Rule-Based Representation of Design in Architectural Practice

Thomas Seebohm
University of Waterloo
tseebohm@uwaterloo.ca

William Wallace
Software Metrics
BillW@metrics.com

It is suggested that expert systems storing the design knowledge of particular offices in terms of stylistic and construction practice provide a means to take considerably more advantage of information technology than currently. The form of the knowledge stored by such expert systems is a building representation in the form of rules stating how components are placed in three-dimensional space relative to each other. By describing how Frank Lloyd Wright designed his Usonian houses it is demonstrated that the proposed approach is very much in the spirit of distinguished architectural practice. To illustrate this idea, a system for assembling three-dimensional architectural details is presented with particular emphasis on the nature of the rules and the form of the building components created by the rules to assemble typical details. The nature of the rules, which are a three-dimensional adaptation of Searle's shape grammars, is described. In particular, it is shown how the rules themselves are structured into different classes, what the nature of these classes is and how specific rules can be obtained from more general rules. The rules embody a firm's collective design experience in detailing. As a conclusion, an overview is given of architectural practice using rule-based representations.
Introduction

Whereas architectural practices using information technology initially mimicked traditional practice by being centered on paper-based representations, future architectural practices supported by information technology will likely see designs represented as a data base of geometric and text information. Paper-based images will be only one of many types of information extractable from the data base but they will not be a primary means of documentation. In this scenario, increasing emphasis will be placed on the structure of the design representation in the data base. Design then involves manipulating the data base from an initial state (e.g. site information, client requirements) to a final state under the architect’s control in the light of feedback from visual simulations and various analyses continuously performed on the data (e.g. cost, lighting, acoustic, energy and structural simulations). This approach would take advantage of the great strength of information technology to manipulate vast quantities of data and to rapidly transform data in any manner conceivable. In the following we concentrate on the geometric component of the representation.

To ease the task of the designer, it is necessary to minimize the time to enter the geometric data into the data base in the first place. Entering designs into the current generation of CAD systems is still very time-consuming, particularly, if the representation is a three-dimensional one (It is only with three-dimensional representations that there is any possibility of performing any analyses and visual simulations). Obviously, reusing data previously entered for other projects could save substantial time. Thus traditional CAD permits the reuse of parts of previous drawings or three-dimensional models and provides the creation of libraries of standard symbols and details. Unfortunately, every building is sufficiently different from every other building to preclude using parts or details from previous projects without requiring substantial, time-consuming changes. Case-based systems (systems based on adapting precedent (e.g. Hovestadt 1993, Flemming and Woodbury 1995)) and parametric representations (e.g. window objects in ArchiCAD (ArchiCAD 1997); where the dimensions of reusable parts can be adjusted to the situation at hand, are only useful within very clearly defined building types. In general, the changes from one building to the next involve topological changes involving different combinations of building materials and components. The possible permutations and combinations that could be encountered are too vast to cover with standard details and reusable building parts.

The solution to this dilemma is to store the design experience of projects, not at the level of the actual design, but at the level of the principles behind the details and the building parts. That is, one should store design knowledge in terms of how the components in details and parts of buildings are actually assembled in three-dimensional space in relation to other components. In this way it is possible to store the stylistic design principles and construction experience of particular architectural firms. It is possible to store such design knowledge in expert systems and thereby to take considerably more advantage of the benefits of information technology than is currently the case.

To illustrate this idea, a 3D Modeling Assembler for assembling three-dimensional architectural details is presented with particular emphasis on the nature of the rules and the form of the building components created by the rules to assemble typical details. The set of rules, which are fired to assemble a particular detail are, in effect, a representation of the detail they caused to be assembled (Different constraints on the shape and material specifications for a particular detail lead to different designs and hence different sets of rules). The nature of the rules, which are a three-dimensional adaptation of Stiny’s shape grammars, is described (Stiny
In particular, it is shown how the rules themselves are structured into different classes, what the nature of these classes is and how specific rules can be obtained from more general rules. The rules embody a firm's collective design experience in detailing. Although the 3d Modeling Assembler was conceived for detailing, it could, in principle, be extended to cover the design of complete buildings (We will revisit this topic at the end).

