Scott Johnson
University of Michigan
sven@umich.edu

Making Models Architectural: Protean Representations to Fit Architects' Minds

A rich vocabulary has evolved for describing architecture. It serves not only as a means of communication, but also as an embodiment of concepts relating to form, space, structure, function, mood, and symbolism. We architects not only speak in terms of walls, rooms, roofs, arches, etc., we see in terms of them and think in terms of them, as well. Such concepts are integral to our ability to design. Typical CAD representations, however, are based on geometric / mathematical elements like points, lines, planes, and symbols. Even more experimental approaches like parametric shapes or procedural assemblies correspond poorly to architectural elements, and seldom lend themselves well to making conceptual changes that would allow exploration of design alternatives. Small wonder some architecture schools experience a division between computer and studio courses, or even between computer and studio faculty. Different ways of talking and thinking are involved. The concepts involved are often mutually exclusive.

This paper discusses an attempt to address this conceptual mismatch, using what are termed "protean" (meaning "very changeable") elements. These are high-level elements corresponding to architectural concepts like "wall," or "dome." They each have parameters appropriate for the particular type of element they represent, and produce the polyhedra necessary for graphics based on these parameters. A system is being implemented to allow models to be constructed using these elements. The protean elements form a loosely structured model, in which some elements hierarchically contain others, and some elements are essentially freestanding, being created and manipulated independently of other elements. Characteristics of protean element are discussed, including the underlying object-oriented structure, the relationship between elements and graphics, and functions associated with the objects. A scheme is explained whereby all parts of a design can be represented even when the design includes extremely unusual forms not conforming to predictable classes of elements. The necessary support framework is also discussed; general flow of the system and mechanisms for viewing the model and editing subcomponents are explained.

The current status of the project, and intentions for future work are discussed. The project has been partially implemented, and the necessary framework to support the system is mostly complete.

Rendre un modèle architectural: Représentations protéennes pour la pensée de l'architecte

Un riche vocabulaire a été développé pour décrire l'architecture. Il sert non seulement comme moyen de communication mais aussi comme concrétisation de concepts relatifs à la forme, l'espace, la structure, la fonction, et le symbolisme. Nous, les architectes, ne parlons pas seulement en termes de murs, chambres, toits, arches, etc., nous voyons et nous pensons en fonction de ces termes. De tels concepts font partie intégrale de notre capacité de conception. Des représentations DAO typiques, cependant, sont basées sur des éléments géométro-mathémathiques tels que les points, les lignes, les plans et les symboles. Des approches encore plus expérimentales, telles que les forme paramétrisées ou les assemblages procédurales, correspondent peu aux éléments de base de l'architecture, et se prêtent rarement bien aux changements conceptuels qui permettraient l'exploration de designs alternatifs. Il est alors peu surprenant qu'il y ait des écoles d'architecture qui ressentent une division entre les cours sur les ordinateurs et les cours studio, ou même entre les enseignants de ces cours. Il s'agit de différentes manières de parler et de penser. Les concepts dont il s'agit sont souvent mutuellement exclusifs.

Ce papier traite d'un essai d'addresser ces difficultés de communication en utilisant ce qu'on appelle des éléments 'protéens' (c'est-à-dire 'très variables'). Ce sont des éléments de haut niveau correspondant à des concepts comme 'mur' ou 'dôme'. Chaque élément a des paramètres appropriés à l,objet qu'il représente, et produit les polyèdres nécessaires pour des images graphiques basées sur ces paramètres. Un système, nommé Protéus, qui permet la construction de modèles utilisant ces éléments. Les éléments protéens forment un modéle quelque peu structuré, dans lequel certains éléments en contiennent d'autres, et d'autres sont essentiellement seuls, étant crées et manipulés indépendamment des autres éléments. Les caractéristiques de ces éléments protéens sont discutés, y compris la structure orienté-objets sous- jacente, la relation entre les éléments et l'infographie, et des fonctions associées avec les objets. Un schéma est expliqué grâce auquel tous les éléments d'un design peuvent être représentés, complétant le modèle, même lorsque le design contient des formes peu habituelles et non conformes à des classes prévisibles d'éléments. Le cadre de supports est aussi discuté; l'enchaînement général du système et des mécanismes pour voir le modèle et éditer ses composantes sont expliqués.

