Layouts, Solids, Grammar Interpreters and Fire Stations

Rob Woodbury

Department of Architecture
Carnegie Mellon University
Pittsburgh, PA 15213 USA

Eric Griffith

Construction Engineering Research Laboratory (CERL)
US Army Corps of Engineers
Champaign, IL 61826 USA

This paper is presented in three main parts. First, it reports the results of an effort to combine two representations (layouts and solid models) within a single generative framework. Second, it describes fire stations as a building type and reports a phased grammar that embodies information about the type and generates fire station designs likely to be members of the type. Third, it describes a useful way of controlling grammatical generation via interactive decisions on rule application, hierarchical decomposition of designs, and ordering of the conflict set of rule instantiations.

Keywords: *fire station, layout, generative system, search, solid modeling, spatial grammar.*

1 Introduction

Several generative formalisms exist, for instance, shapes (Stiny, 1980), layouts (Flemming, 1989), structures (Carlson, 1991) and boundary solid grammars (Heisserman, 1991). Taken singly, these serve as excellent representations for aspects of an overall design process; but each has its limitations, either theoretical (the representation is limited) or practical (the present state of knowledge does not include useful implementations of the formalism). For different stages of design, different representations are appropriate. Combining two or more representations in a single generative process might lead to generation that is more simple, has greater clarity and is more efficient than would be possible using only a single representation. Layouts and solids are two such representations, and their combination in architecture is appealing. Layouts represent plans; solids support massing and articulation.

U. Flemming and S. Van Wyk (eds.), ***CAAD Futures '93****, pp. 75-90.*

2 Combining Layouts and Solids

Layouts and solid models are used often as abstractions in design. Formalisms for their representation exist in many extant systems; generative formalisms that operate over them are somewhat more rare and are presently largely confined to research laboratories. Combining the two abstractions is, in its most simple form, trivial; polygons in a layout are interpreted as base objects for a sweeping operation that generates solid models, one for each polygon. The value of such a combination lies in being able to interpret correctly both rectangles and their related solids as objects in a design. Some examples of such valid interpretations are (1) architectural spaces interpreted in plan as rectangles and in three dimensions as rectangular solids (of possibly varying heights); (2) circuits, in which layouts represent the size and placement of circuit elements and solids represent the actual objects that the circuit comprises (the solid representation would be used, for example, for thermal analysis); and (3) computer cabinets, in which rectangles indicate the placements of objects on a surface of the cabinet and solids serve as an approximation of the spatial extent of such objects in the third dimension.

Though formally simple, the combination of layouts and solids can make the development of generative models of designs more clear and simple than is possible with either formalism used separately. A specific discussion of how we have used such a combination may be found in a later section in this paper; in the current section we present representations for layouts and solids and our method for combining the two representations in a single generative mechanism.

Layouts are considered here as arrangements of rectangles with sides parallel to the axes of a Cartesian coordinate system. Such arrangements provide a useful level of abstraction in many design situations. Several representations of layouts with various characteristics have been reported in the literature. Important to us were two properties, both of which exist in the *wall representation* and its variants (Flemming, 1989, 1992). First, it was important that partial designs could be meaningfully represented. The operators of the wall representation insert (and delete) rectangles individually and a well-formed layout exists after every operation. Second, it was important that the objects inserted were parametric, that is, able to be varied in size and shape within given limits. That the wall representation explicitly separates the representation of discrete and continuous variables in layouts directly supports this need.

Formal descriptions of various schemes based on wall representations are readily available, and we do not repeat such a description here. Our scheme is essentially that reported in Flemming (1989), but is augmented, in a manner similar to Coyne (1991) by allowing each object to in turn comprise a layout of objects, thus creating a hierarchical layout. Our actual implementation is more simple than the ones based on Flemming's (LOOS) and Coyne's (ABLOOS) work, the main differences being:

1. only rule 1 of LOOS is implemented (see Figure 1);
2. the operators apply at any pivot vertex in a layout;
3. orientation of objects is determined at the time of insertion;
4. fixed objects are not allowed in a layout;
5. the system of *Goal Objects (GOB's)* found in ABLOOS is not duplicated in its full sophistication; and
6. object insertion operators are hierarchical in that they can insert an object without the prior existence of parent objects in the hierarchy.

Of these, difference (2) exists because our generative model is interactive. Having operators apply at any pivot vertex means that there are many ways to reach a particular

configuration and, thus, that early insertion decisions exclude less of the space of possibilities. Differences (1), (3), (4), and (5) are largely expedient and reflect the time constraints under which the implementation was completed. Difference (6), hierarchical insertion operators, is due to our desire to use the hierarchical decomposition of a design as an architectural program whose ordering determines the order of insertion of objects. Supporting insertion of objects without the precondition of prior insertion of their parents in the hierarchical decomposition allows a program to be defined in arbitrary order. We call the hierarchical insertion operators and rule 1 taken together the *fundamental operators* in our implementation.

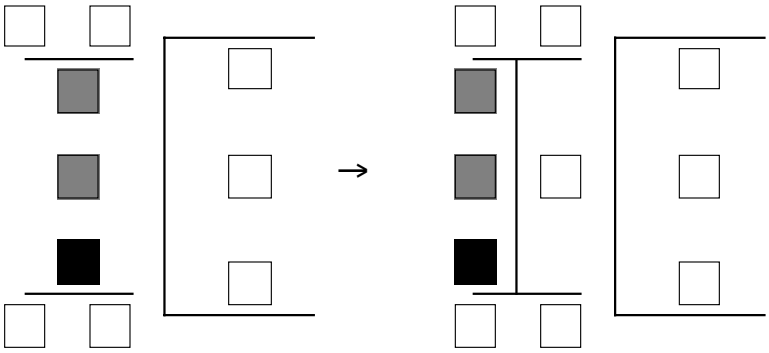

Figure 1. Rule 1 (Flemming, 1989). The black rectangle on each side of the rule denotes the *pivot vertex*, a distinguished vertex from which the gray rectangles denoting the pivot *sequence* are found. The pivot sequence includes some of the vertices that (in this case) are *above* the pivot vertex and adjacent on the *right* to the same *wall* that the pivot vertex is adjacent to. Walls are conceptual separators between vertices; all vertices on the left (right) side of a wall are constrained to be to the left (right) of the vertices on the right (left) side of a wall. Any vertex not the parent of a layout may be a pivot vertex for an application of rule 1. The rule may be rotated by increments of 90 degrees allowing it to be applied in the four principal directions of *left*, *above*, *right* and *below*.

