Georgia Institute of Technology
Atlanta, Georgia

Automated Generative Synthesis Systems in Architectural Design

by Maher El-Khaldi.

Abstract:

Designing by generative synthesis systems (GSSs) can maximize the quality of design solutions. Such a gain is directly related to: A) the system-building process, which entails: dismantling problems' components, unpacking embedded relationships, tracing dependencies, clarifying design objectives, and critically acknowledging reasoning mechanisms (subjective or objective); and B) the systems' ability to generate alternative solutions (Heisserman, 1994), which offers the designer a chance to compare between possible candidates and select the "better" ones. The capacity of such systems is best explored within a computerized environment where automation is not only possible, but also accelerated.

1. Introduction:

The idea of structuring design synthesis processes around generative synthesis systems (GSSs) and design languages is not new. Expressions of such a concept can be found in the works of: Vitruvius, Durand, Louis Sullivan, Le Corbusier, Peter Eisenman, Alvaro Siza, among others. This "movement" continued to evolve as the architectural landscape witnessed the works of Christopher Alexander, Lionel March, George Stiny and James Gips, William Mitchell, Charles Eastman, Terry Knight, and Chris Yessios, among many others.

Parallel to these works, was the continuous flow of computational technologies (mainly programming and CAD) into the field of architecture. This fueled the process of building generative synthesis systems as designers gained access to independent programming languages such as: C, C++, Java, Visual Basic, etc; and CAD-hosted ones such as: Auto-LISP (AutoCAD), MEL (Maya Embedded Language), RVB (Rhino-Script), Max-script (3D Max), FSL (**form•Z** Scripting Language), among others. As a result, a "new" type of synthesis systems started to take its shape through automation.

Krishnamurti described the architectural design process as a function of knowledge and strategy (2006). The deeper the knowledge is, the more informed our design decisions become. And the richer a strategy is, the more alternatives a design process generates. Knowledge and strategy drive the quality and magnitude of alternatives generated by GSSs. In Algebraic terms, these alternatives resemble vectors that span a space of solutions. Solution spaces are system-specific. Their definition depends on the components used to build their generating GSSs. Thus, there are no generic metrics to measure or evaluate their characteristics. However, the fertility of GSSs (how generative they are) is mirrored in the capacity of embedded design rules.

Rules are expressions composed of a left side, an operator and a right side. In algebra, rules can be written as functions like (A(X)-->Y), or equations like (X=2+Y). The first maps (transforms) element X into element Y through a function A, and the second assigns the value of 2+Y to element X. The first type can be also viewed as replacement rules, and the second as associative rules. Replacement rules erase input, and place output. In this regard, (A(X)-->Y) is interpreted as: replace element X by element Y through rule A. Associative rules establish associations between both sides. In this regard, (X=2+Y) is interpreted as an association between element X and 2+Y. Associations are of two types, mono-directional and bi-directional. Mono-directional associations enforce a "Parent-Children type" of relationships where a hierarchy drives the flow of data from top to bottom only. Thus, manipulating "Parents" propagates to "Children", but not vice-versa. Bi-directional associations allow data to flow in both directions.

Rules manipulate elements through their representation. Elements' representation is system-specific. For example, in a Cellular Automata (CA) system (Wolfram, 2002), elements are expressed as cells, usually arranged in orthogonal grids. Figure 1 illustrates how CA rules work in general.