Le présent état du projet, et les intentions pour de futurs travaux sont discutés. Le projet a été partiellement implanté, et le cadre nécessaire pour soutenir le système est pour la plupart complété.

the language of studio

Like professionals in other areas, architects have developed a jargon — a vocabulary describing concepts they encounter frequently in their area of expertise. A person walking through an architecture studio might hear bits of conversation like, “You might try punching a window through the wall, or raising the ceiling a couple of feet to lighten it up and give it a lighter, freer feeling.” People might mention spring lines, soldier courses, sill plates, or even sexpartite vaulting.

However, a person walking through a computer lab is likely to hear completely different jargon, even if the users are making architectural drawings or modeling buildings. A person modeling on a computer, trying to accomplish the same effect as the one in studio, might need to create a volume to subtract from a cuboid, or translate another group of objects by (0”, 0”, 24”). He has to think about concepts like set operations, symbol instances, selection sets, and snapping modes.

These aren’t cosmetic differences. They represent fundamental differences in the way the world is seen, described, and conceptually manipulated. In order to express ideas from one domain in the terms of the other, a person must translate ideas from the person’s mental representation into a representation that differs significantly, inhibiting the cognitive capabilities of the person doing the work (Norman 1991).

This sort of extra mental processing is to be avoided in interface design. It stands in the way of making the interface transparent. It forces an additional degree of removal between the architect and the task of designing architecture. It interferes with design, rather than facilitating it. In order to truly utilize computers in the studio and in the design process, interference has to be avoided. The CAD program must become more user friendly, which in this case means becoming more architect-friendly. It must communicate with the architect *on* the architects terms and *in* the architect’s terms.

augmenting mental abilities

Certain aspects of mental performance can be thought of as mechanisms or resources, such as attention and working (short-term) memory. The resources and mechanisms have certain intrinsic operating parameters, and mental performance falters when these resources are exceeded or mechanisms are overloaded. If a human mind is given too many disparate pieces of information to work with at one time, it will forget some of that information; a person will forget a piece of information before he can use it, or will lose his place and forget what he had been doing. If a mind is given too many things to attend to, something will be put on the back-burner until attention can be devoted to it, assuming memory hasn’t forgotten about it by then (Hayes 1989:111-128).

When people work in a domain, they develop short-cuts to help them operate in that domain. They augment their memories or simplify their tasks by using tools or external media, writing things down or drawing images (Norman 1991, 19-22). As they become experts in the domain, they start to associate (chunk) pieces of information together, so that the information can be remembered and thought about more efficiently (Hayes 1991 121-126; Best 1989, 133). They develop a jargon to describe the things they encounter in their domain, allowing them to communicate ideas concisely to others in their field (Lakoff 1987, 308). Experts in a domain also start to automatize a variety of mental tasks, from recognition to high-level problem solving strategies, allowing them to perform many tasks with minimal attention (Norman 1991 23-24; Anderson 1982; LaBerge and Samuels 1974).

need to use architectural elements

If a system is meant to aid in architectural design, it needs to augment mental performance without inhibiting it. One way to do this is to provide an external medium for visualization, similar to a sketch pad. However, a CAD system needs to do this without interfering with the use of chunks or automatized processes used in the domain. It should also do it without forcing the user to waste cognitive resources by mentally translating architectural ideas into the language of the visualization software. The software should therefore let architects work with the sort of concepts they already utilize for the associated symbolic, geometric, and functional meanings. Architects should be allowed to work with things like rooms, walls, doors, columns, beams, and so forth.

need for flexibility

In addition to using the *concepts* they are used to working with, architects also need to be able to work with the *processes* they are used to working with. Letting architects work with their automatized mental processes would be reason enough for this, but there is more. Architects develop their libraries of chunks and processes because they are so well suited for designing architecture. Even if architects were completely re-trained and given decades of experience using new "computer-friendly" thought processes for designing architecture, it is likely that these new processes would still not be as well suited for visualization, creative ideation, evaluating symbolic meaning, and other tasks associated with architectural design. CAD systems need to accommodate mental processes used in design, not the other way around.

In order to fit the way architects think while designing, a CAD system needs to allow a top-down method of working. It is well established that design proceeds in a primarily top-down manner, with broad decisions (like where to put a certain room) being made before decisions about details. A CAD system needs to support such decision-making by allowing broad decisions to be modeled before all the component details have been refined.

