SGML

A Shape Grammar Meta-Language

Haldane Liew

Massachusetts Institute of Technology

http://architecture.mit.edu/~haldane

Abstract. The shape grammar meta-language creates layers of abstraction in the shape grammar formalism for the purpose of developing descriptions that can organize and sequence rule applications in a grammar, filter information in a drawing, and provide additional matching constraints based on the subshapes found. These concepts are incorporated into the formalism by expanding the rule application process into six phases; control, context, transformations, parameters, descriptors, and application. Seven meta-language descriptive conventions are developed from the six phases; rule-sets, directive, label-filters, focus, maxline, zones, and apply-mode. Examples of how the meta-language descriptions can be used to generate designs are provided.

Keywords. Shape grammars, meta-language, and computational design.

1.0 Introduction

Shape grammar is a general computational formalism that manipulates shapes to generate designs. It has been used in the architectural field to develop designs such as Palladian villas (Stiny and Mitchell 1978), Prairie houses (Koning and Eizenburg 1981), and even Chinese government buildings based on the Yingzao Fashi book (Li 2001). The use of shape grammars is not limited to the architectural field. It has also been used for designs such as window lattice designs in the Iceray grammar (Stiny 1977), Froebel block arrangements in the Kindergarten grammar (Stiny 1980), and coffee makers in the Coffee maker grammar (Agarwal and Cagan 1998). Each grammar generates a design by using a particular technique of the shape grammar formalism.

For the Iceray grammar, a sub-division technique is used to generate the design. Each polygon is divided to create more polygons which recursively gets divided. The Kindergarten grammar uses labeled points to generate distinctive arrangements of Froebel blocks. The labeled points use the symmetry of the blocks to control how the blocks can be assembled. The Yingzao Fashi grammar uses parallelism as its technique for generating a design. The parallel descriptions allow the design of the section and elevation to be in correspondence with the floor plan.

This paper introduces another technique to use shape grammars for design. This technique involves the use of a meta-language which is based upon an extension of the rule application process in the formalism. Unlike the other techniques, which use the existing language in the grammar for design, the meta-language creates new descriptive conventions for designing with shape grammars.

1.1 Shape Grammars

The rule application process in shape grammars is composed of three phases; *transformations*, *parameters*, and *application*. This is evident by the two formulas for rule application (Stiny 1990, 1991): For a given schema rule $A \rightarrow B$ and a drawing C, first determine if schema A, is a subshape of drawing C:

$$t(g(A)) \leq C \qquad (1)$$

Second, the new drawing C' is the result of subtracting schema A from drawing C and adding schema B:

$$C' = C - t(g(A)) + t(g(B)) \quad (2)$$

This two step process describes the mechanics of applying a rule in the shape grammar formalism. Associated with each schema is a transformation function t() and a parameter function g(). These functions determine what transformations and parameters are valid to match a subshape in drawing C. These two functions make up the first two phases; *transformation* and *parameter*.

The third phase is the *application* phase. The first formula (1) will only find one pair of values for t() and g(). Typically, there are multiple subshapes in a drawing which would require a set of values for t() and g(). To produce this set of values, a slight modification is made to the formulas:

$$\text{For all t and g such that } t(g(A)) \leq C \quad (3)$$

To apply the entire set of subshapes to drawing C:

$$C' = \Sigma\,(C - t(g(A)) + t(g(B))) \quad (4)$$

The *application* phase determines how a set of subshapes should be applied to the drawing C. The options range from applying only one of the subshapes to applying all of the subshapes, which is equivalent to the parallel application of the rule.

The three phases described thus far, explain the mechanics of applying any rule. The formulas in the formalism only describe how to find a shape, erase it and add another shape. What is omitted from this process are the decisions that need to be made before applying a rule. In order to apply a rule, a rule must first be selected. How is a rule chosen? Is the user free to select any rule in the grammar or does the grammar restrict which rules can be used? Once a rule is selected, where can the rule apply? Can the rule apply to all parts of the drawing or only certain portions?

These questions address the issue of grammar application and derivation. There are three actions to apply a rule; determination of rule, determination of object, and determination of matching condition (Chase 2002). These actions generate three more phases for the rule application process; *control*, *context*, and *descriptor*. The *control* phase determines which rule to apply. The *context* phase determines what portion of the drawing to apply a rule. And the *descriptor* phase provides additional criteria, other than transformations and parameters, to match a shape.

