The LaHave House Project: Towards an Automated Architectural Design Service

Andrew Rau-Chaplin†
Technical University of N.S.
School of Computer Science
Box 1000, Halifax NS, Canada
arc@tuns.ca

Brian MacKay-Lyons
Architecture + Urban Design
2042 Maynard St, Halifax NS, Canada

Peter F. Spierenburg‡
Technical University of N.S.
School of Computer Science
Box 1000, Halifax NS, Canada
spierepf@tuns.ca

Abstract

The LaHave House Project explores the creation of an automated architectural design service based on an industrial design approach to architecture in which Architects design families of similarly structured objects, rather than individual ones.

Our current system consists of three software components: 1) A design engine that uses shape grammars to generate a library of preliminary level house designs, 2) A design development tool that permits end-users to selected, customize and visualize designs drawn from the library and 3) A building systems configuration tool that transforms customized designs into working/assembly drawings.

Our aim is to generate modern realizable houses that combine beautiful forms with a modern approach to space planning. We are currently completing an integrated on-line prototype that allows end-users to select, customize and visualization generated house designs over the Internet in 3D, using a Java/VRML based design development tool.

Keywords: Automated Architectural Design Service, Shape Grammars, Generative Expert Systems, Knowledge-based Computer Aided Architectural Design, Prolog, VRML, Java.

1. Introduction

The LaHave House project is an ongoing research project of the Faculties of Architecture and Computer Science at the Technical University of Nova Scotia, Canada. The goal of the project is to explore the potential for an *industrial design approach* to architectural design in which Architects design families of similarly structured objects, rather than individual objects, thereby amortizing design costs.

Currently in North America architects are involved in the design of only about 5% percent of the total new house market. Whereas custom architectural design will always have a premier role to play, we believe that an industrial design approach to architecture can bring much of the design quality and variety of custom design to the other 95% of the market, at an affordable price.

At the heart of the project is the use of generative grammars to build design libraries.
We are interested in exploring the creation of an *automated architectural design service* based on such design libraries. The LaHave House grammar is derived from on the work of Brian MacKay-Lyons [2,4] and is inspired by the vernacular architecture of the LaHave river valley in Nova Scotia. It uses abstract versions of archetypal forms of buildings in this region and focuses on simple detailing and efficient construction.

We are interested in generating modern realizable houses. Houses that combine beautiful forms with a modern approach to space planning. Furthermore, we want the generated representations to be at the level of detailed preliminary design, in that functional issues concerning bathrooms, closets, door swings etc. have been addressed.

To support our vision of an automated architectural design service, we have developed a prototype CAAD tool that enables a end-user to select, customize and visualize designs drawn from our design library. We are currently in the process of constructing a single integrated prototype that is capable of running on-line over the Internet using a Java and VRML enabled World Wide Web browser.

† Research partially supported by the Natural Sciences and Engineering Research Council of Canada.
‡ Research partially supported by the Technical University of Nova Scotia - School of Computer Science.

