Elements that Follow Your Rules: Constraint Based CAD Layout

Mark D. Gross
Sundance Laboratory for Computing in Design and Planning
College of Architecture and Planning
University of Colorado, Denver
Boulder CO 80309-0314
mdg@cs.colorado.edu

ABSTRACT

The paper reports on CKB (Construction Kit Builder)—a prototype CAD program that designers can program with positioning and assembly rules for layout of building elements. The program's premise is that designing can be understood as a process of making and following rules for the selection, position, and dimension of built and space elements. CKB operates at two distinct levels of design: the technical system designer, who makes the rules, and the end designer, who lays out the material and space elements to make a design. CKB supports two kinds of rules with constraint based programming techniques: grid and zone based position rules, and assembly rules that position elements with respect to one another. The paper discusses the rationale for CKB and describes its implementation.

DESIGN AS MAKING AND WORKING WITHIN A SYSTEM OF RULES

Rules and Systematicity in Design

Good designers follow a principled approach to arranging physical form. Decisions about the organization and layout of spaces and materials are not made arbitrarily, rather, they are made systematically. There is a consistency, a regularity, to the dimensions of space and material elements, to the joining conditions of materials, and in general to the way elements of the design are placed in relation to one another.

The words 'systematic,' 'consistent,' and especially 'rule' have negative connotations in today's culture of architecture. The use of systematic design methods to produce low quality, cookie-cutter mass housing and industrialized buildings have lent a bad taste to the notion of designing within rules, and even designers who work in highly systematic ways go out of their way to deny it. It is therefore important to point out at the outset that systematicity and consistency need not imply uniformity. On the contrary, good design exhibits a great deal of variation within a system, and certainly rules are occasionally stretched or broken. Often, it is only by observing the range of variation in a design that one can identify a system.

The idea that designers make and follow rules that govern form is hardly new (Habraken, Boekholt et al. 1976; Archea 1987) . The same assumptions of systematicity underlie the use of shape grammars to describe families of physical form. The grammars that describe the works of a single designer or characterize a style assume a consistency that can be captured with rules (Stiny 1980; Köning and Eizenberg 1981; Hersey and Freedman 1992) . In shape grammars the rules are production rules; here the rules are constraints on the positions of material and space elements. The premise of this paper is that designers, even the most flamboyant individualists, operate within a system of rules that govern their arrangement of physical form.

CAD Programs are Construction Kits

Most (if not all) CAD systems in use today adopt a 'construction kit' approach. The user assembles a design (a drawing or model) by selecting elements from a palette and placing them in a work area. The elements may be geometric (lines, arcs, planes, solids, and surfaces); or they may carry additional information that identifies them as specific building components or functional spaces. The user can extend the built in palette of components with catalogs available from third-party vendors. CAD construction kits thus resemble building toys such as Lego or TinkerToy. The designer selects elements from a palette and assembles them to make a design. However,

building toys provide one feature that CAD construction kits do not: Their elements are designed to be assembled only in certain ways, and the building toy physically enforces these constraints.

Constraint Based CAD

Constraint based CAD systems have been built for architectural design. Constraints enable a CAD system to maintain desired spatial (and other) relations that the user asserts. The advantage of constraint based CAD over construction kits without constraints is this: The designer can instruct the computer not only about the positions of individual design elements, but about the relationships that underlie positioning decisions. Gross's CoDraw program, provided a menu of spatial relations that users could apply to the elements of a drawing (Gross 1992) . Tobin's C•Mod system added constraints to a 3D modeling environment (Tobin 1991) . Kolarevic's ReDraw, provided constraints in the form of guide-lines within which design objects were constructed (Kolarevic 1994) .

Since Sutherland's early Sketchpad program (Sutherland 1963) and subsequent work by Borning (Borning 1981, Steele (Steele and Sussman 1979) , and others, constraint based programming techniques have become a well-researched area (Leler 1987) . In addition to the architectural applications mentioned above, constraint based CAD has been explored in mechanical engineering design (i.e. kinematics) (Kramer 1992) , logic design (Fujita, Iwamoto et al. 1994) , and other engineering design domains.

Constraint techniques have been adopted in varying degrees by commercial CAD software products. A limited form of constraint based layout has become a part of many CAD programs in the form of "grid snap" and "object snap." (Bier and Stone call their "SnapDraggin'" system (Bier and Stone 1986) a form of "poor man's constraints"). In grid snapping as users drag elements they snap to position on a background grid. In object snap, elements align their edges or centers with other already placed elements.

