Automatic Generation of Layouts of an Utzon Housing System via the Internet

J. Michael Gerzso
Independent Researcher, Boulder, Colorado, USA

Abstract
The article describes how architectural layouts can be automatically generated over the Internet. Instead of using a standard web server sending out HTML pages to browser client, the system described here uses an approach that has become common since 1998, known as three tier client/server applications. The server part of the system contains a layout generator using SPR(s), which stands for "Spatial Production Rule System, String Version", a standard context-free string grammar. Each sentences of this language represents one valid Utzon house layout. Despite the fact that the system represents rules for laying out Utzon houses grammatically, there are important differences between SPR(s) and shape grammars. The layout generator communicates with Autocad clients by means of an application server, which is analogous to a web server. The point of this project is to demonstrate the idea that many hundreds or thousands of clients can request the generation of all of the Utzon layouts simultaneously over the Internet by the SPR(s) server, but the server never has to keep track when each client requested the generation of all of the layouts, or how many layouts each client has received.

Keywords
Internet, Spatial-Production-Rules Grammars, Utzon

1 Introduction: Something Old, Something New

The work described in this paper is based one of my articles (Gerzso 1974) in which a string grammar known as SNARQ was proposed as a precise description of the assembly rules used by Jorn Utzon in his industrialized housing system (see note 1). It took into account two aspects of his design: a set of standardized spaces made up of industrialized building components, and a set of rules as to how to join the spaces and components together. He then used these spaces and rules to design many variations of a house based on the "mangled T" motif. Upon analysis, Utzon's layouts seemed to suggest that he was using recurrent or "grammatical" rules, and that these rules could be represented by a string grammar.

In addition to SNARQ, my other related work has been frequently confused with shape grammars (Gips 1974; Stiny and Gips 1974). This confusion stems from the fact that both approaches use some form of syntactic rules, or "grammars", to describe design rules. SNARQ was developed independently of shape grammars in that it was a direct outcome of Allan Shaw's work on picture grammars (Shaw 1970)(see note 2). There are other important differences:

- Gips and Stiny were motivated in the early 1970's to develop shape grammars to make aesthetics computable. Gips concedes that there is no one aesthetic viewpoint –as he terms it-, and despite that, he states that this "does not preclude the possibility of precisely stating aesthetic viewpoints". (p. 149, Gips 1974). The word "precisely" is to be read as "computable". He goes on to propose a logical framework of an aesthetic system based on shape grammars. In contrast, the development of SNARQ is simply a hypothesis about the precise nature of *some design rules used by one designer, namely Utzon, for one project* (see note 3).
- The second important difference has to do with underlying assumptions regarding architectural design. In 1974, Gips and Stiny assumed that generative systems are applicable to aesthetics in general, but they did not provide any concrete architectural examples to bolster their argument. In contrast, SNARQ was developed to attempt to represent some of the design rules of a *specific architectural project.*
- The next difference concerns the definition of the grammars themselves. While SPR(s) are standard so-called Chomsky context-free grammars (CFG), shape grammars are not. SPR(s) have a lexicon (terminals or words), have a syntactic structure (production rule grammars), and have nonterminals (see section 5). Shape grammars have a predefined set of shapes more akin to a kit of parts instead of a lexicon, shape production rules instead of a grammar describing sentence structure, and a labeled or marked shape sometimes referred to as a nonterminal.
- Finally, most if not all of the publications on shape grammars deal with the generation of shapes in a "language of designs", and very little on parsing or recognition of "shape sentences". On the other hand, SPR(s) can be used in conjunction with parsers as well as generators.

Comparing shape grammars with standard context-free grammars is not straightforward because the definitions of shape grammars are not always consistent. According to Gips, a shape grammar is defined using string grammar terminology. He replaces words with shapes. He defines terminals as shapes, nonterminals as markers that are associated with shapes, and production rules as rewriting rules of shapes (p 5 and p10, Gips 1974). Most of his grammars contain only one nonterminal. He shows how shape grammars can produce embedded or inscribed squares, arrays, snowflakes and Hilbert curves.

In one of the subsequent definitions of shape grammars, the explicit specification of terminals and nonterminals is dropped (p 347, Stiny 1980). The grammars are cast as sets of lines. In this guise, they seem to be a variation of graphs. An interesting conjecture is that shape grammars are a subclass of graph grammars (Rozenberg 1997). In his explanations of his definitions, Stiny seems to be saying that terminals are shapes and that nonterminals are labeled shapes. A marker in the Gips sense is a type of labeled shape. Terminals and nonterminals exist albeit implicitly. In another version (p 143, p 147, and p 148, Mitchell 1990), the definitions include once again the explicit notion of terminals, but nonterminals remain implicit as markers.

The presentations of the definitions of shape grammars usually include only one marker or nonterminal. What does this mean? One could argue that a shape grammar with one nonterminal is like defining a string grammar in which the syntax only permits concatenating one type of word, such as nouns. This may be interesting in studying algorithms, but not in studying architecture or languages –natural or artificial-.

