Evolution Aided Architectural Design
An Internet based evolutionary design system

Patrick JANSSEN [*]**, John FRAZER, Ming-Xi TANG**

Abstract: This paper describes a framework for evolutionary design system that attempts to overcome two key limitations with existing systems. First, the proposed system explicitly separates the core evolutionary process from the design specific rules and data structures. This allows the rules and data structures to be manipulated so as to reflect the idiosyncratic design approach of the design team. The second limitation to be overcome centres on the integration of existing simulation and analysis programs within the proposed system. These programs tend to be computationally very expensive, and may run on different operating systems. The option of creating a monolithic system is therefore not viable; instead a networked approach is taken.

Keywords: evolution, generative design, Java, servlets, XML, XSLT

1 Introduction

Evolution can be thought of as consisting of two fundamental processes: creation and survival. In nature, the creation process involves creating new organisms through reproduction and development, and the survival process involves the differential survival of organisms competing in an environment with limited resources.

Large populations of organisms individually reproduce to generate new offspring. Each offspring develops from an encoded set of triggers, known as the *genotype*, into a fully developed organism, known as the *phenotype*. The offspring will inherit features from their parents, but may also have some entirely new features. Organisms compete for resources in the environment and those that are best adapted to the environment will have the highest chances of surviving and reproducing.

Universal Darwinism [DAW 83] suggests that the process of evolution can emerge regardless of the medium, be it biological, computational, cognitive, or some other medium. *Evolution Aided Design* (EAD) is a design approach that uses evolutionary software in order to aid the design process. Numerous EAD systems have been developed in a variety of design fields [FRA 95; BEN 99; BC 02]. Such systems can also be conceptualised as two processes: a design *creation* process and a design *survival* process. The creation process generates populations of design alternatives. The survival process edits this population by allowing the most suitable designs to survive. These designs are then used by the creation process as a basis for generating new designs. Gradually, the population as a whole evolves and adapts.

[*] School of Design, Hong Kong Polytechnic University, China. patrick@janssen.name

Evolution Aided Architectural Design (EAAD) is a design approach that focus specifically on designing buildings using evolutionary software. With the EAAD approach, the design team first provides various types of design information, encoded in an appropriate manner. The evolutionary software can then be used to evolve a population of simplified three-dimensional building models representing a range of design alternatives. The design team can then select the most suitable model, and further develop this model to create a fully detailed design proposal.
A general framework for an EAAD system is proposed. The framework sets out a reasoned and coherent approach to developing EAAD systems. Abstract components and processes are identified, and certain constraints and requirements are highlighted. Finally, an implementation strategy is discussed.

2 Framework for EAAD systems

Evolutionary programs tend to decompose the creation and survival processes into a series of smaller sequential steps. Each step uses a set of transformation rules that define how one set of data should be transformed into another set of data. Data structures define the structure, content and semantics of a particular set of data. The rules and data structures within each step are highly interrelated. For example, the data produced by one step needs to be compatible with the rules in the next step.
Historically, the best-known type of evolutionary program is perhaps the genetic algorithm, invented by John Holland in the 1960s [HOL 75]. Within a simple genetic algorithm [MIT 96: 10], the processes of creation and survival can be decomposed into three steps: reproduction, evaluation and selection. The reproduction step is part of the creation process and produces a genotype (or chromosome) from the parents. The evaluation step spans both creation and survival processes and produces a fitness value for the genotype. The selection step is part of the survival process and uses the fitness value as a basis for deciding whether to select the genotype to become a parent.
The majority of the rules and data structures used by the genetic algorithm are very generic, thereby allowing the algorithm to be applied to a wide variety of problems. The genotype and fitness are both represented using standard data structures: the genotype data structure is a fix length binary string and the fitness data structure is a single real number. The reproduction and selection steps also use standard rules: the reproduction step uses single point crossover and mutation and the selection step uses roulette wheel selection.
The evaluation step, on the other hand, requires the definition of a problem specific evaluation rules. For example, one common application of the genetic algorithm is function optimisation, where the goal is to find a set of parameter values that maximize a function. In the case of a one-dimensional function, the evaluation step must first translate the binary string into a real number and then evaluate the function at that value. The fitness of a string is the function value at that point [MIT 96: 9]. If at some later stage a two-dimensional function needs to be optimised, then the evaluation rules will need to be changed in order to translate the binary string into two real numbers rather than one, and to subsequently evaluate the function using these two values. If the genetic algorithm is to be used for something other

