A Generative Approach towards Performance-Based Design

Using a shape grammar implementation

Tiemen Strobbe[1], Ronald De Meyer[2], Jan Van Campenhout
Ghent University, Belgium
http://smartlab.elis.ugent.be
[1]Tiemen.Strobbe@UGent.be, [2]Ronald.DeMeyer@UGent.be

Abstract. *Due to a growing number of regulations and standards, building performance becomes equally important as traditional design drivers. Therefore, it is necessary to quickly explore design alternatives that meet these performance requirements. To support this complex design task, a rule-based design system is proposed that is founded on a shape grammar. This paper describes a graph-based implementation of this shape grammar that allows subshape detection, parametric rules and attributed shapes. The implementation described in this paper forms the basis to further investigate to what extent rule-based design systems can support a generative approach towards performance-based design.*

Keywords. *Shape grammar, evolutionary algorithm, performance-based design, implementation, generative design.*

INTRODUCTION

A growing number of regulations and standards to which nowadays buildings have to comply, has put design performance back on the architectural agenda. This has led to the emergence of a performance-oriented architectural design paradigm in which building performance (regarding sustainability, safety, accessibility, comfort, etc.) becomes equally important as traditional design drivers such as functionality, history or aesthetics (Kalay, 1999). In contrast to the traditional design process, in which performance issues are often dealt with in a post-engineering optimization phase, the performance-based design process takes into account the performance requirements in an early design phase. Recent technologic advances in computational design systems allow the integration of performance requirements and have led to an increased productivity of the design process (Petersen et al., 2010). More specifically, added programming possibilities allow the continuous generation and evaluation of parametric variations in order to select (sub-)optimal design solutions (Strobbe et al., 2012).

Such CAD-systems are often founded on a geometric representation of the design. However, the current increased emphasis on building performance in architectural design starts to question this central role of geometry in CAD. The designer has to work within the constraints of government rules and regulations to accomplish a good compromise from a wide range of design solutions. Such constraints are diverse in nature and often difficult to translate into a graphical or geometric form. Therefore, parametric modeling so far allows the generation of quite restricted geometric variations in the model. Furthermore, a geometric representation of design is inappropriate from a computational and automat-

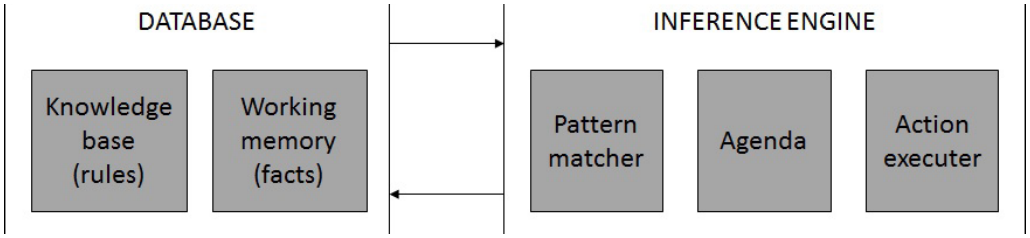

Figure 1
The rule production system consisting of the database and the inference engine.

ic reasoning point of view, as it is computationally expensive to identify features to be computed from a set-theoretic representation of a geometric object.

More appropriate in the context of supporting the design process, is the combination of a geometric representation and its rule-based representation. This allows the representation of the design through a sequence of design rules that preserve design information that otherwise has to be reconstructed. This representation is more suitable for computer implementation and can allow both (1) automated reasoning on the design and (2) the generation of a design grammar (i.e. family of designs) that extends restricted parametric variations. These functionalities can support the designer in different types of problem-solving activities by generating alternative or even sub-optimal design solutions within a grammatical paradigm.

Our research investigates to what extent rule-based design systems can provide essential support in the core design activities of architectural designers. This paper describes an implementation of such a rule-based design system, founded on a graph-based shape grammar. One possible graph-based design representation is described and it is discussed how a grammar of designs can be generated through the application of design rules.

