Why can't CAD be more like Lego?
CKB, a program for building construction kits

Mark D Gross
College of Architecture and Planning
University of Colorado
Boulder, CO 80309-0314

Keywords: CAD application tools, kit-of-parts design, CAD, constraint based design, rule based design, product modeling, component modeling, Lego

Abstract:
The paper describes CKB (Construction Kit Builder), a computational design environment based on defining, then working within, a system of components and rules for their placement. In construction details, the components and placement rules are standard; in less routine design tasks they are not. CKB uses a constraint-based, object oriented, system architecture to provide two levels of design support. At the higher level, designers of technical systems use CKB to specify components and rules for their positioning. At the lower level, building designers use CKB to lay out components of these systems.

1. Introduction: Design as making and following rules

Designing can be seen as a process of making or adopting rules, and then working within these rules to achieve certain programmatic and functional objectives. In architectural design many rules reflect standards of practice, expressed in building codes or effected by the availability of standard construction systems. For example, designers work with standard components of what I shall call (after Habraken) 'technical systems': structural steel beams and columns, infil walls and windows, cabinets and counters, and pipes, wires, and air conditioning ducts. Knowledge of how these technical systems fit together is an important part of a building designer's expertise, yet—perhaps because it seems too mundane—it is often overlooked in discussions of what computers should know about design.

I do not mean that the design of buildings ought to be entirely reduced to following a standard set of rules. Even in routine design, architects and engineers decide how to lay out components, for example, whether to place bearing columns on a 10 or 12 foot grid; what screening components to use and how to connect them with bearing walls. While adhering strictly to the rules of a low level technical system that constrains construction details, the architect is free of them (at a higher level of decision-making) to make larger-scale configurations of built and open space. Even in highly innovative designs standard components are deployed in a regular, if non-standard fashion.

Most computer based construction kits (CAD editors) for buildings operate only at the level of geometry. Although they may provide a library of standard building components for architects and engineers to use, these programs lack knowledge about how components are laid out and assembled together in a design. The designer supplies this knowledge while using the construction kit to make a building model, ensuring that components are assembled into a design in ways allowed by the technical systems.

How different, then, is CAD from the construction toys we played with as children! Lego, Tinkertoy, Lincoln Logs, Kenner Girder & Panel Construction, Meccano—all these toys provided a construction kit of standard components that would only assemble in standard ways. The makers of these technical systems designed the components and their joining conditions to provide a wide range of variability for the end-designer (the child). Thus knowledge about assembly and layout is implicitly built into the system components. In this structured freedom to assemble components, construction kit toys actually resemble real building more closely than CAD.

This paper explores the idea of making CAD editors more like Lego or Tinkertoy, by building in to the editors knowledge about the layout and assembly of building components. It reports on CKB (construction kit builder), a program for designing and working with technical systems of components that 'know' how they are to be assembled and laid out. Although it employs constraints, a representation for knowledge commonly used in artificial intelligence programs, CKB is not an 'expert system' in the traditional sense of a program that makes decisions for a human user; rather it is a 'system for experts' that supports knowledgeable design decision making.

The key point about CKB is that it supports two levels of designing: (1) the design of technical systems and (2) the use of technical systems to design built configurations. In the following sections, the term 'system designer' refers to the designer of the components and rules of the technical systems; the term 'end designer' refers to the designer who uses these systems to make architectural designs. Keep in mind, however, that the system designer and the end designer may well be the same person, playing different design roles.

The remainder of this introduction discusses two kinds of decisions that architects must ultimately make—decisions about selection and position of components—and then briefly discusses the use of constraints to implement position rules. Section 2 presents two ways to express rules about the positions of components: positioning components relative to grids, and positioning components relative to one another. Section 3 introduces CKB, a program for designing and working with rule-governed systems of components. Section 4 is concerned with the implementation of CKB, and provides an overview of its data structures and internal operations, including the propagation of constraint and the handling of conflicts. Section 5 discusses related work on building models, component based, and constraint based CAD. Section 6 points out the application of CKB in coordinating group work and mentions the conceptual limits of the Lego approach to the design of buildings. Finally, CKB is not a CAD product, but a working prototype that demonstrates the principles described in this paper. Therefore section 6 concludes by pointing out some technical limits of the current CKB implementation, which suggest directions for further work.

1.1 Decisions about selection and position of components

Ultimately, architectural designing involves deciding about the selection of built and spatial components and about their dimensions and positions. This, after all, is what the architect records in construction drawings and specifications and hands to the contractor who will make the building. Whatever process the designer may use to move from initial conversations with a client and the analysis of site and program, if a building is to be built then eventually the architect must specify the selection, dimensions, and positions of its components.

When the designer chooses to work with standard components, decisions about dimensions are largely folded into decisions about selection. That is, one can obtain copper pipe from suppliers of building materials only in certain manufactured diameters and in standard lengths. Wood is manufactured (in North America) on a nominal 2" module; and windows, doors, and other screening components are manufactured in sizes to work within a standard construction dimensioning scheme that locates studs and joists on 16" centers. Thus, while the architect has more freedom in choosing the dimensions of higher level elements (e.g., rooms, or lengths of wall), building components come in predetermined sizes. A selection decision therefore often implies a dimension decision.

Decisions about the placement of building components, of course, follow programmatic and functional decisions about overall plan layout, the dimensions and organization of spaces, as well as functional concerns such as heat, light, and sound. However, such 'higher-level' decisions usually do not fix the positions of building components. Rather they constrain component positions roughly, leaving precise positions to be determined by the rules of the technical system. For example, in deciding about a living room, the exact positions of studs in the wall are of little consequence. We consider later (in Section 6.2) whether higher-level positioning decisions about spatial elements might also be regulated in the same ways as low level technical components.

1.2 Position rules as constraints

I have worked previously on constraint based drawing programs, in particular, a program called CoDraw

(Gross 1990; Gross 1991; Gross

1992) . Like its antecedent SketchPad

(Sutherland

1963) , CoDraw enables a designer to apply geometric (and other) constraints to drawing elements; then as the designer edits the drawing, the computer enforces the constraints. For example, once the designer establishes a proportion relation among two elements, an adjacency, or a containment relation, the computer tries to keep the elements arranged to maintain the relations. In a conventional drawing program, by contrast, the designer must keep track of all the desired spatial relationships. Constraint based drawing programs go beyond conventional CAD editors because they enable the designer to instruct the computer about the desired arrangement of elements.