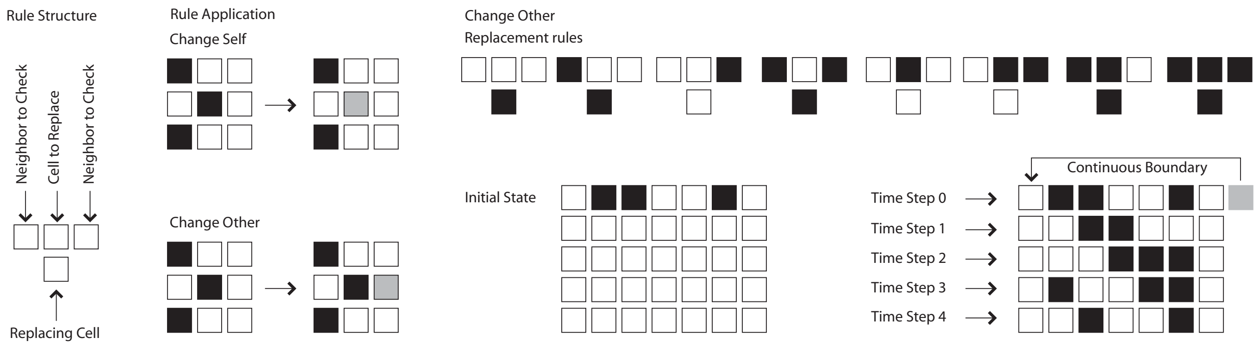

Figure 1: The application and mechanism of applying Cellular Automata rules.

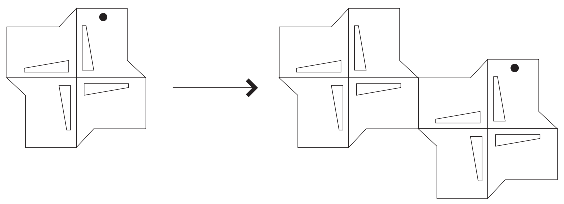

Figure 2: A Shape Grammar rule that translates the initial shape (left side) and creates a copy.

In Shape Grammars, elements are expressed in shapes with labels, axes, and notations. Figure 2 is an example of shape grammars rules.

Where in L-systems (Lindenmayer & Prusinkiewicz, 1990), elements are presented as strings of characters (Axioms) as shown below in Figure 3. L-systems Axioms are usually used to represent (package) geometric data, growth directions, or nested strings of data.

L->RL
MR -> G

Figure 3: L-system replacement rules. The first rule replaces "L" by "RL", where the second replaces "MR" by "G".

When building GSS, designers may choose to combine, edit, or even invent new representations for the design elements within a system.

One might argue that design processes cannot be approached holistically as a set of algebraic "left and right" rules because they do not accommodate for the designers instantaneous intuition. A distinction should be made to clarify the context in which design is best produced by rules.

Architects usually build methodologies for solving different types of problems to develop prototypical solutions. One of the most famous published works in this area is Christopher Alexander's "Pattern Language". In his book, he breaks down different design contexts of different scales to a number of objectives and requirements, and ways to provide possible solutions. More familiar examples on prescribed, "off the shelf" type of solutions can be found in the Architectural Graphic Standards books where extensive solutions and rules can be found for almost any architectural design concern.

In the light of this discussion, design rules can be articulated in different ways such as: if site context (X) is true, then build a fence (Y); or drive the numeric value of slope (A) by the ratio B/C, or the position of stairs (M) is always perpendicular to walls (B); etc. Design rules are processes devised to offer solutions under specified conditions.

Prototypical solutions are not to be mistaken for systems. They only present the right side (output) of design rules. Generative systems are structures capable of processing input and generating output through utilizing design rules. One of the most famous examples on generative synthesis systems is the "Palladian Grammars" that Stiny and Mitchell (1978) built using a Shape Grammars. Figure 4 below shows a number of alternative solutions generated by the system. The grammar was designed to reproduce alternative solutions for an extracted design language from the works of Palladio.

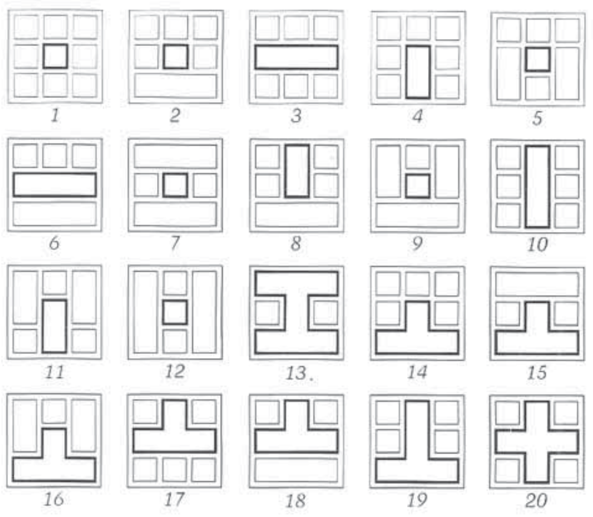

Figure 4: Twenty (of many other) alternative villa designs generated by the grammar. (Mitchell, 1989).

