The Hong Kong Polytechnic University
School of Design

A Computational Kernel for Supporting Generative and Evolutionary Design

Chan, Kwai Hung

A thesis submitted in partial fulfilment of
the requirements for the Degree of
Doctor of Philosophy

June, 2007

Certificate of Originality

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____ (Signed)

Chan, Kwai Hung (Name of student)

Abstract

Evolutionary Computation techniques have been used in design systems for exploring and generating design solutions in recent years. However, most of the current evolutionary design studies concentrate on analysis and optimisation of design solutions for problems at the stage of detailed design. There has been comparatively less research on the synthesis and generation of design solutions through a dynamic process of evolution and refinement, at conceptual stage of design process. Furthermore, many conventional studies on evolutionary design do not support multiple representations of design objects at different levels of abstraction, which are essential for exploring design solutions in an incremental and evolutionary manner.

To overcome the above problems, a computational kernel is developed in this thesis for the development of design supporting system applications, based on a Generative and Evolutionary Design (GED) model. With this kernel, design objects can be dynamically evolved in a specialisation process in which design solutions are developed from abstract levels to detailed levels. Generative mechanisms are integrated with this multiple representation scheme to manipulate and generate new design solutions from basis and abstract design objects in an interactive manner which involves users in making design selections. This study focuses on the three important aspects of this kernel, 1) modelling design object and design process in a generative and evolutionary manner within an integrated computational platform; 2) adapting and capturing the knowledge of how design objects are generated within this platform; and 3) enhancing the exploration ability of generative and evolutionary design applications with the use of a number of different evolutionary and generative computing techniques, including Genetic Algorithms and Cellular Automata.

Three examples of applying the GED kernel to design tasks are tested and evaluated. The results show that it is feasible and applicable to use the kernel as the core architecture of computational design systems for supporting generative and evolutionary design applications, with improved generative, explorative and adaptive ability in producing potential design solutions effectively and efficiently.

Publications Arising from the Thesis:

Chan, K.H., Lee, H.C., Frazer, J., and Tang, M.X. (1999) A Hierarchical Design Interface for Collaborative Design. *Proceedings of the 6th ISPE International Conference on Concurrent Engineering (CE99)*, 1-3 September, Bath, pp.181-186.

Chan, K.H., Lee, H.C., Frazer, J., and Tang, M.X. (1999) A Hierarchical Design Interface for Integrated CAD Systems. *Proceedings of the Fifth International Conference for Young Computer Scientists (ICYCS '99)*, 17-20 August, Nanjing, pp.830-835.

Chan, K.H., Frazer, J. and Tang, M.X. (2000) Handling the Evolution and Hierarchy Nature of Designing in Computer-Based Design Support Systems. *Proceedings of the Third International Conference Computer-Aided Industrial Design and Conceptual Design (CAID & CD '2000)*, November, Hong Kong. Beijing: International Academic Publisher, pp.447-454.

Chan, K.H., Frazer, J. and Tang, M.X. (2001) Interactive Evolutionary Design in a Hierarchical Way. *Proceedings of 4th Generative Art Conference 2001,* 11-14 December, Milan, Italy. (Internet - http://www.generativeart.com/)

Chan, K.H., Frazer, J.H. and Tang, M.X. (2002) An evolutionary framework for enhancing design: a kernel of computational systems for enhancing design with dynamic structure of hierarchical representations. In: J.S. Gero and F. Sudweeks (eds.) *Artificial Intelligence in Design '02*. Boston: Kluwer Academic, pp.383-403.

Acknowledgement

During my research work, a lot of people have contributed to the completion of my PhD study. I have received many invaluable ideas, suggestions, comments, guidance, encouragement and criticism from them.

I owe much debt to Professor Ming-Xi TANG, my supervisor, for his important suggestions, advices and encouragement in my difficult times, and Professor John Hamilton FRAZER, my co-supervisor, for the precious inspiration of my research topic. I would not forget the help and supports from the members of our Design Technology Research Centre. I would also like to thank Professor Daizhong SU, Professor Mitchell Miendger TSENG and Professor Michael Kin Wai SIU, for giving me comments and suggestions to further improve the quality of this thesis.

Finally and most importantly, silent and wholehearted support of my family is a must for the completion of my study. I cannot imagine how my study can be finished without the patience and tolerance of my father, mother, brothers and sisters. I owe them my sincere gratitude for their understanding and bearing.

Table of Contents

List of Figures

Part I:

Introduction and Literature Review

Introduction

The study on design has resulted in the formulation of many models including computational models that provide bases for developing software systems supporting design activities. A typical computational model is based on a process in which design is seen, at least in its computational representation, as an evolution of a description from its most abstract form (the original design problem) to the least abstract one (the final design solution). The process transforms a given problem (a need, an idea for example) to be tackled, through a series of designing activities, to a final solution to that problem, with the support of computation that generates the complete instruction for manufacturing processes to start.

In a process based approach to modelling design, many descriptive and prescriptive design models have been developed in literature. These models give clues to how design activities can be generalized as a problem solving process in which different methods such as search and optimisation might be used. In the meantime, the development of computational design support systems can only be limited to supporting various domain specific tasks of the whole design process in a disintegrated manner. The reason is that there have been no theories or methods to be implemented in the complex domain of design in such a convincing way that they can actually be quantitatively or at least qualitatively evaluated to show that these models are working in the similar but more efficient ways for designers whose immediate task is to produce good designs.

Most current computational design systems suffer from two major problems. Either they cannot be scaled up to tackle a whole design project during which the

designers have the ultimate responsibility to play as the leaders of the project, or they cannot be generalized in a convenient way so as to provide solutions to a wide range of design problems even in the same domain. In other words, a completely workable computational model does not exist to offer a revolutionized way for the alternative approach to solving design problems, even with the most updated technology in 3D computer graphics and Internet based design collaboration systems.

This study develops a computational kernel for the development of computational design support systems, based on a Generative and Evolutionary Design (GED) model. The proposed kernel based on the GED model is built on an evolutionary process in which design objects or products are generated in a process in which design activities are guided through a hierarchical structure which represents a way for generalization and specialisation of the objects or products being designed. In such a hierarchy, design objects are linked with a generalization and specification process in which generative mechanisms are invoked to transform the states of design from one to another, normally in a top down manner. In such a way, a less abstract design with more detailed attribute values of design objects are generated from more abstract and more conceptual ones, by the invocation of relevant generative mechanisms. A design process is modelled as an evolutionary activity which changes those design objects with the support of design generalization or specification mechanisms.

With this kernel, the knowledge of design generation can be adapted or captured in a form of generative mechanisms, such that not only design objects are generated but also the process of how they are generated from a more abstraction form is recorded. This gives indication to solve similar design problems efficiently and effectively so that the knowledge of how a design is evolved is captured by a process model, together with the final result.

1.1 Generative and Evolutionary Design

Evolutionary techniques have been used by many to solve search and optimisation problems in various Engineering fields (Jain and Gea, 1998; Khuri et al., 1995). Furthermore, supporting design activities with evolutionary computation techniques has raised much attention (Bentley, 1999; Frazer, 1995; Gero et al., 1997; Sims, 1991). The basic rationale behind these applications is the belief that the design process is similar to evolutionary processes of nature.

Designing from nothing is rare (if not impossible), and is generally based on the existing or past design primitives or building blocks. Analogous to design of lives in the natural world that environmentally-fit living things and species survive better while the poor ones tend to extinct, searching for an optimum design solution can be seen the same way. Therefore many researchers developed evolutionary computational techniques to simulate design environments in which imaginative design solutions emerge from evolutionary processes and the data structures which embody intelligent properties or inference mechanisms.

Frazer is one of the first who used evolutionary concept in architectural design (Frazer, 1995), while Sims applied GA to graphic design and the design of artificial creatures (Sims, 1991, 1994). In the meantime, many other studies concentrated on finding the methods for exploring design problems in the domain of engineering (Gero et al., 1997; Graf, 1995; Poon and Maher, 1996). In the field of art and design, some researchers applied evolutionary computation techniques to artistic and form design (Rowbottom, 1999; Todd and Latham, 1999; Witbrock and Neil-Reilly, 1999). As a result of these developments, many new evolutionary design methods have emerged (Bentley, 1999). The capability and efficiency of these methods in producing satisfactory design solutions that meet the expectations of designers have proved to be promising although many design tasks tested on these methods remain abstractive or simplified.

Some researchers in computational design also investigate issues in applying generative techniques for supporting design. This approach, termed generative design as in this thesis, uses computational generative techniques to generate or develop design object, unlike conventional parametric approaches which only alter the basic preset parameters of the design object. In these research works, generative evolutionary design is thus related to designing with generative and evolutionary techniques.

However, most of the current evolutionary design studies concentrated mainly on the analysis and optimisation tasks at the stage of detailed design, which is a later stage task in the whole design process. There are comparatively less research in the area of early stage design tasks, i.e., the synthesis and generation of design concepts. At early conceptual design stages, problem specification and design requirement are not concretely definable. In this case, design specifications inevitably keep changing with amendments to the problem requirements as more information becomes available and more problems are discovered. Therefore, conventional evolutionary techniques cannot adequately function with this dynamic nature of design process. In the terminology of evolution, it is difficult to formulate a generic process which requires a pre-determined problem specification in real design for the definition of a suitable solution chromosome structure, an evaluation function and a selection strategy in computational representations which are required by the evolutionary algorithms. In other words, although many applications of evolutionary algorithms have proved to be satisfactory and promising, the formal theory of such a process in design in terms of a generic representation and software architecture has not appeared.

A problem associated with the complexity in defining a unified theory of generative and evolutionary design is that different representations of the design objects are manipulated at different design stages, as the design problem and its solutions are transformed by designers from an abstract level to reach the required level of details for various purposes, such as proposing a concept, tending

contract, generating initial design, completing the detailed design, specifying manufacturing instructions, generating assembly drawings etc.

For example, at the early conceptual stages, descriptive, symbolic and functional representations of design objects are often manipulated while physical geometrical structures in 2D or 3D are handled at later detailed design stages. In conventional evolutionary design with Genetic Algorithms (GA), the genotype (computer representation) can be treated as an abstractive form of the phenotype (the real design solutions). Such a mapping method from genotype to phenotype embeds the knowledge and information of how a design object might be generated from an initially abstract form to the final form with many accurate details. However, if such a mapping can only be formulated at the beginning of the design task and the mapping cannot be changed as the design process goes on, it can only model a fixed design task with limited scope for changing the design problem structure. A basic assumption of formulating design computational terms is that this design problem structure is supposed to change as design proceeds with new variables and parameters that are subsequently resolved with the new constraints. Therefore, this promoted the thinking for this thesis, which deals with this problem in a generic and systematic way, through the development of a generative and evolutionary kernel. The essential objective for such a kernel is that it is able to guide the designer or a software developer to formulate a dynamic design problem space through a hierarchically structured computational representation and integrated software kernel components, and thus provides better mechanisms for formulating the design process rather than a predefined design task.

1.2 Objectives and Research Methodology

In order to provide a generic kernel to deepen the study on generative and evolutionary design, this thesis developed a computational kernel of generative and evolutionary design based on a process based representation and a generalization-specification hierarchy in which design objects are evolved by

evolutionary computation mechanisms, with the ultimate goal of supporting generative and evolutionary design at a level of system configuration and design knowledge acquisition.

The objectives of this thesis concentrate on the formulation, implementation and evaluation of a computational kernel that supports the following generalized design activities:

a) Modelling design objects and design process in a generative and evolutionary manner with a structured representation,
b) Capturing the knowledge of how a design object is generated with generative and evolutionary computation techniques in such a structured representation, and
c) Simplifying the process of mapping design applications to a generative and evolutionary system to allow quicker system configurations with the structured representation and its related evolutionary computing methods and interfaces.

The aim of this study is to test whether it is possible to develop this generic computational kernel, named Generative and Evolutionary Design (GED) kernel, for it to be used as the core architecture of computer-based systems for supporting design, which is similar to the way in which 3D solid modelling kernels such as Parasolid (support EDS Unigraphics) and ACIS (supporting ProEngineer) are used to support parametric design. A prototype of this kernel is developed in this thesis for demonstrating several key generative and evolutionary techniques which are implemented in the GED kernel as the main evolutionary mechanisms. This prototype system can demonstrate its feasibility and applicability in supporting design for solution adaptation and exploration.

The aim and objectives of this study are approached by

a) examining the nature of design with a view that evolutionary computing and structured representation can help to improve the efficiency in using computer based design support systems, and formulating a computational kernel based on this study and understanding of design,
b) developing and implementing the proposed Generative and Evolutionary Design (GED) kernel, which provides a foundation for the application of generative and evolutionary techniques in design domains with examples of realistic scales,
c) integrating several main generative and evolutionary computation methods into the GED kernel so that design assistance in terms of adapting design solutions and exploring design alternatives can be provided, and
d) evaluating the GED-based computational systems for design applications, with the demonstrations with which the feasibility and applicability of the GED kernel can be qualitatively analyzed for improvements and validations.

Following the initial findings achieved at the first stage of the study, related results were reviewed and analyzed. A further literature review was carried out in order to have a deeper understanding of design problems in general and product design in particular. A design model was then studied, formalized and constructed in a general manner, which includes the tasks of synthesis, analysis and evaluation. The applications of generative and evolutionary techniques in the proposed design model were investigated and evaluated. Furthermore, a prototype of the GED kernel was implemented and integrated with a commercial CAD tool. The GED kernel was then applied to solving specific simulated and practical design tasks. The results of applications were further evaluated for validating the feasibility and applicability of the kernel in several different design domains with examples reported and analyzed.

1.3 Significance and Contributions

A computational kernel for generative and evolutionary design offers the opportunity to confront the problems of applying various new computational techniques including genetic algorithms in a generic and scaled-up manner for the ultimate goal of achieving better design with efficiency. As such new representation and integration methods are needed in order to shorten the process of building an application. In the process of developing this kernel, knowledge and strategies are discovered for a unified representation of design objects related to their process of being explored and optimized. This top town approach provides insight on how the knowledge outside the discipline of design can be utilized and integrated to the theories and methodologies of design which by its nature is a multidisciplinary activity and process.

From a perspective of design, it is also necessary to know exactly what the prospective is and where the opportunities are for using computational techniques in improving design in terms of, supporting the tasks achievable by human designers more quickly, and more importantly, supporting the designers in deriving better design solutions which would be otherwise unachievable or difficult to achieve by designers themselves without the support of such kernel and its related computational techniques.

The implementation and evaluation of the kernel involves its application to three different design examples, including the development of 3D product forms and structures which are normally supported in a certain degree partially with parametric technology. The developed kernel in this thesis provides an alternative and potentially more interactive and efficient way of exploring design problems. The contributions of this thesis derive from the integrated nature of this kernel with its formulation, generative and evolutionary computing techniques that have not been tested in such a scale and a generic context.

1.4 Thesis Overview

The thesis presents the Generative and Evolutionary Design (GED) kernel with its theoretic basis on Artificial Intelligence with a focus on knowledge based systems and genetic algorithms. The key concept of the kernel is built on an integration of design object and design process within a structured representation scheme, which takes a design object from an abstract level down to the hierarchical structure to its detailed level. At each level of such a hierarchy, generative and evolutionary mechanisms are attached. Each evolutionary element has its design attributes, data or parameters and is allocated at a specific level in the GED hierarchy. These design attributes represent a design product or object at a specific abstraction form. Design process is then related to the evolution of these elements in the model according to their attached generative and evolutionary mechanisms.

As design changes dynamically, the hierarchical structure of the whole representation as well as the connected elements in each layer are also evolved. When the kernel is applied to solving design tasks, the evolutionary elements at an upper layer represent design products in a much abstract level closer to the original problem. Evolutionary elements at lower layers conversely represent design in much more concrete formats closer to the final design output domain. For example, the upper layer may represent textual specifications while the lower one may represent 3D models.

With this GED kernel potential designs can be explored through changing design parameters and parameters of generation mechanisms. This kernel further supports the evolution of generative mechanisms themselves, and enhances its explorative ability. Through adaptation methods, generative design knowledge can be captured and reconstructed with such dynamically evolving generative mechanisms. Figure 1.1 shows an implemented design supporting system presented in details in Chapter 8, which is embedded with the GED Kernel (GEDK) shown in the bottom left corner of the diagram.

In the rest of this Part I, a literature review is presented. Conventional design representations, models and supporting systems are presented in Chapter 2. In Chapter 3, advanced research work in generative and evolutionary techniques for design is further discussed.

