give a complete set of rules, but just an alternative or complementary view of the zoning parameters implemented in what I have called B.R.A. metaphor, that is part of MCD Downtown Zoning regulations. The different concepts I will expose next and the rules I will derive are based on a contextual approach to regulations and the application of some heuristics about a traditional conception of good city form in a classical urban design language, as opposed to the "legal" language of zoning rules that I exposed in the first section of this part when talking about metaphor. I will show the necessity of using design concepts as opposed to legal specifications about designs to implement a set of regulations that will better reflect the purpose or "spirit" of the regulations. I will show how this heuristic set of rules can be more general than a set of rules written under the traditional "legal" paradigm and at the same time how it will be able to solve more specific cases that the legal set of rules does not give satisfactory solutions.

For a contextual approach the block of attributes, the information database of the lots, will have to be extended with a field or several fields giving

appreciations of existing conditions like age of the building or future possible projects about the lot. Then, some knowledge might be built into DeaRB so as it will know the implications of age and future projects on future changes of the lot. That way DeaRB will know when regulating a lot the likelihood of its context change and it will be able to decide how to regulate it according to its context, as I will explain next.

On the other hand, if use considerations are included in DeaRB, as mentioned in the previous section, then the program might be able to derive some statistics at the block or the district level about use needs and propose several built scenarios in function of use. Since the lot attributes block already contains a field for use, only the knowledge side about use will have to be built into DeaRB. Other considerations about residents needs or social and economic aspects, etc might be also implemented, but I focused on use since I think it constitutes one of the big lacks of DeaRB.

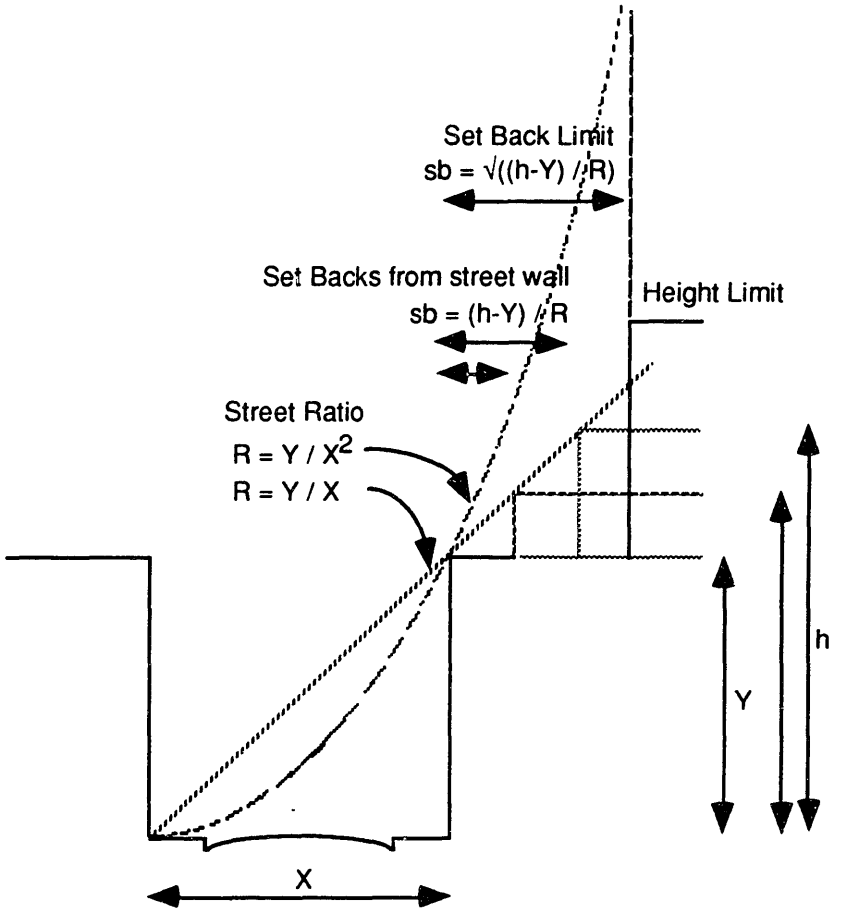
The first heuristics rules I would like to introduce are based on the concept of street ratio to define what is a good street in terms of proportion, view[10] and light (see Figure 17. below). This concept was first introduced by the International Style and become a standard from which to design streets or corridors considering light and health issues. In the MCD since the street wall is defined at 90 feet and the mean width of the district streets is around 60 feet, the street ratio proposed by the regulations will be around 1.5. The way the regulations are written it is difficult to say if the street wall height refers to the concept of street ratio or to a pure formal consideration, like keeping a kind of ceiling at the old height of building cornices. This might depend on the site and, for instance, the fact that the MCD regulations specify an as-of-right height of 90 feet if there is no special considerations about the street wall height in the specific zoning code of the lot, might be very well interpreted as a street ratio consideration. It might also seem realistic to consider a set back limit after which the street ratio will no more be applicable so as to give more possibilities of built out scenarios specially in the case of slender but higher towers, as I will explain next. There is a very simple way to implement the street ratio in DeaRB since the block of attributes of a lot already knows its street name. this

---

10 K. Lynch, G. Hack, Site Planning

procedure will consist of building a database that will contain street names with their associated width and a procedure that given a street name will return its width.

Figure 17.
Street Ratio illustration



As I pointed out in the anterior section, there should be the possibility of defining successive set backs according to either street ratios or fixed parameters for each lot or in function of the zoning code. Also, these set backs do not have to respond necessarily to the same street ratio, and a parameter can be introduced in the set back formula so as to account for this (sb = (h-Y) / R x parameter, R = Y / X), or the formula might change to an exponential curve street ratio (i.e., sb = $\sqrt{((h-Y) / R)}$, R = Y / X^2) in which the limit of the curve will be the set back limit. The exponential formula can be very easily implemented as the above Figure 17. shows and has the advantage that there will not be any need of setting limit

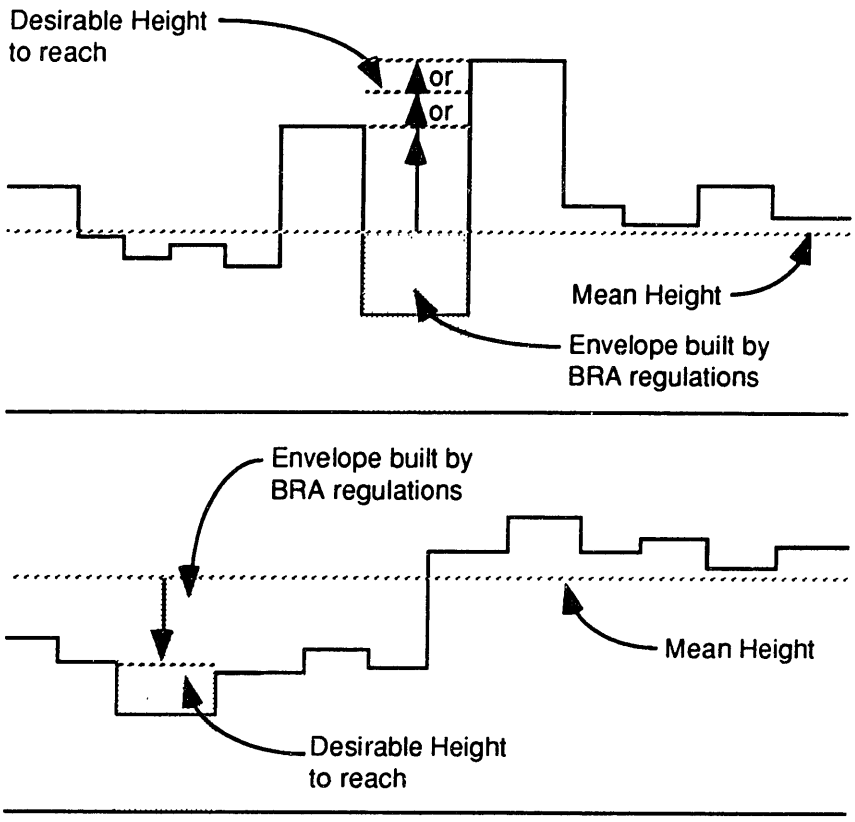
parameters since they will be implicit in the formula itself. If I might generalize, it seems to me that the explicitness of knowledge in any design program is important for the purposes of completeness. This explicitness results in capturing knowledge in the programming code itself rather than by filling in the information database of more or less meaningless parameters that is the lot attributes block in DeaRB.

There are very few cases in which the lot is big enough to be able to consider the possibility of building from the as-of-right height several slender but higher towers. In this sense, the street ratio concept explained above might be helpful to regulate how the towers should be built in terms of height, set back and distance between them. This heuristics might be very useful in a possible extension of DeaRB to contemplate New York regulations and have been also developed in the anterior section.

As a complement to the case of having higher but slender towers, DeaRB will have to know where to locate the tower(s) in relation to the lot. A good assumption is to start with a location at the center of gravity of the lot which is the ideal point to locate the vertical communications core since it minimizes distances from any point of the floor to it. In the case of having more than one tower, DeaRB will have to consider tower dimensions and street ratios to locate them in the lot. Note that the way DeaRB is conceived, the user will have always the possibility to change any DeaRB's decisions more or less manually, since DeaRB might do different design regulatory "moves" in function of the metaphor and hat used to regulate.

Besides what DeaRB already knows about floor heights, use of the floors and B.R.A. regulations, contextual information about neighboring lots will be fundamental to consider other buildings when regulating a lot, as I pointed out at the beginning. The basic criticism of regulations will be that since the actual regulations do not consider what is already built, in cases where buildings are higher or lower than the heights specified by the regulations, the concept of street wall does not stand, either buildings are to low and neighboring walls are seen from the public space or buildings are to high in relation to the existing ones (see Figure 18. below).

Figure 18.
Cases in which The MCD Downtown Zoning Plan fails to give adequate solutions (street elevation)



One can think that the way the heights limit are derived is by using the mean height of the block, street, or district. This seems to me to rough to solve the kind of cases I described above and in practice these are usually solved by negotiations with the B.R.A.. What I propose is to build some heuristic rules by which DeaRB will be able to decide up to what height to build considering context buildings and zoning code. In this way, DeaRB will inform after building the lot about how much area it has built on the lot so as to check wether it has built in excess or in default, as it already does. This will provide the information the program might use to build the area left by some lots in others that by their context will need more F.A.R. than the one specified by the regulations so as they can be in concert with their context. This means that DeaRB will be able to do transfer of air rights automatically and in an integrated way. Finally, it will be also possible to transfer air rights even outside the MCD and a transfer index

should be set up to account for different area quality, maybe in proportion to land price and level of urban infrastructures, etc.

6. DeaRB within EstheR's revised model

The purpose of this experiment, DeaRB, is to research design methodologies, the questions of how regulations as sets of procedural rules influence design, and ultimately how to design rules that might better capture design knowledge. The reality of a program like DeaRB implies also several other applications. On the one hand, it gives a nice and fast idea of what different zoning parameters might allow to build in a specific lot. This can be the starting point of a design, but also it can be used to verify if a building complies with the regulations. The B.R.A. was very interested in a tool like DeaRB since it might be very useful to help communities to visualize very fast and in the fly the specific decisions in terms of zoning allowances.

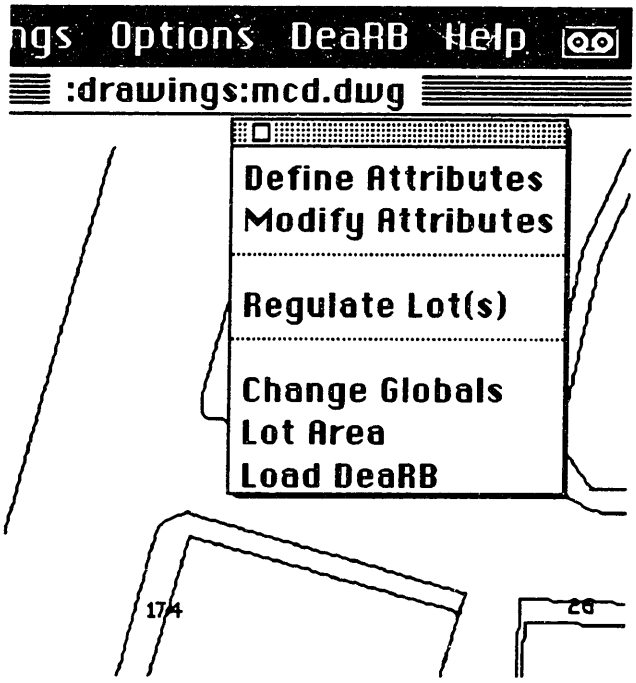
If we think now about EstheR, the experiment presented above, and the revisions I suggested, it will be very interesting to provide a conflict resolution strategy similar to the one of EstheR but designed to handle the kind of design problems of DeaRB so as to implement different metaphor packages and different hats, or different kind of characters, that will interact to propose a final design. This will have the advantage of testing what the different paradigms or sets of metaphors will propose for the same lot giving some insights about the paradigms themselves and about compatibility of "visions" or "worlds". Also, it might be very interesting to test a metaphor package under different hats to realize what are the premises of each character in the real "game".