Generative synthesis systems are advantageous (and applicable) to the design process if and only if the designer was able to define: A) clear definition of design objectives; B) a design language through which these objectives are to be expressed (Yessios, 1975), which is necessary to bound the scope of system building; C) and a formalism to describe the generation process (expression of objectives). Design objectives are expected gains from a certain design schema. Design languages are material expression of design objectives. Design formalisms are combinations of design rules and representations. Rules are built or extracted in relation to both the design context and the design language being implemented. Rules manipulate design elements (units) through their representation. A design element can be represented in various ways in relation to different contexts. Each representation method highlights certain aspects of the design elements. Hence, rules are functions that manipulate units through an interface defined by representation. In that sense, Palladian Grammars is a generative system for synthesizing alternative designs of Palladian Villas through Shape Grammar, a formalism that manipulates units via a representation of drawn shapes.

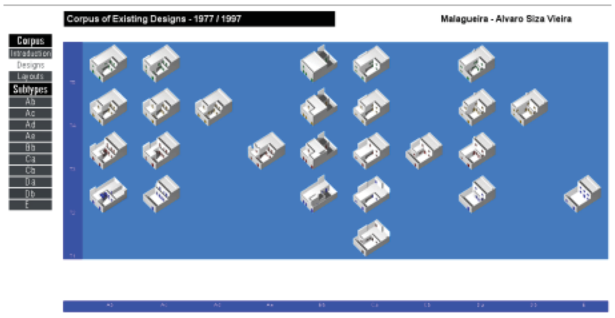
Corpus of Existing Designs - 1977 / 1997
Malagueira - Alvaro Siza Vieira
Corpus
Introduction
Designs
Layouts
Subtypes
Ab
Ac
Ad
Ae
Bb
Ca
Cb
Da
Db
E

Figure 5: Alternatives of Siza's design by Duarte's Malagueira automated grammars (2005).

In Siza's work, one can notice consistent methodologies for form derivations such as site lines, geometric proportions, use of materials, compositions of volumes, relationships between solids and voids, etc. The richness, clarity and consistency of Siza's work made it possible for Duarte (2005) to automate the generation process of Malagueira houses as shown in Figure 5.

2. Projects:

The following two projects illustrate a different design scope, buildings' skins (A more common implementation of generative systems to the architectural practice nowadays). The first project, Smart_Component_01, shows an adaptive assembly of flat panels built with standard joints. It presents a design instance, a singularity or rather a building brick that can be used repeatedly to produce a mechanically adaptive skin. The project was realized by two environments: CATIA and **form•Z**. The former provided a parametric environment allowing for building numeric and geometric rules (relationships), where the second provided a partially parametric environment offering a more transparent modeling experience and allowing for faster conceptual studies of initial design configurations. The second project, Panels_Optimizer_01, offers a paneling methodology for complexly curved surfaces. It shows a top-down approach for finding design solutions through iterative loops of construct-and-improve algorithms. This project was realized by two environments as well: Rhino and **form•Z**. The former was used to automate design rules through the offered scripting language (RhinoScript) where the latter provided extensive tools for conceptual digital modeling and 3D-printing purposes.

2.1 Smart_Component_01:

The initial design objective was to create a surface with various levels of transparency driven by its curvature. This transition was approached as different degrees of porosity. Throughout the early experiments with digital and physical models, the design concept evolved from a holistic view of a surface to a singular view of a component. The design strategy was to create one smart component capable of configuring itself to a range of different conditions, namely surface degrees of curvature. While the component was designed to deliver various adaptation modes, it maintained the flatness of its geometric elements by utilizing a number of strategically placed joints. The robust system of joints helped avoid shearing, bending, and stretching. It also allowed for using only two types of panels, which facilitates easy assembly. Two possible solutions were developed using the same number of joints and panels, but configured differently. Based on the quality of the required adaptation, the second configuration was selected as a final solution. This was due to the fact that it had a fewer number of degrees of freedom, thus a better-controlled behavior in comparison to the first one.