A CAD system also needs to be flexible. As a design evolves, things change. Rooms may move, appear out of nowhere, or disappear altogether. One structural system may be replaced with another. The second floor may be completely unsupported for a time before the first floor is resolved. Volumes might collide with each other before being squeezed into their proper places. A CAD system needs to be flexible enough to allow drastic changes. It needs to allow topology-altering changes, as well as changes which produce temporary states of structural or geometric impossibility.

current representational approaches

The sorts of representations used in most commercial CAD packages do not allow architects to work in a flexible, top-down manner with familiar elements. Most are based on geometric / mathematical elements like lines, arcs, planes, symbols, or polyhedra. While architects are generally familiar with these concepts, they correspond poorly to elements like walls, rooms, columns, beams, and so forth. Conceptually simple changes, like trying to change the number of columns in a colonnade or even trying to change the width of a door (Johnson 1997:10) can be difficult to perform by altering the characteristics of these geometric / mathematical entities. While these elements are reasonably flexible in the sense that one can be changed or eliminated without changing or invalidating others, the elements are not well suited to top-down design. For instance, an architect might have to put several elements together to make a column, then put several columns together to make a colonnade. Only after assembling the colonnade from the bottom up could the architect then consider whether or not some sort of colonnade separated two spaces in the desired manner.

Parametric shapes have been utilized in more experimental systems to provide a top-down working environment, and even improved correspondence to architectural concepts (Mitchell 1988; Mitchell 1992). The user constructs a model out of shapes, each with a particular topology. By stretching various segments or faces of the various instances of shapes, architectural forms of any desired proportions can be modeled. By allowing a shape to be substituted or augmented with more specific shapes, an impressive degree of top-down model building can be allowed. However, these effects generally come at the expense of flexibility. The user can only model those topologies anticipated by the programmer. Furthermore, the ability to create "freestanding" elements (not derived from elements already in the model) can be limited; a system based too heavily on substitution / subdivision can constrain the options available to the designer.

Other representational systems also have drawbacks. Feature-based systems allow a certain degree of top-down decision making, allowing a user to select forms added earlier, and add or remove "features" like additional forms to be unioned with or subtracted from the selected form. But feature-based systems typically are oriented around Con-

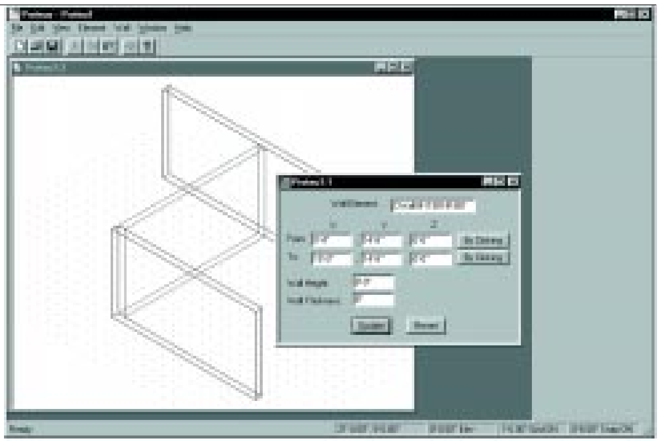

Figure 1. Creating or editing an element.

structive Solid Geometries rather than more architectural elements. Procedural shape systems create shapes or forms based on architectural parameters (like number of risers in a stairway), but don't save this information–low-level geometrical / mathematical elements are created based on the information. Procedural assemblies don't even let the user work graphically–the user has to try to script a series of solid modeling commands that will produce the desired shape. None of these approaches allows the sort of editing needed.

protean elements

general characteristics

Protean elements are an attempt to overcome some of the drawbacks with existing representational approaches, and solve the problem of mismatch between the way architects think of their designs and the way modeling and drawing systems represent them. They follow the guidelines for CAD representations (Johnson 1997). Each protean element corresponds to an architectural element like a wall, room, window, molding, colonnade, dome, etc. Some of these might be nested within others, e.g., a molding might be contained within a window, which might in turn be contained within a wall. Each different type of protean element has attributes appropriate for the type of element it represents, for example, a simple round column might have a radius, height, and location. A description of a straight wall would include a "from point" and a "to point." Wherever possible, overridable default values are assumed so that the architect doesn't have to supply a value for each and every attribute when an element is created. These attributes are stored after each element is created, so that the characteristics can be edited later — graphically, by clicking points in a graphic view, wherever possible. More specific types of elements can be substituted as the design is refined, effectively adding additional attributes to handle more specific characteristics. For instance, a dome with a certain radius and center might later be replaced with a description of a ribbed dome, which has additional attributes to describe the number, placement, and form of ribs.