The CKB program described here goes further than common implementations of grid-snap and object snap in several ways. First, it provides a greater variety of grids and grid-relative rules, which the designer can program. Second, the designer can build specific positioning constraints into the elements, so they know how they should (or shouldn't) relate to other specific classes of elements. (In many implementations of object snap all elements relate in the same way to all others.) Third, the program continues to maintain constraints after the elements are placed into the design.

The remainder of the paper describes the rationale behind CKB as well as its implementation. The next section outlines two methods for describing rules in architectural design: the use of grids to make global positioning rules, and the use of offsets and alignments to describe local positioning conditions. The following section describes the constraint based approach used in CKB's interface: objects that "know" the rules for their placement. Then, the paper provides a brief demonstration of the CKB program, and followed by some implementation details. The paper concludes with a discussion of directions for further work.

TWO KINDS OF POSITIONING RULES

Although rules about the spatial organization of physical form can be expressed in various ways, two kinds of positioning rules are particularly pertinent to architectural design: rules about layout of elements in relation to grids and zones, and rules about assembly of pairs of elements.

Grids and Zones

Grids and zones are used in design as global positioning devices, to establish the locations of elements that occur throughout the spatial extent of the design. For example, a grid (or a system of related grids) is often used to establish the position of columns and beams that make up the structural system of a building. Although in a simple building a single structural grid may suffice, more complex designs employ often several grids to control the positions of different elements in different parts of the design. For example, the designer may take one grid from the building's context to organize the facade and use another to organize the building's internal functions; or use one grid for the public, collective areas of the building and another for the private or individual areas. Although typically

grids in architectural design are rectangular, two grids need not meet orthogonally, but may join at an angle. In the design of a building the architect may employ several different grids, nested hierarchically to organize the layout of elements at different levels. For example, a larger structural grid may organize columns and beams; a grid with smaller dimensions may organize the positioning of infil walls and screens and the functional spaces they define.

Although most CAD programs that employ grid snap support only the relation "centered on the grid", a wider range of grid relative spatial relations are found in design. For example, elements can be related to the horizontal or vertical lines of a grid, restricted to the space (or 'zone') between grid lines, or positioned by offsets from their edge or center to grid lines. The grid need not be limited to a simple square grid, or even a rectangular grid, but may have several spacing dimensions both horizontally or vertically, as in a tartan grid. The relation between a class of elements and a grid need not be fixed, but may allow for variability or tolerance.

Relative Positioning of Pairs of Elements

In contrast to the global positioning scheme of grids and zones, designers also work with local spatial relations between pairs of elements. These spatial relations can be described by combinations of offsets, alignments and adjacencies, and centerings between elements. Relative positioning rules are used to control the assembly of building components, for example, placement of a window in a rough opening; or joining details of a roof and a wall. But relative positioning rules can also describe desired spatial relations between functional spaces, for example between rooms in a dwelling. No doubt other schemes can be devised to describe the placement of material and space elements in a design. However, these two methods can describe many, if not most, of the ways of arranging the elements of buildings.

CKB—A PROGRAM FOR RULE-GOVERNED LAYOUT

CKB (Construction Kit Builder) is a prototype CAD program that supports the rule-governed layout of design elements. CKB enables the designer to specify the elements in the construction kit palette, to specify the grids employed during layout, and to specify rules for placing elements relative to grids and relative to one another. The specification of elements, grids, and rules is performed interactively and can be accomplished at any time during design.

Objects That Follow Rules

CKB is a CAD construction kit in which the elements 'know' how to follow rules that the designer sets for them. In a standard construction kit CAD program, the designer places each element into the design arranging elements to achieve desired spatial relationships. In CKB, the designer programs each element with rules that govern its desired placement in the design. Then, as the designer places elements into the design, the elements take their places according to the pre-programmed rules.

The rules in CKB—grid-relative and assembly rules—are programmed as constraints. CKB goes beyond other constraint based programs because, in standard constraint based CAD, the designer must apply constraints to individual elements during layout; the elements then maintain the specified spatial relationships. In CKB the designer programs the element classes with default spatial relationships.

Defining Elements, Grids, and Rules

CKB provides a simple window based interface for defining elements, grids, and placement rules. Figure 1 shows the interface for defining an element. The designer can use a simple draw program to enter an element, or import a picture made in another application. A catalog of 'systems' organize the elements, so that for example, windows and doors belong to one palette, and columns and beams to another. The designer can also specify the 'hot spot' of an element, a point on the element used for applying grid relative rules.