Solids are considered here as 3-manifolds with boundary. Several representations for such objects exist (Vanacek, 1989; Karasick, 1988). We use a variant of the *split-edge* data structure that is modified to allow multiple vertex neighborhoods and cycles of pairs of faces around edges (a more complete description of the representation may be found in Heisserman (1991)). Our representation scheme is restricted to planar surfaces; thus, the embedding of a surface in three-space can be described by assigning coordinates to the vertices of the polygons. The fundamental operators on this representation are variants of the well-known Euler operators and a set of vertex-based geometry assignment operators.

Layouts and solids are combined in an almost trivial way. Solids may be generated from a layout as rectangular cuboids, given a description of the plane on which the layout exists and a height property for each of the objects in the layout. If the height property of a layout object is nonexistent or zero an interpretation of it is made as a lamina "solid," that is, a 2-manifold with boundary. This requires an extension of the model for a solid, but no change in the representation for solids. Operators such as the Boolean operations must, of course, recognize and not accept such laminar solids. As shown in Figure 2, hierarchical layouts correspond to solids in a number of ways. Every level of a hierarchical layout comprises a parent, which must be an object allocated in some layout in the hierarchy (un-

less it is the top-level object) and a set of objects (called the *children* of the layout) allocated in a layout with the same name as the parent. Every such level can be represented as a collection of solids by generating solids from either its parent, its children, both its parent and its children or neither its parent nor its children. Thus, a given hierarchical layout corresponds to a set of collections of solid objects.

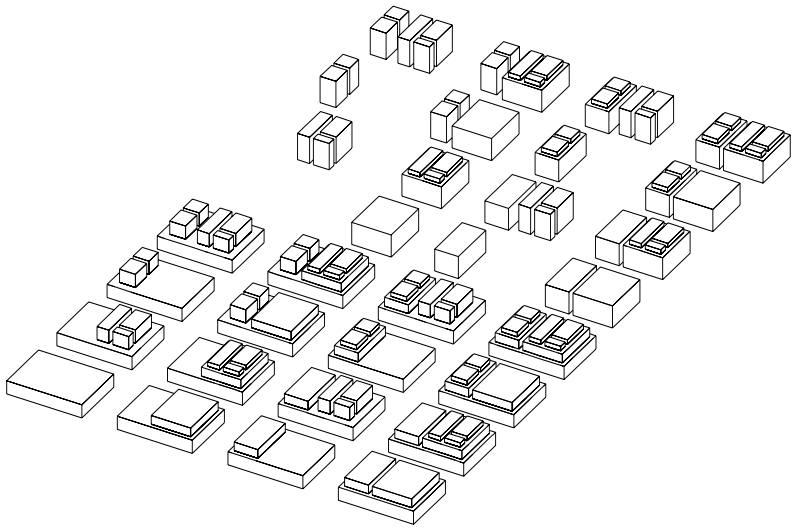

Figure 2. The different interpretations of a two-level (+ root) hierarchical layout. For clarity in this diagram, each level in the hierarchy is given a greater height than its parent in the hierarchy and the rectangles of each level are shrunk slightly relative the their parent.

2.1 A Grammar Interpreter

A *boundary solid grammar* uses the boundary representation described above. The *topology* is represented as a graph. Nodes of the graph are topological elements and arcs are pairwise adjacencies between elements. The *geometry* is represented as vertex coordinates. In addition to topology and geometry, labels can be associated with any node of the boundary representation and a state is associated with each instance of the representation.

Matching is performed on the current state of a design. Matching on solids, possibly recursive and/or hierarchical, is accomplished by logical combinations of the primitive relational properties of node existence, arc existence, geometry values, label existence and value, and state value. A single condition can be used to express the conditions of an infinite set of graphs using recursive logical definitions. Alternately, a condition may match greatly varied topology graphs using several logical clauses with the same head.

Operations are performed on the current state of a design. Primitive operations on solids are the Euler operators, geometry assignments, label operators and the state change operator. Composite operations on solids are expressed by sequences of primitive operations, match conditions and composite operations.

A solid rule is a set of match conditions and a sequence of operations. The match conditions determine when a rule may be applied to a given boundary representation. Each of the rule's conditions must be satisfied with respect to the given boundary graph and

bindings for all free variables must be found. When the rule is applied, the sequence of operations transform the boundary graph, modifying the representation of the solid(s) and/or creating additional solids.

A boundary solid grammar comprises: (1) an alphabet appropriate for boundary graphs, (2) a topologically valid initial boundary graph, (3) a finite set of logical conditions partitioned into *conditions* and *operations*, and (4) a finite set of solid rules formed from the elements of (3). A boundary solid grammar produces a language of boundary graphs. The language is all graphs producible by the grammar in which the state is the special state *done*.

Rule instantiations (rules that may be applied and their free variable bindings) determine a *conflict set*, more properly a *conflict sequence*. The rule instantiations in the conflict sequence are ordered as part of the control scheme of the device that implements our grammar.