To our knowledge, the present software is the first operational prototype for assembling three-dimensional architectural details. A subsystem for designing three-dimensional building enclosures of a Software Environment to Support Early Phases in Building Design (SEEP) is expected to have some of the capabilities of the 3d Modeling Assembler (Fleming and Woodbury 1993). A two-dimensional precedent is the FAVE software (Mitchell and Hadano 1987). The main impetus for the present approach was the object-oriented strategy in Woodbury's Ph.D. thesis (Woodbury 1988). A more complete overview of the origins of the current prototype was previously written by the first author (Seebohm 1993).

At this point the reader might be apprehensive that we are proposing an approach to architectural practice which will restrictive creativity and good design. Hence, before describing the 3d Modeling Assembler and how it might be used in practice, consider a traditional practice that produced designs of the highest caliber and used principles that anticipate the information age with an approach to practice very suitable for support by expert systems. We are referring to Frank Lloyd Wright's practice as it relates to his Usonian Houses. In many ways Frank Lloyd Wright is a most unlikely model to propose for the information age because of his emphasis on expensive craftsmanship and his insistence on designing every aspect down to the smallest details and the furnishings. There was a moment, however, in the mid 1930's when Frank Lloyd Wright recognized the need to design houses for families of moderate means. The result was the Usonian house (Usonia was the reformed American Society that Wright envisaged). The first Usonian was the Jacobs House of 1936. It had 500 square feet and cost $5500 (Sergeant 1976). The Jacobs House, identified by Sergeant as the polliwhig type, was to be the first of five Usonian house types designed by Wright (The others were the diagonal, in-line, hexagonal and raised or two-story), each having a consistent set of relationships between the various living spaces and using a consistent set of details, a kit-of-parts, or "grammac," as Wright called it. The parts consisted of a slab-on-grade with embedded heating coils for radiant heating (there were no basements), board and batten walls consisting of a 7/8 inch plywood core with specially-milled board and battens on either side, often incorporating bookshelves on the inside for structural stability, and a multi-tiered system of roof framing for flat or slightly sloping roofs consisting of 2x4's supported (when the apprentices managed to sneak them in) by steel girders in some Usonians. In later years Frank Lloyd Wright added a concrete block wall system thereby building on his experience with the textile block houses in Los Angeles of the 1930s. Details for these houses were provided on a standard detail sheet that remained the same for each house. Each Usonian house therefore had the family resemblance of its type, yet within the vocabulary of the type, Wright was able to develop designs of great variability, unique to each site and client. Without designing all aspects of each house from the beginning, Wright was able to build on his experience from one commission to the next. To facilitate dimensioning Wright used dimensioning modules both in plan and in section. In plan he used a two foot by four foot module (in view of the four foot by eight foot dimension of plywood sheets) and in section he used one foot plus one inch.
to coordinate with the board and batten joints and the mortar joints of exposed brickwork.

Wright’s detailing practices for these house lend themselves to the use of the proposed type of detailing software. The detailing knowledge for all the house types would be stored in the rule base and updated to reflect Wright’s evolving design experience and predispositions. With such a detailing system an architectural practice need not reinvent the details for every project but can, instead, concentrate its energy and budget on the overall design and on gradually refining its details (even, occasionally inventing new details) by updating the rules in the database.

**a three-dimensional modeling assembler overview of capability and features**

The 3D Modeling Assembler is written in Common LISP allowing us to take advantage of object-oriented concepts. Objects are like the blocks or objects of CAD systems. They can store information about themselves in named “slots” and can have operations (methods) performed on them. Objects can be arranged in hierarchical classes. They can inherit slots and their values from superclasses (classes higher in the hierarchy) as well as methods. Object-oriented concepts are reflected in how the software operates and in how it is used. Currently the software runs on a Power Macintosh but we are in the process of porting the software to PCs where we will use Allegro Common LISP and a Java-based user interface.

The 3D Modeling Assembler can assemble three-dimensional models of construction details showing all of the construction components (and conceivably, given a sufficiently large rule set, complete building models). It was developed to help create, with relatively little effort, 3D animations showing how construction details should be assembled to avoid building failures due to contractors not understanding the details as presented in conventional twodimensional representations, particularly, how they come together at corners.

