

RELATIONAL MODELING

Mark D. Gross
Design Technology Research
14 Centre Street
Cambridge Massachusetts 02139

ABSTRACT

Today's computer assisted design (CAD) systems automate traditional ways of working with tracing paper and pencil, but they cannot represent the rules and relationships of a design. As hardware becomes faster and memory less expensive, more sophisticated fundamental software technologies will be adopted. This shift in the basis of CAD will provide powerful capabilities and offer new ways to think about designing.

Recently parametric design, a technique for describing a large class of designs with a small description in code, has become a focus of attention in architectural computing. In parametric CAD systems, design features are identified and keyed to a number of input variables. Changes in the input values result in variations of the basic design. Based on conventional software technologies, parametric design has been successfully applied in many design domains including architecture and is supported by several commercial CAD packages. A weakness of parametric techniques is the need to predetermine which properties are input parameters to be varied and which are to be derived.

Relational modeling is a simple and powerful extension of parametric design that overcomes this weakness. By viewing relations as reversible rather than one-way, any set of properties can be chosen as input parameters. For example, a relational model that calculates the shadow length of a given building can also be used to calculate the building height given a desired shadow length. In exercising a relational model the designer is not limited to a pre-selected set of input variables but can explore and experiment freely with changes in all parts of the model.

Co is a relational modeling environment under development on the Macintosh-II computer, and Co-Draw, a prototype CAD program based on Co. Co's relational engine and object-oriented database provide a powerful basis for modeling design relations. Co-Draw's interactive graphics offer a flexible medium for design exploration. Co provides tools for viewing and editing design models in various representations, including spreadsheet cards, tree and graph structures, as well as plan and elevation graphics. Co's concepts and architecture are described and the implications for design education are discussed.

TODAY'S COMPUTER ASSISTED DESIGN

Computer tools have only begun to support architectural designing from schematic design through construction. Good at drafting and rendering tasks, they allow users to construct and edit drawings by using primitive geometric objects and parts from libraries. Current technology permits designers to work with layers, and to zoom in and out while working on a drawing.

This project began with my Ph.D. dissertation at M.I.T, where S. Ervin and A. Fleisher collaborated on previous related research^[3,4]. The development team for this project included C. Fry, J. Habraken, and M. Ruano. I am indebted to A. Dula, J. Nilsson, M. Ruano, and H. Sayed for comments on earlier drafts of this paper. Support from Shimizu Construction Corporation, Coral Software Inc. and Apple Computer's Cambridge Advanced Technology Group is gratefully acknowledged.

Combined with three-dimensional construction and viewing, these capabilities comprise the repertoire of most commercial packages on the architectural CAD market today.

Although today's CAD tools automate many aspects of drafting, they do not let us record the reasons for the decisions we make — the rules and relationships that govern the design. A drawing is inherently static; a design is dynamic. Drafting and rendering, while essential to architectural design, represent only the “external” aspects of designing. A complex “internal” process of reasoning and judgement lies behind every design decision. Even such a simple change as moving a window can cause far-reaching effects in a design. A drawing cannot convey these design dependencies; therefore we keep them in our head. *It would be a significant advance if computer-based design tools would enable us to record the design relationships we intend, along with the specific decisions that accomplish these intentions.*

NEED FOR SMARTER DRAFTING TOOLS

Here are three examples that illustrate the limitation of today's computer-based drafting tools: It is easy to draw two circles and make a line segment tangent to both. But once the drawing is made we cannot move or resize one of the circles and expect the line segment to adjust to maintain the tangency. Rather, we must repeat the “make tangent” operation after moving or resizing one of the circles. Likewise, in most CAD programs it is easy to select two elements and align them, for example, along their top edges. But the alignment will not be preserved if one of the elements is subsequently moved or resized. A final example is grid-gravity in which elements snap to center (or align) on grid lines. We might use this feature to locate columns at crossings in a structural grid. In today's drafting programs when we change the grid dimensions, grid lines move while elements remain as originally positioned. If elements “knew” to move with the grid then columns would remain located at grid crossings as intended.

