EXTENDING SHAPE GRAMMARS WITH DESCRIPTORS

HALDANE LIEW
Massachusetts Institute of Technology, USA

Abstract. One of the problems in using shape grammars is being able to describe succinctly and precisely the conditions necessary to perform the appropriate transformation in a drawing. This paper proposes new descriptors in shape grammars that can control rule selection and the matching conditions between a schema and a drawing. The framework for the descriptors is based on the decision making process when applying a rule. This process is divided into six phases: rule selection, drawing state, parameter requirements, transformation requirements, contextual requirements, and application method. The characteristic of each phase is described and some examples are presented.

Introduction

Using the descriptors introduced in this paper, the author of a shape grammar can (1) explicitly determine the sequence in which a set of rules is applied, (2) restrict rule application with a filtering process, and (3) use context to guide the rule matching process. The framework for developing the new descriptors is based on the rule application process. There are four basic actions to apply a rule: rule selection, drawing state, matching conditions, and application method. These four actions can be expanded into six phases specific to shape grammars as show in (figure 1). The descriptors in each phase are characteristic of their respective phases.

The first step in applying a rule is to determine which rule to apply. The rule selection phase controls the derivation of the grammar by determining the availability and sequencing of rules. This phase has two descriptors: directive and rule-set. The directive controls what rule should be applied next based upon the success or failure of the given rule to apply in the drawing. The rule-set descriptor uses a parallel description to determine what set of rules is available to the user at any point during the derivation of the grammar.

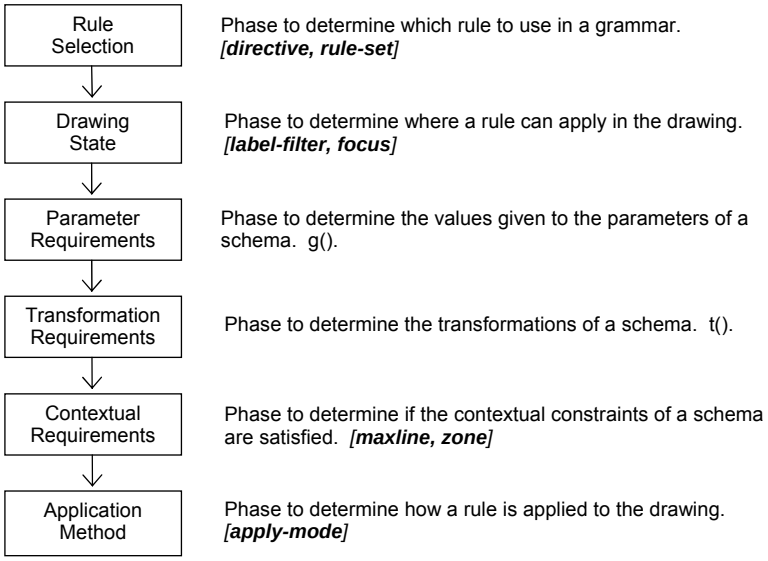

Figure 1. The six phases of the rule application process. The corresponding descriptors for each phase are shown in italics.

Once a rule is selected, the user has to determine where to apply the rule. The drawing state phase determines what portions of the drawing can be used for rule application. The two descriptors in this phase are label-filter and focus. Label-filter controls which labeled shapes are applicable based on the labeled shapes in the left-hand schema of a rule. The labeled shapes in the drawing that are not part of the left-hand schema are temporarily removed. The focus descriptor restricts the application of a rule to specific areas of the drawing demarcated by a special focus labeled polygon. A rule can only apply to the areas inside the polygons.

The next three phases, parameter requirements, transformation requirements, and contextual requirements, combine together to determine the matching conditions for finding a subshape in the drawing. In order for there to be a match, the requirements of all three phases must be fulfilled. The parameter requirements phase determines what values are assigned to the parameters of a schema in order to match a subshape in the drawing. The transformation requirements phase determines what combinations of transformations are necessary to match a subshape in the drawing. There are no new descriptors in these two phases.

The contextual requirements phase adds an additional constraint that determines if the context of the subshape satisfies a predicate. This phase has two descriptors: maxline and zone. The maxline descriptor specifies that a line in the subshape has to be a maximal line in the drawing. The zone

descriptor associates an area of the schema with a predicate function which can be used to test portions of a drawing relative to the subshape. A commonly used predicate function is the void function which tests if the area relative to the subshape is void of all shapes.

The application method phase is the last phase in the rule application process. This phase determines how a set of subshapes can be applied to the drawing. Typically, the user applies the rule to one selected subshape, but there are other possibilities such as a parallel application where the rule applies to all the subshapes in a drawing. The only descriptor in the application method phase is apply-mode.

There are descriptors for every phase except the parameter requirements and transformation requirements phases. The other phases have one or more descriptors for a total of seven new descriptors: directive, rule-set, label-filter, focus, maxline, zone, and apply-mode. The rest of this paper will provide details of each phase and any corresponding new descriptors. The explanations will start in reverse order from the application method phase to the rule selection phase.

Application method phase

The application method phase determines how a set of subshapes is applied to a drawing. There can be many different subshape matches between the left-hand schema of a rule and the drawing. Typically only one subshape is used but there are other possibilities such as a parallel application which would apply all subshapes at the same time. The descriptor, apply-mode, allows the author of a grammar to specify what type of application a rule should have.

APPLY-MODE

The apply-mode descriptor has three options named single, parallel, and random. The single option allows the user to select one of the subshapes for application. Figure 2B shows the selection of a single 2x2 square in the upper right hand corner using the rule in figure 2A.

The second option is parallel. This option will apply the rule to all possible subshapes in the drawing. In a 3x3 square grid, there are 14 different square subshapes. A derivation showing the effects of a parallel application using the rule in figure 2A is shown in figure 2C.