Genesis is an implementation of the boundary solid grammar formalism sketched above. It provides facilities for the representation and display of solids, match conditions and operations, rule definition, and searching the languages of grammars. It is implemented primarily in *CLP(R)*, a constraint logic programming language.

Using essentially the same structure for representation, matching, operations, rules, and grammar, we have implemented a layout generator for hierarchical graphs that is tightly-integrated with *Genesis* in the following sense: rules in the interpreter can freely include conditions that match and operations that act on either the layout or the solid representation. In particular layouts are displayed as collections of lamina in the solid representation, arrayed on some plane in three-space.

3 Fire Stations

Fire stations have become a remarkably standard building type with respect to their basic configuration. In essence, a fire station comprises three zones: a garage called an *apparatus room*, where trucks and equipment are stored; *dormitory* facilities where personnel can rest and exercise; and *day use* facilities, where personnel eat and where administrative, training, and other daily activities of the fire station occur. The spaces of these last two zones are often called *service spaces.* Several constraints force fire station design into a few basic organizations. Variants of these are certainly possible and experiments with non-rectangular geometries have been attempted, but few current station designs stray far from the basic set. In Figure 3, diagrams A, B, and C violate fewest of the constraints discussed below. Diagrams D, E, and F represent types that have been recently realized in the United States as actual fire stations but which are problematic with respect to the constraints.

Fire station design is dominated by functional concerns related to rapid access to equipment and separation of use zones. There is a strong incentive to reduce the "turn-out time" implied by the design, that is, to make access to the equipment of the station as rapid and safe as possible in the event of a call to a fire. This has tended to restrict fire station designs to single story, or at most, split-level, structures, thus eliminating organization F and some variants of D and E. Placing the dormitory and administrative wings each with direct access to the apparatus room also reduces turn-out time.

When at all possible within given site conditions, a drive through apparatus room is preferred. This constraint eliminates configurations with U- and L-shaped service areas (organizations D and E) from consideration in most cases. Plans for future expansion of existing fire stations sometimes dictate that all of the service spaces be placed on one side

of the apparatus room, thus allowing additional apparatus bays to be constructed in the future, thus arguing against alternatives B and E.

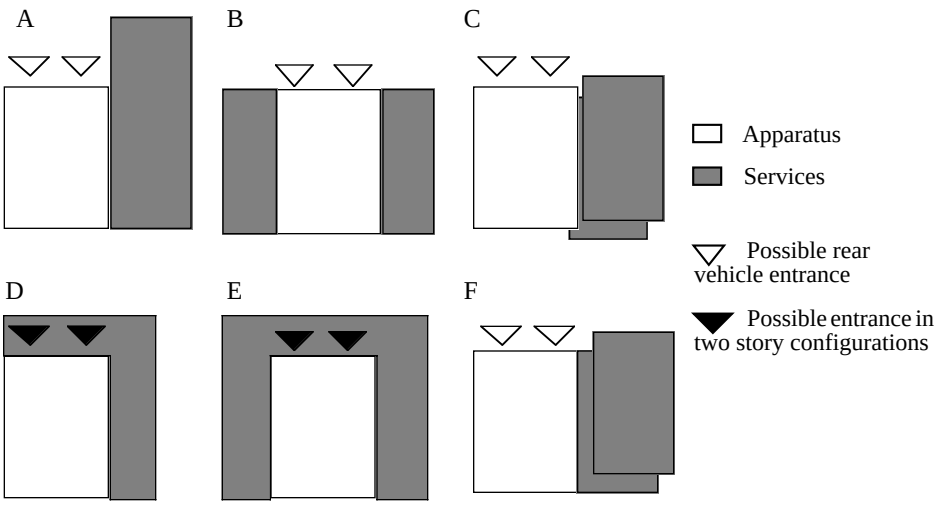

Figure 3: Basic organizations for fire stations.

Separation of the dormitory and administrative zones is desirable for acoustic isolation. Organization A does not exhibit strong separation of these two zones, but good separation can still be accomplished with it.

Within the day use zone, a room called the *watch room* must be adjacent to all of the street, the apparatus room and the main public entrance of the fire station.

The site provides several constraints on fire stations. An apron of 30 feet minimum is required in front of the apparatus room to ensure that operators can see the traffic and road conditions as they exit the fire station. The apron also performs a secondary function as a space for cleaning equipment. Currently, environmental requirements on the disposition of liquids from the cleaning process are tending to separate these two functions. Specific cleaning spaces now tend to be located in the rear of the station.

The watch room should have unobstructed views of the apron area, although it is desirable for the watch room to have views along the road in both directions. North or south facing access directions are preferred over east or west facing ones, in order to eliminate the effect that low sun angles can have on vision when exiting from the apparatus room.

Once-common architectural devices in fire stations, for example, drying towers and pole slides, have been largely eliminated due to development of hose drying technology and increasing regulation of on-the-job safety.

3.1 A Grammar for Fire Stations

We have used the combination of layouts and solids described above as the representation for a grammar to generate designs for fire stations. Our study posed the following question: can a grammar be used as a way to describe standardized designs with specific properties? Standardization of designs is a strategy increasingly pursued by owners and manufacturers of buildings. Owners of large numbers of functionally similar

buildings are attracted to a standardization strategy for the control it provides over cost, image, and/or building function. Manufacturers of buildings and building components are attracted to standardized designs for the rationalization of parts and assembly procedures promised by well-considered modules and standard details. Different owners are concerned with different aspects of standardization.