The initial studies of design configurations were achieved through utilizing **form•Z**. The richness of its environment allowed for fast construction of digital representations, and translation to different file formats to communicate with fabrication equipments. After the exploration of different design ideas through **form•Z**, a GSS was built within CATIA by embedding numeric and geometric rules (relationships). The generative synthesis system was composed of four sub-systems: "Controlling geometry" responsible for driving the assembly of components; "Responsive geometry" responsible for mapping the changes in the previous sub-system onto a hexagonal point-grid; "Mediating geometry" responsible for translating changes in the previous system to the "Adaptive geometry". Figures 6, 7 and 8 show initial studies of joints, GSS structure, and population of the selected configuration.

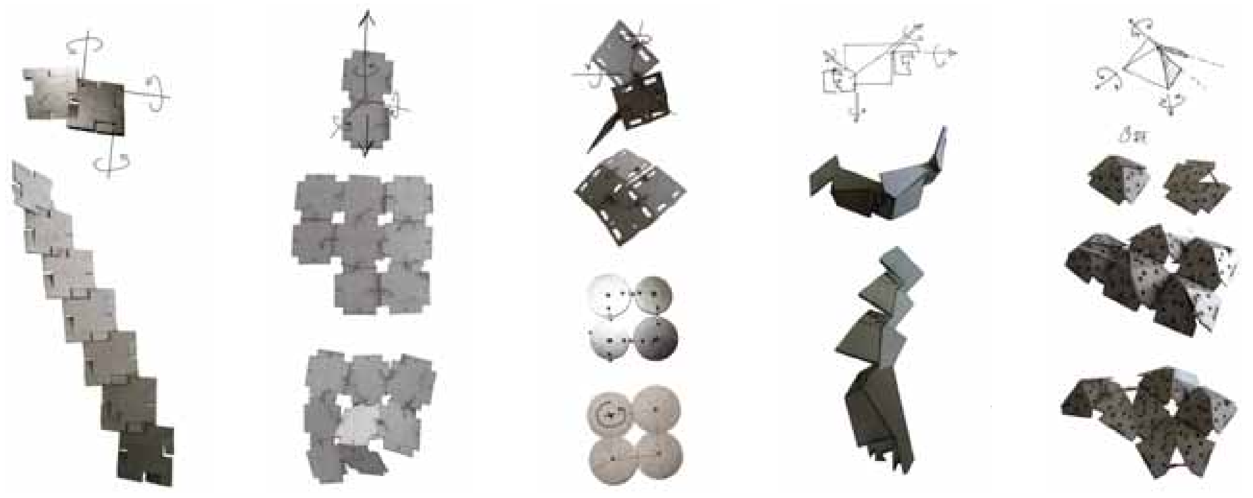

Figure 6: Initial studies of joints.

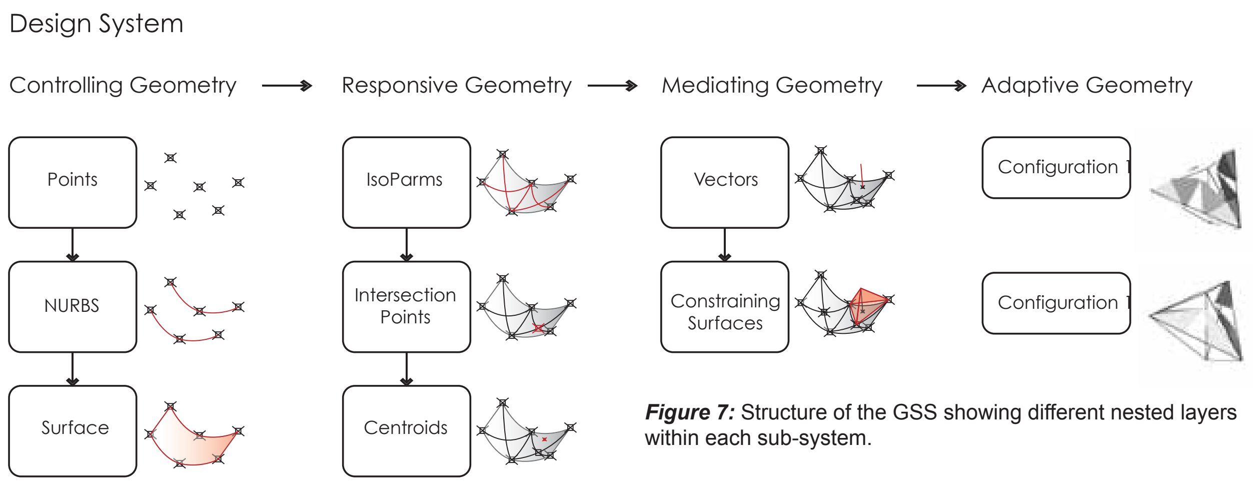

Figure 7: Structure of the GSS showing different nested layers within each sub-system.