use in the design process

A test implementation, called "Proteus," is being developed, wherein protean elements are intended to be used during schematic design, when the designer is still making decisions about massing, which rooms will be in the building and where they will be located, what the facades of the building look like, etc. To build a model, the designer creates a set of elements, choosing each from a list of element types. Objects can be added or deleted more or less at will, with only a few restrictions (e.g., windows and doors can only be added in other elements, like walls). When an element is being created, a dialog box appears, as shown in Figure 1, showing the attributes of that type of element, along with the default values of those attributes, if any. The user provides values for the attributes, graphically, if desired. Once the architect is ready, the element is created, using the attributes given.

Existing elements in Proteus can be modified by selecting an element in a graphic view. The dialog box for the element appears, showing the current values of attributes. New values can be supplied, and when the changes are applied, the new appearance of the object is computed and used. If a subcomponent of the selected element needs to be edited, the user selects a menu item and a new graphics window is created for editing the subcomponents. These are edited in exactly the same manner as elements in the document's main view: by selecting them and changing their values.

It is intended that as the design is refined, additional changes may be made. An object of one type may be replaced with an object of another type. A wall might be replaced with a colonnade, or a dome might be replaced with a more specific subtype, like a ribbed dome. When this happens, Proteus copies over any attributes the objects have in common, and replaces the old object with the new one.

description of elements

The behavior described above is being achieved through an object-oriented representation, where each class of programming object represents one type of architectural element. Associated with each class of element are certain data and functions. The data includes parameters describing attributes of the architectural element (e.g., "to point, " "from point, " and other attributes for a wall). A function associated with the element class examines these attributes whenever they change, and generates a set of polyhedra which represent the element graphically. The polyhedra are then stored with the element as data. Other information stored with each element includes pointers indicating any component elements, and a pointer to a screen dialog box (if any) where the element's attribute values are currently shown. Other associated functions perform operations like displaying and retrieving data from the dialog box, or performing operations specific to a particular type of element (like adding a window to a wall). A sample element is shown in Table 1.

Wall Data
- Name
- Class-specific attributes:
  - From point
  - To point
  - Height
  - Thickness
- Class-specific subcomponents:
  - List of openings
  - List of moldings and other features
- Pointer to the current polyhedral representation
- Pointer to a dialog box showing attributes and values

Wall Functions
- Function to display appropriate dialog box
- Function to update menu
- Functions to retrieve / change attribute values
  - Function to return an element's name
- Function to return the wall's height
  - Etc.
- Function to generate polyhedral representation
- Functions to perform class-specific operations
  - Function to add a door to a wall
  - Function to convert a wall to a colonnade
  - Etc.

Table 1. A sample protean element.

The classes of elements are hierarchical in terms of specificity. There are some attributes and functions which are common to all elements, some that are common to all doors, some that are common to only hinged doors, and so forth. For instance, all elements have a name and a pointer to their current polyhedral representation, as well as procedures for retrieving or reassigning these values. All elements also have a procedure to generate a new polyhedral representation, but each element subclass has a different such procedure, so this is implemented at the level of the particular element subclasses, like "wall." Similarly, each element subclass has procedures to display and implement commands specific to that type of element, like an "Add Door," command for walls. Element subclasses also have data fields necessary to define an element of that particular subclass, for instance, from point, to point, height, and thickness for the simple wall implemented so far. Below this level, there may be element sub-subclasses, with additional parameters and procedures appropriate for the type of element represented.

Table 1 does not list certain attributes like R-value, colors of the wall surfaces, construction materials, or other information relating to various analyses. This should not be construed as meaning that such information is incompatible with a protean representation. These additional characteristics are merely omitted from the table because they have not been implemented yet. One plan for future work is to investigate how well protean elements can support such analyses and the data required to perform them.