The details assembled by the software can be constrained so that they will fit into a specified geometric shape, called a geometric envelope (currently this is simply a box whose dimensions and the position of the detail within it, can be varied). The components used to assemble a detail reflect actual construction components whose dimensions and other attributes such as color, are taken from a linked building product database. The dimensions of these building components can be specified completely, as in the case of bricks, or they can be left free to be determined by the available space, as in the case of drywall, where two dimensions are free, or, as in the case of concrete, where all dimensions can be left free. This ability to use components with variable dimensions determined by the context is a major feature of the 3D Modeling Assembler. Moreover, inserted components have the ability to push or compress other component objects with variable dimensions to make room for themselves. These features, to our knowledge, have not previously been attempted. (This differs from parametric rule systems in the sense that the parameters in our case are not predetermined but set on the basis of the context at the time that components are inserted.) The interdependence of objects is made possible by the use of KR (Constraint-Based Knowledge Representation System) developed at Carnegie Mellon University by Brad Myers et. al. as part of the Garnet Project (Myers 1993).

Currently the only shapes that we can install in a model are rectangular boxes. This covers the vast majority of shapes. Ultimately, we plan to include non-rectangular shapes. The software already has the framework for a simple CSG (Constructive Solid Geometry) implementation.

The rules for specifying how components are inserted relative to each other into the model are built up from simpler generic rules.
by several means. One of these is object inheritance whereby rules inherit features from other rules. Another approach is based on the fact that the rules can be set up in a hierarchical manner by associating sub-rules with a given rule which in turn are executed recursively until there are no more sub-rules to execute. A third method is to use "before" and "after" rules which are executed immediately before and after a given rule. This ability to construct rules from simpler rules means that the task of creating the many hundreds of rules required in realistic building models is greatly diminished. In practice it turns out that there are only very few basic generic rules. These generic rules are a distillation of the essence of the kinds of insertion operations encountered in a 3D construction detail.

The task of creating rules from other rules is accomplished by the "rule builder" component of the expert system. It provides a special dialogue interface for a user to create new rules which inherit characteristics from existing rules by copying or modifying rules. Among the rules changes one might wish to make, is the creation of a rule to insert a new building component. In this case the "rule builder" operates in conjunction with the building product data base. Knowing the type of use of a building product, an existing rule for that type of use is adapted to create a rule for the insertion of the selected building product. The features for creating rules from other rules is another area in which the 3D modeling assembler represents a significant advance over previous work in this area such as the generation of fire stations (Woodbury and Griffith 1993) or the generation of Queen Anne houses (Woodbury 1993). Indeed, we believe we have the beginnings of a language for rule creation and hence for storing design experience. It is a new way for interfacing with a new generation of 3D modeling systems.

It turns out that each part of a detail or a building uses rules which do not apply in other parts. For example, rules for laying bricks are quite different from inserting the different layers of a roof membrane. Rules for installing roof joists are different again (although, as will be shown, the generic rules used to build brick rules and joist rules have some commonalities). Consequently, the expert system does not have to search through all the rules but only those rules applicable to a particular area of a detail, when trying to insert a component. The local character of the rules is accommodated by the hierarchical nature of the rules. Thus, by means of sub-rules, one is able to group all those rules together which are applicable to a particular area of a design and to exclude others. Furthermore, when the rules are stored in the rule data base, there is also recognition of the local character of the rules: Every rule is associated with a group. Thus all brick rules are part of the brick group, which is stored as one file in the rule data base.

The final output from the 3D Modeling Assembler is a conventional, three-dimensional model which can be displayed as a wire frame model in a 3D window or it can be exported in various formats including the GDS THINGS format, DXF format and VGM for viewing with a Web browser. The advantage of the latter is that component colors can be exported. In the future we plan to export animations of the assembly sequence as VGM frames because the sequence of rule execution is intended to reflect the sequence of actual construction.

To explain the nature of the rules and the nature of the real (and virtual) building components used by the 3D Modeling Assembler, consider in the next section, how a parapet detail is partly assembled.

**object and rule structure**

**a parapet example**

For clarity, consider, in Figure 1, a two-dimensional section through a schematic parapet detail showing the major groups of building components that are to be inserted to create the model of the detail. The constraining gro-
metric envelope will limit the extent of the roof, wall and the height of the parapet in the model to be assembled. In Figure 2 we show two-dimensional section views through the constraining geometric envelope representing various stages as real and virtual objects are inserted.

The process begins in Figure 2 (a) with the insertion of a geometric envelope. Next a structural wall envelope, SB-R, is inserted by an envelope rule at a specified starting point. The initial dimensions of this envelope are set to a minimal dimension (e.g. mm). By setting the starting point a specified distance form the x+ boundary of the geometric envelope, we are, in effect, limiting the thickness of the wall. Any rules causing wall constructions which overlap the geometric envelope will be rejected.