CKB takes this approach a step further than CoDraw. CKB uses constraints in the same way as in CoDraw to establish and maintain the positions of design components. But in CKB, a designer (the system designer) programs each component with a set of possible constraints that describe how it can be placed on the site and in conjunction with other components. For example, the system designer can program a beam with a set of alternative spatial relation constraints that describe how it may be placed in the site and assembled with columns, walls, and other components. When the end designer (perhaps the same person as the system designer who set the constraints), places a beam into the design, it may take only certain positions relative to other components.

2. Rules about positioning components

One might think of many ways to express rules governing the positions of components, but two methods are common in architectural design: the placement of components relative to a grid or a system of grids, and the placement of components relative to one another. For example, it is common practice to establish a grid for placing structural components—bearing walls, beams, and columns. On the other hand, the joining conditions of walls and windows are usually expressed in a detail that describes the relative positions of the two components using offsets and alignments.

Of course, these two methods are not mutually exclusive: a grid scheme for positioning columns can also be described in terms of relative positions among the columns. Alternately, if one considers a grid as a kind of component, then grid positioning is simply another kind of relative positioning. For example, one can think of Lego pieces either as governed by local rules that specify how individual pieces fit together, or as governed by a global grid into which all pieces must fit. (This is not true of all construction kit toys). Nevertheless, it is helpful to think of grids and relative position rules as distinct ways to express rules.

2.1 grid-relative rules
Architects use grids to control the positions of components distributed widely throughout the site

(Gross

1991) ; that is, grids are global positioning rules. The use of structural grids to govern the positions of columns, beams, and bearing walls has already been noted. Grids are also often employed to make the dimensions of spaces modular. For example, a 4-foot grid might be used for residential space planning to design spaces that are 4' (stairs and closets), 8' (small bedrooms, baths, and kitchens), 12' (bedrooms, small living rooms) and 16' (larger living rooms). Often a coordinated system of grids working together at different scales, or levels of hierarchy, are used. For example, a 20-foot grid might be used to plan the layout of a multi-dwelling building, within which a 4-foot grid (as above) structures room sizes, and a 16-inch construction grid governs the positions of boards and panels that make up the technical system at the lowest level.

Different grids may be used in different parts of a design; for example, one grid governing the larger dimensions of the public zone of a building and another governing the smaller dimensions of private offices and work areas. Grids may be superimposed, or they may join at non-orthogonal angles, resulting in special conditions where they meet (figure 1).

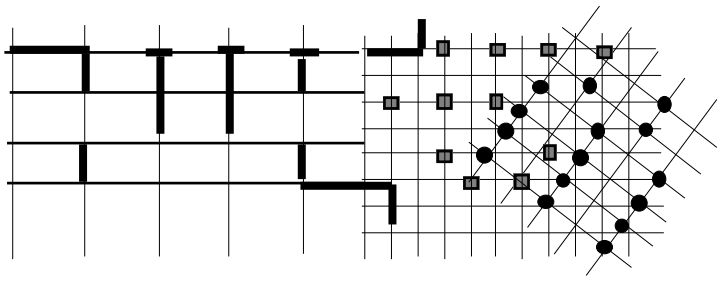

Figure 1. Grids in different parts of a building may meet in different ways.

Various grid-relative rules are common; for example, in figure 2, columns (solid squares) center on grid crossings; wall panels (long rectangles) center on either horizontal or vertical lines remain but free to move along their linear direction; and lighting fixtures (open circles) center in grid squares.

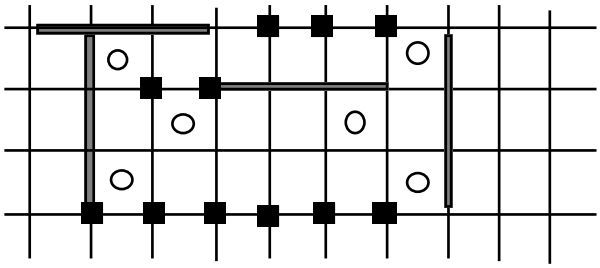

Figure 2. Different components with different position rules relative to a square grid.

Although the simple square grid is most common in CAD, other types of grids are frequently used in architectural design:

• The tartan grid (figure 3) has more than one spacing module. By using different combinations of the spacing modules, (a, b, a+b, 2a+b, 2b+a, etc.), a designer can specify a variety of dimension and positioning options relative to the grid. Tartan grids are the basis for modular coordination schemes, such

as the SAR design method (Habraken, Boekholt et al.

1976; Kroll 1987) .

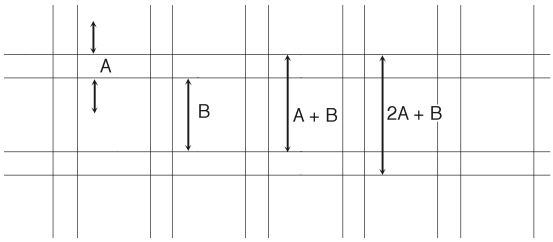

Figure 3. A tartan grid offers a range of dimensional guidelines.

• In a rectangular grid spacing along one dimension differs from the spacing in the other dimension. Although most CAD systems merely support square grids, rectangular grids are often used in architectural design, reflecting real differences in the two orthogonal directions. For example, in a structural grid, beams typically span in one direction and not the other (figure 4).

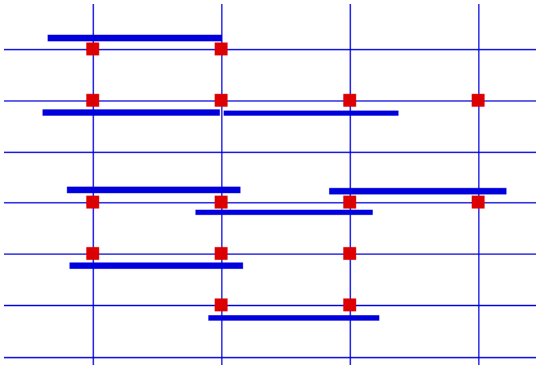

Figure 4. A rectangular grid is usually the basis for post and beam construction.

• Nonrectangular grids such as hexagonal grids, and radial grids are also used, but far less frequently than rectangular ones. Occasionally for amphitheatres and round buildings designers use a radial grid. However, nonrectangular grids are the exception rather than normal building practice.