The third option is named random. With this option, any subshape can be applied to the drawing. The subshape can be randomly selected by the user or if used in conjunction with a shape grammar interpreter, a computer can randomly select the subshape (Chase 1989, 2002). This option can also be used when all subshapes are known to produce an equivalent shape in spite of differences in parameters and transformations. An example of this is

shown in figure 2D where the application of the rule in figure 2A will produce the same result regardless of which transformation is selected.

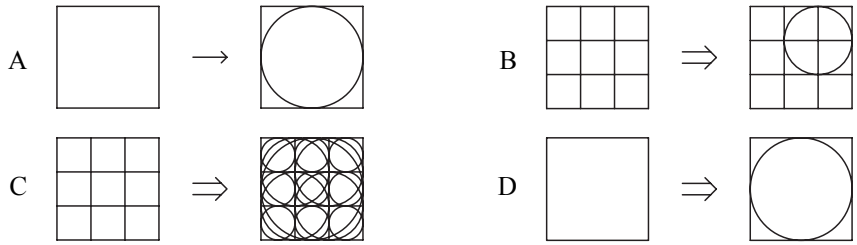

Figure 2. (A) Rule that finds a square and places a circle in the middle of it. (B) The derivation of the rule when the apply-mode is single. Here the user has selected the upper right-hand square. (C) The derivation of the rule when the apply-mode is parallel. A circle is placed in all 14 squares. (D) The derivation of the rule when the apply-mode is random. All possible matches produce the same result.

Contextual requirements phase

The matching conditions between a schema and the drawing are traditionally determined by the parameter function g() and the transformation function t() which correspond to the parameter requirements and transformation requirements phases. The third phase, contextual requirements, provides additional matching constraints for the determination of a subshape based on the context of the subshape in the drawing. These three phases, in combination, determine the matching conditions of a schema.

One of the difficulties in using shape grammars is that the matching conditions between the schema and the drawing works only on the shapes found. It is not easy to define a schema that uses the surround conditions of the subshape as part of the matching constraint. For instance, suppose the author of a grammar wanted to define a schema that found rectangles that were clear of any shapes on the inside.

This is difficult to define in the shape grammar language because there is no direct convention that states that the inside of a subshape found in the drawing must be empty of shapes. Instead such a condition could be defined using a series of compound rules that combine shapes along with a parallel description grammar (Knight 2003). The function in the parallel description grammar is the algorithm that would determine if the inside of the rectangle is void of any shapes. Other techniques include, adding additional geometries or labels, constraining the transformations, or varying the parametric values in order to isolate the desired condition.

Notice that these techniques complicate the situation and deviate from the simplicity of the original condition which is succinctly stated as finding a rectangle that is clear of any shapes on the inside. Instead of relying on compound rules with parallel description grammars or using transformation and parameter restrictions, the descriptors in the contextual requirements phase rely on the visual properties of the subshapes. Determining if the inside of a rectangle subshape is empty is a visual property of the subshape relative to the drawing.

There are two descriptors in the contextual requirements phase: maxline and zone. Maxline constrains a line in the subshape to be a maximal line. The zone descriptor evaluates a predicate function against an area of the drawing defined relative to the subshape. This method allows subshapes in a drawing to assess the surrounding conditions. The zone descriptor can directly define a schema that finds rectangles with no shapes on the inside.

MAXLINE

The maxline descriptor adds an additional constraint that the matching subshape line must be a maximal line. By definition, a maximal line is a line that can not be embedded in another line. The use of the maxline descriptor allows the author of a grammar to specify that a line in the subshape is a line that is not a smaller portion of a larger line in the drawing.

For example, figure 3A 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, can be found because of the restriction that all the lines in the square must be maximal lines (figure 3B). A parallel application of the rule without the maxline descriptor would result with a circle in all 14 possible squares as shown in figure 2C. The use of the maxline descriptor therefore defines a schema that differentiates the larger outer square from the smaller inner squares in the drawing.

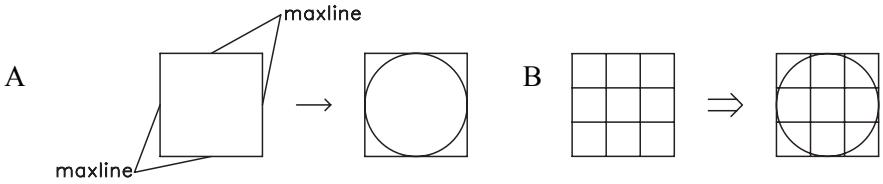

Figure 3. (A) Rule where the left-hand shape is composed of lines with the maxline descriptor. (B) The parallel application of the rule on a 3x3 grid. Only the larger outer square is possible for selection.

Another key use of the maxline descriptor is to define schemata that are determinate. A determinate schema is one that has a finite number of subshapes in a drawing. An indeterminate schema has an infinite number of

subshapes. A schema composed of only one line is indeterminate because there are an infinite number of lengths that can be embedded in a line. Often times, a determinate shape is desired to fix the number of possibilities. The maxline descriptor can be used to make a line a determinate shape.

The descriptors are based on the shape grammar language. Therefore, the same effect can be achieved without the use of the new descriptors. To describe a schema that finds a maximal line without the use of the maxline descriptor requires thinking about how to specify the same conditions using the transformation and parameter components. The maxline descriptor is the equivalent to restricting the scaling factor of the line to be at 100%. In other words, the line must match the entire line.

The descriptors in this phase can be considered shortcuts for the equivalent definitions in the shape grammar language. But more importantly the descriptors describe a rule by emphasizing certain visual conditions as opposed to emphasizing the descriptive methods of the shape grammar language. For the maxline descriptor it is the difference between seeing a line as a whole versus seeing the line as a scaling factor of 100%.