In applying shape grammars to specific architectural examples, the shape grammar formalism changes once again. Stiny presents the idea of applying shape production rules in two or more stages so as to be able to produce Mughul gardens (Stiny 1980) and Palladio villas (Mitchell 1990). For each stage, there is a set of rules to be applied. However, taken as a whole, shape grammars are no longer grammars in the CFG sense. The rules for the first stage are CFG's, but the rules of the remaining phases are related to Chomsky transformation rules (p 89, Smith and Wilson 1979). In the construction of compilers, transformations are rarely used. SPR(s) contain no transformations (see note 5).

Since 1974, the SNARQ spatial grammar seemed to be relevant only as an academic exercise. However, in the last several months, it became evident that this grammar could be useful for a test case implementation of a web based generative system. To that end, SNARQ was reworked into a new version of the Utzon a string grammar, as well as being classified as an SPR(s), meaning "Spatial Production Rules, string version". The Utzon SPR(s) better reflects the "mangled T" motif than SNARQ did, and it is cast in terms of an object oriented computer language.

2 Web Based Client/Server Architecture Machines

As is well known, the first version of HTML (Hyper Text Markup Language) is static. That is, the text that is received by a client was simply executed by the browser interpreter and displayed. The next versions of the web languages have become progressively more interactive.

SPR(s) are like the first static HTML. The generation of sentences using SPR(s) is a perfect match for web in that sentences are sent out as plain text. They can be considered as an architectural HTML, a mark up language for layouts.

In its present version, it is evident that, the web based architecture machines is a "one trick pony" in that the server can only produce Utzon sentences one at a time for one or more clients. It is not intended to be a general design tool for architects. It is intended as a first step in testing the idea of a web based generative system based on a specialized application client and server.

As in the case of HTML and related web languages, SPR(s) should become more interactive. In its current version, the architecture machines cannot provide a way for the architect to direct the generation process or send parameters to the server. Nor is there a billing mechanism for services rendered. However, in comparison to interacting with a generative process, billing is simple. In its most basic form, a client needs an account so that the server can add up charges for each sentence sent out. More sophisticated versions could take advantage of the Microsoft Transaction Server, or equivalent application servers.

Web based client/server architecture machines is an example of a three tier client/server system. This system is similar to the ubiquitous HTML web systems (Figure 1) except that instead of "serving up" web pages to client browsers, such as Netscape or MS Internet Explorer, they serve a wide variety of data produced by server application programs and sent to client application programs or browsers. A common example of server application programs is a super market inventory database. The application on the client machine can be a special purpose program or can be

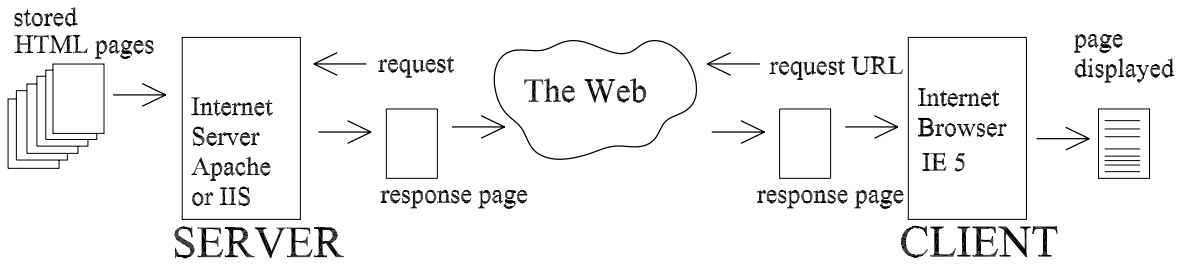

Figure 1. Typical 2-Tier Organization of HTML Web Systems.

plugged into a standard browser. (Edwards 1999; Thai 1999)

In the client/server architecture machine system, both parts contain application programs. The server program generates Utzon house layouts represented as sentences using the SPR(s) string grammar. It sends the sentence as plain text over the web to one of many possible client architecture machine. The client is a special purpose program embedded in Autocad. It interprets the sentence and as a result, produces an Autocad drawing that corresponds to the Utzon layout generated by the server. (Figure 2)

The use of the term "architecture machine" is borrowed from Nicholas Negroponte (Negroponte 1970). Its original meaning referred to a machine capable of creating architectural designs. He proposed the application of new insights and techniques developed in artificial intelligence to CAD. However, in this paper, the term is given two new meanings, depending upon whether it is a server or a client.

- As a server, an architecture machine generates all of the design variations of a particular architecture using an SPR of, for example, an Utzon housing system.
- As a client, an architecture machine is an interpreter of sentences of an architectural (computer) language. During the process of executing a sentence, the interpreter relies on a grammatical description of the language, in this case, the Utzon SPR(s). As a result, it produces an Autocad drawing of a layout of an Utzon house. The term "architecture machine" is a play on the words "virtual machine" which refers to language interpreters, such as those used in Smalltalk (Goldberg, A. and Robson, D. 1983) or Java. This may appear to some readers as rather obscure, but it is technically correct.