Present technical tools for representing standardization are arguably weak: they comprise little more than example drawings and written specifications, taken in various combinations. Such drawings have the shortcoming of providing only one (or at best a few) exemplars – in this respect they are overly rigid. They also tend to center on the parts (be they construction or spatial elements) and provide little control of configuration – in this respect they tend to be overly permissive. In examples of standard design specifications that we used, the two forms of documentation sometimes conflicted. The example designs were in demonstrable conflict with the specifications. Our premise was that a grammar with appropriate test criteria applied to members of its language could provide a way of specifying standard designs that would be flexible in the type of control that it would permit, yet open in the possible designs that would be acceptable under the standard. We have been able to partially support this premise. Generation of exemplar standard designs and of variations close to the exemplars is readily possible. As usual in writing grammars, it proved to be much more difficult not to generate undesirable designs that it was to generate designs from a given set of examples. Apparently if grammars are to be used as a way of describing standardized buildings, they must be coupled with post-processing steps that accept only those designs that meet the criteria of the standard. The value of grammars, then, is that they are a means to express known desirable patterns in buildings in a constructive sense.

If grammars are to be useful for describing designs, people must be able to use them. There must be ready ways to describe grammar rules and to search the spaces of designs given by these rules. In the GENESIS formalism and implementation, describing grammar rules is a straightforward if skilled task, but searching a space of designs is only weakly supported, this latter issue we address in the present work. A particular limitation in the present implementation is that only a single current state is supported with no backtracking. Accepting this limitation means that the tools available for controlling search are the order of insertion of objects, and the order in which rule instantiations appear in the conflict set. Even in these apparently tight limits, we found that useful control of search could be achieved.

The description of standard fire stations that formed the basis of our study emphasized the separation of design decisions into blocks, to distinguish those blocks controlled by standards from those controlled by local concerns. Siting, planning, and three-dimensional development, in particular, were to be separated. The standard specified some criteria for siting, but left considerable latitude for adaptation. The plans of the actual fire stations were tightly specified, both as specifications supported by drawings of individual spaces and by exemplar designs of entire fire stations. Three-dimensional development of the designs was explicitly left mostly to local control, primarily because the local context and codes could not be anticipated centrally. Our grammar mirrors this separation; more strongly, it is organized into a sequence of phases, each generating some aspect of the design. The early phases operate on layouts; the later ones operate on solids. A list of the phases follows. For each phase, the input and output representations are specified.

> **Site:** A rectangular site is presumed. *Input*: the size of the site. *Output*: a hierarchical layout containing only a single object denoting the site.

Roads: Roads may occur on any side of a site and in any combination. *Input: th*e site as a hierarchical layout. *Output*: a hierarchical layout of one level containing the site and rectangles for the roads adjacent to the site.

Setbacks: Site setbacks differ depending on the presence or absence of roads along site boundaries. *Input:* the output from the road phase. *Output*: a two level hierarchical layout. The site in the input layout is expanded to a layout containing the required setbacks and a zone for the buildable area.

Station layout: A station is laid out one space at a time, in the order given in a hierarchical decomposition of the station. *Input:* The output from the site setback phase plus a building program as a hierarchical decomposition of the design. *Output:* A hierarchical layout in possibly many levels, but generally including the three main fire station zones (apparatus room, administration wing and dormitory wing) as the main components of the station level (the one immediately below the buildable area object in the hierarchical layout).

Interior wall placement: Spaces in a fire station are distinguished by a function property from the set *{circulation, apparatus zone, exterior, general}*. Objects of *general* functionality are presumed to have the same function as their name; thus their actual function is unique in a particular design. Interior walls are placed around objects of either *circulation* or *general* functionality. *Input:* the station layout specified above. *Output:* A hierarchical layout with the hierarchy describing the station collapsed into a single layout (a hierarchical layout is collapsed into a non-hierarchical layout by replacing objects in the hierarchical layouts by the layouts which they head). In this graph, objects denoting half-walls are placed around every object of *circulation* or *general* functionality and on the boundary that separates those objects from others of different functionality (see Figure 4). Half-walls inherit the height property of the spaces they partially enclose. Half-walls greatly simplify the later placement of exterior walls, and this is the motivation for their use.

Wall and space extrusion: Walls and spaces are extruded into the third dimension using the height parameter associated with each object. At this stage, the representation used by the grammar is transformed to a representation of solid objects. The layout representation, although retained and used in the matching of many rules, is no longer changed by rule application. *Input:* the above station layout with half walls from the previous phase. *Output:* The station layout plus a collection of solid models, one for each object in the station layout.

Exterior walls: Exterior walls are added to all exposed half-walls. Heights of adjacent spaces may be adjusted at this stage to simplify the exterior walls and subsequent roofs. *Input:* the output of the previous phase. *Output:* the input plus solid models representing the exterior walls.

Openings: Windows and doors are added to the design. Doors may join any space to an adjacent circulation space and any two spaces within a level of a hierarchical decomposition. Doors vary from standard openings for single doors to the complete absence of a wall. Windows may be inserted in each space adjoining the exterior in a number of options related the position of a window within a room. *Input:* the output from the phase above. *Output:* the input with openings added to the solids representing the exterior walls.

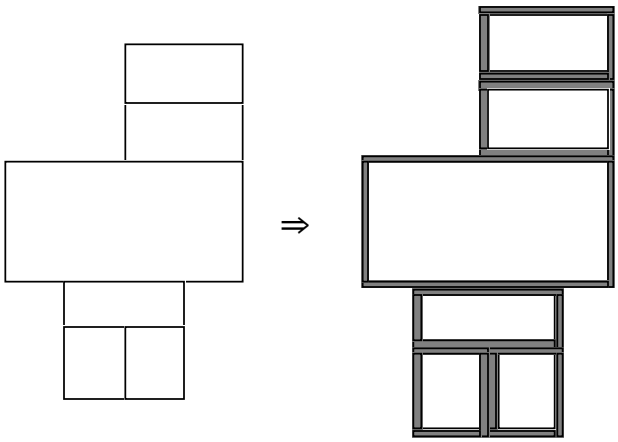

Figure 4. A layout and the same layout with half-walls added around each of its constituent rectangles.