SHAPE GRAMMAR IMPLEMENTATION

Rule production system

A rule-based design system founds on the collection and management of design 'knowledge' in form of rules, as is the case in rule production systems. A rule production system generally consists of a database and an inference engine (Figure 1). The database contains all the design knowledge in form or production rules (knowledge base), together with information about the current state or knowledge (working memory). A production rule is modeled as a transition between a 'before' and 'after' state. The left-hand side of the production rule describes all the preconditions that need to be satisfied, while the right-hand side describes the action of the rule execution. If the production rule's precondition matches the current state of the working memory, the production's action is executed.

The inference engine starts with the facts in the working memory and uses the matched production rules to update, remove or add new knowledge to the working memory. In order to select and execute the production rules, the inference engine contains a pattern matcher, agenda and action executer. Firstly, the pattern matcher uses an algorithm to collect production rules of which the preconditions are satisfied with the facts in the working memory. The collection of rules resulting from the matching algorithm is called the conflict set. Secondly, the agenda determines the resolution strategy of the conflict set, for example according to priorities assigned to the rules or according to the order in which the rules were written. Thirdly, the action executer performs the production rule's action and removes it from the agenda. This process is continuously iterated, resulting in different production system derivations.

Shape grammar

The use of production systems to provide some form of artificial intelligence is found useful in several scientific domains. In the domain of architectural design, shape grammars (Stiny et al., 1972), a class of production systems that generates geometric shapes, are used to capture design knowledge into shape rules. Shape grammars define a rule-based formalism to represent visual thinking and to handle ambiguities that are characteristic of architectural design. A key concept in the shape grammar formalism is the embedding relation that lies at the basis of handling ambiguities (Stiny, 2006). Embedding allows the recognition of emergent (sub)shapes that are not predefined in the shape grammar, but emerge from shapes generated by rule applications. Once a rule is applied, all the shape parts fuse and the new shape needs to be 'reframed' in order to proceed with the rule application. Therefore, embedding allows an open-ended way to generate multiple design derivations and to explore a design solution space towards novel or unexpected areas.

One of the current challenges is the computer implementation of shape grammars. Gips (1999) provides an overview of previous research efforts on shape grammar implementations and describes several issues and obstacles, of which most of them are still valid to date. These obstacles include the lack of parametric shape support and the subshape detection problem. The support of parametric shapes involves parameterized shape rules. The subshape detection problem is to determine whether a shape contains a specific subshape. The main challenge of the computer implementation is thus to find an underlying representation that is both able to support the visual nature of shape grammars and amenable to computer interpretation. In the remainder of this paper, an approach is described that uses a graph-based representation of the design (facts) and design rules, together with a graph transformation system (inference engine).

GRAPH-BASED REPRESENTATION

The graph-based representation of a design consists of pairwise relations between objects. In the domain of architectural design, graphs are mainly used to model topological configurations of architectural spaces or to describe geometric data structures. As an example of the former approach, Grasl (2012) describes a graph representation of a Palladian-style villa that consists of an adjacency graph for the grid underlying the design. While this approach is mainly useful to implement specific graph representations, the latter approach is more suitable for general implementations. For example, Heisserman (1994) introduces a boundary graph to represent three-dimensional solids. The graph nodes represent topological elements (such as vertex, edge, face, etc.) and the arcs represent the adjacencies between the elements. Geometric information (coordinates) is associated with each vertex node as an attribute. The boundary graph is based on the generalized split-edge data structure that is capable to store adjacency information of geometric objects. Heisserman's approach has been further investigated in several research efforts, for example the GRAMATICA three-dimensional interpreter (Correia et al., 2012) or the web-based implementation GRAPE (Grasl et al., 2011).

The main advantage of the graph-based implementation of shape grammars is that both parametric shape rules and subshape detection are supported (Grasl et al., 2011). Firstly, the ability to support parametric shape rules is caused by the topological nature of graphs. A graph can represent multiple geometric shape variations. For example, the graph representation of all quadrilateral shapes (square, rectangle, trapezoid, etc.) are isomorphic or topologically identical. Secondly, the ability to support subshape detection is based on the subgraph isomorphism problem (Ullmann, 1976). Subgraph isomorphism involves determining whether a graph contains a subgraph that is isomorphic to a given graph. While the subgraph isomorphism problem is proven to be NP-complete, Grasl et al. (2011) demonstrate that practical solutions can nevertheless be created. Thirdly, attributed graphs also contain non-topological information that is described in the form of attributes.