Finally, the design of the arbiter in EstheR's revised model might be really useful to model real negotiations or conflict of interests. In this way, different scenarios can be designed and also different arbiters, as if arbiters would have their own hats. Then, the different design instances proposed by the system under different assumptions might be compared to better assess a design problem.

To solve the kind of problems that EstheR solves a CAD platform like AutoCAD is not appropriate since it implies a lot of computation, and AutoLisp (the programming language used by AutoCAD) is not powerful enough to handle the kind of conflict resolution strategy that EstheR uses. It might be possible though, to keep apart the drawing side from the knowledge base and have two software packages running at the same time. That is, some procedures will call the program with the knowledge base handled outside AutoCAD (maybe in Common Lisp) and in turn Common Lisp will give back the processed information so that AutoCAD will be able to store it and draw it. In this sense the database structure (block of attributes) I designed in DeaRB to interact with AutoCAD would be very useful. A lot might have as many blocks of attributes as metaphors are implemented in the program and hats will have their specific fields inside each metaphor block of attributes. If the knowledge base will need a lot of interaction with the information database, then part of this might be written to a file that both Common Lisp and AutoCAD might share.

The integration of EstheR and DeaRB is not a trivial problem and maybe a platform like Arris, that I mentioned at the beginning of this Part, might be the best alternative. It might turn to be to complicated to include a more or less complete set of metaphors and this is why I think a specific purpose should be found to integrate EstheR and DeaRB. I mean that maybe very few metaphors might be implemented so as to replicate just an expert on regulations with very few hats, depending on the kind of design problems to be solved by the program. In this sense, the integration of EstheR and DeaRB will probably be more similar to the classical expert systems that usually have very clear and defined purposes. It is in this sense that the design problem of regulations is a very good problem to start with when researching in the design field since by definition it is normative. In the next Part I will expose several considerations and generalizations about these issues and the specific questions I think are fundamental to address in future research studies.

Part 3. Thoughts about design methods



1. Designing Rules and Expert Systems

DeaRB shows a way of using new technology in design which is in some aspects different from a more "traditional" way of designing. It also illustrates in practice the idea of metaphors, that was developed in Disposable Metaphors with Gestaltatron and used by EstheR. The different worlds from which designs are "seen-as" is a useful way to propose and reconcile different solutions for the same design problem in which the interaction between metaphors, as well as the metaphors themselves, will tell about the "kind of designer". Specific "design moves" in specific design contexts would be characteristic of certain designers. Also, different metaphors will lead to the same design moves or rules and ultimately to the same designs, a point which is interesting to study to reveal some fundamental relationships between the process and the product.

As I pointed out in the last section of the previous Part, an advantage of DeaRB is that it might be used for different purposes, although I designed it basically to research in designers world and to illustrate how new technology will change the way designers work. I think an interesting step DeaRB can take is to implement with more completeness the regulations of the MCD under few and very specific metaphor packages, that is with specific purpose. For instance, at the B.R.A., a tool like DeaRB might become very useful to test different scenarios that will help to design rules or regulations, to make policies. It is in this sense that DeaRB might become an Expert System to design regulations where the expert could be the regulators of the B.R.A. itself. In a way, regulating is designing and a more ambitious use of DeaRB could be to change what I called the legalistic algorithms of the zoning code by design oriented algorithms, or sets of rules that will capture design knowledge. Maybe in the future programs like DeaRB might be able to be used in court substituting the actual legal regulations or zoning code. And if I am allowed to do a little bit of science fiction, when enough metaphors and hats would be designed along with different arbiters, DeaRB will be able to model negotiations and dispute resolutions. A computer program will be able to provide different views of the same design problem and find commonalties isolating what are the real problems that in reality tend to be disguised by second priority issues.

When designing Expert Systems, it is very important to find the right representation of the knowledge that the program is intended to capture, and I think this is especially important in design. Thus, a lot of attention has to be devoted in the implementation of a more complete system than DeaRB or a real regulator system since there might be some rules not very knowledge rooted but very decisive for the final outcome. What I mean is that although I think a regulator implementation should be thought more in terms of design than in terms of "legality", as the thesis defends, one might expect to implement some rules or regulations deeply based in legal terms and low knowledge based. It is then that representation issues become important since they might affect the internal program operations and the final design. As I have shown in the heuristic approach to regulations, it is important though to make an effort in finding general ways to capture regulatory knowledge and make it as explicit as possible in terms of rules set or metaphor packages. In other terms, I think the better is the regulator the more it recovers "amnesiac" or dead metaphors as I exposed in the first section of Part 2.

Because of time constraints, DeaRB turned to be a very limited design experiment. I think though DeaRB shows in a very practical way the power of using computers to design in what I think is the right way, that is deepening into the designer's world. Given the complexity of design, a lot of simplifications and assumptions need to be done to be able to design the design tool. It is important then to know how to simplify without trivialize, and I think the simplification comes with the understanding of the design question by rediscovering and capturing the fundamental knowledge that might be disperse or translated to different paradigms, like for regulations the legal paradigm. As I said, it is just a matter of convenience the fact that I used regulations to write DeaRB and talk about design; I do not mean that only designs consequence of sets of regulations, a kind of constraint approach to design[11], are the only possible computable designs, but very much the opposite. One of my interests when starting thinking about design methodologies using computers was to research on regulated "design moves" as opposed to non-regulated design moves. As it is implicit in the title of the thesis, Designing Rules is about the design of the rules themselves as well as the designs that the rules produce.

---

11 Gross, Ervin, Anderson, Fleisher, Constraints: Knowledge representation in design

Unfortunately I could not deepen on this issue. I would like to say, though, that if we think about design replication, as illustrated by EstheR, in my opinion it will be almost impossible to build a program that will produce what I might call "exceptional" designs, as I referred to in EstheR. The main reason is that it will be almost impossible to implement all the metaphors to replicate or produce instances of exceptional designs since I claim that any exceptional design has to have very few but fundamental contradictions about its own generation principles. This will be very difficult to capture in an automated design process from a replication point of view. Notice that I specified a replication point of view of the design process, since I think computers constitute a whole world or autonomous discipline which does not have yet been fully exploited in its own terms offering the possibility of creating exceptional designs. When a new tool is discovered, it takes generations to use it the "right" way, the way "it wants to be used", to discover its own and authentic means of expression and it is often used in a replication way, to replace old and less efficient procedures. It is in this sense that I would like to have contributed with this thesis, in showing a creative way to use computers under the point of view of a designer, how to design rules, and how the rules will design in turn.

I would like to emphasize the importance of new approaches to design that will use the current technology, specially in design practice. Today design is no more the end product of a single person's work, but rather the expression of sometimes dramatic processes where a lot of people interact with each other and with the design artifact towards producing a better design or environment. Also since designers produce designs from their different worlds or metaphors, understanding design means understanding the different worlds from which designs are created and the connections and incompatibilities between these worlds.

2. Kinds and levels of knowledge

As Allen Newell says[12], "If a system has (and can use) a data structure which can be said to represent something (an object, a procedure, ... whatever), then the system itself can also be said to have knowledge, namely the knowledge embodied in that representation about that thing". When designing DeaRB, It was particularly difficult to understand what would be the right abstractions barriers from which to build the program. To put it in terms of Newell, since CAD tools do not provide appropriate design tools, it was particularly difficult to work in different "knowledge levels" and not mixing the levels "competencies", and in a way DeaRB was also about finding the right abstractions from which starting building a designing system.

There is a side of DeaRB that consists, as I said, in finding more complete algorithms to capture design or just regulatory knowledge, like for instance the algorithm I propose for defining set backs (see section 5 of Part 2). If it is true that the design of the cross section of a street to define the street space is a typical designing method, and if it might be rooted in a specific way of looking at the world or metaphor, this might be very possibly a dead metaphor. In this sense, maybe this knowledge is a kind of lower level knowledge on top of which higher levels of knowledge should be built. In this sense, I think the effort should be devoted to design an algorithm that captures the generality of the rule, that puts together what might seem different but complementary conditions or situations.

In this sense, the engineering approach to design, or how engineers model the world so as to design is particularly interesting since the language they use is closer to the computer language that is, the abstractions computer languages are built from. At this point I would like to introduce a project I participated at M.I.T. and the kind of knowledge I implemented. We had to design a CAD system to design toys (model of animals) with rotating tails. The CAD system had to provide the tools to design the toy's geometry, rendering and dynamic simulation. In the team I participated we produced GRaDS (Geometry, Rendering and Dynamic Simulation) in which I implemented the dynamic

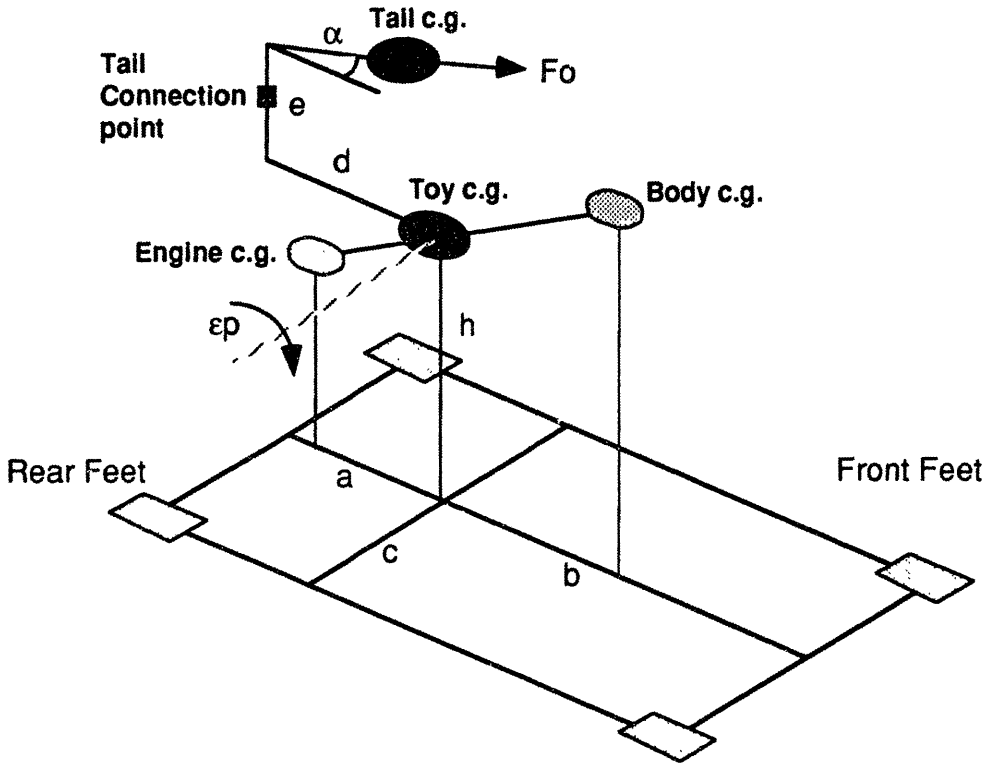
---

12 Allen Newell, "The Knowledge Level"

module. The interesting side of the dynamics was that toys had rotating tails producing interesting toy behavior in function of their tail rotation speed. To make the model more realistic I decided to include a movable point mass to simulate the engine that, in reality, will make the tail rotate, and that will completely change the dynamics of the toy depending on its location. Figure 19. below shows the schematics of the model representation and the algorithm I used to derive the dynamics where the angular distance was derived by double integration of the tail angular acceleration in function of time. The program was implemented in C language running on the Personal Iris by Silicon Graphics that allowed GRaDS to do the double integration on the fly for each time differential. I would like to point out how different is this kind of design knowledge based in physical laws from a more heuristic design based in architectural knowledge.

It is interesting to realize how GRaDS dynamic module was able to capture by a formula and with impressive accuracy the behavior of the toy apparently quite complicated. For a system like EstheR, what GRaDS was able to do was nothing more than designing under a specific metaphor. For the purpose of this thesis, the relevance of GRaDS was that it somehow brought what I will call, in this thesis context, a low level knowledge in the sense of abstraction barriers. I think there are regulations with rules that, if well conceived, might be able to be expressed as more complete algorithms instead of a set of conditional statements or parameters. Then a higher level knowledge will build from the lower level and will be based in heuristics according to specific metaphor packages. It is in this sense that I think constraints in design, as defined at M.I.T., constitute a basic or lower level knowledge, a kind of dead metaphor, from which other levels of knowledge should be implemented, but by no means they constitute a complete design paradigm.

Figure 19.
GRaDS, Dynamic Module internal representation and algorithm used to simulate the toy dynamics



Rotation about rear feet :

Integration of angular acceleration using 4th order Runge-Kutta integration

m_b = body mass; m_e = engine mass; m_t = tail mass

I_p = I_{zx} (pitch); I_t = I_{xy} (tail)

α = angle of tail with respect to the body = f(t)

ω = angular velocity of tail

$Fo = m_t * \omega^2 * r$ = tail centrifuge force

ε_p = toy rotation about the Y-axis, pitch

ε_p'' = angular acceleration of toy about the Y-axis

$$\varepsilon_p'' = \frac{m_b*g*(a_b*\cos(\varepsilon_p)+h_b*\sin(\varepsilon_p)) + m_e*g*(a_e*\cos(\varepsilon_p)+h_e*\sin(\varepsilon_p))) + (h+e)*Fo*\cos(\alpha)}{I_p + m_b*(a_b^2+h_b^2) + m_e*(a_e^2+h_e^2) + I_t + m_t*((r*\cos(\alpha)+(a-d)^2+(h+e)^2)}$$

3. Representation and Conception strategies

When using computers to design, it is important to know how to simplify the external representation of the design. In this case representation refers to how the designed artifact is presented to the designer, not how it is internally represented in the program, as I explained in the previous Part. For instance, the way DeaRB internally represents the lot, as a list of parameters or block of attributes with the number of office and residential floors, etc., reflected in Figure 15. of Part 2., is different from what was displayed in the screen to the user. As I explained, the lot was extruded up to the as-of-right height without considering the possible different heights of the cornice and the last floor although the program considers this difference of height to build the set back plate.