These examples make clear the distinction between a drafting *operation* and a design *relation*. Of course, sometimes we only want to perform a one-time operation. But often we would like to declare the tangency, the alignment, or the grid-gravity as a relation, *to be remembered and maintained dynamically by the computer* as we edit the design: moving, resizing, and rotating elements.*

PARAMETRIC DESIGN

Parametric design tools[†] allow the architect to describe a family of designs which vary according to certain key design variables, or parameters. A simple example is a parametric model for a stair,

* We sometimes annotate design drawings with instructions like “align these walls” or “keep 6' minimum clearance”. These instructions are more effective at conveying design intent than measured dimensions, with which a subsequent change may inadvertently cause a desired relationship to be lost.

[†] PASCAL programs to implement parametric design techniques are discussed in Mitchell et al. [9] Parametric design templates can also be constructed using macro or programming language facilities of conventional drafting programs. Recently, commercial packages with parametric capabilities have appeared: on the high-end are ICAD's Design Language product and Wisdom Systems "Concept Modeller", which require a designer to write code to describe a family of designs. On the low end, the "Synthesis" package provides parametric design capabilities to AutoCad.

with height between floors and riser height as inputs. We can make a “dataflow diagram” of the model with input parameters on the left and output parameters on the right:

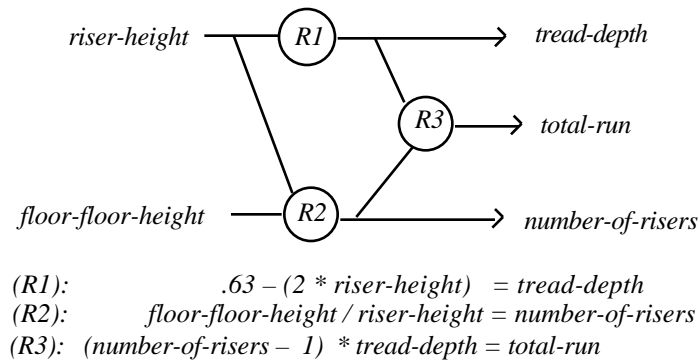


Figure 1. Dataflow diagram of a parametric stair model

A standard formula (*R1*) assures a comfortable stair. Dividing the height between floors by the riser-height gives the number of risers (*R2*), from which the total stair run can be computed (*R3*). The parametric model is really a *procedure* for designing — in this case — a stair.

A parametric model is appropriate for routine design, where we want to rapidly generate design alternatives according to a formula worked out in advance. But for conceptual design, which is characterized by interactive exploration, parametric modeling is less useful because in conceptual design *we don't want to determine in advance which properties of a design we will use as input parameters*. Suppose we need to design a stair with a given tread depth, total run, or number of steps. Our original model won't serve; its input parameters are riser-height and height between floors. We can work around this problem by building several parametric models expressing the same stair relationships, differing only in which properties are input parameters. But there are simpler and more efficient solutions.

RELATIONAL MODELING AND CONSTRAINTS

What we need (and what is lacking in parametric design) is a single model that can be used in several different ways, depending on which properties are to be set as inputs and which are to be calculated as outputs. This would allow us to retain desired flexibility in decision-making for conceptual design. The model would define only the relations between the stair's properties (and not which properties are to “drive” the model); at insertion time we must set a sufficient number of parameters to specify the new stair instance. (Our diagram for the relational stair would look like figure 1 without the arrows indicating the direction of data-flow.) We would use the same model to make a stair using any of the variables: floor height, riser height, tread width, number of steps, total run, or stair slope, as input parameters. This key idea, *allowing any of the variable properties of a design to become input parameters*, distinguishes relational modeling from parametric design.

Relational modeling, also called “variational” CAD, uses a software technology called “constraint-based programming”^[6], first explored in Sutherland's famous Sketchpad program^[14].

Research [2, 7, 8, 12, 13] has developed the concept to the point where mechanical engineering (MCAE) applications* software based on relational modeling has begun to appear.

