Toward Making the Language of CAAD Match the Language of Architecture:

A Protean Elements Approach

Scott Johnson

University of Michigan

Doctoral Program in Architecture

2000 Bonisteel Blvd.

Ann Arbor, MI 48109-2069

USA

Both in education and in practice, architecture is experiencing a division between designers and "CAD specialists." One reason for the division may be the inherent division between design concepts and CAD concepts. In a very real sense, computer use and design utilize different languages. Becoming an expert in the "craft" of CAD means having to learn to recognize and manipulate a different set of conceptual elements than is used in design. The set of concepts we use affects our thought and behavior incredibly deeply, and translation from one set of concepts to another has significant cognitive cost. This paper discusses the mismatch between architectural and CAD concepts, and proposes protean elements as a solution to the problem. Protean elements are CAD system elements which correspond to architectural elements and have attributes appropriate for the elements they represent. They can be gradually refined in a top-down manner, without demands for certain pieces of missing data, or requirements for "correctness." The goal is to help CAD systems come closer to speaking the same language as architects. A test implementation of a system based on protean elements is currently underway, and aspects of this implementation are discussed.

Introduction

Many schools of architecture experience a division between faculty who teach design studio, and faculty who teach computer courses. While the division is not universal, there are many cases where the groups seldom interact, seldom become involved in each other's classes, and seldom advise the same students.

There is reason to think that the division between studio and computer courses is more fundamental than mere interpersonal problems between faculty. Disproportionately many students who do well in studio courses seem to do poorly in computer courses, and vice versa. It is as if studio design and use of CAD systems require different skills, different ways of thinking.

A person walking through an architecture studio might hear bits of conversation like, "You might try putting a window in that wall, and raising the ceiling about half a meter to lighten it up and give it a more open feeling."

Rooms, walls, columns, windows, ceilings, light sources, and other elements of design are manipulated as a designer works.

However, a person walking through a computer lab is likely to hear completely different terminology. Instead of "putting a window in that wall," a CAD user should "create a polyhedron and subtract it from that polyhedron." Instead of "raising the ceiling half a meter," a CAD user should "move it by (0,0,0.5)." Lines, arcs, polygons, symbols, CSG trees, or even scripting instructions are manipulated as a CAD user works.

CAD systems for architecture need to do a better job of using the terminology and underlying concepts of architects. It is more than a matter of etiquette. A person's mental performance is actually inhibited if the person is forced to translate while trying to perform a task.

Cognitive Resources

A person's mental performance is limited by the operating characteristics of the human mind. Psychologists describe the mind as being composed of certain cognitive resources and mechanisms, some of which are quite limited in nature. Key among these are short term memory and attention.

Short-term memory is used to hold information that is currently in use: numbers that are being added, sentences that are being composed or read, and so forth. We also use it to keep our place when performing a complicated task, and to imagine a scene when forming a mental image. However, short term memory is very limited. It holds information for only a few seconds (unless the subject constantly repeats it). It also holds only a small amount of information at a time: the verbal component holds about seven words, and the organizational and visual components are similarly limited (Hayes 1989, 111-112, 120-128; Schacter 1989).

Attention directs mental processing. Well-practiced tasks can proceed without conscious attention, but in general, a person can only attend to one thing at a time (Anderson 1990, 52-56).

When these cognitive resources are exceeded, performance suffers. When short-term memory capacity is exceeded, or when information decays, we find ourselves forgetting information we are using. We forget the phone number before we can dial it, forget how a sentence started, or lose track of what we were doing. When there are too many demands on our attention, we must slow down. Something must be delayed until we can get around to devoting attention to it (which can, in turn, give the contents of short-term memory time to decay). For peak mental performance, drain on short-term memory and attention should be minimized.

Expert Representations

Experts in a field -- any field -- have ways of minimizing drain on short-term memory and attention. When they encounter a certain situation or combination of elements over and over, they can mentally "chunk" the parts together and think of them as a single thing (Hayes 1991, 121-126). When they perform a procedure repeatedly, they can "automatize" it (Norman 1991, 23-24; Anderson 1982). That is, they can "compile" it into a mental process that uses less short-term memory and demands little or no attention once started. By using these mental chunks and automatized thought processes that deal with them, experts make more efficient use of their cognitive resources, allowing more resources to be brought to bear on the task at hand.