than function optimisation, then the evaluation rules may need more extensive redesign. The other rules and data structures, on the other hand, remain unchanged. As a result, genetic algorithms can be applied to many problems with only minimal modification.

Genetic algorithms have been highly successful at solving a wide variety of problems. For example, Mitchell lists optimisation, automatic programming, machine learning, economics, immune systems, ecology, population genetics, evolution and learning, and social systems as areas where genetic algorithms have been successfully applied [MIT 96: 15-16]. She writes: "Interest in GAs has been growing rapidly in the last several years among researchers in many disciplines. The field of GAs has become a subdiscipline of computer science, with conferences, journals, and a scientific society" [MIT 96: 16]

However, despite their massive success, there are nevertheless many problems that would seem to be amenable to the evolutionary approach but for which genetic algorithms have been found to be too restricted. This is due to the fact that any rules and data structures inevitably result in biases and constraints being created within the evolutionary process. These biases and constraints will result in an evolutionary process that will favour some possibilities, while denying other possibilities. Although the genetic algorithm aims to be highly generic, the biases and constraints are nevertheless still significant, and these biases and constraints tend to conflict with certain types of problem. Koza writes: "Representation is a key issue in genetic algorithm work because the representation can severely limit the window by which the system observes its world… String-based representation schemes are difficult and unnatural for many problems and the need for more powerful representations has been recognized for some time" [KOZ 90] (quoted in [MIC 96: 4]).

Such conflicts result in two possibilities: either the problem is adapted to fit the genetic algorithm or genetic algorithm is adapted to fit the problem. For complex problems, many researchers have taken the latter approach. As a result, over the last two decades, a wide variety of problem specific variations of genetic algorithms have been created [MIC 96: 4]. Such algorithms embed problem specific rules and data structures with the algorithm. In *Genetic Algorithms + Data Structures = Evolution Programs*, Michalewicz advocates "the use of proper (possibly complex) data structures (for chromosome representation) together with an expanded set of genetic operators" [MIC 96: 3]. Throughout the book he argues for and presents a wide range of problem specific evolutionary algorithms that employ complex rules and data structures. These algorithms have been successful in solving complex problems that genetic algorithms have not been able to solve. However, a major drawback is that these algorithms are no longer generic. If any of these algorithms are applied to a new problem, they require major modifications.

EAD systems tend to follow a similar problem specific approach. Such systems are used to evolve designs rather than solutions to problems. These designs are often complex entities with many interrelated parts. The rules and data structures used by genetic algorithms have been found to be too restricted for evolving such designs and as a result EAD systems have tended to embed complex rules and data structures within the algorithm. For example, the GADES system developed by

Bentley uses a hierarchical genotype data structure, specialised reproduction rules, and a complex mapping process that generates design models from the genotype [BEN 96]. All these rules and data structures are embedded within the algorithm in such a way that changing or modifying them becomes impossible. Although Bentley claims that the system is very generic, the types of models that can be evolved are nevertheless highly restricted.

The same drawback therefore applies to EAD systems: every time a new design is to be evolved, the whole system needs to be recreated almost from scratch. This is particularly problematic for design systems, since the rules and data structures will affect the kinds of designs that are evolved. Designers will therefore want to define custom rules and data structures that reflect their idiosyncratic design approach. Designers must be able to experiment with a variety of rules and data structures and explore the resulting constraints and biases.