ZONE

The zone descriptor adds an additional constraint on the matching conditions of a schema in the form of a predicate function. With the use of the zone descriptor, not only must the geometry of the schema be embedded in the drawing, the predicate must also be true. The predicate function evaluates a marked area of the schema to determine if the predicate is true or false. By using the zone descriptor, the schema can use the context of the drawing in which the subshape is found as part of the constraints.

A commonly used function is the void function which states that the demarcated area must be void of any shape. The void zone enables the schema to detect empty spaces relative to the subshapes found in the drawing. Other computational formalisms, such as structure grammars (Carlson and Woodbury 1992), have also used the concept of a void.

The void zone can be used to differentiate subshapes in a drawing as demonstrated in figure 4. Suppose the author of a grammar wants a rule that will pick out only the nine smaller squares in a 3x3 grid. Using the rule in figure 2A will find all 14 different types of squares. In order to find only the nine smaller squares, the rule in figure 4A is used which has a void zone to demarcate that the area inside the square must be empty. A parallel application of the rule is shown in figure 4B.

The use of the void zone therefore differentiates the smaller squares from the larger squares. This occurs because whenever a larger square is selected, the void zone constraint rejects the subshape since there are lines in the interior of the square. In other words, the void predicate function returns

false. The void zone can also be used to distinguish other types of squares in the grid. To define a schema that selects only the corner squares, one could use the rule in figure 4C.

The difference between figure 4A and 4C is that the void zone in figure 4C has been enlarged to cover one corner of the square. A corner square has the characteristic that one corner of the square does not have any protruding lines. By expanding the void zone to cover one corner, the rule is guaranteed to select only the corner squares. A parallel application is shown in figure 4D. A similar approach is used to define a rule that finds only the edge squares as shown in figures 4E and 4F.

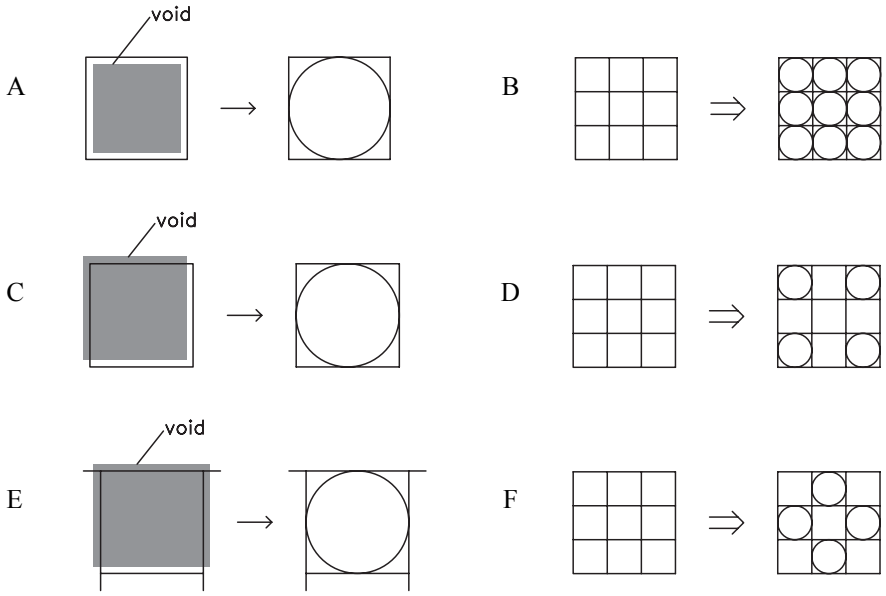

Figure 4. (A) A rule that finds a square such that the inside of the square is void of all shapes. (B) The parallel application of figure 4A. (C) A rule that finds a corner square. (D) The parallel application of figure 4C. (E) A rule that finds an edge square. (F) The parallel application of figure 4E.

Transformation requirements phase

The transformation requirements phase determines what transformations are necessary in order to have a subshape match in the drawing. This phase corresponds with the t() function. There are no new descriptors in this phase. The transformations include translation, rotation, reflection and scaling. Restrictions can be stipulated by specifying a value such as setting the scaling factor to be 100%, which is the equivalent of the maxline

descriptor, or by specifying the type of transformations to use such as all transformations except for reflection.

Parameter requirements phase

The parameter requirements phase corresponds with the g() function and determines what values are given to the parameters of a schema in order to have a subshape match in the drawing. There are no new descriptors in this phase. A common use of the g() function is to determine the parameters for the size of a shape. Restrictions can be placed on the parameters to match certain types of subshapes in the drawing.

Drawing state phase

The drawing state phase determines where a rule can and can not apply by altering the state of the drawing. Typically, the entire drawing is used to look for a possible subshape match between the left-hand schema of a rule and the drawing. But sometimes this is not desirable. For instance, when affecting changes on a drawing that only affect the walls of a building, it might be useful to alter the drawing so that only the walls are visible for manipulation. In another situation, maybe the design rules are concerned with affecting changes in only one room. It might then be advantageous to alter the drawing so that only the one room is visible for manipulation (figure 5).

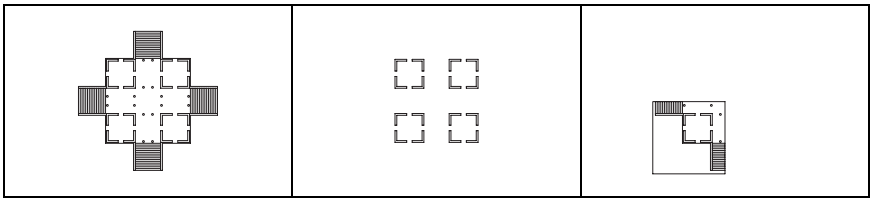

Figure 5. A plan drawing of a building by Durand (left) with alternative views where only the walls are visible (middle) and where only one room is visible (right).

