Computing Architectural Designs Using an Architectural Programming Language

Ganapathy Mahalingam

Keywords
Computing architectural designs, architectural programming language, architectural grammars

Abstract

The purpose of this paper is to present the issues related to computing architectural designs at a very fundamental level. The paper does this by proposing an architectural programming language with which to write programs that generate architectural designs.

The paper explores the potential and pitfalls of computing architectural designs using the proposed language. The paper presents the programming language, complete with the Backus-Naur form (BNF) for the language. The purpose of developing this language is to provide a tool to write programs that generate architectural designs when executed. A complete syntactical description of the language including its starting symbol, its non-terminal symbols, its terminal symbols, and its set of production rules is provided. The adoption of alphanumeric symbols versus graphic symbols in the language is presented. The Chomsky-Hierarchy, and the range of grammars that it provides, is briefly explored for the suitability of these grammars for architectural design.

The problems of representational thought occurring in architectural design, and how they are transposed when the representations are part of a programming language, are briefly discussed. The evaluations of designs produced by programs written in the programming language, their validity, and the problems presented by the language are also briefly discussed. The potential of automating architectural design using computer programs written in the architectural programming language is projected.

Introduction

The potential success in developing a programming language for architectural design depends on a careful mapping of the fundamental operations in the creation of architectural designs onto a set of computable operations. A characterization of architectural design at a fundamental level is needed before a programming language can be defined to enable the creation of architectural designs. Architecture has been defined as the art and science of designing buildings and supervising their construction. The creation of a work of architecture is the result of a complex interaction of diverse processes. However, the complexity in the creation of an ar-

chitectural design belies a set of simple, fundamental operations. This paper is an exploration of the creation of a programming language for architectural design based on those simple, fundamental operations.

Programming Languages

A programming language is defined by its syntax and semantics. The syntax of the language describes the rules for creating structures (programs) using the language, and the semantics of the language reveals the meaning of valid structures (programs) that can be created with the language. Of these, the syntax is formally represented. Examples of formal description systems for the syntax of a programming language are the Backus-Naur notation and syntax diagrams.

A complete syntactical description of a language is called a grammar. A grammar can be considered a tuple of the following elements [4]:

- Starting symbol (S)
- Terminal symbols (T)
- Non-terminal symbols (N)
- Production rules (P)

The notation for a grammar is thus: G (S, T, N, P)

A language (L) based on a grammar is defined thus: L (G) = L (S, T, N, P)

To create a programming language for architectural design, one has to define the starting symbol, terminal symbols, non-terminal symbols and production rules for the creation of architectural designs. This may seem a daunting task, but, if we realize that the fundamental entities in architecture consist of form and space, solids and voids, the definition of a language for architectural design becomes viable.

Mapping Architectural Design Onto A Programming Language

The task of creating a programming language for architectural design starts with the definition of a grammar for the creation of architectural designs. Using the 4-tuple form for the definition of a grammar, G (S, T, N, P), architectural design can be mapped thus:
Starting symbol: Architectural form (f)
Terminal symbols: Solid polyhedron (p_s), Void polyhedron (p_v), Union (U), Difference (\)
Non-terminal symbols: Architectural form (f), architectural space (s)
Production rules:
f Y p_s C f U f C f \ s
s Y p_v | s U s
The union operation (U) has precedence over the difference operation (\) in the production rules. The vocabulary (V) of the grammar or language is defined as N U T, that is the union of the non-terminal and terminal symbols. The use of the symbol * after V, N or T indicates all possible strings over the sets of V, N and T.

These production rules defined give rise to other production rules of the form:
f Y p_s U p_s
This production rule allows an architectural form to be created by unioning a solid polyhedron with another solid polyhedron.
f Y p_s U f
This production rule allows an architectural form to be created by unioning a solid polyhedron with another architectural form.
f Y p_s \ p_v
This production rule allows an architectural form to be created by differencing a void polyhedron from a solid polyhedron.
s Y p_v U p_v
This production rule allows an architectural space

to be created by unioning a void polyhedron with another void polyhedron.

s Y p_v U s

This production rule allows an architectural space to be created by unioning a void polyhedron with another architectural space.