2.2 assembly rules

Whereas grid position rules are global devices for controlling the positions of components distributed throughout the design, assembly rules are local devices that govern the relative positions of components to one another. Perhaps the most obvious examples of assembly rules are *intra-system* assembly rules—the spatial conditions for putting together components of a single technical system. For example, assembly rules govern the mating of pipes and fittings, electrical conduit and junction boxes, beams and columns, walls and windows. In addition, *inter-system* assembly rules govern connection conditions between components of different systems, for example, the location of conduit with respect to a stud wall or the way infil walls join the components of the structural bearing system.

Components often have slots and notches or other physical cues built in to ensure correct assembly. For example, it is almost impossible to incorrectly assemble a copper pipe with a copper elbow fitting, provided the two have the same diameter. The pipe slides into the fitting a certain distance and then stops. On the other hand, other systems, such as the 2x4 wood framing system, rely on the builder to assemble their pieces correctly according to the system's rules. In any case, an assembly rule describes a set of spatial relations between the components that constrain their positions in three dimensions. This is usually done by describing alignments, offsets, and centerings relative to the components' edges, centerlines, or specific reference points..

3. CKB - a program for building construction kits

CKB is a program for making construction kits of components that know the ways they may be placed and assembled. CKB is a working prototype, implemented in Common Lisp for the Macintosh. It is an object oriented, constraint based program, built using the Common Lisp Object System (CLOS) and Macintosh Quickdraw graphics.

CKB operates at two levels, that of system designer and that of end-designer. System designers (like the designers of Lego) define the components of technical systems and their position rules. End-designers (like the child who builds with Lego) deploy the components of technical systems to make designs, but they do not program the position rules that determine their behavior. As its name suggests, the focus of CKB (Construction Kit Builder) is supporting the task of the technical system designer.

A technical system (such as a system of columns and beams or infil walls) is defined by four interacting parts: components, grids, grid positioning rules, and assembly rules. CKB provides the system designer with a simple interface for defining each of these parts of the technical system.

3.1 Components

Figure 5 shows CKB's Component Maker window. The display pane shows the current component, and two scrollers at the right display the current technical system and the current component. Two typeins with witness lines (arrows) show the size of the component. The system designer can use a simple drawing editor to enter a shape for the component, or load a shape drawn in another program. If the system designer does not define a shape for the component, then a rectangle is displayed. The system designer can also locate the component's reference point, with respect to which grid-relative rules operate.

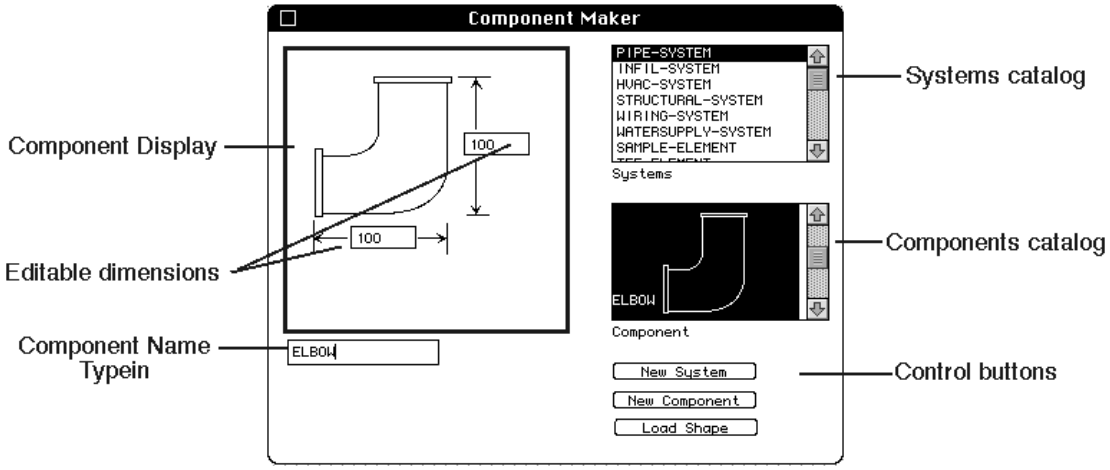

Figure 5. Component Maker Window: an Elbow is a component in the Pipe System

Catalog components come with predefined dimensions. The current implementation of CKB does not support components with parametric or variable dimensions, for example, a variable length pipe or an elbow whose dimensions depend on its diameter. (This is admittedly an important feature missing from the current version of CKB.) However, the end designer can modify the dimensions of a component once it is placed in the design.

3.2 Grids

Figure 2 shows CKB's Grid Maker window. The display pane shows the current grid selected in the catalog scroller at the right. CKB shows the current grid's horizontal and vertical spacing sequences in editable text displays as well as the grid's horizontal and vertical offsets. The system designer can use the

spacing sequences and offsets to design a family of associated grids. For example, in figure 6, the designer has made a main square grid and an auxiliary tartan grid to define a zone around each of the main grid lines.

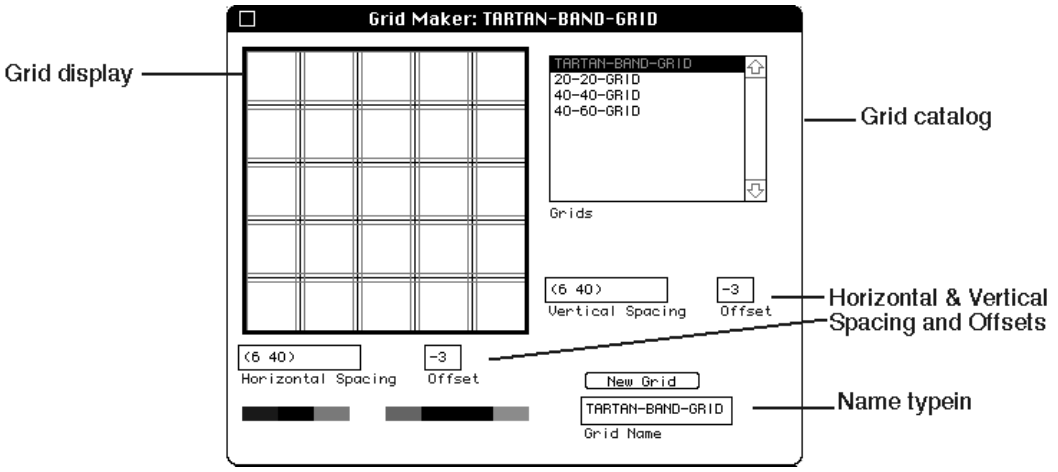

Figure 6. Grid Maker Window: a band grid defines a narrow region around the lines of a larger, square grid.