Approach

Based on Heisserman's approach (1994), the attrib-

Node Type	*Attributes*	*Type*
Vertex	ID	String
	x, y, z	Float
Edge	ID	String
	length	Float
Face	ID	String

Table 1
Node types and their attributes.

uted graph represents (geometric) objects as nodes, and relations between these objects as directed arcs. Several node types are distinguished that represent different objects (vertices, edges and faces) (Table 1). This approach can be extended to an unlimited number of nodes that represent different objects (for example: "*Solid*", "*Wall*", "*Door*", "*Window*", etc.). Similarly, several arc types are distinguished that represent different relations between the geometric objects, for example: "*hasVertex*", "*hasEdge*", etc. In addition, several attributes are associated with the graph objects in order to store non-topological information: unique ID's are associated with all nodes, coordinate geometry is associated with vertex nodes, etc.

The graph representation contains only topological information of the shape, which allows the support of parametric shapes. Additional attribute information (e.g. coordinate geometry) is needed to restrict the parametric shapes to specific geometric shapes. Therefore, an attributed graph-based representation ensures a unique mapping between the shape and the graph. An example of the graph representation of a geometric line object is given in Figure 2. The graph consists of two vertex nodes (white), one edge node (black), and two directed arcs from the edge node to the vertex nodes. The line is considered parametric, if the coordinate geometry attributes contain parametric values.

GRAPH RULES

Parallel to the graph-based representation of the design, the rule set can also be described using the same representation. This graph-based rule representation enables the collection and management of design knowledge with a far greater expressive power than pure data. Once a rule is described, it can be applied in different environments using an inference engine. Graph rules are described as a transition between two graphs, following a typical IF-THEN statement. The left-hand side of the graph rule describes the graph that needs to be matched to the host graph, together with additional conditional statements. These conditions include attribute conditions and negative application conditions (NAC). Attribute conditions define restrictions on the attributes of the graph, while NACs specify requirements for non-existence of sub matches. The right-hand side describes the transformation of the host graph. This graph transformation includes deleting or manipulating existing graph objects, creating new objects, and also performing computations on the object attributes.

As an example, the graph-based representation of a shape rule that generates a Koch curve is displayed in Figure 3. The Koch curve is a mathematical fractal curve that is constructed by recursively replacing a line segment with an equilateral triangle. Therefore, the left-hand side of the rule represents a line segment and the right-hand side represents a graph with both modified and new graph ob-

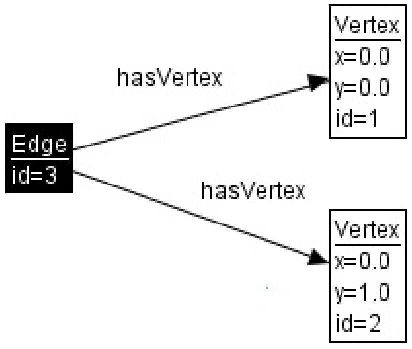

Figure 2
Graph representation of a non-parametric line object.

