Automation in Construction 94 (2018) 417–437

Contents lists available at ScienceDirect

Automation in Construction

journal homepage: www.elsevier.com/locate/autcon

An algorithmic design grammar for problem solving

Dan Hou[a,b], Rudi Stouffs[a,*]

[a] School of Design & Environment, National University of Singapore, 117566 Singapore, Singapore
[b] School of Architecture, Tianjin University, 300181 Tianjin, China

ARTICLE INFO

Keywords:
Shape grammar
Description grammar
Algorithmic structure
Layout generation problem (LGP)
Local reasoning

ABSTRACT

Shape grammars, in conjunction with description grammars, have potential for applications beyond stylistic design generation, such as problem solving. However, few case studies elucidate how to map design logic or knowledge onto the development and organization of rules. As such, the application process usually remains manual or controlled by an optimization algorithm. The aim of this study is to create a design grammar which can rely on logic embedded within or between rules to guide the derivation process towards an objective in an automated way. We propose a general approach to develop an algorithmic design grammar which allows logic to be embedded at varying complexity level. To improve the clarity of logic and automate the application of grammars, rules are structured in algorithmic patterns using sequence, selection and iteration. A layout generation problem is used as a case study for demonstration, and design grammars with different deduction strategies are compared regarding their applicability and limitations.

1. Introduction

Undoubtedly, computational techniques significantly enhance the efficiency, precision and creativity of design problem solving. These systematic methods show obvious superiority in dealing with complex problems, but the innate features of design, like multi-disciplinarity [1], open-endedness [2] and ill-structuredness [3], still pose great challenges for them.

Representation is one of the important topics in Simon's design curriculum [3], and it is always placed in the first phase of the design synthesis process, as the basis for the sequential generation, evaluation and guidance [4]. From the perspective of problem-solving, a good representation is a way of facilitating a problem formulation so that we can gain access to the acceptable solutions efficiently [2, 5]. It plays an important role in clarifying the design requirements and constraints, identifying the variation elements, determining the appropriate search and generation techniques, etc. Therefore, a representation has a great influence on the overall performance of a design synthesis system [6].

Generally, design representations could be classified into two types: the representation of an artifact being designed, and the representation of a process to achieve the design goals [2]. We denote these types as model-based and rule-based, respectively.

Adopting a model-based representation, the synthesis of a solution then, actually, means the assembly of values for every variant, so in a sense, it can be simply considered as an assignment problem. By contrast, when embracing a rule-based representation, a design solution gradually completes via the recursive application of rules. As such, a rule-based representation is not able to depict the explicit boundaries of the solution space since it is difficult to exhaustively predict the way the rules will apply corresponding to the changing context. However, a rule-based approach to problem solving is particularly beneficial when the application logic is dynamic (i.e., where a change in policy needs to be immediately reflected throughout the application) and rules are imposed on the system by external entities [7].

In fact, the main difference between both approaches to design synthesis lies in whether the state of the initial inputs is the same as that of the final outputs. Here the meaning of state is similar to the one used by Woodbury [8], referring to the abstract or concrete configuration. For a model-based approach, designers are supposed to provide the final state of solutions. Searching in this case is to eliminate those invalid solutions within that state space by judging if they violate the constraints. Compared to such a passive way, a rule-based approach actively traverses the paths to the valid solutions. These paths indicate the change of states, which are in fact implied in the rules or inferred by the rules. The accessibility to that incomprehensible state space via the partial and intentional representation in the rules [8] is the most distinctive function of a rule-based approach.

In this paper, we choose a rule-based approach, formalized as a design grammar, to solve a complex layout generation problem (LGP). For a grammatical approach, such as shape grammars, even when we

* Corresponding author.
E-mail addresses: houdan@tju.edu.cn (D. Hou), stouffs@nus.edu.sg (R. Stouffs).

https://doi.org/10.1016/j.autcon.2018.07.013
Received 2 February 2018; Received in revised form 22 June 2018; Accepted 16 July 2018
Available online 01 August 2018
0926-5805/

ignore the coding problem, the development of the grammar is still a difficult task because designers need to decompose some global intentions like problem requirements and generation mechanism into information fragments and map them onto the rules or the relations between them. To address this problem, some studies adopted a compromise strategy that combines simple production rules with extra algorithms as global or local control [9]. But in fact, this way is limited in alleviating the burden of the grammar developer, while it makes part of the generation logic opaque to other users of this grammar. Further, the IF-THEN structure of rules determines that they inherently have inference ability [6]. Encoding a certain degree of automated reasoning to rationalize solutions is one critical advantage of using grammars [10], though it is not as easy to organize as the production ability in grammars. The key lies in how to convert design logic to machine readable and operable patterns.

Therefore, this study aims to develop an algorithmic design grammar by creating and organizing rules in the common structural ways of sequence, selection and iteration. In this way, the exploration for a valid solution fully depends on the design grammar with the algorithm expressed within the design grammar. In addition, rather than only taking the design grammar as a production system, this paper emphasizes its use for problem-solving in a deductive way. Benefiting from the modular organization of rules, several design grammars are adapted from the basic design grammar, to encode more advanced deductive strategies. The comparison between them illustrates the potential and limitations of an algorithmic design grammar in problem solving. LGP is a good case for testing the proposed approach since the complex and intertwined conditions in the problem specification need some algorithmic structures and deduction logic.