The next section combines the three new phases, *control*, *context*, and *descriptor*, with the original three phases, *transformation*, *parameter*, and *application*, to create the six phases of the rule application process. Each phase and its corresponding descriptions in the shape grammar meta-language will also be explained.

2.0 Six Phases

The six phases of the rule application process are; *control*, *context*, *transformation*, *parameter*, *descriptor*, and *application* (figure 1). The sequence of the phases is determined by the chain of decisions that need to be made in order to apply a rule. The first phase, *control*, determines which rule to use. Once a rule has been selected, the second phase, *context*, isolates a portion of the drawing to apply the rule. This is achieved by selectively removing parts of the drawing. With a modified drawing from the *context* phase, transformations and parameters are evaluated which are the third and fourth phases. The fifth phase, *descriptors*, provides additional matching constraints based on the subshapes found. The sixth and final phase, *application*, determines how a set of subshapes is applied to the drawing.

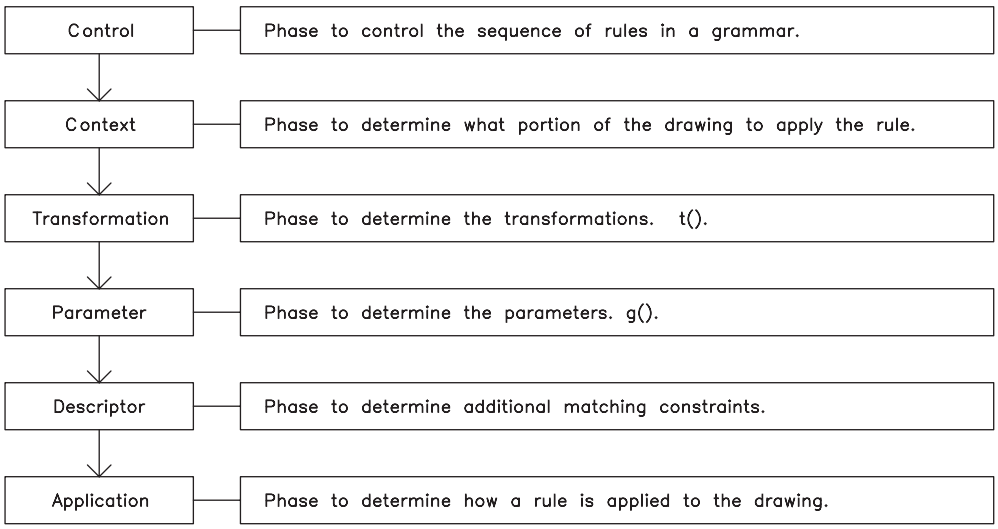

Figure 1. Six phases of the rule application process.

The rest of this section will provide details of each phase and any corresponding descriptive conventions from the shape grammar meta-language (figure 2). The explanations will start in reverse order from the *application* phase to the *control* phase.

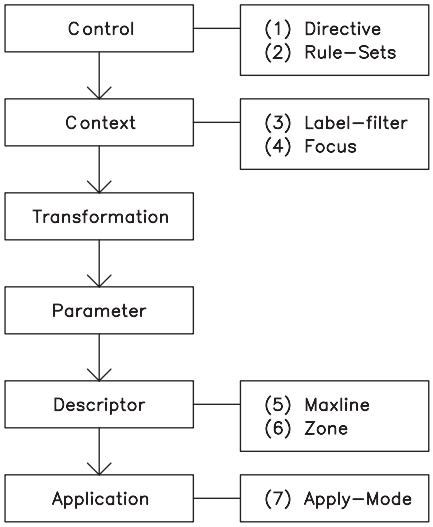

Figure 2. The 7 meta-language descriptions and their corresponding phases.

2.1 Application Phase

The *application* phase determines how a set of subshapes are applied to a drawing. Typically only one subshape is used. However, parallel application is also possible which would apply all the subshapes at the same time.