Envelopes are virtual objects in that they have no physical counterpart in the physical building detail. Their purpose in the assembly process is to constrain the insertion of components inside them so that they do not extend beyond the boundaries of the envelope. Envelopes may be given fixed dimensions or the may have their faces specified to expand or stretch in the directions of the principal axes. Faces which have neither stretch nor expand specified remain fixed in their initial position unless specified otherwise. Stretching means that the envelope will stretch in the specified direction to accommodate the insertion of

---

Figure 3. Stages in the assembly of a 3D detail.
building components inside it. Expand means that the envelope will expand in the specified direction until it reaches the geometric envelope. Envelopes check when building components are inserted inside them so that there is no overlapping of components. If overlapping is detected, the insertion is rejected causing the software to backtrack and try another rule to insert another component. Envelopes correspond at a physical level with groupings of building components that can be considered independently of other groupings. For example, the structural wall is considered one group. All wall components including structure, insulation and sheathing and vapor barriers would be included in this group.

Figure 2 (g) shows how the SB-R envelope expands in the -y direction to meet the y-face of the geometric envelope. Figure 2 (g) shows how envelope, SB-R, stretches in the -x direction when layer-like drywall and concrete components of specified depths are inserted. A sub-rule of the rule that inserted SB-R, prepares the envelope for the insertion of the layer-like drywall and concrete components so that, when inserted, they are oriented to stretch in -x. Next, in Figure 2 (h), another structural wall envelope, SB-E, is inserted at the starting point, causing both envelopes to overlap at the corner. A join rule is then applied specifying how the layer-like concrete and drywall components are to meet at the corner. Figure 2 (i) a) and b) show two possible corner joints for inside corners. (Note, although the concrete should be continuous around the corner, we are ignoring this point until we can deal with L-shaped objects. In any case, the joint will not be visible in a rendered solid model.) It is only after the join type and the layer orientations have been specified, that the envelopes are actually filled with the drywall and concrete components.

Next, in Figure 2 (i), another envelope rule is invoked causing a structural parapet envelope, SP-B, to be inserted such that its vertices on the
lower right link to those of the SB-R envelope below. Envelope SP-R is allowed to expand until it meets the top of the geometric envelope and it is allowed to expand in the x direction to accommodate a layer-like concrete component of a specified depth in Figure 2 (7) and (8). A three-dimensional view of the detail as it has been developed so far (assuming envelope SP-F has also been inserted) is shown in Figure 2 (9).

A roof envelope, ROOF, is then inserted such that the lower vertex of its inside corner links to an inside corner vertex of the structural parapet, SP-R, because the structural parapet determines how much space is left on the structural wall for the roof to rest on. Figures 2 (9b) to (13) show the ROOF insertion. The insulation rain-screen may now be inserted. Again this consists of a right and a front envelope. A prerequisite for the rules inserting these components is that the structural wall and parapet have been previously inserted because the insulation rain-screen envelopes link to the vertices of the prerequisite envelopes as shown in Figure 2 (14). The IR-B and IR-F envelopes are allowed to stretch to the exterior in the -x and -y directions, respectively. The two envelopes are then joined by an exterior join rule and filled with the layer-like components comprising the insulation rain-screen. A rendered view of the parapet detail after the remaining components have been inserted is shown in Figure 3.

If the cladding rule is changed to a brick laying rule, we obtain the detail shown in Figure 4. Adding a rule for a soldier course produces the detail in Figure 5. Changing the brick to concrete block produces the detail in Figure 6. The brick cladding rule involves a set of sub-rules which treat rows of bricks as horizontal layers that are inserted in the -z direction beginning at the top of the envelopes. Within each layer bricks and mortar joints are treated as layers inserted in the -z direction with every second layer starting with a half brick offset.

Figure 4: Parapet detail with brick cladding rendered in GDS.

Figure 5: Parapet detail with soldier course rendered in GDS.

