A Framework For Generating And Evolving Building Designs

Patrick H.T. Janssen, John H. Frazer and Ming-Xi Tang

IJAC 4-3 Final 19/12/05 1:40 pm Page 450

A Framework For Generating And Evolving Building Designs

Patrick H. T. Janssen, John H. Frazer and Ming-Xi Tang

This paper describes a comprehensive framework for generative evolutionary design. The key problem that is identified is generating alternative designs with an appropriate level of variability. Within the proposed framework, the design process is split into two phases: in the first phase, the design team develops and encodes the essential and identifiable character of the designs to be generated and evolved; in the second phase, the design team uses an evolutionary system to generate and evolve designs that embody this character. This approach allows design variability to be carefully controlled. In order to verify the feasibility of the proposed framework, a generative process capable of generating controlled variability is implemented and demonstrated.

IJAC 4-3 Final 19/12/05 1:40 pm Page 451

1. INTRODUCTION

Evolutionary design systems are loosely based on the neo-Darwinian model of evolution through natural selection. A population of individuals is maintained and an iterative process applies a number of evolution steps that create, transform, and delete individuals in the population. Each individual has a genotype representation and a phenotype representation. The genotype representation encodes information that can be used to create a model of the design, while the phenotype representation is the actual design model. The individuals in the population are evaluated relative to one another, and on the basis of these evaluations, new individuals are created using 'genetic operators' such as crossover and mutation. The process is continued through a number of generations so as to ensure that the population as a whole evolves and adapts.

Two types of evolutionary design may be broadly identified: parametric evolutionary design and generative evolutionary design. Parametric evolutionary design is the more common approach. A design is predefined and parts that require improvement are parameterised. The evolutionary system is then used to evolve these parameters. Examples of parametric evolutionary design systems include [1-4].

The generative approach, although more complex, can also be much more powerful. This approach may be used early on in the design process and focuses on the discovery of inspiring or challenging design alternatives for ill-defined design tasks. A generative process is created that uses information in the genotype to generate alternative design models. This process consists of a rule-based growth procedure that is capable of generating design alternatives that vary significantly from one another. (In section 4.1, the most commonly used techniques will be briefly discussed. These are parametric, combinatorial, substitutional, agent-based, mathematical, and spatial partitioning techniques.) The evolutionary system will tend to evolve a divergent set of alternative designs, with convergence on a single design often being undesirable or even impossible. Such systems are sometimes described as divergent systems or exploration systems. Examples of generative evolutionary design systems include [5-13].

This paper describes a framework that supports a generative evolutionary design approach that would allow a design team to evolve the overall configuration and organisation of buildings. The aim behind creating such systems is not to duplicate or mimic an existing conventional design process. Rather, the aim is to create an alternative design approach that allows designers to work in ways that were previously not possible [5]. In particular, one of the key goals is to develop a system that can be used to design low-energy buildings.

1.1. The variability problem

For generative evolutionary design, a distinction can be made between systems whose main purpose is to inspire, versus systems whose main purpose is to challenge. Many of the existing systems evolve forms that may be inspiring to designers. These types of systems may produce highly abstract forms that trigger new possibilities in the minds of the designers. However, if the forms do not incorporate a relatively explicit and comprehensive description of the design, then the designers will be required to interpret and read meaning into the forms.

In order to evolve challenging designs, the designer cannot be relied upon to interpret the forms. Instead, explicit designs rather than abstract forms must be evolved. In order to achieve this the variability of the designs must be carefully controlled so as to ensure that the designs are complex, intelligible, unpredictable and desirable:

- Complex: The level of complexity within the designs must be commensurate with the complexity of the entities being designed. Designs must therefore consist of three-dimensional models consisting of a realistic number and variety of related elements.
- Intelligible: The forms must be directly intelligible as designs by both people and by other software systems. Important characteristics of the forms must therefore be explicitly represented, thereby allowing for unambiguous understanding.
- Unpredictable: The forms must vary from one another in significant ways. This must include variation in the organisation and configuration of the elements that comprise the form.
- Desirable: The forms must embody certain qualities that are seen to be desirable by the designers using the system. These qualities may be either qualitative or quantitative.

These criteria depend primarily on how the generative process is implemented. This process creates a design by applying a set of generative rules to some starting condition. The same set of rules will be used to generate each design. The genotypes then encode variations in the starting condition and in how the rules are applied. However, given a certain level of complexity, it can be difficult to create a process that generates designs that are both intelligible and unpredictable: unpredictability results in forms that are chaotic and therefore unintelligible; intelligibility results in designs that are all very similar and therefore predictable. This conflict is referred to as the variability problem.

1.2. Families of designs

In order to overcome the variability problem, generative rules and representations are required that ensure that the variability of designs that are produced is neither over-restricted nor under-restricted. This is referred to as controlled variability. Developing such rules and representations requires the identification of a set of characteristics common to all the designs to be generated. Rules and representations can then be created based on these shared characteristics.

In the domain of architecture, it is difficult to identify any significant shared characteristics. Buildings simply vary too much for this to be possible. As a result, some sub-set of designs needs to be considered that includes designs that are similar in some way, but that nevertheless vary significantly in overall organisation and configuration. Since the included designs will share certain characteristics, they may be described as a family of designs. The question then becomes, on what basis should such a family of designs be defined and demarcated?

One possible approach is to focus on conventional designs. With this approach, characteristics common to a large number of conventional buildings may be used to define a generic family of designs. However, this generic approach is problematic since it fundamentally limits the creativity of the designer. The aim of the generative evolutionary design approach is to enhance the creative process by allowing designers to explore populations of alternative designs. If the designs being generated are required to be conventional, then the creative process will be hindered rather than enhanced.

An alternative approach is to develop an evolutionary system that is highly customisable and that allows a design team to define their own families of designs. In such a case, the focus would naturally tend to gravitate towards the design team's own body of work or oeuvre, which will typically incorporate sequences of designs that share a similar design character. This character can then become the basis for defining families of designs that are specific to that design team. By customising the system in an appropriate way, the design team could then generate and evolve design alternatives that incorporate their personal and idiosyncratic character.