2.1.1 Apply-Mode

The meta-language provides a description named “apply-mode" which has 3 options; user, parallel, and any. The user option allows the user to select one of the subshapes

for application. Figure 4 shows the selection of a 2x2 square in the upper right-hand corner using the rule from figure 3. The second option is “parallel”. This option will automatically apply the rule to all the subshapes. A derivation using the rule from figure 3 is shown in figure 5. The third option is named “any”. With this option, any one of the subshapes will be applied to the drawing. This option is used when all subshapes are known to produce an equivalent shape in spite of differences in transformation. An example of this is shown in figure 6 where the application of the rule in figure 3 will produce the same result regardless of which transformation is selected.

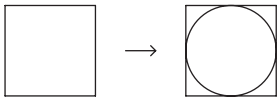

Figure 3. Schema rule that finds a square and places a circle in the middle of it.

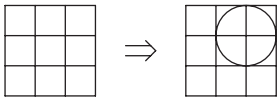

Figure 4. Derivation of the rule in figure 3 when the apply-mode is user. Here the user has selected the upper right-hand square.

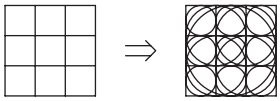

Figure 5. Derivation of the rule in figure 3 when the apply-mode is parallel. There are 14 different squares in the drawing.

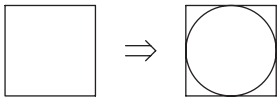

Figure 6. Derivation of the rule in figure 3 when the apply-mode is any. All possible transformations produce the same result.

2.2 Descriptor Phase

The *descriptor* phase has descriptions that provide additional matching constraints for the determination of a subshape. Traditionally the constraints on finding a subshape have been limited to the use of labeled points or lines along with parametric variations. The meta-language provides additional matching constraints based on the visual properties of the subshapes found to supplement the use of labels. There are two meta-language descriptions in the *descriptor* phase: maxline and zone.

2.2.1 Maxline

The maxline description adds an additional constraint that the matching subshape line must be a maximal line. A maximal line is a line that can not be embedded in another line. For example, figure 7 shows a rule where the left-hand schema is a square composed of maxline lines. By using this rule only one subshape, the outer square, will be found because of the restriction that all the lines must be maximal lines. A parallel application of the rule without the maxline description would result with a circle in all 14 possible squares as shown in figure 5. The use of the maxline description therefore allows the rule to differentiate the outer square from the inner squares.

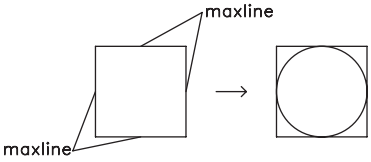

Figure 7. Rule where the left-hand shape is composed of lines with the maxline description.

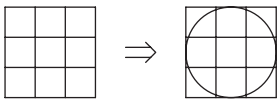

Figure 8. Derivation of the rule in figure 7 on a 3x3 grid. Only the larger outer square is possible for selection.

2.2.2 Zone

The zone description demarcates an area and associates that area with a predicate function. The demarcated area is defined relative to the schema and the associated function can be any function as long as the return value is true or false. One simple and useful function is the void function which returns true if the demarcated zone is void of any shape. Other computational formalisms, such as structure grammars (Carlson and Woodbury 1992), have also used the concept of a void. The use of the void function enables the rule to detect empty spaces relative to the subshapes found.

For example, suppose the developer of a grammar wants a rule that will pick out only the 9 smaller squares in a 3x3 grid. Using the rule in figure 3 will find 14 different types of squares. In order to find only the 9 smaller squares, we use the rule in figure 9 which uses the void zone to demarcate that the area inside the square must be empty. The use of the void zone therefore differentiates the smaller squares from the larger squares because whenever a larger square is selected, the void zone constraint rejects the subshape since there are lines in the interior of the square. A parallel application of the rule is shown in Figure 10.

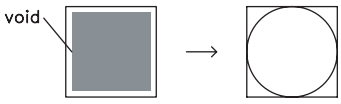

Figure 9. A rule that finds a square such that the inside of the square is void of shapes.

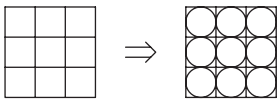

Figure 10. The derivation of the parallel application of the rule in figure 9.

The void zone can also be useful as an overlap detector. Cagan and Mitchell (1993) show an example of the shape annealing technique with the half hexagon grammar. One of the requirements in the example is that there are no overlapping pieces. Their solution was to test the result of each rule application to determine if the rule created an overlap. This step in the algorithm could be avoided by using the void zone description in the shape rule to insure that the area where the shape is being added does not have a half-hexagon piece. By incorporating the void restriction into the shape rules it is guaranteed that each rule will produce the appropriate result.