The architecture machine server receives one of two types of requests from the web:

- Start the generation of all of the Utzon sentences. Not only will the response include the first sentence, but also the list of SPR(s) rules used for generating this first sentence. The list is referred to as a "state".
- Generate the next sentence of the Utzon language given the state of the previous sentence. The response will not only include the generated sentence, but also its state.

In making the requests, the client architecture machine either sends no state, which means that it is requesting the first sentence, or sends a state, which means that it has *already received at least one Utzon sentence. The fact that the clients receive the state information along with the generated sentences relieves the server from having to maintain the generation sequence of sentences for each client on the web.*

The implementation of the web based architecture machines contains many parts, some of which are specially programmed in C/C++ and TM (Buchmann and Gerzso 1985), and some are standard products developed by Microsoft and Autodesk.

The server is made up of two main parts (Figure 2): the generator architecture machine and the Microsoft Transaction Server (MTS) –or equivalent-. The MTS receives requests from the web and routes it to the generator, which is written in TM and C/C++ for this project. The gruesome communication details between the generator and MTS are controlled by a Microsoft software component (COM) written in C++, also for this project. A COM is a standard program stored and used by one or more applications according the COM mechanism, which exist on all Windows systems, except Windows 3.x. In this case, its function is very simple. It receives the requests from MTS –or equivalent- and forwards them to the architecture machine. It waits for the resulting SPR(s) sentence and sends it back to the MTS.

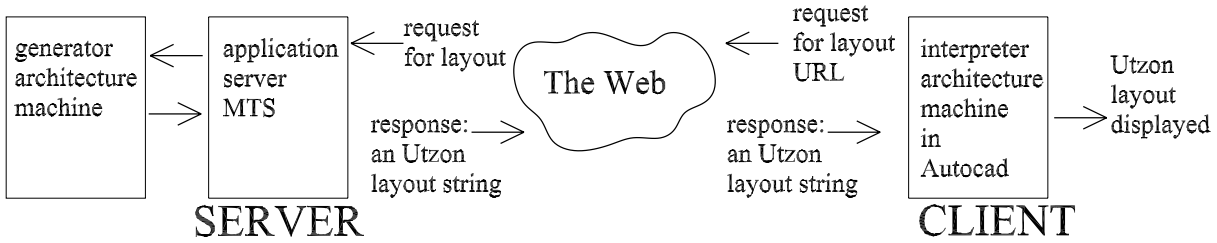

Figure 2. Web Based Client/Server Architecture Machines.

The client is also made up of two main parts: the interpreter architecture machine and Autocad. As in the case of the generator, the interpreter is written in C/C++ and TM and communicates with Autocad with a specially written COM.

3 The Utzon House System

As mentioned above, the motivation for selecting the Utzon house system was that it is a good example of a housing system comprised of a reduced set of well-defined spaces and building elements. Utzon designed it in 1969 (Phillip 1972) and named it the Espansiva Byg A/S Timber Component House System. The system was never used but a prototype was constructed at Gammel Hellebaek, North Zealand. The basic spatial components are:

- A hall module
- A bathroom and storage module of the same dimensions as the hall module.
- Specific function modules such as bedrooms, study or kitchen.
- General room modules (Figure 3)

The various modules are only joined in a limited number of ways. By analyzing the floor plans, Utzon seems to use the following joining rules (Figure 4):

- Hall modules are joined together either end to end or in an "L" configuration.
- Bathroom modules are placed at the end of an end-to-end sequence of hall modules. They can also be joined to this sequence either end-to-end or in an "L" configuration, like the hall modules.
- General purpose room modules are plugged into the hall modules. The short side of the room modules is equal to the long side of the hall modules.

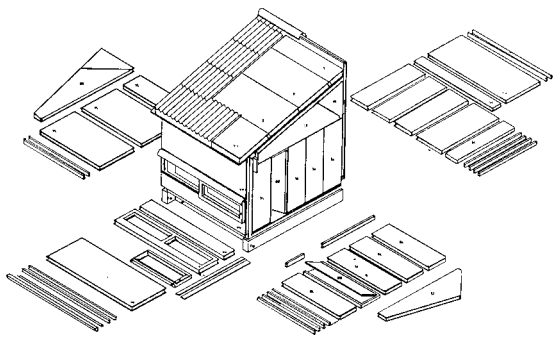

Figure 3. Room Module with Parts Breakdown.