1.3. Proposed framework

A generative evolutionary design framework has been developed that consists of a design method and an evolutionary system [14]. The design method broadly defines a set of tasks to be carried out by the design team. The evolutionary system is a software system used by the designer team for generating and evolving alternative designs.

The core concept within this framework is the notion of a design entity that captures the essential and identifiable character of a varied family of designs by one designer or design team. This conceptualisation is defined as a design schema. It encompasses those characteristics common to all members of the family, possibly including issues of aesthetics, space, structure, materials, and construction. Although members of the family of designs share these characteristics, they may differ considerably from one another in overall organisation and configuration.

When a design schema is codified in a form that can be used by an evolutionary system, it provides a way of overcoming the variability problem. The encoded schema allows complex designs to be generated that are both intelligible and unpredictable. This approach is based on the work of [5, 13, 15].

Section 2 will discuss the design method, and section 3 will describe the evolutionary system. Section 4 gives an overview of a demonstration of key components within the framework. Finally, section 5 draws some conclusions.

2. THE DESIGN METHOD

In the 1960's, the design process was typically idealised as being highly rational and objective, with the three core stages being analysis, synthesis, and evaluation [16]. Today, such rational and objective design models are no longer seen to be realistic. Instead, the design process is understood to be a highly complex process involving multiple participants exchanging information and knowledge that tends to be incomplete, inconsistent, and often incompatible.

A number of researchers have argued that, in practice, designers bring a set of preconceptions into a project, and that these preconceptions play a significant role in the design process. Two types of preconception may be identified:

- The design stance has been referred to as the designer's 'paradigmatic stance' [17], 'guiding principles' [18], and 'theoretical position' [19]. This stance consists of a designer's broad philosophical beliefs and cultural values, which emerge over an extended period of time through multiple projects. The design stance often evolves and changes throughout a designer's career. However, a designer will generally adhere to a single design stance that is applicable to any projects that they work on.
- Design ideas have been described as 'primary generators' [20], 'enabling prejudices' [19], and 'working methods' [21]. The main characteristic of such design ideas is that they are not derived from a process of reasoning or analysis, but are self-imposed subjective judgements that define and direct the design process.

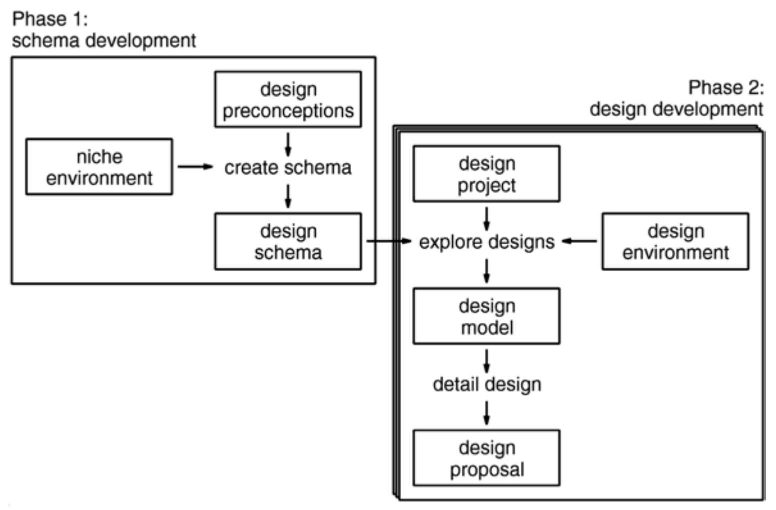
The relationship between a set of design ideas and a design project may take a number of different forms. Lawson's analysis of the design process suggest that there are three possibilities [18]. First, the design ideas may be directly related to some specific aspect of a project. Second, the design ideas may be related to the type of project, but not specifically to the one project. Third, the design ideas may not arise from the project but may originate from the designer's own personal design stance. Design ideas that are not project-specific may be re-used within a variety of projects. Through previous projects or competitions, designers implicitly develop interrelated sets of design ideas that may be applied to a range of new projects.

2.1. An existing design method

An existing design process can be defined that incorporates both design preconceptions and design schemas. This process, although not necessarily universal, is seen as one that many designers loosely follow. This design process is shown in Figure 1. Although the direction of the flow should be understood to be predominantly top to bottom, most designers do not work in a linear manner, but move backwards and forwards between stages. The design process consists of a schema development phase and a design development phase.

In the first phase, a design schema is developed that consists of an interrelated set of design ideas applicable to certain types of projects. Designers will usually develop such ideas by working on a series of similar projects or competitions, and this stage may unfold over an extended period of time.

In the second phase, an initial design model is developed by adapting the design schema to the constraints and context for a specific project. Designers typically explore and evaluate a range of alternative designs at this stage. This stage involves analysing the brief and the site, and applying the design ideas in an appropriate manner. The design model is then further developed into a detailed design.



▶ Figure 1. An existing design method that many designers follow.

This design method encompasses two types of environment: the design environment and the niche environment. The design environment includes the project specific constraints and context, while the niche environment is not one specific environment, but instead encompasses a range of design environments that are similar to one another. The schema developed in the first phase is not specific to one design environment. Instead, the design team develops it with a certain type of environment in mind, encompassing a range of possible constraints and a range of possible contexts. The schema can be used in any project whose design environment falls in the niche environment for which the schema was developed.