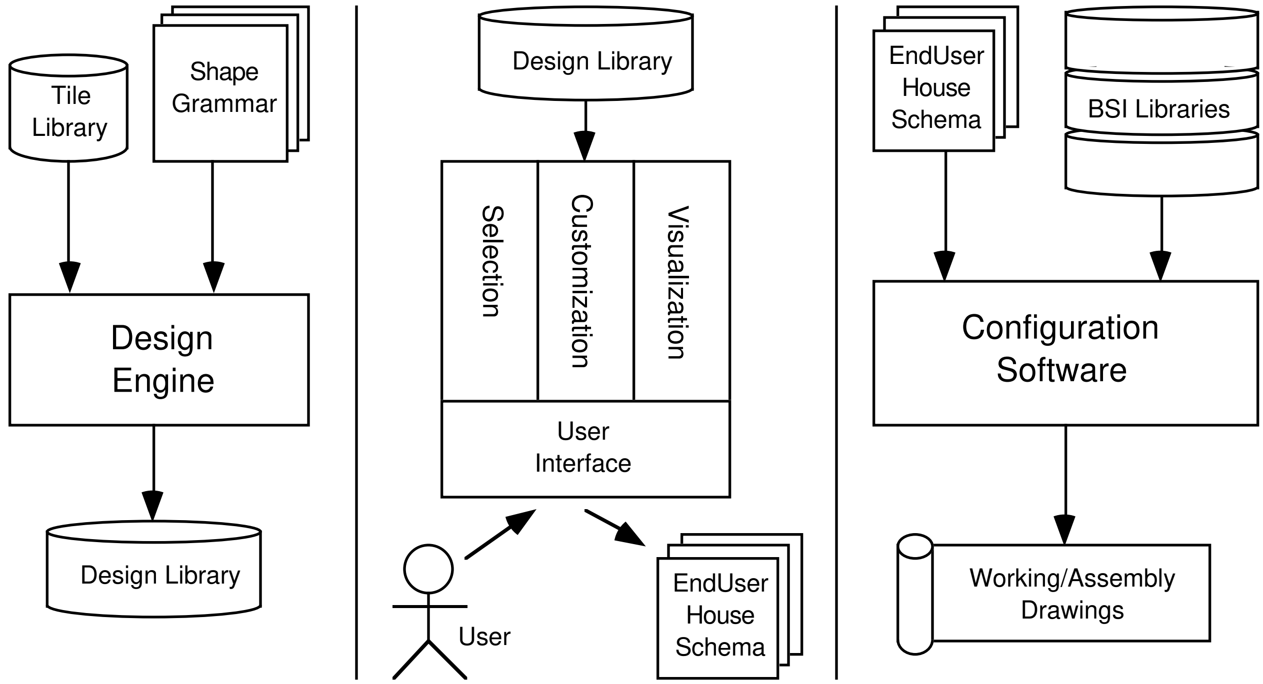

Figure 1: Components of LaHave House System

2. Overview

The LaHave House System consists of three major software components (see Figure 1):

1) A Design Engine that generates a library of house designs.

2) A Design Development Tool that allows an end-user to Select, Customize and Visualize designs.

3) Building Systems Configuration software that transforms developed house designs into sets of working/assembly drawings.

In this paper we will focus on our approach to design generation and the design development tool. The building systems configuration software is currently in a very preliminary form and will be discussed elsewhere.

2.1 The design engine

The role of the design engine in the LaHave house project is to generate a library of "base house designs" which can be used as a starting point for an end-user driven design development process. One can think of this library as the digital analog to the pattern books of the 19th century in that both contain house organizations, as much as they contain individual house designs. The generated design library contains houses that differ radically from each other in form, organization, size, amenity level, and "style", but despite this diversity share an underlying deep structure. This shared deep structure results from them all being productions of the same Shape Grammar [5,10] and is intrinsic to the systems as a whole, because it is precisely this shared deep structure that allows us to create an effective design development tool. Each base house design or *house schema* consists of a complete description of the geometry of the house and the layout of each floor. The design of individual spaces in the house, for example the layout of the kitchen or bathrooms, is not currently generated rather pre-designed room arrangements, called *tiles,* are allocated from a tile library. Our current tile library consists of over 500 room tiles and 200 wall tiles.

The LaHave house design engine and grammar will be described in more detail in Section 3.

2.2 The design development tool

In order to support our goal of an automated architectural design service we needed to build a user interface that was appropriate for end-users with little or no design experience. On the one hand, the tool needed to be powerful enough to give end-users the ability to create highly idiosyncratic designs. While on the other hand restrictive enough to ensure that neither the architectural nor structural integrity of the house design was compromised. Clearly the usability of the design development tool hinges on striking the right balance between these two opposing design goals. Our approach to date has been to combine "design by selection" [5] with a powerful, but restricted, form of user customization and to provide tools to help the user visualize the consequences of their design decisions. Our current design development tool consists of a shared user interface and three software components supporting Selection, Customization and Visualization, respectively (see Figure 1).