In a relational model, each design object is described by *variables* and *relations*. Variables represent the object's properties, and relations, its behavior. Variables and relations are connected in a network similar to the diagram of a parametric model shown in figure 1. Unlike a parametric model, however, inputs and outputs are not determined when the model is defined, and hence the direction of data-flow through the network depends on the sequence in which variable values are supplied. The designer may freely set and change values of design variables in any order, as well as to improve or modify the model by adding and deleting relations at any time throughout the design process. Combined with object-oriented database and programming techniques, relational modeling offers a powerful and flexible medium for building computer assisted design tools.

CO-DRAW

Co-Draw is a prototype for demonstrating relational modeling as a basis for computer-assisted architectural design. However relational modeling techniques are not limited to graphics — they can be employed in a wide range of domains including engineering calculations, scheduling and facilities management, and financial planning. This section describes an interactive graphics environment, Co-Draw, built using the Co relational modeling language. Figure 2 shows Co-Draw's user-interface. Drawing takes place in one or more Work Sheet windows. Command and relations menus are provided, along with a click-and-drag interface for moving and resizing elements. The part-structure, or assembly model of the design is viewed and edited in a Part Graph window.

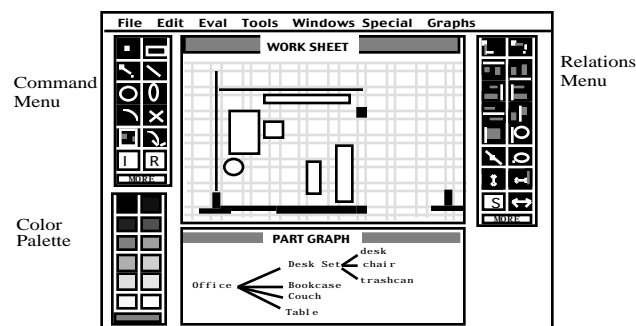


Figure 2. the Co user interface

Using the command menu we enter primitive graphic elements: points, lines, line-segments, rectangles, circles, arcs, and poly-lines. (Grids are defined and deployed using a separate Grid Manager module). The command menu also provides facilities for grouping, rotating, and inspecting elements. The relations menu offers predefined geometric relations for graphic elements including alignments and abutments, centering and edge-offsets, and dimension ratios.

* Perhaps the best known is Cognition's Mechanical Advantage program. Other products that employ relational modeling techniques are described in a recent article in Computer Graphics World [11].

Co-Draw supports relations as discussed above. If we apply "align tops" to two elements in the Work Sheet, Co-Draw maintains the relation: when we move one element, the other moves to keep the tops aligned. The relation is displayed graphically (as a red line across the element tops) but can be hidden on request. If we draw two circles and a line-segment, then apply the "tangent" relation between each circle and the segment, Co maintains the relation dynamically as we move and resize the circles. And if we fix elements to a grid, they remain in their original relationship to the grid (centered, aligned, etc.) as we change the grid-unit dimensions.

Example: abutments and alignments

A brief example shows how Co-Draw maintains geometric relations as we apply them from the menu. First we draw three rectangles (A). Next we establish relations to keep the rectangles abutting horizontally (B) but which leave them free to move vertically (C). Adding top alignment relations fixes the rectangles' relative positions (D). But if we allow the middle rectangle to vary in size then dragging the end rectangles apart will stretch the middle one to satisfy the abutment relations (E).

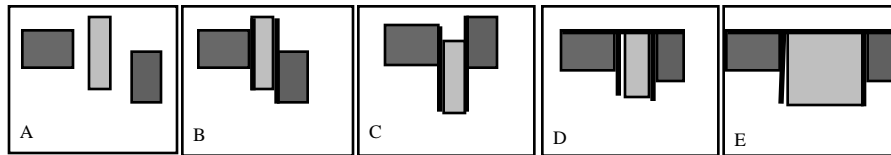


Figure 3. Co maintains alignments and abutments

Example: a simple shed design

We might construct the simple shed design shown in figure 4, entering graphic primitives and applying relations from the menus. We draw a rectangle for the slab first and then two rectangles for walls, and enter vertical abutments (**R1**, **R2**) to fasten them to the slab. As we apply the relations, the elements move and resize themselves. We fix offset distances between the walls and the edge of the slab (**R3**, **R4**). Next, we enter a sloped line for the roof, and a "fixed slope" relation (**R5**). The roof and walls are not connected, so we add beams resting on top (**R6**) of each wall, supporting (**R7**) the roof and centered (**R8**) on the wall. Finally we add an overhang relation (**R9**) to keep the roof extending past the slab.