If you visualize the creation of an architectural design, an architect starts with an existing architectural form, the site of the design. The architect then synthesizes a new form by creating a solid polyhedron, combining solid polyhedra (material) or removing void polyhedra (empty space) from the solid polyhedra (material). The production rules defined to create architectural forms are both recursive and non-recursive. Since there are are an infinite number of solid and void polyhedra, this grammar does not preclude any architectural form.

The grammar presented above is context-free like most programming languages. The actual grammar to create specific types of architectural forms will be a refined version of this grammar. Our grammar captures the essence of a real grammar that creates an architectural form.

Since polyhedra are themselves complex entities, a nested grammar can be defined to generate polyhedra. This series of nested grammars can then be used to develop a comprehensive programming language for architectural design.

Hierarchy of Grammars

Grammars have been classified based on a classification system called the Chomsky-Hierarchy. The types of grammars in this hierarchy have been defined by Teufel [4] as:

- No-restriction grammars
- Context-sensitive grammars
- Context-free grammars
- Regular grammars

The classification is based on the restrictions imposed on the production rules in the grammars. In no-restriction grammars, as the name implies, there are no restrictions imposed on production rules.

In context-sensitive grammars, production rules can take the form:

aAb Y agb

where, A X N, a, b, g X V* and g is not an empty string

This production rule allows the substitution of a non-terminal symbol between (the context) elements of a vocabulary with another element of the vocabulary. These types of production rules, if used, allow restrictions in the creation of architectural forms based on contextual conditions.

In context-free grammars production rules can take the form:

A Y a

where, a XV* and A XN

This production rule means that an architectural form or space (the non-terminal symbols) can be created by the unioning or differencing of other architectural forms and spaces, or solid and void polyhedra. These types of production rules are the common way in which we (at least the users of computer-aided design systems) understand the creation of architectural forms.

In regular grammars, production rules can take the form:

A Y a | aB | Ba

where, a XT* and A, B X N

This production rule means that an architectural form can be created from a solid polyhedron or the combination of a solid polyhedron and an architectural space. These types of production rules are more specific in specifying how architectural forms are created.

These range of grammars define languages for the creation of architectural forms in an abstract sense. For all the complexity there is in architectural forms, their **grammar of creation** may very

well be simple and fall within one of the four types of grammars in the Chomsky-Hierarchy.

Architectural Design as Representational Thought

Architectural design can be considered a form of representational thought, where constructed forms and defined spaces are manipulated using a representational medium such as graphics. The process of synthesizing an architectural design is achieved in the representational medium. The use of graphics as a representational medium has been successful in architectural design, because the visual experience of a work of architecture is its primary mode of consumption. Drawings are also useful because physical and structural properties of materials can be derived from the geometry, so it is graphically possible to design a structure of a certain material that would be safe if constructed. It is also possible for instance, to predict at a reasonable level of accuracy, the daylight in a space, or its acoustical conditions from its geometry.

The fundamental problem of representational thought in architectural design is how the representations approximate the properties and behavior of the various architectural entities. A drawing of a brick does not behave like a brick, nor does it have the properties of a brick. It may have a visual likeness of the brick, often reduced in scale, but that is as far as it goes. The drawing of the brick needs to be supplemented with the user's knowledge and calculations to become useful.

The notion of programming an architectural design, rather than drawing the design, or to make the situation more complex, programming a design by drawing using a visual programming language, poses some fundamental questions about representational thought that characterizes architectural design. How do we deal with bricks when we program them? If the goal of representational thought about architectural entities is capturing their properties and behavior, and allowing the flexible manipulation of the entities in the representational medium, then the representational medium should be evaluated based on those factors. When comparing representational media, the amount of information brought to bear on the representations in a medium by the user should be taken into account. Graphical media in architecture are not successful because they provide a facile environment to manipulate architectural entities according to their properties and behavior, but because of the wealth of knowledge brought to bear on the representations by the user. An architectural programming language should also be evaluated for its usefulness, taking into account the knowledge brought to bear on it by the programmer/designer. We have to develop the same facility we have with sketching in creating programs to generate architectural designs.