Another nice example is the method I proposed to implement the shadow module. To render an image representing shadow impacts might not be a question of really deriving all the real shadows by complex algorithms that will calculate real intersections, etc, if the model is limited to a plan view. At this point what I am defending is the fact that more or less elegant "low level knowledge" algorithms have to be found to solve specific problems that the practice of design requires and that CAD systems are not able to solve with efficiency. What I have been defending throughout the thesis is that designing rules or rules about design will be more complete if they are thought from the design point of view, subject to their metaphor. But there might be specific sets of rules where other design paradigms -like an engineering approach to design- might be more useful or convenient to implement them.

Specially in architecture, if we might now change to a different scale, issues about representation become very important under this formal point of view. Representation and conception ideas might sometimes differ towards modeling more efficiently the artifact, depending on the CAD tool. For example, to express the texture of the outside "envelope" of a building, its skin, it might not be necessary to "build" an accurate model to the conception of the building, that is its structure, basic volumes, installations, etc. If the purpose of modeling a building is to explain how the building is conceived or designed, I claim that the designer has to pay specific attention to the generating ideas of the building even

if these are designed rather than literal, although there might be cases where external representation issues might impose themselves to the expression of the internal representation (because of hardware and software limitations). I think the idea of designing, if we constrain the term to design replication aspects, should not be taken literally in a kind of "detectivistic" approach to the model. As I said before, to use computers in general design terms is probably the best way to use them, by their own means. Any model should in some ways be "drawn" with a purpose that has to be made explicit at some point in the model.

The differences between representation and conception strategies depend on convenience matters related to the model itself, as well as on the CAD platform. In my experience as teaching assistant at M.I.T., I realized how many problems that students faced when drawing a specific model of a building in a computer were not related to the understanding of the CAD platform, but rather to the understanding of the building itself. In this way, usually teachers make an effort to explain how computers work deepening -and sometimes confusing- students in high end technical skills and rendering techniques. I think a more difficult task is to teach how to use computers to express architectural meanings, to learn about architecture. In this sense, it is the use of very simple CAD tools what might reveal the design knowledge embedded in a building or design, not the ability to manage sophisticated techniques.

4. Future CAD tools

Today there is still no CAD environment that can be compared to a more or less complete design tool. What CAD platforms provide today is drafting means with very low level knowledge implementation. I always argue about that a designer should have descriptive geometry knowledge to allow his/her internal representation of the designs, the conception and visualization of the forms, and computers should not substitute it. If in the future computers might become another paradigm from which looking to the world, since, as I defended in the thesis, they have their own logic and language, I am not certain to what extent the design tools they might provide will trivialize the designers work, and at the end will contribute to establish an impenetrable frontier between designer and

design, or will become so fundamental that designers will not be able to create without a computer.

I have also approached the concept of abstraction barriers applied to CAD tools, that is, how low level knowledge procedures might build up the ground from which higher level functions would be implemented. In this sense, if future CAD tools are built on the concept of metaphor packages and abstractions, then it might be possible that in a more or less conscious manner, users or designers would be able to tailor their CAD environment, so as the computer will know how the designer wants to apply a specific high level function. Maybe the computer will ask either the first time the CAD program is used, or each time a different CAD tool is first used, the kind of qualifier the user wants the program to use, and it will remember it for next sessions. In this sense, every time a different designer uses the machine, this will change its drawing and designing tools according to user's metaphor and presenting a tailored environment where tools behave according to the designer's expectations.

Conclusion

As the thesis shows, implementing a program that designs with more or less completeness implies some thinking about the design process and about the way designers interact with designs and with other designers. Metaphors are useful constructs that allow designers to understand a design by reducing its complexity. I have shown how this is precisely the way to approach the design of rules that will capture design knowledge, since design as language is fundamentally metaphorical in character. I have shown the usefulness and the limits of the different models proposed, and I have proposed some future steps that these might take. I would like this thesis to be seen as an introspection into the designers' world and into the designs they create, specially calling the attention to the fact that new thinking should be devoted to new technology, specially computers, since it brings very valuable designing tools that have not yet been fully exploited in their own terms.

I have shown how designing rules implies different knowledge levels which implementation might be radically different and decisive towards obtaining a good design environment or CAD platform, as well as its internal and external representation. In this sense, I would like to emphasize how abstraction barriers in the implementation of computer programs for designing, have to be very related to the design abstractions themselves. In the case of programs to implement urban regulations I have shown the need to recover the fundamental spirit or purpose of their rules. This rediscovery does not need to be literal, but might be more design oriented or a matter of interpretation, as long as it is useful and/or "equivalent". I have argued also that in design matters the equivalence of paradigms or sets of metaphors might be more questionable, but in lower level knowledge implementation it might be an extremely powerful abstraction from which to build higher level procedures or the design metaphors or sets of rules. The thesis has introduced also the idea of "hats" or kind of characters that would qualify the use of metaphors in their own way. That is, different hats would act in differently on the same design and using the same metaphor packages.

The thesis proposes also different uses that a "regulator" program might take besides researching in design itself, like helping decision-makers or communities

to better understand the relationship of zoning parameters by visualization, or assessing experts in regulations by capturing their knowledge, or to model real negotiations or dispute resolutions where several hats will interact with each other under different worlds or metaphor packages, and where different arbiters would be designed to take decisions. This will help to realize the fundamental matters of convergency or discordance of the different worlds' perspectives about the same design question.

Finally the issues brought up by the thesis are fundamental to develop future CAD tools, or tools more design oriented as opposed to drafting oriented. The basis of metaphors will allow in the future the users to tailor their CAD environment so as to make the computer understand design as they do being able to use the design tools the way they will have specified what will be different for each user.

Appendix 1. EstheR's metaphors, rules and Immediacy formula

```
;;; structures

(defstruct person
            name
            persistence
            metaphors)

(defstruct metaphor
            name
            number-of-times-active
            move-candidate
            seer
            seeing-as-rules
            move-rules
            rule-used
            immediacy)

(defstruct rule
            name
            immediacy
            LHS
            RHS
            priority)

;;; instances

(setf replicant
        (make-person    :name 'pris
                        :persistence 0.5
                        :metaphors '((statics 0.5) (centering 0.5))))

(setf statics
      (make-metaphor    :name 'statics
                        :number-of-times-active 0
                        :move-candidate nil
                        :seer 'statics-seer
                        :seeing-as-rules '*statics-s-rules*
                        :move-rules '*statics-m-rules*
                        :rule-used '(X 0)
                        :immediacy 0))

;;;seeing-as-rule

       (make-rule
        :name "A) LINES DO NOT INTERSECT ->
                        CONSIDER STATICS METAPHOR"
        :immediacy 0.5
        :LHS
            '(notany #'(lambda (c)
            (eq (intersection-type c) 'LL))
            (statics-world-corners
            *statics-s-description*))
        :RHS
            nil
        :priority 0)
```

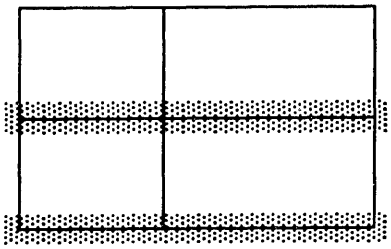
```
;;; move-rule

       (make-rule
        :name "F) BOTH BLOCKS CORNER-FIXED ->SWAP BLOCKS"
        :immediacy 0.1
        :LHS
            '(and
            (eq *block-one-status* 'corner-fixed)
            (eq *block-two-status* 'corner-fixed)
            (not (overused 'F 'statics)))
        :RHS
            '(progn
            (increment-use 'F 'statics)
            (swap-blocks *statics-s-description*))
        :priority 2)
```

The following is the formula we designed for Immediacy

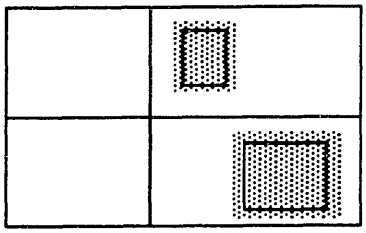
```
                (history (+ p
                             (* (- 1 p)
                                (/ 1 (* (+ n 1) (sqrt (+ n 1)))))))

                   (immediacy (+ weight
                                 (* (- 1 weight)
                                    (+ history
                                       (* (- 1 history)
                                          (/ i 2)))))))

              (if (> immediacy top-immediacy)
                (progn
                  (setf *top-move* (metaphor-move-candidate meta4))
                  (setf top-immediacy immediacy)
                  (setf (metaphor-number-of-times-active meta4) (+ n 1))
                  (print '-)
                  (princ "active metaphor: ") (princ meta-name)))))

         (person-metaphors subject)))
     (if (null *top-move*) (return nil)
         (progn
           (setf *picture-log* (append *picture-log* *picture*))
           (eval *top-move*)))
  )))
```

The following slides illustrate how the rules are implemented in the Statics and Centering metaphors for each sub-package of the program: s-description, seeing-as rules, move-rules and in the case of Statics, corner priority.

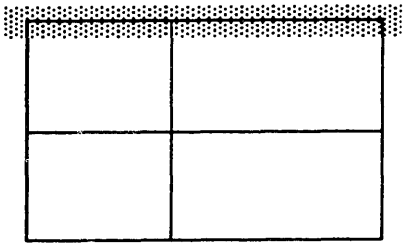
Statics-s-description



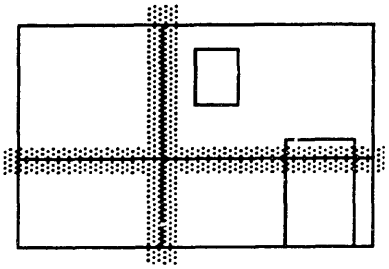
Tabletops



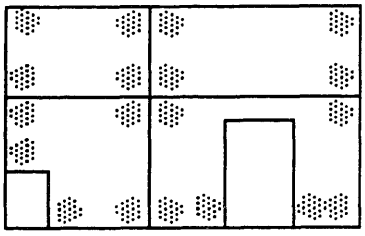
Blocks



Ceiling

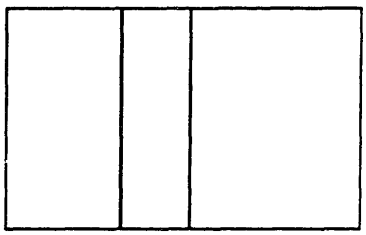


Lines



Corners

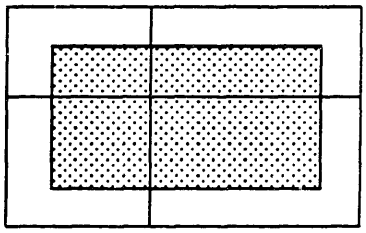
Statics Seeing-As Rules



A

(Priority 0, Immediacy 0.5)

NOT

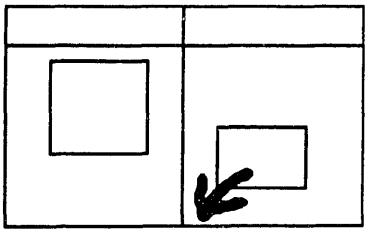


B

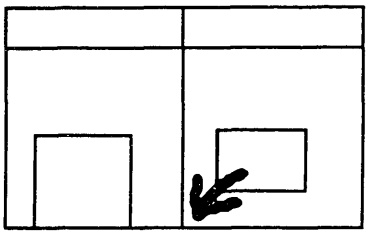
(Priority 0, Immediacy 0.2)

Either no center or center close to the edge

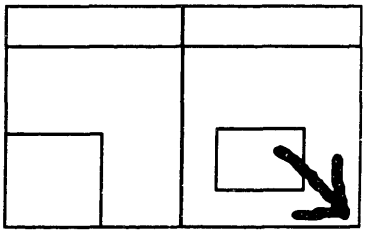
Statics Move Rules



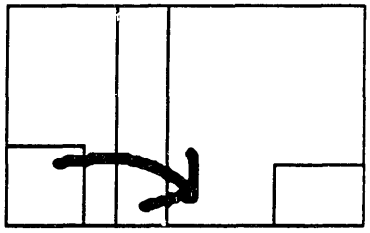
C
Move lower block to most-convenient-corner lower than the bottom of the block
(Priority 4, Immediacy 0.5)



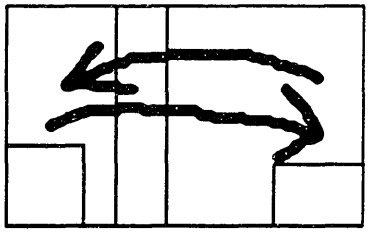
OR



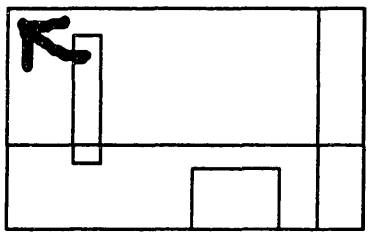
D
Move "free" block to most-convenient-corner, lower than it
(Priority 5, Immediacy 0.8)