Roofs: The shape and structural pattern of the roof is established. Presently only the roof forms from the example corpus are generated. These are horizontal roofs of low slope and long overhang (vaguely reminiscent of the Prairie School style). *Input:* the output of the above phase. *Output:* the input plus solid models representing the overall roof geometry.

3.2 An Example Derivation of a Fire Station

Figure 5 shows sample steps in an example derivation of a fire station. The derivation follows the steps given in the grammar described above. Successive slides show the site; roads and setbacks; basic station organization; development of apparatus room, administrative and dormitory wings; insertion of half-walls; extrusion of spaces and walls; addition of outside walls; and finally placement of apparatus openings, doors and windows.

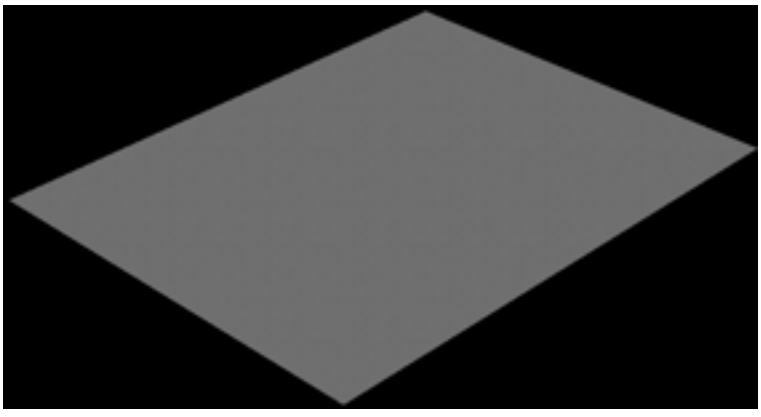

Figure 5a. A site for a fire station.

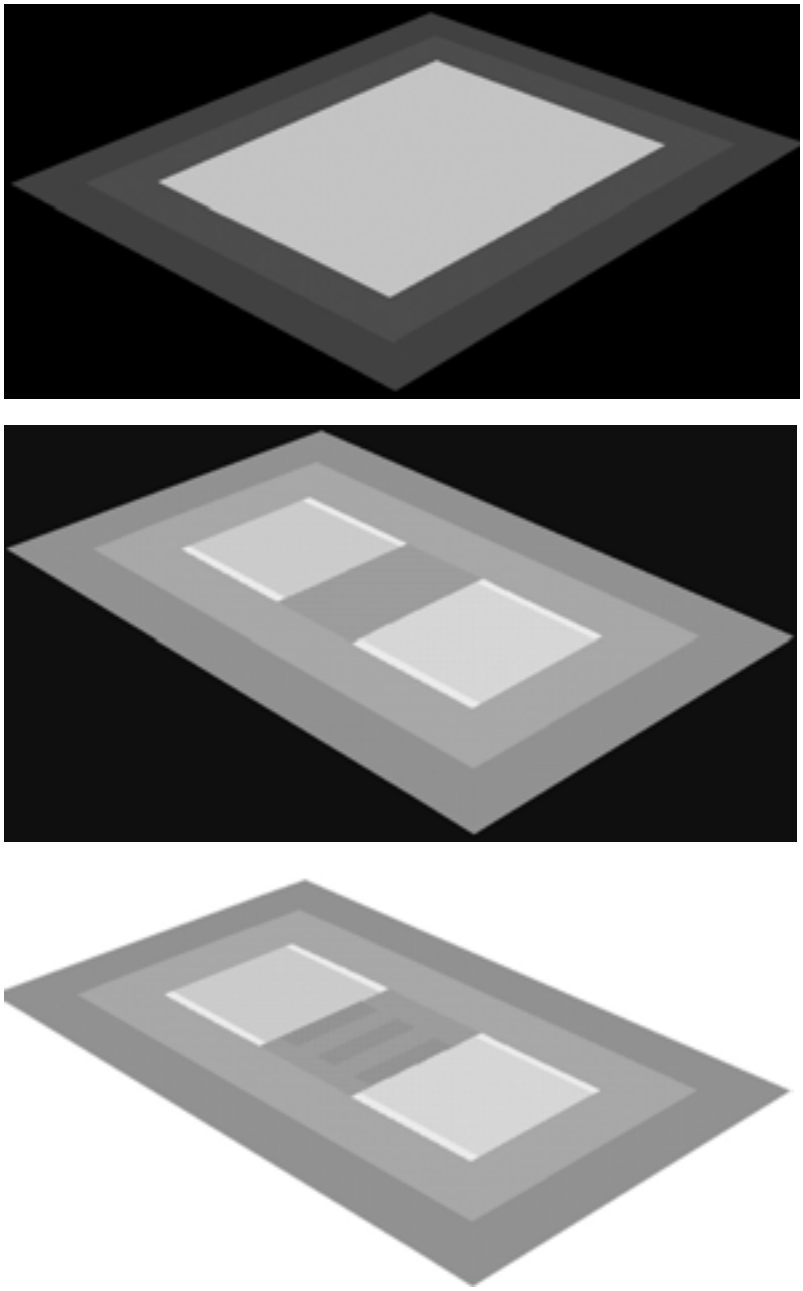

Figure 5b-d. Roads and setbacks from the fire station (b); a basic spatial organization for a fire station (c); and development of the apparatus room of a fire station (d).