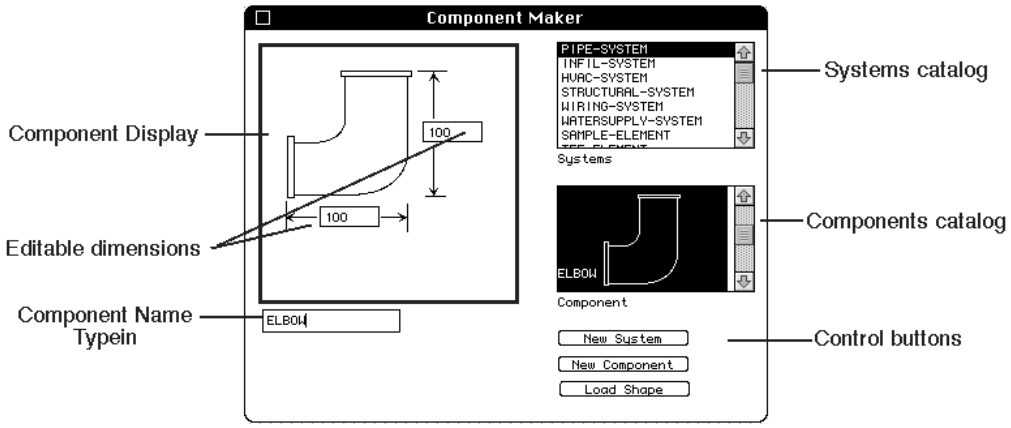

Figure 1. Element Maker window

Figure 2 shows the window interface for defining new grids or for modifying the characteristics of existing ones. Each grid is named and appears in a catalog. The Grid Maker window provides type-in boxes for specifying the horizontal and vertical spacing sequences and offsets.

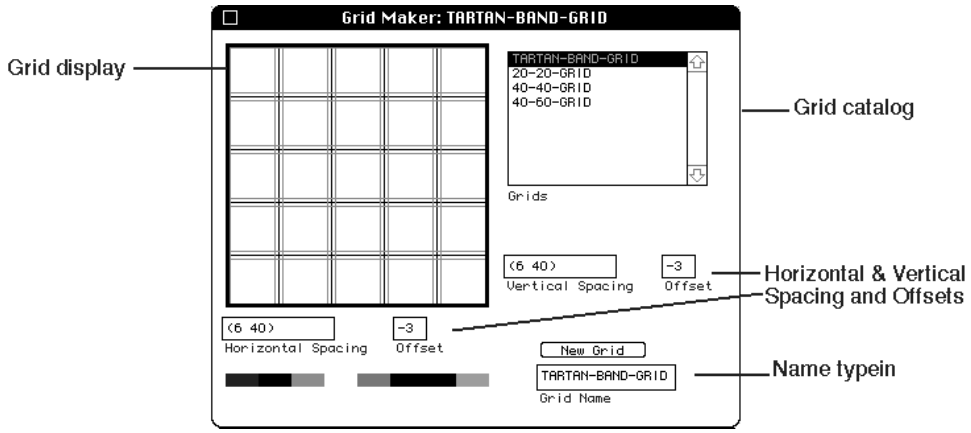

Figure 2. Grid Maker window

Figure 3 shows the window interface for defining grid-relative placement rules for a class of elements. (Here the designer is defining grid placement rules for 'Wiring Junction' elements.) The designer selects a grid and an element from catalogs.