2. Antecedents

2.1. Shape grammar for designing

Shape grammars were introduced by Stiny and Gips in 1972 as a production system that computes with shapes and symbols [11]. There are a few well-known strengths of shape grammars for design, such as versatility (representation of any shapes), intuition (ease of observation), automation, guarantee of style consistency, ability to generate emergent shapes, and so forth [9, 12]. Many works highlight their generational power for both analytical (to capture and imitate a certain architectural style, e.g. the Palladian grammar [13]) and synthesis (to create new designs, e.g. the kindergarten grammar [12]) purposes. Compared with analytical grammars, design (synthesis) grammars are much more meaningful for design practice as design synthesis is an inevitable step to obtain a new design.

In terms of the grammar development process, the development of an analytical grammar and a design grammar, both require the same five steps [12]: specifying the vocabulary, spatial relations, shape rules, initial shape(s), and shape grammar. While these may be relatively straightforward for the development of an analytical grammar, as the target designs are predefined, they are much more complex for a design grammar, since for a developer it is difficult to exactly predict what elements a desired design should have before he or she finishes a design. The inherent unpredictability of design leads to a complex interaction between the design process and the development of the supporting design grammar. This is one important reason why design grammars are still far from practical.

Ruiz-Montiel et al. [9] consider two types of shape grammars: expert grammars which hard-code expert domain knowledge in rules, and naïve grammars without any guides for rule selection. Rather than adopting such a control mechanism point of view, we prefer to classify design grammars from the perspective of the generation logics into object-oriented and goal-oriented. The former aim to map solutions onto the assembly of feasible design components, while the latter focus on the design exploration driven by specific evaluation criteria. Note that these terms are used here a little different than in computer science.

In the object-oriented type, in order to construct feasible solutions, developers need to extract the characteristics of desired designs and allocate them to the rules in a certain logic. Obviously, most expert grammars belong to this kind. There are mainly three requirements for object representation: feasibility, diversity and efficiency. Generally, feasibility is achieved by semantic information and constraints. For example, in his design grammars for urban planning, Beirão [14] built ontologies to organize the semantics of design objects and combined constraint descriptions with shape rules to guarantee that designs comply with specifications. Elsewhere, labels have been added to facilitate valid designs, e.g., indicating functions [15] or indicating design states [16]. For diversity, three evident aspects are involved: shapes, parameters and spatial relations. In the digital camera grammar, Lee [17] extended the pool of rules by using Boolean operations to create new shapes and adjusting the spatial relations between shapes. Finally, most studies do not reflect much on efficiency since it is difficult for developers to assess a representation before the application of the grammar. A reasonable way is to optimize the representation after analyzing the results of a grammar (e.g., [18]). It is not hard to see that object-oriented grammars attempt to clarify as many things as possible, no matter for the design space or the search path. Thus, designers are able to easily understand how a grammar works and predict its products, although it imposes restrictions [19].

In the goal-oriented type, rather than predicting the likely characteristics of a desired design, developers obtain some definite information about them, that is, goals. Evaluation is embedded to give feedback about how far off the current design is from the goals and then to guide rule selection. Essentially, the goal-oriented approach attempts to narrow the solution space down to those areas of interest [20]. Generally, there are two kinds of mechanism to achieve this:

- Global search is the most common one and evaluates complete designs. It often combines grammars with external control algorithms. For example, Gero et al. [21] applied a genetic algorithm to vary the rule sequence to generate satisfactory solutions. Shea [22] integrated a simulated annealing algorithm and a shape grammar to explore the layout of discrete structures, such as planar trusses and space trusses. Ruiz-Montiel et al. [9] made use of a reinforcement learning technique to select rules and generate a large variety of 2D house layout schemes complying with design requirements. In addition to changing rule selection or sequence, Gero et al. [23] also used genetic engineering to evolve shape rules to construct solutions in a more efficient way. The grammars in most of these studies are naïve grammars so that more freedom and creativity is allowed in the rule sequence. However, it does not mean goal-oriented approaches are not appropriate for expert grammars. Lee [17] applied genetic programming and genetic algorithms to the expert grammar for digital camera design. The key point is to guarantee the selection of rules in a rigorous grammatical sequence.
- Local reasoning requires the design grammar to act not only as a production system, but also as an interaction system. This means a rule needs to recognize the gap between the evolving design and the desired design during its application, and respond to bridge that gap. To achieve this goal, developers have to code control principles in the rules. However, as local reasoning often needs some additional non-geometric information, most work still adopts external algorithms or calculation formulas to control rule selection. The discursive grammar by Duarte [24] is a typical example, although it is in principle an analytical grammar. Heuristics determine if a rule can be applied based on the predicted result. Another example is the minaret design grammar by Al-kazzaz [16], in which the rule and the grammar are assessed in real-time by evaluation criteria so that the user can choose rule applications accordingly. Agarwal [15] considers a similar evaluation system in his coffee maker grammar.