Selection. Our approach to determining the user's requirements is modeled largely on a typical "first meeting" between Architect and Client. In order to determine the users program (requirements), the user is asked to complete a "questionnaire" consisting of approximately twenty questions concerning Budget, Site, Space, and Style issues. In keeping with the idea of a design service each question is accompanied by a window of "Architectural Advice" which attempts to explain to the user the ramifications and tradeoffs inherent in each decision.

The *Budget questions* address the issues of budget range and level of interior finish. The *Site questions* address issues such as the site's width, relationship to the road, slope and orientation with respect to the sun. The *Space questions* concern the number of various rooms required (i.e. # of bedrooms, bathrooms, etc.) and issues of space planning. The *Style questions* concern a range of issues from roof shape, to the degree of symmetry and articulation of form. Some style questions take the form of a text-based question, while others require the user to select images that they particularly like or dislike. Getting at a users likes and dislikes is not always easy is this straight question and answer format. We hope in the future to explore other approaches to eliciting qualitative requirements perhaps along the lines described in [7].

Having answered these questions the user is presented with a matrix of external 3D views of base house designs that satisfy their requirements according to their responses to the questionnaire. Associated with each base house design is a "digital brochure" consisting of a set of floor plans and summary information (i.e. square footage, # of bathrooms etc.) in the form of a typical "real estate cut sheet".

By a process of examination and elimination the user is expected to decide on a single base house design which best satisfies their needs/dreams. This design then becomes the starting point for customization.

Customization. The task of the Customization component is to allow the user to modify/evolve their base house design into one that more fully meets their needs. The current customization tool is a constraint based editor that allows the user to perform the following basic actions:

1) Replace one tile (room organization) with another - provided the new tile is "compatible" with its neighbor, has associated wall types that are compatible with its neighbors, and maintains the connectivity of the house's underlying circulation graph.

2) Add a secondary form ("bump") to the house design - provided it is compatible with its neighbors (as determined by information computed by the grammar at generation time) and that it does not violate any of a set of important 3D constraints (e.g., second story secondary forms must be over first story ones).

The user interface is based on the idea of replacement (see Figure 2). The user is presented with one window containing a floor plan and another that will display possible replacement tiles. When the user selects a room or wall tile in the plan window the interface computes, based on the constraints, a set of possible replacements that are then displayed in the replacement window. The user can then browse the set of possible replacements and select one to effect the replacement.

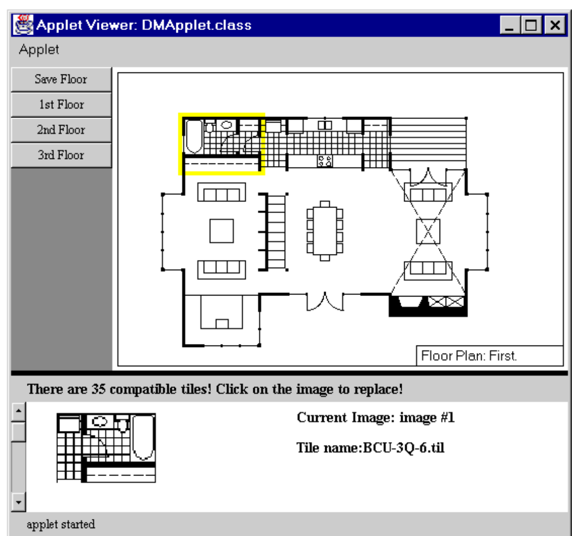

Figure 2: Customization Applet

Addition of secondary forms is handled in the same way, with the user selecting a "glue line" in the plan window which is a special type of tile that is created during generation to express the potential for adding various kinds of secondary forms to a particular region of a plan. Secondary forms currently include bay windows, small rooms (study, den, sun-room etc.), balconies, and decks. We have focused on the addition of secondary forms as they are far less likely to adversely effect the massing and proportions of the house as carefully constructed by the design engine. The customization tool also supports the modification of internal partitions and the editing of fenestration.

It is interesting to note that both the simplicity and power of the customization tool are directly derived from the fact that the designs to be edited were created by a shape grammar. The underlying deep structure in the designs allows individual parts of the design to be reworked relatively independently. The constraints that maintain the geometry, circulation and servant/served relationships between spaces can be effectively expressed in terms of relationships that the grammar creates and records in the house schema data structure.