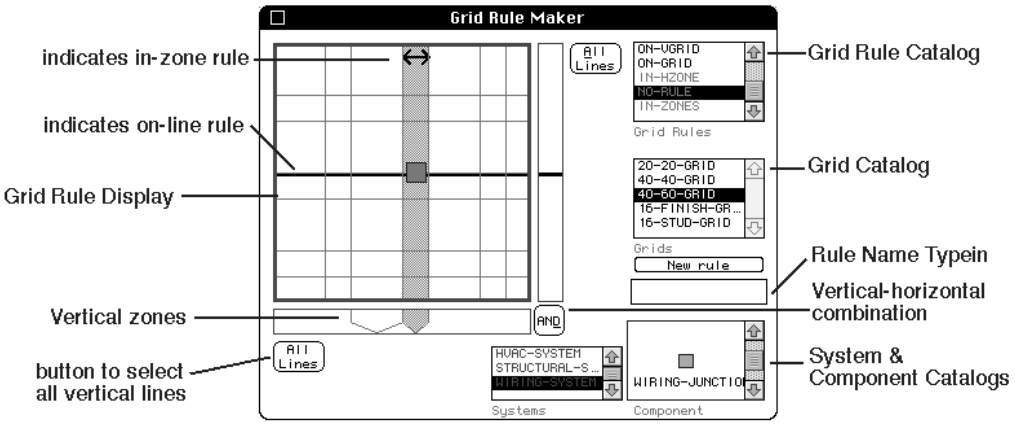

Figure 3. Grid-rule Maker Window

The designer can then specify that the element is to fall on certain grid lines (by selecting the lines), or in the zones between grid lines (by selecting the zones). The 'and/or' button enables the designer to specify whether the horizontal and vertical components of a grid rule are to be applied in 'and-combination' (both must be adhered to) or in 'or-combination' (only one must be adhered to). That is, a rule may specify that an element must be placed on horizontal AND vertical grid lines, or it may specify that an element must be placed on horizontal grid lines OR vertical ones.

Figure 4 shows the window interface for defining assembly rules between pairs of elements. The designer selects two elements from catalogs (system and component scrollers at bottom right).

The designer can assign a previously defined assembly rule for the two elements, by selecting it from the catalog of assembly rules, or define a new assembly rule for the pair. To define a new assembly rule, the designer first arranges the elements as they are to be assembled. CKB provides offset arrows or alignment lines that reflect its best guess as to the relationships that the designer has in mind. The designer can then modify the assembly rule by selecting and operating on the offsets and alignments. Finally, the designer must attach the new assembly rule to the elements, using the 'rule->components' button.

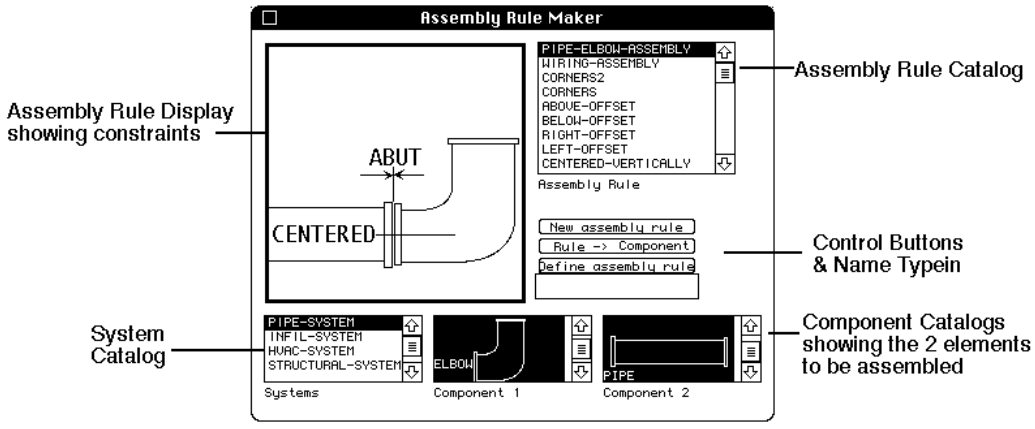

Figure 4. Assembly rule maker window

Laying Out Configurations and Arrangements

Rules for placement and assembly of elements come into play when the designer begins to lay out elements in a design. As the designer selects elements from a hierarchically structured palette to place them in the design, CKB maintains both grid placement and assembly rules. As the designer drags elements over the layout window they snap into place according to their grid rules. As the designer drags an element near to another element CKB checks whether the two elements can be assembled. If the designer has programmed an assembly rule for the two

elements, CKB snaps the new element into place. If the two elements can be assembled in more than one way, then CKB provides a menu of assembly options. Of course, in all cases, the designer can override the previously defined rules and place the element in an arbitrary way.

IMPLEMENTATION

CKB is implemented in Macintosh Common Lisp and uses CLOS (Common Lisp Object System) to define classes for elements, grids, grid-position rules, and assembly rules. These data structures are connected with one another by pointers, as shown in figure 5. Each element may have one or more grid-relative rules, and one or more assembly rules. Slots in an instance of a grid-relative rule associate an element (or class of elements) with a grid (or a class of grids). Other slots in the grid-relative rule identify the specific spatial relation between the element and the grid, that is, whether the element is to be located on horizontal or vertical grid lines or in the bands between the lines. Similarly, slots in an assembly rule identify the classes of elements that may be related, the specific elements that are related (in an assembly rule that has been instantiated) and the specific spatial relation among the elements that is to be maintained.