3.3 Grid Rules

Figure 7 shows CKB's Grid Rule Maker, where the system designer defines a grid-relative position rule and associates it with a component. The display pane shows the grid rule with respect to a component (in this case a wiring junction), selected from the system and component catalogs at the bottom of the window. At right is a catalog of previously defined grid positioning rules, a catalog of grids, and a button to define a new grid rule from the current display pane. The system designer can drag the component and observe the current grid rule's behavior. For example, the rule in figure 7 requires wiring junction components to center horizontally on grid lines while remaining vertically within the grid's narrow band.

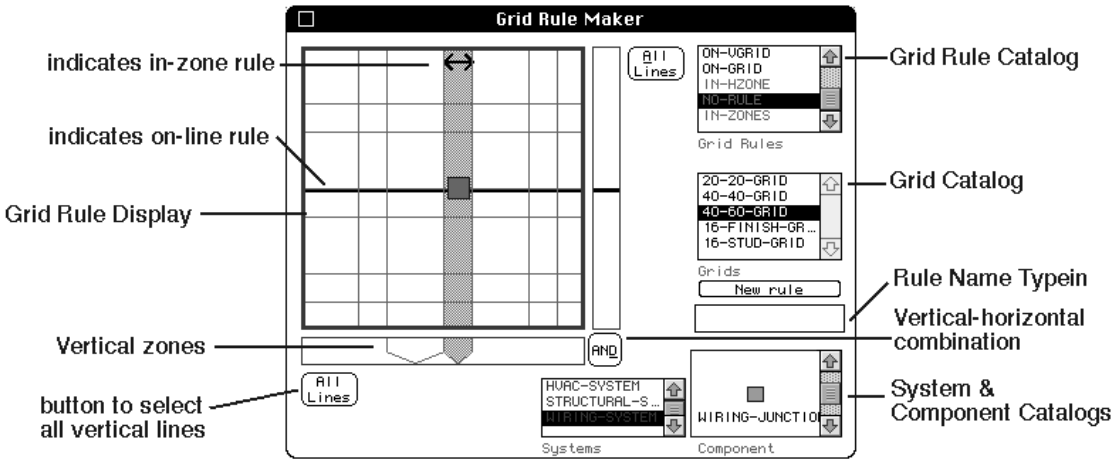

Figure 7. Grid Rule Maker Window - Wiring junctions must be positioned in the narrow vertical band of the grid, and centered on horizontal grid lines.

CKB supports two kinds of grid position rules: positioning relative to grid lines and positioning relative to the bands between them. These may be combined in various ways. The system designer selects a grid band or grid line to be part of a rule by pointing at it; a rule may specify several grid lines or bands. The 'and/or'

button (bottom right of display pane) enables the system designer to specify how to combine the horizontal and vertical components of a rule. Figure 7 shows an 'and' combination: wiring junction components must satisfy both horizontal and vertical position constraints. Or, for example, suppose the system designer specifies a grid rule using horizontal and vertical narrow bands. The 'and' combination would require components to be positioned only in the small squares where the bands cross, while the 'or' combination would allow components to be placed in either vertical or horizontal bands.

3.4 Assembly Rules

Figure 8 shows CKB's assembly rule maker window. The display pane shows two components, selected from the system and component catalog scrollers at the bottom. The current assembly rule (named PIPE-ELBOW-ASSEMBLY), illustrated in the display pane by offset arrows and alignment lines, was selected from the scroller at the right. The system designer can add new rules, change rule names, and associate rules with components using the buttons at the right below the assembly rule scroller.

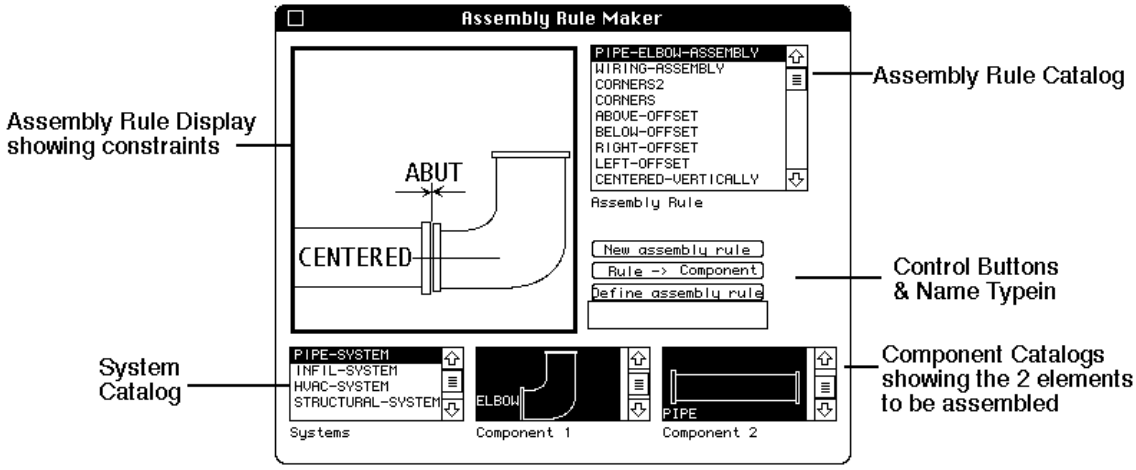

Figure 8. Assembly Rule Maker Window.

The system designer constructs an assembly rule by direct manipulation, dragging the two components in the display pane to define the rule's spatial relation. CKB instantiates a pair of horizontal and vertical offsets that describes the configuration, choosing by default the nearest outside edges of the two components. These offsets are displayed as witness lines (arrows showing dimension and alignment) in the Assembly Rule Maker display pane. The system designer can modify the assembly rule by selecting and operating on these witness lines.

The designer can modify the new assembly rule in several ways. (1) By selecting a witness lines and deleting it the designer removes one of the offsets, enabling the component to move freely in one dimension while remaining fixed in the other. (2) By selecting a witness line and performing the "set interval" operation, the designer can change a fixed offset to an interval, enabling the component to move freely through a limited range. (3) By selecting an offset and performing the "change reference position" operation the designer can change an edge offset to a center alignment or to an arbitrary reference position on the component, or change the selection of edges from which offsets are calculated.

After constructing an assembly rule the system designer can associate it with the pair of components (with the ‘Rule —> Component’ button in figure 8). This step is important, for assembly rules are only available for components if the system designer explicitly says so.