Although we have come a long way there are still many challenges to be addressed in customization. The issue of how to handle overly constrained planning situations is particularly challenging. Also, although the customization tool maintains important physical constraints, it does not maintain a "reasonable plan" from the perspective of function planning. For example, it will permit a user to place the dinning room and kitchen at extreme opposite ends of a floor plan without comment. The issue here is one of locus of control, and in the case of function planning we feel it is best to rely on the users superior knowledge of their needs and common sense. We are currently examining whether the use of a "design critic" [8] might not aid the user in functional issues, without directly constraining them. It would also be interesting to explore a more cooperative approach, like the one described in [1], in which the locus of control moves back-and-forth between end-user and the design development tool.

Visualization. At any point during customizing the user may need to visualize their evolving house design in 3D in order to really understand it. The task of the Visualization component is to transform a house schema into a complete 3D model. The Visualization component consists of two programs. The first is a compiler that parses a house schema and, using a 3D kit of pre-manufactured parts/models (analogous to the 2D tile library), constructs on-the-fly a complete 3D model of the house. The second program is a 3D user interface or browser that allows an end-user to view and "walk-through" the completed 3D model.

{ EMBED Word.Picture.6 }

Figure 3: Generated 3D View of an Example House

Our initial implementation of the Visualization component was prototyped in a programmable CAD package with basic rendering features (see Figure 3). This allowed us to explore the challenges of on-the-fly 3D model construction in an environment rich in primitive geometric operations, but was much too slow and required each user have an expensive CAD package.

We are currently completing an on-line Internet based version of the Visualization component that uses the World Wide Web based VRML format to describe the models. These models can then be viewed and "walk-through" using any of several widely available VRML broswers. VRML appears to be a good platform for supporting on-line user-driven walk-throughs, but currently is rather limited in its rendering and interactive facilities.

Again there remain many interesting open questions concerning visualization. In particular we are interested in the architectural presentation issues of how best to present 3D representations of structures to end-users. We are exploring how abstract 3D models that convey the underlying structure of LaHave house designs, rather than presenting realistic models, might be used to better "reveal" the design to users.

Our current design development tool is very much a prototype; an environment in which to explore design development by watching an end-user at work. After constructing initial prototypes of various parts of the tool using a diverse range of tools from Prograph to AutoCAD we are now in the process of constructing a single integrated prototype that is capable of running on-line over the Internet using a Java and VRML enabled World Wide Web browser.

3. Generating the design library

3.1 The grammar

The LaHave House grammar is based on a systematic approach to house design developed by Brian MacKay-Lyons in over fifteen years of custom architectural design practice [2,4] The forms generated by the grammar have been inspired by the vernacular architecture of the LaHave river valley of Nova Scotia, Canada. Using shape grammars to capture the design space of a living architect is in many ways different to much of the existing shape grammar work [3,9]. The goal is not so much to create a grammar that generates an existing corpus of design, but rather to work with the Architect to extract from their existing design corpus a robust set of generation principles.

One feature of the grammar is its tendency to produce dense cores for services and sparse open spaces for living in. The grammar is constructed in terms of a set of five elementary components: Rooms, Tartans, Machines, Bays, and Totems. The rooms are the principle places for human action, the tartans provide space for circulation, the machines (bathrooms, kitchens, laundry, entry etc.) are the dense service spaces, the Bays provide outlook space and secondary living space, and the totems (hearths, staircases, cabinetry etc.) provide focus for the rooms.

The grammar makes much use of the idea of Served Vs. Servant spaces advanced in [6]. This tends to be a three level hierarchy in which rooms are served by primary machines, which may in turn be served by secondary machines. For example, a dinning room may be served by a kitchen, which is in turn served by a pantry.

Although we are interested in generating complete functional modern houses, (i.e. we care where the bathrooms and closets go), the grammar is driven purely by issues of form. The function plan is derived only when the form has been completely generated.