The examples above all alter or "see" the drawing in a different context. If we are concerned with only walls, we see only walls. If we are concerned with seeing a particular room, we see only that particular room. This process of seeing a drawing as something else is part of the design process. Schön and Wiggins (1992) has characterized this seeing as a part of the cyclical see-move-see process in design. 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.

In a similar fashion, the modifications permitted by the drawing context phase gives a rule the ability to see the drawing as something else. What we see varies greatly and depends, in part, on the context in which the rule is applied. The ultimate goal of this phase is to view the drawing in such a way as to isolate the parts of the drawing that are important. A drawing can be quite complex and the drawing state phase is a means to manage that complexity by removing parts of the drawing that are not of interest at the time.

The descriptors in the drawing state phase provide two generic methods for altering the drawing. The first method is information filtering which filters out unnecessary shapes from the drawing. What shapes to filter out is dependant on the labeled shapes in the left-hand schema. This method has the same effect as looking at only a specific labeled shape, such as walls, in a drawing (figure 5 middle). The name for this descriptor is label-filter.

The second method is based on visual attention. When a person pays attention to a particular object in a drawing, they produce what is commonly called the searchlight of attention (Posner 1980). The location and scope of the searchlight determines what the person is focused on. This effect is abstractly mimicked using the focus descriptor. An area of focus is demarcated by placing an enclosed polygonal shape in the drawing. All areas outside of the polygonal shape are ignored. This method has the same effect as looking at a specific room in a drawing (figure 5 right).

LABEL-FILTER

The label-filter descriptor filters out any labeled shapes in the drawing that does not have the same labels as those used in the left-hand schema of a rule. In an architectural setting, the label-filter option could be used to filter out background information in a drawing. For instance, a drawing can consist of walls where the square grid pattern is used as the centerline (figure 6B). The black lines represent wall labeled lines and the gray lines represent the grid labeled lines. To select an empty room, the schema in figure 6A, which looks for a rectangular room that is void of shapes on the interior, could be used.

Unfortunately this schema will find no rooms in figure 6B because the void zone will detect the grid labeled lines whenever it finds a room. To fix this problem, the label-filter option in the schema is turned on (figure 6C). This has the effect of removing any labeled shapes in the drawing that are not part of the labeled shapes used in the schema (figure 6D). Since the schema does not have any grid labeled lines in it, the grid labeled lines in the drawing are temporarily removed before searching for an empty room. With the grid lines removed, the three empty rooms can now be selected.

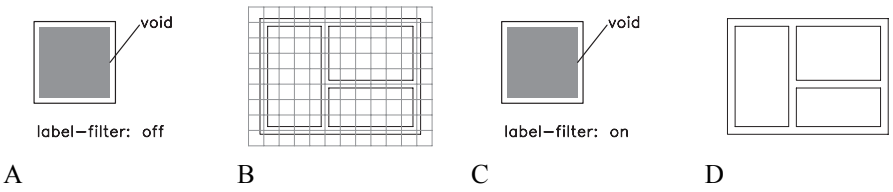

Figure 6. (A) A schema that looks for an empty rectangular room with the label-filter option turned off. (B) A drawing where the grid lines are used as the centerline for the wall. (D) A schema that looks for an empty rectangular room with the label-filter turned on. (C) A drawing showing the effects of the label-filter option. Only the wall labeled lines, shown in black, remain.

FOCUS

The focus descriptor controls what area or areas of the drawing can be used to find a subshape match. These areas are demarcated by enclosed polygons composed of a special labeled line named "focus". The focus lines can be altered and erased, just like any other labeled line and the enclosed polygons can be of any shape, concave or convex, as long as the lines do not intersect themselves or another focus polygon.

When a drawing contains an enclosed polygon composed of focus lines, all lines outside of the enclosed area are temporarily removed from the drawing. This temporary state is persistent until the focus lines are erased. Therefore, once a focus line is placed, all subsequent rules can only apply inside the demarcated areas. To apply a rule outside of the focus area, the focus lines have to be erased.

In an architectural setting, the focus lines can be used to define where changes can occur. For instance, suppose the task is to replace a wall with a series of columns at grid intersections (figure 7). The straightforward method is to write rules that transform walls of various sizes into a series of columns. The problem with this method is that you will need an infinite number of rules to accommodate the infinite number of possible wall and grid sizes.

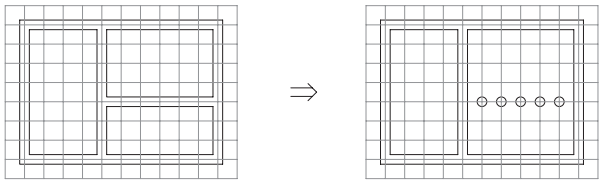

Figure 7. The derivation shows one wall of the floor plan transformed into a series of columns where the columns are placed at the grid intersections.

Another method is to develop a procedural series of rules that can vary according to the size of the wall and grid. A derivation of this process is shown in figure 9 using the rules in figure 8. The first step is to define the area where the columns will be placed. In this particular case, it is in a wall. We can use rule A1 to place a focus rectangle around the desired wall. The next step is to place columns at the grid intersections. This can be achieved by applying rule A2 in parallel. If the focus rectangle did not exist, then the rule would have applied to all the grid intersections in the drawing as opposed to only the grid intersections inside the focus rectangle. The last step is to replace the walls and remove the focus polygon so that subsequent rules can apply to the entire drawing using rule A3.