2.2 Panels_Optimizer_01:

In the second project, the design objective was to devise a generic subdivision methodology for complexly curved surfaces. Initial studies of subdivision methods were achieved in **form•Z**. Its powerful engine provides a state-of-the-art solid modeling tools making it a perfect design environment for generating physical mockups via 3D printers. This helped define a design language and extract a set of design rules in a timely manner. After defining a language, the GSS was built in RhinoScript as a recursive algorithm. A main system component, Moderator, dictates the solution generation process (rule implementation) and selection (recognition) of elements.

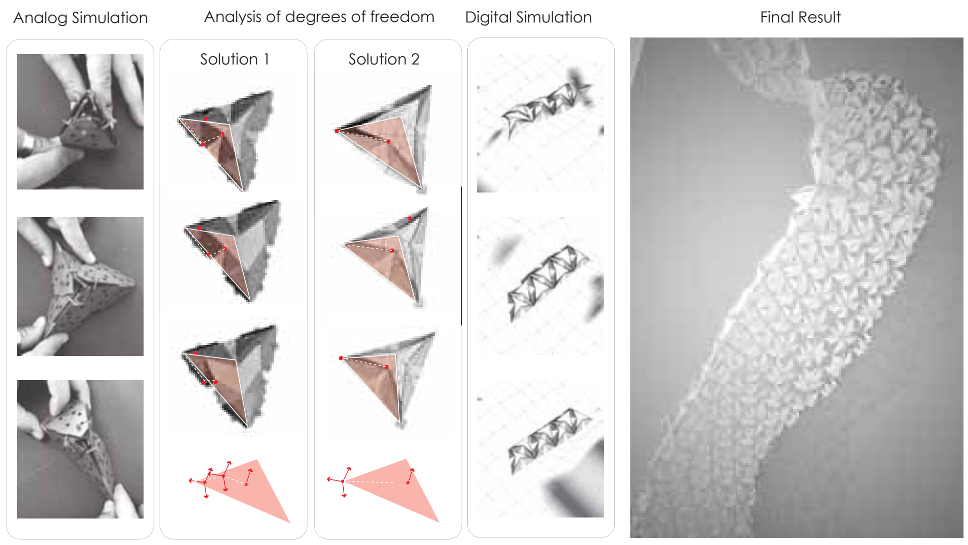

Figure 8: Showing a developed physical model; two designed configurations; a strip of "Solution-2" adapting to a deforming surface; and finally 1,000 components populated within a space defined by a curved surface.

There were two major goals: achieving speed in assembly (evaluated by the number of panels); and attaining the smoothest appearance (evaluated by the minimum size of panels). The trade off between these two drivers demonstrates a very simple, but true model of the architectural design process where conflicting demands have to be resolved. For example, the more panels there are, the slower the construction process becomes, but also the smaller the panels are and the smoother the final shape is. In contrast, the fewer number of panels, the faster the construction process becomes, but also, the bigger the panels are, the more angular the final shape is. To resolve this conflict, two other parameters were introduced: panel-curvature and style. Curvature suggests the integration of cost-related variables where flatness is desired. Style empowers aesthetic judgment to guide the generation process. In this case, the style chosen was fractal-patterns, which were generated by subdividing the diagonal panels. The design strategy was to recursively sub-divide and rebuild optimized panels until a stopping condition is achieved. In this case: A) an acceptable panel curvature, or B) a minimum panel size. The number of panels was driven by these two parameters, while the fractal pattern was generated by the embedded sub-division rules. Finally, each panel was given a thickness based on its curvature as a suggestion of material properties. Figure 9 shows the structure of the implemented GSS and a solution for a curved surface.