2.2. Proposed design method

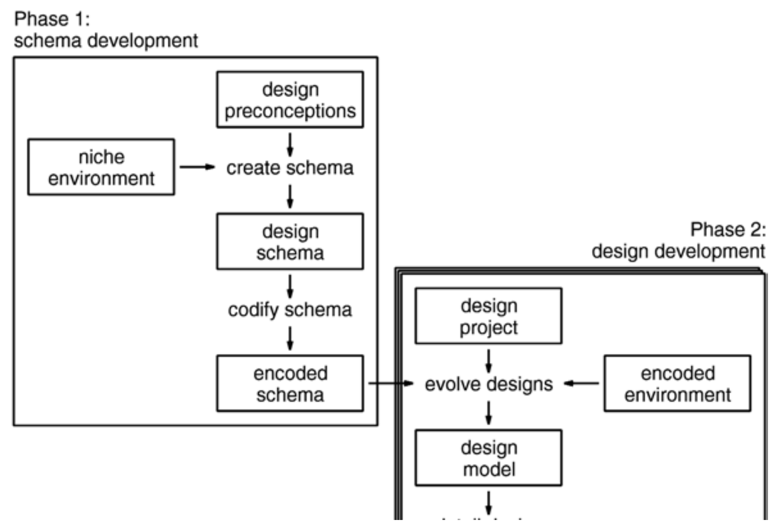
The proposed generative evolutionary design method is a modification of the existing design method described above. Figure 2 shows the overall structure of the proposed method. As with the previous method, design process is broken down into a schema development phase and the design development phase. In the first phase, an additional stage has been added during which the schema is encoded as a set of evolutionary rules and representations. In the second phase, the stage of design exploration has been replaced by a stage of design evolution using the evolutionary system. The two stages of the schema development phase are as follows:

- The schema creation stage is similar to the existing design process and requires the design team to conceptually develop a design schema for a particular niche environment. The main guiding forces at this stage are the niche environment and the preconceptions of the design team. The design team must then define the design schema is some relatively explicit way. The design team may, for example, create a number of prototypical designs that capture the variability that they desire. At this stage, the design team works at a purely conceptual level and as a result, no specialised programming or computational skills are required.
- The schema encoding stage involves encoding the design schema in a form that can be used by the evolutionary system. This involves defining a set of evolutionary rules and representations for the evolutionary system. The requirements from the design team are now different, and specialised programming and computational skills are essential. The evolutionary process must be well understood and, in particular, the design team must be aware of the relationships and interactions between the various rules and representations.

For the design development phase, the two stages are as follows:

- The design evolution stage allows a large variety of alternative designs to be evolved and adapted to a specific design environment. At this stage, the design team will need to encode the design environment for the specific project, and will then need to configure and run the evolutionary system. The process of encoding the design environment is generally much more straightforward than encoding the design schema. The evolutionary process will result in a set of alternative designs. The design team must choose one of these for further detailed design.
- The detailed design stage is identical to the existing design method and requires the design team to further develop the design model

selected in the previous stage to the level of detail required for construction. As with the first stage, no specialised programming or computational skills are required.



► Figure 2: The schema-based generative evolutionary design method.

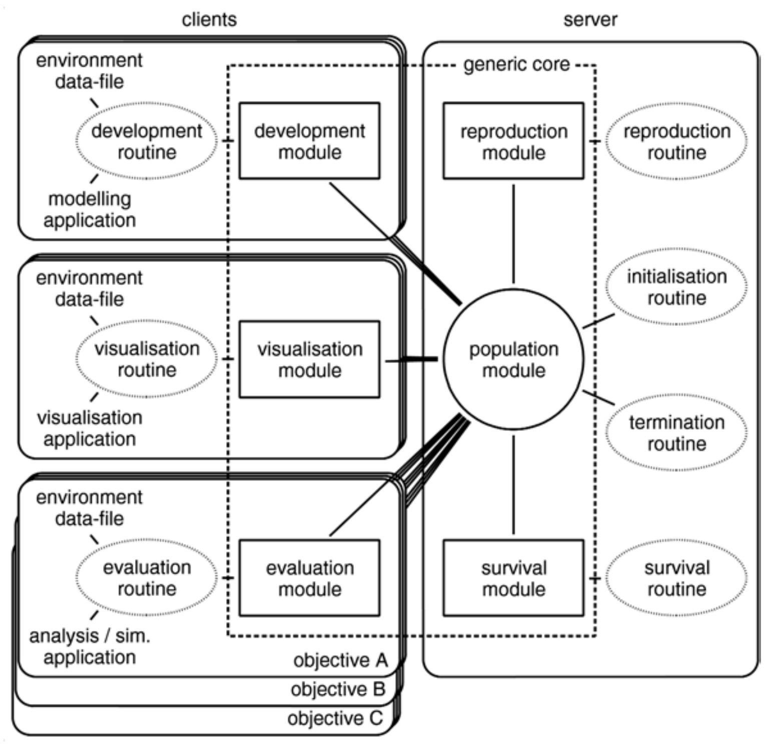
Compared to the existing design method shown in Figure 1., two key changes have been made. First, the schema development phase has an additional stage, during which the design schema is encoded as a set of evolution routines. Second, during the design development phase, manual design exploration has been replaced with automated design evolution.

3. THE EVOLUTIONARY SYSTEM

The design evolution stage requires a software system that allows the design team to generate and evolve alternative designs. This system must fulfil two key requirements:

- The system should be customisable, allowing for the modification and replacement of the rules and representations used by the system. This will allow each design team to incorporate their encoded schema and encoded environment.
- The system should be scalable, allowing for the evolution of large complex designs without performance being adversely affected. In most cases, the developmental and evaluation steps are likely to be the most computationally demanding.

A computational architecture for an evolutionary system has been developed that fulfils these two requirements. This architecture is shown in Figure 3. The evolutionary system maintains a single population that is manipulated by four evolution steps: reproduction, development, evaluation, and survival. In addition there is also a visualisation step that allows users to select and visualise designs in the population, and initialisation and termination steps for starting and stopping the evolutionary process.



◀ Figure 3: Schema-based generative evolutionary design system using a standard networked computing environment. The system consists of a generic core, and a set of specialised components that need to be plugged into the generic core. Each design team needs to develop specialised components that encode their design schema.

A parallel implementation is employed using a standard client-server model in a networked computing environment. The server manages the population of designs and performs the reproduction and survival steps, while multiple client computers perform the developmental and evaluation steps. The visualisation step is also performed on client computers. For a detail description of this architecture, see [14].