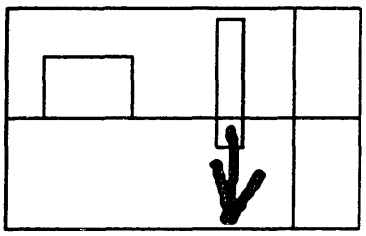
E
Move most-convenient-block to most-convenient-corner
(Priority 3, Immediacy 0.1)



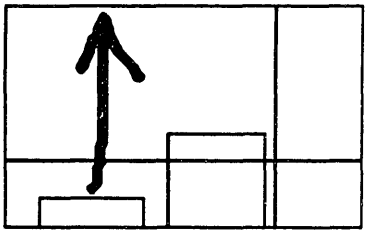
F
Swap positions
(Priority 2, Immediacy 0.1)



OR

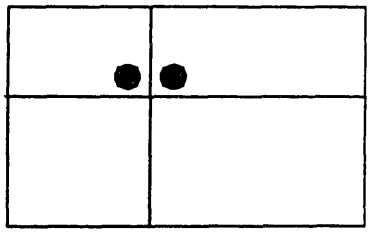


G
Move to the closer of the most-convenient non-intersecting-corner or the tabletop
(Priority 6, Immediacy 0.5)



H
Move the "flatter" block to the ceiling
(Priority 1, Immediacy 0.3)

Statics Corner Priority

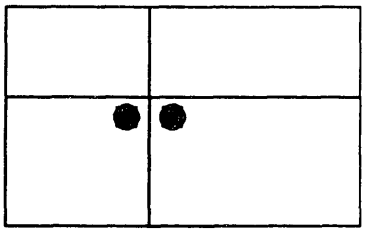


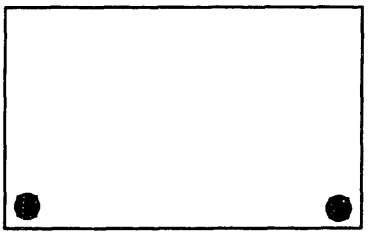
First

UL, LL
UR, LL

Second

DL, LL
DR, LL



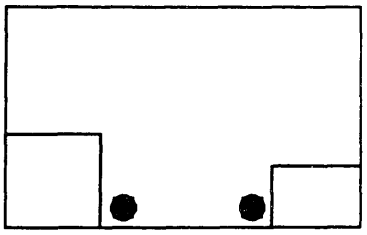


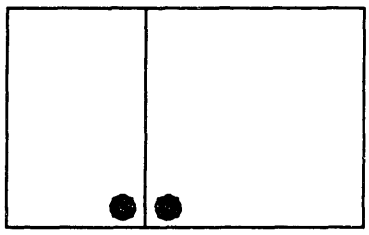
Third

UL, FF
UR, FF

Fourth

UL, FB
UR, FB



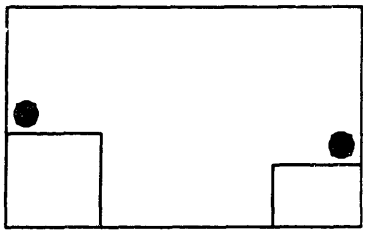


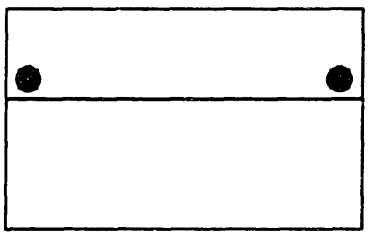
Fifth

UL, FL
UR, FL

Sixth

UL, BF
UR, BF



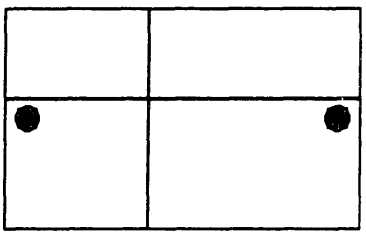


Seventh

UL, FL
UR, FL

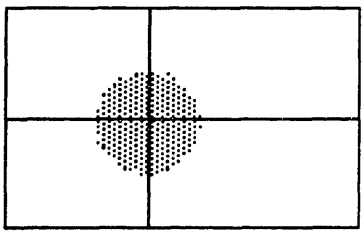
Eighth

DL, FL
DR, FL

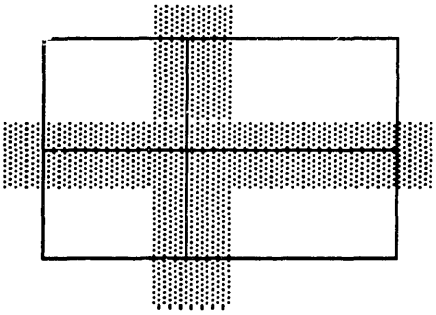


Types of UpDownness and Handedness of corners:
UL: Up, Left UR: Up, Right DL: Down, Left DR: Down, Right
Types of intersection:
FF: Frame-Frame FL: Frame-Line LL: Line-Line
FB: Frame-Block (block on tabletop) BF: Frame-Block (Block next to border)

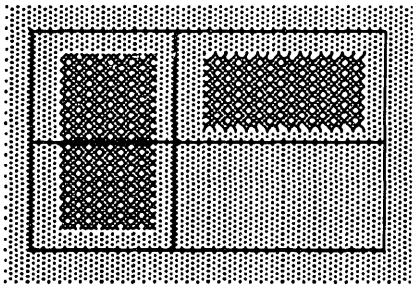
Centering-s-description



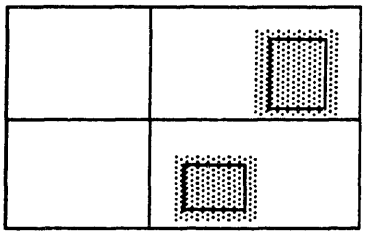
Origin



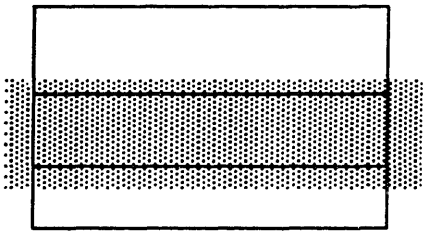
Axes



Regions

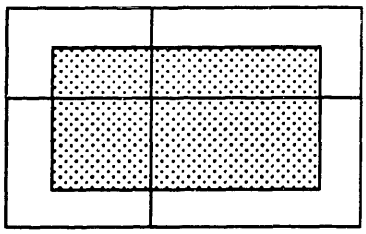


Rectangles

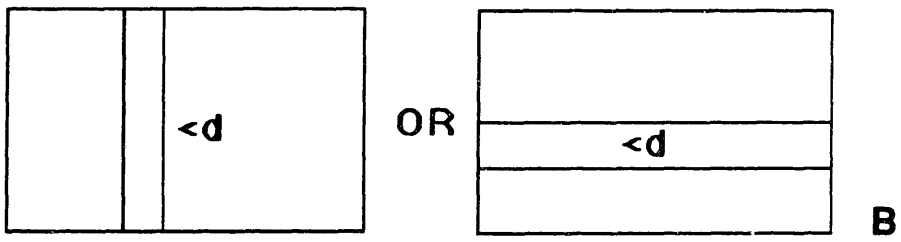


Bands

Centering Seeing-As Rules

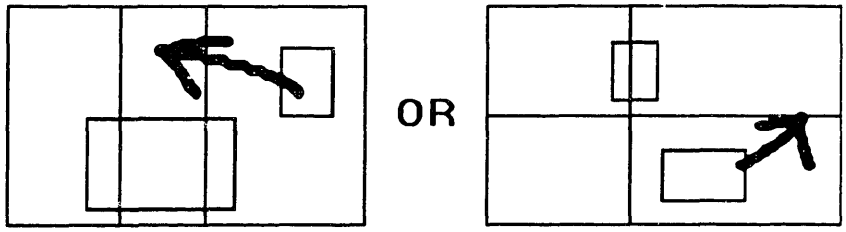


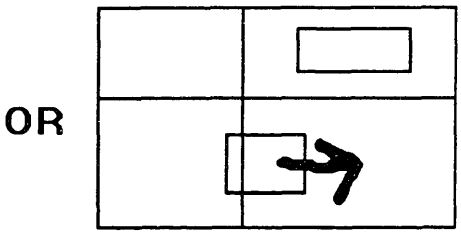
Origin of Axes close to the center of the picture
(Priority 0, Immediacy 0.8)



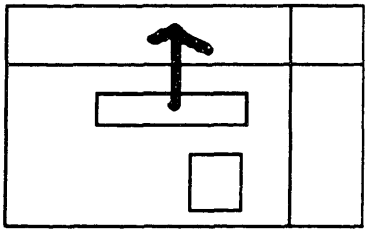
Parallel lines close together
(Priority 0, Immediacy 0.5)

Centering Move Rules

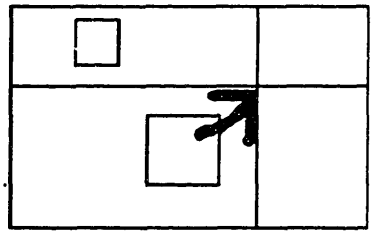




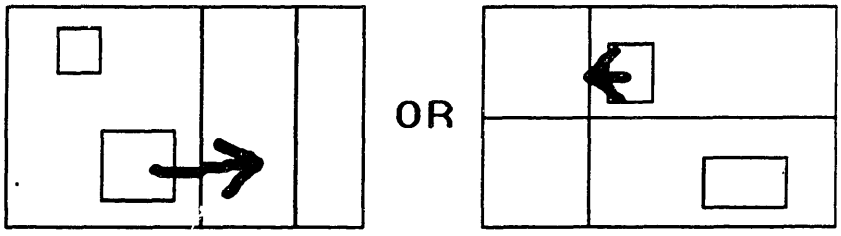
C
Move free rectangle to some band, or closest axis or region respectively
(Priority 1, Immediacy 0.8)



D
Move rectangle into the similar-proportion region
(Priority 2, Immediacy 0.6)



E
Move the closest rectangle to the origin to the origin
(Priority 4, Immediacy 0.2)



F
Center the rectangle closest to the band or axis to it
(Priority 3, Immediacy 0.4)