Language is one example of a domain where chunks and automatized processes are used (LaBerge and Samuels 1974). Growing up, we learn to associate words with phenomena, characteristics, actions, feelings, and so forth. We also learn how to make letters, and get good enough to be able to put letters together into words. After years of practice, our proficiency improves. We no longer have to concentrate on how to draw the letters. We reach a point where we can read and write without "sounding out" the words, and we can devote out attention instead to the *content* of the text. If we learn a second language, it takes many years of practice before we can understand difficult text in that language as well as we can understand difficult text in our native language.

Just as we develop automatized processes for reading, we develop automatized processes for recognizing elements and situations and acting accordingly in other domains (Anderson 1985). In the same way that readers learn to chunk penstrokes into letters and letters into words, and to associate meanings and connotations with them, we architects similarly learn to recognize architectural forms and to associate them with structural, functional, and symbolic meanings (Schön 1988). We even develop a jargon, so that we can discuss these concepts with others in our line of work. Thus, the chunks and processes underlying our jargon actually help us think more efficiently.

Importance of Chunks and Automatization

It is impossible to over-emphasize the importance of these chunks. They are extremely deeply rooted, influencing us sub-consciously, and affecting even the way we move our eyes (Hayes 1989, 72-73).

The significance of automatized processes is demonstrated by the Stroop Effect (Boutilier 1998). The Stroop Effect involves a list of color words, with each word written in a different color of ink. For instance, the list might have the word „red" written in green ink, followed by the word „yellow" written in blue ink and the word „green" written in yellow ink, and so forth. The subject's task is to go through the list of words and indentify the color ink each word is written in. The difficulty of the task is surprising – the reader should try this in order to get understanding of whet is happening. As the sbject tries to name the ink colors, automatized processes for reading execute automatically, causing a barely suppressible urge to name the wrong color. The effect is immediately obvious to the subject and all observers as the subject struggles to keep from naming the written words. Subjects in the experiment take considerably more time to identify colors, compared to people merely trying to name the colors of colored squares.

Developing a library of chunks and automatized processes is no trivial matter. A person can study a language for years and still not be able to read a difficult technical paper written in that language. Weisberg (1986, 13) estimates that the library of an expert contains 20,000-50,000 chunks. Hayes (1989, 293-298) estimates that it takes 10 or more years of working 70-80 hours a week to develop this library.

Language of Architecture

A number of researchers in the area of design studies have identified concepts essentially similar to chunks, that are used by architects in the process of design. Alexander's (Alexander, et al., 1977) patterns are an example, as are the elements described by authors like Krier (1988), Thiis-Evensen (1991), Ching (1979), and even Durand (1802) in architectural primers. Schön (1988), and Peter Rowe (1987) describe more extensive categories of "enabling prejudices," "heuristics," "types," typologies," etc.
The use of these elements in design is not unlike the way that letters and words of a language are used, with elements being recognized and used with little effort. Indeed, design researchers have also noted the similarity. Francis Ching (1979, p.11), for instance, writes:

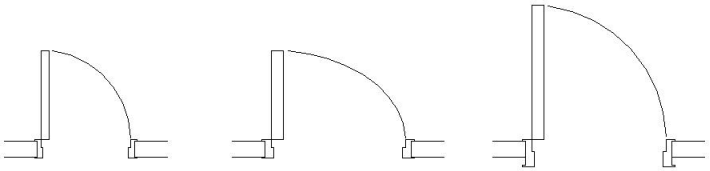
> The analogy can be made that one must know and understand the alphabet before words can be formed and a vocabulary developed; one must understand the rules of grammar and syntax before sentences can be constructed; one must understand the principles of composition before essays, novels, and the like can be written. Once these elements are understood, one can write poignantly or with force, call for peace or incite to riot, comment on trivia or speak with insight and meaning. It should be useful, therefore, for the student of design to recognize the basic elements of architectural form and space, understand how they can be manipulated in the development of the design concept, and realize their visual implications in the implementation of a design solution.