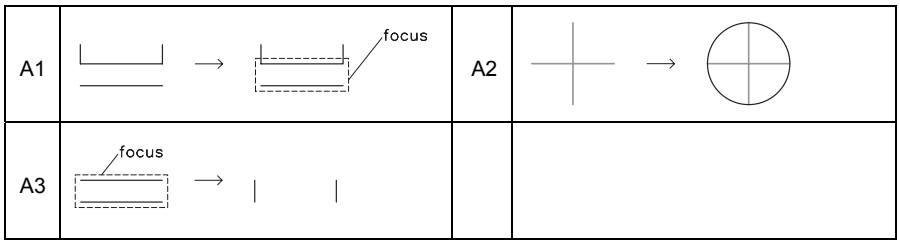

Figure 8. Rule A1 places a focus rectangle around a wall. Rule A2 places a column at a grid intersection. Rule A3 removes the focus lines and caps off the walls.

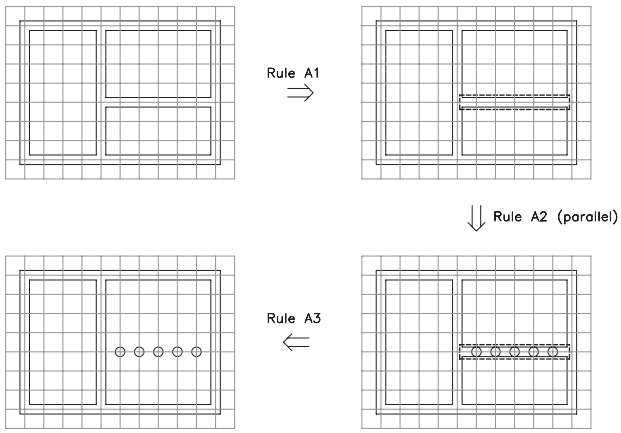

Figure 9. Derivation to replace a wall with a series of columns using the rules in figure 8. Rule A1 adds a rectangular focus to the selected wall. Rule A2 is applied in parallel to place a column at every grid intersection inside the focus area. The last rule applied is rule A3 which removes the focus lines and patches up the walls.

The three step process I have just described requires that the rules are applied sequentially and in the correct order. Typically this is achieved by

using state labels. The next section will show some alternative methods for controlling the execution of a grammar.

Rule selection phase

The rule selection phase determines the flow of the grammar by manipulating the availability and sequencing of rules. A grammar is composed of rules which can be placed into three general categories (figure 10). Of all the rules in the grammar, typically only a subset of the rules will have an effect on the drawing at any point during the derivation. This is the first category: applicable vs. inapplicable. The applicable rules can be further divided into constructive or destructive rules. Just because a rule can affect changes in the drawing does not necessarily mean it is the productive change for the design. The final category subdivides constructive rules into salient and deterministic rules (Li 2002). Salient rules provide the user of a grammar with design choices. Deterministic rules, on the other hand, are mechanistic rules used to complete a design transformation.

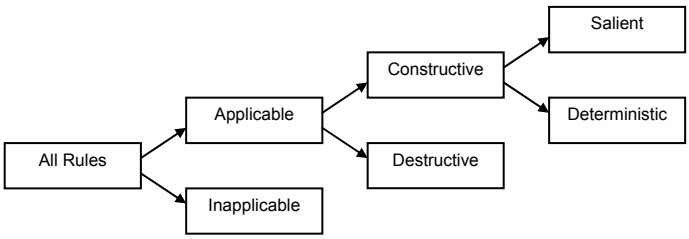

Figure 10. The three general categories of rules are applicable vs. inapplicable, constructive vs. destructive, and salient vs. deterministic.

Ideally, a grammar should provide the user with rules that are applicable, constructive and salient in nature. There are two descriptors in the rule selection phase towards achieving this goal: directive and rule-sets. The directive is used to define an explicit sequence of rules called a macro. The sequence is dependant upon the success or failure of a given rule to apply. The rule-set descriptor determines the availability of a set of rules at any point during the derivation of the grammar. Sets of rules are typically associated with stages of a grammar.

DIRECTIVE

The directive descriptor adds an additional component to the rule that dictates which rule to apply next depending upon the success or failure of the current rule to apply. There are two options to the directive descriptor: success rule and failure rule (figure 11). The success rule determines which rule to apply next if the rule was successfully applied. The failure rule

determines which rule to apply next if the rule fails to apply. This occurs when the left-hand schema of a rule does not exist in the drawing. The rule specified in the success rule or failure rule can be any rule in the grammar including itself.

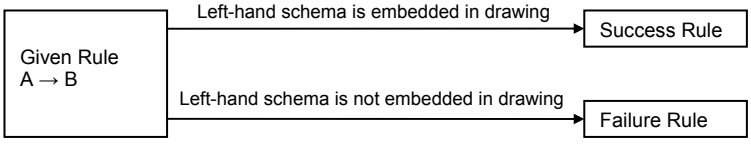

Figure 11. Diagram showing the components of the directive descriptor.

By using the directive descriptor, the author of a grammar is able to link a series of rules together to create a macro. A macro is composed of a primary rule and one or more secondary rules. The primary rule is the first rule in the macro. The secondary rules are the rules that succeed the primary rule.

An example of how the directive can be used in a grammar is shown in figures 12-14. Suppose the design task is to generate a series of walls using an underlying pattern as the centerline (figure 12). One method to achieve this effect is to offset the centerlines half the thickness of the walls and then trim off lines at the intersections where the walls join. This process can be achieved by applying the four rules in figure 13 in sequential order.