3. Conclusions

Generative synthesis systems require clear descriptions of the architectural languages being explored. Once a language is defined, it becomes possible to extract rules and formalize generation processes. Generative design processes offer a larger number of synthesized alternatives in comparison to the classical design processes. Building design synthesis systems urges the designer to understand their intuition, dismantle design problems, define design objectives critically, acknowledge decision drivers whether subjective or objective, and outline solution spaces rich of alternatives.

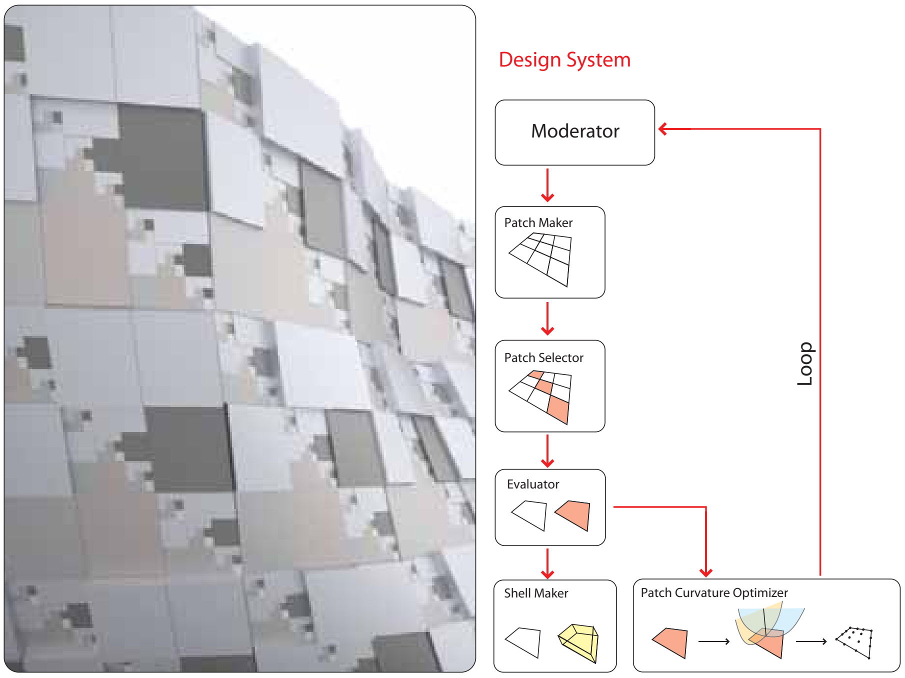

Figure 9: Left: showing a surface sub-division with different thicknesses based on each panel's curvature. Right: Showing the structure of the generative synthesis system for Panel_Optimizer_1.0.

References

Duarte, J. (2005). Towards the mass customization of housing: the grammar of Siza's houses at Malagueira. Environment and Planning B: Planning and Design 32(3), 347 – 380

Heisserman, J. (1994, March/April). Generative Geometric Design. IEEE Computer Graphics and Applications, 14(2), 37-45.

Krishnamurti, R. (2006). Explicit design space? Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 20, 95–103.

Lindenmayer, A. , Prusinkiewicz, P. (1990). The Algorithmic Beauty of Plants. New York: Springer-Verlag.

Stiny, G. , Mitchell, W. (1978). The Palladian Grammar. Environment and Planning B: Planning and Design. 5, 5-18.

Mitchell, W. (1989). The logic of architecture. Cambridge: The MIT Press.

Wolfram, S. (2002). A New Kind of Science. Champaign: Wolfram Media Inc.

Yessios, C.I. (1975). Formal Languages for Site Planning. In C. Eastman. (Ed.), Spatial Synthesis in Computer-Aided Buildings Design. New York: Wiley.

Maher El-Khaldi is a researcher at Georgia Institute of Technology pursuing a PhD in Design Computing. He received a Master of Science in Design and Computation (2007) from the Massachusetts Institute of Technology, and a Bachelor of Architecture (2004) from the American University of Sharjah, United Arab Emirates (UAE). Maher is the director of design computing division at Shape Architecture Practice + Research, UAE. Email:maher@shape-arc.com.