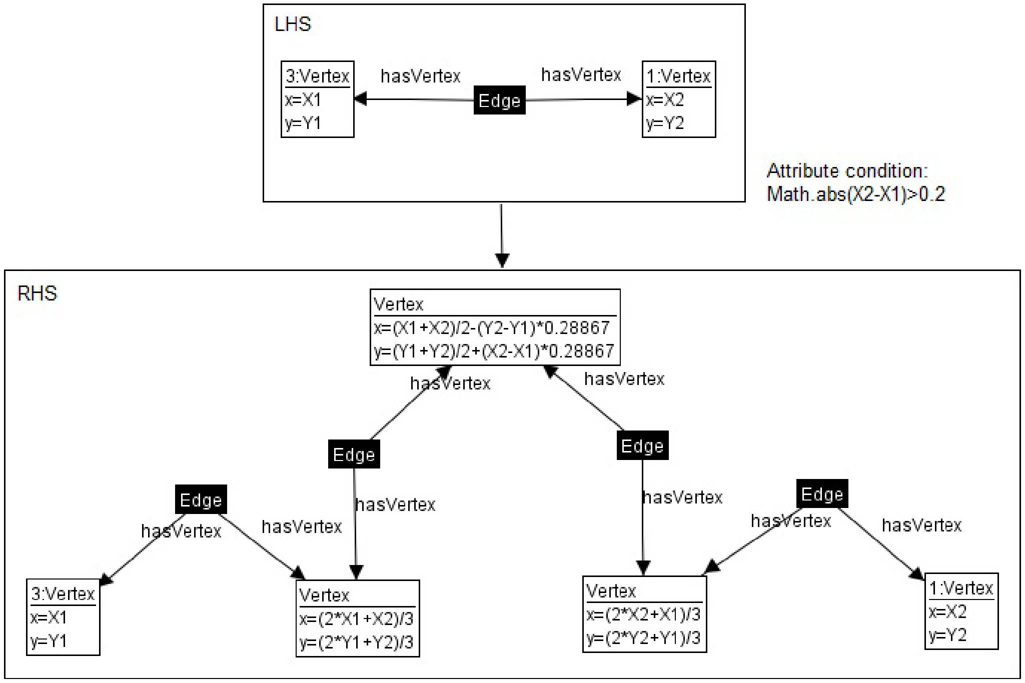

Figure 3
Graph-based representation of a shape rule that generates a Koch curve.

jects. The attributes of the new nodes are defined as expressions that take into account the attribute context of the matched host graph *(X1, Y1, X2, Y2)*. Furthermore, an additional attribute condition is implied in order to ensure a minimum length of the generated line segments.

GRAPH TRANSFORMATION SYSTEM

A graph transformation system is used to allow the stepwise application of graph rules on the original host graph. Among others, AGG is a general graph transformation system written in JAVA (Rudolf and Taentzer, 1998). Rule application is performed by matching the left-hand side of a graph rule to the host graph and replacing it using a preservation morphism. The AGG system solves the problem of graph matching, i.e. the subgraph isomorphism problem, as a constraint satisfaction problem (CSP). As indicated previously, this feature enables the detection of subshapes, which is an important challenge for shape grammar implementations. Also, the matching conditions are evaluated in order to recognize specific shape features. The preservation morphism describes the mappings of the objects of one graph to those of another, using tags to indicate the mapped objects. If multiple matches are found, all possible morphisms are calculated and stored. The selection of the morphism can happen non-deterministically or using a user-defined sequence (for example through priorities assigned with the rules). The user can go back and forth in this transformation process, and generate multiple alternatives.

In the following example, the initial line shape that is described in Figure 2 and the Koch rule that is described in Figure 3 are used to generate shape grammar transformations. Figure 4 shows several graph transformation steps in the generation process (step 0, 1, 2 and 10). Rule matching and selec-