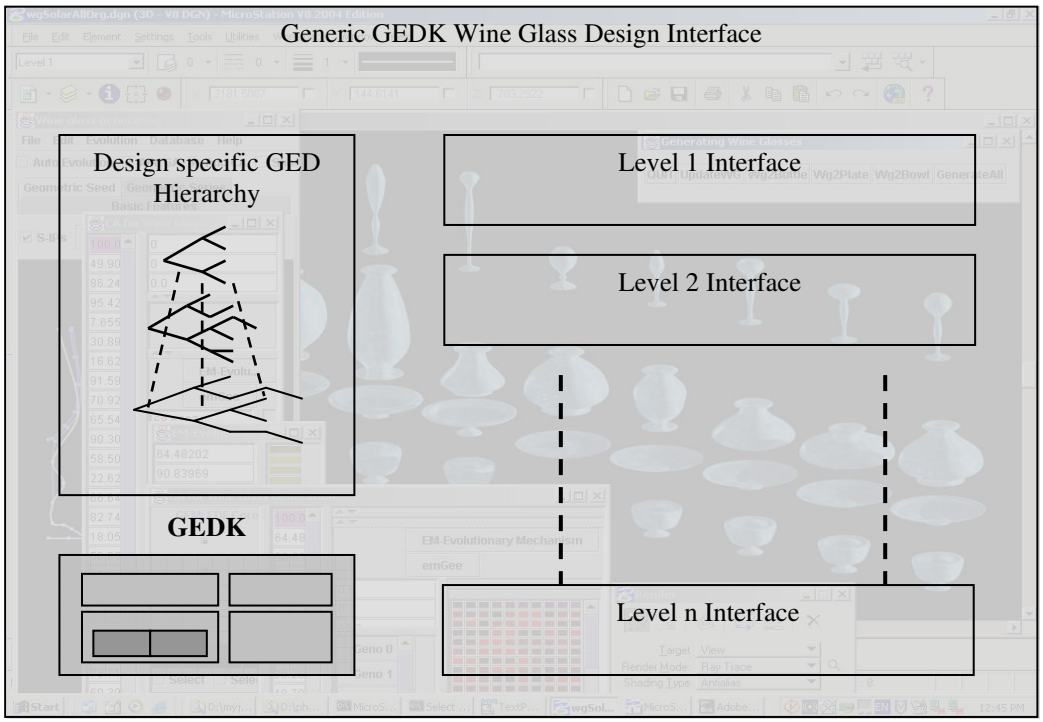


Figure 1.1: A GEDK-embedded design supporting system.

In Part II, a Generative and Evolutionary Design (GED) model is presented. Discussion in Chapter 4 is related to the issues of generative design representation, design exploration and adaptation, and in multiple representations of designs, on which this thesis focuses. In Chapter 5, the fundamental features required in a GED model are presented, followed by a discussion on the importance of design exploration and knowledge adaptation in multiple representations of designs in such an evolutionary process. The formulation of the computational kernel based on this Generative and Evolutionary Design (GED) model is also described in details, and an implemented GED kernel prototype is introduced.

In Part III, three examples of applying the kernel to computational design tasks are presented. They demonstrate how the kernel supports designing with enhanced explorative and adaptive ability. The first demonstrative example in Chapter 6 illustrates how the GED kernel can automatically build a GED hierarchy. It forms a complex plant-like structure from a single "self-replicating" evolutionary element root, and different effects can be explored when evolutionary elements located at different levels of the hierarchy are manipulated. The second demonstration in Chapter 7 shows how the kernel supports design adaptation through simple knowledge reconstruction in an example of 2D image pattern generation and matching with Cellular Automata.

In Chapter 8, a design demonstration presents an application for generating a family of wine glasses and their "relative" utensils, when the GED kernel is integrated with commercial CAD tools. Comparison of these three examples in various aspects is then discussed and evaluated. Finally a conclusion and the issues for further research in this direction are presented in Chapter 9, the final chapter.

Design and Support Systems

Design involves complex processes, with activities in different areas, including idea generation, aesthetics, cognitive expression, problem identification, market evaluation, problem specification, conceptual solution searching, detailed design, product modelling, manufacture engineering and product evaluation. Cross (1994) categorized these activities into four main groups: communication of designs, evaluation of designs, generation of designs, and exploration of designs.

However, we do not have a "universal" definition for the term "design" as yet. Neither can we compromise for a "generalized" agreement of what "design" is related to, without having to refer to different contexts. This section introduces the research related to the understanding of design in the context of developing computer based design support systems. Most of the research reviewed in this chapter focuses on the cognitive and computational modelling of the design process with which Artificial Intelligence techniques can be used in a formulation involving designers using computer program to deal with a design problem or at least a part of a design problem.

2.1 Design and its Contexts

The meaning of design and the activities associated with it differ from one field to another and from one aspect to another. Craftsmen, architects, and the public all have different perspectives of what design is and how it is related to their working and living. Mitchell (1996) suggested that various design professionals in different domains, such as architects and design educators, see design dramatically differently. For the people who develop computational representation

for the implementation of computer based design support systems, design is seen in symbolic terms that are closer to computer coding and programming languages. Even for this kind of professionals, they still have to understand design in general terms or their methodologies will not match the expectation of the users who will use the systems to participate in design activities.

Although design is generally related to human behaviours and activities, there is no restriction to extend the concept beyond that. Many regard design in the context of nature as the very origin of design. Various articles present and appreciate the beauty of natural design such as French's (1994). Still the meaning of design deviates from one perspective to another even in a natural context in which one can perceive design in terms of a macro (such as the whole ecological system) view or a micro (the cellular mechanism) perspective. Design can even be considered in two extremes in (some) human's perspectives – scientific versus artistic. The comparison of gearbox and fashion design in Lawson's article (1990) gives an example of this case.

Without losing its generality, design involves a process of making things the way they are expected to become. When one talks about the design of nature, there is an expectation that such a design emerges dynamically from an environment through competing with species within a process that may take a long time to evolve. When one talks about the design of a craftsman, there is an expectation that the result would show some unique features that are only made possible by skill and experience.

However design is mostly referred to human design in our daily life, where this design is for the benefit of human beings in terms of convenience and comforts. In a human context, design involves a process of making artefacts the way they are expected to function. The scope and focus of this study combines these two kinds of understanding in design, and a software kernel for supporting design is developed by the motivation of bringing computational design techniques that

simulate the evolution of a natural environment in which a design emerges from evolution to normal design tasks such as designing wine glasses or drinking devices.

2.2 Design Representation

There are some research works in analyzing design activity such as the Delft project (Cross et al., 1996), which analyzes designers' practices through video recordings of designers working on engineering product design. There are also studies in natural intelligence of human being in design (Cross, 1999) and in design psychology (Crozier, 1994; Lawson, 1990). Furthermore, a large number of design research studies propose theories to reflect its nature, and develop concepts and techniques to represent, support or even try to automate it.

There are many studies in representing design, from formal mathematical design representation (Braha and Maimon, 1998; Gero and Tyugu, 1994) to descriptive and prescriptive models such as Quality Function Deployment (Menon et al., 1994) and Design Function Deployment (Shahin et al, 1998; Shahin and Sivaloganathan, 1998). Although there are diverse interpretations and understandings of what design is in different contexts, there are many common features in the basic nature of design representation.

2.2.1 The Knowledge of Design

If the main goal of design is to make artefacts for fulfilling our expectations of certain new comforts brought about by new functions that cannot be provided by the designs that already exist, knowledge and problem solving skills are needed to make new artefacts or to improve the existing artefacts, both serving the original goal of design to a certain degree of satisfaction and efficiency. Since this goal is based on the people and the environment they act on, design is closely related to the knowledge of people's inner (such as cognitive) and outer (such as physical

and chemical) world. The knowledge of a designer, a team of designers, and even the database of a design supporting system for a specific design domain becomes the fundamental "nutrition" of that targeted design object. Lack of this knowledge means lack of nutrition and leads to an ill resolved design.

Knowledge involves skills, experiences and techniques in various aspects, particularly the ability of analysis and synthesis. With adequate knowledge, suitable solution(s) can be searched or explored to solve a given problem. This requires the acquisition of knowledge in both the problem domain and the solution domain of the design task.

Some people argue that design is solution-oriented. Lawson's study (1979) of cognitive strategies used by architecture and science students for solving a given structuring problem showed that they concentrated on the solution-oriented and problem-oriented approaches respectively. When the main goal of a design has been achieved or the design problem has been solved satisfactorily, it does not really matter how much the user knows about the design activities involved. This nature is reflected on the long history of mysterious, but workable, "blacksmith" type design involving implicit/tacit knowledge.

However, in developing computational tools for supporting design the understanding of the design problem becomes more important. Without such an understanding it is difficult to obtain and store the knowledge useful for solving general design problems such as generalization, specialization, searching, and optimization. There is a need for formulating a repeatable process during which variations of design or entirely new designs can be generated and re-generated. Some who tried to view design as a matter of science have been working hard to identify methodologies or processes in a systematic way. Without such methodologies or processes, it is difficult to deal with the problem of design in a generalized way in order to achieve better efficiency and quality. Over time, the environment, designers and other influential factors related to the original design

problem may change, the understanding of the problem in a domain independent context helps to establish a pattern or procedure to produce similar designs or improve imperfect ones.

Understanding the design problem, even a simple one, to a level at which one can formulate computational representation and a process is not straight forward. To seek design solutions in the unconsolidated solution domains further complicates the task. There is still a large "black-box" in design. One cannot completely know how a designer works, and how human links problems to solutions, or solutions to problems. Therefore, in a computational process of design, there should be a mechanism to cope with uncertainty of problem definition and there should be a mechanism for evolving an initial design solution formulation into detailed one with more and more uncertainly cleared up in the process of exploring both the solution and the problem.

2.2.2 Design Process

Generally, it is agreed that most design processes start with an identified need (Black, 1996) (French, 1999), while the final output (products, services, etc.) is the eventual one to make contact with users. However, the final output can only tell us the final output of a design, but not the design itself. One basic nature of design is its process (Blessing, 1994; Navinchandra, 1991), the process to make things function or behave in our expected way.

Design process is seen by many as an evolutionary process during which the solution to the problem or the objective of the goal is explored. As Medland (1992) wrote - *"The design process is the activity of turning ideas into reality"*. This evolving process can be endless, that it cannot have a finite and identifiable end (Lawson, 1990). Donald Norman also mentioned in his conversation with Mitchell (Mitchell, 1996) that *"Design never ends. Even the most successful*

design will have to keep evolving continually in response to new practices" (p.xviii).

This process can be in various forms. However, it is commonly decomposed into several stages, especially in systematic designing (Braha and Maimon, 1998). The final stage of this design process leads to a direct implementation or an application of the solution to the problem. Various models, methods or approaches have been proposed and studied. Some refer this process in a chaotic way while others are more concerned with systematic formulation of the process for design automation, but both approaches agree that an evolving process perhaps suits computational formulation of design better since this allows both control and interaction to take place during evolution, which together is seen as design exploration, rather then pure design problem solving.

2.2.3 Design Problem Hierarchy

Although a few design products do introduce totally new principles and concepts, the majority of designs are rearrangements of not only the existing principles and concepts, but also the existing standard components (Black, 1996), especially in engineering design. Design may range from "invention" to "redesigning". In fact, it is a part of daily practices in which choices and decisions (design solutions) are made among various alternatives (possible design solution domains) for different problem solving. It is not difficult to find examples of these in designed products one encounters everyday, from simple paper clips to complex bridges and buildings that Petroski (1996) introduced.

Creativity is an important factor in design as a professional practice, which provides users with surprises, and sometimes fun. The emphasis is on the uncertainty, un-commonality and unexpected results, particularly to the users. Thus, this creativity provides an "unexpected" way for "expectedly" solving a given problem. Exploring ability is an important element in creativity, which

searches for unknown, unusual or unexpected alternatives to solve the design problem. This also requires the ability to adapt important and useful features from the explored solution candidates. With this adaptability, the exploration tends to converge to an optimum or at least a suitable solution within all possible solution domains.

Often the degree of creativity is proportional to how unexpected the design solution might be. This unexpectedness is closely related to the seeking of new or alternative ways to meet the goal. When a design can be realized in different degrees of abstraction in terms of how close the design representation is to the final solution, more uncertainty exists and the problem space is less constrained at a higher level of abstraction of describing a design problem. The alternation of design representation in higher abstraction produces higher degree of creativity while the lower level produces lower, resulting in more knowledge and skills being needed in the process of exploration and optimization.

With extensive analysis or long-term practices of the design problems, the knowledge of that design problem can be built up and the understanding of how to solve the problem, under a specific context, can be systematized. When the understanding is at the stage that any alternation of the problem within a finite domain can be solved systematically, this type of design is sometimes referred to "routine" design or "mature" design. This design process can often be structured in a hierarchical or layered-network form at this stage. Any alternation in a higher level of the hierarchy would not require a "re-design" in the levels below.

One of the ways with which the degree of complexity of exploring creative and abstract design is to build a hierarchy of knowledge of certain products in which problems are explored with a top down approach, that is, conceptual solutions or so called creative ideas are explored at higher level of the hierarchy with more abstract definitions of the problem with less constraints and variables. As the

solution space becomes more and more confined, lower level details and constraints are introduced to provide optimized results to the problem.

2.3 Computer-Based Design Support Systems

2.3.1 Models and Methodologies

Jones (1992) mentioned that *"the new methods that have appeared so far are only partial solutions to modern design problems"*(p.27). It is not realistic to construct a general model and methodology for all design tasks, because we still cannot fully understand and formulate what design is in all aspects. However, there are many "simplified" versions of design models and methodologies, which have been developed and shown many successful applications, particularly in engineering design. Cross (1994) gave a detailed description of the nature of engineering design, its activities, problems and abilities. Others (Dhillon, 1996; Dhillon, 1998; French, 1999; Pahl and Beitz, 1996; Pugh, 1991) also provided design methodologies in various engineering aspects.

Cross identified four themes in this relative short history (Vries, 1993), that he labelled with four words that typified the activities in those themes: prescription, description, observation and reflection. The first three themes focus on the role of flowchart representations for design process and the extent to which experienced and beginning designers follow the steps in these flowcharts. Computer models were made to represent the thinking modes that were found with designers and from these possibilities for computer assistance to designers were developed. The theme of reflection is a more philosophical aspect that has become a major issue of consideration in design methodology.

As Navinchandra (1991) explained, conceptual design is the part of design process in which: problems are identified, functions and specifications are laid out, and appropriate solutions are generated through the combination of some basic

building blocks. Conceptual design, unlike analysis, has no fixed procedure and involves a mix of numeric and symbolic reasoning.

A typical starting point in a design process is still sketching, despite the great advance of computing technology. In fact, the importance and influence of sketching in design does not diminish after these years of studies and developments of many computer-based design support tools. The special issue (Volume 19, Number 4) in the journal "Design Studies" provides some thorough discussions on this matter. It reflects that the very primitive approach to modelling design in 2D sketching is still a very effective one. There are some computer-based supporting tools for sketching, such as Jenkins and Martin's (1993).

There are studies in creative conceptual design (Navinchandra, 1991) (Sekimoto and Ukai, 1994). At the conceptual design stage, a vast among of ideas and inspirations are processed, extracted and captured. This requires an effective and efficient way to represent the concept of the design. Speed and correctness of the modelling process are crucial. Sketching is one method that can convey an idea in a 2D visual form that not only captures the thinking of the designer, but also further inspires others involved in the design process.

Traditionally before the actual production of a design, the final designed product will be modelled. It typically includes textual description, 2D drawing and/or physical 3D prototype modelling. Even nowadays, most Computer Aided Design (CAD) tools developed still use simple 3D geometrical modellers. All these are not exactly modelling the actual design process, but the design product itself instead, i.e., the final outcome of process. It is understandable that most of the works done are related to this outcome, as the final product is the least abstract and most tangible outcome of the whole design process that is directly applicable to solving the original problem. To develop a model that represents the whole design process is comparatively much more of a difficult task.

There are some models representing design process. Blessing (1994), Bliek (1995) and Tomiyama (1995) give a comprehensive study in process-based design. Quality Function Deployment (QFD) is a proper design process model, which provides a structured framework to translate the 'voice of the customer' into the actions and resource commitments needed to meet customer expectations (Menon et al., 1994). The model maps the customer requirements into specific design features (and eventually into manufacturing processes) through one or more matrices of expectations and fulfilment options.