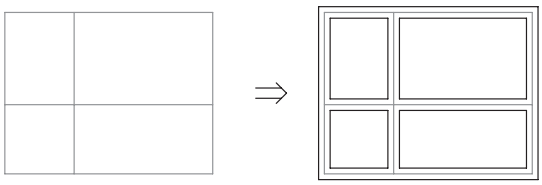

Figure 12. The derivation shows orthogonal centerlines (shown in gray) used as a basis to generate the walls (shown in black).

Each rule is linked to the next rule through the use of the directive. Rule R01 is applied in parallel first then rule R02, R03, and R04. R01 is the primary rule and R02, R03, and R04 are the secondary rules. 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 a dead-end state (Liew 2002). The derivation of the four rule macro is shown in figure 14.

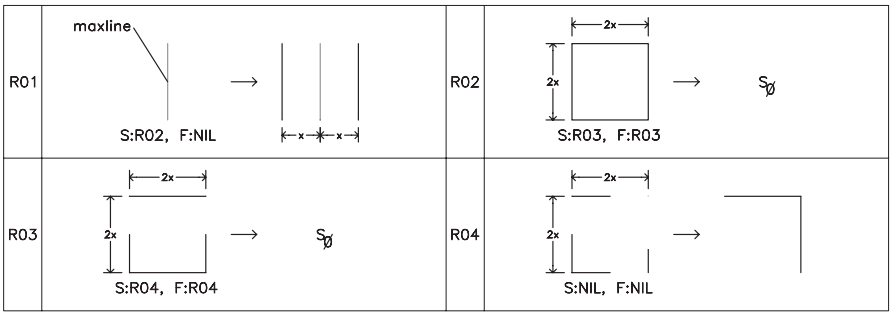

Figure 13. A macro composed of four rules linked together using the directive descriptor. Rule R01 offsets a pair of lines by distance x. R02 trims off the excess lines that occur in a cross intersection. R03 does the same for T intersections. And R04 trims and connects lines in an L intersection.

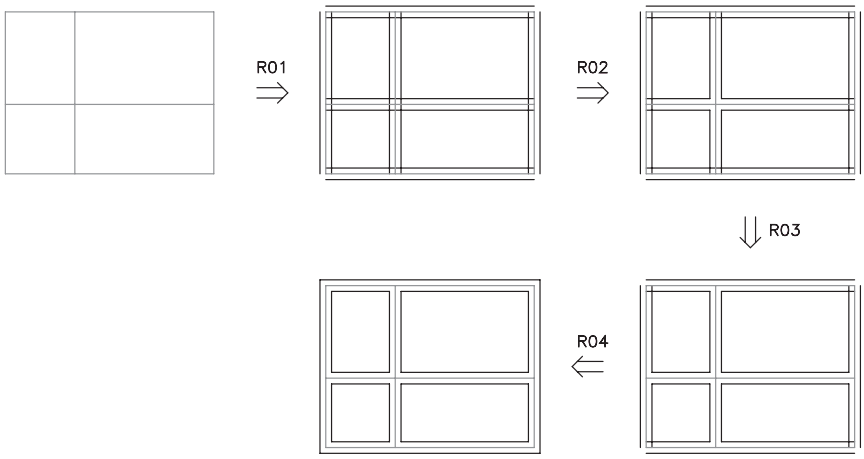

Figure 14. Derivation showing the result of applying the four rule macro shown in figure 13. All rules applications are parallel applications.

To achieve the same effect without the use of the directive descriptor requires the use of labeled points which act as state labels. Each rule would need to have a unique state label so that there is no confusion as to which rule to apply. For a grammar with few rules, this is a manageable task. But if there are hundreds of rules with hundreds of unique state labels, the grammar becomes daunting and unmanageable. The directive manages the complexity of having numerous state labels by placing the information of which rule to apply within the rule.

The main difference between using state labels and the directive descriptor are: (1) The mechanism for determining the next rule, when using the directive, is not in the drawing but in the rule; (2) The directive creates a

macro which can not be subverted by adding a rule with the same state label; (3) The directive has a failure component which allows an alternative rule to apply if a rule fails. The failure component can be used as a terminal case for recursive rules.

A macro is designed to start from the primary rule. If the user picks a secondary rule to apply, the macro will not necessarily work as intended. Often times, the author of a grammar wants to prevent the user from selecting the secondary rules of a macro because it can have negative effects on the design. One means of controlling this restriction can be accomplished by using the rule-set descriptor described in the next section.

RULE-SET

The rule-set descriptor provides the user of a grammar with a set of rules at any point during the derivation of a grammar. This is achieved with a parallel description that contains a group of rules called a rule-set. A rule-set usually corresponds to one stage of a grammar. A change in the rule-set is the same as going from one stage of the grammar to another. A rule is able to modify the rule-set with three control options: set-rule, which defines the set of rules that are available, add-rule, which inserts additional rules into the rule-set and sub-rule, which removes rules that exist in the rule-set. An illustrative derivation is shown in figure 15 where the rule-set of each step is complete different from the previous step.

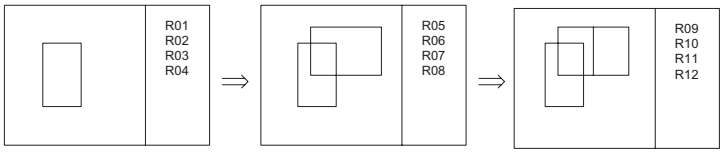

Figure 15. An illustrative derivation of a grammar showing the changes in the rule-sets. Each step is composed of a drawing on the left and the parallel rule-set description on the right. In the first step, the rule-set is composed of R01, R02, R03, and R04. In the second step, the rule-set changes to R05, R06, R07 and R08.

The appropriateness of the rules in the rule-set is determined by the author. Ideally, the rule-set should include only rules that are applicable, constructive and salient. By presenting the user with all viable options, the rule-set mechanism alleviates the need for the user to determine which rules can and can not apply at any stage of the grammar.