There is increasing consensus that architectural design involves elements like walls, columns, rooms, roofs, floors, doors, windows, and so forth. Psycho-linguistic evidence suggests this is no coincidence; these categories are based on salient features of and our methods of interacting with these elements (Rosch 1978). These elements are appropriate for the task of architectural design. They have distinct, identifiable forms. They have distinct functional, structural, and/or symbolic roles.

Language of CAD systems

Elements in CAD systems are generally based on geometric/mathematical representations that facilitate coordinate transformations and other mathematical operations. Elements like lines, circles, planes, polyhedra, and symbols are found in most drafting and modeling programs. Combinations of these elements often look like architectural elements, but it can be hard to manipulate them in the ways we would like to manipulate elements of an architectural design. For example, figure 1 shows an attempt to manipulate a door symbol to reflect changing the width of the door. Neither changing the x scale factor, nor changing both the x and y scale factors produced the desired effect.

Figure 1. Trying to change a 24" wide door symbol (left) to represent a 36" door, by changing one (middle) or two (right) scale factors. Note how the thickness of the door and size of the jambs is affected by scale factors.



A few systems use other sorts of elements. Systems based on parametric shapes typically utilize a set of shapes, that are put together to build a model. Parametric values for each instance of a shape describe how much to "stretch" a certain parts of the shape. While this is much more flexible than simple symbols, the approach is inherently limited topologically. Consider parametric shape representations for windows, for instance (figure 2). By altering the parameters of the window on the left, many different windows can be produced. However, no combination of parametric values will produce the window on the right. A different parametric shape is required, because the window on the right is topologically different from the window on the left. An infinite number of parametric shapes would be required to represent every conceivable window.

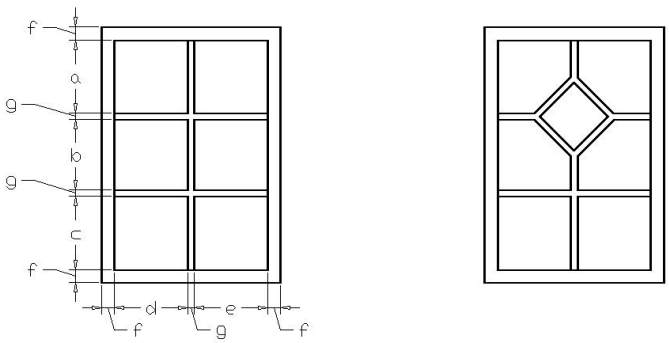


Figure 2. The parametric shape on the left can not handle the topological change to produce the window on the right

Constructive Solid Geometries (CSG's) are sometimes used as representations in modeling packages. A CSG model is constructed by creating solid volumes, transforming (moving, rotating, and/or scaling them) and combining them using set operations (union, intersection, or difference). Sometimes, the commands by which a model was generated can be edited (for instance substituting a cylinder for a sphere or a union for an intersection). CSG systems are complete, in that any form imaginable can be modeled, but they tend to be oriented toward the creation and combining of geometric primitives, rather than architectural elements.

Other systems allow the user to create models through a procedural assembly method. Instead of interacting with the model graphically, the user edits a command script. When the command script is run, the computer executes the scripted commands to generate a model. Alterations to the model are made by editing the command script and generating an entirely new model. The approach allows any form to be modeled, but only after a rather severe translation from thoughts about form and space to computer programming concepts.

Some other experimental systems have been created, where elements are more directly architectural in nature. Examples include ARCH:PLAN (Turner and Hall 1985), P3 (Khemlani et al. 1997), and perhaps EDM (Eastman 1992). These systems, however, have rigid requirements governing what constitutes a "correct" model: appropriate hierarchies of sub-elements must have been painstakingly created, or relationships between certain elements (such as walls and rooms) must be carefully maintained at all times. These requirements can make such systems difficult to use in early schematic design, when such relationships may not yet have been clearly resolved.