Based on QFD, a Design Function Deployment (DFD) (Shahin and Sivaloganathan, 1998; Shahin et al., 1998) was also proposed. It is one approach in modelling design process, which is structured in a hierarchical manner. These models mainly provide guidance, management, and documentation of design processes.

2.3.2 Computer Tools Supporting Design

There have been new methods and approaches to assist designing work, such as feature-based and parametric modelling methods that are two common contemporary approaches used in CAD systems (Andrews and Sivaloganathan, 1998). Although computers and computation techniques have been used in Computer-Aided Design (CAD) for decades, their applications are mainly limited in computerized representations of design product models. Particularly computer support in the conceptual stage of designing is still in a very preliminary stage. In design automation, the current CAD software tools available cannot provide sophisticated assistance in solving design problems.

Although with the great advance of the computing power and different computational techniques to solve different engineering problems in recent decades, it is still a very difficult task to develop computer aided systems that

provide a real design environment. There were some research works on Computer Aided Conceptual Design in recent years. Some of them emphasize on the importance of free-hand sketching in conceptual design (Jenkins and Martin, 1993) and some on developing computational methods to assist specific design problems, such as surface modeller (Van Dijk, 1994). However it is still in a very premature stage because of the difficulty of understanding the conceptual design process and formulating any computational methods to assist designing at this conceptual stage.

There have been many intelligent, integrated CAD systems developed in recent years. In these computer-based design support systems, many studies investigate the integration of intelligent computational methods and design models to optimize and search a design solution. Knowledge-based design is one of the popular directions in this aspect and is presented in many articles (Rodgers, 1998; Tang and Wallance, 1997). Another research (Yoshioka et al., 1993) categorizes designs into design object knowledge and design process knowledge, and proposes a framework in which a computable design process model navigates to generate and modify design models.

2.4 Summary

Based on the nature of design and the problem of developing computational models for computer-based design support, two main criteria in developing computational models for supporting design can be identified. The first one is a systematic structure that reflects the progressively evolving nature of design process, from a more abstract level to a less one. The second is the adaptive and explorative ability of the model, which is an essential element in creative conceptual design. The model should also provide mechanisms for designer interface and future enhancement.

There are many process-based design representations. However, they do not provide evolutionary mechanism, which supports the evolution of the design process. There is a need to develop the design process model in a computational form, which has engines or mechanisms for evolving the model.

There is little work in developing computational techniques for the generation of creative conceptual designs. Most of the conceptual design studies are still immature. The main problem of developing computational models for conceptual design is the unclear problem domain, which must be refined, modified or even redefined during the evolving process of design. Furthermore the generation of conceptual design alternatives in computer-based supporting systems is still very limited. The explorative and adaptive ability is essential for this creativity. It requires further studies and developments of creative design modules for improving the systems. The next chapter further reviews the research works on generative and evolutionary techniques for supporting design, on which this thesis focuses.

Generative and Evolutionary Techniques for Design

When the design process is realized as a problem solving process, the final ideal goal will be a solution (optimal if possible) that can solve the given problem. In this case, the design process can then be related to searching techniques that seek the best solution in the solution domain.

3.1 Evolutionary Techniques for Design

There are many searching techniques developed for finding suitable solutions for a problem. The very fundamental one is brute force, which exhaustively seeks all possible solutions in the searching domain. However, it becomes impossible to be achieved when the problem is too complex and the domain is too large or infinite. Therefore, there are more sophisticated approaches to work in this complex or large domain. Evolutionary Computation (EC) is one of them.

3.1.1 Evolutionary Computation

Evolutionary Computation (EC) is an approach based on mimicking the natural evolutionary process for survival. EC is one of the soft computing techniques. Together with Neural Network and Fuzzy Logic, they form the foundation of knowledge-based systems (Raton, 1999). EC conventionally involves Evolutionary Algorithm (EA), Evolutionary Strategy (ES), Genetic Algorithm (GA) and Genetic Program (GP). All these techniques mimic the natural evolution of real life. Although there are some differences among their mechanisms of mutation and crossover reproduction, all involve a set of evolutionary solutions (evolving population) based on preferential selection of the

fittest in an environment (objective function). There are many articles and materials introducing the working principles and applications of EC (Back, 1996; Eiben, 1996; Fogel, 1995; Michalewicz et al., 1996). Figure 3.1 illustrates a typical operation of Genetic Algorithms, in which a population of *k* candidates evolves from generation *n* to *n+1* through the iterative process of selection, crossover and mutation.

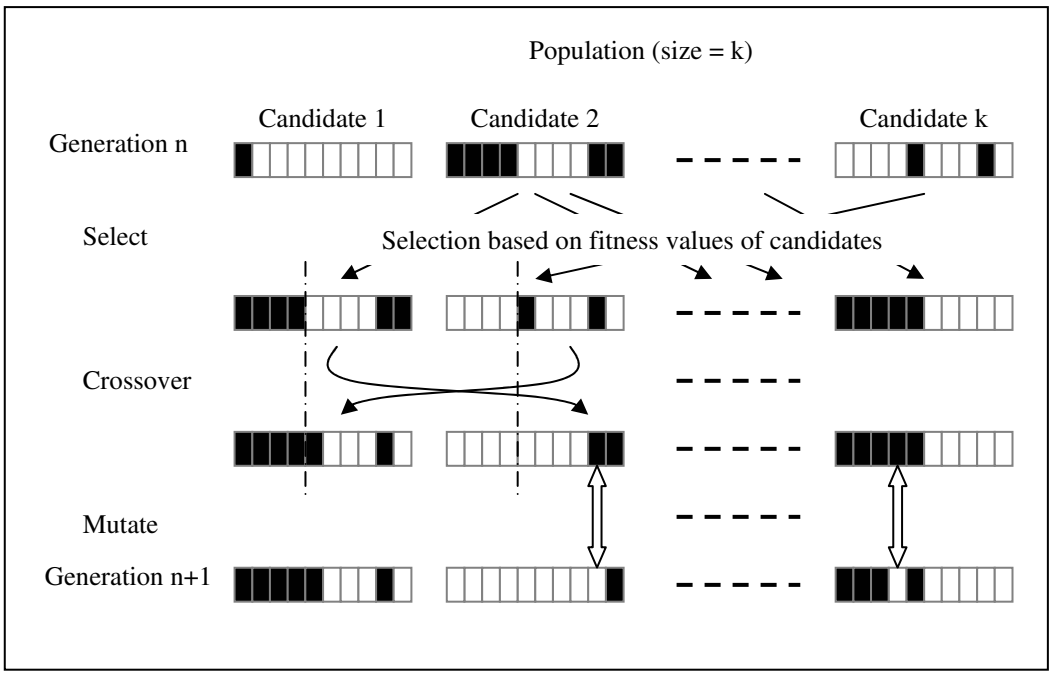


Figure 3.1: A typical operation of Genetic Algorithms (GA)

3.1.2 Evolutionary Design

EC techniques have been applied to solve searching and optimization problems in various engineering fields, such as packing optimization (Jain and Gea, 1998; Khuri et al., 1995), spare parts allocation optimization (Lee et. al., 2008), optimization of manufacturing cell (Dimopoulos, 2006), and optimization of manufacturing systems (Youssef and ElMaraghy, 2006). However, their applications in design areas are still at a very preliminary stage.

Frazer is one of the first who applied evolutionary techniques in design, particularly in architectural and structural designs (Frazer, 1995). Many new

evolutionary design methods have been developed (Bentley, 1999). Some have applied evolutionary computation techniques in artistic design and form design (Rowbottom, 1999; Todd and Latham, 1999; Witbrock and Neil-Reilly, 1999). Sims is one of the first who applied GA in graphic design and designing artificial creatures (Sims, 1991; Sims, 1994). There are many works on applying GA and other EC techniques in form generation for architectural and structural design areas (Ceccato, 1999; Kicinger et. al., 2005; Janssen, 2005). In fact evolutionary techniques have been applied to numerous design application areas, including online auction (Cliff, 2003), product design and manufacturing systems (Pierreval et. al, 2003; Sun et. al., 2007), and robot control (Kondo, 2007).

Other studies concentrate on other issues, such as proposing methods to explore the possible design domain in engineering areas (Gero et. al., 1997; Graf, 1995; Poon and Maher, 1996), enhancing evolutionary techniques (Gong et. al., 2008; Mühlenbein et. al, 2007), improving design navigation (Chien and Flemming, 2002; Gu, 2006), tackling multi-objective issues (Lee et. al., 2008; Limbourg and Kochs, 2008; Liu et. al., 2005), proposing multi-agent or parallel computing approaches (He et. al., 2007; Liu and Tang, 2006), and even forming hybrid systems with other techniques (Nariman-Zadeh et. al., 2005; Pahl, 2004; Park et. al., 2007).

Many terms have been used to describe the way in which design is created with additional merits such as creative design, innovative design, evolutionary design and generative design. In this research, generative design is taken as a process that differentiates itself from other terms of design by the use of evolutionary algorithms as a way to generate, evaluate and select design solutions.

In the next section, an attempt is made to explain the basic scope and methodology of generative design with a reference to the context of this research which develops a generic software kernel for supporting it. Without controversy and having to define and generalize what generative design is in its most generic

terminology, the concern of this research is more on the understanding of how computational design techniques such as genetic algorithms and other evolutionary algorithms can be utilised in such a process.

3.2 Computational Techniques for Generative Design

When a design is considered creative, it means that the outcome of the design gives surprises to the users who interact with the functionality of the product. Generative design can be considered creative since it produces many unexpected design outcomes together with their alternatives. The outcome is achieved in an uncertain, uncommon and unexpected way that involves computations of many iterations, but in the meantime without failing the originally intended goal.

There are research works that focus on supporting generative design in various aspects, including the issues on providing aids for generative design to novice designers (Chase, 2005), and for supporting design applications in structural and architectural areas (Caldas, 2008; Fischer et. al., 2005; Janssen, 2005; Shea and Gourtovaia, 2005). Many of these generative design works are closely related to evolutionary design so that the generative, explorative and adaptive abilities of these approaches can be integrated together to support the fundamental properties in design.

In Bentley's book "Evolutionary Design by Computer" (1999), he defines Generative Evolutionary Design as "*The use of evolutionary algorithms to generate new designs from scratch*". Analogously, generative technique is the technique that generates new designs from scratch. In Bentley's book, some introductions and discussions are given in defining evolutionary and generative design from a more technical aspect. Since the matter of defining whether a design solution is creative and generative or not is a much of a subjective topic, most researchers in computational design mainly investigate and study generative design focusing on the aspect of design processes or design techniques. Bentley's

book concentrates on applying evolutionary techniques to design applications. In his view, generative evolutionary design is designing with generative evolutionary techniques. In particular, the relationship between genotype (the internal representation of design inside the genetic algorithms) and phenotype (the design which the genotype represents in the application domain) is identified as one of the key issues. He believed that generative and evolutionary techniques must work on the final form of design rather than on some high-level representations:

> *"Using computers to generate the form of designs rather than a collection of pre-defined high-level concepts has the advantage of giving greater freedom to the computer."* (p.40)

This reflects the importance of relaxing the constraints which are pre-defined prematurely, and giving much flexibility and diversity to the possible solutions that are being generated. Such flexibility and diversity can be better obtained in development processes instead of simple mappings from the problem domain directly to that of the final solutions. Therefore giving computational support to a generative design process has the potential of generating new designs which are not well defined before the process starts. The process itself evolves the solution from an abstract concept to detailed configurations.

Without further elaborating on how to judge whether a design is creative, generative and innovative or not, this study concentrates much on design by providing generative capability to the design process and examines such an approach through a generic kernel of supporting systems. The main focus in this research is to formulate the design process as a generative one with the support of evolutionary computing techniques. A generic framework of generative design can be closely examined more in the context of comparing it with those systems without such generative capability.

Generative computational techniques can be used to *develop* the design object, instead of only alter the basic preset parameters of the object in the way in which many other non-generative techniques do. In other words, generative technique can be realized as an approach to support design that develops a design from one form to another, leading to a final solution. Shape grammar and L-systems which will be discussed below can be regarded as two examples of these generative techniques, while pure searching and optimisation techniques such as simulated annealing are not.

For example, if we have a 3D model of a chair, which is formed by its geometrical information (the legs, arms, back and seat) and the attributes (say, the material and colour), a technique will not be generative if it merely produces a new chair in form of a 3D geometry having the same geometrical information but different attributes based on the same chair template. However, if a technique produces chairs in a way that is based on building chairs with lower primitives of the chair template, the technique will have some degree of generative ability. The generated chairs may be structurally different from the original template, perhaps with the legs and seat upside down or with new attributes.

Although there are many computational techniques which support design process generatively, much attention has been given to those techniques which produce simulated life forms in our natural world. These techniques often operate in a recursive and iterative manner and can generate different life-like patterns or form structures. They are closely related to evolutionary development and repetition of natural lives. The following subsection introduces some of these techniques, which will later be examined again as the possible inference mechanisms to be used in the generative design kernel developed in this thesis.

Many current interests are in those generative techniques for form design which mimic the behaviour of nature. These techniques attempt to mimic natural form

in a computational environment. In particular, many of these techniques often operate in an evolutionary manner with iterative and recursive processes.

3.2.1 Iterative and Recursive Development Process

Although there are many methods claimed to be generative in some sense, most computational generative techniques exhibit an iterative and recursive way of operation, i.e., the way that is closely related to evolution. The techniques to be discussed in this section are in this vein, and most of them have been used to support evolutionary design. Although iterative and recursive processes can operate independently, many generative techniques combine both.

There may be different interpretations for iteration and recursion. In programming, recursion operates in a self-calling manner. During the execution of a recursive function or method, another instance of the same recursive function is called and executed, before the execution of the calling instance has finished. Iterative process operates in a similar manner as the recursive process. However, instead of calling another instance of the function during the calling instance, each iterative or repetitive cycle of the iterative process will only be called after the finish of another one.

The typical type of iteration process in programming is often in a loop form. The typical iterations of programming languages are the while-iteration or for-iteration. The flow diagram in figure 3.2 shows the basic operation of an iterative loop. The pseudo-code below shows an example of a loop, which iteratively performs the Loop-Process from *0* to the *loopLength*:

```
From loop = 0 to loop = loopLength, if condition OK
    Loop-Process-Once( )
```

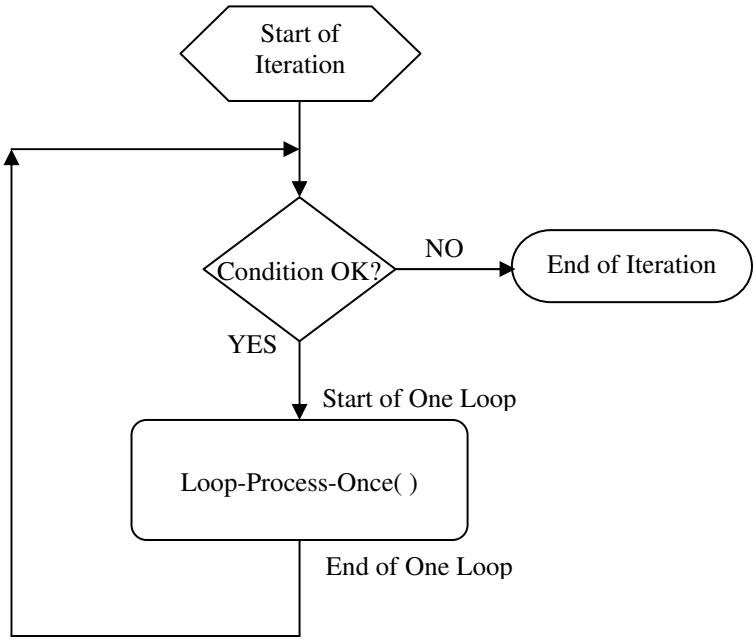


Figure 3.2: A typical iterative process.

The recursive processes are often implemented with a recursive or "self-calling" function. Recursive process may be better realized with the pseudo-code shown below, while the flow diagram in figure 3.3 shows the basic operation of this recursive process.

```
Recursive_Opn( )
        Do_Something( )
        If (Another_Recursion_Condition = OK)
                Recursive_Opn( )
        Else
                Exit_Recursion
```

With the two pseudo-code programs of the iterative and recursive function above, a combination of these two can be easily formed. The simple iterative-recursive codes in figure 3.4 perform the recursive operation ***Recursive_Opn( )*** once, from the first loop *0* to the last one *loopLength*, as shown in the flow diagram in the figure.