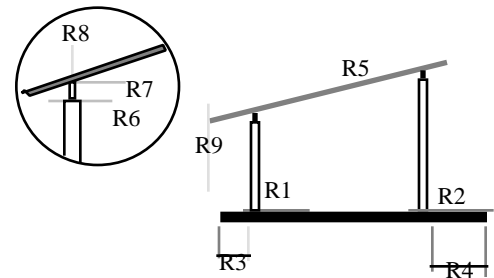


Figure 4. A simple shed design

Because of these relations, the shed design will "behave" in certain ways as we change dimensions and positions of its parts. If we raise the roof, the walls will become taller so as to maintain relations **R6** and **R7**. Or, if we extend the slab towards the left, the shorter (leftmost) wall moves with it, to maintain the position relation **R3**. In order to maintain the roof's fixed slope **R5** and support relations **R6** and **R7**, the wall height must decrease, and the roof extends to maintain the overhang relation **R9**.

We can continue to add elements and relations to the design. For example, we can add minimum and

add a loft whose interior extension is related to the height from its top to the roof (fig 5).

In this simple example we began with a simple drawing and, by applying relations between its parts, we built into it *design behavior*. Through applying the relations,

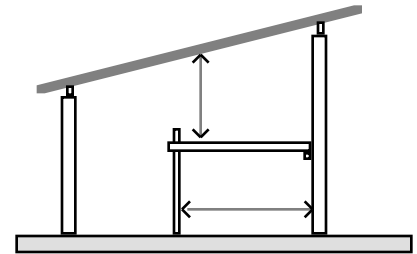


Figure 5. Adding a loft

lines and rectangles take on meaning; they begin to behave like the building elements that we intend them to represent. This process of assigning meaning and behavior to lines and symbols is essential to the use of drawings in design and is by no means a new concept introduced with computers. Whenever we communicate by drawing we rely on a shared understanding of the marks we make on paper, and learning to operate with this code is central to design education.

Notice that in constructing this simple model we had no need to write, compile, link, or debug any computer code; *yet we have programmed the behavior of our shed design*. We have proceeded entirely by entering graphic primitives and applying relations from the Co-Draw menus. Code describing the design behavior was generated automatically (and we can now view and edit it) but we constructed the model by adding relations interactively to the drawing.

THE Co RELATIONAL MODELING LANGUAGE

Graphic elements and relations in Co-Draw are defined in the Co relational modeling language. The language integrates an object-oriented database and a “reversing” spreadsheet. The object-oriented database organizes element descriptions in a hierarchic class structure; individuals inherit default properties and behavior from class definitions. The “reversing” spreadsheet provides two-way calculation; we can work forward from design decisions to calculate performance, or backward from performance specifications to determine appropriate settings of design variables. Special “card” and “graph” windows for viewing the data-structures provide an interactive user-interface for programming. We can set and modify behavior of design elements as we explore consequences of decisions. This section introduces principal features of this language with simple examples.

Classes and Individuals

Every Co element is either a *class* or an *individual*. A class defines variables (properties) and relations (behavior) for all its members. A tree-like taxonomy organizes class definitions with specific information added in subclasses, general information stored at higher levels. For example, the class “unit-masonry” (see figure 6) describes general characteristics inherited by subclasses “brick” and “concrete-block,” each of which specifies a material value inherited by its subclasses which specify dimensions, color, glazing surfaces, etc. The Library Graph window displays the structure of element classes, and we can make new definitions in this structure interactively. For example, we could add a new “shed” class based on the example above.