The rule-set descriptor helps the author to organize a complex set of rules when designing a grammar. The author has the flexibility to use the same rule in different stages of the grammar. He can also prevent the user from using a rule, even if it is applicable, by not including the rule in the rule-set.

This method is commonly used in conjunction with macros. By including only the primary rule of a macro in the rule-set, the author of a grammar can prevent the user from selecting the secondary rules.

Figure 16 is a diagram of a sample grammar that illustrates how rule-sets in combination with directives can be used to define and manipulate stages of a grammar. The diagram shows the first three stages of a sample grammar. In the first stage, the user has four options: R01, R03, R06, and R07. This stage also has two macros: R01+R02 and R03+R04+R05. Although the actual number of rules in the first stage is seven, the user can only select four of the rules. The rule-set descriptor acts as a filter to differentiate the salient rules from the deterministic rules of a macro.

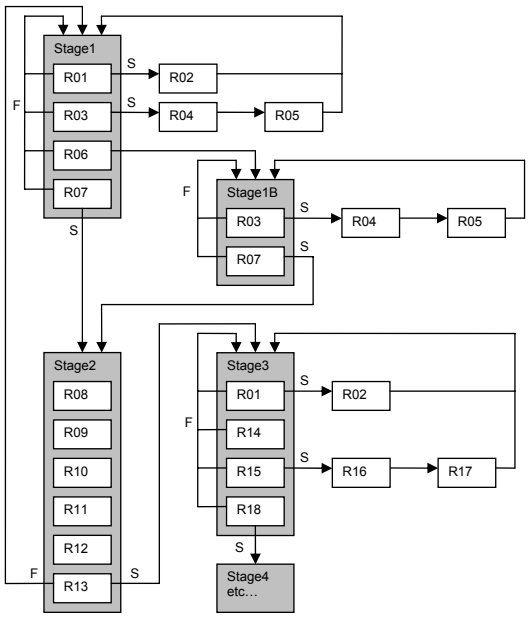

Figure 16. Diagram showing how stages of grammar can be defined using the rule-set and directive descriptors. Each stage created by the rule-set is represented by a gray box and the white rectangles inside are rules. The user can select any of the rules in the gray boxes. The lines connecting the rules together represent a directive link. A success rule is denoted by the letter "S" and a failure rule is denoted by the letter "F". If there is no letter then both the success and failure rules are the same.

Two of the rules in the first stage change the rule-set with the rule-set controls. Rule R06 uses the sub-rule control to remove rule R01 and R06 from stage1. This might occur when a rule affects some changes to the drawing which makes other rules no longer useful. Rule R07 changes the rule-set to go to stage2 which is composed rules R08 through R13. This can be achieved by using set-rule or combining both the sub-rule and add-rule

controls to define the new rule-set. Rule R13 changes the rule-set in stage2 to go to stage3 if it is applied successfully to the drawing. If rule R13 can not be applied to the drawing then the rule-set is changed to go back to stage1. Such a move might occur if certain conditions do not exist in the drawing and requires rules in another stage to implement the necessary changes.

The rule-set can also reuse rules from other stages of the grammar as shown in the stage3 of the diagram. Stage3 has seven rules but the rule-set only allows the user to select four of the rules. Two of the rules in the rule-set are the primary rules of a macro. One of the two macros, R01+R02, is a macro from stage1. The rule-set description allows a rule to be used in any rule-set without having to modify the rule.

The rule-set descriptor simplifies the process of modifying state labels in rules by providing a separate mechanism that explicitly groups rules together. The key difference between using the rule-set descriptor and using state labels is the parallel description in rule-sets that succinctly displays for the user what rules are available for application. Both methods are control flow mechanisms and may be combined together to obtain even more detailed control over the execution of a grammar.

Implications of the six phases

I have described characteristics of the six phases in the rule application process and their corresponding descriptors. Three of the phases, parameter requirements, transformation requirements, and application method are based on the original formulas (equations 1-2) which are concerned with the mechanics of applying a rule (Stiny 1980, 1990, 1991).

$$t(g(A)) \leq C \tag{1}$$

$$C' = C - t(g(A)) + t(g(B)) \tag{2}$$

The new phases, rule selection, drawing state, and contextual requirements complement the original formula by addressing the decision making process when applying a rule. These phases can be incorporated together to generate a new set of formulas.

Traditionally, the matching conditions between a schema and the drawing are determined by the parameters, the transformations, and the parts relation. The contextual requirements phase introduces an additional matching constraint based on a predicate function. In order for a schema to have a subshape match in a drawing, all three components must be true. The parameters and transformations must produce a shape that is embedded in the drawing and the predicate function must be true. This changes the original formula as follows:

$$p(t(g(A))) \leq C \tag{3}$$

Here p() is the additional predicate function of the contextual requirements phase. There are two descriptors in this phase: maxline and zone. The maxline descriptor tests to make sure the subshape lines used for embedding are maximal lines. The zone descriptor tests a predicate function against a demarcated area of the subshape in the drawing. One commonly used predicate is void which checks if an area of the drawing is void of all shapes.

The contextual requirements phase modifies the matching condition between the schema and the drawing by altering the schema. The drawing state phase, on the other hand, modifies the matching condition by altering the drawing. The goal of the drawing state phase is to isolate portions of the drawing for rule application. This is achieved by hiding elements or areas of the drawing. This phase is composed of functions that "see" the drawing in a different context. The "see" function can be incorporated into the original formula in the follow manner:

$$t(g(A)) \leq s(C) \tag{4}$$

Here s() is the function that changes what the rule sees in the drawing. The drawing state phase has two descriptors: label-filter and focus. The label-filter descriptor determines what elements of a drawing can be used for a subshape match. When the label-filter option is used, all labeled shapes in the drawing that are not in the left-hand schema of a rule will be removed. The focus descriptor, on the other hand, determines what areas of the drawing can be used for a subshape match. The applicable areas are defined by enclosed polygons marked by special focus labeled lines. Any shape outside of the enclosed polygons can not be used for a subshape match.