The position rules are composed from a small set of primitives that compute coordinates for the elements they constrain. For example, a grid relative rule comprises two constraints, one for the horizontal and one for the vertical position of the element. Each component consists of one of two relations, in-zone or on-grid, with specific arguments that identify the particular grid zone or grid line. Likewise, assembly rules identify the two elements that are to be constrained, local positioning information for each (i.e., specifying the mating surface for assembly).

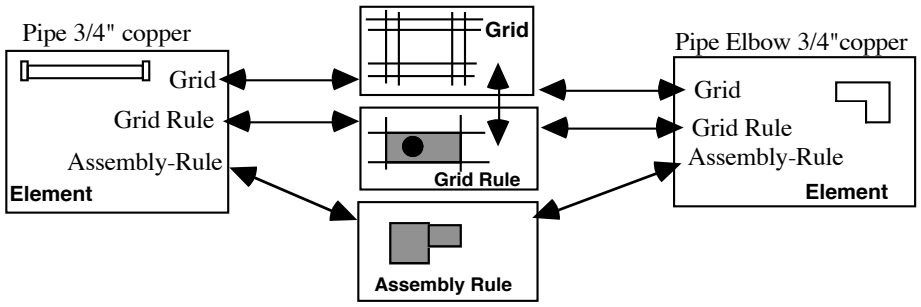

Figure 5. Connections among instances of CKB's basic classes.

Two Levels of Operation

CKB operates at two conceptual levels. At the 'system design' level, the designer specifies elements and rules for their placement in the design. At the 'system layout' or 'end design' level, the designer assembles elements into a particular layout, in accordance with previously specified placement rules.

Constraint Consistency and Maintenance

A straightforward constraint management module at the heart of CKB comes into play whenever the designer places or moves an element. First, the position of the new or moved element is determined as a function of the element's initial (raw mouse) position. The initial position is adjusted by any grid-relative rules that constrain the element, as well as any assembly rules that constrain the element with respect to other, more dominant elements. Each element class carries a dominance value that determines which element will adjust its position when a constraint comes into play. For example, a foundation wall has a higher dominance value than a pipe; the pipe cannot force the wall to move, but the wall can push the pipe around. Two elements of the same class (e.g., a pipe and a fitting) have the same dominance value, so the element that the designer is controlling will force the other to move.

Then, the positions of any dependent elements (with which the element has been assembled) are recomputed. In turn, any further dependent elements are polled and their positions updated as necessary. A simple

propagation algorithm is used to maintain the constraints, with inequality checking to handle the 'in-zone' placement rules.

For example, consider what happens when the designer adds a new element to a design. As the designer drags the element, its raw (unconstrained) position is first modified by any grid relative constraints. If the element is associated with a grid relative rule that constrains it to lie on grid crossings, its position is modified accordingly. If it is constrained to lie on either horizontal or vertical grid lines, then the nearest (least difference) constrained position is chosen. If it is constrained to lie in a zone between grid lines, then its position is only modified if necessary.

After applying grid relative position rules to the element's raw position, CKB checks to see if the new element can be assembled with any nearby elements. If not, the element is simply placed in its grid constrained position when the designer releases the mouse button. If there is an element near to the current position of the new element with which it can be assembled, the assembly rule is applied next, modifying the position of the new element accordingly. Each element maintains a list of 'applicable assembly rules' that describe the ways it can mate with other elements. As the designer drags an element around the work area, CKB continually checks the applicable rule lists for the dragged element and its neighbors.

CKB does not attempt to resolve conflicts between grid-relative rules and assembly rules. This can lead to odd behavior if the designer has not carefully designed the rules. For example, a grid-relative rule might keep an element snapping to grid centers, until the designer drags it near a candidate for assembly; then the element jumps off the grid to mate with an already-placed element.