Figure 6: Parapet detail with concrete block cladding rendered in GDS.
Rule-Based Representation of Design in Architectural Practice

vertices to previously inserted envelopes. In this case there is no need to join corners because the caps are discreet components. Note that the envelopes beneath the right and front caps, SP-R, SF-F, IR-F, and IR-R were allowed to expand to the geometric envelope. Thus, at first glance, it would appear that there is no room to insert the caps. What actually happens is that insertion of the caps causes the abutting envelopes below the cap IR-R, IR-F, SF-F, and SP-R, to contract to make room for the caps inside the geometric envelope if there is room. Currently, this will only happen if the caps have in their list of "after" rules some contract rules which specify which envelopes may cause what other envelopes to contract.

rule objects, syntax and evaluation

Whereas the physical objects inserted by the 3d Modeling Assembler are represented as CLOS (Common LISP Object System) objects, the rules are KR objects. These use a slightly different syntax from CLOS for object creation and related operations such as the creation of methods, for extracting information from slots and for placing information in slots. Instead of classes of objects, KR uses prototype instances of objects.

To illustrate the syntax used in the creation of rules we present two rules, the ROOT envelope rule for the parapet and the SB-R envelope rule. The format of the ROOT rule is as follows:

\begin{verbatim}
(root-rule parapet-group root-envelope-rule
  (add-items 'null)
  (sub-items 'null)
  (scale '(0 0 0) (0 0 0))
  (sub-rules '((sb-r sb-r sb-f sp-f sp-f sb-r sp-r sp-r sb-f sb-f sb-r sp-r sp-f sb-r sp-r)
    (root (cap-f (cap-r)))
    (save-name (root))
    (executions 1)
    (editable t))
)
\end{verbatim}

The above lines are actually the arguments for a function "register-rule" to create a rule object. The ROOT, whose prototype rule is an ENVELOPE-RULE, which is to be placed in the PARAPET-GROUP of rules. ADD-ITEMS, SUB-ITEMS etc. are slots of the rule which (except for the first two) determine the action that this rule causes when it is evaluated (executed). The first two slots can be used to cause more or fewer slots and their values to be displayed. The SCALE slot determines the x, y and z dimensions of the geometric envelope. (ignore the period). The SUB-RULES slot lists the sub-rules which will be run by this slot. The EXECUTIONS slot determines how many times this rule may be executed.

The format for one of the sub-rules, SB-R, is as follows:

\begin{verbatim}
(root-rule parapet-group sb-r envelope-rule
  (add-items 'null)
  (sub-items 'null)
  (scale '(0.0 0.0 0.0) (1.0 1.0 1.0))
  (sub-rules '((layer=layer-group))
    (save-name (sb-r)
    (vertex= (y 0.0) (depend 0 15) (y 0.0) (depend 0 15))
    (transform (transform (t-right))
      (executions 1)
      (require '(sb-f)
        (expand '(void) (void) expand expand-stretch) void)
  (editable t))
)
\end{verbatim}

Here the SCALE slot is set to a nominal small value because the size of this envelope will be automatically set. There is one sub-rule LAYER-Y which prepares the envelope for the insertion of layers whose depth is oriented in the Y+ direction. The VERTEX- slot determines the linking of vertices. The values in this slot specify that vertex, V4, of the new envelope will be linked to vertex, V3, of object, o, that is, SB-F, the first in the list of required objects in the required slot (REQUIRED). Vertex, V3, will be linked to vertex, V5, of the required object. The
TRANSFORM slot specifies the transformation to be applied to the envelope before it is inserted (in this case 90 degrees about the Z axis, using the right hand rule). The EXPAND slot specifies which surface of the envelope to be inserted shall stretch or expand. The convention is that the list is in the order X, -Y, -Z, X, Y, Z. Thus we have expansion on the -Z face and on the X face and stretching on the Y face (and the orientations are those obtained after applying the transformation). Stretching and expanding are mutually exclusive.

**relationship to shape grammars**

Shape grammars are rules of spatial relationships which cause the shape(s) on the left side of the rule to be replaced by one or more shapes on the right hand side. Sometimes the shape(s) on the left have one or more markers to specify the orientation of the shape(s) on the left with respect to those on the right as in the case of Stiny’s kindergarten grammars (Stiny 1980b). There may also be other symbols which determine when certain rules may be applied or not.

As noted earlier, the rules which have been described so far are loosely based on shape grammars. As such they can be used to generate the designs for Froebel blocks in Stiny’s paper (Stiny 1980b, as shown in Figure 7). The rules also differ in several ways. However, as others have pointed out, when it comes to implementing shape grammars in practice, they have several deficiencies having to do with structuring complexity and with controlling the sequencing of execution, among others (Carlson 1993).