Putting the two modifications together we get the following formulas:

$$p(t(g(A))) \leq s(C) \tag{5}$$

$$C' = C - p(t(g(A))) + t(g(B)) \tag{6}$$

Here p() is associated with the contextual requirements phase and s() is associated with the drawing state phase. And finally, to have a set of all possible subshapes:

$$\text{For all t and g such that } p(t(g(A))) \leq s(C) \tag{7}$$

And to apply the entire set of subshapes to drawing C:

$$C' = \sum(C - p(t(g(A))) + t(g(B))) \tag{8}$$

The formulas mentioned so far deal with the mechanics of the rule application process. This includes the drawing state, parameter

requirements, transformation requirements, contextual requirements and application method phases. The rule selection phase does not apply to the above formulas because it deals with the overall control of the grammar.

The two descriptors in the rule selection phase are rule-set and directive. Rule-sets are an explicit mechanism for grouping rules in a grammar. The descriptor uses a parallel description to show the user which rules are available to choose from at different stages in the grammar. The directive is another control mechanism that dictates which rule to apply next depending upon the success or failure of a given rule to apply. Both descriptors are alternatives to the use of state labels which is the traditional method used in shape grammars to control the execution of the grammar.

The six phases of the rule application process is the framework for developing the new descriptors (figure 17). The characteristic of each phase deals with the decisions necessary to apply a rule. By developing descriptors that can manipulate those decisions, greater control is obtained over how a rule is selected and what the matching conditions are between the left-hand schema of a rule and a drawing. The descriptors provide mechanisms for rule selection, a method to incorporate context as part of the schema definition, and a means to filter information in a drawing. The set of new descriptors creates a type of meta-language for the shape grammar language (Liew 2003).

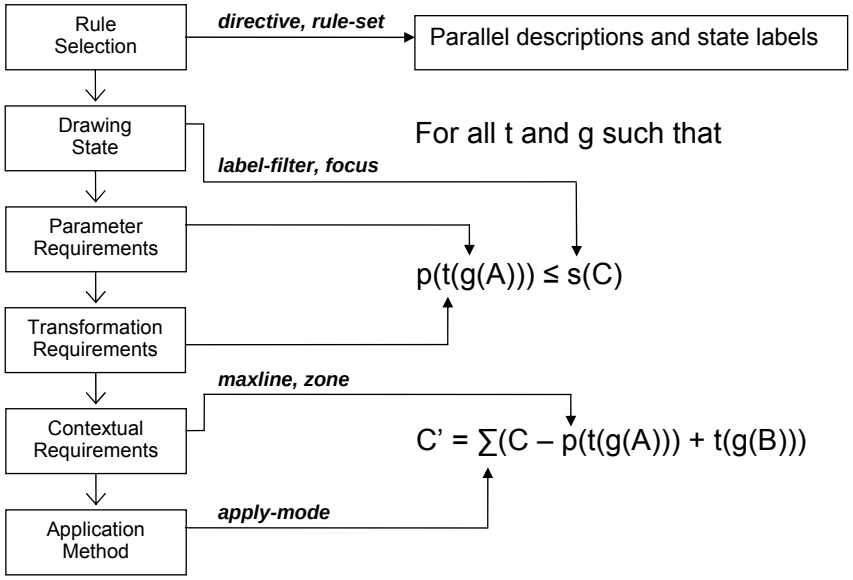

Figure 17. The diagram shows how the seven descriptors relate to the six phases of the rule application process and the new formulas in the shape grammar language.

References

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

Chase, S: 1989, Shapes and shape grammars: from mathematical model to computer implementation, *Environmental and Planning B: planning & design* **16**: 215-242.

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

Knight, T: 2003, Computing with emergence, *Environmental and Planning B: planning & design* **30**: 125-155.

Li, A: 2002, A prototype interactive simulated shape grammar, *in* K Koszewski and S Wrona (eds), *Connecting the Real and the Virtual - design e-ducation, Proceedings of the 20th Conference on Education in Computer Aided Architectural Design in Europe*, eCAADe, Warsaw, pp. 314-317.

Liew, H: 2002, Descriptive Conventions for Shape Grammars, *in* G Proctor (ed), *Thresholds - Design, Research, Education and Practice, in the Space Between the Physical and the Virtual, Proceedings of the 2002 Annual Conference of the Association for Computer Aided Design In Architecture*, ACADIA, Pomona, pp. 365-378.

Liew, H: 2003, SGML: A Shape Grammar Meta-Language, *in* W Dokonal and U Hirschberg (eds), *Digital Design, Proceedings of the 21st Conference on Education in Computer Aided Architectural Design in Europe*, eCAADe, Graz, pp. 639-647.

Posner, M: 1980, Orienting of attention, *Quarterly Journal of Experimental Psychology* **32**: 3-25.

Schön, DA and Wiggins, G.: 1992, Kinds of Seeing and Their Functions in Designing, *Design Study* **13**(2): 135-156.

Stiny, G: 1980, Introduction to shape and shape grammars, *Environmental and Planning B: planning & design* **7**: 343-351.

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

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

This is a copy of the paper (with corrections):

Liew, Haldane (2004) Extending Shape Grammar with Descriptors, in J. Gero (ed), *Design Computing and Cognition '04*, Kluwer Academic Publishers, Dordrecht, pp. 417-436.