3.1. Customisability

In order to support customisability, the evolutionary system is broken down into two parts: a generic core and a set of specialised components. The generic core defines the main structure of the evolutionary system and can be used unmodified by any design team, on any project. This core consists of underlying program modules that communicate and interact with one another. However, in order to be functional, these modules must be linked to the specialised components.

The specialised components are completely customisable by the design team. Three types of specialised components can be defined: routines, data-files, and applications.

- Routines encapsulate the rules and representations used by the evolution steps. The design team must create a set of such routines that together constitute the encoded schema.
- Data-files encapsulate information about the design environment,

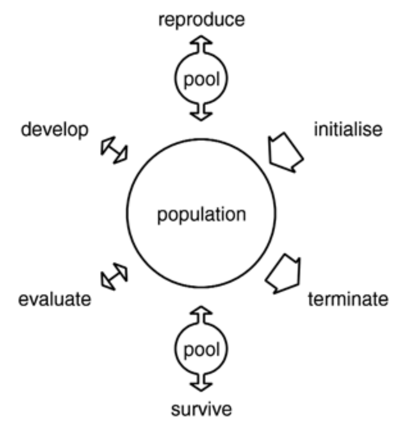
encompassing both design constraints and design context. Examples of design constraints may include the budget, the number of spaces, floor areas, performance targets and so forth. The design context may include site dimensions, site orientation, neighbouring structure, seasonal weather variations, and so forth.

- Applications are existing software applications whose functionality the design team may require, in particular for modelling, visualising and evaluating design models.

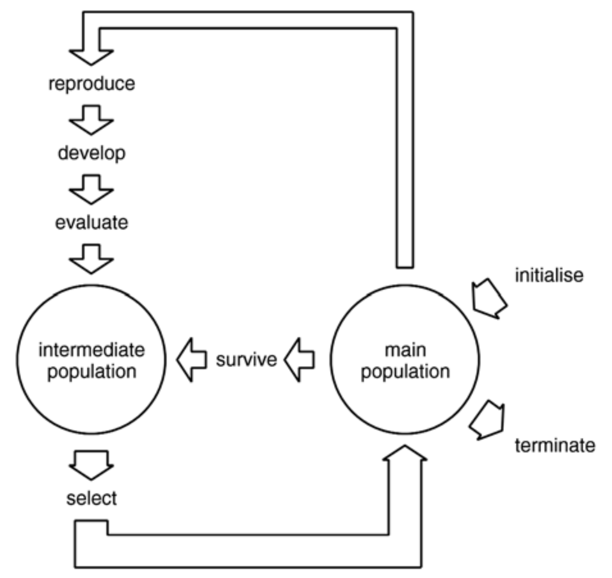
3.2. Scalability

In order to support scalability, the evolutionary system employs an evolutionary process that can be described as using a decentralised control structure and an asynchronous evolution mode. This architecture is shown in Figure 4.

- A decentralised control structure is use, whereby the four evolution steps act independently from one another. Each step extracts a small number of individuals from the population, processes these individuals, and either inserts the resulting individuals back into the population or - in the case of the survival step - deletes a number of individuals in the population.
- An asynchronous evolution mode is used, whereby the evolution steps process individuals or small groups in the population as soon as they become available. Individuals in the population will therefore be in various different states, depending on whether they have been developed or evaluated.



▶ Figure 4: The general architecture of the proposed evolutionary system using a decentralised control structure together with an asynchronous evolution mode. The evolution steps are autonomous and operate independently from one another. The development and evaluation steps act on individuals in the population, while reproduction and survival steps require small pools of individuals.



◀ Figure 5: The general architecture of most evolutionary systems, using a centralised control structure together with a synchronised evolution mode. The evolution steps are applied to the population in sequence, one after another.

This type of evolutionary process differs markedly from the process employed by most existing evolutionary system. A wide variety of evolutionary algorithms exist, with the four main types being genetic algorithms [22], evolution strategies [23], evolutionary programming [24], and genetic programming [25]. Such algorithms differ in the rules and representations that they use in order to implement the evolution steps, but the overall evolutionary process is similar for all of them. The general architecture of such algorithms is shown in Figure 5, and uses a centralised control structure and a synchronised evolution mode. The centralised control structure consists of a main loop that repeatedly invokes and executes the evolution steps. The synchronous evolution mode consists of a procedure that stops and waits for the processing of all individuals by one evolution step to be completed before proceeding onto the next evolution step.

For scalability, the decentralised asynchronous approach has various advantages over the centralised synchronous approach. The decentralised control structure allows client computers to be easily be added and removed from the evolutionary process and allows the system to cope gracefully with failure of one or more client systems [14]. The asynchronous evolution mode reduces the execution time and is highly effective in situations where the development and evaluation steps are costly [14, 26].

3.3. Evolution Steps

The population module manages the population on the central server, which consists of individuals are in various different states: some only have a genotype, others have a phenotype, while others also have evaluation scores.

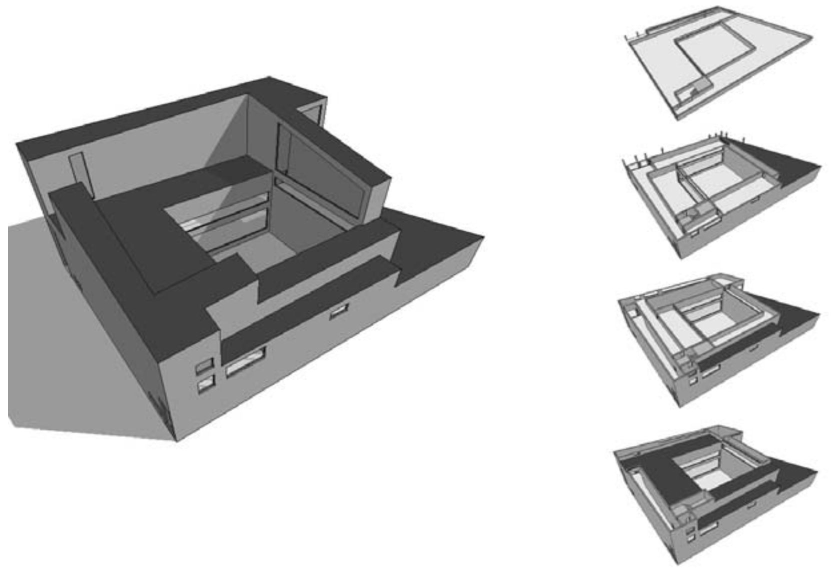
The population module does not control the evolution modules, but instead passively waits to be contacted by these modules. The evolution modules repeatedly perform the following actions:

- The reproduction module requests a random pool of fully evaluated individuals. This module then produces one or more new individuals from these parent individuals by executing the reproduction routine.
- Each developmental module requests a single undeveloped individual. This module then creates a phenotype for the individual by executing the developmental routine. The developmental routine may make use of existing CAD modelling applications or software libraries.
- Each evaluation module requests one individual that has not been evaluated for the objective in question. This module then creates an evaluation score for the individual by executing the evaluation routine. In order to perform the evaluations, the evaluation routine will invoke existing simulation and analysis applications. (Possible aspects that might be analysed and simulated include energy consumption, daylight levels, views out, construction cost, and so forth). If multiple objectives are being evaluated, at least one evaluation module per objective must be defined.
- The survival module requests a random pool of fully evaluated individuals. This module then identifies one or more individuals to be deleted by executing the survival routine.

The routines executed by the reproduction and survival modules can apply selection pressure by favouring individuals with high evaluation scores. If multiple objectives are being evaluated, some form of scalarisation will need to take place in order to rate individuals relative to one another. Various scalarisation techniques exist, including calculating the weighted average of the performance scores, or used Pareto-optimal ranking methods.

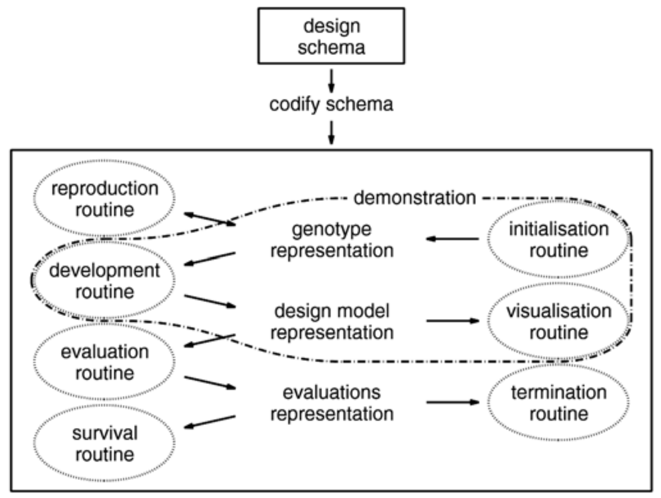
4. DEMONSTRATION

The evolutionary system is currently under development. One of the critical aspects of this system is the ability of the design team to develop a generative process that is capable of generating designs that vary in a controlled manner, referred to as controlled variability. The process of encoding a design schema has therefore been demonstrated. (Since the evolutionary system is still under development, the demonstration does not include the process of evolving designs.) Figure 6 shows an example of one of the designs within the schema. This schema should be thought of as being specific to one design team, and will therefore incorporate various design preconceptions.



◀ Figure 6: One member of a family of designs.

Figure 7 shows the routines and representations that constitute the encoded schema. In order to verify the variability that could be achieved, a developmental routine, an initialisation routine and a visualisation routine have been implemented. These routines have been used to generate and visualise a variety of design models. Figure 10 shows a selection of generated models. The designs that are generated are complex, intelligible, and unpredictable. Controlled variability has therefore been achieved.



◀ Figure 7: The process of encoding the design schema. (See the 'codify schema' stage in Figure 2.) In order to encode the schema, the design team must define seven routines and choose three representations. For the demonstration, three routines and two representations were implemented; these are shown within the dotted line.

4.1. Some generative techniques

In order to achieve controlled variability, the generative process will need to be a hybrid process that combines a number of generative techniques. In this way certain parts of the process can be highly restricted, while in other parts variability can be permitted to emerge. Some of these techniques may be developed specifically for the schema, while others may be existing techniques. Six commonly used existing techniques are identified:

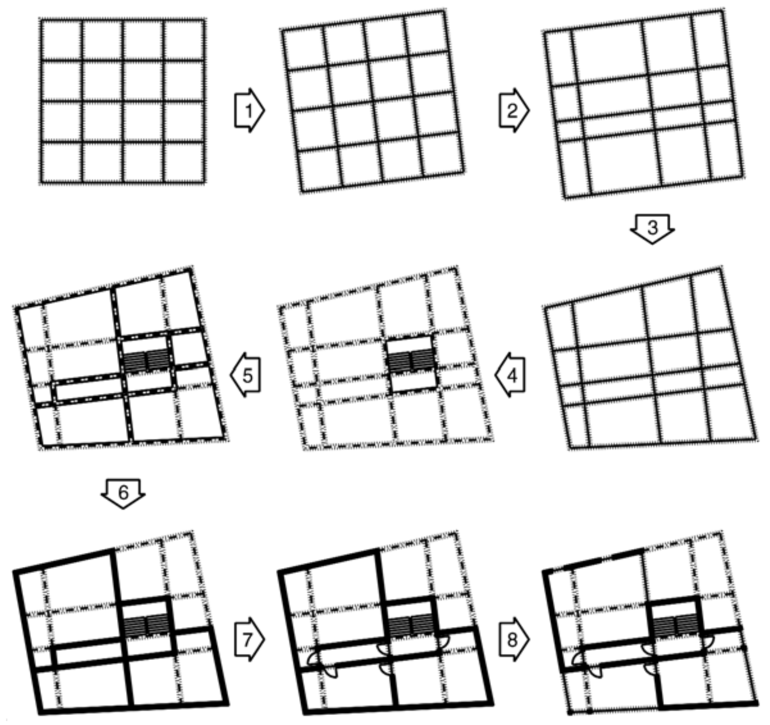
- Parametric techniques generate forms by varying a number of parameters associated with the form. The parameters may either be associated with a model of the form, or they may be associated with a sequence of operations that create the form. This latter approach is used by a number of parametric CAD systems such as CATIA [27] and Autodesk Inventor [28]. Various types of parametric modelling approaches are discussed in [29].
- Combinatorial techniques generate forms by combining predefined set of elements. Algebra-based techniques predefine a set of element types together with a set of operations for manipulating these elements. Template-based techniques define an organisational template into which elements can be inserted. These techniques are discussed in [30].
- Substitution techniques generate forms by defining an initial starting form and then repeatedly substituting parts of this form with new parts. The substitution procedure may be based either on the shape of the parts, or on particular patterns within a grid. Examples of shape-based substitution techniques include Fractals, Shape Grammars and L-systems. An example of a grid-based substitution system is the Cellular Automata. These techniques are discussed in [15, 31, 32]
- Agent-based techniques generate forms by defining virtual agents that move, interact and collaborate. This approach is often inspired by insect colonies such as those built by ants and bees. These kinds of programs are also closely related to research in Artificial Life systems. Agent-based techniques are discussed by [15, 31].
- Mathematical techniques generate forms by defining three-dimensional scalar fields and then selecting isosurfaces within these fields. Such forms are also referred to as implicit surfaces. The fields are created by defining a function that allows a value to be associated with every point in space, and the isosurface then connects points of equal value. The function is often defined relative to some underlying primitive objects, such as points, lines and planes. Mathematical software such as Mathematica, Maple and MATLAB [33-35] can be used to experiment with these types of techniques. These techniques are discussed by [36, 37]
- Spatial partitioning techniques generate forms by subdividing space

and assigning properties to the spatial units that result. A variety of different types of techniques exist. Voxel partitioning divides space into a set of regular shaped voxels, such as cubes. Hierarchical techniques subdivide space into volumes that can themselves be further subdivided. Examples of such techniques include Octrees and BSP trees. Voronoi decompositions subdivide space into volumes determined by distances to a specified discrete set of objects in the space. These techniques are discussed in [38, 39].

These techniques are not mutually exclusive and in many cases, the same form can be generated by a number of alternative techniques. They may be thought of as attractors for certain types of forms, in that they allow some forms to be more easily generated than others.

4.2. An example of a generative process

A hybrid generative process for the example schema has been developed, where a variety of generative techniques are sequentially applied to a grid generated using a simple spatial partitioning technique. The process consists of a sequence of eight steps that gradually change an orthogonal grid into a three-dimensional building model. Figure 8 shows (diagrammatically in two-dimensions) the eight generative transformations. The grid consists of as a set of intersecting faces that subdivide the space into an array of cells.



◀ Figure 8: The generative process consisting of eight steps.

Initially, the faces are orthogonal to one another and regularly spaced, creating an array of cubic cells.

Most transformations require a set of parameters encoded within the genotype. The genotype consists of a fixed length list of real numbers in the range 0.0 to 1.0. Each number is referred to as a gene. The genes may be mapped to a value within a different continuous or discrete range as required. Some transformations may also require certain parameters or data encoded in the environment data-file.

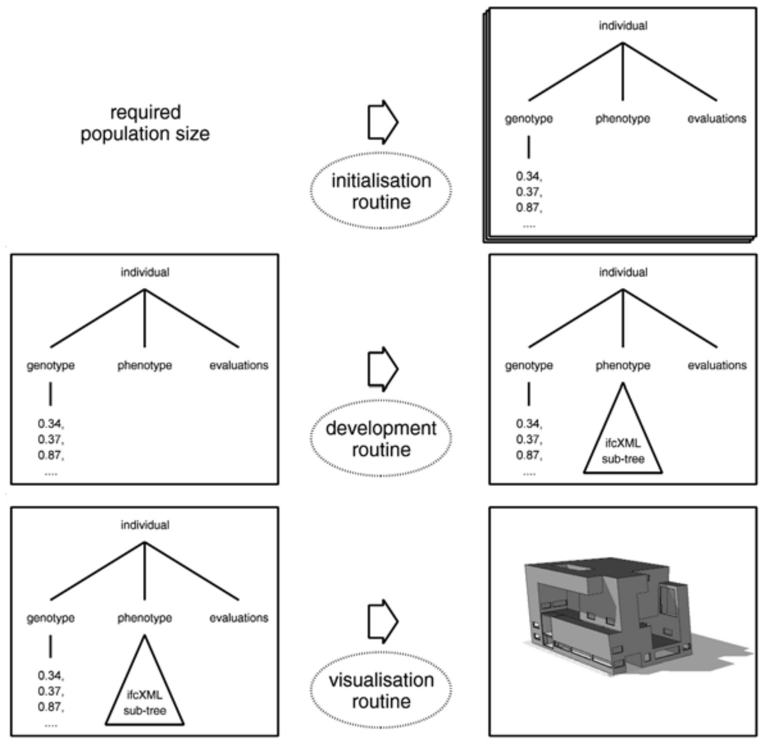
1. The grid is positioned within the site boundaries using parametric techniques. The grid is first placed in the centre of the site. Genes then encode a rotation followed by a translation relative to the site boundaries.
2. The grid is stretched and compressed in orthogonal directions using parametric techniques. Genes encode the amount by which the grid is stretched or compressed.
3. Outer grid-faces are inclined using parametric techniques. Each face is inclined by rotation around two axes. Genes encode the size of the rotations relative to a minimum and maximum.
4. The staircase is created and inserted into the building. The location resulting in the smallest possible stairwell is selected. (No genes are involved.) parametric techniques are used to generate the staircase.
5. The spaces within the building are defined using new type of substitution technique. Genes encode 'pressure values' within each space. The dividing surface between adjacent spaces with large differences in pressure can 'burst', thereby merging two spaces into a single space.
6. The spaces are assigned as being interior or exterior spaces using a technique similar to the previous step. The pressure values for the spaces are also used. Spaces that have the highest pressure differential relative to the space outside can 'burst', thereby becoming exterior spaces.
7. Doors are inserted to allow access between spaces using a hybrid of combinatorial and parametric techniques. Doors are inserted based on a set of deterministic rules. (No genes are involved.)
8. Windows are inserted in the exterior walls using a hybrid of combinatorial and parametric techniques. Genes encode a set of window types for each space. The four possible types are no window, fully glazed window, rectangular window, and horizontal strip window.