3.5 The end designer’s view

The layout window (figures 9 and 10) is the end designer’s view of CKB. An underlay drawing may be placed beneath the grid or grids in the layout. The scrollers at the left display catalogs of systems and components, assembly rules, grids, and grid position rules. The end designer may not alter the definition of a technical system, only view its various rules; therefore often the catalogs merely provide feedback to the end designer as to the currently operating grid, grid-rule, and assembly rule.

Below are two examples of using CKB to design within a technical system. The first illustrates the use of grid positioning rules to coordinate the layout of materials in a stud wall. The second example illustrates the use of assembly rules in connecting a pipe layout.

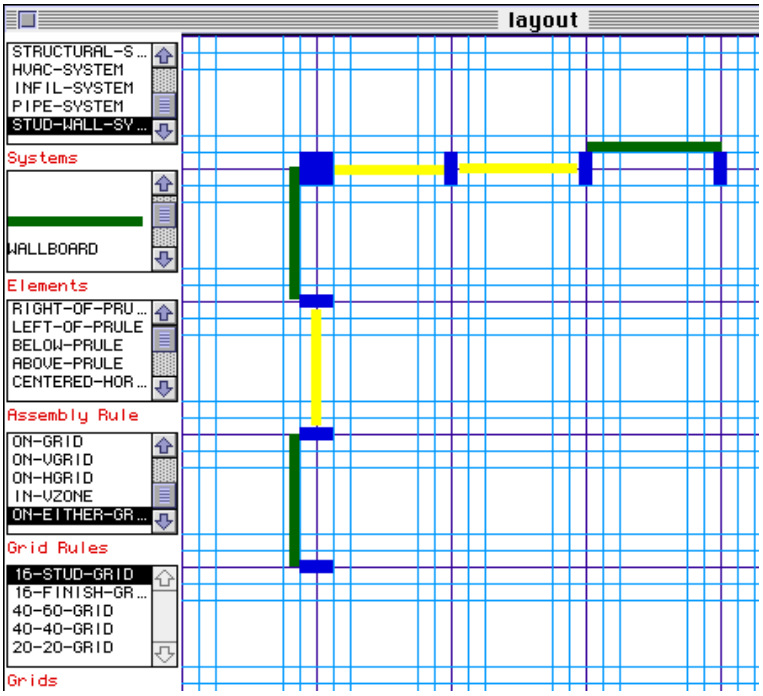

Figure 9. End designer’s view—grid position rules keep the studs on 16” centers, the windows between them, and the plyboard in the bands of an outer tartan grid.

In figure 9 an end designer is working with the components of a stud wall system. When the stud wall system is selected, CKB displays the 16” square grid (dark grid lines) associated with that system, and the grid positioning rule that locates studs on 16” centers. After placing 2x6” studs and a 6x6” corner piece (which snap into place on the 16” grid), the designer then placed narrow windows between some of the studs, and plyboard to finish the outside of the walls where there were no windows. The windows' grid rule centers them along their length on the 16” grid lines, and centered in the bands between the 16” lines in the other direction. When the designer selects the plyboard, CKB introduces another grid to the layout, (shown

in light gray) which provides bands for material on either side of the studs. The plyboard snaps into place in these bands to make the outer shell of the wall.

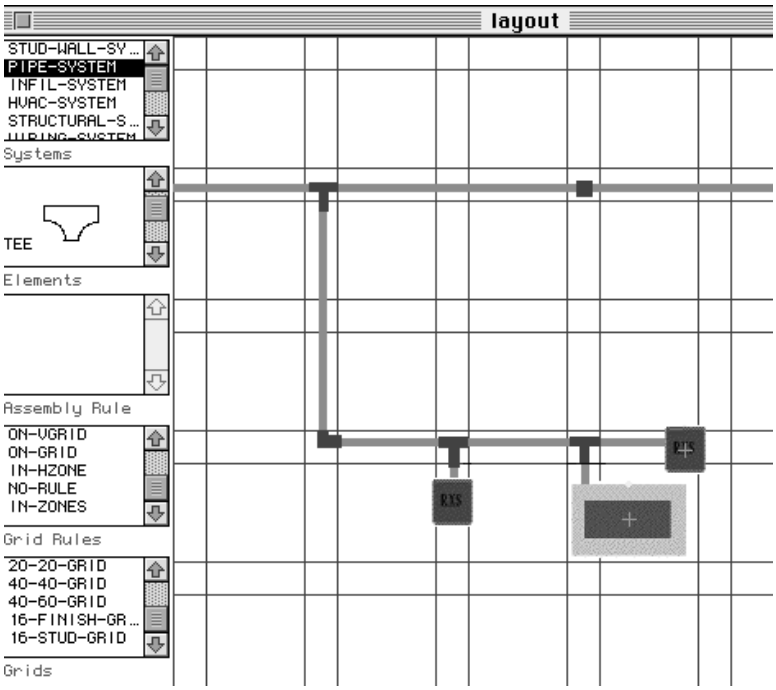

Figure 10. End designer's view— grid rules keep the pipes and fittings in the narrow bands and assembly rules ensure they join properly.

In figure 10 an end designer is assembling a layout of pipes, fittings, and fixtures. As with the previous example, when the designer selects the pipe system CKB displays the tartan piping grid. The system designer has programmed pipes to remain in the narrow bands, so this grid rule positions the pipes and fittings As the end designer drags each fitting, fixture, or pipe it snaps into the nearest narrow band. Then if the new component is near to a component already in the design, then CKB tries to find an assembly rule to join them. If none do, CKB lets the end user continue to drag the component. If any assembly rules can join the two components in their current orientations these become immediate candidates. If only one assembly rule is viable then the rule is applied and the components joined. If more than one rule is viable, then CKB filters them further to see which assembly rule most closely matches the current spatial relation of the two components and then applies that rule. If the program cannot decide, it leaves the choice to the end designer, who must select an applicable rule from the assembly rule scroller. The pipes, fittings, and fixtures snap together according to the assembly rules that the system designer has previously specified.

If the end designer brings together two components for which there are no legal connections, (e.g., a water pipe and a doorknob) then CKB simply does not provide assembly options. The end designer is still free to place the two components in close proximity (assuming no grid position rules prohibit it) but CKB will not assemble them.

4. Implementation

The implementation of CKB follows a straightforward object oriented, constraint based approach. CKB provides CLOS classes for the four parts of a technical system, and the various 'Maker' windows provide an interface for the system designer to construct and extend the class hierarchy. The key slots of these classes are described below.