Translations

Despite the various attempts that have been made, representations in CAD systems still do not do a good job of matching the mental representations used by architects. This is a problem because it means that architects must translate their design ideas from their internal representations into the representations inherent in the CAD system. It takes cognitive resources away from other concurrent tasks (Day 1988), so it is actually detrimental to thinking and decision making in design. Having the architect perform translations is to be avoided, if at all possible.

Although one sometimes hears talk of architects changing the way they think about architecture, this is also not a very viable solution. This would eliminate the need to translate ideas into the "language": of the CAD system, but it would take years to thoroughly learn the new library of elements and the associated mental processes. Even if we, as educators, spent the next decade teaching the next generation of architects to think of architecture in terms of mathematical descriptions, the effect would still be likely be detrimental to their ability to design. Architects think and talk about architecture in terms of elements like walls, rooms, doors, and so on, because it is easier to think about buildings in these terms. These concepts chunk together information about appearance, function, meaning, and other topics relevant to architectural design. Psychologist Donald Norman (1991, p. 29) notes that while it is possible for experts to learn to work with less efficient representations, performance will suffer at times of peak cognitive demand.

Forcing the architect to translate design ideas into the CAD system's representation interferes with design, rather than aiding it. Trying to change the memory structures and subconscious thought processes used by architects is not a promising approach, either. A better approach is to build CAD systems that work with the architect on the architect's terms.

Protean Elements: An Attempt Bridge Gap Between Design and CAD

At the University of Michigan, an attempt is currently underway to bridge the gap between architectural elements and CAD elements. A test program is being developed using what are called "protean" (meaning "easily changed") elements. Protean elements correspond to elements like walls, rooms, doors, windows, and so forth. They have characteristics appropriate for the element they represent, like from point, to point, thickness, and height of a wall, or location, radius, and height for a round column. The characteristics of protean elements are stored in memory so that they can be edited later. Protean elements are nestable, so that one element might be included within or used to generate another protean element. The elements also have a hierarchy of specificity, so that as the design becomes better resolved, more basic elements that were used earlier (e.g., a dome) can be replaced with more specific elements with additional attributes (e.g., a ribbed dome, with additional attributes describing shape and placement of ribs).

Of course, high-level elements like wall, room, and so forth can not be drawn directly. Instead protean elements produce graphical representations based on the information they contain. In the test implementation, all protean elements produce polyhedral representations, which are then drawn using conventional viewing methods. A display list is used to track backwards from lines in the drawn image to the elements that produced them, allowing elements to be selected by clicking. Thus, it is the program which converts information from architectural to graphical elements, rather than the architect.

Use of Protean Elements in the Design Process

Protean elements are intended to be used in the schematic design portion of the design process, when the designer is experimenting with different massings, different arrangements of rooms, and so forth. At this point in the design process, the design is not completely defined. Parts of the design may overlap or conflict with each other, the design may be vague and sketchy, and the design may change considerably in a short period of time. It is by drawing or modeling these situations that the flaws become apparent, and the architect gradually figures out ways to correct the flaws and inconsistencies in the design. This is a natural stage in the design process, and it is important that the architect be able to keep working in this manner. He should not have any new design methods thrust upon him, and he should not be interrupted and pestered for information. Such disruptions could interfere with the automatized processes he is using.

Protean elements are intended to let the architect to keep using familiar mental processes, with a minimum of intrusion. They assume default values for attributes the architect does not wish to be bothered with. They do not prohibit actions or values that would cause the model to be "incorrect" or self-conflicting. Walls are not

constrained to have a “left room” and “right room,” structural elements are not constrained to bear on other elements, and solids and voids are not prohibited from colliding with other solids and voids. The user is allowed to create whatever impossible, incorrect, or unstable models he desires, and is allowed to correct these problems whenever he is ready.