A full implementation of a system based on protean elements would include perhaps a couple hundred elements, arranged in a hierarchy similar to that shown in Table 2. The list shown is not exhaustive, but should give an indication of the types of elements intended.

unusual elements and completeness

It is, of course, impossible for a hierarchy of high-level, architecturally-oriented elements to be absolutely complete without falling back on some sort of lower-level, geometric / mathematical representation. There will always be a few parts of some architectural designs that are so anomalous that it is impossible to anticipate them in detail. Sooner or later, some architect somewhere will incorporate some bizarre sculpture, some unique detail, or some other previously unimagined element into an architectural design. While such elements comprise a relatively small part of even the most bizarre architectural designs, it is still vital that a modeling representation be able to accommodate them. A modeling system that accommodates only certain designs is of doubtful value as a design aid.

Several classes of elements are needed in order to round out the elements available and make the system complete. These additional elements must be as general-purpose as possible in order to accommodate any conceivable type of form. A combination of geometric primitives, process-oriented descriptions (e.g., solids of revolution), and if all else fails, Constructive Solid Geometries (CSG's) supply the necessary completeness.

Few systems have incorporated both architectural elements and a mechanism for complete, general-purpose modeling. Yet one does not preclude the other. Both architectural and general-purpose elements have characteristics that describe their particular geometry, and it is sensible to make both editable in the same manner. Both can be made to produce polyhedral representations of themselves, for use by the same drawing routines.

The use of general-purpose vs. architectural classes may have some relevance for certain analytical calculations. For example, structural calculations might be affected by whether a certain element is technically a column or a cylinder. This possible relevance has not yet been explored.

elements not included

It can be argued that the list of elements in Table 2 excludes a number of concepts used by architects as they design. It does not include, for instance, concepts / elements like "structural system," "monumental entrance," "circulation," or "HVAC system."

The omission of such elements is deliberate. While the significance of such elements in the thoughts and reasoning of architects can not be denied, this alone does not indicate that such an element should be included as such in a CAD system. The analogy may be made that while elements like "foreshadowing," "comic relief," and "irony" are elements of literature, it is not useful to organize a text editor around such elements. They lack consistent, predictable form; it's not even easy to say with great certainty exactly where such elements begin and end. One would not, for instance, press a key to add an instance of comic relief to a story in the same way that one would press a key to add a letter or paste a word or phrase.

In the same way, there are numerous concepts which can be considered "elements" of architecture, but which lack distinct bounds or predictable characteristics. Different instances of "structural system" or "circulation" share some characteristics, but not enough to fall into what psychologists would call a "basic level category" (Rosch 1978). They do not share a large number of common visual features, or characteristics of use. Often they are composite elements, composed of numerous smaller elements put together. These elements are deliberately omitted from the test implementation.

current status

Proteus is being implemented under Windows 95, on an IBM pentium machine, using Microsoft Visual C++ 4.0 and the Microsoft Foundation Classes object library. Currently, enough has been implemented to test the technical feasibility of the basic approach. No major barriers have been discovered that would disprove the idea that it is possible to base a modeling system on protean elements that produce low-level representations of themselves. The approach seems workable from a technical standpoint.

General support code has been completed. This includes the code necessary for file access, opening and closing windows and dialog boxes,

Wall	**Window**	**Door**	**Column**	**Room**	**Floor**
1 / 2wall	dbl. hg.	doorway	round		
arcade	rect. fixed	pocket	rect.		
colonnade	rect. casemt.	sliding	Doric		
screen	rect. awning	folding	Ionic		
CSG	rect. louver	hinged	Corinthian		
	arched	double	Tuscan		
	round	revolving	Composite		
	octagonal	garage	const. sectn.		
	Palladian	transom	series sectn.		
	custom	w / side window	revolved		
	CSG	CSG	CSG		
Roof	**Ceiling**	**(primitive solids)**	**(primitives)**	**(furnishings)**	**(others)**
flat	flat	cube	line	w.c.	arch
gable	sloped	sphere	arc	tub	beam
hip	beamed	cone	circle	sink	molding
mansard	corbelled	wedge	plane	cabinet	mullion
gambrel	cross-beam	CSG		stove	sill
shed	barrel vault			refr	jamb
helm	groin vault			washer	bracket
barrel	dome			dryer	niche
dome	truss			dishwasher	quoin
CSG	space frame			furnace	pendentive
	CSG			wtr. htr.	staircase
				fireplace	circ. stair
				chair	ramp
				couch	CSG
				table	
				desk	
				plant	
				CSG	

Table 2. Protean elements intended for a full implementation.

setting viewing parameters, entering points graphically by clicking, and other general-purpose code. This portion of the project is necessary, but not particularly interesting from the point of view of researchers.