When executing these eight steps, the generative process must verify than certain hard constraints are not violated. For example, the constraints specify that all spaces must be higher than a certain minimum height, must have a horizontal floor level, and must be accessible from the staircase. In addition, a soft constraint specifies that the preferred spaces are rectangular shaped, followed by L-shaped spaces, followed by all other spaces (such as T-shaped spaces).

4.3. Implementation of routines

The initialisation routine was used to generate a population of genotypes, the developmental routine was then used to generate a population of design models, and finally the visualisation routine was use to view these models.

An individual is represented as an XML tree, with three main nodes: 'genotype', 'phenotype' and 'evaluations'. The genotype node contains a set of real numbers that are the genes for the generative process. If the individual has been developed, then the phenotype contains the representation of the model, encoded using an XML representation. Finally, the evaluations node will contain one or more evaluation score nodes, each of which encodes the result of analysing one objective. Since the evaluation routines were not implemented in this demonstration, this part was not used.

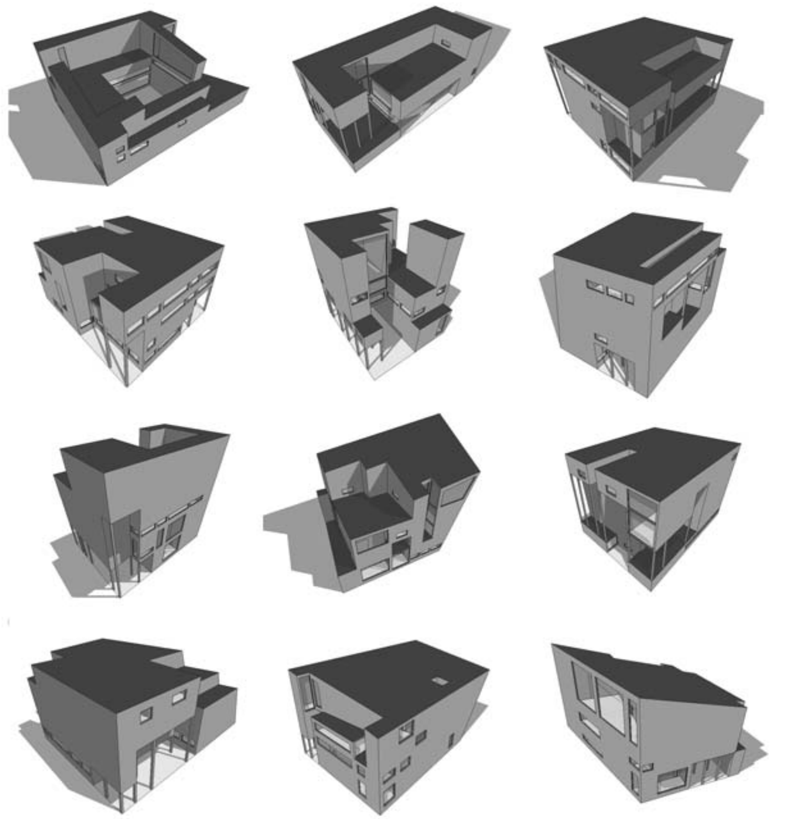


◀ Figure 9: The inputs and outputs for the three implemented routines.

The three implemented routines, shown diagrammatically in Figure 9, are described as follows:

- The initialisation routine is used to generate a population of individuals with randomly generated genotypes, but with no phenotypes or evaluation scores. This routine calculates the length of the required genotype, and creates a random value for each parameter. The input to this routine is the required number of individuals.

- The developmental routine is used to create phenotypes for each individual. The generative process used by this routine has been described above. The input tree is an individual with an empty phenotype node, while the output individual is the same individual but with the IFC [40] representation of the design model added to the phenotype node. The developmental routine also requires various parameters and data stored in the environment data-file.
- A visualisation routine has been created that uses an existing software called Ecotect [41] to visualise the design models that are generated. This routine extracts the phenotype from each individual, translates the IFC representation to the model representation used by Ecotect, and then allows the user to visualise the design using the Ecotect interface. Figure 10 shows a set of generated models visualised in Ecotect.


▶ Figure 10: A set of generated (but not evolved) designs.

5. CONCLUSIONS

The demonstration has shown that it is possible to create a generative process that generates complex three-dimensional models of building designs that are both intelligible and unpredictable. Controlled variability has therefore been achieved.

Since the designs have not yet been evolved, they have not yet become adapted to any objectives or environment. The next stage of the research will focus on developing the complete evolutionary system. This will allow designs to evolve and adapt in response to the environment and the evaluation criteria.