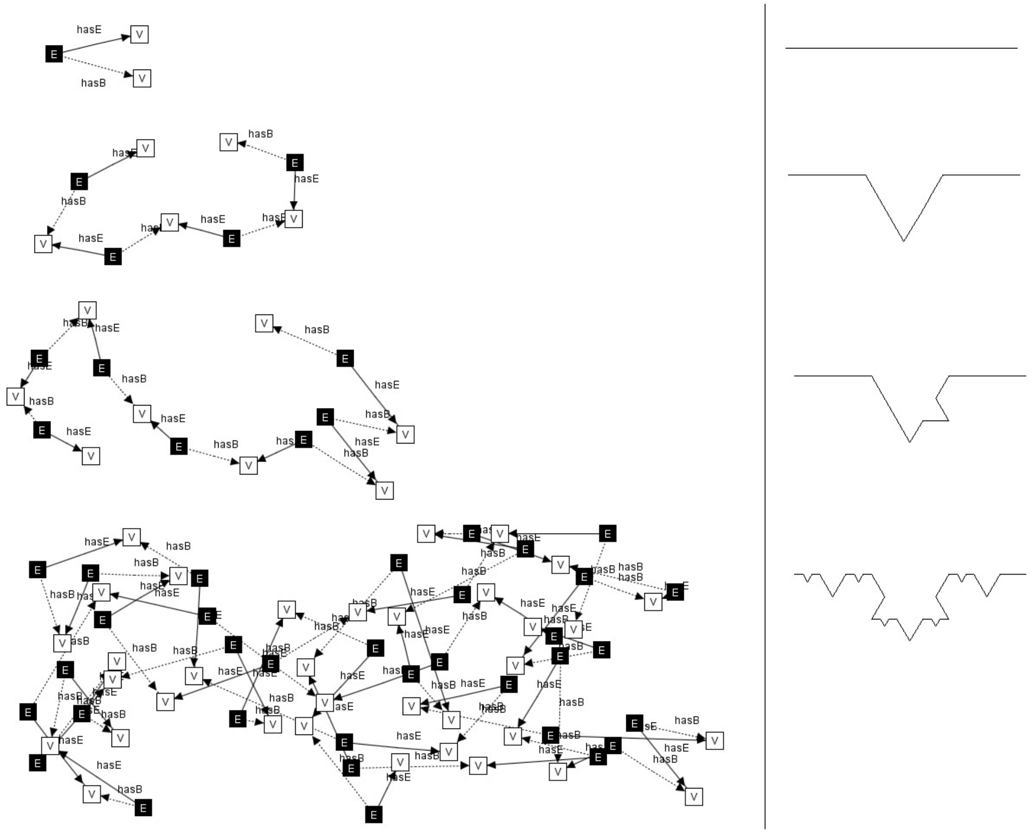

Figure 4
Graph-based (left) and shape (right) representation of several graph transformation steps (0, 1, 2, 10).

tion is performed non-deterministically, in order to allow a depth-first design exploration process.

CONCLUSION AND FUTURE RESEARCH

This paper describes the implementation of a rule-based design system using a graph-based representation of a shape grammar. An approach is proposed that allows subshape detection, parametric shape rules and attributed graph objects. Therefore, it is possible to quickly explore design alternatives that extend parametric variations. A simple case study has demonstrated the feasibility of this approach, however, a more complex design context is needed in order to test the practical value of this approach. In the case of performance-based design, further research is needed to determine which domain knowledge can be incorporated in the rules and whether heuristics are needed to go back and forth in the transformation process. Therefore, the work demonstrated in this paper forms the basis to further investigate to what extent rule-based design systems can provide support in the design process.

REFERENCES

Kalay, Y 1999, 'Performance-based design', *Automation in Construction*, 8(4), pp. 395-409.

Petersen, S and Svendsen S 2010, 'Method and simulation program informed decisions in the early stages of building design', *Energy and Buildings*, 42(7), pp. 1113-1119.

Strobbe, T, Pauwels, P, Verstraeten, R, De Meyer, R, Van Campenhout, J 2012, 'Optimization in compliance checking using heuristics: Flemish Energy Performance Regulations (EPR)', *eWork and eBusiness in Architecture, Engineering and Construction*, pp. 477-482.

Stiny, G, Gips, J 1972, 'Shape grammars and the generative specification of painting and sculpture', *Information Processing*, 71, pp. 1460-1465.

Stiny, G 2006, *Shape: talking about seeing and doing*, MIT Press, Cambridge.

Gips, J 1999, 'Computer implementation of shape grammars', *Workshop on Shape Computation*.

Grasl, T 2012, 'Transformational Palladians', *Environment and Planning B*, 39(1), pp. 83-95.

Heisserman, J 1994, 'Generative geometric design', *IEEE Computer Graphics and Applications*, pp. 37-45.

Correia, R, Duarte, J, Leitao, A 2012, 'GRAMATICA: A general 3D shape grammar interpreter targeting the mass customization of housing', *Proceedings of the 30th eCAADe Conference*, pp. 489-496.

Grasl, T and Economou, A 2011, 'GRAPE: A parametric shape grammar implementation', *SimAUD 2011 Conference Proceedings*, pp. 45-52.

Ullmann, J 1976, 'An algorithm for subgraph isomorphism', *Journal of the ACM*, 23(1), pp. 31-42.

Rudolf, M and Taentzer, G 1999, 'The AGG approach: language and environment', *Handbook of graph grammars and computing by graph transformation*, pp. 551-603.