For EAD systems, the constraints and biases imposed by the rules and data structures can vary widely. Examples of constraints include the geometrical range of forms, the transformational operations available, the inclusion or exclusion of material properties such as colour, transparency and texture, and so forth. It is impossible for a program to produce designs that fall outside such hard constraints. Biases, however, are statistical and probabilistic in nature. The rules and data structures, while not actually making the evolution of certain types of forms impossible, may nevertheless make them very unlikely. For example, the rules and data structures may affect the stylistic and aesthetic characteristics of designs.

2.1 Creating a generic system

A framework for EAAD systems is proposed that advocates the development of complex rules and data structures tailored to the evolution of building forms that reflect the idiosyncratic design approach of a particular design team. However, rather than embedding these rules and data structures within the evolutionary algorithm, the framework allows them to be defined separately. This separation will permit the rules and data structures to be changed without requiring modification of the algorithm itself. The rules and data structures used by an evolutionary program are collectively described as the *evolutionary schema*.

In addition to these rules and data structures, the proposed framework also separates out another important aspect of the evolutionary process: data about the environment. This environment will include both design criteria and the design context. The design criteria may incorporate such things as spatial requirements and performance specifications, while the design context may consist of a description of the site and neighbouring buildings. Collectively, these are referred to as the *evolutionary environment*. This evolutionary environment plays an important role in the process of evaluating genotypes. For example, if the process of generating a building design from a genotype has access to data about the site, then this process can ensure that the building being generated does not extend beyond the site boundaries. Such non-genetic developmental influences are described as being *epigenetic* [FRA 95: 119]. The generative and evolutionary systems developed by Frazer often use such epigenetic growth processes [FRA 95: 57-58]. For example, in

the Reptile system – a system for generating building enclosures – "the configuration of the minimal construction or seed is compared with the user's requirements for a particular building, and the seed is grown, stretched, deformed and pruned, until it conforms with these requirements" [FC 79].

Due to the complexity of the evaluation step, the framework decomposes this step into four smaller steps, thereby resulting in a total of six steps. Three steps constitute the creation process and three constitute the survival process. These steps are seen as being distinct in a practical rather than formal sense. They organise aspects of the evolutionary process in a useful manner. The algorithmic procedure for each of the six steps is almost identical: they all consist of transforming one set of data into another set through the application of a set of rules. The rules are defined explicitly and can be changed without requiring any change to this algorithmic procedure. The data structures are defined implicitly by the rules and the data.

The creation process manipulates three different data structures:

- A data structure used to describe a highly abstract encoded representation, referred to as the *genotype*.
- A data structure used to describe simplified three-dimensional model of the building, referred to as a *skeletal model* (analogous to the phenotype).
- A data structure used to describe a skeletal model with additional information relating to its validity, referred to as a *validated model*.

The three steps that constitute the creation process are the *reproduction* step, the *generation* step and the *validation* step. The *reproduction* step produces the new genotype from genotypes of existing designs. This step extracts material from the parent genotypes and rearranges it to create new genotypes. The *generation* step produces the skeletal model from the genotype and the evolutionary environment. This step is a complex developmental process that 'grows' a new design alternative in response to the environment. The *validation* step produces the validated model from the skeletal model and the evolutionary environment. This step checks the skeletal model for errors and inconsistencies.

The survival process manipulates two data structures:

- A data structure used to describe a list of scores, each of which rates the quality of the design in some particular area, referred to as the *score list*.
- A data structure used to store the overall assessment of the relative suitability of the design, referred to as the *fitness*.

The three steps that constitute the survival process are the *prediction* step, the *assessment* step and the *selection* step. The *prediction* step produces the score list from the validated model and the evolutionary environment. This step simulates and analyses the skeletal model within the environment in order to try and forecast how the design would perform if built. The *assessment* step produces the fitness from the score list. This step calculates the overall fitness by comparing the scores in a score list to other score lists in the population. Finally, the *selection* step uses the fitness in order to decide whether to allow the design to survive or whether to delete the design form the population.