It is well known that simple propagation of constraint has an important limitation: it cannot handle cycles in the graph of constraints. Such cycles amount to a system of simultaneous equations, and propagation, a local technique, cannot detect or solve such systems. CKB, which uses only local propagation, can indeed be trapped into trying to solve systems of simultaneous constraints by propagation. A more sophisticated constraint management and solving system (like that employed in my earlier CoDraw program) could of course identify and solve systems of simultaneous constraints. However, in the sorts of layout problem that CKB is designed to support, systems of simultaneous constraints do not seem to arise very frequently. One reason for this is the dominance values attached to each class of element, which order the constraint relations between individual element instances in a layout. Another reason is that in this kind of layout most elements have fixed sizes; constraint relations cannot change their dimensions, only their positions. The constraints thus tend to be relatively easy to manage. The program is not asked to solve the sorts of complicated systems of constraints that one finds, for example, in a space assignment problem. It is only asked to supervise the proper placement of elements that are intended to go together in the first place. Thus, the constraints are there to help the designer do the layout correctly, but (in this application) constraints are not used to solve a complicated design problem. For that, more sophisticated solving techniques would indeed be required.

DISCUSSION & FURTHER WORK

CKB is a working prototype but it lacks several features that would be needed for a fully functional CAD program. First, it is essentially two-dimensional, although the elements can carry 3D positioning information. Its displays are all two dimensional; a fast 3D engine (such as QuickDraw 3D) will be needed to extend CKB in this direction. Second, although CKB supports assembly rules, it does not support a representation for grouped objects, or subassemblies. This would clearly be needed for a workable CAD system. Related to this latter limitation, CKB does not support the detail specification of abstract higher-level configurations. Thus it is only useful when the designer is working with specific components, not with abstractions that are to be later instantiated in further detail. For real designing, this must be considered a serious limitation, and one that future versions of the program should overcome.

The point of CKB is to enable designers to program the elements of their design with systematic spatial relations. Some will object to this on principle, arguing that design should be unfettered by predefined rules, even if the rules are made by the designer. Others may cautiously accept the idea of a rule governed layout program, but may point out that the rules that CKB supports are too simple to make architecture, or even good buildings. This suggests perhaps the most interesting direction for future work on CKB, articulating a language for rules in architecture that computers can follow.

ACKNOWLEDGMENTS

Support from the National Science Foundation under grant DMII 93-13186 has been instrumental in developing CKB. Ellen Yi-Luen Do and Raymond J. McCall provided valuable suggestions in discussion.

REFERENCES

Archea, J. (1987). Puzzle Making: What Architects Do When No One is Looking. Computability of Design. New York, Wiley Interscience. Y. Kalay, ed.

Bier, E. A. and M. C. Stone (1986). "Snap-Dragging." Computer Graphics **20**(4): 233-240.

Borning, A. (1981). "Programming Language Aspects of ThingLab." ACM Transactions on Programming Languages and Systems **3**(4): 353-387.

Fujita, S., M. Iwamoto, et al. (1994). Constraint Based Technology Mapping in Logic Design. Artificial Intelligence in Design '94. Amsterdam, Kluwer. J. S. Gero and F. Sudweeks, ed. 347-362.

Gross, M. D. (1992). Graphical Constraints in CoDraw. IEEE Workshop on Visual Languages. Seattle, IEEE Press. S. Tanimoto, ed. 81-87.

Habraken, N. J., J. T. Boekholt, et al. (1976). Variations - The Systematic Design of Supports. Cambridge, MA, MIT Press.

Hersey, G. and R. Freedman (1992). Possible Palladian Villas. Cambridge MA, MIT Press.

Kolarevic, B. (1994). Lines, Relations, Drawings and Design. ACADIA-94. St. Louis, MO, ACADIA. A. Harfmann and M. Fraser, ed. 51-62.

Köning, H. and J. Eizenberg (1981). "The Language of the Prairie." Environment and Planning B **8**: 295ff.

Kramer, G. (1992). Solving Geometric Constraint Systems. Cambridge MA, MIT Press.

Leler, W. (1987). Constraint Programming Languages. Boston, Addison Wesley.

Steele, G. L. and G. J. Sussman (1979). "CONSTRAINTS - A Language for Expressing Almost-Hierarchical Descriptions." Artificial Intelligence (14): 1-39.

Stiny, G. (1980). "Introduction to Shape and Shape Grammars." Environment and Planning B **7**: 343-351.

Sutherland, I. (1963). Sketchpad - a Graphical Man-Machine Interface. Ph.D. dissertation, M.I.T.

Tobin, K. (1991). Constraint Based Three Dimensional Modeling as a Design Tool. ACADIA-91. Los Angeles, CA, ACADIA. G. Goldman and M. S. Zdpeski, ed. 193-210.

"Elements that Follow Your Rules: Constraint Based CAD layout", Gross, M.D., *Proceedings of Association for Computer Aided Design in Architecture (ACADIA) '96*, Tuscon, AZ, pp. 115-122.