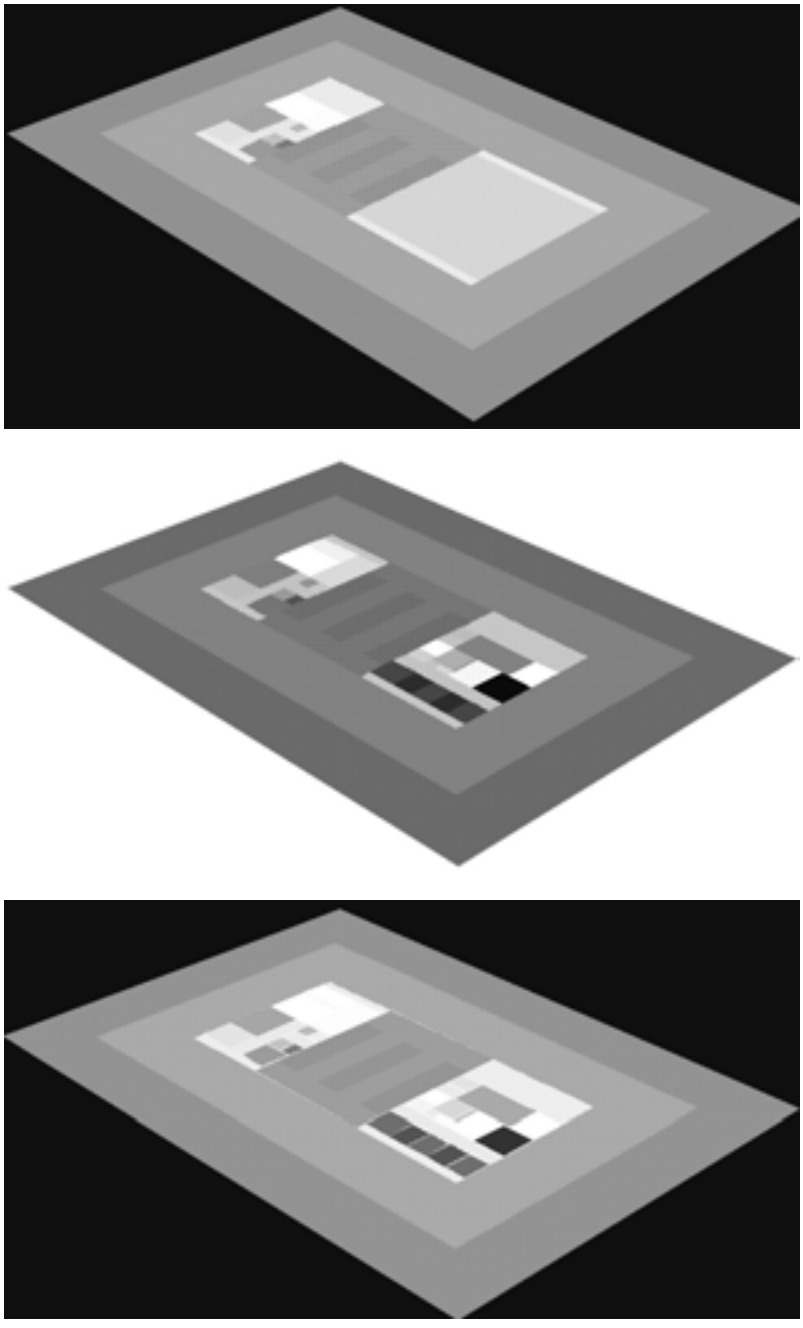

Figure 5e-g. Development of the administrative wing of a fire station (e); development of the dormitory wing resulting in a complete fire station plan (f); and insertion of interior walls (g).