We list some of the major differences from shape grammars:

- Spatial relationships are specified in terms of linked or offset vertices thus avoiding the need for markers
- The left hand side of shape grammars corresponds to the objects listed in the required slot. All envelope shapes are named objects. Thus searching for matching shapes in the current shape is not required. Other predicates besides the existence or non-existence of objects are possible
- The rules are hierarchical in that they may have sub-rules and are grouped in groups
- Rules can inherit behaviors from prototype rules
- There is no checking for the orientation of previously inserted objects. Named objects occur in only one orientation whereas in shape grammars they can be in any orientation. One can deal with non-standard orientations by inserting components in transformed envelopes.
---the function of symbols for controlling
execution in shape grammars is supplant
ed by other control mechanisms (e.g. collect rules run a
set of sub-rules repeated, the EXECUTIONS
slot determines how many times a rule may be
run and alternate rules are run as alternatives to
other alternate rules which fail to run). To date
we have not explored the sensitivity of these
mechanisms on the type and number of alter-
mate details generated.

using the 3D modeling assembler

So far we have shown, with the parapet
example, a case where there are just the right
number of rules to accomplish the task of
assembling a desired detail. The rules also hap-
pen to be arranged in the order of execution but
that is not necessary because the Modeling
Assembler will search for the applicable rules
by cycling through them). Normally, one would
expect to have many alternate rules for a partic-
ular detail. Most likely, list of sub-rules of the
primary or root rule would then be much larger
and each sub-rule, in turn, would have a large
list of sub-rules. Whatever the number of rules,
however, they would be carefully arranged in a
hierarchical structure of rules and sub-rules.
Furthermore, the rules groups to which the
rules belong would be considerably enlarged.
In practice, this larger set of rules would be
gradually built up from the initial set that is
delivered with the software. The enlarged set of
rules would reflect alternate design approaches
both at a technical and at a stylistic level. The
set would also reflect different structural sys-
tems for the structural components. For exam-
ple, the roof might have a system of wood or
metal joists supporting it, the structural wall and
parapet could be built of wood or metal studs
with particular corner details and the structural
wall could be built with wood studs or rein-
forced concrete block. (Note that repetitive com-
ponents such as wood studs can be inserted by
rule sets similar to those used for bricks).

To create the rule set for a new variation or
a new type of detail one follows the sequence
in which the detail would be physically con-
structed. As each component is added, a rule is
written or adapted that would place the com-
ponent in the correct spatial relationship to pre-
viously inserted components. The spatial rela-
tionship is specified by linking vertices of approp-
riate envelopes. In particular, one must ascer-
tain that the vertices of the containing envelope
are linked to the vertices of those envelopes
which actually determine the dimensions of the
inserted component. In other words, one must
think through the basis on which a designer
would determine his dimensions. In a con-
struction detail these bases are mostly func-
tional but sometimes, as in the width of a fascia
board, there may be a stylistic principle
involved. As long as the stylistic principle is
geometric in character, we know from the suc-
cessful attempts at capturing elements of style
with shape grammars and other rule systems
that this is possible (e.g. König and Eisenberg

Now suppose that we have captured a fairly
large body of rules reflecting the construction
and stylistic practices of an architectural office.
How would we go about using the 3D Modeling
Assembler? There are four main steps: First, we
begin by selecting the rule group correspond-
ing to the type of detail to be constructed and
by loading the selected rules (this compiles the
selected LISP rules). The rule groups are stored
in directories (or folders), classified by the type of
building structure (e.g. concrete, wood stud,
steel frame etc.), then by type of detail having a
default insulation rain-screen such as metal
cladding where this applies. Loading brings up
a dialogue window with a list of the rules in the
selected rule group (e.g. the concrete parapet
group) as in Figure 8. Second, with the aid of the
"rule builder" which presents a dialogue win-
don showing the slots and values for any select-
ed rule, the user can then change the values of
the various slots to edit the rules. One can for example, change the initial rule which inserts the geometric envelope to change the size of the geometric envelope and the location of the starting position within it (ultimately, we expect a user to be able to cause the initial rule to use shapes other than rectangular envelopes). In addition, one may want to change the cladding from metal to brick by changing the sub-rules for the insulation rain-screen which insert the cladding layers so that they insert brick in a selected bonding pattern. Changing the default bricks to a different brick is done by using the building product data base to select a different brick and then asking it to create a rule to insert the new brick. Figure 9 shows the main data card of the Hypercard-like interface for selecting building products and the data card for a chosen building product from which the creation of a rule to insert the product is initiated. Third, when all the rules are as desired, the initial, or ROOT rule, of the set is selected and “run”. Four, when the rules have all been run a 3D projection may be viewed as wire frame in a 3D window or it may be viewed with a Web browser having a VRML plug-in.