Proteus also currently includes the framework necessary to work with protean elements: maintaining the list of elements, drawing representations of elements on the screen, selecting elements graphically, and opening dialog boxes and filling them with element data. This code is necessary in order for protean elements to function. Code for these tasks works with modular code for each particular type of element. For instance, general purpose code for Proteus determines that a right-click was made, searches a display list for a line drawn on the screen which passes near the clicked point. If a nearby line is found, each element then checks to see if it produced the line in question. When the correct element is found, Proteus then tells the element to open a dialog box and fill it with the attributes and values describing the particular element selected. This code is modular in nature, and details are different for each subclass of element. Proteus just needs to invoke the function, and each element takes care of particulars for its own case. In a similar way, Proteus draws the polyhedra provided by the elements, displays a menu appropriate for the current element, and opens windows for editing sub-components of complex elements.

Most element-specific code has not yet been implemented. At this point, only CSG's and a rudimentary version of walls have been implemented; implementing these was necessary to make sure that the general code and the framework for protean elements were working correctly. This supporting framework seems to work properly, so work will proceed to the point of implementing other element classes.

discussion

One approach that is sometimes taken for representation of architectural designs is a "kit-of-parts" approach. A kit-of-parts approach involves a large set of pre-defined architectural "elements" or "parts," which are combined in order to define a model. Usually symbols or similar grouping methods are used to define the parts, although parts could conceivably be more like procedural or parametric shapes in nature.

In some respects, protean elements are similar to elements used in a kit-of-parts approach, particularly a more sophisticated, parametric representation. However, there is a fundamental difference in the way models are created in a protean approach. A kit-of-parts is inherently bottom-up in nature. Models are made by combining parts into a larger aggregate.

A protean approach, however, is not necessarily bottom up, though protean elements can be used in that manner if desired. With protean elements, a column could be defined and then used in a colonnade (in a bottom-up manner), but the colonnade could also be created first, leaving the definition of the columns for later. A window could be added before the mullions and moldings are decided on. Protean elements allow design to proceed in a top-down manner, with details being resolved after broader design decisions.

"high level" approach to element definition

Protean elements have "high level" definitions in the sense that they are defined in terms of the attributes of architectural elements like rooms, walls, and colonnades. The graphical entities (polyhedrons and lines) necessary for the graphical representation are derived from the architectural definitions, and updated whenever the "high level" attributes change.

This is in contrast to a "low level" approach, where an attempt would be made to represent high-level architectural attributes by combining low-level graphical entities. A "low level" approach might at first look promising when attempting to add architectural elements on top of an existing drawing or modeling system. However, prior experience has indicated that the "low-level" strategy has inherent problems. It is difficult to represent attributes like "column spacing" directly with lines, polygons, polyhedra, and so forth. Even traditional ways of grouping graphics, like groups or symbols, do not work well for this purpose.

One way to get past this difficulty might be to create additional data structures to hold "architectural" information, and list the component drawing entities. This approach is effectively very similar to the one taken; high level attributes are stored for possible editing later.

Another possible "low level" approach would be to have a model consisting entirely of graphical elements, and to recognize architectural elements from the graphics, similar to the way architects can "read" a model. This approach might potentially be very powerful. It might, for instance, allow a set of independently placed columns to be recognized as a colonnade, without having been expressly created as such. However, this approach assumes recognition capabilities that have not yet been realized. It also assumes that there is a set of architectural elements that the graphics can be recognized *as*. In effect, the strategy requires the ability to recognize elements very similar to protean elements out of graphical ones. The major difference is that such elements would be temporary, being synthesized on demand to allow changes, and being deleted once the changes were implemented in the graphical elements (possibly to be temporarily re-created for further changes). This approach is therefore considered to be difficult to implement, but not entirely incompatible with protean elements.