Structure of Protean Elements

A test implementation, called “Proteus” is currently being developed, using Microsoft Visual C++ running on an IBM Pentium machine. Development is still underway, but so far the support structure for the operation of protean elements is essentially complete, and a couple classes of elements have been implemented to verify the operation of the support structure. This work has verified that protean elements can be kept track of, drawn, modified, identified from screen clicks, and so forth. The next phase of implementation will be to implement a wider variety of classes of architectural elements.

In Proteus, a separate object class represents each type of architectural element. This means that each class has several pieces of information and several procedures associated with it, which allow the element to be drawn and edited. These are shown in table 1.

Wall Object
Wall Data
Name
Class-specific attributes:
From point:
To point:
Height:
Thickness
Class-specific subcomponents:
list of openings:
Polyhedral representation
Associated dialog box
Wall functions
Function to generate polyhedral representation
Functions to retrieve/change attribute values
Function that returns the element's name
Function that returns the wall's height
etc.
Function to display appropriate dialog box
Functions to update menu
Functions that perform class-specific operations
Function that adds a door to a wall
Function that converts a wall to a colonnade
etc.

Table 1. A sample protean element.

Many classes of elements will have subclasses, which will contain the same data fields, plus some additional ones to describe additional features. Element classes will have functions to convert elements to a subtype, or from one subtype to another, allowing the architect to add details as the design is refined.

It is, of course, impossible to provide specific classes for the strangest of architectural elements. There always be a few bizarre sculptures or unique details that are so anomalous that it is impossible to anticipate their characteristics in detail. While such elements comprise a relatively small part of even the most bizarre architectural designs, it is still vital that a CAD representation be able to accommodate them. To this end, a system based on protean elements must include a few classes for describing forms generated through processes like revolving a cross-section around an axis, or sweeping a cross-section along an extrusion path. For situations where even these classes are not enough, a protean system needs classes which handle CSG's or other bottom-up methods for generating forms. This way, a protean element-based system can allow anything to be modeled.

Conclusion

A person learning to design architecture spends an extended period of time learning to recognize and work with a set of architectural elements. The process is very similar to the process of learning a language. Unfortunately, most current CAD systems have a different set of elements, which do not map very well to architectural concepts. The user is forced to either translate architectural ideas into the terms of the CAD system, or try to think in terms of CAD elements instead of architectural ones. Doing either wastes cognitive resources. It is wrong to expect the user to learn a new language or to do translations while he is trying to design. Instead, the computer should do the translation, letting the architect use the processes and concepts that are vital to his ability to design well.

Protean elements are a proposed solution for this problem. By storing information about high-level, architectural elements, and using this information to generate the low-level geometric entities necessary for graphics, the elements allow the architect to work with the sort of concepts used in architectural design. By letting the architect work in a top-down manner, devoting attention to whatever he considers significant, protean elements let the architect continue to use the well-learned processes that have been developed for architectural design.

The initial stage of the test implementation is complete, and the approach is working thus far. Technical issues, like production of a graphical representation based on the architectural data, identification of elements based on screen clicks, have not proved to be a problem. The test implementation is ready to proceed to the next stage, investigation of issues concerning implementation of a variety of element classes.

References

Akin, Ömer. 1986. *Psychology of Architectural Design.* London: Pion Limited.

Alexander, Christopher, Sara Ishakawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, and Schlomo Angel. 1977. *A Pattern Language: Towns, Buildings, Construction*. New York: Oxford University Press.

Anderson, John R. 1982. Acquisition of cognitive skill. *Psychological Review*, 89, no. 4: 369-406.

Anderson, John R. 1985. The development of expertise. Chapter 9 of *Cognitive Psychology and Its Implications*. 2nd ed. New York: W. H. Freeman & Co.

Anderson, John R. 1990. *Cognitive Psychology and its implications*. 3rd ed. New York: W. H. Freeman & Co.

Boutilier, Clay. 1998. What is the Stroop effect? http://www.cgl.uwaterloo.ca/~cjboutil/stroop.html

Chi, Michelene T. H., Paul J. Feltovich, and Robert Glaser. 1981. Categorization and representation of physics problems by experts and novices. *Cognitive Science*, 5, 121-152.