Component:	Grid:	Assembly Rule:	Grid Position Rule:
Reference-point	Horizontal Spacing	Component-1	Horizontal Type
Grid Position Rule	Vertical Spacing	Component-2	Vertical Type
Grid	Horizontal Offset	Horiz.Relative Position	And/Or Combination
Assembly Rules	Vertical Offset	Vert. Relative Position	Horizontal Lines/Bands
Applicable Asbly Rules			Vertical Lines/Bands

Table 1. Key slots of the four CKB classes.

The *reference-point* of each component (table 1-a) is its 'hot spot', or the position used when placing the component relative to a grid line. This slot value defaults to a function that computes the component's center. Each component also contains *grid* and *grid position rule* slots. If these two slots are filled, an instance will obey that grid position rule with respect to that grid. In a similar fashion, each component has a slot called *assembly rules* that lists the assembly rules that constrain its position. The value of the *applicable assembly rules* slot determines which assembly rules can be used with the component (discussed further below). Of course, components also have slots for their position, orientation, color, and other display characteristics but for simplicity these are not shown in Table 1.

Each grid has slots that determine its *horizontal* and *vertical spacing* and *offsets* (table 1-b).
Each assembly rule has two slots *component-1* and *component-2* that point to the components it joins (table 1-c). In addition, the *horizontal* and *vertical relative position* slots determine the actual spatial relation of the two components; these slots are filled with functions that compute the horizontal and vertical positions of one component from the other.

Each grid position rule has slots *horizontal type* and *vertical type.* (table 1-d) These slots are filled with an indicator, either 'in-band' or 'on-grid-line', which determine the behavior of the rule. The *and-or-combination* slot determines how the horizontal and vertical parts of the rule interact. The *horizontal* and *vertical-lines/bands* list the lines or band positions for a component constrained under the rule.

Essential to CKB's implementation is the way instances of the grids, components, grid rules, and assembly rules are connected. Figure 11 shows schematically how CKB repesents the connection between a pipe and an elbow. Each component points to a grid and grid positioning rule. The two components belong to the same system so they have the same grid and grid rule (though this is not strictly necessary). Both components also point to the assembly rule that joins them, and the assembly rule points back to the two components.

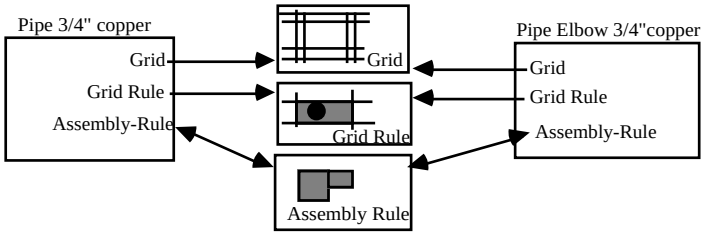

Figure 11. Connections among instances of a component, grid, and position rules

CKB's layout editor determines what assembly rule to use by looking on a list stored in the *applicable assembly rules* slot in each candidate component (see table 1-c). An assembly rule can only join two components if the rule appears on both components' lists, and if the two components are in the correct orientations. When the end designer brings one component near another to join them, CKB searches to see whether any assembly rules apply.

Simple constraint machinery carries out CKB's rule governed positioning behavior. The constraint managing routines (1) ensure that components remain correctly placed according to their grid relative positioning rules and (2) propagate changes in the positions of components assembled together. The constraints involved in grid and assembly rules involve only linear inequalities and integer rounding.

Each position rule is implemented as a function that converts raw coordinates to comply with the constraint. As the designer drags a component with the mouse, the function computes its modified position according to the rules constraining the component. The grid rule that centers a component on grid crossings converts raw component coordinates using a function based on the grid spacings in grid g_1:

$$\{x_{actual}, y_{actual}\} = (\text{g-rule}_{crossings}\ g_1\ \{x_{raw}, y_{raw}\})$$

When the component is assembled with other components, then the assembly rules also provide functions to modify the component's position, in a similar way to the grid rules. An assembly rule relates two components: therefore it encapsulates two complementary functions, each computing the position of one components in terms of the other. For example, the rule for two components c_1 and c_2 assembled with abutting left and right sides comprises two complementary functions for computing the coordinates of c_1 and c_2 :

$$c_1: \quad \{x_{actual}, y_{actual}\} = (\text{a-rule}_{right\text{-}abut}\ c_2\ \{x_{raw}, y_{raw}\})$$
$$c_2: \quad \{x_{actual}, y_{actual}\} = (\text{a-rule}_{left\text{-}abut}\ c_1\ \{x_{raw}, y_{raw}\})$$

These various functions that modify a component's position are composed, with global (grid) positioning rules operating first, followed by local assembly rules. The order of composition is the order in which the parts were assembled. For example, the expression below computes the actual coordinates $\{x_{actual}, y_{actual}\}$ of the reference point of a component c_3, given the initial coordinates $\{x_{raw}, y_{raw}\}$ subject to its assembly with two components c_1 and c_2 and its placement relative to two grids g_1 and g_2.

$$c_3: \{x_{actual}, y_{actual}\} = (\text{a-rule}_1\ c_1, (\text{a-rule}_2\ c_2\ (\text{g-rule}_2\ g_2\ (\text{g-rule}_1\ g_1 \ldots\ \{x_{raw}, y_{raw}\}))))$$

Each component maintains a list of co-dependent components with which it is assembled so that when the end designer moves one component, CKB updates the others as necessary, propagating the change in standard constraint based fashion. As each component is updated, its position is again run through the modifying function to ensure compliance with other constraints.

Conflicts are possible among the various positioning rules that apply to a component. There are three opportunities for detecting a conflict among rules: (1) when the technical system designer sets up grid and assembly rules for a component (system definition time); (2) when the end user assembles a new component into an existing layout; and (3) when the end designer selects and moves a component. CKB's current implementation does not detect all potential conflicts between grid-relative and assembly rules at system definition time. However, CKB does help avoid conflicts during system definition. It prevents the system designer from assigning a grid relative rule that is incompatible with a component (e.g, because the component is larger than the space the rule would allow it). During layout, the constraint propagating functions detect and handle conflicts among assembly rules and grid position rules. The layout editor will not apply a new assembly rule or move a component if the action will result in a position for the component inconsistent with any of its rules.