4 The "Mangled T" Motif: The General Structure of the Utzon Grammar

Before describing the Utzon SPR(s) string grammar, an overview of its grammatical structure will be dealt with first. The grammar takes into account the fact that each house is a variation on the "mangled T" idea (Figure 4). The grammatical rules describe the valid ways in which to construct the many "T" layouts. But in order to make the spatial production rules readily understandable, it is useful to present the joining rules on a very general level first. A first approximation of the rules is illustrated in Table 1.

This verbal description of joining rules can be improved upon by means of a diagram (Figure 4). It can be read in the following way:

- Above the "start" node, either M_CHAIN or M_GROUP can be connected.
- Below the "start" node, either an "L" shaped pair of modules, a single hall module or nothing can be connected. This is the group "C".
- Below the group "C", either one of two bifurcation connectors "T" can be connected.
- Below the left "T" connector, either M_GROUP, MGB or M_CHAIN can be connected.
- To the right "T" connector, either M_CHAIN, MGB or M_GROUP can be connected.
- And so on.

An important thing to note is that the proper rotations of groups of modules as seen in Figure 5, such as M_CHAIN, are *included as part of the SPR(s) grammar* in section 5, in terms of operators such as "rotate_p90".

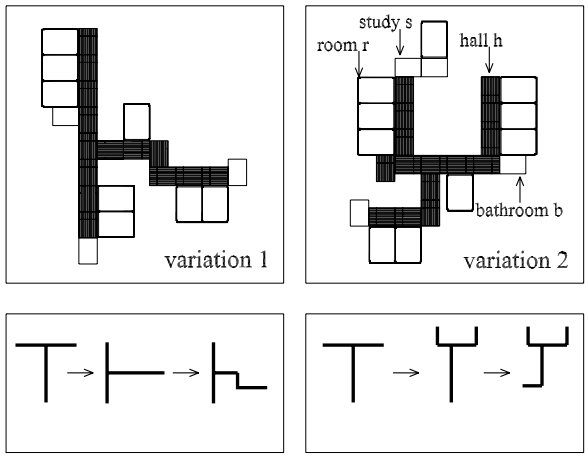

Figure 4. Two Layouts Designed by Utzon.

Table 1. A first approximation of the joining rules

1	house	an Utzon house is: -a chain of hall-room modules (hrm) joined to -a bifurcation or "T" joined to -an hrm chain
2	hrmc	a hall-room module chain is -one hrm group (hrmg) joined to another hrm OR -one hrm group
3	hrmg	an hrm group is -three hrm with ONE hall module as a connector OR -two hrm with ONE bathroom module with ONE hall module as connector OR -two hrm with ONE hall module as connector (hmc)
4	hrm	a hall room module is -a hall module joined to a room module.
5	hmc	a hall module connector is -a hall module joined to one end or as a corner

5 The Grammatical Structure of the Utzon Housing System

SPR(s) is one type of spatial production rules (see note 4). The definition of the Utzon SPR(s) is as follows:

Language SPR(s) = {start_symbol, nonterminals, terminals, production_rules}

The start symbol is UH.

The nonterminals are: UH, MC, M_CHAIN, C, C1, MC2, MC3, M_GROUP, M_G2, T, T0, T1, T11, T2, T3, MGRB, HC, M, BM, SM, HM, BLNKM.

The terminals are divided into five groups: the message passing operator, rooms, spatial joining operators, rotation operators, and joining rectangle stack operators.

- The message passing terminal is "<="
- The rooms are: r, b, s, h which correspond to room, bathroom, study, and hall modules (Figure 4).
- Spatial joining operators: jr, jt, jl, jb, and jo (Figure 5).
- Rotation and mirroring operators: rotate_p90 (Figure 7), rotate_n90, rotate_p180, and mirror_v .
- Joining rectangle stack operators: set_frame, popN, sjsN, pet, peb. These operators are NOT described in detail in this paper.

The rules are illustrated in Table 2.

An important characteristic of the Utzon SPR(s) is the message passing operator. The idea is that a room or group of rooms are objects in the object oriented programming sense, and that operations are carried out by means of sending a message to them. For example,