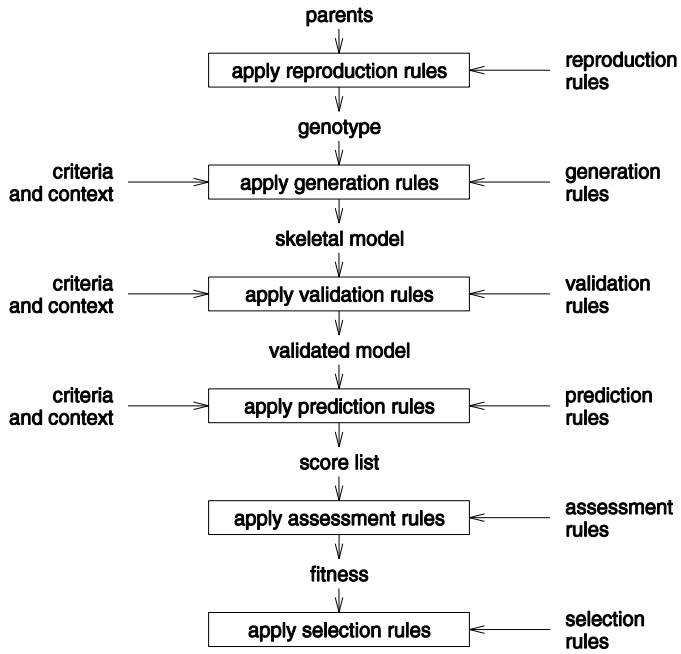

Figure 4. Separation of the evolutionary schema and the evolutionary environment from the main evolutionary process.

2.2 Integrating existing prediction programs

The whole evolutionary process is guided by the prediction step. The designs will evolve and adapt so as to satisfy the simulations and analyses defined within this step. These simulations and analyses aim to forecast how the design would perform if built. It is therefore essential that they are as accurate as possible.

Some of the more straightforward analyses may be defined as prediction rules. However, it would not be practical to attempt to define complex simulations and sophisticated analyses in this way. In these cases, the approach taken is to link to third party programs that are able to perform these simulations and analyses. Numerous programs already exist, such as lighting simulation programs, energy simulation programs, and structural analysis programs.

The linking of the evolutionary system with third-party programs imposes a number of requirements. These programs tend to be large and complex and often run on different operating systems. As a result, creating an evolutionary system that runs on a single computer is not feasible. Instead, the implementation must use a networked approach that allows the various prediction programs to be running on separate computers with different operating systems.

This approach has two additional advantages. First, it allows the different simulation and analysis programs to be configured and controlled by specialists in different geographical locations. Second, it can be extended to allow all steps to be performed in parallel and to be duplicated. For instance, several dozen networked computers could be set up so that each step might be duplicated on several computers with all steps running in parallel. This is perhaps most useful for the prediction step since the simulations and analyses are often complex and very slow. The evolutionary process requires each design alternative to be separately assessed. The networked approach can result in significantly speeding up this process by running simulations and analyses in parallel.

The proposed framework for EAAD systems can broadly be described as follows. A central database contains the population of designs at various stages of development. Each design in the database will go through six life cycle steps. Each of these six life cycle steps is treated as a distinct software component that may be running on a separate computer independently from the other components. Each component will have the ability to obtain data relating to one of the designs in the population database, apply the transformation rules to the data, and then return the newly produced data to the database. This new data will then be linked to the design. As well as these six life cycle components, two additional components are required: an initialisation component and a visualization component. The initialisation component allows the database to be initialised with random designs. The visualization component allows a design team to view the skeletal models that have been evolved.