2.3 Parameters Phase

The *parameter* phase determines what parametric values are acceptable in order to have a subshape match in the drawing. Additional criteria can be specified by restricting the parametric values to fall within a specified range such as greater than zero and less than 10 (10>x>0). There are no meta-language descriptions in this phase.

2.4 Transformations Phase

The *transformation* phase determines what transformations are acceptable in order to have a subshape match in the drawing. The transformations include rotation, reflection and scaling. Additional criteria can restrict the set of transformations and their parametric values such as all transformations except the reflection transformation. There are no meta-language descriptions in this phase.

2.5 Context Phase

The *context* phase determines what portion of the drawing to apply the rule. This is achieved through descriptions that construct a new perception of the drawing which isolates the desired working area. The design process has been characterized as a see-move-see cycle (Schon 1983, 1992). The designer sees the drawing as something and evaluates the design to make a move. This in turn generates a new drawing which can be re-interpreted and re-evaluated to make new moves. The see-move-see concept has been used in artificial intelligence to develop design systems (Papazian 1993).

The see-move-see cycle suggests that before applying a rule, the drawing may be interpreted in a different manner. How the drawing is perceived is dependent upon the context or design domain of the rule, which is dictated by the developer of the grammar. The goal of the *context* phase is to view the drawing in such a way as to isolate the parts of the drawing that are of interest to the rule. Two methods for achieving this are information filtering, which filters out unnecessary shapes from the drawing, and attention, which focuses the area where a rule should apply. The corresponding descriptions are label-filter and focus.

2.5.1 Label-Filter

The label-filter description, as its name suggests, is to filter out any labeled line that is not part of the left-hand shape in a rule. For example, the rule in figure 11 uses a void

zone description so that only the smaller squares are selected along with the label-filter option on. Applying this rule, without the label-filter, to the first image in figure 12 will not work because of the grey overlapping grid. The void zone will return false every time because of the other grid pattern. But by using the label-filter option, the other grid pattern is filtered out and the rule can apply as shown in figure 12 where the user has selected the middle right square.

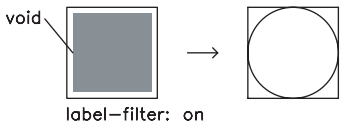

Figure 11. A rule that uses the label-filter to filter out irrelevant lines. The shaded area is the void zone description.

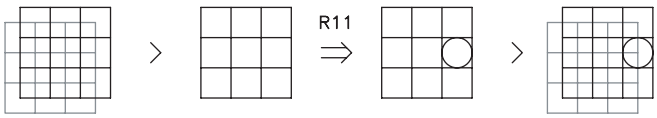

Figure 12. An example derivation showing the effects of the label-filter. The greater-than symbol signifies the effects of the meta-language description. When the rule from figure 11 is applied, the label-filter removes all the grey lines from the drawing. The user then selects the middle right square. After the rule is applied, the label-filter effect is removed which brings back the grey lines.

2.5.2 Focus

The focus description is composed of labeled lines named “focus” which behaves like any labeled line but has special meaning. These focus lines are used to demarcate one or many areas of attention. The demarcated areas are polygons composed of focus lines. Any shape outside of the polygons is ignored. The focus lines therefore force subsequent rules to apply only on a subset of the entire drawing. Whenever a drawing has focus lines, a rule can apply to only those areas inside the focus lines. In order to apply the rule to the entire drawing, the focus lines must be removed.

For example, the derivation in figure 15 shows how the focus lines can be used to isolate an area of the drawing for subsequent rule applications. The derivation begins with the use of the rule in figure 13, which draws four focus lines around a square. In this case, the user has selected the lower left corner. After using this rule, any subsequent rule applied to the drawing will occur only in the lower left-hand corner within the focus lines. The rule in figure 9 is applied next in parallel to place four circles inside the focus area. The last step of the derivation uses the rule in figure 14 to remove the focus lines so that the next rule will apply to the entire drawing.

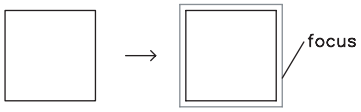

Figure 13. A rule that places focus lines around a square.