r <= jb h

Table 2. Spatial Production Rules

1		UH	→ MC <= jb C
2	a	MC	→ M_CHAIN <= pet
	b		→ (M_GROUP <= rotate_p90)
3		M_CHAIN	→ M <= jl HC <= sjs2 <= jb (M_GROUP <= rotate_n90)
4	a	C	→ C1 <= jb T <= pet <= jb (HM <= rotate_p90) <= jb MC2
	b		→ C1 <= jb (HM <= rotate_p90 <= sjs5) <= pet <= jr (T <=rotate_n90) <= pet <= jr MC3
5	a	C1	→ (HM <= jr (HM <= rotate_p90 <=sjs2 <= pet) <= pet
	b		→ (HM <= rotate_p90) <= pet
	c		→ <nothing>
6	a	MC2	→ (MGB <= rotate_p90 <= mirror_v)
	b		→ (M_CHAIN <= pet <= rotate_p180)
	c		→ (M_GROUP <=rotate_n90)
7	a	MC3	→ (MGB <= mirror_v)
	b		→ (MGB <= rotate_p180)
	c		→ (M_CHAIN <= ped <= rotate_p180)
	d		→ M_GROUP
	e		→ (M_GROUP <= rotate_p180)
8	a	M_GROUP	→ M <= jr M_G2
	b		→ M_G2
9		M_G2	→ M <= jr M
10		T	→ HM <= rotate_p90 <= sjs2 <= jr HM <= jr T0 <= peb
11	a	T0	→ T1
	b		→ T2
	c		→ T3
12		T1	→ M <= jr T11
13	a	T11	→ (MGRB <= rotate_p180)
14	b		→ (HM <= rotate_p90 <= jb MGRB) <= peb <= sjs2
15		T2	→ HM <= jr T11
16		T3	→ (HM <= sjs2 <= jb MGB <= mirror_v <= rotate_n90 <= peb)
17		MGRB	→ MGB <= jr HM <= pet <= sjs2
18	a	MGB	→ BM <= sjs5 <= rotate_p90 <= jr M_G2 <= pet
19	b		→ BM <= jr MG2 <= pet
20	a	HC	→ HM
	b		→ SM
	c		→ HM <= jl BM <= p1
21		M	→ r <= jb HM <= pet
22		BM	→ b <= jo BLNKM <= set_frame
23		SM	→ s <= jo BLNKM <= set_frame
24		HM	→ h <= jo BLNKM <= set_frame
25		BLNKM	→ ((e1 <= jr e2) <= jo (e2 <= jr e1))

means "send a message to the room 'r' to join itself to the bottom to the hall module 'h' ". The joining together of spaces is done by means of a joining rectangle, described in section 7. The result of this operation is a new joining rectangle which covers both "r" and "h". In Figure 5, the same joining operation is illustrated, but the receptor is "joining_r" and the space being joined to it is "r". This idea is a variation of Shaw's picture grammar (Shaw 1970). Instead of picture lines, they basic elements are architecture spaces.

Terminals such as "r", "b", etc. *are not variables*, as would be the case if the SPR(s) were a typical computer programming language. They represent literal architectural spaces.

The rules of the SPR(s) are a way to represent all of the Utzon house system variations. The idea is

Table 3. Derivation of layout variation 1.

	Sentence	State
Step 1		
UH =>	MC <= jb C	rule 1
Step 2		
=>	M_CHAIN <= pet <= jb C	rule 2a
Step 3		
=>	M <= jl HC <= sjs2	rule 3
	<= jb (M_GROUP <= rotate_n90)	
	<= pet <= jb C	
Step 4		
=>	r <= jb HM <= pet	rule21
	<= jl HC <= sjs2	
	<= jb (M_GROUP <= rotate_n90)	
	<= pet <= jb C	
Step 5		
=>	r <= jb h	
	<= jo BLNKM <= set_frame	rule 24
	<= pet	
	<= jl HC <= sjs2	
	<= jb (M_GROUP <= rotate_n90)	
	<= pet <= jb C	
Step 6		
=>	r <= jb h	
	<= jo ((e1 <= jr e2) <= jo (e2 <= jr e1)) <= set_frame	rule 25
	<= pet	
	<= jl HC <= sjs2	
	<= jb (M_GROUP <= rotate_n90)	
	<= pet <= jb C	
Step 7		
	r <= jb h	
	<= jo ((e1 <= jr e2) <= jo (e2 <= jr e1)) <= set_frame	
	<= pet	
	<= jl HM <= jl BM	rule 20c
	<= p1 HC <= sjs2	
	<= jb (M_GROUP <= rotate_n90)	
	<= pet <= jb C	
Step 8		
	r <= jb h	
	<= jo ((e1 <= jr e2) <= jo (e2 <= jr e1)) <= set_frame	
	<= pet	
	<= jl h	
	<= jo ((e1 <= jr e2) <= jo (e2 <= jr e1) <= set_frame	rule 24
	<= p1 HC <= sjs2	rule 25
	<= jb (M_GROUP <= rotate_n90)	
	<= pet <= jb C	
Step 9	Etc.	

that all of the sentences of this language "L" correspond to all of the Utzon house layout drawings. This is an example of a "verbal-visual equivalence" (Fleisher 1992). But in order to make this equivalence work, rotations and mirroring of rooms are included in the grammar as terminals. So are operators on a joining rectangle stack, which controls the selection of the joining rectangle before a joining operation is carried out (Figure 7). (Even though it is critical in the interpreter, he stack is not described, because it not essential for understanding the basic idea of this paper).

6 The Generator Architecture Machine

The generator takes the rules and applies them to the nonterminals systematically from left to right. It applies the rules blindly in the sense that it does not deal with the meaning of the sentence, or any part thereof. It only constructs grammatically correct sentences according to the SPR(s) rules.

The derivation that appears below in Table 3 corresponds to layout variation 1 (Figure 4).