In order to allow the components to operate independently from one another, the evolutionary process proceeds in an asynchronous manner. Designs in the population will be at different life cycle stages. When a component accesses the database for a design to process, a list of those designs currently at an appropriate developmental stage is created. One of the designs from this list is then randomly chosen and sent to the component. After processing the design, the component will send the design back to the database. The design will have progressed to the next developmental stage and will now be ready to be picked up by another component. All components will blindly repeat this process. If at some point a component finds that there are no designs left to process, then the component will ‘sleep’ for a short time before being reactivated. This asynchronous process means that the change from one generation to the next is not clearly defined, but will instead be a gradual transition.

The reproduction step will continuously be creating new designs, and the selection step will be continuously deleting designs. The desired size of the population is specified at the start of the evolutionary process. Whenever the population falls below a certain size the selection step will be put to sleep, and whenever it rises above this size, the reproduction step will be put to sleep. As a result, the population will remain dynamically stable.

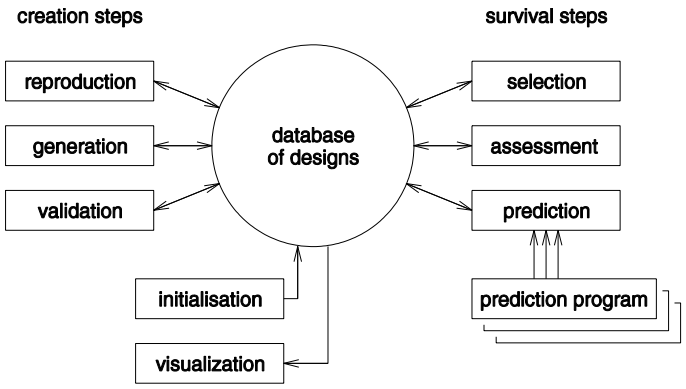

Figure 5. Networked configuration of the evolutionary process.

2.3 Implementing an EAAD system

The implementation strategy needs to fulfil a number of requirements. Foremost is perhaps the requirement for a rule processor that can apply a set of rules to a set of data. Such a rule processor could be used in all of the six life cycle steps. In addition, such a processor could also be used by the visualisation component in order to generate a file compatible with existing model viewers and CAD programs. In each case, the processor would transform one set of data into another set by applying a set of rules. Clearly, the processor would have to be capable of manipulating a wide variety of rules and data.

Neither the rules nor the data can be specified in advance. Furthermore, the data structure of the data can also not be specified in advance. The data structure of the rules, on the other hand, can be specified. This will allow the rule processor to access a document that contains a set of rules and to recognise the different the rules. However, in order to apply these rules to a document containing the data, the processor needs to be able to recognize and parse this data. Therefore, two languages are required: a language with a very generic structure that allows a wide variety of data (with different data structures) to be described; and a language with a data structure that defines how rules should be described. In addition, in order to minimise any new skills that would have to be learnt, it is essential that the languages are not proprietary or esoteric.

Two languages that fulfil these requirements already exist: XML and XSLT. XML (Extensible Markup Language) is a language for describing structured data. XSLT (Extensible Stylesheet Language: Transformations) is a language for describing rules that transform of XML documents. Furthermore, a large number of rule processors for these languages already exist. For example, the Apache Xalan-Java processor is a Java implementation of such a rule processor. These processors apply the XSLT rules to a specified XML source document and create a new XML result

document. Currently, there is also an explosion of documentation and support for XML and XSLT, thereby substantially lowering the skill-learning threshold for non-technical users.

For implementation, the Java programming language is being used. Java has advanced built-in networking capabilities that make it very suitable. The networked configuration is created using Java servlet technology. Servlets are Java programs that run within a servlet container on the server. The six life cycle components are each implemented as Java clients that connect and communicate with the servlets running on the server. The Apache Xalan-Java processor is used within each of the six clients. For example, to perform the reproduction step, the reproduction client downloads the parent genotypes from the server and merges them to create a single XML source document, the client then transforms this document according to the reproduction rules defined in an XSLT document, and finally the client uploads the XML result document that is the new genotype back to the server. The server then stores the new genotype in a database. The Apache Xalan-Java processor is also being used in order to generate VRML files that can be viewed in standard VRML browser.