Ching, Francis D. K. 1979. *Architecture: Form, Space & Order.* New York: Van Nostrand Reinhold Co.

Day, Ruth S. 1988. Alternative representations. *The Psychology of Learning and Motivation*, 22, 261-305.

Durand, Jean-Nicolas-Louis. 1802. *Précis des Leçons d'Architecture données a l'Ecole Polytechnique*. Paris, France: Ecole Polytechnique.

Eastman, Charles M. 1992. Use of data modeling in the conceptual structuring of design problems. In *Computer Aided Architectural Design Futures: Education, Research, Applications*, ed. Gerhard N. Schmitt, 225-243. Braunschweig/ Wiesbaden: Friedr. Vieweg & Sohn Verlagsgesellshaft mbH.

Hayes, John R. 1989. *The complete problem solver*. Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Hernandez, Antonio. 1969. J. N. L. Durand's architectural theory: A study in the history of rational building design. *Perspecta* 12: 153-60.

Johnson, Scott E. 1997. What's in a representation, why do we care, and what does it mean? Examining evidence from psychology. *ACADIA 97: Representation and Design*, eds. J. Peter Jordan, Bettina Mehnert, and Anton Harfmann, 5-15. Cincinnati, Ohio: The Association for Computer-Aided Design in Architecture.

Khemlani, Lachmi, Anne Timerman, Beatrice Benne, and Yehuda E. Kalay. 1997. Semantically rich building representation. *ACADIA 97: Representation and Design*, eds. J. Peter Jordan, Bettina Mehnert, and Anton Harfmann, 207-227. Cincinnati, Ohio: The Association for Computer-Aided Design in Architecture.

Krier, Rob. 1988. *Architectural Composition*. New York, New York: Rizzoli.

LaBerge, David, and S. Jay Samuels. 1974. Toward a theory of automatic information processing in reading. *Cognitive Psychology* 6, no. 2: 293-323.

Lakoff, George. 1987. *Women, Fire, and Dangerous Things: What Categories Reveal about the Mind* . Chicago: University of Chicago Press.

Norman, Donald A. 1988. *The Psychology of Everyday Things*. New York: Doubleday.

Norman, Donald A. 1991. Cognitive Artifacts. Chap. in *Designing Interaction: Psychology at the Human-Computer Interface*, ed. J. M. Carroll. New York: Cambridge University Press.

Ockman, Joan. 1990. *Rizzoli Essays on Architecture*. Vol. 3, *J.N.L. Durand:(1760-1834): Art and Science of Architecture*, by Sergio Villari. Translated by Eli Gottlieb. New York, New York: Rizzoli International Publications, Inc.

Rosch, Eleanor. 1978. Principles of Categorization. Chap. in *Cognition and Categorization*, ed. Eleanor Rosch and Barbara B. Lloyd. Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Rowe, Peter G. 1987. *Design Thinking*. Cambridge, Massachusetts: The M.I.T Press.

Schön, Donald A. 1983. *The Reflective Practitioner: How Professionals Think in Action*. New York, New York: Basic Books, Inc., Publishers.

Schacter, Daniel L. 1989. Memory. Chap. in *Foundations of Cognitive Science*, ed. M. I. Posner, 683-725. Cambridge, Mass.: M.I.T. Press.

Schön, Donald A. 1988. Designing: Rules, types, and worlds *Design Studies* 9, no. 3: 181-190.

Thiis-Evensen, Thomas. 1991. *Archetypes in Architecture*. Oslo, Norway: Norwegian University Press.

Turner, James A. and Theodore W. Hall. 1985. The automated generation of room polygons from a weighted and directed planar graph of wall lines. Seminar presentation paper from the 5th Annual Pacific Northwest Computer Graphics Week, Eugene, Oregon. Photocopy obtained from Architecture and Planning Research Laboratory, College of Architecture and Urban Planning, University of Michigan.

Weisberg, Robert W. 1986. *Creativity: Genius and Other Myths*. New York: W. H. Freeman and Co.