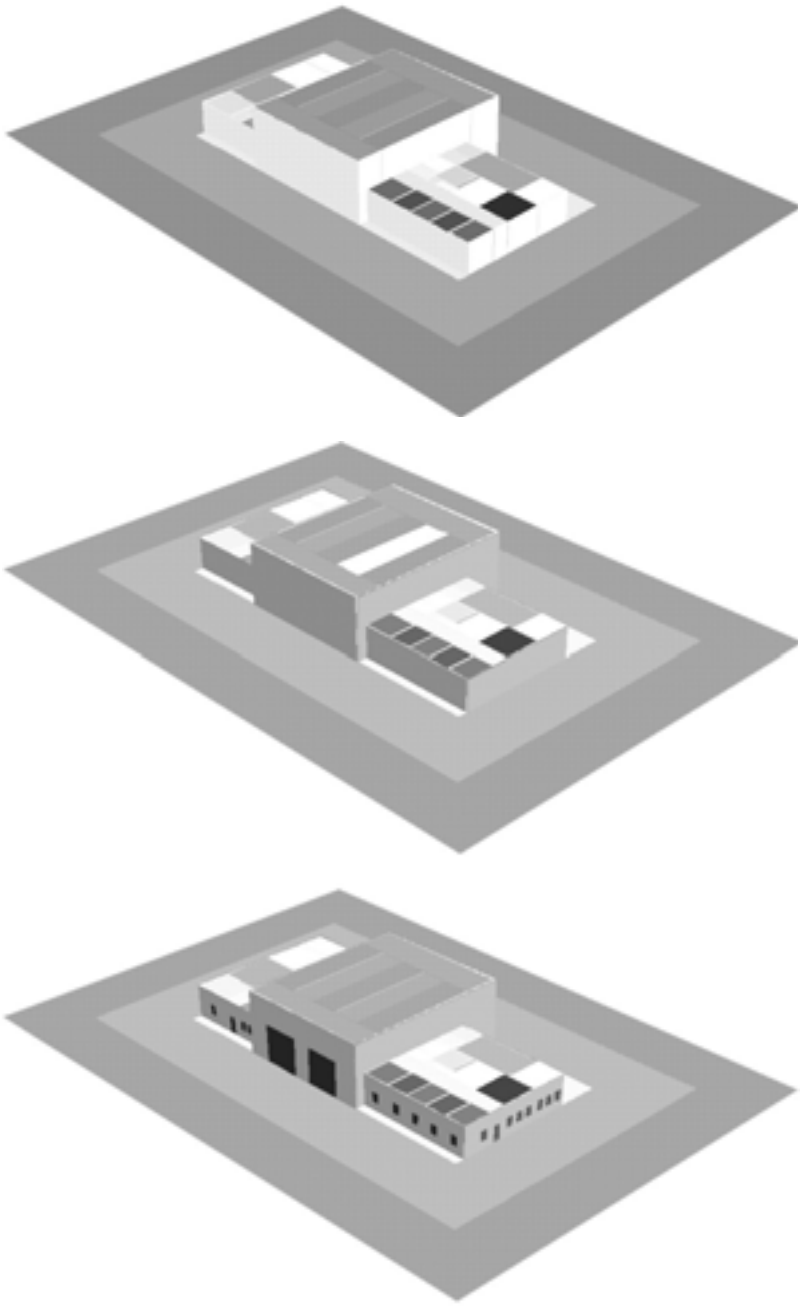

Figure 5h-j Extrusion of spaces and walls (h); addition of outside walls (i); and placement of apparatus openings, doors and windows (j).

:4 A Control Scheme for Rule Application

A control scheme is necessary in any generative formalism. The control scheme we use is a combination of *hierarchical decomposition*, strictly interactive decisions about application of rule instantiations from the conflict set, and a depth-first priority tree for the appearance of rule instantiations in the conflict set. In this priority schedule, *substitution of pre-generated partial designs* is first, followed by specializations of the preconditions for rule 1 inserting *special objects*, followed by the order of the hierarchical decomposition, followed by specializations of the preconditions for rule 1 inserting *program objects*, and lastly followed by the fundamental rules themselves inserting program objects. Both the components of this control scheme and their order in the conflict set developed from our experience using the layout system. Certainly, the control strategy is heuristic. Other orderings are possible and may very well work well—this one merely has the advantage that it appears to work well in the grammars we tested.

An object, once placed, may be expanded into a layout if one or more layouts for that object already exists. For example, Figure 6 shows that a truck bay in an apparatus room of a fire station may be allocated as a single object. Depending on its context, different clearance zones may be added around the truck bay. Thus the truck bay is replaced by a layout containing an object for the actual space for the truck as well as objects representing each of the clearance zones.

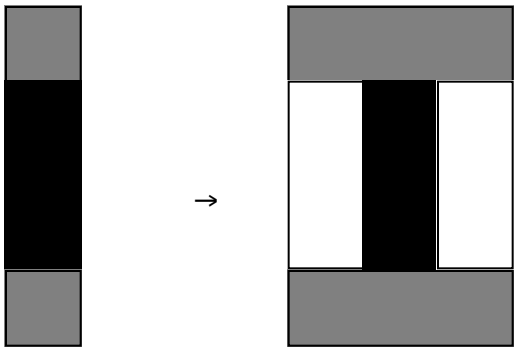

Figure 6. A truck bay (black rectangle) may be expanded into an actual bay for a truck and clearance zones at its sides. The grey rectangles represent the clearance zones that must exist at each end of a truck.

A hierarchical decomposition of a design is specified as a tree of unique names, one for each space to be allocated in the design. In the notation we use, a *group* denotes a single layout and its parent object. Objects appearing as parents must appear as objects in some other group with the exception of a single object denoted as the top-level object. An example decomposition for one part of a fire station follows:

top_level (station).
group (station, (adm_wing)).
group (watch, (bath, storage, alarm)).
group (day_area, (kitchen, dining, day_train)).
group (offices, (shift_super_office, fire_insp_office)).
group (adm_supply, (admin_supply, training_supply)).
group (adm_wing, (day_area, offices, hall5, adm_supply, hall4, watch)).

Decisions about inserting objects from this hierarchical decomposition are made in the order in which the objects are specified as members of a group. The appearance of an object as the parent of a group is ignored in the insertion order. When an object without an inserted parent is encountered its parent and recursively all of its uninserted parents are inserted into the layout. Thus in the group specification above, the order of insertion would be:

> adm_wing, watch, bath, storage, alarm, day_area, kitchen, dining, day_train, offices, shift_super_office, fire_insp_office, adm_supply, admin_supply, training_supply, hall5, hall4.

As shown in Figure 7, the LHS of rule 1 finds a *pivot-vertex* in the layout in which an object is to be allocated; a *direction* from that pivot-vertex that indicates the direction in which the new object is to be inserted and identifies a *full-pivot-sequence* of objects that are eligible to participate in the operation; and a *pivot-sequence* that indicates a prefix of the full-pivot-sequence. This LHS will find precisely once every possible instantiation of rule 1 in a given layout.

```
lhs(add_room_in_sublayout,
        (RoomName,Layout,Pivot,Direction,PivotSequence) ):-
    state(add_room),
    current_selection(RoomName,GroupName),
    inserted(GroupName),
    named_graph(GroupName,Layout),
    pick_pivot_vertex(Layout,Pivot),
    pick_direction(Direction),
    pick_pivot_sequence(Pivot,Direction,PivotSequence,Layout).

rhs(add_room_in_sublayout,
        (RoomName,Layout,Pivot,Direction,PivotSequence)):-
    inverse_direction(Direction,Inverse),
    add_rotated_vertex_at(RoomName,Inverse,Room,Pivot,
        PivotSequence,Direction,Layout,NewLayout),
    stow(NewLayout),
    pop_program_item(RoomName),
    set_state(select_room).
```

Figure 7. The Prolog code of rule 1. The left-hand side (lhs) of the rule matches if the grammar is currently adding rooms, if the layout that will contain the Room has been inserted and can be found, and if a pivot vertex, direction, and pivot sequence can be found. The right-hand side (rhs) adds the vertex, thereby transforming the Layout to the NewLayout, puts the NewLayout into the representation, removes RoomName from the list of items to be added and puts the grammar into a state in which it will look for another room to add.