At the lowest level of this “inheritance hierarchy”, every individual element is a member of one or more classes. Individuals add specific information (such as position) to the default

values and relations they inherit from their classes. An individual may also *override default values* from its classes, which makes it an *exception*^[1].

Every individual is also a *part*, either of some other individual, or of a top superpart element called “World”. A Part Graph window displays the hierarchy of part-whole relations among individuals in the design (see figure 2).

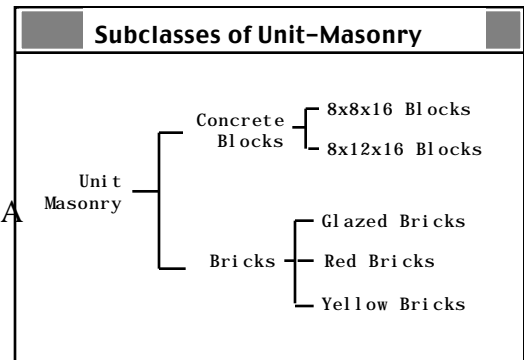


Figure 6. Library Graph

The part-structure of the design is retained when we open a configuration to edit a part; when we finish editing the part we can simply close the group (unlike with MacDraw, we do not need to “group” again). The part structure can also be edited directly in the graph window by adding and deleting links.

Spreadsheet Cards

Variables and relations for each individual and class are viewed and edited using special windows called “spreadsheet cards”. These cards enable us to interactively extend Co’s capabilities by defining new classes of elements and relations and by modifying and adapting built-in definitions. Variables Cards (see figure 7) display the variables that define elements’ properties: coordinates, dimensions, color, parts and superparts. We can fix and unfix (retract) values, add new variables and inquire about chains of inference that lead to derived values. The relations that define elements’ behavior are viewed and edited in Relations Cards. We can add and delete relations and inquire how a relation was derived. We can control the display to show only relations defined for a particular individual or to include all relations inherited from superclasses.

Example: adding Area calculation to Circles

A Circle is defined in Co as a relational object with three variables: CX and CY, defining its center coordinates, and RADIUS. We can extend this definition to include an AREA calculation. In the Relations Card for the class CIRCLE, we enter:

$$AREA = 3.14159 * RADIUS^2$$

Immediately all new and existing Circle individuals will calculate their area using this formula inherited from the CIRCLE class. Co’s spreadsheet is “reversing”; therefore the area relation can be used “backwards” to compute RADIUS as a function of AREA. If we enter a circle into the Work Sheet and view its Variables Card (figure 7) we see that its RADIUS is fixed (indicated by underlined typeface) and AREA is calculated.

If we change the value of CIRC-1’s AREA, Co recomputes RADIUS, switching it from “fixed” to “calculated” status, and updates the graphic view in the Work Sheet.

Variables of Circ-1	
AREA	6.2832
<u>CX</u>	30
<u>CY</u>	40
<u>RADIUS</u>	20

inspect
terms
why ?

Fix
Unfix
Rel s
Inits
C
I
A
D
X

Figure 7: Variables Card for a Circle individual showing fixed Radius and calculated Area

Example: Defining Element SubClasses

We can define new element types that inherit from the class RECTANGLE but which have additional properties or behavior. For example, we can make subclasses with default colors (BLUE-RECTANGLE) or dimensions (BIG-RECTANGLE, SMALL-RECTANGLE). We can define a SQUARE subclass with the additional relation HEIGHT = WIDTH. Or we can define a class of elements which automatically calculate their cost based on size. We define the class MASONRY-WALL as a rectangle with the relation:

$$\text{COST} = \text{LINEAR-COST} * \text{LENGTH}.$$

Next we fix a default LINEAR-COST value in the MASONRY-WALL class. As we enter individual MASONRY-WALL instances (each with a different LENGTH) into a design, each individual computes its COST, multiplying the *inherited* LINEAR-COST by *its own* LENGTH. When we change the lengths of individual MASONRY-WALLS, their COST values change dynamically*. If we change LINEAR-COST in the MASONRY-WALL class, individual COST values also change. We can further specialize MASONRY-WALL, making subclasses BRICK-WALL and CONCRETE-WALL, and fix different default LINEAR-COST values for each.