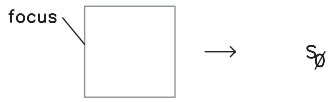

Figure 14. A rule that removes the focus lines.

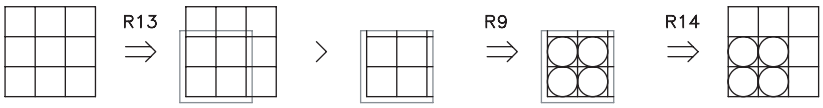

Figure 15. Example derivation showing the effects of the focus lines. The focus lines are shown in gray.

2.6 Control Phase

The *control* phase determines the flow of a grammar by manipulating the availability and sequencing of rules. This is achieved with two descriptions: rule-sets and directive. The directive is used to define an explicit sequence of rules which is called a macro. The rule-set description determines the availability of a set of rules at any point in time in the grammar. Sets of rules are typically associated with stages of a grammar.

2.6.1 Directive

The directive description dictates which rule to apply next depending upon the success or failure of the current rule to apply. There are two components to the description: success and failure. The success component describes which rule to apply next if the rule was successfully applied. The failure component describes which rule to apply if the rule failed to apply. This occurs when the left-hand shape of a rule does not exist in the drawing. The use of the directive allows the developer of a grammar to link a series of rules together to create a macro.

Suppose the design task is to generate a series of walls using an underlying grid pattern as the centerline. One method to achieve this effect is to offset the grid lines half the thickness of the walls and then trim off lines at the intersections where the walls join. The derivation for such a task is shown in figure 17. To produce this derivation, the four rules in figure 16 must be applied in order.

Each rule is linked to the next rule through the use of the directive. Rule R01 is applied first then rule R02, R03, and R04. Since rule R03 and R04 are subshapes of R02, it is possible for rule R03 and R04 to apply in situations where rule R02 should have been applied instead. For this reason, if the rules are applied in a different order, the results could potentially be incorrect and produce an invalid design state (Liew 2002). The appropriate derivation of the four rule macro is shown in figure 17.

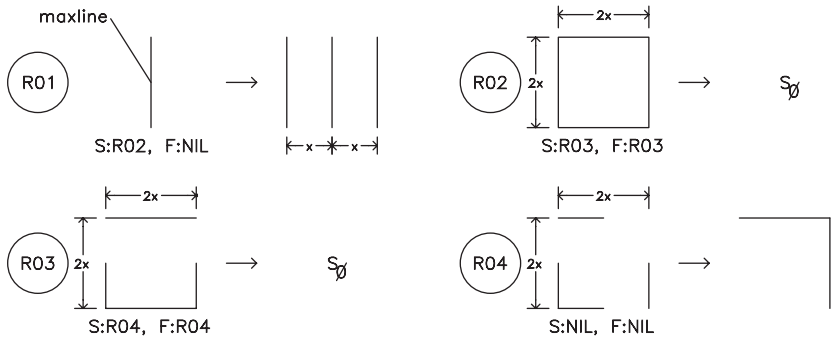

Figure 16. Four rules linked together using the directive description. The success rule is denoted with an "S" and the failure rule is denoted with an "F". Each rule is applied using the parallel apply-mode. Rule R01 offsets a pair of lines by distance x. R02 trims off the excess lines that occur in a cross intersection. R03 does the same for T intersections. And R04 trims and connects lines in an L intersection.

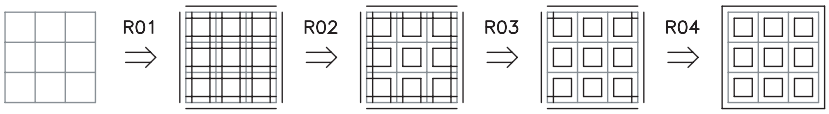

Figure 17. Derivation as a result of applying the rules in Figure 16.

2.6.2 Rule-Sets

The primary use of rule-sets is to allow the developer of a grammar to define a salient group of rules for the user in a particular stage of the grammar. The group of rules defines a subset of all the rules in the grammar. There are three descriptions; set-rule, add-rule, and sub-rule. Set-rule defines a set of rules that are available. Add-rule inserts additional rules into the rule-set. And sub-rule removes rules that exist in the rule-set. These descriptions are associated with a schema rule. Applying a rule with a rule-set definition, changes what rules are available to the user. Rule-sets are also a means of preventing the user from applying the secondary rules in a macro defined by the use of the directive. This is achieved by including only the first rule of a macro in a rule-set. The rule-sets also allow the developer of a grammar the flexibility to reuse rules from different groups without having to rewrite an entirely new rule.