"no rules" approach to element creation

Shape grammars, which are essentially graphical versions of rule-based systems, are sometimes proposed as a basis for CAD systems. The idea is that the design process consists of a series of changes in the state of the design. Design can supposedly be simulated or facilitated by invoking a series of rules which perform these changes. Each rule has a specific situation in which it can be invoked (usually a configuration of graphical or architectural elements). If the conditions of the rule are met, the rule can be triggered to perform the associated design change. For example, given a column, one can subdivide it into a capital, a shaft, and a base (Mitchell and others 1988, 259). When the conditions for several rules are met, a rule-based system usually has a conflict resolving mechanism, has some sort of random selector, or relies on a human to select a rule.

The Topdown system demonstrated that a system using shape grammars can provide admirable ability to build models in a top-down manner, with the model becoming gradually more detailed as the design is refined. However, the use of shape grammars in a modeling system limits the design evolution to those ideas anticipated by the programmer. At any given point in the evolution of the design, a particular change can only be made if there is a rule for transforming some part of the model in the desired manner. Each possible change requires a corresponding rule, and since the number of rules in a system is necessarily finite, the flexibility of a rule-based system is necessarily limited.

A rule-based system could get around this limitation by having rules along the lines of, "In any situation, a wall can be added." Since this effectively eliminates the conditions of the rule, however, this really would represent a degenerate case of a rule system. There would be little reason to implement such a rule system, as opposed to a system that let the user add elements without filtering them through ineffectual rules.

The protean approach is intended to allow gradual definition and refinement of a design, without reliance on generative rules. The choice of what changes to make is left entirely to the architect.

"correctness" of models

Many modeling representations (e.g. ARCH:PLAN (Turner and Hall 1985), EDM (Eastman 1992), or P3 (Khemlani and others 1997) rely on a restriction that models be "correct" or "well formed." This means that the model must conform to certain strict standards regarding topology. For example, some systems require that every room be surrounded by a wall at every point of its perimeter, and every wall must have a "left room" and a "right room." Such requirements prevent "nonsensical" configurations of elements like overlapping solids, and facilitate analytical calculations by prohibiting problems like floors receiving more than one set of loads.

Correctness typically carries a heavy price in terms of how easily the model can be manipulated. Often, special, unintuitive steps have to be taken in order to get a design idea to conform to the correct representational topology. For instance, in some systems, all rooms must have walls along their entire perimeters. If, for instance, the breakfast nook and the kitchen are not separated by an actual wall, the designer must create an "invisible" or "zero-thickness" wall in the modeling system in order to separate them.

If our goal is to have modeling system elements correspond to architectural ones, and we have to invent "invisible walls" to make the modeling system work, we are falling short of our goal.

Protean elements do not assume or require that a model be "well-formed" or "correct." It does not require a strict relationship between rooms and walls, it does not require elements to be supported structurally, and it does not even prohibit solid objects that intersect. This is intentional.

Ideas for well-formed designs to not spring spontaneously into a designer's head. Design is an ongoing process, and initial ideas are often conflicting or mutually incompatible. This does not mean that these ideas should not be sketched or modeled. On the contrary, it is by modeling these ideas that they are mentally evaluated by the designer, their faults are seen, and various attempts at rectifying them can be proposed. Eventually the various problems and geometrically impossible arrangements are ironed out.

Protean elements are intended to model *evolving* designs. These means representing things which are ill-formed. It is true that ill-formed models do not facilitate structural, thermal, lighting, or probably any other sort of numerical analysis. However, there is little need to perform such analyses on models that are ill-formed. For instance, one does not need to perform a complicated structural analysis to know that a roof with nothing under it is unstable. The model can be corrected and the analyses performed later, once the design has reached a state where such an analysis would be meaningful.

future work

The next step in the project is implementing a greater selection of elements. Probably a great deal of functionality can be achieved by implementing just a few classes of architectural elements: walls, rooms, floors, ceilings, moldings, beams, columns, colonnades, doors, windows, and a few types of roofs. Process-oriented elements, like solids of revolution or extruded forms need to be implemented. Graphical elements like lines, planes, and circles should also be implemented, for use as graphical guides and so forth.

Proteus is intended as a test implementation; testing of certain aspects of protean elements is stressed over finishing a complete system. Given the limited resources available (a single programmer), only a couple dozen elements are likely to be implemented. Most likely, these will include the most basic architectural elements (walls, rooms, beams, and so on).

One class of element will be chosen for further exploration. Specific subtypes of this element will be implemented to explore the number of subtypes needed and whether implementing subtypes becomes easier or harder than the basic-level classes.