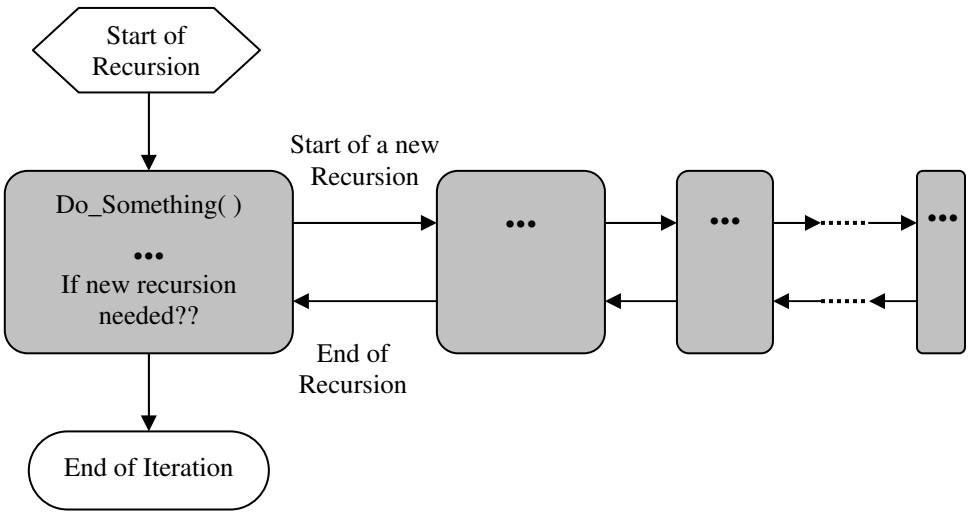


Figure 3.3: A typical recursive process.

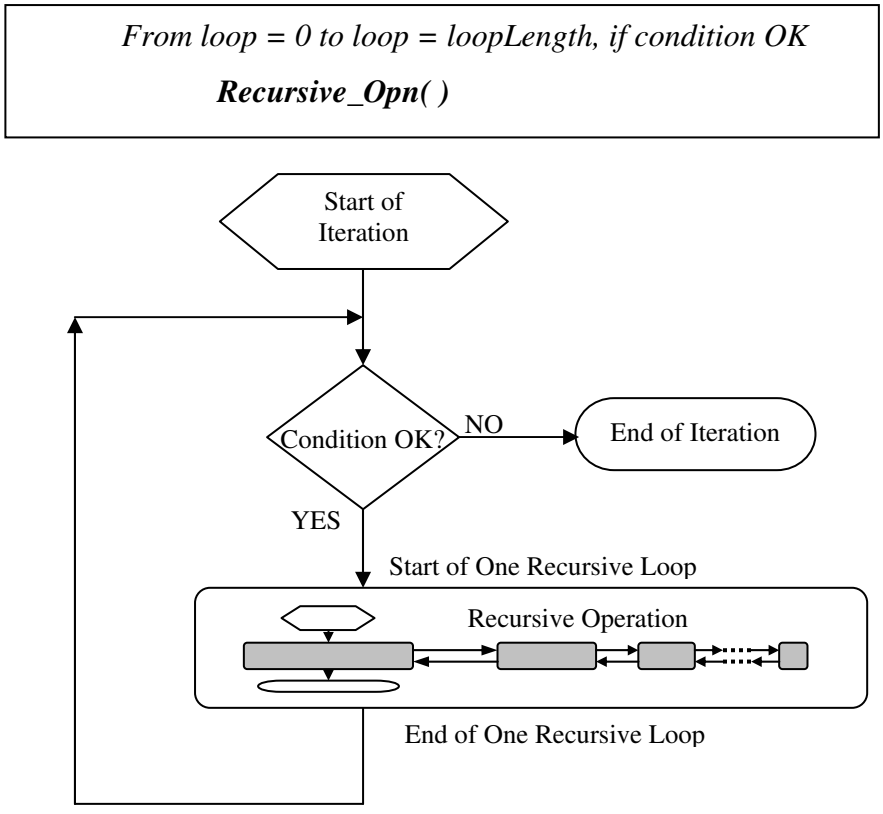


Figure 3.4: A simple combined iterative-recursive process.

Although these two processes are seemingly simple, many of the generative techniques employ both for producing interesting varieties of patterns of the same structural configurations. Replications with similarity or symmetry of forms and patterns are often produced by using these processes as the evolving and developing mechanisms which work on data structures and allow the substitution of different value ranges for the variables and constraints encoded in the problem solution space. However, the results generated by these processes often seem perceptually irregular, complex and chaotic, even if they are generated with very simple generative techniques. The following introduces the basic concept of two common generative techniques known to various design fields: Shape Grammar, and L-System. These two techniques form the basis of self-replicating evolutionary elements in the proposed computational kernel in this research.

3.2.2 Shape Grammar

There are computational techniques, which can produce similar self-symmetric forms. Shape grammar is one of them. Shape grammar was proposed by Stiny and Gips (1972). It involves a set of rules that generates shapes in a stepwise manner, with the results ranging from primitive shape forms to more complex ones.

Shape grammar can be realized as a specific type of formal grammars, which is related to the very origin of Chomsky' grammars. Chomsky, a scholar specializing in linguistics, presented a model for characterizing natural languages, called generative grammar, and produced a commonly-used definition of grammar, which is the vocabulary of symbols or words, together with a set of rules that specify how elements in the vocabulary may be combined to form strings of symbols, or sentences, in a language (Knight, 1994).

In Stiny and Gips' original shape grammar, there are four main elements:

a) A set of primitive shapes, ***p***, (and a set of terminal shapes ***T*** can then be derived from ***p***, which is a finite arrangement (sequence) of scaled-and-orientated ***p***).
b) A set of variable shapes, ***v***, which is disjoint with the shape terminals.
c) Shape rules, ***Rs***, an order pair having the first element *(t, v)* and the second element is 1) (*t*), 2) (*t, v'*), or 3) (*t, t', v'); t, t'*$\in$ ***T*** and *v, v'*$\in$ ***v***
d) Initial shape, ***I***, a combination of elements in ***T*** or ***v***.

The process of shape generation using shape grammar can be realized as the development of a shape object with an initial shape (a combination of terminal and variable shapes), and the elements in the initial shape are then mapped to other shapes with the grammar rules. This process may continue until the generated shape consists of primitive shapes only. The terminal elements in a given shape will be preserved with the shape rules that always transform a shape with the terminals to other shapes having the same terminals. In a design situation, as the design objects in shape grammar are often represented in geometrical forms, common geometrical transformations such as scaling and rotating are used in the shape rules. While shape grammar can be formally explained with mathematical notations, Knight's book (1994) gives a better descriptive explanation of shape grammar and is a good reference for people who may feel uneasy to those strange mathematical symbols.

3.2.3 L-System

L-System is named after A. Lindenmayer. In 1968, Lindenmayer introduced the L-System, a parallel rewriting system, to simulate the development process of natural lives (Lindenmayer, 1968). While L-System can be regarded as formal grammar and language systems as shape grammar, the system is often used for modelling the developmental process of natural lives and produces them in a fractal way (Flake, 1998). Although there are many variations of L-system

suitable for different applications, the most basic form of it consists of only three elements:

a) a set of primitives, constants or alphabets in formal language,

b) the set of all possible components, variables, or words in formal language, associated with the primitives, and

c) a set of rewriting rules, that maps every possible component to another.

To simulate an object with a specific L-system, one starts by giving the object in its most infancy stage with a specific component that can produce different components at different temporal frames by recursively applying the rewriting rules to the components. With its formal basis on syntactic rules and symbols, L-system has the same root of shape grammar in formal grammar and language. It is not surprising that both L-systems and shape grammars operate in a similar manner, and may have a similar system structure in an actual implementation.

3.2.3.1 Deterministic and Non-deterministic Generation

With these two generative techniques (Shape Grammar and L-System) discussed above, many interesting patterns can be generated. Figure 3.5 shows a fractal pattern generated with a simple recursive-and-iterative affine transformation. The development rules for the pattern are preset and the pattern grows systematically from the left, with the simplest vertical line, to the right after 8 generations. As the recursion, iteration and affine transformation are preset, the pattern is deterministically produced.

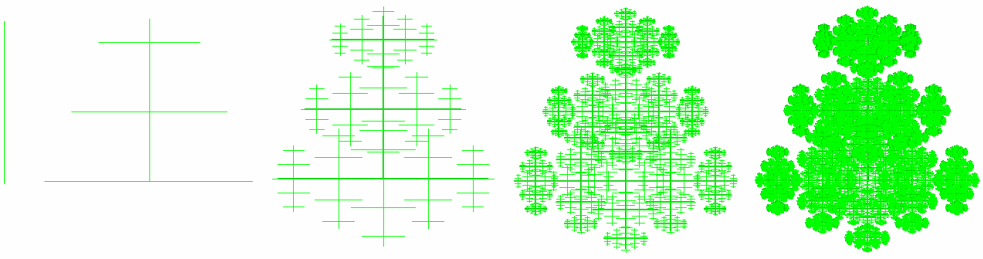


Figure 3.5: A deterministic pattern generated with iterative-recursive affine transform.

To simulate natural lives, some careful but simple thought may produce a much better effect. Plants are sometimes used as examples to illustrate the power of these generative techniques. A simple alternation of the setting may produce much better effect, even deterministically. For example, when one considers the branches of plants in our natural world will most likely grow in some upward directions, a little change of a preset parameter may then lead to producing much better result. The generation of the plant-like patterns in figure 3.6 is based on the simple system to produce figure 3.5, except that the degree of rotation in the affine transformation for figure 3.5 is 90 degree while it is 30 degree for figure 3.6.

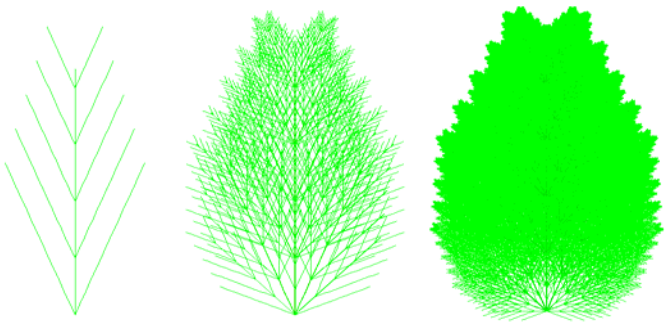


Figure 3.6: Another deterministic pattern that is more plant-like.

A little change may lead to more desired results. However, in a natural world absolute regularity is rare. To produce more realistic natural forms, adding some randomness to the generative systems can further enhance the ability of it in modelling our nature in a non-deterministic way. Even a very simple form of randomness can lead to a higher complex matter. In practical programming, pseudo-randomness is often applied instead. When the pseudo-random generator is added to the same programming, the results shown in figure 3.7 can be generated, with straight line as the basis entity.

When human users interact with a generative system, much realistic life forms can be produced. Figure 3.8 shows two plants generated with the same generative mechanism as the ones above. Instead of using straight lines as their basic

elements for breeding, users sketch their preferred curves (shown on the left most pictures). Similarly, the pictures show how two plants grow correspondingly after 2, 4, and 6 generations.

Figure 3.7: A pseudo-random plant-like pattern generated with straight lines.

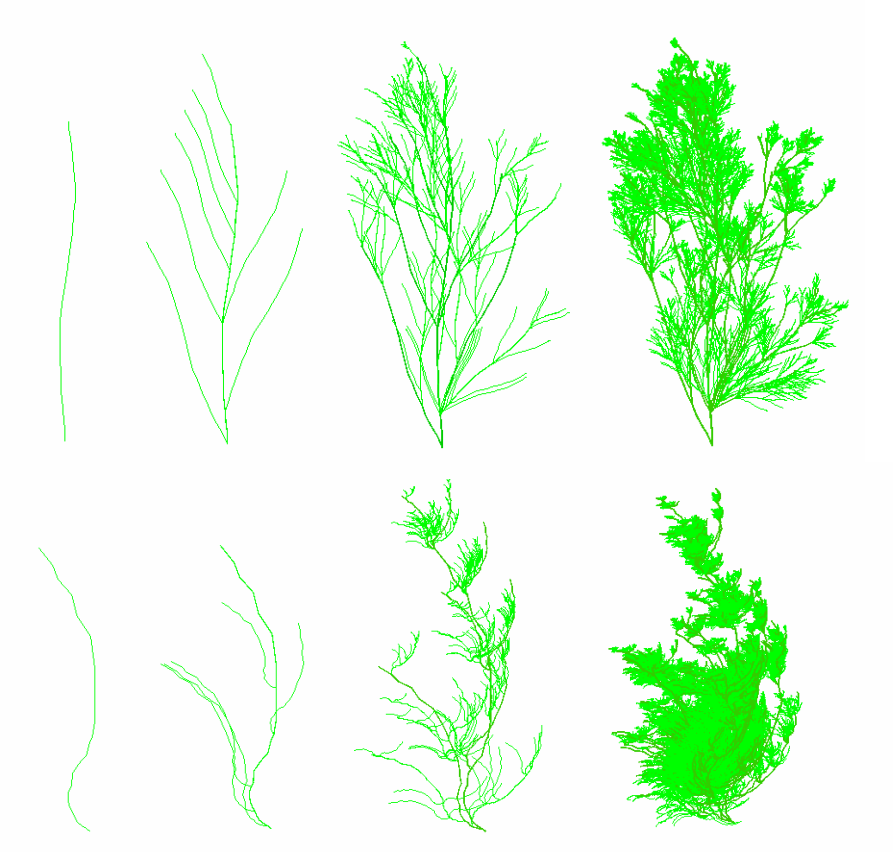


Figure 3.8: Two patterns generated with sketching curves as the basic breeding elements.

Another generative technique called Cellular Automata has been applied to many design areas. In particular, various studies of applying CA to urban planning and simulation have been done, including those works presented in a special issue in the journal "Environment and Planning B" in 1997 (Batty et al., 1997). Since the original CA is regarded as rather limited and restrictive, there are many modifications of CA for various applications (Herr and Kvan, 2007; O'Sullivan, 2001).

3.2.4 Cellular Automata

Many researchers have been connected to the development of Cellular Automaton (CA), but undoubtedly von Neumann is the most cited one. CA was proposed by John von Neumann in the late 1940's (von Neumann, 1966). Von Neumann was interested in self-organizing and reproducing automata which includes associated theories to construct large computers with certain degrees of complexity. Aspray and Burk's book (1987) is a good reference for understanding more about von Neumann's research, including the theory of CA. From a practical point of view, extensive interests were not initiated until the famous CA, Conway's Game of Life, which is a CA that best illustrates the ability of connected "simple" autonomous elements which produce "complex" emergent behaviours or patterns.

Cellular Automaton (CA) is closely related to the studies of complexity, self-organization, emergent pattern, artificial life and adaptive complex systems. Instead of giving a rigorous mathematical formulation and formal explanation of CA that other references have already presented well, this section introduces its basic concept and operating principles which will be utilized in the software kernel developed in this research.

3.2.4.1 Structure and behaviour

There are two basic properties of CA, its static spatial structure and the dynamic temporal state behaviour. CA is formed with a static structure. It consists of a lattice of cellular elements (cells) located in a discrete space with a homogenous neighbourhood relationship.

- Cellular elements (cells):

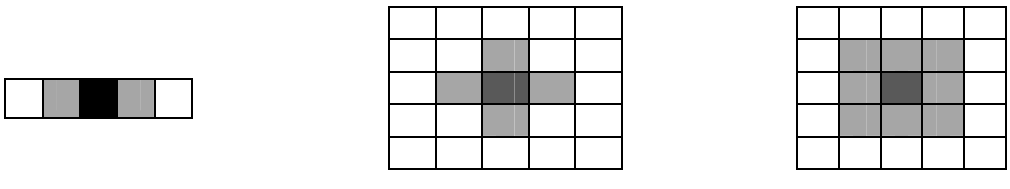
CA consists of a collection of elements, or cells, that have "static locations" but "dynamic states".

- Regular discrete space:

Each cell is located in a unique point at a regular discrete space of n-dimension. In case of 1-dimension, the cells can be realized as the elements in a sequence, while in case of 2-dimension, the cells live in a 2-D grid.

- Neighbourhood:

As the cells are positioned in a regular space, a set of neighbour cells corresponding to each cell (often a central cell) can then be identified. In most cases, these neighbour cells include cells that locate within a distance from the central cell in the space. Figure 3.9 (a) shows the immediate 2 neighbours (in grey colour) of the central cell (in black). In a 2-D CA, two most commonly used neighbourhoods are 4-connected (or von Neumann) neighbourhood and 8-connected (or Moore) neighbourhood, as shown in figure 3.9 (b) and (c).