Upon completion of the derivation, the generator sends out the sentence and its corresponding state (the list of rules used in generating this sentence) to a client on the web. For example,

sentence:

r <= jb h <= jo ((e1 <= jr e2) <= jo (e2 <= jr e1)) <= set_frame <= pet

<= jl h <= jo ((e1 <= jr e2) <= jo (e2 <= jr e1) <= set_frame <= p1

<HC derivation completed> <= sjs2 <= jb

(<M_GROUP derivation completed> <= rotate_n90) <= pet <= jb C

state:

[rules: 1,2a,3,21,24,25,20c,24,25 ……etc]

7 The Interpreter Architecture Machine

In comparison to the generator, the interpreter is much more complicated. To begin with, it must group characters into syntactic and semantic units. For example, it must determine that the string "r <= jb h <= jo " is actually a sequences of units "r", "<=", "jb", "h" and so on. It must then determine that some of these units are objects, such as "r", which means "room", and some of them are operators, such as "jb" which means "join receptor space at the bottom to a parameter space", and so on. The unit "<=" means "send a message to a receptor space and request it an operation". The

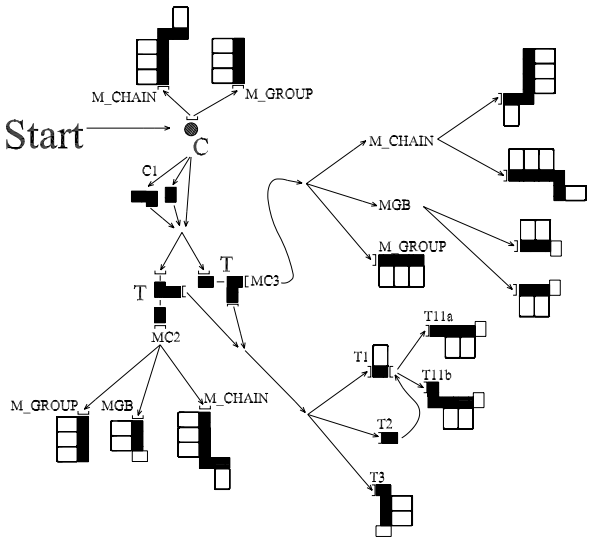

Figure 5. The Structure of the SPR(s) Grammar.

interpreter can only accept previously defined objects –that is spaces- and operators.

Once the interpreter has isolated the units, it must then determine if their combination is syntactically correct. It uses the SPR(s) rules to do this. For example, if the string contains "r <= <= r jb jo jl", the interpreter will at least identify that "r <= <=" is incorrect.

Having determined syntactic correctness, the interpreter then looks for semantic units in the sentence. The first thing it does is to identify the combination of a receptor, a "<=", an operator, and finally optional parameters. If successful, it carries out the operation. (Aho and Ullman 1977)

For example, in the fragment:

r <= jb h <= jo ((e1 <= jr e2) <= jo (e2 <= jr e1)) <= set_frame ….

the first semantic unit is "r<= jb h", the second one is "<result of the first operation> <= (<three operations done here>)" and the sixth one is "<result of the FIFTH operation> <= set_frame…". The result of the operation is a portion of the layout displayed in Autocad. A more detailed description of this process can be seen in Figure 8.

The SPR(s) can be interpreted because there has been a set of fundamental spatial operators defined in the language: jr, jt, jl, jb, and jo, among others. The joining operators can only work on two rectangles (squares) at a time. The rectangles are joined only if sides are of equal length. The result of a joining operation is a new joining rectangle, which covers the two spaces joined (Figure 6). If this new rectangle is to be joined again, then the same operators can be used and the same restriction of only joining equal sides applies (Figure 8). (In these figures, the shaded area is the joining rectangle).

In applying this approach to constructing the Utzon house layouts, two problems must be overcome. The first one has to do with being able to join together two hall modules in an "L" configuration (Figure 4, and step 1 in Figure 8). It is not possible to join the short side of a hall module to a long side of another one by using the idea of joining rectangles. The solution is to overlay blank modules "e1" and "e2" on a hall module (Figure 7). In this paper, this group of modules is frequently referred to as "HM" (rule 24). Therefore, if two hall modules need to be joined in the "L" configuration, then the short side of one "h" is joined to one of the "e1" modules of an "HM" module.

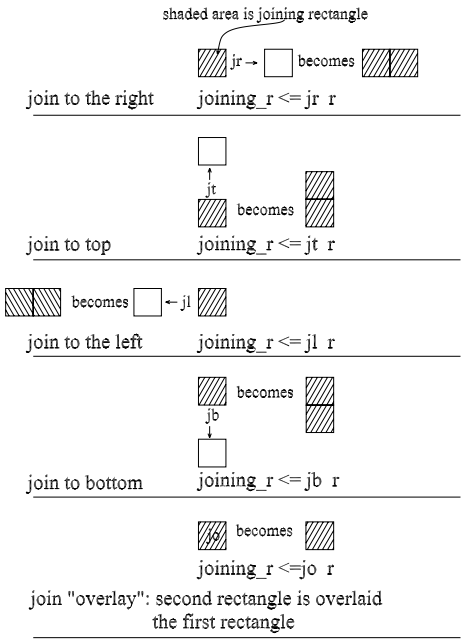

Figure 6. Operations of the Joining Rectangle.