Eventual plans are to investigate other aspects of protean elements. Can a protean model be elaborated to adequately represent buildings at the level of studs, sill plates, and vapor barriers? How easily could analyses like structural or lighting calculations be incorporated into protean elements? The answers to these questions are unclear at the moment.

conclusion

Protean elements are proposed as a computer representation intended to fit the concepts and processes used in architectural design better than existing representational approaches. They emphasize correspondence to architectural elements and flexibility of the model and of the model creation process. The intent is to provide an environment for architectural design which is conducive to early schematic design and the sort of uninhibited design exploration that takes place therein. By helping computer software to better fit the way architects think, it is hoped that computers can be more successfully brought out of the computer lab or "CAD rooms" and into the design studio. Proteus is an attempt to implement a system based on these elements.

Initial results are promising. Protean elements allow characteristics of architecturally-oriented elements to be changed, without sacrificing model completeness. Highly anomalous forms can still be created. Protean elements can generate a polyhedral representation to allow the necessary graphics, without sacrificing performance like being able to select elements by clicking. Although the naturalness with which Proteus fits the design process can not be evaluated at the current stage of the implementation, the approach conforms to current information on cognition and mental representation. The protean elements approach appears potentially very useful. Additional aspects of the approach will be explored in the future.

references

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

Anderson, John R., 1985. "The Development of Expertise," Chapter 9 of *Cognitive Psychology and its implications*, 2nd ed. New York, NY: W. H. Freeman & Company.

Anderson, John R., 1982. "Acquisition of Cognitive Skill," *Psychological Review*, 89, no. 4, p. 369-406.

Best, John B., 1989. *Cognitive Psychology*, 2nd ed. Saint Paul, MN: West Publishing Company.

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

Eastman, Charles M., 1992. "Use of Data Modeling in the Conceptual Structuring of Design Problems," in Gerhard N. Schmitt (ed), *CAAD*

Futures '91: Computer aided architectural design futures: Education, research, applications. Braunschweig / Wiesbaden, GR: Friedr. Vieweg & Sohn Verlagsgesellshaft mbH., p. 225-243.

Hayes, John R., 1989. *The Complete Problem Solver*, 2nd ed. Hillsdale, NJ: Lawrence Erlbaum Associates.

Johnson, Scott E., 1997. "What's in a Representation, Why Do We Care, and What Does It Mean? Examining Evidence from Psychology". In J. Peter Jordan, B. Mehnert, and A. Harfmann (eds.), *ACADIA'97: Representation and Design*. The Association for Computer-Aided Design in Architecture, Cincinnati, Ohio, p. 5-15.

Khemlani, Lachmi, Anne Timerman, Beatrice Benne, and Yehuda E. Kalay, 1997. "Semantically Rich Building Representation," in J. Peter Jordan, B. Mehnert, and A. Harfmann (eds), *ACADIA'97: Representation and Design*. The Association for Computer-Aided Design in Architecture, p. 207-227.

LaBerge, David, and S. Jay Samuels, 1974. "Toward a Theory of Automatic Information Processing in Reading." *Cognitive Psychology* 6, no. 2, p. 293-323.

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

Mitchell, William J., Robin S. Liggett, Spiro N. Pollalis, and Milton Tan, 1992. "Integrating Shape Grammars and Design Analysis," in Gerhard N. Schmitt (ed.) *CAAD Futures '91: Computer-Aided Architectural Design Futures: Education, Research, Applications*. Braunschweig / Wiesbaden, GR: Friedr. Vieweg & Sohn Verlagsgesellshaft mbH, pp. 17-32.

Mitchell, William J., Robin S. Liggett, and Milton Tan, 1988. "The Topdown System and Its Use in Teaching: An Exploration of Structured, Knowledge-Based Design". In Pamela J. Bancroft (ed.) *Computing in Design Education: ACADIA'88 Workshop Proceedings*. The Association for Computer-Aided Design in Architecture, p. 251-262.

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

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

Rosch, Eleanor, 1978. "Principles of Categorization," in Eleanor Rosch and Barbara B. Lloyd (eds), *Cognition and Categorization*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Rowe, Peter G., 1987. *Design Thinking*. Cambridge, MA: The MIT Press.

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

Schön, Donald A., 1988. "Designing: Rules, Types, and Worlds" *Design Studies* 9, no. 3, p. 181-190.

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.