(a) (b) (c)

Figure 3.9: Three different CA neighbourhoods: (a) 2 neighbour cells in 1D CA, (b) 4-connected neighbours in 2D CA and (c) 8-connected neighbourhoods in 2D CA.

Although CA is structured with spatially static elements, the dynamic state behaviour of these elements often leads to produce seemingly complex behaviours with the properties of their synchronized transition and a universal transition rule.

- Dynamic state:

All cells of a CA are in one of the states in a set of finite states, at any specific discrete time.

- Synchronized transition:

All cells in a CA have the same transition rule, and the transition of the state of all cells is synchronously activated in a discrete time interval.

- Locally universal transition rule:

The same transition rule is applied to every cell in a CA, based on the states of the cell and the neighbour cells. Therefore the transition of a cell is based regionally on the states of its neighbours, and its own.

From a computational point of view, Cellular Automata (CA) is a specific computational model with a very simple autonomous mechanism. Much attention is given to this simple mechanism applied locally that seemingly leads to produce unexpected complex global behaviours or patterns. Figure 3.10 shows the operating principle of a typical 1-dimensional binary CA. The CA in the figure has a size of nine (the number of cells) and each binary-cell can have a state of either one (black in colour) or zero (white in colour), while the neighbourhood of each cell is limited to its nearest neighbours. Thus the transition rules only concern the direct two neighbours (left and right) of a cell (middle). At any generation, say *n* as shown at the top row of the figure, each cell of the CA will be mapped to a new state according to the current state of its own and those of its neighbours.

For example, the current state of the second leftmost cell is one (black) while those of its neighbours (both left and right) are zero. Following the arrow under this second leftmost cell leads to the third transition rule, which maps any cells having the state pattern of "zero-one-zero" to a state of zero. Therefore the new state of the second leftmost cell in the generation *n+1* is zero. This transition mapping process is applied to each cell of the CA. The figure shows the mapping (arrows) of only five outmost cells in order to avoid the confusion of arrows. A typical CA has boundaries (finite) and the boundary cells require special handling as they have fewer neighbours. A circular approach is used in the figure, that the two boundary ends are treated as circularly connected such that the leftmost cell becomes the right neighbour of the rightmost cells while the rightmost become the left neighbour of the leftmost. In this case, the same mapping process can be applied to the boundary cells.

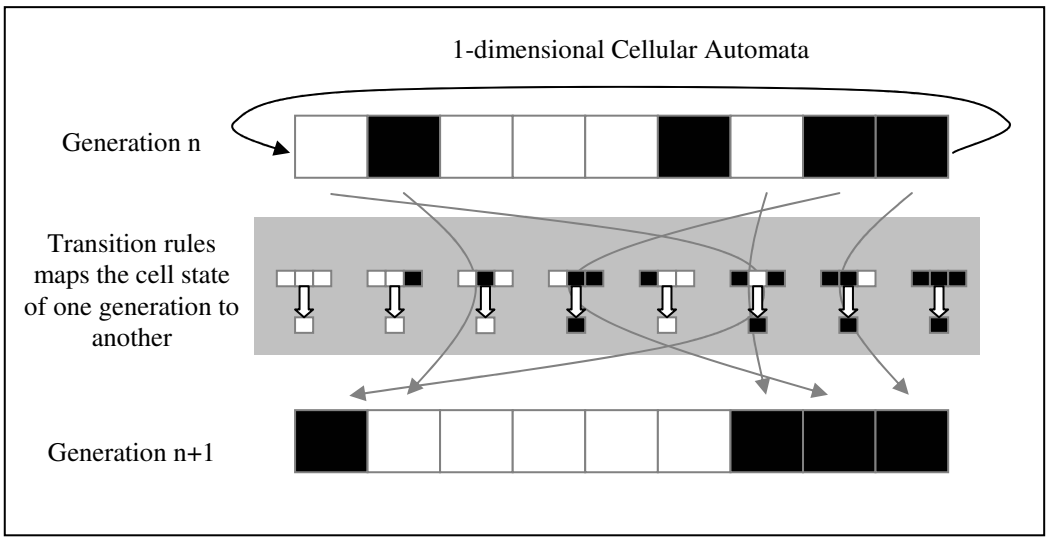


Figure 3.10: The basic operation of CA in two consecutive generations.

When the number of possible states of the cells increases even with very simple transition rules, some complex behaviours or patterns can be obtained. Figure 3.11 shows the result of such a CA. The 1-dimensional CA that produces this pattern has 256 states. This CA starts with one non-zero cell in the middle of the 1D array, at the top of the image. The transition rule is simply based on mapping the sum of the regionally effective cells (the 2-neighbour cells and the central cell)

to the same value (with special handling in overflow cases) as the next state of the central cell. The states of the CA cells are then visualized with a mapping to corresponding RGB (red, green and blue) colours, according the some threshold levels.

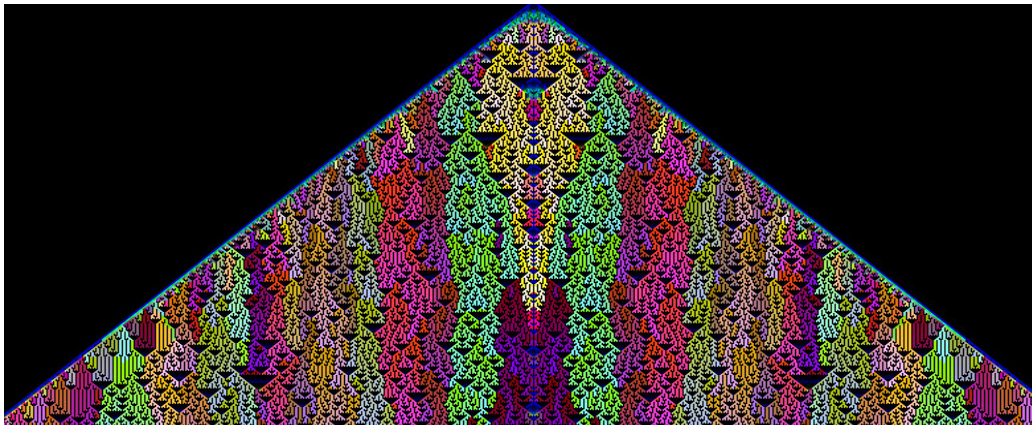


Figure 3.11: A pattern generated by a simple1-D multi-state CA, from earlier generation (top) to later generation (bottom).

3.2.4.2 Totalistic and semi-totalistic

As shown in the earlier figure 3.10, the number of different state patterns of 3 binary-cells (the central cell with 2 neighbours) is 2^3=8. The number of possible transition rules is then 2^8=256. In fact, given that the number of states of a CA is *s*, and the number of locally-effective cells (including the neighbour cells and the central one) is *k*, the number of total possible transition rules is *s* to the power of s^k. When the number of states of the CA is increased, the number of possible transition rules will be increased dramatically.

Many studies restrain the investigation to a much simple type of transition rules. In fact, the colour pattern shown in Figure 3.11 is produced by a 1D multi-nary CA, which falls in a specific category of CA, a totalistic CA. Totalistic CA have transition rules that can be realized as density functions that the next state of a cell is solely dependent on the density of its local region, and thus the sum of the state of its neighbourhoods. Furthermore, if the transition rules lead the next state of a cell to a new state that depends not only on the sum of the states of neighbouring

cells but also on the state of its own, such CA and their rules are called semi-totalistic.

The well-known Conway's Game of Life CA belongs to this type. Conway's Game of Life is a semi-totalistic binary (2-state) CA, working on a 2D space with an 8-connected neighbourhood. The transition rule is simple, while the emergent behaviour produced is unexpectedly complicated. Because of this, this CA particularly attracts much attention in various research fields and studies. The transition rule of this CA is as following.

a) An "off" cell will become "on" in the next state, if and only if 3 of its neighbours are "on".
b) An "on" cell will become "off" in the next state, if more than 3 of its neighbours are "on", or fewer than 2 of its neighbours are "on".
c) In all other cases, the cell will remain the same in the next state.

This transition rule is also often presented in a way of living systems, for a better understanding of its emergent behaviour (the "Game of Life"), as below:

a) A new cell will be "born" when 3 of its neighbours are "alive",
b) A cell will die because of "overcrowded" (more than 3 neighbours are alive) or "isolated" (fewer than 2 alive neighbours).

The following figure 3.12 shows some of the emergent patterns produced by this CA. Be noted that the seemingly "life" patterns are mainly realized on the illusion of the temporal motion (from left to right) of lives (formed by groups of cells); while in facts all the cells of the CA themselves are stationed statically in their own positions.

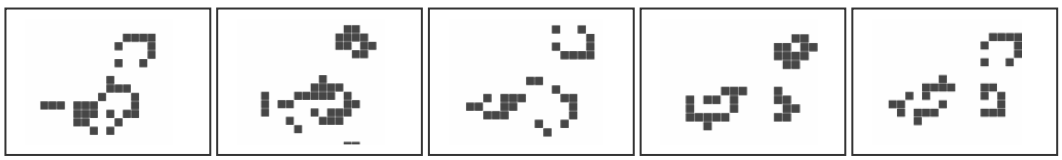


Figure 3.12: A series of "lives" produced in the "Game of Life" in consecutive generations, from left to right.

3.2.4.3 Study of emergent behaviour and computation

While many totalistic and semi-totalistic CA are studied in many research works, another challenging task is associated with applying CA in distributed computation and studying the underneath emergent behaviour. If Conway's Game of Life is the most well-known CA, design-classification problem is possibly the most well-known example of emergent computation of CA being studied.

Given a list of binary bits with 0 or 1, the computation of the total number of 0's and that of 1's in the list can be done with very simple programs in diverse methods. This seemingly easy task becomes an extremely difficult one when the computation is changed from the central approach with "global vision", to a distributed one that the computation is relied on the locally autonomous entities that only have limited information of their local regions.

In the design-classification problem solved by CA, the goal is to seek the right transition rule(s) that can be applied to every cell of the CA for leading to obtain a desired outcome that either (a) the states of all cells will converge to 1 if the majority of the cells have 1 as their very initial state, or (b) the states of all cells will converge to 0 otherwise. To avoid further complexity, the length of the list is often restricted to odd numbers. The interest of this problem solved with CA falls in two aspects: seeking the right or best transition rules to solve the problem and studying the emergent properties of different transition rules that leads to the right solutions. Further details of recent study in density-classification problem can be found in (Das et al., 1995; Ferreira, 2001).

3.3 Summary

In this chapter, some evolutionary and generative computational techniques for generative design in evolutionary approach are presented. These computational

techniques can be used as the fundamental mechanisms for exploration, adaptation and generation of potential design candidates. Further discussion on issues related to design generation and exploration for generalizing a generative and evolutionary design model will be given in next chapter. Design knowledge adaptation and reconstruction in such a generative and evolutionary approach will be discussed. The theoretic foundation of such a kernel and its technical innovation for implementation in real design applications will also be presented in the later chapters in Part II.

Part II:

A Generative and Evolutionary Design Model

Issues in Modelling Generative and Evolutionary Design

The computational Generative and Evolutionary Design (GED) kernel proposed in this study is intended for modelling design in a generation and evolutionary approach. In this chapter, issues on design generation and exploration are further discussed. Design knowledge adaptation and reconstruction in such a generative and evolutionary approach are also presented, as well as the importance of multiple representations of design in such an approach.

4.1 Design Generation and Exploration

Gero has categorized design into three types: routine, innovative and creative design (Gero, 1990). This approach is closely related to the state-space searching perspective in problem solving (Newell and Simon, 1972). Some researchers argued that general design activity should not be treated as a process of pure searching problems and their corresponding solutions (Janssen et al., 2002), as this approach tends to over-rely on searching within a static set of parameters.

Instead of handling design process as a pure domain searching issue, this study emphasizes on the methods to generate and explore potential designs with generative techniques. These generative techniques or mechanisms produce less abstract, more complex and detailed design objects from a more abstract, simpler and conceptual design representation. This is similar to generate a mature plant from its seed or analogous to making the phenotype from a much abstract form of genotype in an evolutionary computation.

4.1.1 Design Generation and Exploration

Supporting design generation and exploration with computational systems has been studied for a few decades, including evolutionary design approach. Some computational systems in particular improved the efficiency and accuracy in many design aspects, including analysis, geometrical modelling and design project management. However, it is still a question at what level of abstraction at the conceptual design stage the computer power could be better utilized to provide support to designers since there is no generic theory of design computation. In many cases, design applications have to be modelled as one of the search or optimisation problems at which level the design problem has already been simplified or constrained with additional limitations imposed.

In a design process, supporting the generation and exploration of potential solutions is one of the major objectives in developing computational design supporting systems. There are different works on studying how this generative and explorative ability can further be enhanced, including Hornby's work on applying a generative design representation approach to supporting design and emphasizing its scalability in design exploration (Hornby, 2003). To further enhance this exploration ability in the proposed kernel in this research, the issue of multiple representation of design is discussed below.

4.1.2 Multiple Design Representations

Many evolutionary design methods have been developed for supporting the exploration of dramatic and creative potential designs with those evolutionary techniques as discussed in (Bentley, 1999). The term "creativity" is often related to "unexpectedness", "surprise" or "new". It is also relative and subjective to specific groups of people. Different groups of observers, designers or users in different specific times or spaces would find the same design having different

degrees of creativeness. Some people may find a design very ordinary while others may find it very creative.

Design models have also been created as multiple-representation, network, layer-network or hierarchy in some studies (de Vries, 2006; Rosenman and Gero, 1999; Stouffs, 2008; Suh, 1990; Tomiyama, 1995). When attention is put further on the creativity of design, "creative leap" (Cross, 1997) or "sudden mental insight" (Akin and Akin, 1996) becomes an important factor for successful design computation, which emphasizes on the mapping from one design representation to another.

There are two perspectives in viewing multiple design representations. One is based on different kinds of representations which provide different views and each representation type captures a specific aspect and neglects others (Gero and Reffat, 1997). For example, emphasis may be given to the kind of representations related to the aspects of aesthetics, psychology, technology, structure and geometry.

The other perspective concentrates on managing design multiple representations at different degree of abstractions with specific abstraction properties, from more abstract to less abstract and more detailed levels. There are various formal theories of abstraction (Giunchiglia and Walsh, 1992; Giunchiglia et al., 1997), in particular relation to formal grammar and language, and formal models of abstraction hierarchies (Fikes and Nilsson, 1971; Knoblock, 1994; Sacerdoti, 1974).

The importance of developing multiple representations at different levels of abstraction of design problem has been discussed in recent studies (Heisserman et al., 2000; Kim and Yoon, 2005; Liu et al., 2000). However, these studies adopted a static hierarchical structure for representing design product, which restricts the flexibility of evolving design with dynamic representations that can be specialized

during the design process from a general initial concept. This study emphasizes on the multiple representations of design which allow a design to develop and evolve from an abstraction perspective. The proposed kernel tackles this problem by providing a generic representation that can be developed in a hierarchical manner with evolutionary inference algorithms serving as the transition mechanisms to transfer an abstract representation to a less abstract one.

Although abstraction can be related to different aspects, in formal theories they are often related to property generalization and refinement. In this case, abstraction is basically a grouping process of less abstract objects (or its representation) to a higher abstract one, based on certain abstraction properties. Extending from this concept, a specific representation, R1, of design objects is more abstract than another, R2, if R1 contains only the subset of the information specifying the same objects in R2. In other words, the set of all possible objects represented by R2 is a subset of design objects in R1. Based on this concept, it can be realized that more possible objects can be generated and explored in an action at a higher abstraction than at a lower one.

Apart from the issue of multiple representations, another important point for attention is related to design adaptation. While design exploration in this study emphasizes on the issues of what potential designs can be generated at different abstractions from one level to another until the process reaches the lowest level of abstraction (i.e., with the most detailed design outcome), design adaptation is concerned with why and how they can be generated. Proper adaptation requires knowledge for the reasons or goals of generating certain designs and the methods for achieving them.

To support generative and evolutionary design with multiple representations, the explorative ability of the system must be complemented with adaptive functions. While design systems explore to generate possible design solutions, they should also know how to generate the most potential ones efficiently. Without proper

adaptation, pure exploration would behave like an inefficient “blind” searching. In a blind way, random generation of weir objects are often obtained.