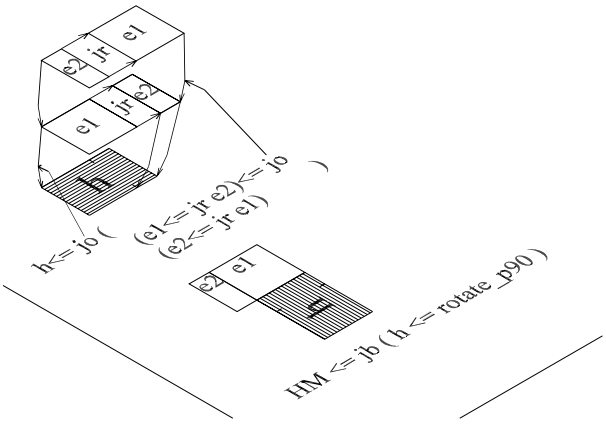

Figure 7. The Construction of the Hall Module (HM) with Overlaid Blank Rectangles.

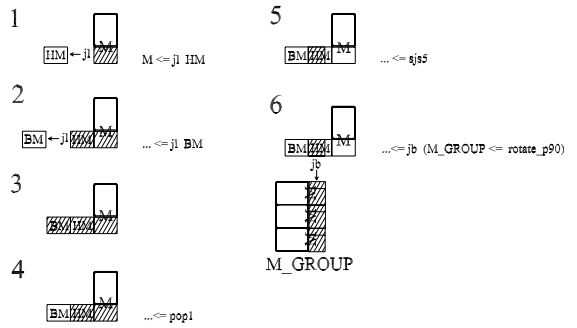

Figure 8. Sequence of Operations of the Interpreter to Construct Part of Variant 2.

The second problem has to do with being able to select a joining rectangle in a group of modules that have already been joined. As modules are joined, the joining rectangle gets larger and larger (steps 1, 2, and 3 in Figure 8). Without selector operators, it would not be possible to select the joining square "e1" in step 5, Figure 8 in order to perform the operation in step 6.

8 Conclusion

The conclusion of this paper can be divided into two parts: the first one has to do with grammatical generation of layouts, and the second one has to do with the idea of a web based generative system for architecture.

The idea of representing design rules grammatically is an idea that goes back to 1972. However, by applying this approach to generate layouts on the web, several things have become very clear that may not have been obvious in 1972. First, the design or layout rules like those of the Utzon SPR(s) appear to be purely grammatical. But this appearance is deceptive. The fact that these rules can represent an Utzon layout has more to do with the definition of the message passing mechanism of computation, joining rectangle, the joining operators (jr, jt, jl, etc) and the joining rectangle selectors (sjsN, popN, etc) than the syntax of the SPR(s). In computer science jargon, this would be referred to as the "joining rectangle semantics". Without message passing, operators or selectors, the Utzon SPR(s) would not mean anything besides a grammatically correct way of combining syntactic units such as "r", "<=", and "jr".

The generative system applying the Utzon SPR(s) in this paper bolsters the argument made by Aaron Fleisher (Fleisher 1992) that representing design rules by purely grammatical means is very limited. To make matters worse, the limitation also has to do with the fact that a string grammar like SPR(s) represents design rules in one notation, e.g. a string of text, of a building layout, which is usually represented by a different "notation", namely a drawing. The problem is to assume that there is a strict correspondence between the two, which Fleisher "verbal-visual equivalence". Section 7 illustrates why this is problematic. The reason has to do with the fact that the logic of an SPR(s) does "not (always) hold across a visual verbal switch". It is also referred to as the problem of indirection (Dalrymple and Gerzso 1998) that is a layout is indirectly represented by some other means, such as a text string or text grammar.

It is possible that too much attention has been spent on the grammatical structure of SPR(s) and shape grammars, and not enough on the web.The reason is that the web is the easiest part to implement. With servers and clients, the sending and receiving of plain text is straightforward. The complicated part is generating the plain text and interpreting it.

Despite its limitation, SPR(s) string grammar representation of Utzon design rules has provided a useful way to test a first version of a web based generative system. Such as system opens interesting possibilities, such as a centralized depository of architectural production rule systems. Defining and maintaining production rule systems is usually beyond their technical expertise for the everyday architect. Assuming that there is a real need for it, an alternative could be a centralized system, which provides a service for the architectural community. The question would then be: what would be the characteristics of such a service? This paper has dealt with only the feasibility of such as service by means of a web based architecture machines.