Concluding Thoughts

Kalay [1] calls computer models of real-world phenomena "languages of representation." What if this language of representation is a programming language? Symbols sets used in computer programming languages include the binary set (1,0) or the number set (1,2,3,4,5,6,7,8,9,0) or the English alphabet set (a,b,c,d…z). Such sets allow for programs to be written in an alphanumeric language. The traditional language of architectural design is graphical. Therefore, a programming language for architectural design should probably use graphical symbols instead of alphanumeric symbols. This would make an architectural programming language a visual programming language. What if the symbol sets in an architectural programming language are graphical? Can one then draw a program instead of writing one? The equivalent of a sentence in an alphanumeric programming language would be a

drawing in the visual programming language. What are the problems or benefits related to checking the validity of a program if it is drawn using graphic symbols? Actually, the problems related to checking the validity of a program written in a visual programming language should be no different than syntax checking in an alphanumeric programming language, if the graphic elements directly correspond to alphanumeric elements.

In the grammar for an architectural programming language presented in this paper, if the alphanumeric symbols are replaced with graphics representing the polyhedra, then the string of alphanumeric symbols generated by the production rules has a graphical equivalent. The architectural programming language can generate different strings based on the production rules. These strings can then be converted into graphics by substitution. Each sentence in the language will then become a spatial composition. When the substitution is made, there may be invalid forms created by some of the production rules. This is because the alphanumeric symbols are not spatial. For example a void polyhedron that is larger than a solid polyhedron cannot be differenced from it. Similarly, two solid polyhedra that do not overlap cannot be unioned to create a single architectural form. A mechanism is needed for checking spatial parameters of the polyhedra when implementing the production rules.

Drawing an architectural design may not be essentially more complex than programming an architectural design except for the visual immediacy of the drawing and the unstructured (or very complexly structured, depending on your viewpoint) nature of the drawing process. If graphical symbols are used in the architectural programming language, then programming an architectural design can become another form of drawing, a shorthand graphical notation of the design that reveals its full visual form when the program is executed. Even symbols for operators in the architectural programming language can be given graphical equivalents. **A drawing will then be a program.** This will be possible if the sequence of elements and operations used to create the drawing is accessible in order to map it onto a program. A finished drawing on paper using traditional media does not have a record of the sequence of graphic elements and operations used to create it, but a computer-based drawing does! Computer-based drawings can then provide a computational medium for the generation of architectural designs in a completely different sense.

With a well-defined architectural programming language, architectural designs can be generated by executing programs written as you would with a general-purpose programming language like Smalltalk. Programs can then be written (drawn?) to generate programs that generate architectural forms. This can lead to a powerful form of automation in the creation of architectural designs. The author's work in developing algorithms and design systems for the generation of auditorium designs [2,3] is founded on the fact that you can write a program to generate an architectural form based on various functional and programmatic parameters.

References

1 Kalay, Y. E. **Modeling Objects and Environments.** John Wiley & Sons, New York, 1989.

2 Mahalingam, G. **The Algorithmic Auditorium.** In proceedings of the CAADRIA '98 conference, Osaka, Japan, pp. 143-152, 1998.

3 Mahalingam, G. **Object-Oriented Computer-Aided Design Systems for the Preliminary Design of Auditoria.** Journal of Architectural Planning and Research, Vol. 13:3, pp. 214-229, 1996.

4 Teufel, B. **Organization of Programming Languages.** Springer-Verlag, New York, 1991.

Ganapathy Mahalingam
North Dakota State University
mahaling@plains.nodak.edu