Appendix 2. DeaRB program

```
;;DeaRB.lsp


;;(vmon)

;;LDRB loads the procedures in this file
(defun C:LDRB() (load ":progs:DeaRB"))

;;puts handles on if starting a new drawing
(command "HANDLES" "ON")


;;global parameters

(setq h1 15                    ;;first floor height
      ho 13                    ;;office floor height
      hr 10                    ;;residential floor height
      min-floor-plate 5000     ;;minimum floor plate
      max-floor-plate 25000)   ;;maximum floor plate (according to BRA)


;;CHGGLOBAL changes the global parameters

(defun C:CHGGLOBAL (/ temp)
  (princ "\nEnter new First Floor Height, or <cr> to keep <")
  (princ h1)
  (setq temp (strcase (getstring ">: ")))
  (setq h1 (set-global temp h1 "First Floor Height"))

  (princ "\nEnter new Office Floor Height, or <cr> to keep <")
  (princ ho)
  (setq temp (strcase (getstring ">: ")))
  (setq ho (set-global temp ho "Office Floor Height"))

  (princ "\nEnter new Residential Floor Height, or <cr> to keep <")
  (princ hr)
  (setq temp (strcase (getstring ">: ")))
  (setq hr (set-global temp hr "Residential Floor Height"))

  (princ "\nEnter new Minimum Floor Plate, or <cr> to keep <")
  (princ min-floor-plate)
  (setq temp (strcase (getstring ">: ")))
  (setq min-floor-plate (set-global temp min-floor-plate
                                                  "Minimum Floor Plate"))

  (princ "\nEnter new Maximum Floor Plate, or <cr> to keep <")
  (princ max-floor-plate)
  (setq temp (strcase (getstring ">: ")))
  (setq max-floor-plate (set-global temp max-floor-plate
                                                  "Maximum Floor Plate"))

  (princ "\n..Global Parameters Changed...")
  (prin1)
 )
```

```
(defun set-global (temp global type / ans)
  (if (not (eq temp ""))
    (if (eq 0 (atof temp))
      (progn
        (princ "\n")
        (princ type)
        (setq ans (strcase (getstring
              " equals 0, do you want to change it? <y>es, or <cr> no: ")))
        (if (or (eq ans "Y") (eq ans "YES"))
          (progn
            (princ "\nRe-enter ")
            (princ type)
            (getreal " value: "))
          (atof temp))
        )
      (atof temp))
    global)
  )


;;LOTATT attaches the block of attributes "LOTATT" to a lot in layer
;;"LOTATTS".  All blocks of attributes "LOTATT" are in layer "LOTATTS" and
;;their attributes are: lhandle, ward, lnum, stnum, stname, luse, lzoning,
;;                      hl, hlmax, stepback, far, farmax

(defun C:LOTATT (/ oldblip oldecho oldattdia oldos oldlayer
                   lhandle ward lnum stnum stname luse lzoning
                   hl hlmax stepback far farmax hfpt0 hfpt1
                   vname ipt0 ipt1 stnum0 zplst baname fpt0 fpt1 loop)
  (setq oldblip (getvar "BLIPMODE")
        oldecho (getvar "CMDECHO")
        oldattdia (getvar "ATTDIA")
        oldos (getvar "OSMODE")
        oldlayer (getvar "CLAYER"))
  (setvar "CMDECHO" 0)        ;;speeds up lisp since no comments to screen
  (setvar "BLIPMODE" 0)

  (setq lhandle (cdr (assoc 5 (entget
              (car (entsel "\nPick a lot to define its attributes: "))))))
  (redraw (handent lhandle) 3)

  (setq ipt0 (cdr (assoc 10 (entget (entnext (handent lhandle)))))
        ipt1 (cdr (assoc 10 (entget (entnext (entnext
                                          (handent lhandle)))))))
  (setq ipt0 (list (/ (+ (car ipt0) (car ipt1)) 2)
                   (/ (+ (cadr ipt0) (cadr ipt1)) 2)
                   (/ (+ (caddr ipt0) (caddr ipt1)) 2)))
  (setq ward (getint "\nEnter lot's ward number: ")
        lnum (getint "\nEnter lot number: ")
        stnum (strcase (getstring "\nEnter lot's street number: "))
        stname (strcase (getstring T "\nEnter lot's street name: "))
        luse (strcase (getstring "\nEnter lot use: "))
        )
  (princ "\nEnter lot's zoning code")
  (setq lzoning (getint "\n<0>Historical, 1, 2, 3, 4, 5, <10>PDA I,
                        <11>PDA II, <12>PDA III, or <13>Hinge Block: "))

  (setq stnum0 (atoi stnum))  ;;since stnum might have letters at the end
```

```
(setq zplst (zoning-parameters lzoning stnum0 stname))
(setq hl (nth 0 zplst)
      hlmax (nth 1 zplst)
      stepback (nth 2 zplst)
      far (nth 3 zplst)
      farmax (nth 4 zplst)
      )

;;asks for the initial and final points of the lot polyline facade, that
;;have to be entered again clockwise
(setvar "OSMODE" 1)               ;;to get end point, vertex of polyline
(setq fpt0 (getpoint "\nEnter first facade point in clockwise sense: ")
      fpt1 (getpoint "\nEnter last facade point in clockwise sense: "))
(setvar "OSMODE" oldos)
(setq vname (entnext (handent lhandle))
      loop 1)
(while loop
       (if (equal fpt0 (cdr (assoc 10 (entget vname))))
         (setq hfpt0 (cdr (assoc 5 (entget vname)))
               loop '())
         (setq vname (entnext vname))))
(setq vname (entnext (handent lhandle))
      loop 1)
(while loop
       (if (equal fpt1 (cdr (assoc 10 (entget vname))))
         (setq hfpt1 (cdr (assoc 5 (entget vname)))
               loop '())
         (setq vname (entnext vname))))

;;arguments are reversed from attribute definition, and only "stnum"
;;is visible
(setvar "ATTDIA" 0)          ;;no dialogs for attributes
(command "LAYER" "m" "LOTATTS" "")
(command "INSERT" "LOTATT" ipt0 1 1 0 lhandle
         ward lnum stnum stname luse lzoning
         hl hlmax stepback far farmax hfpt0 hfpt1
         "" "" "" "" "" "")
(command "LAYER" "s" oldlayer "")
(setvar "ATTDIA" oldattdia)

;;changes block's attributes to layer "LOTATTS"
(setq baname (entlast))
(while baname
       (entmod (subst (cons 8 "LOTATTS") (assoc 8 (entget baname))
                      (entget baname)))
       (setq baname (entnext baname)))

(setvar "BLIPMODE" oldblip)
(setvar "CMDECHO" oldecho)
(princ "\n...Lot Attributes Assigned...")
(prin1)
)
```

```
;;zoning-parameters returns the list '(hl hlmax setback far farmax)

(defun zoning-parameters (lzoning stnum stname)
  (cond ((or (eq lzoning 0) (eq lzoning 13))
         ;;historical bld. or Hinge Block lot
         (list (getreal "\nEnter lot's As of Right Height Limit: ")
               (getreal "\nEnter lot's Maximum Height Limit: ")
               (getreal "\nEnter lot's Step Back: ")
               (getint "\nEnter lot's As of Right F.A.R.: ")
               (getint "\nEnter lot's Maximum F.A.R.: ")
               ))
        ((eq lzoning 1) (list 90 155 10 8 10)) ;;General Area
        ((eq lzoning 2) (list 65 0 0 4 0))   ;;Bay Village Protection Area
        ((eq lzoning 3)  ;;Boston Common and Public Garden Protection Area
         (cond ((and (eq stname "TREMONT")
                     (or (>= stnum 104) (<= stnum 147)))
                (list 90 125 10 8 10))       ;;between BLOMFIELD and WEST
               ((and (eq stname "TREMONT")
                     (or (>= stnum 148) (<= stnum 186)))
                (list 90 155 10 8 10))       ;;between WEST and BOYLSTON
               ((and (eq stname "BOYLSTON")
                     (or (>= stnum 62) (<= stnum 176)))
                (list 90 130 10 8 10)) ;;between TAMWORTH and PARK SQUARE
               ((and (eq stname "BOYLSTON")
                     (or (>= stnum 178) (<= stnum 242)))
                (list 90 155 75 8 10))       ;;between PARK SQUARE and
                                             ;;HADASSAH WAY
               ((and (eq stname "BOYLSTON")
                     (or (>= stnum 244) (<= stnum 300)))
                (list 85 130 50 8 10)) ;;between HADASSAH WAY and ARLINGTON
               (t (princ
                "\n...No Zoning Parameters for this lot qualified as 3...")
                  (list 0 0 0 0 0))
              ))
        ;;Ladder Blocks and Washington Street Theater Protection Area
        ((eq lzoning 4) (list 90 125 10 8 8))
        ;;Newspaper Row / Old South Protection Area
        ((eq lzoning 5) (list 90 125 10 8 8))
        ((or (eq lzoning 10) (eq lzoning "PDAI")) (list 90 350 35 10 14))
        ((or (eq lzoning 11) (eq lzoning "PDAII")) (list 90 400 35 10 14))
        ((or (eq lzoning 12) (eq lzoning "PDAIII")) (list 90 300 25 10 14))
        (T (princ "... ")
           (princ lzoning)
           (princ " is not a known zoning qualification...")
           (list 0 0 0 0 0))
        )
  )


;;MODATT modifies lot attributes

(defun C:MODATT (/ oldecho oldattdia oldlayer abhandle)
  (setq oldecho (getvar "CMDECHO")
        oldattdia (getvar "ATTDIA")
        oldlayer (getvar "CLAYER"))
  (setvar "CMDECHO" 0)  ;;speeds up lisp since no comments to screen
  (command "LAYER" "s" "LOTATTS"    ;;to be able to pick properly block
           "off" "ENVELOPE"         ;;attributes
```

```
            "off" "3DF-ENVELOPE"
            "off" "ASOFRIGHT"
            "off" "3DF-ASOFRIGHT"
            "off" "MAXIMUM"
            "off" "3DF-MAXIMUM"
            "")
   (setvar "ATTDIA" 1)          ;;to use attribute dialogue prompts
   (setq abhandle (cdr (assoc 5 (entget (car (entsel
         "\nSelect the lot street NUMBER to modify lot attributes: "))))))
   (redraw (handent abhandle) 3)
   (redraw (handent (getparam "LHANDLE" abhandle)) 3)
   (command "DDATTE" "L")
   (setvar "ATTDIA" oldattdia)
   (command "LAYER" "s" oldlayer
            "on" "ENVELOPE"
            "on" "ASOFRIGHT"
            "on" "MAXIMUM"
            "")
   (setvar "CMDECHO" oldecho)
   (prin1)
   )


;;getparam gets the specified attribute value given the attributes block
;;handle

(defun getparam (param abhandle / ename)
  (setq ename (entnext (handent abhandle)))
  (if (eq (cdr (assoc 0 (entget ename))) "SEQUEND")
    '()
    (if (eq (cdr (assoc 2 (entget ename))) param)
      (cdr (assoc 1 (entget ename)))
      (getparam param (cdr (assoc 5 (entget ename))))))
  )


;;putparam stores the specified attribute value given the attributes block
;;handle

(defun putparam (param value abhandle / ename)
  (setq ename (entnext (handent abhandle)))
  (if (eq (cdr (assoc 0 (entget ename))) "SEQUEND")
    '()
    (if (eq (cdr (assoc 2 (entget ename))) param)
      (entmod (subst (cons 1 value) (assoc 1 (entget ename))
              (entget ename)))
      (putparam param value (cdr (assoc 5 (entget ename))))))
  )


;;REGULATE applies different regulations sets to lots

(defun C:REGULATE (/ oldblip oldecho meta abhandle hat how what)
  (setq oldblip (getvar "BLIPMODE")
        oldecho (getvar "CMDECHO"))
  (setvar "CMDECHO" 0)          ;;speeds up lisp since no comments to screen
  (setvar "BLIPMODE" 0)         ;;put it after dialogs
```

```
;;ask the user what regulations to use
(princ "\nWhat Regulations Metaphor do you want to apply?")
(setq meta (strcase (getstring
                      "\n<B>RA, <H>euristics, ..., or <cr> to Quit: ")))
(princ "\nWhat Hat do you want to wear?")
(setq hat (strcase (getstring
    "\n<5>th Paradigm Planner, ..., or <cr> Unscrupulous Developer: ")))

(cond
 ((not (or (eq "B" meta) (eq "BRA" meta)
           (eq "H" meta) (eq "HEURISTICS" meta)))
  (princ "\n...")
  (princ meta)
  (princ " is not a valid metaphor, No Regulations applied...")
  )
 ((or (eq "5" hat) (eq "5TH" hat))
  (princ
   "\n...Not yet implemented since 5th Paradigm is not yet well
   defined...Just kidding...")
  )
 (T (princ "\nHow do you want to apply the Regulations?")
    (setq how (strcase (getstring
   "\n<A>s-of-right, <M>aximum allowed, or <cr> for a rough envelope: "))
          what (strcase (getstring
             "\nApplied to <A>ll lots, or <cr> to one or more lots?: ")))
    (cond ((or (eq "B" meta) (eq "BRA" meta))
           (cond
            ((or (eq "5" hat) (eq "5TH" hat))
             (setq hat "5th Paradigm hat...")
             (regulate-as-BRA-5 what how)
             )
            (T
             (setq hat "Unscrupulous Developer hat...")
             (regulate-as-BRA-ud what how)
             )
            )
           (princ "\n...BRA Regulations applied wearing ")
           (princ hat)
           )
          ((or (eq "H" meta) (eq "HEURISTICS" meta))
           (cond
            ((or (eq "5" hat) (eq "5TH" hat))
             (setq hat "5th Paradigm hat...")
             (regulate-heuristically-5 what how)
             )
            (T
             (setq hat "Unscrupulous Developer hat...")
             (regulate-heuristically-ud what how))
            )
           (princ "\n...Heuristics Regulations applied wearing ")
           (princ hat)
           )
          )))

(setvar "BLIPMODE" oldblip)
(setvar "CMDECHO" oldecho)
(prin1)
)
```

```
;;regulate-as-BRA-ud applies BRA's specified zoning regulations to all or
;;just one lot it applies attributes of block "LOTATT" to lot polygons
;;associated to them thinking as an unscrupulous developer which means
;;optimizing offices over residences

(defun regulate-as-BRA-ud (what how / oldlayer abhandle
                           ssablk abnum abnum0 abhlst abhlst0
                           vertnext ptlst flst rflst fpt0 fpt1 loop)
  (setq oldlayer (getvar "CLAYER"))
  ;;abhlst is the list of attributes' block handles
  (if (or (eq "A" what) (eq "ALL" what))
    (progn
      (setq ssablk (ssget "X" (list (cons 0 "INSERT")
                                    (cons 2 "LOTATT")
                                    (cons 8 "LOTATTS"))))
      (setq abnum (sslength ssablk))
      (princ "\n...There are ")
      (princ abnum)
      (princ " lot attributes in layer LOTATTS...")
      (princ
       "\n...Proceeding to apply regulations, this might take a while...")
      ;;gets abhlst, list of attributes block handles
      (setq abhlst '()
            abnum0 0)
      (while (< abnum0 abnum)
             (setq abhandle (cdr (assoc 5 (entget (ssname ssablk abnum0))))
                   abnum0 (1+ abnum0))
             (setq abhlst (append abhlst (list abhandle)))
             )
      )
    (progn
      (setq ssablk (ssget))
      (setq abnum (sslength ssablk))
      ;;gets abhlst, list of attributes block handles and checks conditions
      (setq abhlst '()
            abnum0 0)
      (while (< abnum0 abnum)
             (setq abhandle (cdr (assoc 5 (entget (ssname ssablk abnum0))))
                   abnum0 (1+ abnum0))
             (if
               (and
                (eq "INSERT" (cdr (assoc 0 (entget (handent abhandle)))))
                (eq "LOTATT" (cdr (assoc 2 (entget (handent abhandle)))))
                (eq "LOTATTS" (cdr (assoc 8 (entget (handent abhandle))))))
               (setq abhlst (append abhlst (list abhandle))))
             )
      (princ "\n...From selection there are ")
      (princ (length abhlst))
      (princ " lot attributes in layer LOTATTS...")
      (princ