An example is shown in figure 18 which shows a grammar composed of 15 rules. Some of these rules are linked together using the directive to create macros. There are two macros in this grammar, one composed of three rules and another composed of four rules. Suppose the developer wants to divide the rules into 3 stages. This can be achieved through the use of the rule-set description. Figure 19 shows the rules grouped into 3 sets. The first set is composed of 6 rules where only 4 of the rules are made available to the user. The other two rules, R04 and R05, are secondary rules of a macro and are accessible only from the primary rule, R03. The second set is composed of 7 rules and also uses the rule-set to prevent the user from choosing secondary rules in a macro. The third set is composed of 6 rules where some of the rules are taken from previous sets. To go from the first stage to the second stage, rule R06 would use the set-rule definition to define a new set composed of rules R07, R08, R12, and R13. The same technique would apply to rule R13 to go from the second stage to the third stage.

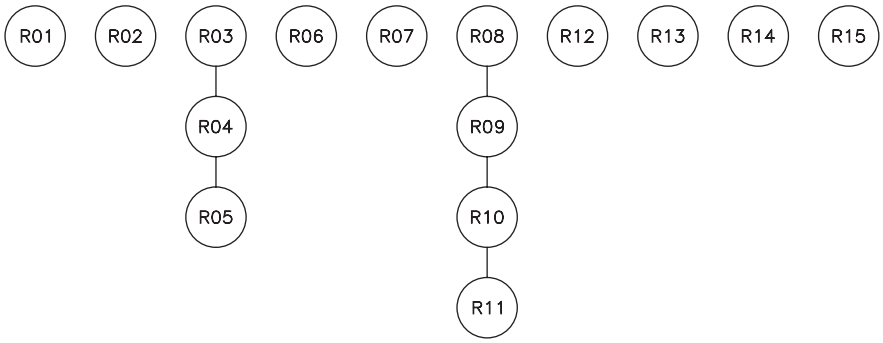

Figure 18. A grammar composed of 15 rules. Some of the rules are linked together using the directive to create macros such as rule R03, R04, and R05.

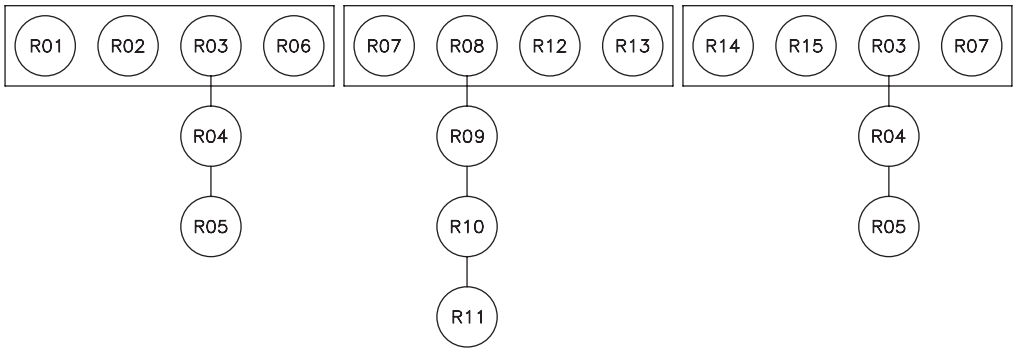

Figure 19. The grammar is grouped into 3 stages. At each stage, the user can choose from 4 rules. Some of the rules in the third stage are reused from other stages.

3.0 Putting It All Together

I have described characteristics of the six phases in the rule application process and their corresponding meta-language descriptions. Three of the phases, *transformation*, *parameter*, and *application* are based on the original formulas (1-4) which are concerned with the mechanics of applying a rule. The new phases, *control*, *context*, and *descriptor* complement the original formula by addressing the decision making process when applying a rule. The phases can be incorporated together to generate a new set of formulas.