5. Related Work

N. John Habraken has developed the theoretical framework from which CKB follows. His *General Principles about the Way Built Environments Exist*

(Habraken

1979) made the deceptively obvious observations in section 1.1 about selection and positioning decisions. The SAR design methods

(Habraken, Boekholt et al.

1976) of the ‘sixties and ‘seventies and the recent

Matura™ system (Matura

) for dwelling infil are based on these observations, and provide numerous examples of the uses of grids and zones in design, as well as the value of working explicitly according to rules that govern the position and dimensions of components. Wouter Habraken's

MaturaCAD/S program (Habraken

1994) which supports design using the Matura™ system, is a good example of the kind of highly specialized technical system editor that could be built within CKB.

Coordination of components in a dimensional system is an old idea in computer aided design, yet surprisingly modern systems do not take advantage of it. Mitchell

(Mitchell

1977) (p 243) mentions associating position rules with components and describes two CAD systems. The Harness program for hospital design employed automatic positioning of structural members according to a modular

system (Jacobsberg

1975) . Likewise, the OXSYS system performed automatic detailing of components once a plan was determined

(Hoskins 1973; Richens

1974) . However, both these systems operated with a predefined building system written in to the program, rather than allowing the designer to specify the rules for locating components. Despite such promising early efforts, CAD systems today (e.g., AutoCAD) do not support component coordination in any straightforward manner.

Building models, in which a single representation of the components of a design generates views for diverse purposes have been a central theme in CAD research

(Eastman, Lividini et al. 1975; Björk 1992; Eastman

1992; Harfmann and Chen 1993) . The A4 system

(Hovestadt

1993) , initially derived from Fritz Haller's modular construction kit MIDI and arrangement methodology ARMILLA, addresses some of the same issues as CKB, especially spatial coordination of mechanical systems and building services. Within this framework CKB can be viewed as managing semantic information about placement and assembly conditions attached to the various components. However, most building models do not make CKB's conceptual distinction between technical system designer and end designer.

CKB uses constraint programming techniques to enforce and maintain position rules. Constraint based approaches have been developed for many CAD domains, especially assembly planning and kinematics for

mechanical engineering (Kramer

1992) . In addition to my own work (cited above), the application of constraint based programming techniques to architectural design has been explored by (among

others) Frazer (Frazer

1988) , Tobin

(Tobin

1991) and Kolarevic

(Kolarevic

1994) . CKB goes beyond these efforts in that default constraints are associated with components, and instantiated automatically as the user adds the components to the design.

The positioning behavior of components in CKB will remind CAD users of snap-dragging (also called

'object snap'). Snap-dragging (Bier and Stone

1986) , is a user interface technique for graphic editors—dragged elements snap to align with edges and centers of other elements in the drawing. It has been employed in CAD editors such as Ashlar's Vellum™ and auto-des-sys's **form•Z**™. In CKB preferred positions are customized for the components rather than using generic edge and center alignments. The preferred positions correspond to assembly rules, which the system designer has defined. Also, snap-dragging is a one-time operation that happens only when dragging and placing a drawing element, but in CKB once two components are joined their assembly condition is a constraint, maintained by the program until the end designer takes them apart.

6. Discussion

6.1 Applications in coordinating group work

CKB began as part of a larger project to coordinate team design work, specifically the layout of architectural

subsystems in design (Gross

1994) . If, as in Habraken's design methods, the various members of a design team responsible for the different subsystems—pipes, wires, HVAC ducts, and so on—can agree to work within certain position rules, then they can largely avoid interference conflicts that can be costly in big buildings if discovered late. By agreeing to locate each subsystem in its own grid band or on its own grid line, interference conflicts can be minimized and reduced to predictable occurences where bands or lines cross. Then a stored library of predefined conflict resolutions can be provided, indexed on the grid band intersections.

6.2 Application in space planning and other levels of architectural design

CKB is described here as a program for building construction kits for technical systems, the low-level components that are assembled to make buildings. This is an obvious place to start because these technical systems work according to standard position rules and most architects accept these rules without argument. Therefore, a CAD system that knows and can maintain the rules of construction assembly would seem to be unarguably a good thing.

But it is not difficult to imagine using CKB to program construction kits for higher level systems of built components and even for systems of spaces as well. Of course, this requires the designer to work with these higher level components and spaces as a technical system, that is, according to predefined position rules. In this case it is necessary for the architect to play both the roles of system designer and end designer. At this level of design the rules are by no means standard; rather they are both made and followed by the designer.

6.3 Conceptual limits of the Lego approach

Lego is a closed-system construction toy that imposes strong constraints on the configurations that designers can make. It has a capability that today's CAD programs still lack: it operationalizes rules for the arrangement of components. But children playing with Lego operate as end designers only—unable to alter the technical system. The creative architect not only follows rules, but operating as a system designer, sets up the rules as well. CKB is not intended to force architects into a single predefined technical system. Rather, CKB provides the ability to define a technical system.

CKB is an open-system design editor. A good designer knows not only when to follow the rules, but when to break them. CKB gives the designer the discretion to override or relax placement and assembly rules, and occasionally to use special building components in ways to achieve special effects. The program provides support for rule-making and rule-following, but the designer must ultimately decide when to follow the rules and when to abandon them. Constraints can always be relaxed locally. In a similar vein, the designer need not cover the entire design with a single uniform grid. More often several grids will be combined, perhaps rotated to set up two non-orthogonal directions. Where two grids meet or overlap, the designer must determine which rules to apply, or whether to handle the condition as an exception.

I have chosen simple examples to illustrate the CKB program. This raises the question of whether design methodologies that employ rules will stifle creative design, producing perhaps buildings but never Architecture. Experience with modular building systems and particularly closed systems suggests to many architects that unimaginative outcomes inevitably result from rule governed design. However, I believe this conclusion is false. On the contrary, most skilled designers lay out the components of their buildings systematically. The architecture of Lucien Kroll

(Kroll

1987) demonstrates that even within a standard, modular, and rule-governed system, a creative designer can achieve imaginative results. But other architects too, (e.g., Hertzberger, Eisenman) though perhaps less obviously, work systematically. The question is not whether rule governed layout limits good design; the question is, can we build programs like CKB that are up to representing the more subtle rules that characterize well designed buildings?

6.4 Technical limits of the CKB implementation and directions for further work

The project has raised a number of directions for further work, both detail enhancements that would make it practical and useful for architects, and more deeper limitations of the implementation that should be addressed in a future round of development. For example, CKB handles two levels of design hierarchy but designing can be seen more generally as a deeper hierarchy of systems. It seems clear that the approach followed in CKB could support this multi-level hierarchy of design decision making, but the details (e.g., the relations between levels) remain to be worked out.