This example suggests how we could program Co-Draw to display constantly updated totals for area, number of window, costs, etc. This ability to calculate on-the-fly and display running totals while designing offers a significant improvement over current schemes, which require the designer to leave the drawing environment and run the design through a separate costing module.

Relations, not just equations

Although many important relations can be conveyed by algebraic equations, many others cannot. Inequality relations and tolerances, for example, are needed to represent minimum and maximum dimensions, placement of elements in zones, and other concepts in architectural and engineering design. Co interprets inequality relations as interval equations, and Co's arithmetic routines support calculations with interval or range values. By introducing the symbolic value ∞ (infinity) as a special number, Co represents the inequality $X \geq 20$ as an interval equation $X = [-20, \infty)$ and the relation $8 \leq \text{WIDTH} \leq 12$ becomes the interval equation $\text{WIDTH} = [8, 12]$. If this value is multiplied by $\text{HEIGHT} = [6, 10]$, the result is an $\text{AREA} = [48, 120]$, or $48 \leq \text{AREA} \leq 120$.

Selecting elements from a limited set is another case where Co's support for relations goes beyond simple algebra. Building components are made certain sizes and design relations that specify dimensions must recognize this. It is easy in Co to define tables and catalogs of components and to index them relationally by size, cost, and other properties.

Finally, Co provides the ability to define new relations *that apply to objects, not only to numbers*. For example, although the internal definitions of built-in alignments and abutments relate edge

* The present implementation of the Co language is *busy*, computing all values as soon as possible. A previous implementation was *lazy*, computing values only on demand.

coordinate values, alignments and abutments are *relations between elements*. In turn, abutments, alignments, and offsets can be combined into higher-level spatial relations. At each level the new relations hide their detailed internal definition with an *abstraction barrier*, so we can work at a level appropriate to our needs.

APPLICATIONS

Laying out horizontal curves in road design is a process of placing control points connected by line segments, then inscribing curves in the angles between segments. The curves must be tangent to the line segments and road safety standards dictate additional relationships between radius of curvature, safe travel speed, and lengths of the straight segments. Once an initial road layout is made, the designer edits the layout by moving control points, changing curvatures, or by adding and deleting control points (and thus curves)^[3].

In a space planning problem, a fixed program of functions is placed into a designated site, subject to certain dimensional and adjacency requirements among functions and areas of the site. For example, an entrance area has minimum and maximum dimensions and must be adjacent to the building facade. Once functions are located in the site, the designer edits the layout by adjusting room dimensions. As dimensional adjustments are made, neighboring rooms stretch and squeeze to avoid gaps or overlaps.

In structural design, a beam's cross-sectional dimensions, span, and load are related. If cross-section and span are given, the safe load can be determined; conversely, if load and span are given, the required cross-section can be derived. When the beam is placed in a building, its dimensions are related to other building components. Its span is related to column and bearing wall locations and its depth is related to storey-height. As bearing walls are moved in plan, spans may change, requiring increased beam-depths. In turn this may lower the clear-space between floor and ceiling, or raise the overall building height.

A final application is the coordination of subsystems^[5]. Each system — foundation walls, structural steel, partitions, piping, electricity, HVAC — has its own selection of components and rules for assembly. Additionally, certain position relations must be maintained between elements of different systems. Subsystem placement rules can be coordinated using a grid system.

IMPLICATIONS FOR DESIGN EDUCATION

The introduction of computers into the design curriculum offers exciting opportunities to reflect on architectural knowledge and the process of design. For many, learning to operate available software for drafting, solid-modeling, and rendering in a studio setting is a first step to understanding both the potential and limitations of today's computer tools in the profession. However if this is the full extent of engagement then we have failed to take full advantage of our opportunities.

An effective approach to employing computers in teaching mathematics asks students to write programs to carry out the algorithms they are studying. For example, in instructing the computer to test whether a given number is prime, or to find a common denominator of two fractions, the student is forced to think carefully about how to solve the problem. The computer provides a medium for expressing implicit "how to" knowledge in an explicit and testable form.