4.2 Knowledge Reconstruction

Design involves a process of identifying the problem, analysing it with the existing knowledge, reconstructing the existing knowledge to synthesize the potential solutions. The process is adaptive in nature as more and more features are added into the product with the introduction of new knowledge into the process. This design process is a knowledge-intensive activity, through which creative and innovative outcomes are highly desirable.

This activity of exploration and adaptation is highly dynamic since the overall domain for defining and searching the design space keeps changing as design proceeds. Gero studied this and referred it as the problem of state-space enlargement (Gero, 1996). With any newly added knowledge into the design, the overall domain is continuously modified, and the search space is enlarged or narrowed. There is need to model this adaptive process with proper data structure and control mechanisms in order to provide flexible support, particularly when highly divergent inference mechanisms such as genetic algorithms or cellular automata are employed.

There are several proposals for achieving evolution of knowledge from a cognitive and computational perspective in the AI-based design area. For example, Gero (1996) proposed process models based on notions of additive and substitutive variables resulting in additive and substitutive schemas for creative design. Two kinds of computationally supporting design approaches, discovery and learning, are also introduced in (Mukesh et al., 2001). They are related to the issue of having an explorative and adaptive ability in computer based supporting tools and systems for design applications.

There are also many studies of theories and applications of computational adaptation in different areas, in Machine Learning, which requires adaptation functionality. Different approaches were also studied in applying machine learning techniques to design. Some studies relate learning and creativity in design to transformation and evolution of knowledge from one perspective to another (Sim and Duffy, 2002; Wu and Duffy, 2002), while others to analogical reasoning (Goel, 1997). Some articles (Duffy, 1997; Grecu and Brown, 1998; Sim and Duffy, 1998) presented findings of studies on machine learning in design, and introduced the fundamental techniques developed. In AI, intelligent systems for supporting learning and creativity in design have also been introduced (Brazier and Wijngaards, 2002; Brown and Grecu, 2000).

Machine learning is typically adequate for solving analysis problems. However adaptation in design has further emphasis on the issue which is related to how to generate and explore design solutions at a knowledge level. For ill-defined problems like design, soft computing techniques (such as Genetic Algorithms and Artificial Neural Network) are often applied and the knowledge of design generation is implicitly embedded.

4.2.1 Design Adaptation

In this research, adaptation of knowledge is handled in an evolutionary and generative design process. Such evolution of design knowledge involves the knowledge of not only design data as objects but also generative process of producing these objects, powered by mechanisms or algorithms such as cellular automata, genetic algorithms or other computation methods including machine learning. In this investigation, the evolution of knowledge is related to the learning theory of Piaget's well-known child psychology theory.

Knowledge is related to understanding things (e.g. objects) in a form of representation at a certain level of abstraction. Piaget's concept on knowledge and learning is based on three issues:

- schemata,
- concepts, and
- structures.

Schemata are the actions to be taken under certain situations for achieving specific goals. Concepts are relations and abstractions (of objects, situations and actions). Structures are about the organisation of knowledge (about objects, schemata, concepts). Knowledge development process is a form of construction. In particular, Piaget proposed two forms of knowledge construction based on the concepts of assimilation and accommodation (Piaget, 1970, 1971, 1983).

In Piaget's view, most of the time we are in an equilibrating state. Our existing knowledge and experiences govern our behaviour acting upon various situations and our understanding of the world. When new situations are encountered, we then need some ways to relate them to our existing knowledge. Assimilation is the organisation of knowledge with our own logical structures or understandings. Given a new situation, it will be related to our existing knowledge in order to preserve a consistent view of the world. However, when new situations contradict the present knowledge and our understandings of the world, and cause a dis-equilibrating state of our intelligence, we have to reconstruct our knowledge in order to better accommodate the new situations. This concept is illustrated with the diagram in figure 4.1.

Relating this knowledge reconstruction approach to the domain of design an accommodation action will alter the whole solution domain. Figure 4.1 illustrates exploration and adaptation of new problems with domain knowledge. When encountering new tasks, new requirements, and new needs as the design problem evolves, intelligent evolutionary mechanisms react and attempt to make a

response. At this exploring stage, synthesis and generation of potential sub-solutions take place, with the current form of generative mechanisms within the existing design knowledge. These generative mechanisms explore potential solutions within the existing knowledge domain in an assimilation way. This assimilation exploration may be supported through evolving the internal rules and logics of an evolutionary mechanism.

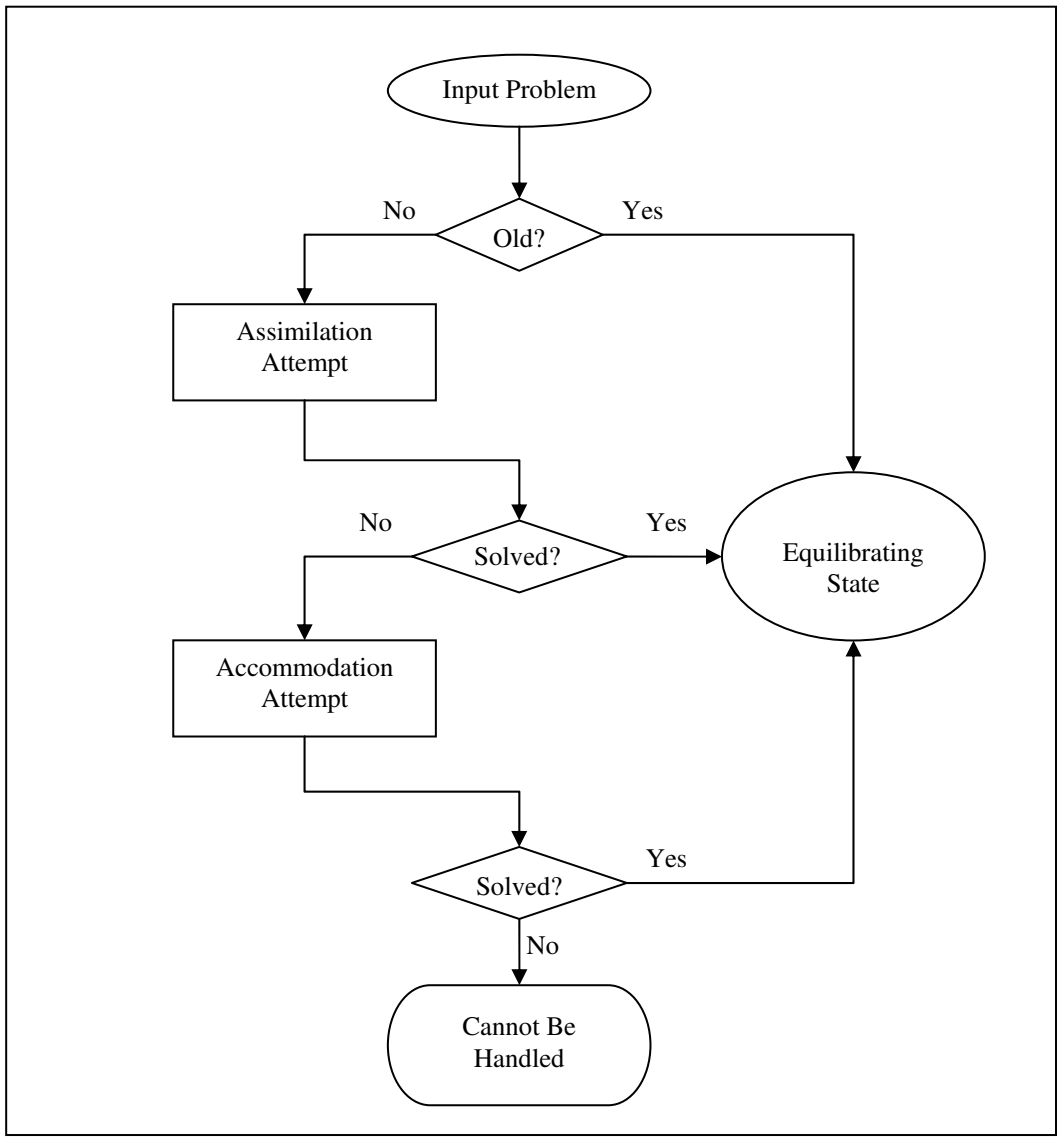


Figure 4.1: Principle of Piaget's knowledge reconstruction, with assimilation and accommodation.

When the existing knowledge is not enough for generating the right design outcomes and acceptable results, the system may have to update, modify or even reconstruct the existing knowledge. Through such knowledge reconstruction, the domain of potential solutions is modified in a way analogous to Piaget's accommodation. This accommodation can be obtained by using different approaches, including merging the knowledge represented, extending current generative mechanisms separately or reconstructing entirely known knowledge by merging generative mechanisms themselves to form a more powerful generative mechanism. In a demonstration example to be presented in the chapter 7 it will be illustrated with an example of using a simple constraint relaxation approach as a knowledge reconstruction mechanism.

Based on this perspective, the process of knowledge evolution in design can be linked to assimilation and accommodation, for producing design alternatives by the exploration of the existing knowledge and new knowledge. In a conventional evolutionary design approach, the knowledge adaptation issue is mostly handled in an assimilation way. With Genetic Algorithms as the main evolutionary mechanism the exploration is limited by the way in which the genotype is mapped to phenotype, with the adaptation being implicit in its selection process. It is a goal-oriented approach with certain degree of randomness because the genotype is generated and explored and guided towards the populations who best fit to certain preset objective functions. However, once the genotype-phenotype mapping is formulated, there is no mechanism to support knowledge reconstruction as in Piaget's accommodation. The proposed kernel in this research attempts to tackle this problem. Furthermore, this adaptation ability can be enhanced with the method of multiple representations.

4.2.2 Multiple Representations

Design knowledge has many different perspectives and can be related to design objects, design processes, interpretations and relationships of the processes and

objects (Kakihara and Sorensen, 2002). Knowledge construction has also been related to the field of organisational management. Its fundamental issues are about solving problems with a group of people, autonomous entities or modules. Some research projects modelled such a social system to a hierarchical form, as presented in the book by Burton and Obel (1995).

Compositional design approach based on process and knowledge composition was also introduced in (Brazier et al., 2001). In the paper, design knowledge is associated with the knowledge emergence of social (or organisational) systems. Some relates this problem of emergence to hierarchical complexity (Ueda, 2001). Miller's living systems (Miller, 1978), Simon's hierarchical complexity of the artefacts (Simon, 1996) and multinational enterprises (Westney, 2001) are all related to a hierarchical complexity.

In a generative and evolutionary system, design can be regarded as an evolutionary process that evolves from a conceptual stage to a detailed stage. That is, the representation of designs evolves from the most abstract level to the most detailed level through some forms of generation. During the design process, design objects are manipulated by different participants (such as designers or computational modules) at various abstractions.

The proposed computational kernel in this is based on such a generative and evolutionary design (GED) approach. It supports modelling design process as an evolutionary process of generating design objects from a much abstract form to more detailed ones at multiple representations. With this approach, design generation and exploration can be enhanced in such an evolutionary hierarchy of abstractions, and knowledge of design generation can be adapted or captured by generative mechanisms.

4.3 Summary

In this chapter, several key concepts are introduced, which will be further elaborated in the remaining chapters. These include the concept of computational design as an exploration and adaptation process in which generative and evolutionary algorithms can be employed as the mechanisms that transfer a design representation from an abstract level to a more specific level. Therefore a design object can have multiple representations, and the evolution of which with generative and evolutionary algorithms is seen as the process of finding solutions that meet the requirements at a desired level of details. These concepts form the basis for constructing a kernel that supports design exploration and adaptation, thus providing a way for elaborating design problem and identifying its solutions in an evolutionary manner. The theoretic foundation of such a kernel and its technical innovation for implementation in real design applications will be presented in the coming chapters.

A Computational Model of GED Kernel

This thesis focuses on modelling design with a dynamically evolving structure, in which evolutionary elements and evolutionary mechanisms are integrated in a hierarchical architecture. This hierarchical architecture allows the development of design solutions from abstractive concepts to detailed specifications by invoking the inference mechanisms attached to the nodes in the hierarchy, which represent the intermediate results of the solution. As the hierarchical structure is explored downwards, the problem is explored. To build such a kernel it is necessary to formulate a process model for the integration of design objects and the evolutionary mechanisms which infer on these objects.

5.1 Modelling Generative and Evolutionary Design (GED)

As discussed in previous chapter, this study develops a computational kernel for supporting generative and evolutionary design. In design abstraction is a way of synthesizing the problem with the available information at a proper level that the conceptual solutions can be developed first. The detailed solutions can then evolve from these initial concepts.

5.1.1 Generative Process as Abstraction

When a specific design task is structured within a hierarchical data structure, a change made at a higher level of the hierarchy results in more possible domain changes than that at lower abstraction. Analogy to practice of design, changes made at earlier conceptual stages with abstractive representation of design

problem formulation often generate much radical outcomes than that at a later stage.

Figure 5.1 illustrates the idea of multiple representation of design which evolves from abstractive concept to detailed solution. This abstraction structure is related to 4 design stages using Suh's axiomatic design example (Suh, 1990). In figure 5.1, the representation at a higher abstraction is represented with "conceptual 2D sketch" while "detailed 3D model" represents a less abstraction of the design. Although this cannot be fully formulated in a quantitative way, an alternation in the 2D sketch may mean that a change happens at an abstract conceptual level. This leads to a much more radical change to the final product than what could do if a change were made to the 3D solid, which derives from the sketch in the first place.

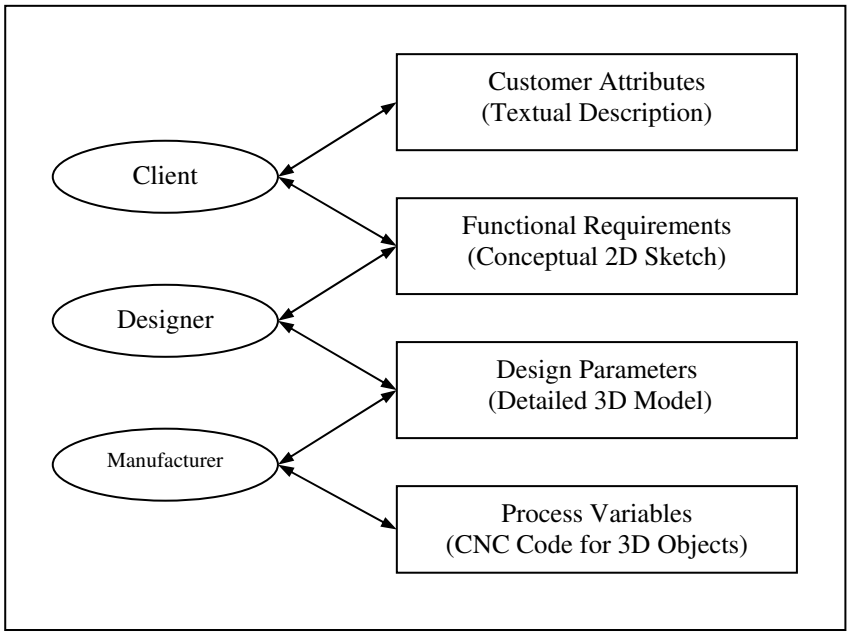


Figure 5.1: An example of manipulating design representations at different abstraction levels, at different stages by clients, designers, and manufacturer.

Furthermore, exploration of potential design solutions at different levels of abstraction can be enhanced by using generative and evolutionary mechanisms. In the process model proposed in this thesis for the implementation of a generic

kernel of generative and evolutionary design, design objects which have variables and constraints are attached with certain generative and evolutionary computation mechanisms such as genetic algorithms or cellular automata. In this process, a design concept is firstly explored at an abstractive level with fewer variables and constraints that would have at a lower level of abstraction, with evolutionary algorithms that only need to manipulate a few design variables and constraints. This generates a population of candidate solutions at that level of abstraction. These candidate solutions are then explored further, which means that any of the candidates can be selected and explored further by adding new variables and constraints to it. At this level, evolutionary algorithms need only to work on the newly added variables and constraints, thus reducing the complexity of having to work on all the variables and constraints at both levels in one go. If a change needs to be made at this level in order to explore more possibilities, in most cases this would not affect those candidate solutions at the first level.

This top town approach continues if a specified level of abstraction is reached and the solutions contain enough details for the successful evaluation of them against the original evaluation criteria, then the process can be terminated, or it is possible to backtrack to a higher level of abstraction for the exploration of more alternatives. This approach has been reported in Engineering Design (Chakrabarti and Bligh, 1996; Chakrabarti et al., 2002) in a system perform functional synthesis, but the approach was not formulated as a generic kernel with generative and evolutionary mechanisms to support a wide range of industrial design applications.