One way to use the 3D Assembly Modular is to simply allow it to create an initial detail and then react to it by making changes. Thus a novice can learn how to design a detail by knowing little about a particular detail to begin with. Then seeing the default detail she can ask for changes more suited to her needs by changing the building materials with the aid of the “building product data base”. She may adjust the geometric envelope to create higher or lower parapets but may also, if the changes to the geometric envelope are significant enough, initiate a design with a completely different topology as in the case of the low-profile parapet detail in Figure 10. Another way to use the Modeling Assembler is to let it assemble a number of possible alternatives by allowing it to execute all the alternate rules (rules which are alternatives to others) in the rule group for a particular detail.

rule-based modeling in practice

Having seen how an expert system for detailing can be used in practice, consider now how future generations of such systems, when applied to detailing and even to whole buildings, might be used. We have already shown how the detailing practices which Frank Lloyd Wright used for his Usonian Houses lend themselves to the type of detailing software we have described. While the proposed approach to detailing can improve design standards and reduce the cost of design, it says nothing about the major issue of entering complete building representations into a design data base. Here again Wright’s Usonian design practices offer a clue. Just as there are underlying design principles or rules for assembling details so the overall designs for the Usionis also follow rules beginning (with lowest level of detail) at the level of massing with layouts reflecting Wright’s consistent topological relationships between the major living spaces (Knight 1954). Thus Wright used a grammar that operated not only at the level of details but of the entire house. Thus, as much as the consistent detailing, contributed to the family resemblances of the houses. It is conceivable that an expert system can, given enough
time, generate all the possible Usonian designs of a certain type (not taking into account that a real designer might always make up some new rules). Although it has already been shown (Knight 1995) that the number of basic massing alternatives (for polliwogs) is comprehensible, the total number of detailed design alternatives is likely to defy the imagination. Thus there is no question of automating design because the difficulty, as with all large design spaces, is how to cull the applicable and promising designs.

Instead of attempting to cull a large design space, we envisage a designer making preliminary design sketches and then coaxing an expert system based on Usonian or other rules to incrementally lay out a design corresponding to the sketch in effect executing rules that gradually build up the desired layout by suitable constraints such as a local bulging of the geometrical envelope to cause the addition of a room) proceeding from insertion of overall massing elements to the insertion of increasingly detailed elements and backtracking where later detailed insertions are not compatible with earlier less detailed insertions. Every insertion would take its dimensions from the context so that only overall building dimensions and any independent dimensions would have to be set. As first suggested by Fraser (Fraser 1993). This process of coaxing an expert system can be taken to whatever level of detail that previous design experience has been encapsulated into rules. Finally, it is conceivable that the sort of interdependencies set up by means of linked vertices in the current Modeling Assembler could be retained in the final object-oriented design representation (that is, the representation that is created by the execution of the rules before it is translated and exported as XML, DXF, etc.). These interdependencies could then allow manipulation of design elements (such as the configuration of rooms) in one area such that they are automatically updated in all related areas. This capability would, of course, be predicated on the availability of a graphical user interface.

It is clear that for the proposed design approach assisted by expert systems to be workable an architect must work within some set of design principles or grammar. Furthermore, dimensioning in relation to some sort of geometrical module, whether regular as in the Usonian houses or Tartan as in the Prairie Houses or based on some other consistent geometrical principle, simplifies the encapsulation of design principles into rules for an expert system. Contrary to restricting creativity, it turns
out that many of the greatest architects have consistently followed a particular design approach for some time while allowing this design approach to evolve gradually before switching or adding to their repertoire with significantly new approaches.

acknowledgments

This project was carried out with the support of the Industrial Assistance Program (IAP) of the National Research Council of Canada by MMIC Inc. Architects of Brantford, Ontario with Donald McIntyre partner in charge. The first author was the principal consultant and the second author chief programmer. The work of Stan Martin on the Building Product Data Base, of Peter Gallagher and Stan Martin on the typical details to be addressed, the helpful criticism of Donald McIntyre and of Colin Bent, Simon Dyer and Russell Thomas of the National Research Council, are gratefully acknowledged. Bruce Jakeway wrote the first versions of the software.

references


CAD Future 5/1, pp. 453–458 (especially Fig. 7c).