Non-orthogonal grids. CKB works within the limits of orthogonal grids. For example, it does not permit the designer to work with two grids that meet at an angle other than 90°. This is clearly a limitation for practical design work. However, it does not pose serious conceptual difficulties.

Assemblies. CKB currently does not provide for working with assemblies of parts: It is a flat system. Another direction for further work is to allow for abstract representations of components at higher levels to be replaced by more detailed configurations. For example, what appears to be a wall configuration at one level could be specified as a more detailed configuration of studs and wallboard. At each level of design, a different set of position rules would come into play. At the wall configuration level, the entire wall might be located on a large scale grid; while at the stud-wallboard level another set of material positioning grids would govern the positions of studs and wallboard.

Variable and parametric dimensioning. The current implementation of CKB does not support components with variable dimensions or parametric dimensioning. Components in the catalogs have fixed dimensions and the end designer can change the dimension of an individual component after placing it into a layout.

However, variable length components (e.g., a pipe that stretches to stay connected to fittings at either end) and parametrically dimensioned components (e.g., a beam whose cross section becomes larger as its span is lengthened) would be a valuable addition to the program.

Selection advisor. I noted in section 1.1 that design involves decisions about the selection and positioning of components, but in the rest of the paper I have only discussed positioning decisions. CKB does not support decision making about the selection of components. Another direction for further work is the development of a selection advisor, which could respond to specific queries by end users: What component can work here? (providing these functions, meeting these constraints, etc.)? Component selection advisors have been built using various AI techniques, and it would not be difficult to add a module like this to the end designer environment.

Three dimensional display. Perhaps, most obviously lacking from the current implementation is three-dimensional display. Extending CKB to model three-dimensional forms would be a valuable enhancement, and nothing intrinsic to CKB's architecture limits it to two dimensions. Indeed, the current version can keep track of three-dimensional positioning information, though it can only display it in two dimensional (plan and section) views.

6.4 Conclusion

CKB is a CAD program that operates at two levels of design. At the higher (technical system) level CKB enables the designer to specify components and the rules for their placement and assembly. At the lower (end user) level, CKB provides layout editing facilities for a designer to work within previously programmed technical systems. I have deliberately chosen to illustrate CKB with construction details. When the technical systems reflect standard construction methods, the end designer usually accepts the given components and rules of the system. But for less routine design tasks, such as layout of functional spaces, often an architect first designs the elements and rules of a system, then works within them to realize particular built configurations. In these nonroutine design domains, the technical system designer and the end designer are likely to be the same person playing different design roles.

Acknowledgments

This work was supported by NSF grant DMI 93-13196. Thanks to Ellen Yi-Luen Do and Raymond J McCall for commenting on early drafts, and to the anonymous reviewers for constructive remarks that strengthened the paper.

References

Bier, E. A. and M. C. Stone (1986). “Snap-Dragging.” Computer Graphics **20**(4): 233-240.
Björk, B.-C. (1992). “A conceptual model of spaces, space boundaries, and enclosing structures.” Automation in Construction **1**(3): 193-214.
Eastman, C., J. Lividini, et al. (1975). A Database for Designing Large Physical Systems. Proceedings of the 1975 National Computer Conference. Montvale NJ, AFIPS Press.
Eastman, C. M. (1992). “Modeling of buildings: evolution and concepts.” Automation in Construction **1**(2): 99-110.
Frazer, J. (1988). Plastic Modeling - the flexible modeling of the logic of structure and space. CAAD Futures ‘87. Amsterdam, Elsevier. T. Maver and H. Wagter, ed. 199-208.
Gross (1994). “Avoiding Conflicts in Architectural Subsystem Layout.” Concurrent Engineering: Research and Applications (2): 163-171.
Gross, M. (1990). Relational Modeling. *The Electronic Design Studio*. Cambridge, MA, MIT Press. M. McCullough, W. Mitchell and P. Purcell, ed. 123-136.
Gross, M. (1991). Grids in Design and CAD. ACADIA ‘91, Los Angeles, CA, ACADIA, 33-43.
Gross, M. D. (1992). Graphical Constraints in CoDraw. IEEE Workshop on Visual Languages. Seattle, IEEE Press. S. Tanimoto, ed. 81-87.
Habraken, N. J. (1979). General Principles about the Way Built Environments Exist. Stichting Architecten Research (SAR), Eindhoven.
Habraken, N. J., J. T. Boekholt, et al. (1976). Variations - The Systematic Design of Supports. Cambridge, MA, MIT Press.
Habraken, W. (1994). Tools for managing design information in the construction process. Design and Decision Support Systems II, Vaals, Netherlands.
Harfmann, A. C. and S. S. Chen (1993). “Component-based building representation for design and construction.” Automation in Construction **1**(4): 339-350.
Hoskins, E. (1973). “Computer Aids for System Building.” Industrialization Forum **4**(5): 27-42.
Hovestadt, L. (1993). A4 Digital Building: Extensive Computer Support for Building Design, Construction, and Management. CAAD Futures ‘93. Amsterdam, Elsevier. U. Flemming and S. V. Wyk, ed. 405-421.
Jacobsberg, J. (1975). Computer Design Aids for Large Modular Buildings. Models and Systems in Architecture and Building. Hornsby, Lancaster, England, The Construction Press. D. Hawkes, ed.
Kolarevic, B. (1994). Lines, Relations, Drawings and Design. ACADIA-94, St Louis, MO, ACADIA, 51-62.
Kramer, G. (1992). Solving Geometric Constraint Systems. Cambridge MA, MIT Press.
Kroll, L. (1987). An Architecture of Complexity. Cambridge, MA, MIT Press.
Matura The Matura System. Voorerf 14, 4824 GN Breda, Postbus 6903, 4802 HX Breda.
Mitchell, W. J. (1977). Computer-Aided Architectural Design. New York, Petrocelli/Charter.

Richens, P. (1974). OXSYS: Computer Aided Building for Oxford Method. Cambridge, UK, Applied Research of Cambridge.
Sutherland, I. (1963). Sketchpad - a Graphical Man-Machine Interface. M.I.T. PhD dissertation.
Tobin, K. (1991). Constraint Based Three Dimensional Modeling as a Design Tool. ACADIA-91, Los Angeles, CA, ACADIA, 193-210.