The grammar we are currently working with has a single longitudinal axis of growth, with a dense machine zone flanking a body zone, which is in turn surrounded by a bay zone (see Figure 4). The machine zone tends to be contiguous and completely filled, particularly in smaller houses, while the bay zone tends to be more sparsely populated.

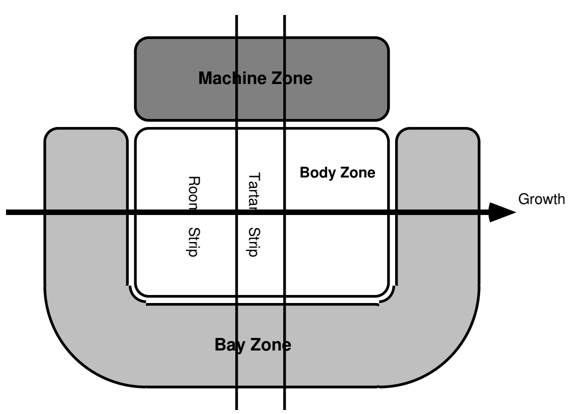

Figure 4: Space Zoning

3.2 A three phase approach

Our current design engine is based on a hybrid approach to the generation of designs.

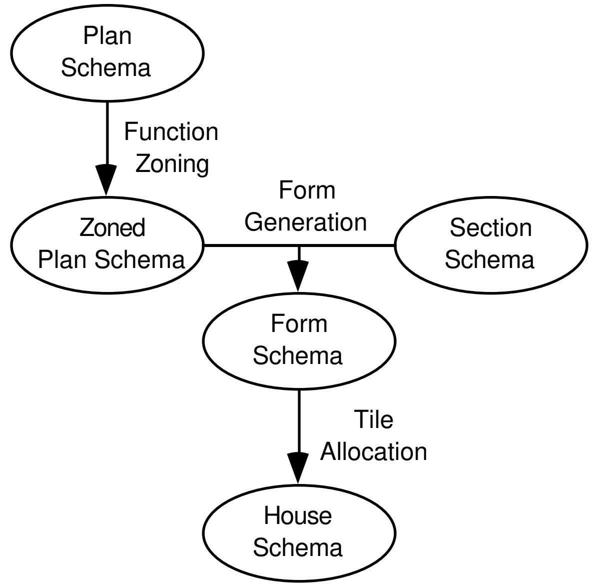

Figure 5: Phases of Generation

A hierarchical set of Shape Grammars as popularized by Stiny and Mitchell [5,10] are used to generate both *plan and section schema*. A set of backward chaining rules are then used to identify/recognize within plan schema possible groupings of individual spaces into abstract *functional zones* denoting Public, Private and Semi-private spaces. Given compatible plan and section schema, a *form schema* can now be created. This entails creating floor plans (from the plan schema), dimensioning these plans, creating walls to define/enclose the interior spaces, adding roofs and generally computing all of the 3D information required to define a complete form. Finally, form schema are transformed into completed house designs (*house schema*) by a rule based system that first assigns to each space unit within the form schema a function (i.e. kitchen, bedroom, entry, study etc.) and then assigns a room organizations from a library of room level designs. We call these pre-designed room organization *tiles* and the problem of assigning tiles to space units, in a pleasing and functional arrangement, the *tile assignment* problem.

The current design engine is implemented SWI-PROLOG and, whereas some attempt has been made to create a general design generation tool, much of the structure of the current system is specific our underlying grammar.

A difficult question, one which we have yet to answer to our own complete satisfaction is, “when are two houses sufficiently different to warrant both being included in the library?” Currently we take a pragmatic approach to this issue. At the level of form schema, we exclude simple mirror images and rotations from the library, also minor variations in secondary forms or roof shape are excluded, except where they have significant effect on the available floor space. At the level of tile allocation we only generate multiple house schema from a single form schema when the tile allocation process has led to significantly different circulation/living patterns. Just switching the position of the living room with the dinning room does not warrant the production of a new house design!

Currently the design engine generates over 100,000 different house designs and whereas some of them are "duplicate designs" arrived at by different paths, we believe that overall they strike a reasonable balance between variety and needless variation. There are many interesting problems within this context still to be investigated.