With the data being substantially different at different levels of the hierarchy, it is difficult to have a unified set of evolutionary algorithms that can manipulate any data no matter what level of abstract they are. If that is the case, it is possible to fix the inference programs to allow the data to evolve. However, if a new design problem is encountered, the whole hierarchical structure would have to be reconstructed and integrated. As a generic kernel for generative and evolutionary

design, the purpose of this research is to develop the hierarchical kernel in such a way that it is used to configure the applications with generalized data structures and evolutionary mechanisms. To achieve this, an adaptation mechanism is needed in order to alter the complexity of generative techniques to suit the problem at hand based on exploration at an upper level of abstraction. This point will be visited again with an example.

Without limiting the application of the kernel to practical design tasks, evolution in this thesis is treated in a broader meaning that includes progressively dynamic changes in both data structures and inference mechanisms. Within such a formulation, the kernel can employ a wide range of evolutionary algorithms as inference engines to derive design solutions by increasing their specifications and reducing abstractions. In this thesis Genetic Algorithms (GA) and other Darwinist evolutionary mechanisms are treated as evolutionary mechanisms as in conventional evolutionary design. Other mechanisms, which may simply change a design object or its related evolutionary mechanisms, are also treated as evolutionary mechanisms. There are three basic functions that these evolutionary mechanisms support the process in general. That is, generation, exploration and adaptation.

Generative techniques often involve a certain degree of procedural repetition. In the proposed generative and evolutionary kernel, a generative mechanism itself can be further generalized to become the one that generates a more abstract object. Cellular Automata, L-System, and Shaper Grammar are some of these mechanisms, as well as those performing simple self-replications. These mechanisms explore possible design solutions based on the existing knowledge, while adaptation mechanisms reconstruct the existing knowledge in certain way. GA is both an explorative and adaptive mechanism. Furthermore for handling practical design applications with human designers, interfaces for manually interaction with the system are also treated as a form of evolutionary mechanisms.

With this process model of generalization and specification, the following subsections describe how design objects and design process are handled with three evolutionary mechanisms to form a hierarchy, referred to as Generative and Evolutionary Design (GED) kernel.

5.1.2 An Architecture of Generative and Evolutionary Design

The architecture for generative and evolutionary design is an extendable hierarchical structure in which the development and instantiation of *evolutionary elements* and *evolutionary mechanisms* resemble the process of design exploration and adaptation. Within such an architecture, a design starting as an evolutionary element is gradually specialized using evolutionary mechanisms, resulting in alternative design solutions at each level of the hierarchy being generated, evaluated, selected, and further elaborated within the same hierarchy downwards until the process comes to an end either by the satisfaction of the users or the system hierarchy can no longer be expanded due to implementation limitation.

In the hierarchy, each evolutionary element has its design attributes or parameters which represent a specific level of abstraction for the design problem. An element evolves in the hierarchy according to its attached evolutionary mechanisms. With its expendable hierarchy, the kernel offers a generative capability based on the evolutionary mechanisms for individual evolutionary elements. In the hierarchy, a design problem can be represented with different degrees of complex, each of which has a different representation. These representations at different levels may form a general to specific relation in a top down direction. The representation at a lower level of abstraction may inherit those attributes above, but with additional variables and constraints introduced at its own level to further specialize a design solution.

When the kernel is applied to solving design problems, the evolutionary elements at an upper layer represent the problem, solution or sub-solutions at a much more

abstractive conceptual level than those at a lower layer. Those evolutionary elements in lower layers represent in much more concrete formats closer to the final design output domain. For example, an upper layer may represent textual specifications while a lower one may represent 3D models.

Figure 5.2 shows the dynamical structure of the GED hierarchy during the evolutionary process, and illustrates how it is developed from a simple form to a complex one in a design application. The hierarchy at each time frame represents a set of design solutions, situated at different layers of representation abstraction. Each layer represents the set of connected evolutionary elements (the vertices in the figure) that construct the design solution(s) at that representation domain.

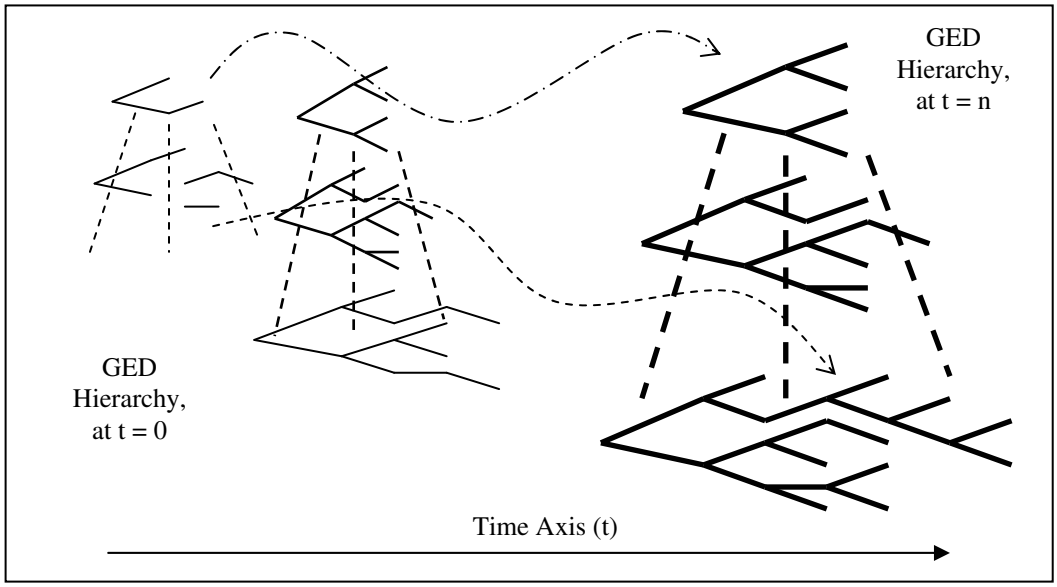
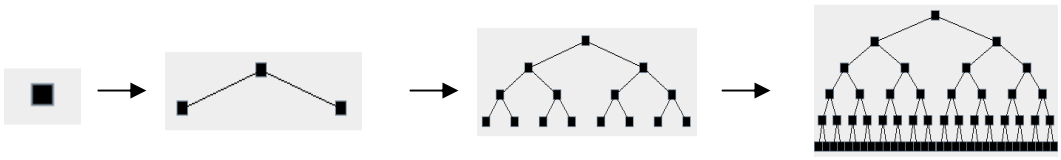


Figure 5.2: Dynamics of the GED hierarchy in a temporal axis.

Figure 5.3 illustrates such evolutionary structure of an actual example discussed in details in later Chapter 6. At time *t=0* in figure 5.3(a), there is only one evolutionary element seed in our GED, the root seed, which has no linked lower abstract elements. The attached evolutionary mechanism is a simple self-replicating mechanism, which will replicate the root to two children in the next level as in figure 5.3(b). Such replication is then propagated to lower abstraction levels and the process will continue at the following time frames. A dynamically

evolving hierarchical structure is formed, as shown in figure 5.3(c) and 5.3(d). More than one solution may situate at one representation level, especially when population-based evolutionary techniques such as GA are used as the evolutionary mechanism.



(a) (b) (c) (d)

Figure 5.3: Dynamics of an evolving GED, at (a) t = 0, (b) t = 1, (c) t = 3, and (d) t = 5.

The growth of the tree structure in breadth first or depth first manner reflects a possible way in which a design solution is explored. Design objects are represented as evolutionary elements which contain design attributes or parameters, with attached methods to allow themselves to be realized, visualized or presented in meaningful design representations. Each of them may also be linked with two sets of evolutionary elements (one above and one below) and one set of evolutionary mechanisms.

Evolutionary mechanisms are essential parts of the kernel to make the computational GED hierarchy change dynamically, and evolutionarily. They support three basic functions: design generation, exploration and adaptation. These functions are achieved through

1) evolving the evolutionary elements to which they are attached,
2) influencing other evolutionary elements at adjacent layers in the hierarchical structure, and/or
3) adapting to new inference mechanisms when the data they received require doing so.

Therefore, changing design objects at one abstraction level may lead to the corresponding modification of those at levels down below, and their evolutionary mechanisms.

In this study, evolutionary mechanisms are not restricted to genetic algorithms. They are not limited to the category of conventional evolutionary computation techniques either. Any module can be an evolutionary mechanism in the proposed kernel as long as it offers a dynamic mechanism to evolve design objects, based upon specific dynamic environment towards a specific tendency or preference. Designers, users, expert systems, Genetic Algorithms are potential evolutionary mechanisms since their actions interacting with the system in an intelligent way may achieve the same or sometimes even better results than an invoked computer program. These mechanisms which involve human participations may be described as entities, or agents, that run autonomously, and preferably collaboratively, under a dynamic environment. However, in general, user interfaces and evolutionary mechanisms are treated differently in the kernel that has been implemented as a prototype in this thesis. In this prototype, several standard evolutionary mechanisms are integrated. The general notation of any thing such as user interactions acting also as evolutionary mechanisms are not fully addressed due to the fact that it is out of the scope of the thesis.

5.1.3 Abstraction and Interpretation of Design Objects

In the proposed kernel, internal data are meaningless without correct representation and interpretation of them. This is because internally it is important to have an efficient representation domain in order for the computer systems to manipulate them with evolutionary mechanisms. But with a specific design application, domain specific representation or interpretation functions are needed in order to map the kernel data to some meaningful, interpretable, or even interacting forms to the external computers, designers, and users, as shown in Figure 5.4.

The proposed computational kernel itself is application-independent. How this kernel is applied to specific design tasks relies on the correct interpretation of the

data in the kernel to human-understandable information. Thus there is an interpretation middle-layer that links the kernel data to the external domain.

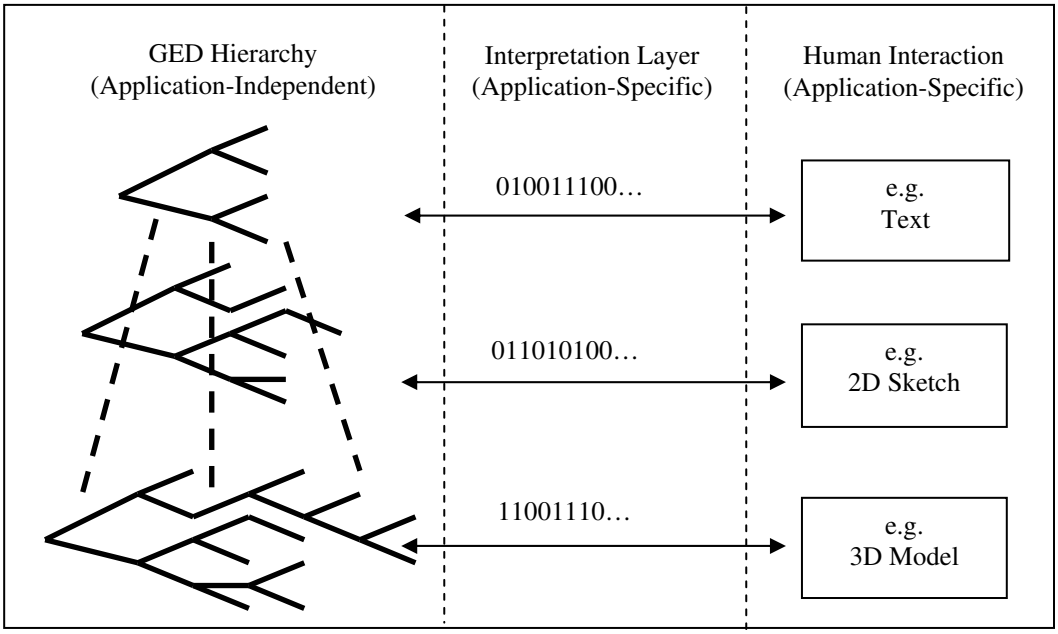


Figure 5.4: A computational system using the GED Kernel with human interaction.

Figure 5.5 shows an example of how this GED hierarchy can be mapped to the representation of wineglasses at different abstraction levels. In this example, internal data of each evolutionary element are represented with wineglasses at various abstraction levels, such as parametric features of wineglasses, the 2D profile and the 3D geometric models as shown. Details of this example will be discussed in the later chapter 8.

The kernel of generative and evolutionary design developed in this research is based on a hierarchical structure in which evolutionary elements and evolutionary mechanisms are instantiated in a dynamic process during which a design is developed from an abstract concept to a detailed specification. The model for such a hierarchical structure can be formulated as a process of generalization and specialization during which a design solution and its alternatives are explored with the participation of designers interacting with the system.

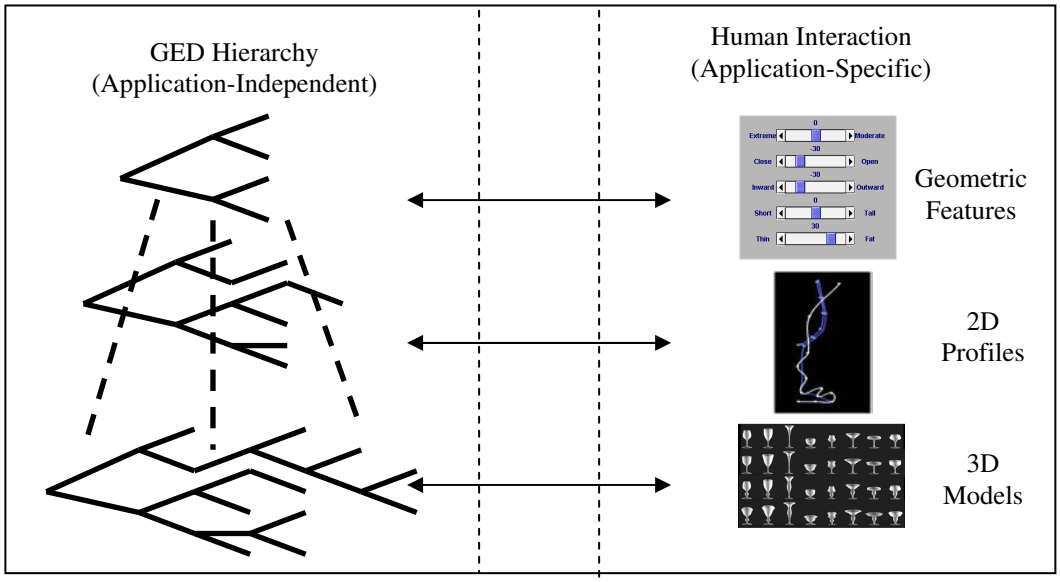


Figure 5.5: An example system using the GED Kernel with wineglass representation.

5.2 Formal Representation of GED

The proposed GED supports the exploration and adaptation of design in a generative and evolutionary process in which design objects are derivable from the developed evolutionary elements in a hierarchical representation. The representation includes a set of connected evolutionary elements and their attached evolutionary mechanisms. With this representation, a GED hierarchy of multiple design objects can be instantiated for a given specific design task. The formal representation defines these evolutionary elements and mechanisms with the data primitives that can be mapped into design attributes or parameters of a specific design.

5.2.1 Representation of Design Objects

The formal representation starts from computational primitive *p*, which is the most fundamental element in the GED system. The primitive is a numerical set, such as real number or integer, which is used to represent physical design object in a computational form. Then a design parameter *dp* can be defined in the GED

system that represents a (or a part of) potential design object at a specific level of abstraction. It can be either:

a) a simple primitive, *p*, or
b) a finite sequence of primitives < *p_1, p_2, p_3, …p_n* >

A design parameter *dp* in the representation is application-independent. It is only meaningful within the context of a specific domain in which it can be interpreted by designers or users. Using the GED kernel, when a specific domain of design applications is introduced, a design parameter must be mapped to a domain of a specific application, which is conceivable to human users or designers. For example: a design parameter *dp* can be mapped to a 2D graphic domain, ***g2d***, with a representation function ***fr***,

$$\boldsymbol{fr}(dp) = g,\ g \in \boldsymbol{g2d}.$$

5.2.2 Evolutionary Elements and Evolutionary Mechanisms

The basic element, the evolutionary element *e*, is a 4-tuple, *(dp, **M**, **Eh**, **El**)*, which consists of

1) a design parameter *dp*,
2) a set of evolutionary mechanisms ***M***,
3) a set of linked evolutionary elements of higher level abstraction ***Eh***, and
4) a set of linked evolutionary elements of lower level abstraction ***El***.