       "\n...Proceeding to apply regulations, this might take a while...")
      ))

  (setq abhlst0 abhlst)
  (while abhlst0
         (setq abhandle (car abhlst0))

         ;;ptlst is the list of vertices of a lot polygon
```

```
          (setq vertnext (entget (entnext (handent
                                           (getparam "LHANDLE" abhandle))))
                ptlst '()
                fpt0 (cdr (assoc 10 (entget (handent
                                             (getparam "HFPT0" abhandle)))))
                fpt1 (cdr (assoc 10 (entget (handent
                                           (getparam "HFPT1" abhandle))))))
          (while (eq "VERTEX" (cdr (assoc 0 vertnext)))
                 (setq ptlst (append ptlst
                                     (list (cdr (assoc 10 vertnext)))))
                 (setq vertnext (entget (entnext
                                           (cdr (assoc -1 vertnext))))))

          ;;flst is the facade points list and rflst is the rest of the
          ;;facade points
          (setq loop 1
                rflst '()
                flst ptlst)
          (if (not (equal fpt0 fpt1))
            (while loop
                   (if (equal fpt0 (car flst))
                     (setq rflst (cdr (member fpt1 flst))
                           flst (reverse (member fpt1 (reverse flst)))
                           loop '())
                     (setq flst (append (cdr flst) (list (car flst)))))))

          ;;builds scenario considering parameters HL, HLMAX, SETBACK, FAR
          ;;and FARMAX
          (cond ((or (eq "A" how) (eq "AS-OF-RIGHT" how))
                 (as-of-right-regs abhandle ptlst flst rflst))
                ((or (eq "M" how) (eq "MAXIMUM" how))
                 (maximum-regs abhandle ptlst flst rflst))
                (T (rough-envelope abhandle ptlst flst rflst)))

          (setq abhlst0 (cdr abhlst0));;loops over the list of block handles
          )
  (command "LAYER" "s" oldlayer
           "off" "3DF-ENVELOPE"
           "off" "3DF-ASOFRIGHT"
           "off" "3DF-MAXIMUM"
           "")
  )


;;rough-envelope only considers HL, HLMAX and SETBACK to build a rough
;;envelope

(defun rough-envelope (abhandle ptlst flst rflst /
                       lhandle hl hlmax setback far farmax)
  (setq lhandle (getparam "LHANDLE" abhandle)
        hl (atoi (getparam "HL" abhandle))
        hlmax (atoi (getparam "HLMAX" abhandle))
        setback (atoi (getparam "SETBACK" abhandle))
        far (atoi (getparam "FAR" abhandle))
        farmax (atoi (getparam "FARMAX" abhandle)))

  (cond
   ((and (not (eq 0 hl)) (eq 0 hlmax) (eq 0 setback))
```

```
    (draw-regs ptlst flst rflst lhandle '"ENVELOPE" '"3DF-ENVELOPE" hl 0 0)
    )
   ((not (or (eq 0 hl) (eq 0 hlmax) (eq 0 setback)))
    (draw-regs ptlst flst rflst lhandle '"ENVELOPE" '"3DF-ENVELOPE"
               hl hlmax setback)
    )
   ((and (eq 0 hl) (eq 0 hlmax) (eq 0 setback) (eq 0 far) (eq 0 farmax))
    (princ "\n...No Zoning Parameters for this Lot..."))
   (T (princ
    "\n...Combination of Zoning Parameters Not Known by The System..."))
   )
  )


;;as-of-right-regs considers the zoning parameters HL and FAR, and builds
;;the as-of-right regulations.  It assumes that HL represents the concept
;;of "street wall" and thus it always builds up to this limit.  The user is
;;given a chance to modify this so as to only build the needed height
;;according to first floor, residential and office heights

(defun as-of-right-regs (abhandle ptlst flst rflst /
                         lhandle hl hlmax setback far farmax
                         larea nr1 nr no)
  (setq lhandle (getparam "LHANDLE" abhandle)
        hl (atoi (getparam "HL" abhandle))
        hlmax (atoi (getparam "HLMAX" abhandle))
        setback (atoi (getparam "SETBACK" abhandle))
        far (atoi (getparam "FAR" abhandle))
        farmax (atoi (getparam "FARMAX" abhandle)))

  (cond
   ((not (or (eq 0 hl) (eq 0 far)))
    (if (<= ho (/ (- hl hl) (- far 1)))
      (progn                    ;;builds all offices until hl
        (draw-regs ptlst flst rflst lhandle '"ASOFRIGHT" '"3DF-ASOFRIGHT"
                   hl 0 0)
        ;;stores parameters list
        ;;(regs-type not-used-area usedh new-setback no nr)
        (putparam-lst "REGS-TYPE"
                      (list '"BRA AS-OF-RIGHT" 0 hl 0 (- far 1) 0)
                      abhandle)
        )
      (progn      ;;builds until HL and far might be all used or not
        (setq larea (area-polygon ptlst))
        (setq nr1 (fix (/ (- hl hl) hr)))
        (if (> nr1 (- far 1)) ;;checks far
          (setq nr1 (- far 1)))
        (setq no (fix (/ (- hl hl (* nr1 hr)) (- ho hr))))
        (setq nr (- nr1 no))
        (draw-regs ptlst flst rflst lhandle '"ASOFRIGHT" '"3DF-ASOFRIGHT"
                   hl 0 0)
        ;;stores parameters list
        ;;(regs-type not-used-area usedh new-setback no nr)
        (putparam-lst "REGS-TYPE"
                      (list '"BRA AS-OF-RIGHT"
                            (* larea (- far no nr 1)) hl 0 no nr)
                      abhandle)
      )))
```

```
   ((and (eq 0 hl) (eq 0 hlmax) (eq 0 setback) (eq 0 far) (eq 0 farmax))
    (princ "\n...No Zoning Parameters for this Lot..."))
   (T (princ
    "\n...Combination of Zoning Parameters Not Known by The System..."))
   )
  )


;;maximum-regs considers all zoning parameters HL, HLMAX, SETBACK, FAR and
;;FARMAX

(defun maximum-regs (abhandle ptlst flst rflst /
                     lhandle hl hlmax setback far farmax
                     larea sblarea far0 nr no nr0 no0 nr1 no1
                     sbrfar sbofar sbnr sbno
                     new-sblarea new-setback sbfar-fix usedh)
  (setq lhandle (getparam "LHANDLE" abhandle)
        hl (atoi (getparam "HL" abhandle))
        hlmax (atoi (getparam "HLMAX" abhandle))
        setback (atoi (getparam "SETBACK" abhandle))
        far (atoi (getparam "FAR" abhandle))
        farmax (atoi (getparam "FARMAX" abhandle)))

  ;;calculates larea and sblarea considering maximum floor plate (minimum
  ;;floor plate is handled when calculating new-setback.  Note that setback
  ;;might "mutate" according to the maximum floor plate, but this mutation
  ;;do not alter the attribute list
  (setq larea (area-polygon ptlst)
        sblarea (area-polygon (check-plate
                                    (setback-facade setback flst rflst))))
  (if (> sblarea max-floor-plate)
    (setq sblarea max-floor-plate
          setback (calculate-new-setback setback flst rflst
                                                  max-floor-plate)))

  (cond
   ;;cases in which there is no setback and no hlmax, similar to
   ;;as-of-right case or cases in which lot is too small to apply setback
   ((or (and (not (eq 0 hl)) (eq 0 hlmax) (eq 0 setback) (not (eq 0 far)))
        (and (not (or (eq 0 hl) (eq 0 hlmax) (eq 0 setback)
                      (eq 0 far) (eq 0 farmax)))
             (< sblarea min-floor-plate)))
    (if (eq 0 farmax)
      (setq far0 far)          ;;case in which there is only hl and far
      (setq far0 farmax))      ;;case in which there is hl and farmax
    (if (<= ho (/ (- hl hl) (- far0 1)))
      (progn             ;;builds all offices until hl using far or farmax
        (draw-regs ptlst flst rflst lhandle '"MAXIMUM" '"3DF-MAXIMUM"
                   hl 0 0)
        ;;stores parameters list
        ;;(regs-type not-used-area usedh new-setback no nr)
        (putparam-lst "REGS-TYPE"
                      (list '"BRA MAXIMUM" 0 hl 0 (- far0 1) 0)
                      abhandle)
        )
      (progn   ;;builds until HL and far or farmax might be all used or not
        (setq nr1 (fix (/ (- hl hl) hr)))
        (if (> nr1 (- far0 1))      ;;checks far
```

```
      (setq nr1 (- far0 1)))
     (setq no (fix (/ (- hl hl (* nr1 hr)) (- ho hr))))
     (setq nr (- nr1 no))
     (draw-regs ptlst flst rflst lhandle '"MAXIMUM" '"3DF-MAXIMUM"
                hl 0 0)
     ;;stores parameters list
     ;;(regs-type not-used-area usedh new-setback no nr)
     (putparam-lst "REGS-TYPE"
                   (list '"BRA MAXIMUM"
                         (* larea (- far0 no nr 1)) hl 0 no nr)
                   abhandle)
    )))
;;general case with all parameters
((not (or (eq 0 hl) (eq 0 hlmax) (eq 0 setback)
          (eq 0 far) (eq 0 farmax)))
 (setq nr0 (fix (/ (- hl hl) hr))
       no0 (fix (/ (- hl hl) ho)))
 (setq sbrfar (/ (* larea (- farmax 1 nr0)) sblarea)
                                  ;;resid. area fitting in sblarea
       sbofar (/ (* larea (- farmax 1 no0)) sblarea)
                                  ;;office area fitting in sblarea
       sbnr (/ (- hlmax hl (* nr0 hr)) hr)
                                  ;;max. residences num. up to hlmax
       sbno (/ (- hlmax hl (* no0 ho)) ho))
                                  ;;max. offices num. up to hlmax
 (cond
  ((<= ho (/ (- hl hl) (- farmax 1)))
   ;;similar case as as-of-right, builds all offices until hl
   (draw-regs ptlst flst rflst lhandle '"MAXIMUM" '"3DF-MAXIMUM"
              hl 0 0)
   ;;stores parameters list
   ;;(regs-type not-used-area usedh new-setback no nr)
   (putparam-lst "REGS-TYPE"
                 (list '"BRA MAXIMUM" 0 hl 0 (- farmax 1) 0)
                 abhandle)
   )
  ((and (<= hr (/ (- hl hl) (- farmax 1)))
        (< (/ (- hl hl) (- farmax 1)) ho))
   ;;can build only residential until hl with all farmax, checks offices

   (cond
    ((<= sbofar sbno)   ;;farmax more restrictive than hlmax, builds all
     ;;offices using all farmax, tower might change to fit them
     (setq new-sblarea (/ (* sbofar sblarea) (1+ (fix sbofar))))
     (setq new-setback
               (calculate-new-setback setback flst rflst new-sblarea))
     (if (eq 0 new-setback)            ;;checks if new-setback is OK
       (setq new-setback setback))
     (setq sbfar-fix (fix sbofar))
     (setq no (+ no0 sbfar-fix))
     (setq usedh (+ hl (* no ho)))
     (draw-regs ptlst flst rflst lhandle '"MAXIMUM" '"3DF-MAXIMUM"
                hl usedh new-setback)
     (putparam-lst "REGS-TYPE"
                   (list '"BRA MAXIMUM" 0 usedh new-setback no 0)
                   abhandle)
     )
    ((<= sbofar sbnr)      ;;farmax more restrictive than hlmax, builds
```

```
    ;;residences on top of offices using all farmax, tower might change
    (setq new-sblarea (/ (* sbofar sblarea) (1+ (fix sbofar))))
    (setq new-setback
              (calculate-new-setback setback flst rflst new-sblarea))
    (if (eq 0 new-setback)            ;;checks if new-setback is OK
      (setq new-setback setback))
    (setq sbfar-fix (fix sbofar))
    (setq no no0    ;;no need to optimize, it will only get one office
          nr sbfar-fix)
    (setq usedh (+ h1 (* nr hr) (* no ho)))
    (draw-regs ptlst flst rflst lhandle '"MAXIMUM" '"3DF-MAXIMUM"
               h1 usedh new-setback)
    (putparam-lst "REGS-TYPE"
                  (list '"BRA MAXIMUM" 0 usedh new-setback no nr)
                  abhandle)
    )
   (T   ;;farmax more restrictive than hlmax, builds all residences
        ;;no need to optimize offices up to hlmax since very few might
        ;;be gained
    (setq no (fix (/ (- h1 h1 (* nr0 hr)) (- ho hr))))
    (setq nr (- nr0 no))
    (draw-regs ptlst flst rflst lhandle '"MAXIMUM" '"3DF-MAXIMUM"
               h1 0 0)
    (putparam-lst "REGS-TYPE"
                  (list '"BRA MAXIMUM" 0 h1 0 no nr)
                  abhandle)
   ))
 )
(T       ;;builds as much as offices as possible unless loosing far
 (cond
  ((<= sbofar sbno)   ;;farmax more restrictive than hlmax, builds all
   ;;offices using all farmax, tower might change to fit them
   (setq new-sblarea (/ (* sbofar sblarea) (1+ (fix sbofar))))
   (setq new-setback
             (calculate-new-setback setback flst rflst new-sblarea))
   (if (eq 0 new-setback)            ;;checks if new-setback is OK
     (setq new-setback setback))
   (setq sbfar-fix (fix sbofar))
   (setq no (+ no0 sbfar-fix))
   (setq usedh (+ h1 (* no ho)))
   (draw-regs ptlst flst rflst lhandle '"MAXIMUM" '"3DF-MAXIMUM"
              h1 usedh new-setback)
   (putparam-lst "REGS-TYPE"
                 (list '"BRA MAXIMUM" 0 usedh new-setback no 0)
                 abhandle)
   )
  ((<= sbofar sbnr)      ;;farmax more restrictive than hlmax, builds
   ;;residences on top of offices using all farmax, tower might change
   (setq new-sblarea (/ (* sbofar sblarea) (1+ (fix sbofar))))
   (setq new-setback
             (calculate-new-setback setback flst rflst new-sblarea))
   (if (eq 0 new-setback)            ;;checks if new-setback is OK
     (setq new-setback setback))
   (setq sbfar-fix (fix sbofar))
   (setq no no0    ;;no need to optimize, it will only get one office
         nr sbfar-fix)
   (setq usedh (+ h1 (* nr hr) (* no ho)))
   (draw-regs ptlst flst rflst lhandle '"MAXIMUM" '"3DF-MAXIMUM"
```

```
                hl usedh new-setback)
       (putparam-lst "REGS-TYPE"
                     (list '"BRA MAXIMUM" 0 usedh new-setback no nr)
                     abhandle)
       )
      ((<= sbrfar sbno)      ;;farmax more restrictive than hlmax, builds
       ;;offices on top of residences using all farmax, tower's shape
       ;;might change
       (setq new-sblarea (/ (* sbrfar sblarea) (1+ (fix sbrfar))))
       (setq new-setback
                 (calculate-new-setback setback flst rflst new-sblarea))
       (if (eq 0 new-setback)            ;;checks if new-setback is OK
         (setq new-setback setback))
       (setq sbfar-fix (fix sbrfar))
       (setq no sbfar-fix
             nr (+ nr0 sbfar-fix))
       (setq usedh (+ hl (* nr hr) (* no ho)))
       (draw-regs ptlst flst rflst lhandle '"MAXIMUM" '"3DF-MAXIMUM"
                  hl usedh new-setback)
       (putparam-lst "REGS-TYPE"
                     (list '"BRA MAXIMUM" 0 usedh new-setback no nr)
                     abhandle)
       )
      ((<= sbrfar sbnr) ;;farmax more restrictive than hlmax, builds all
       ;;residences (optimizing offices)  using all farmax, tower might
       ;;change
       (setq new-sblarea (/ (* sbrfar sblarea) (1+ (fix sbrfar))))
       (setq new-setback
                 (calculate-new-setback setback flst rflst new-sblarea))
       (if (eq 0 new-setback)            ;;checks if new-setback is OK
         (setq new-setback setback))
       (setq sbfar-fix (fix sbrfar))
       (setq no (fix (/ (- hlmax hl (* (+ nr0 sbfar-fix) hr)) (- ho hr))))
       (setq nr (- (+ nr0 sbfar-fix) no)
             usedh (+ hl (* nr hr) (* no ho)))
       (draw-regs ptlst flst rflst lhandle '"MAXIMUM" '"3DF-MAXIMUM"
                  hl usedh new-setback)
       (putparam-lst "REGS-TYPE"