The *descriptor* phase provides a means to include additional matching constraints on a shape within the rule. Typically, the matching constraint is only determined by the transformations, the parameters, and the parts relation. The additional matching constraint is a predicate function which returns true or false. This changes the original formula as follows:

$$t(g(p(A))) \leq C \qquad (5)$$

where p() is the additional predicate function. The meta-language has two descriptions in this phase; maxline, which determines if the shape is a maximal line, and zone, which associates a predicate function, such as void, with the shape.

The second phase is the *context* phase. The goal of the *context* phase is to isolate a portion of the drawing for rule application. This is achieved by hiding elements or areas of the drawing. The *context* phase is composed of functions that “see” the

drawing in different ways. This function can be incorporated into the original formula in the follow manner:

$$t(g(A)) \leq s(C) \quad (6)$$

where s() is the function that changes what the rule sees in the drawing. The meta-language has two descriptions in this phase; label-filter, which filters out labeled lines, and focus, which restricts the area where the rule can apply.

Putting the two modifications together we get the following formulas:

$$t(g(p(A))) \leq s(C) \quad (7)$$

$$C' = s(C) - t(g(p(A))) + t(g(B)) \quad (8)$$

where p() is associated with the descriptor phase and s() is associated with the context phase. And finally, to have sets of possible subshapes:

$$\text{For all t and g such that } t(g(p(A))) \leq s(C) \quad (9)$$

To apply the entire set of subshapes to drawing C:

$$C' = \Sigma\ (s(C) - t(g(p(A))) + t(g(B))) \quad (10)$$

The shape grammar meta-language provides seven descriptive conventions which are based on the six phases of the rule application process in the shape grammar formalism. The phases are an extension of the original formalism and the descriptions presented are an alternative method of using shape grammars for design.

4.0 References

Agarwal M, Cagan J: 1998, A blend of different tastes: the language of coffeemakers, *Environmental and Planning B*, 25, pp. 205-226.

Cagan J, Mitchell M J: 1993, Optimally directed shape generation by shape annealing, *Environmental and Planning B*, 20, pp. 5-12.

Carlson C. and Woodbury R.: 1992, Structure grammars and their application to design, *in* D.C. Brown, M. Waldron, and H. Yoshikawa (eds), *Intelligent Computer Aided Design*, Elsevier Science Publishers, Amsterdam, pp. 107-132.

Chase S: 2002, A model for user interaction in grammar-based design systems, *Automation in Construction*, 11(2), pp. 161-172.

Koning H, Eizenburg J: 1981, The language of the prairie: Frank Lloyd Wright's prairie houses, *Environmental and Planning* B, 8, pp. 295-323.

Li A: 2001, A *Shape Grammar for Teaching the Architectural Style of the Yingzao Fashi*. PhD Dissertation, Department of Architecture, Massachusetts Institute of Technology, MA.

Liew H: 2002, Descriptive Conventions for Shape Grammars, ACADIA 2002 Conference Proceedings, pp. 369-382.

Papazian, P: 1993, Incommensurability of Criteria and Focus in Design Generation, *in* U. Flemming and S. Van Wyk (eds), *CAAD Futures '93*, Elsevier Science Publishers, Amsterdam, pp. 111-125.

Schon D: 1983, *The Reflective Practitioner: How Professionals Think in Action*, Basic Books, New York.

Schon D.A, and Wiggins, G.: 1992, Kinds of Seeing and Their Functions in Designing. *Design Study*, Vol.13, No.2, pp.135-156.

Stiny G: 1977, Ice ray: a note on the generation of Chinese lattice designs, *Environmental and Planning B*, 4, pp. 89-98.

Stiny G: 1980, Kindergarten grammars: designing with Froebel's building gifts, *Environmental and Planning B*, 7, pp. 409-462.

Stiny G: 1990, What is a design?, *Environmental and Planning B*, 17, pp. 97-103.

Stiny G: 1991, The Algebras of Design, *Research in Engineering Design*, 2, pp. 171-181.

Stiny G, Mitchell W: 1978, The Palladian Grammar, *Environmental and Planning B*, 5, pp. 5-18.

This is a copy of the paper:
Liew, Haldane (2003) **SGML - A Shape Grammar Meta-Language**, Digital Design [21th eCAADe Conference Proceedings / ISBN 0-9541183-1-6] Graz (Austria) 17-20 September 2003, pp. 639-648