3.3 Issues in generation

The principle challenge that must be addressed in the mass generation of house forms is how to avoid the inherent combinatorial explosion. In our current section grammar, for example, the bay and body zones come in three widths each, the machine zone in two. There can be up to three stories in the body zone, and two in each of the machine and bay zones. There are nine roof styles for primary roofs and five for secondary roofs. Thus, there are nearly fifty thousand different fundamental cross-sections of which less that 10% are valid.

The central challenges in addressing this combinatorial explosion are 1) How can you deal with a space that can increase n-fold with the addition of a single new factor? and 2) How can you be sure that all the schema being generated are truly valid when there are far too many to manually examine? After all, perhaps a certain case has been neglected, or worse, poorly programmed.

As we see it, there are two opposing approaches to generating valid schema. The first is to generate all possible combinations of all the elements, and then prune away all those which prove invalid using a system of validation rules. The basic advantage of this approach is simplicity. The generation rules are relatively simple and regular and each completed production can be validated by a equally simple and regular set of rules. There are two primary disadvantage to this approach. Firstly, it is unlikely that one has enough time or space to generate all possible configurations before testing for validity. Secondly, if a generated schema is invalid for more than one reason, (i.e. it has features A, B, and C which are each invalid by themselves) then it is possible (and as it turns out quite likely) that the combination of invalid features are jointly consistent enough to slip past the validation rules. For example, in the Figure 6, there are two invalid features. The left hand secondary form is too large for the primary form it connects to, and the roof of the primary form is too shallow to support a second story:

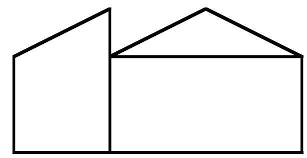

Figure 6 - An Invalid Section Schema.

The rules which prevent these features from occurring are given below.

invalid_section
  if roof is not flat **and**
    stories in body < stories in bump.

invalid_section
  if distance from roof-peak to next floor < 8'.

If the second rule is stated incorrectly, allowing the primary form to be created with two stories, then the first rule (which is correct) will allow the secondary form to be added. After all, there are two stories in the primary form, but only one in the secondary form. By correcting the second rule, the two story version will be rejected by the second rule, and the one story version will be rejected by the first one.

An approach at the other end of the spectrum is to generate only valid schema. Here, the idea is to write the generation rules so craftily that invalid schema are never generated. Unfortunately, this approach is impossible to realize for anything but very trivial classes of productions. There are just too many design issues to be considered at every choice point to make this approach workable.

The approach we have taken is an attempt to combine the advantages of both extremes, while minimizing the disadvantages. In the generation of each schema we build-up structure hierarchically (See Figure 7). A complete structure is comprised of substructures, which in turn are comprised of their own substructures. The atomic elements, like roof shape, wall sizes, and room types are combined into intermediate elements like trays, and zones. These substructures have their own sets of rules to insure their validity in-and-of themselves.

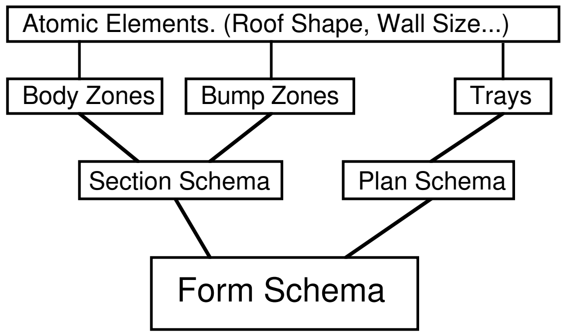

Figure 7 A Hierarchical Approach.

Another technique for controlling combinatorial explosion is based on the idea of generating within equivalence classes. Once substructures have been created they must be combined, and the combined structure checked for validity. For example, a plan and section schema must be combined to create a form scheme. Rather than try to combine all section scheme with all plan schema, such schema are generated in equivalence classes such that any plan in class X can be combined with any section in class Y to result in a form schema that has a high probability of being valid. These equivalence classes are defined in terms of the critical factors that govern the intersection of plan and section, like for example, the width of zones, the number of stories in each zone and the amount of usable space on the top story etc. Any factor which always makes a section-plan pair invalid as a schema is included in the definition of the equivalence classes.