                     (list '"BRA MAXIMUM" 0 usedh new-setback no nr)
                     abhandle)
       )
      (T   ;;hlmax more restrictive than farmax, optimizes offices
       ;;case in which (< sbnr sbrfar)
       (setq no (fix (/ (- hlmax hl (* (+ nr0 (fix sbnr)) hr))
                        (- ho hr))))
       (setq nr (- (+ nr0 (fix sbnr)) no)
             not-used-area (- (* lalea (- farmax 1 nr0))
                              (* sblarea (fix sbnr))))
       (draw-regs ptlst flst rflst lhandle '"MAXIMUM" '"3DF-MAXIMUM"
                  hl hlmax setback)
       (putparam-lst "REGS-TYPE"
                     (list '"BRA MAXIMUM"
                           not-used-area hlmax setback no nr)
                     abhandle)
       )
      ))
  ))
((and (eq 0 hl) (eq 0 hlmax) (eq 0 setback) (eq 0 far) (eq 0 farmax))
```

```
    (princ "\n...No Zoning Parameters for this Lot..."))
   (T (princ
       "\n...Combination of Zoning Parameters Not Known by The System..."))
   )
  )


;;setback-facade returns the new list of polygon vertices set back from the
;;facade

(defun setback-facade (sb flst rflst / ffpt flst-sb rfpt)
  (setq ffpt (car flst))  ;;first facade point

  (if (eq '() rflst);;all block is set back, or block has neighboring walls
    (setq flst-sb (list (sb-int-pt (last flst) sb (car flst)
                                   sb (cadr flst))))
    (setq flst-sb (list (sb-int-pt (last rflst) 0 (car flst)
                                   sb (cadr flst)))))

  (while (not (eq '() (caddr flst)))      ;;flst has more than 3 points
         (setq flst-sb (append flst-sb
                               (list (sb-int-pt (car flst) sb (cadr flst)
                                                sb (caddr flst))))
               flst (cdr flst)))

  (if (eq '() rflst);;all block is set back, or block has neighboring walls
    (setq flst-sb (append flst-sb
                          (list (sb-int-pt (car flst) sb (cadr flst)
                                           sb ffpt))))
    (setq flst-sb (append flst-sb
                          (list (sb-int-pt (car flst) sb (cadr flst)
                                           0 (car rflst))))))

  (append flst-sb rflst)
  ;;(apply 'command (append (cons '"PLINE" (append flst-sb rflst)) '("c")))
  )


;;check-plate checks that the set back lot does not have any negative area
;;and returns the new polygon vertices set back from the facade

(defun check-plate (plst / new-plst pt2 pt3 pt4 pti)
  (setq new-plst '())
  (if (not (cddddr plst))     ;;checks if it has more than 4 vertices
    (setq new-plst plst)
    (progn
      (while (cdddr plst)
             (setq pti (inters (car plst) (cadr plst)
                               (caddr plst) (cadddr plst) T))
             (if pti
               (setq new-plst (append new-plst (list (car plst)))
                     plst (cons pti (cdddr plst)))
               (setq new-plst (append new-plst (list (car plst)))
                     plst (cdr plst))))
      (if (not (cddr plst))
        (if (setq pti (inters (car plst) (cadr plst)
                              (car new-plst) (cadr new-plst) T))
          (setq new-plst (append (list pti) (cdr new-plst)
```

```
                                (list (car plst))))
          (progn
            (setq new-plst (append new-plst (list (car plst))))
            (if (setq pti (inters (cadr plst) (car new-plst)
                                  (cadr new-plst) (caddr new-plst) T))
              (setq new-plst (append (list pti)
                                     (cddr new-plst)
                                     (list (cadr plst))))
              (setq new-plst (append new-plst (list (cadr plst)))))))
        (progn
          (setq pt2 (car plst)
                pt3 (cadr plst)
                pt4 (caddr plst))
          (if (setq pti (inters pt2 pt3 pt4 (car new-plst) T))
            (progn
              (setq new-plst (append new-plst (list pt2)))
              (if (setq pti2 (inters pti (car new-plst)
                                     (cadr new-plst) (caddr new-plst) T))
                (setq new-plst (append (list pti2) (cddr new-plst )
                                       (list pti)))
                (setq new-plst (append new-plst (list pti)))))
            (progn
              (setq new-plst (append new-plst (list pt2)))
              (if (setq pti (inters pt3 pt4
                                    (car new-plst) (cadr new-plst) T))
                (setq new-plst (append (list pti) (cdr new-plst)
                                       (list pt3)))
                (progn
                  (setq new-plst (append new-plst (list pt3)))
                  (if (setq pti (inters pt4 (car new-plst)
                                        (cadr new-plst) (caddr new-plst) T))
                    (setq new-plst (append (list pti)
                                           (cddr new-plst)
                                           (list pt4)))
                    (setq new-plst (append new-plst (list pt4)))
                    ))))))))
    ))
  new-plst
  )

;;sb-int-pt returns the intersection point of 2 lines set back of a
;;distance sb clockwise

(defun sb-int-pt (pt0 sb01 pt1 sb12 pt2 / pt0-sb pt10-sb pt12-sb pt2-sb)
  (setq pt0-sb (list (+ (car pt0) (* sb01 (cs-ang 's pt0 pt1)))
                     (- (cadr pt0) (* sb01 (cs-ang 'c pt0 pt1)))
                     (caddr pt0))
        pt10-sb (list (+ (car pt1) (* sb01 (cs-ang 's pt0 pt1)))
                      (- (cadr pt1) (* sb01 (cs-ang 'c pt0 pt1)))
                      (caddr pt1))
        pt12-sb (list (+ (car pt1) (* sb12 (cs-ang 's pt1 pt2)))
                      (- (cadr pt1) (* sb12 (cs-ang 'c pt1 pt2)))
                      (caddr pt1))
        pt2-sb (list (+ (car pt2) (* sb12 (cs-ang 's pt1 pt2)))
                     (- (cadr pt2) (* sb12 (cs-ang 'c pt1 pt2)))
                     (caddr pt2)))
  (inters pt0-sb pt10-sb pt12-sb pt2-sb nil)
  )
```

```
;;cs-ang returns sine or cosine given 2 points

(defun cs-ang (cs pt0 pt1 / x0 y0 x1 y1 detx dety det)
  (setq x0 (car pt0)
        y0 (cadr pt0)
        x1 (car pt1)
        y1 (cadr pt1))
  (setq detx (- x1 x0)
        dety (- y1 y0))
  (if (eq cs 'c)
    (setq det detx)    ;;for cosine
    (setq det dety))   ;;for sine
  (/ det (sqrt (+ (* detx detx) (* dety dety))))
  )


;;calculate-new-setback calculates the new setback and returns it according
;;to area which is new-sblarea.  Note that an initial impossible setback
;;has to be handled before calling this procedure
;;Note that this procedure does not checks the set back plate to speed
;;computation, this assumes that the possible error when computing the area
;;would be very small

(defun calculate-new-setback (setback flst rflst area /
                              lim0 lim1 new-sb new-area)
  (if (< area min-floor-plate)   ;;this might be changed according to needs
    0
    (progn
      (setq lim0 setback
            lim1 (if rflst    ;;try several values for an approximation
                   (* 25 setback)
                   (* 10 setback)))
      (setq new-area (area-polygon
                       (setback-facade (/ (+ lim1 lim0) 2.0) flst rflst)))
      (while (not (equal area new-area 1))
             (setq new-sb (/ (+ lim1 lim0) 2.0))
             (setq new-area (area-polygon
                                (setback-facade new-sb flst rflst)))
             (if (< new-area area)
               (setq lim1 new-sb)
               (setq lim0 new-sb)))
      new-sb))
  )


;;area-polygon returns the area of a given polygon list (a closed polyline)

(defun area-polygon (ptlst / area xold yorig yold)
  (setq area 0.0
        xold (caar ptlst)
        yorig (cadar ptlst)
        yold 0.0)
  ;;(apply 'command (append '("AREA") ptlst '("")))
  ;;(princ (getvar "AREA"))
  (while ptlst
         (setq area (+ area (* (- xold (caar ptlst))
                               (+ yold (- (cadar ptlst) yorig))))
               xold (caar ptlst)
```

```
                yold (- (cadar ptlst) yorig)
                ptlst (cdr ptlst))
          )
   (/ area -2.0)
   )

 (defun C:POLAREA (/ ptlst vname)
   (setq ptlst '()
         vname (entnext (car (entsel "\nSelect a Lot's Polyline: "))))
   (while (not (eq "SEQEND" (cdr (assoc 0 (entget vname)))))
          (setq ptlst (append ptlst (list (cdr (assoc 10 (entget vname)))))
                vname (entnext vname)))
   (princ "\nThe Area of the Lot is: ")
   (princ (area-polygon ptlst))
   (prin1)
   )


;;draw-regs draws lot's regulations

 (defun draw-regs (ptlst flst rflst lhandle reg-layer 3df-layer
                         hl hlmax setback)
   (command "LAYER" "m" reg-layer "")
   (command "CHANGE" (handent lhandle) "" "P" "T" hl "")
   (command "LAYER" "m" 3df-layer "")
   (poly3dface (add-height-lst hl ptlst))

   (if (not (or (eq 0 hlmax) (eq 0 setback)))
     (progn        ;;check-plate is invoqued so as to draw it well
       (setq ptlst (check-plate (setback-facade setback
                                                (add-height-lst hl flst)
                                                (add-height-lst hl rflst))))
       (command "LAYER" "m" reg-layer "")
       (apply 'command (append '("PLINE") ptlst '("c")))
       (command "CHANGE" (entlast) "" "P" "T" (- hlmax hl) "")
       (command "LAYER" "m" 3df-layer "")
       (poly3dface (add-height-lst (- hlmax hl) ptlst))
       ))
   )


;;add-height-lst adds the given height to the given list and returns the
;;new list

(defun add-height-lst (h ptlst)
  (mapcar '(lambda (pt) (list (car pt) (cadr pt) (+ h (caddr pt))))
          ptlst))
```

```
;;putparam-lst stores the specified attribute values given the attributes
;;block handle it considers the list already ordered and stores attributes
;;starting with the first parameter until valuelst is nil

(defun putparam-lst (firstparam valuelst abhandle / ename)
  (setq ename (entnext (handent abhandle)))
  (if (eq (cdr (assoc 0 (entget ename))) "SEQUEND")
    '()
    (if (eq (cdr (assoc 2 (entget ename))) firstparam)
      (progn
        (entmod (subst (cons 1 (car valuelst))
                       (assoc 1 (entget ename))
                       (entget ename)))
        (setq valuelst (cdr valuelst)
              ename (entnext ename))
        (while (not (eq '() valuelst))
               (entmod (subst (cons 1 (rtos (car valuelst) 2 0))
                              (assoc 1 (entget ename))
                              (entget ename)))
               (setq valuelst (cdr valuelst)
                     ename (entnext ename))
               )
        )
      (putparam-lst firstparam valuelst (cdr (assoc 5 (entget ename))))))
  )


;;poly3dface builds a 3DFACE surface given a clokwise list of polyline
;;points

(defun poly3dface (ptlst / pt0 pt1 pt2 pt3 rpt1 rpt2 rpt3)
  (if (eq '() (cddddr ptlst))
    (progn
      (setq pt0 (car ptlst)
            pt1 (cadr ptlst)
            pt2 (caddr ptlst)
            pt3 (cadddr ptlst))
      (command "3DFACE" pt0 pt1 pt2 pt3 ""))
    (progn
      (setq rev0ptlst (cons (car ptlst) (reverse (cdr ptlst))))
      (setq pt0 (car ptlst)
            pt1 (cadr ptlst)
            pt2 (caddr ptlst)
            pt3 (cadddr ptlst)
            rpt1 (cadr rev0ptlst)
            rpt2 (caddr rev0ptlst)
            rpt3 (cadddr rev0ptlst))
      (cond
       ((and (eq '() (check-inters (list pt0 pt2) (cdddr ptlst)))
             (eq 1 (check-area (list pt0 pt1 pt2)))
             (eq 1 (check-area (list pt0 pt2 pt3))))
        (command "3DFACE" pt0 pt1 pt2 "" "")
        (poly3dface (append (list pt0) (cddr ptlst))))
       ((and (eq '() (check-inters (list pt0 rpt2) (cdddr rev0ptlst)))
             (eq '() (check-area (list pt0 rpt1 rpt2)))
             (eq '() (check-area (list pt0 rpt2 rpt3))))
        (command "3DFACE" pt0 rpt1 rpt2 "" "")
        (poly3dface (append (list pt0) (reverse (cddr rev0ptlst)))))
```

```
        (T (poly3dface (append (cdr ptlst) (list (car ptlst)))))
        ))
    )
  )


;;check-inters returns nil if the line does not intersect with the given
;;list of lines, or it returns the first intersection point if it does
;;intersect

(defun check-inters (lnlst ptlst)
  (if (< (length ptlst) 2)
    '()
    (if (inters (car lnlst) (cadr lnlst) (car ptlst) (cadr ptlst) T)
      1
      (check-inters lnlst (cdr ptlst))))
  )


;;check-area returns nil if the area of the triangle is negative and 1 if
;;positive area is negative when vertices are given clockwise the following
;;procedure assumes that lot vertices were entered clockwise

(defun check-area (trilst / xt0 yt0 xt1 yt1 xt2 yt2 areatri)
  (setq xt0 (car (car trilst))
        yt0 (cadr (car trilst))
        xt1 (car (cadr trilst))
        yt1 (cadr (cadr trilst))
        xt2 (car (caddr trilst))
        yt2 (cadr (caddr trilst))
        )
  (setq areatri (+ (* (- xt0 xt2) (- yt1 yt0))
                   (* (- xt1 xt0) (- yt2 yt0))))
  (if (>= areatri 0)    ;;means counter-clockwise
    '()                 ;;area is negative
    1)                  ;;area is positive
  )


;;regulate-heuristically-ud

(defun regulate-heuristically-ud (what how)

  (princ "\nNot Yet Implemented, sorry")

  )
```

```
;;polygon-cg computes the C. G. of a given polygon list and returns the
;;list (xcg ycg), Center of Gravity coordinates
;;Note that polygon is a list of vertices counterclockwise defined and
;;closed!! Polygons are of the form:
;;(setq apoly (list (list 0 0) (list 10 0) (list 10 10) (list 0 10)
;;'("c")))

(defun polygon-cg (polygon / xcg ycg areasum xorg yorg xt1 yt1 xt2 yt2
                             areatri cg-pt)
  (setq xcg 0
        ycg 0
        areasum 0)
  (while (>= (- (1- (length polygon)) 3) 0)    ;;since (last polygon) = "c"
         (setq xorg (car (car polygon))
               yorg (cadr (car polygon))
               xt1 (car (cadr polygon))
               yt1 (cadr (cadr polygon))
               xt2 (car (caddr polygon))
               yt2 (cadr (caddr polygon)))
         (setq areatri (+ (* (- xorg xt2) (- yt1 yorg))
                          (* (- xt1 xorg) (- yt2 yorg))))

         (setq xcg (+ xcg (* areatri (+ xt2 xt1)))
               ycg (+ ycg (* areatri (+ yt2 yt1)))
               areasum (+ areasum areatri)
               polygon (cons (car polygon) (cddr polygon)))
         )
  (setq cg-pt (list (/ (+ (/ xcg areasum) (car (car polygon))) 3)
                    (/ (+ (/ ycg areasum) (cadr (car polygon))) 3)))
  ;;(command "point" cg-pt)   ;;this draws the C.G. Point
  cg-pt
  )


;;print message after loading DeaRB.lsp

(princ "\n...DeaRB.lsp Loaded...")
(prin1)
```