A similar approach can be applied in design education. Architects however, unlike mathematicians, are unaccustomed to expressing knowledge in concrete algorithmic terms. Thus, the student becomes a researcher engaged in an effort to capture and convey architectural knowledge to the computer, creating an *epistemology of design*. As a first step in this direction, students learn to write macros and programs to extend built-in CAD features and to automate frequently repeated actions. Constructing shape grammars and knowledge-bases for expert systems should also be classified under this rubric, as students reflect on and express architectural knowledge in a computer code where it can be examined and exercised by others.

A significant obstacle to this approach in design education is the lack of an appropriate language. Traditional computer languages such as BASIC and Pascal were developed for scientific programming and are unsuited to the expression and exploration of design knowledge. The control and data structures are restrictive and the syntax is clumsy. The programming language C and its object-oriented extensions, though popular among programmers, will not serve designers directly, nor even will symbolic languages like Lisp, Prolog, and Smalltalk.

This paper has described the Co relational modeling language and argued the benefits of the relational approach to computer assisted design in architecture. We need interactive computer languages in which we can easily express architectural design concepts. The task of inventing these languages is not easy, for it requires an explicitness that architectural design has hardly known. But if such a thing can be, then architects must participate in the inventing. That should be one goal of integrating computers in design education.

REFERENCES

- [1] Alves, M. and M. Ruano 1987. "Towards Meaningful Computational Descriptions of Architectural Form" in Proc. ARECDAO International Symposium on Computer Aided Design in Architecture and Civil Engineering.
- [2] Borning, A. H. 1981. "Programming Language Aspects of ThingLab" in *ACM Trans. on Programming Languages and Systems* vol 3 no 4 (Oct) pp 353-387.
- [3] Ervin, S. and Gross, M. 1987. "RoadLab - A Constraint-based Laboratory for Road Design." *Artificial Intelligence and Engineering*, vol. 2 no. 4.
- [4] Gross, M., S. Ervin, J. Anderson, A. Fleisher 1988. "Constraints: Knowledge Representation in Design". *Design Studies*, vol. 9 no. 3 (July)
- [5] Gross, M., Habraken, N.J., Ruano, M., Fry, C. 1988. Spatial Coordination Demonstration Program - Report to Shimizu Corporation. (unpublished).
- [6] Leler, W. 1987. *Constraint Language Programming*. Boston: Addison Wesley.
- [7] Lin, V.C., D.C. Gossard and R.A. Light 1981. "Variational Geometry in Computer Aided Design" *Computer Graphics*(15)3:171-177 SIGGRAPH Proceedings
- [8] MacCallum, K. J. and A. Duffy 1987 "An expert system for preliminary numerical design modelling", *Design Studies* Vol. 8 No 4 (October) pp 231- 237.
- [9] Mitchell, W. J., R. Liggett and T. Kvan 1987. "The Art of Computer Graphics Programming" Addison Wesley.
- [10] Nelson G. 1985. "Juno — A Constraint-based Graphics System," *Computer Graphics* (19)3:235-243 SIGGRAPH Proceedings
- [11] Robinson, P. 1989. "The Design Debate," *Computer Graphics World* (12) 4 (April), pp 101-106
- [12] Serrano, D. and Gossard, D. 1987. "Constraint Management in Conceptual Design" in AI and Engineering: Planning and Design ed. R.A. Adey and D. Sriram. Computational Mechanics Press.
- [13] Steele, G. J. and Sussman, G. 1979. "CONSTRAINTS - A Language for Expressing Almost-Hierarchical Descriptions", *Artificial Intelligence* 14:1-39
- [14] Sutherland, I. 1963. "Sketchpad - a Graphical Man-Machine Interface", M.I.T. Ph.D. Dissertation
- [15] Weinzapfel, G. and S. Handel 1975. "IMAGE: Computer Assistant for Architectural Design" in *Spatial Synthesis in Computer-Aided Building Design* ed. C. Eastman. pp 61-68 New York: Wiley.