The use of XML and XSLT present considerable advantages, both in terms of simplifying implementation and in terms of reducing the skill-learning threshold. By using an existing XSLT processor further enhances these benefits. However, XSLT may not always be an appropriate language for defining certain complex transformations. The framework has already identified the need to integrate third-party prediction programs within the prediction step. In addition to this, the generation step may also require a more powerful language than XSLT. The generation step involves the definition of complex three-dimensional forms requiring large amounts of numerical manipulation, while XSLT is primarily focused on symbolic manipulation. Two possible approaches present themselves. One approach is to extend the power of XSLT with extension functions. For example, Xalan-Java allows Java extension functions to be called from within the XSLT rules. A second approach if to use a different kind of processor in this step. For example, existing solid and surface modelling kernels such as ACIS or Parasolid might be used as a basis for developing a generative process. This second approach, although potentially powerful, has the disadvantage of further complicating the process of defining rules and data structures. As a result, the use of XSLT with Java extension functions will first be explored.

3 Conclusions

The EAAD system is under active development. This system is seen as a general-purpose toolkit for performing evolutionary design experiments. The two key characteristics of the system have been highlighted. First, the separation of the evolutionary rules from the evolutionary process allows new design schemas to be encoded. Second, the networked configuration allows third-part prediction programs to be integrated with the system and allows the life cycle steps to be executed in parallel. The system currently under development aims to provide the community of experimenters and researchers with the core functionality of an EAAD system.

Users of the system will require a certain level of programming knowledge in order to create the evolutionary schema and encode the evolutionary environment.
The system is therefore not targeted at the design community. However, in the long term, this system is seen as just the beginning of a lengthy process of eventually developing more user-friendly EAAD systems. With systems such as the one currently under development, the community of experimenters and researchers will gradually develop an understanding of the types of rules and data structures that are most successful in the generation and evolution of building forms. It is envisaged that certain patterns of successful combinations of rules and data structures will emerge. Once such successful patterns are identified, then a second stage of software development can take place whereby user-friendly systems are developed that embody these patterns in their mode of operation and in their user interface.

4 Acknowledgments

Our research project is supported by a UGC PhD project grant from the Hong Kong Polytechnic University.

5 Bibliography

[BC 02] Bentley, P. and Corne, D. (eds.), 2002. *Creative Evolutionary Systems*, Academic Press.

[BEN 96] Bentley, P. J. 1996. *Generic Evolutionary Design of Solid Objects using a Genetic Algorithm*. Ph.D. Thesis, Division of Computing and Control Systems, Department of Engineering, University of Huddersfield.

[BEN 99] Bentley, P. (ed.), 1999. *Evolutionary Design by Computers*, Morgan. Kaufmann Publishers Inc, San Fransisco.

[DAW 83] Dawkins, R. 1983. Universal Darwinism, *Evolution From Molecules to Men*, edited by D. Bendall, Cambridge University Press, pp. 403-425.

[FC 79] Frazer, J. and Connor, J. 1979. A Conceptual Seeding Technique for Architectural Design, *PArC 79, Proceedings of the International Conference on the Application of Computers in Architectural design*, Berlin (Online Conference with AMK, 1979), pp. 425-34.

[FRA 95] Frazer, J. 1995. *An Evolutionary Architecture*, Architectural Association Publications.

[HOL 75] Holland, J. 1975. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor.

[KOZ 90] Koza, J.R. 1990. *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems*, Report No. STAN-CS-90-1314, Stanford University, 1990.

[MIC 96] Michalewicz, Z. 1996. *Genetic Algorithms + Data Structures = Evolution Programs*, Sringer-Verlag.

[MIT 96] Mitchell, M. 1996. *An Introduction to Genetic Algorithms*, MIT Press.