Bibliography

ABELSON, H.; SUSSMAN, P., Structure and Interpretation of Computer Programs, MIT Press, Cambridge, 1985.

ADAMS, HOWARD and OPPERMANN, Back Bay Development Plan 1967, Report for BRA and Back Bay P&D Corp., 1967.

ALEXANDER, Christopher, Notes on the Synthesis of Form, Harvard University Press, Cambridge, London, 1964.

BOSTON REDEVELOPMENT AUTHORITY, Downtown Zoning Midtown Cultural District Plan, 1989.

BOSTON REDEVELOPMENT AUTHORITY, Midtown Cultural District Plan, Plan to Manage Growth, 1989.

BOSTON REDEVELOPMENT AUTHORITY, The Hinge Block Plan, 1990.

BROADBENT, G.; WARD, A., Design Methods in Architecture, Symposium Portsmouth School of Arch., Lund Humphries, London, 1969.

BURROUGH, P. A., Principles of Geographical Information Systems for Land Resources Assessment, Oxford University Press, N.Y., 1986.

COOPER, David, Metaphor, Aristotelian Society Series Volume 5, Basil Blackwell, Oxford, 1986.

EASTMAN, Charles, "Fundamental problems in the development of computer-based Architectural Models." Presented at Computers and Design Research Symposium, M.I.T., Cambridge, MA.,1986.

ERVIN, Stephen, "The structure and function of diagrams in environmental design: a computational inquiry", Ph.D Dissertation, MIT Department of Urban Studies and Planning, 1989.

Design Methods Group, Application of Systematic Methods to Design., Donald P Grant, Berkeley, CA., July 7-9, 1975, 1976, 1984-85, 1986, 1988.

GHYKA, Matila, A Practical Handbook of Geometrical Composition and Design, Alec Tiranti Ltd., London, 1952.

GHYKA, Matila, Essai sur le Rhythme, Gallimard, Paris, 1938.

GHYKA, Matila, Esthétique des Proportions dans la Nature et dans les Arts, Gallimard, Paris, 1927.

GHYKA, Matila, The Geometry of Art and Life, Sheed and Ward, Inc., N.Y., 1946.

GOODMAN, Nelson, Languages of Art: An Approach to the Theory of Symbols, The Bobbs-Merill Company, Inc., New York, N.Y., 1968.

GROSS, Mark, "Design as the Exploration of Constraints", PhD Dissertation, MIT Department of Architecture, 1985.

GROSS, Mark, Ervin, Stephen, Anderson, James, Fleisher, Aaron, "Constraints: Knowledge representation in design." In Design Studies/Volume 9,Number 3, 1988.

HABRAKEN, John, The Appearance of the Form, Awater Press, Cambridge, MA., 1985.

HILLIER, Bill, "How is Design Possible?: a Sketch for a Theory" , in the Design Activity International Conference, 1973.

HUBBARD, William, Complicity and Conviction: Steps toward an Architecture of Convention, .M.I.T. Press, Cambridge, MA., 1986.

JOHNSON, Timothy, "A three-dimensional graphical design system" M.I.T. Lincoln Laboratory, Lexington, MA., 1963.

KALAY, Yehuda, (Ed.), Principles of Computer-Aided Design: Computability of Design, John Wiley & Sons, Inc., N.Y., 1987.

KALAY, Yehuda, (Ed.), Graphic Introduction to Programming, John Willey & Sons, Inc., N.Y., 1987.

KEPES Gyorgy, etc., Module, Proportion, Symmetry, Rhythm, George Braziller, N.Y., 1966.

KRIPKE, Saul A., Wittgenstein on Rules and Private Language, Harvard University Press, Cambridge, MA., 1982.

KROLL, Lucien, An Architecture of Complexity, 1987, The MIT Press, Cambridge, MA.

KWARTLER, Michael, Daylight as a Zoning Device for Midtown ID No. T-52-6B, Kwartler Assoc. and Rensselaer Poly. NY.

LAI, Richard Tseng-yu, Law in Urban Design and Planning The Invisible Web, Van Nostrand Reinhold Company Inc., England, 1988.

LYNCH, Kevin and HACK, Gary, Site Planning, The MIT Press, Cambridge, MA., 1984.

MAC CORMAC, Earl, A Cognitive Theory of Metaphor, The MIT Press, Cambridge, MA., 1985.

MARK, Earl, "The Sagrada Familia and the Electronic Design Studio Project", Presented at ARECADO 89, Barcelona, Spain, 1989.

MITCHELL, William, Computer Aided Architectural Design, Petrocelli-Charter, New York, NY., 1977.

MITCHELL, LIGGETT, KVAN, The Art of Computer Graphics Programming (A Structured Introduction for Architects and Designers), Van Nostrand Reinhold Company, New York, NY., 1977.

MITCHELL, William, The Logic of Architecture (Design, Computation and Cognition), The MIT Press, Cambridge, MA.,1990.

MOUDON, Anne Vernez, Build for Change, Neighborhood Architecture in San Francisco, MIT Press, Cambridge, MA., 1986.

MUNARI, Bruno, Discovery of the Square , George Wittenborn, Inc. N.Y., 1965.

MUNARI, Bruno, The discovery of the Circle, George Wittenborn, Inc. N.Y., 1965.

NEWELL, Allen, 1980, "The Knowledge Level", in AI Magazine, Summer 1981.

NEW YORK CITY PLANNING COMMISSION, Zoning For Housing Quality, 1975.

PERKINS, D.N., The Mind's Best Work, Harvard University Press, Cambridge, London, 1891.

PORTER, William, "Notes on the inner logic of designing: Two thought experiments." In Design Studies/Volume 9,Number 3, 1988.

ROWE, Peter, Design Thinking, Cambridge,Massachusetts: M.I.T. Press, 1987.

SCHMITT, Gerhard, Microcomputer Aided Design for Architects and Designers, John Wiley & Sons, N.Y., 1988

SCHÖN, Donald, "Designing: Rules, Types and Worlds" in Design Studies Volume 9, Number 3, 1988.

SCHÖN, Donald , Wiggins, Glen, "Kinds of Seeing and Their Function in Designing", Forthcoming, University of Manchester, 1990.

SCHUSTER, SIMMONDS, FRENCHMAN, etc, Housing Design and Regional Character, A Premier for New England Towns, Environmental Design and Dev. Group, 1988.

SEKLER, E; BUCHWALD H; GREGORY A., Proportion, a Measure of Order, Carpenter Center for V.A., Harvard U., 1965.

SIMON, Herbert, The Sciences of the Artificial, The MIT Press, Cambridge, MA., 1969.

SUTHERLAND, Ivan, "Structure in drawings and the hidden surface problem." In ed. Negroponte, Nicholas, Computer Aids to Design and Architecture, Petrocelli-Charter, New York, NY., 1975.

TZONIS, Alexander , Lefaivre, Liane, Classical Architecture: the Poetics of Order, 1987, The MIT Press, Cambridge, MA.

ULRICH, Karl Thatcher, "Computation and Pre-parametric Design", PhD Dissertation, MIT Department of Mechanical Engineering, 1987.

VALDES-PEREZ, Raul E., "Knowledge-Based Schematic Drafting: Aesthetic Configuration as a Design Task", Working Paper 292, MIT Artificial Intelligence Laboratory, 1987.

WINSTON, Patrick Henry, Artificial Intelligence (2nd Edition), Addison Wesley, 1984.