The control strategy is interactive in the following sense. After each insertion of a rectangle the entire conflict set of instantiated rules is, in principle, made available to the user in the order given above.

Each object which can be expanded into a predetermined layout appears in the conflict set in order of the recency of insertion of the object to be expanded. Thus most recently allocated objects will be matched for expansion first. If not expanded at this stage, an object will remain unexpanded until one of the members of the group of which it is head is inserted. Once this occurs, none of the predetermined solutions for that group apply and the rest of the group must also be inserted interactively.

After object expansions in the conflict set come the insertion of special objects. These are spaces that are pertinent to all designs generated by a grammar even though they do not appear in a specific architectural program. For example in fire stations, the apparatus bay is often pushed forward of the rest of the building. Insertion of objects denoting exterior offset spaces accomplishes this movement.

The rest of the rule instantiations appear in the conflict set in the order given by a depth-first traversal of a three-level tree. The first choice is the object to be inserted, the first object chosen being the first uninserted object given in the hierarchical decomposition. Once an object is chosen, rule instantiations are formed from a sequence of rules that contain first specialized and then general rules. The pivots, directions and pivot-sequences for these compose the third level of the tree and are found in the order of recency of insertion of the objects. Most recently inserted objects appear first as pivots. Figure 8 diagrams the order of appearance of rule instantiations in the conflict set.

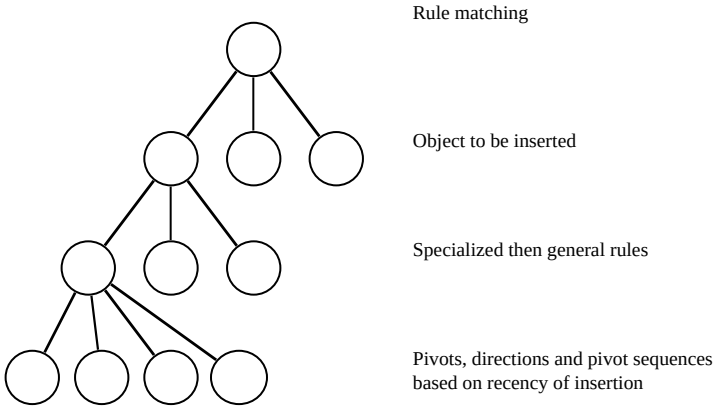

Figure 8. The order of choices in inserting an object.

The rest of the rule instantiations appear in the conflict set in the order given by a depth-first traversal of a three-level tree. The first choice is the object to be inserted, the first object chosen being the first uninserted object given in the hierarchical decomposition. Once an object is chosen, rule instantiations are formed from a sequence of rules that contain first specialized and then general rules. The pivots, directions and pivot-sequences for these compose the third level of the tree and are found in the order of recency of insertion of the objects. Most recently inserted objects appear first as pivots. Figure 8 diagrams the order of appearance of rule instantiations in the conflict set.Decisions about which rule instantiation in the conflict set to apply are made by the user. The rule instantiations are presented in order described above. A choice of one instantiation fires the chosen rule and invokes the creation of a new conflict set of rule matches. Evaluation of the conflict set is entirely lazy—a rule instantiation is computed only when the instantiation immediately preceding it in the conflict set is declined by the user.

5 Conclusions

Given the present state of this work, our conclusions must be modest. It is theoretically simple and practically possible to combine layout and solid representations in a single grammar implementation. A simple control structure that comprises hierarchical decomposition of a design with ordering of rule instantiations in the conflict set gives surprisingly good control for interactive rule-based generation of designs. Grammars for fire station designs can be written in a clean sequence of phases, each of which is visited once in a process of generation. Examples from a corpus of fire station designs and some interesting variations of those examples can be generated with relative ease. We hope to continue this work and to produce more insightful results on the relation between grammars and standardized building programs.

Acknowledgements

The research described here was partially supported by the Construction Engineering Research Laboratory of the United States Army Corps of Engineers and by the Engineering Design Research Center, a center at Carnegie Mellon University funded by the National Science Foundation. The author ackowledges the considerable assistance of Kevin Zawicki, for programming much of the grammar reported, and to Jeff Heisserman, for programming advice and for comments on a draft of the paper.

References

Carlson, C., McKelvey, R., and Woodbury, R.., 1991. "An Introduction to Structure and Structure Grammars," *Planning and Design* 18(4), pp. 417-426.

Coyne, R., 1991. *ABLOOS*: *An Evolving Hierarchical Design Framework*, Ph.D. Dissertation, Department of Architecture, Carnegie Mellon University.

Flemming, U., Baykan, C.A., Coyne, R.F., Fox, M.S., 1992. "Hierarchical Generate-and-Test vs. Constraint Directed Search: A Comparision in the Context of Layout Synthesis," in J.S. Gero and F. Sudweeks (eds.), *Artificial Intelligence in Design '92*. Dordrecht: Kluwer Academic Publishers, pp. 817-838.

Flemming, U., 1989. "More on the Representation and Generation of Loosely Packed Arrangements of Rectangles," *Planning and Design*, Vol. 16, pp. 327-359.

Heisserman, J.A., 1991. *Generative Geometric Design and Boundary Solid Grammars*, Ph.D. Dissertation, Department of Architecture, Carnegie Mellon University.

Karasick, M, 1988. *On the Representation and Manipulation of Rigid Solids*, Ph.D. Dissertation, Department of Computer Science, McGill University, Montreal, P.Q., Canada.

Stiny, G., 1980. "Introduction to Shape and Shape Grammars," *Environment and Planning B* 7(3), pp. 343-352.

Vanacek Jr., G., 1989. "Protosolid: An Inside Look," *Technical Report CSD-TR-921*, Computer Science Department, Purdue University.

Zurier, R., 1982. *The American Firehouse*: *An Architectural and Social History*, New York: Abbeville Press.