Our current design engine uses two different classes of rules to check the validity of schema: Aesthetic rules, and Pragmatic rules.

Aesthetic rules are those concerned with the architectural aesthetic of the generated forms. These rules check everything from roof shape to plan symmetry. Figure 8 for example, illustrates a section schema in which roof angles change, but not sharply enough to warrant the break in "symmetry". The current grammar contains aesthetic rules to invalidate such schema.

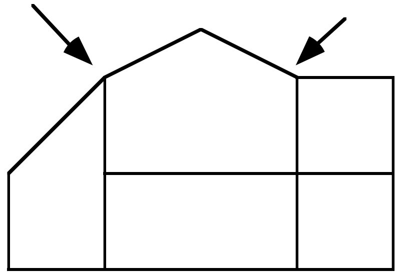

Figure 8 An Example Aesthetic Rules: Weak Roof Angles.

Pragmatic rules are those concerned with the physical realizability and reasonability of the form described by a schema. For example the section schema in Figure 9 is not valid for pragmatic reasons. The primary form has only one story, there is no floor on the same level as the balcony. Thus this section schema is rejected for pragmatic reasons.

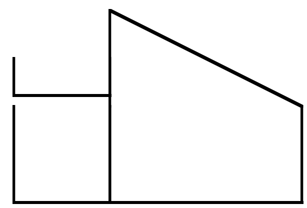

Figure 9 An Example Pragmatic Rule: Inaccessible Spaces

4. Concluding remarks

In this paper we have given a high level overview of the LaHave House Project. The goal of which is to investigate that application of an industrial design approach to architectural design practice.

To date we have developed a prototype design engine and a generative grammar that together produce modern realizable houses. The sophistication of these generated representations is gradually approaching that of detailed preliminary design. We have also developed a prototype design development tool for end-users. This tool allows an end-user to Select, Customize and Visualize house designs.

Although we have come a long way there are many interesting fundamental questions still to be addressed.

Acknowledgments
We would like to thank the TUNS Faculty of Architecture and School of Computer Science for their support and acknowledge the work of the following individuals:

Architects: J. Smirnis, D. Wigle, E. Jannasech, N. Savagen, P. McClelland

Computer Scientists: H. Ning, T. Doucette, X. Hu, G. Li, D. Gemmell, A. Gajewski, D. Curry, D. Peters, G. Burrell, H. Lui, S. Gauvin, W. Wu

References

[1] M. Friedell and S. Kochhar, "Design and Modeling with Schema Grammars", Journal of Visual Languages and Computing, 1991, Vol. 2 , pp. 247-273.

[2] T. Fisher, "Folk Tech", Progressive Architecture, August 1995, pp. 63-72.

[3] H Koning and J Eizenberg, "The Language of the Prairie: Frank Lloyd Wright's Prairie Houses", Environment and Planning B: Planning and Design 8, 1981, pp. 295-323.

[4] B. MacKay-Lyons, "The Village Architect". Design Quarterly 165, MIT Press, Editor R. Jensen, Summer 1996.

[5] W. Mitchell, "The Logic of Architecture". MIT Press, 1990.

[6] C. Moore, G. Allen and D. Lyndon, "The Place of Houses", Holt, Rinehart and Winston publishers, 1974.

[7] I. Petrovic, "On Some Issues of Development of Computer-Aided Architectural Design Systems.", Knowledge-Based Computer-Aided Architectural Design, 1994, Elsvier Science B.V., pp 269-301.

[8] B. Silverman, "Survey of Expert Critiquing Systems: Practical and Theoretical Frontiers", Communications of the ACM, 1992, Vol. 35, No. 4, pp. 107-127.

[9] G. Stiny and W. Mitchell, "The Palladian Grammar", Environment and Planning B: Planning and Design 5, 1978, pp. 5-18.

[10] G. Stiny, "Introduction to Shape Grammars", Environment and Planning B: Planning and Design 7, 1980, pp. 343-351.