Acknowledgments

The work reported here would not have been possible without the advice of J.P. Protzen, L. Zadeh, N. Negroponte, A. Fleisher and R. McCall.

Notes

1) The use of the term "string grammar" refers to the standard notion of a Context Free Grammar which was first proposed by Chomsky and widely used in computer science (Aho and Ullman 1972).
2) The person who originally suggested to that I look into Alan Shaw's picture grammar work (Shaw 1970) as starting point for thinking about a "computer language" for architecture was L. Zadeh. Subsequent work was also based on Zadeh's suggestion of considering other approaches to graphical grammars: plex and web grammars (Gerzso 1978).
3) Having grown up in a household with painters and musicians, I have always been extremely skeptical of making aesthetics or "art" computable. But at the same time, this experience has also taught me that artists do not make purely random decisions during their creative process. They frequently use rules.
4) The SPR(s) is a type of string version architectural (spatial) production rule system. There are other versions of SPR's which attempt to overcome the problems related to the "verbal-visual equivalence" problem. SPR(l), SPR(a), SPR(d) are line and graph based SPR's (Gerzso 1978). Other ones are SPR(qt) and SPR(ed) (Gerzso 2000)
5) SPR(a) and SPR(d) are used in conjunction with transformational rules. They are used to model variations of functional layouts in San Francisco Victorian housing and SAR housing systems. (Gerzso 1978)
6) This reference is included because it is used by Gips. It was not consulted for this paper.

References

Aho, A., and J. Ullman. (1972). *The Theory of Parsing, Translation and Compiling, Vol. 1, Parsing*. Englewood Cliffs, N.J.: Prentice Hall.

Aho, A., and J. Ullman. (1977). *Principles of Compiler Design.* Reading, Mass.: Addison-Wesley Publishing Co.

Buchmann, A.P., and J.M. Gerzso. (1985). TM: An Object-Oriented Language For CAD and Required Data Base Capabilities. *Proceedings of the IEEE Workshop on Languages for Automation.* New York, N.Y.: The Institute of Electrical and Electronic Engineers.

Dalrymple, M.J. and J.M. Gerzso. (1998). Executable Drawings: The Computation of Digital Architecture. *Proceedings of Acadia'98.* Eds. S. Van Wyk and T. Seebohm. Quebec City.

Edwards, J. (1999). *3-Tier Client/Server At Work*. New York, N.Y.: Wiley Computer Publishing.

Fleisher, A. (1992). Grammatical Architecture?, *Environment and Planning B*, Vol. 8.

Gerzso, J.M. (1974). The Hunting of the SNARQ and Other Tales of Computer Languages for Architecture. *DMG Journal, January.*

Gerzso, J.M. (1978). *A Descriptive Theory of Architectural Built Form and its Applications.* Ph.D. Dissertation, University Microfilms.

Gerzso, J.M. (2000). Generating Trullo Italian Vernacular Architecture with Production Rules and Executable Drawings. *Proceedings of the Millennial Open Symposium on the Arts and Interdisciplinary Computing.* Eds. D. Salesin and C. Sequin. Seattle, Wa.: University of Washington.

Gips, J. (1974). *Shape Grammars and Their Uses.* AIM 231, Stanford Artificial Intelligence Laboratory, Dept. of Computer Science, Stanford University

Goldberg, A., and D. Robson. (1983). *SMALLTALK-80, The Language and Its Implementation*. Reading, Mass: Addison-Wesley.

Negroponte, N. (1970). *The Architecture Machine: Toward a More Human Environment.* Cambride, Mass.: MIT Press.

Mitchell, W.J. (1990). *The Logic of Architecture.* Cambridge, Mass: MIT Press.

Phillip, D. (1972). *The Third* Generation. New York, N.Y.: Praeger Press.

Shaw, A. (1970). Parsing of Graph-Representable Pictures, *Journal of the ACM*, Vol. 17, No. 2.

Smith, N., and D. Wilson. (1979). *Modern Linguistics, The Results of Chomsky's Revolution.*Bloomington, Ind.: Indiana University Press.

Stiny, G. and J. Gips (1972). Shape Grammars and the Generative Specification of Painting and Sculpture. In *Proceedings of IFIP Congress 71*, Amsterdam: North Holland Publishing Co., also in O. Petrocelli (ed.), *The Best Computer Papers of 1971*, Auerbach, Inc. (see note 6).

Stiny, G. (1980). Introduction to Shape and Shape Grammars, *Environment and Planning B*, Vol. 7.

Stiny, G. and Mitchell, W. J., (1980). The Grammar of Paradise: On the Generation of Mughul Gardens, *Environment and Planning B*, Vol. 7.

Rozenberg (1997). *Handbook of Graph Grammars and Computer by Graph Transformation.* World Scientific.

Thai, T. (1999). *Learning DCOM*, Sebastopol, CA: O'Reilly & Associates.