References

1. Dasgupta, D. and Michalewicz, Z., eds, *Evolutionary Algorithms In Engineering Applications*, Springer-Verlag, 1997.
2. Rasheed, K. M., *GADO: A Genetic Algorithm for Continuous Design Optimization*, PhD Thesis, Department of Computer Science, Rutgers University, New Brunswick, NJ. Technical Report DCS-TR-352, 1998.
3. Monks, M., Oh, B. M. and Dorsey, J., Audioptimization: Goal-Based Acoustic Design, *IEEE Computer Graphics and Applications,* May/June 2000, 20(3), 76-91.
4. Caldas, L., An Evolution-Based Generative Design System: Using Adaptation to Shape Architectural Form, PhD Thesis, Massachusetts Institute of Technology, 2001.
5. Frazer, J. H. and Connor, J., A conceptual seeding technique for architectural design, in: *Proceedings of International Conference on the Application of Computers in Architectural Design and Urban Planning (PArC79)*, Berlin, AMK, 1979, 425-434.
6. Graham, P. C., Frazer, J. H., and Hull, M. C., The application of genetic algorithms to design problems with ill-defined or conflicting criteria, in: Glanville, R. and de Zeeuw, G., eds., *Proceedings of Conference on Values and, (In) Variants*, 1993, 61-75.
7. Frazer, J. H., The Interactivator, *AA Files,* 1995, 72-73.
8. Bentley, P. J., *Generic Evolutionary Design of Solid Objects using a Genetic Algorithm*, PhD Thesis, Division of Computing and Control Systems, Department of Engineering, University of Huddersfield, 1996.
9. Rosenman, M. A., An exploration into evolutionary models for non-routine design, in: *AID'96 Workshop on Evolutionary Systems in Design*, 1996, 33-38.
10. Shea, K., *Essays of Discrete Structures: Purposeful Design of Grammatical Structures by Directed Stochastic Search*, PhD Thesis, Carnegie Mellon University, Pittsburgh, PA, 1997.
11. Coates, P., Broughton, T., and Jackson, H., Exploring three-dimensional design worlds using Lindenmayer Systems and Genetic Programming, in: Bentley, P. J., ed., *Evolutionary Design by Computers*, Morgan Kaufmann Publishers, San Francisco, CA., 1999, 323-341.
12. Funes, P. and Pollack, J., Computer evolution of buildable objects, in: Bentley, P. J., ed., *Evolutionary Design by Computers*, Morgan Kaufmann Publishers, San Francisco, CA., 1999, 387-403.
13. Sun, J., *Application of Genetic Algorithms to Generative Product Design Support Systems*, PhD Thesis, School of Design, Hong Kong Polytechnic University, 2001.
14. Janssen, P. H. T., *A design method and a computational architecture for generating and evolving building designs*, PhD Thesis, School of Design, Hong Kong Polytechnic University, 2004.
15. Frazer, J. H., *An Evolutionary Architecture*, AA Publications, London, UK, 1995.
16. Cross, N., A history of design methodology, in de Vries, M. J., Cross, N., and Grant, D. P., eds., *Design Methodology and Relationships with Science,* Dordrecht, Netherlands: Kluwer Academic Publishers, 1993, 15-27.
17. Broadbent, G., *Design in Architecture: Architecture and the Human Sciences*, David

Fulton Publishers, London, UK, 1988.

18. Lawson, B., *How Designers Think: The Design Process Demystified*, Architectural Press, Oxford, UK, 3rd edition, 1997.
19. Rowe, P. G., *Design Thinking*, MIT Press, Cambridge, MA, 1987.
20. Drake, J., The primary generator and the design process, *Design Studies*, 1979, 1(1), 36-44.
21. Frazer, J. H., Creative design and the generative evolutionary paradigm, in: Bentley, P. and Corne, D., eds., *Creative Evolutionary Systems*, Morgan Kaufmann Publishers, UK, 2002, 253-274.
22. Holland, J. H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.
23. Rechenberg, I., *Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien der Biologischen Evolution*, Frommann-Holzboog Verlag, Stuttgart, Germany, 1973.
24. Fogel, D. B., *Evolutionary Computation: Towards a new philosophy of machine intelligence*, IEEE Press, 1995.
25. Koza, J. R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA, 1992.
26. Rasheed, K. M. and Davidson, B. D., Effect of global parallelism on the behaviour of a steady state genetic algorithm for design optimization, in: *Proceedings of the Congress on Evolutionary Computation (CEC'99)*, Volume 1, IEEE Press, 1999, 534-541.
27. Dassault Systems: *http://www.3ds.com* [1-9-2005].
28. Autodesk: *http:///www.autodesk.com* [1-9-2005].
29. Monedero, J., Parametric design: a review and some experiences, *Automation in Construction*, 2000, 9(4), 369-377.
30. Mitchell, W., *The Logic of Architecture: Design, Computation and Cognition*, MIT Press, Cambridge, MA, 1990.
31. Flake, G. W., *The Computational Beauty of Nature: Computer Explorations of Fractals, Chaos, Complex Systems, and Adaptation*, MIT Press, Cambridge, MA, 1998.
32. Stiny, G., Introduction to Shape and Shape Grammars, *Environment and Planning B*, 1980, 7, 343-351.
33. Wolfram research: *http://www.wolfram.com/* [1-9-2005].
34. Maplesoft: *http://www.maplesoft.com/* [1-9-2005].
35. Mathworks: *http://www.mathworks.com/* [1-9-2005].
36. Wyvill, B., and Wyvill, G., Field Functions for Implicit Surfaces, *The Visual Computer*, December 1989, 5, 75-82.
37. Bloomenthal, J., ed., *Introduction to Implicit Surfaces*, Morgan Kaufmann Publishers, August 1997.
38. O'Rourke, J., *Computational Geometry in C*, Cambridge University Press, 2nd edition, 1998.
39. Kim, D.-S., Cho, Y., and Kim, D., Euclidean Voronoi diagram of 3D balls and its computation via tracing edges, *Computer-Aided Design*, 2005, 37(13), 1412-1424.
40. International Alliance for Interoperability: *http://www.iai-international.org/* [1-9-2005].
41. Square One: *http://www.squ1.com/* [1-9-2005].

Patrick H.T. Janssen

Hong Kong Polytechnic University
School of Design

patrick@janssen.name

John H. Frazer

Gehry Technologies
Digital Practice Ecosystem

john.frazer@gehrytechnologies.com

Ming-Xi Tang

Hong Kong Polytechnic University
School of Design

sdtang@polyu.edu.hk