An evolutionary mechanism, *m*, is a member of ***M*** that evolves the attached evolutionary element *e*. The evolutionary mechanism not only changes the design parameter *dp* in *e*, but also influences its evolutionary mechanisms ***M*** and the linked evolutionary elements ***Eh*** and ***El***. This influence may even further

propagate to the evolutionary mechanisms of the linked evolutionary elements. In general, they can be represented in a temporal form at time *t* as

$$e_t = (dp_t, \mathbf{M_t}, \mathbf{Eh_t}, \mathbf{El_t}) \tag{5.1}$$

When only one evolutionary mechanism m_{t-1} is attached to an evolutionary element at time *t-1*, there is a relation in the form of

$$\begin{aligned} e_t = m_{t-1}(e_{t-1}) &= m_{t-1}(dp_{t-1}, \mathbf{M_{t-1}}, \mathbf{Eh_{t-1}}, \mathbf{El_{t-1}}) \\ &= (dp_t, \mathbf{M_t}, \mathbf{Eh_t}, \mathbf{El_t}), \text{ where } \mathbf{M_{t-1}} = \{m_{t-1}\} \end{aligned} \tag{5.2}$$

There are three special cases regarding the results of evolving an element. When an evolutionary element is the same as before, $e_t = m_{t-1}(e_{t-1}) = e_{t-1}$, it can be regarded as at a saturated or at an inactive stage. When an evolutionary element is periodically repeated within a sequence of temporal pattern, it is at a vibrated or cyclic stage. Finally an evolutionary element may also be eliminated, and vanishes.

In conventional evolutionary design with a Genetic Algorithms (GA), some direct mappings are used to transform a genotype to a phenotype. They can be realized as a specific function that only maps a design parameter *dp*, to another design parameter *dp'*, with the evolutionary mechanism $dp_t = m(dp_{t-1})$. In this study they are handled in the kernel as a specific form of evolutionary mechanism, *dm*, such that they can only change the design parameters of the linked lower abstraction elements.

$$\begin{aligned} e_t = dm_{t-1}(e_{t-1}) &= dm_{t-1}(dp_{t-1}, \mathbf{M_{t-1}}, \mathbf{Eh_{t-1}}, \mathbf{El_{t-1}}) \\ &= (dp_{t-1}, \mathbf{M_{t-1}}, \mathbf{Eh_{t-1}}, \mathbf{El_t}) \end{aligned} \tag{5.3}$$

Or

$$\begin{aligned} e_t = dm_{t-1}(e_{t-1}) &= dm_{t-1}(dp_{t-1}, \{dm_{t-1}\}, \mathbf{Eh_{t-1}}, \mathbf{El_{t-1}}) \\ &= (dp_{t-1}, \{dm_{t-1}\}, \mathbf{Eh_{t-1}}, \mathbf{El_t}) \end{aligned} \tag{5.4}$$

Only the design parameters related to a lower level of abstraction $\boldsymbol{El_{t-1}}$ is changed to $\boldsymbol{El_t}$ while all dp_{t-1}, $\boldsymbol{M_{t-1}}$ and $\boldsymbol{Eh_{t-1}}$, are kept unchanged in this case.

Extending this concept to another evolutionary mechanism, a generative mechanism, *gm*, can then be realized as a function that changes the design parameter of an evolutionary element and those linked to it at a lower level of abstraction.

$$e_t = gm_{t-1}(e_{t-1}) = gm_{t-1}(dp_{t-1}, \boldsymbol{M_{t-1}}, \boldsymbol{Eh_{t-1}}, \boldsymbol{El_{t-1}}) = (dp_t, \boldsymbol{M_{t-1}}, \boldsymbol{Eh_{t-1}}, \boldsymbol{El_t}) \quad (5.5)$$

Or

$$e_t = gm_{t-1}(e_{t-1}) = gm_{t-1}(dp_{t-1}, \{gm_{t-1}\}, \boldsymbol{Eh_{t-1}}, \boldsymbol{El_{t-1}}) = (dp_t, \{gm_{t-1}\}, \boldsymbol{Eh_{t-1}}, \boldsymbol{El_t}) \quad (5.6)$$

In this case, only design parameters, dp_t, and those at a lower level of abstraction, i.e., $\boldsymbol{El_{t-1}}$ will be changed.

When an evolutionary mechanism performs an exploration and adaptation activity, it will affect those evolutionary elements and evolutionary mechanism at a higher level. In an ideal case, a GED-based computational design support system with all these evolutionary mechanisms can generate, explore and adapt potential design solutions automatically. In practice, design must accommodate human interaction. Therefore in this formulation human interaction is handled as an external evolutionary mechanism through an interface. Such an interaction usually involves user's invoking a mechanism or a software system to perform a task which changes the concerned evolutionary element and its immediate lower and upper level neighbours.

The evolution of one evolutionary element, *e_k*, at layer *k*, leads to the updating or influencing those elements in the connected layers *k-1* and *k+1* in the bi-directional GED hierarchy, although in a practical implementation it can often be simplified as a top-down one way process.

When there is more than one evolutionary mechanism attached to an evolutionary element in the set $\boldsymbol{M_t}$, there are a variety of execution orders when applying the attached evolutionary mechanisms to the element. Evolving an element with a set of evolutionary mechanisms can be handled in the following ways:

a) Once in a queue: execute only one mechanism in a time, sequentially in a queue,
b) Random: execute one mechanism in a time, randomly, and
c) All in a queue: execute all mechanisms in a time, sequentially in a queue.

In this study the option of "All in the Queue" is implemented in the demonstration examples, as will be introduced in the later chapters. For example, if there are 3 evolutionary mechanisms attached, $\boldsymbol{M_t} = \{m\_1_t, m\_2_t, m\_3_t\}$, then element e_t will be evolved in the order of:

$$e_t = m\_3_{t-1} (m\_2_{t-1} (m\_1_{t-1} (e_{t-1}))) \quad (5.7)$$

If no evolutionary mechanism is attached to an evolutionary element, the evolutionary element is not self-evolvable or in an in-active state. However, it may still be evolved by its linked elements in higher or lower levels.

5.3 General Architecture of GED

The generative and evolutionary design (GED) is generalized as a design support system with a set of connected evolutionary elements and their attached evolutionary mechanisms structured in a hierarchical form. These elements are

allocated at different levels of hierarchy, representing different abstractions of a design or a partial design problem.

A generative and evolutionary design, $\boldsymbol{GED_t}$, contains multiple n layers of connected evolutionary element sets, $\boldsymbol{GED_t = \{ E\_1_t, E\_2_t, \ldots E\_i_t, \ldots E\_n_t \}}$. Each set of evolutionary elements, $\boldsymbol{E\_i}$, represents design at a specific abstraction level, i, and the elements at each level are $\boldsymbol{E\_i_t} = \{ e\_i1_t, e\_i2_t, \ldots, e\_ij_t, \ldots e\_im_t \}$. Elements at higher levels, such as those in $\boldsymbol{E\_1_t}$ and $\boldsymbol{E\_2_t}$ represent design objects in a more abstract form such as functional features while at lower levels $\boldsymbol{E\_n\text{-}1_t}$ and $\boldsymbol{E\_n_t}$ represent in a more detailed form, for example 3D geometric models.

With the above definitions, the formal representation of the GED can be summarized as:

a) *Generative and Evolutionary Design*, $\boldsymbol{GED_t}$, is structured with a set of connected evolutionary element levels in a hierarchical form, at a specific time frame t:
   - $\boldsymbol{GED_t = \{ E\_1_t, E\_2_t, \ldots E\_i_t, \ldots E\_n_t \}}$, where $\boldsymbol{i}$ represents a specific level in the hierarchy.
   - $\boldsymbol{E\_i_t} = \{ e\_i1_t, e\_i2_t, \ldots, e\_ij_t, \ldots e\_im_t \}$, a set of evolutionary elements representing design objects at the specific abstraction level, i.

b) An *evolutionary element*, e_ij_t, in a GED consists of 1) design parameters, 2) a set of evolutionary mechanisms, 3) a set of linked higher abstraction elements, and 4) a set of linked lower abstraction elements:
   $e\_ij_t = (dp\_ij_t, \boldsymbol{M\_ij_t}, \boldsymbol{Eh\_ij_t}, \boldsymbol{El\_ij_t})$, where
   - dp_ij_t is the design parameters of the element, that represent design objects at that specific abstraction.
   - $\boldsymbol{M\_ij_t} = \{ m\_ij1_t, m\_ij2_t, \ldots, m\_ijk_t, \ldots m\_ijr_t \}$, where $\boldsymbol{M\_ij_t}$ is the set of evolutionary mechanisms attached to e_ij_t, i is level-index, j is the element-index, and k is the mechanism-index.

- $\boldsymbol{Eh\_ij_t \subset E\_i\text{-}1\_j_t}$, where $\boldsymbol{Eh\_ij_t}$ is the set of evolutionary elements linked to e_ij_t, at the "higher" abstraction level $\boldsymbol{E\_i\text{-}1\_j_t}$.
- $\boldsymbol{El\_ij_t \subset E\_i{+}1\_j_t}$, where $\boldsymbol{El\_ij_t}$ is the set of evolutionary elements linked to e_ij_t, at the "lower" abstraction level $\boldsymbol{E\_i{+}1\_j_t}$.

c) Each *evolutionary mechanism*, m_ijk_t, in the set, $\boldsymbol{M\_ij_t}$, attached to a specific evolutionary element, e_ij_t, will be applied to the element and produce a new element in the following time frame, $t+1$.

- $e\_ij_{t+1} = (m\_ijr_t \ldots m\_ijk_t(\ldots((m\_ij1_t (e\_ij_t))\ldots)\ldots),$

  where $m\_ijk_t \in \boldsymbol{M\_ij_t}$

  In case of only one evolutionary mechanism attached to the element,
- $e\_ij_{t+1} = m\_ij1_t (e\_ij_t)$, where $\boldsymbol{M\_ij_t} = \{ m\_ij1_t \}$.

d) *Design parameter*, *dp*, in our GED represents a (or a part of) potential design object at a specific abstraction representation. It can be:

1. a simple primitive, *p*, or
2. a finite sequence of primitives $< p\_1, p\_2, p\_3, \ldots >$.

When this GED kernel is applied to model a conventional evolutionary design with standard GA as the evolutionary mechanism, two different architectures can be obtained.

In those cases where the genotype is the same as the phenotype,

- *Root:* $e\_11_t = (genotype_t, \boldsymbol{\{GA_t\}}, \boldsymbol{\{\}}, \boldsymbol{El\_11_t})$.
- *Population:* $e\_2j_t = (geno_t, \{\}, \{ e\_11_t \}, \{\})$,

  where $e\_2j_t \in \boldsymbol{El\_11_t}$

In this case, the $genotype_t$ represents itself as well as the phenotype, and $\boldsymbol{El\_11_t}$ contains the population of all individuals e_2j_t evolved by GA. In those cases where the genotype is different from the phenotype, a 3-level GED can be created:

- *Root:* $e\_11_t = (genotype_t, \{GA_t\}, \{\}, El\_11_t)$
- *Population (geno):* $e\_2j_t = (geno_t, \{gpm\_2j_t\}, \{ e\_11_t \}, El\_2j_t)$, where $e\_2j_t \in El\_11_t$
- *Population (pheno):* $e\_3k_t = (pheno_t, \{ \}, \{ e\_2j_t \}, \{\})$, where $e\_3k_t \in El\_2j_t$

In this case, the *genotype* represents the form of itself as well as the phenotype, and El_11_t contains the population of all individuals in genotype form e_2j_t evolved by GA while gpm_2j_t maps genotype individuals to their corresponding phenotype elements e_3k_t.

The software kernel proposed in this thesis provides basic data structures and inference mechanisms for the development of an application in a domain of design, mainly in product design. The kernel is based on a hierarchy of evolutionary elements and evolutionary mechanisms. The next section gives a summarized description of the kernel and its implementation. The detailed description of how the GED kernel was developed and implemented is given in the Appendix A.

5.4 Steps for Building an Application System with the GED Kernel

An object oriented programming language (Java) has been used to implement the proposed kernel, which consists mainly of four parts:

1) a generic core containing classes of design parameters, evolutionary elements and evolutionary mechanisms,
2) a set of direct interfaces of the classes in the core,
3) a set of graphical user interfaces (GUI) for visualizing and interacting with the GED kernel, and
4) a GED builder that constructs a specific GED application for a specific design task.

The kernel has been implemented as a software package (or library), which can be integrated with other software that supports Java application interfaces, including external commercial CAD tools. Figure 5.6 shows the block diagram of the developed kernel.

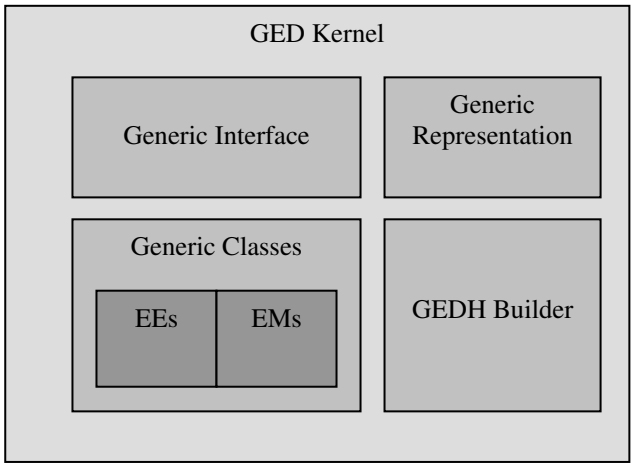
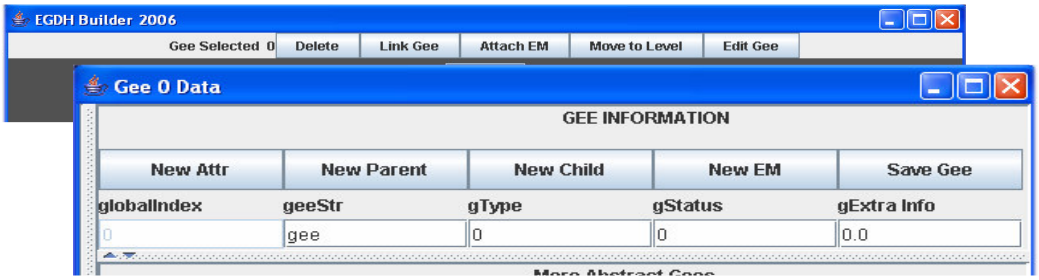


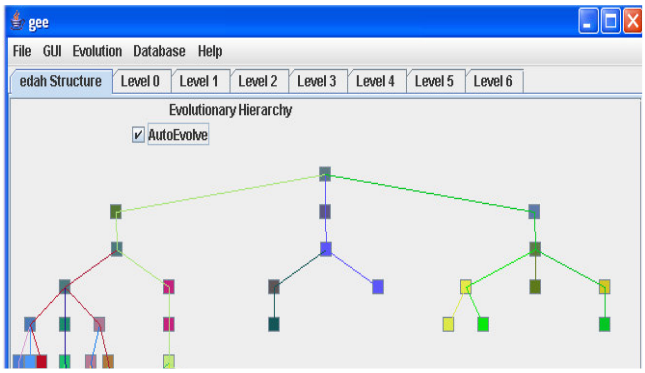
Figure 5.6: Block diagrams of the implemented GED kernel.

To construct a specific GED application with the kernel for a specific design task, the GED builder is initialised with a root empty seed without design parameters, evolutionary mechanisms, or linked higher or lower neighbours, as shown in Figure 5.7(a). An interface for a specific evolutionary element can then be instantiated and edited for its design parameters or attributes as shown in Figure 5.7(b). With the appropriate assignment of evolutionary elements, their design parameters, evolutionary mechanisms and linked neighbours, a GED application is established and the instantiated element as an initial abstract design concept can then be evolved, either manually or automatically, with the input from the users or with those evolutionary mechanisms which have been activated.

Figure 5.7(c) shows the process of how a root element is manipulated and attached with a self-replication evolutionary mechanism. A more complicated hierarchy with more elements is evolved from this root element after a few generations.



(a) Creating an initial empty seed in the kernel architecture



(b) Editing an evolutionary element (c) Evolving a seed into a hierarchy

Figure 5.7: Building a design application with GED Kernel.

In general, an instantiated GED application can be developed in the following steps:

a) Starting with the generic builder to construct a GED hierarchy for a specific design task by creating an initial evolutionary element, as shown in Figure 5.8.

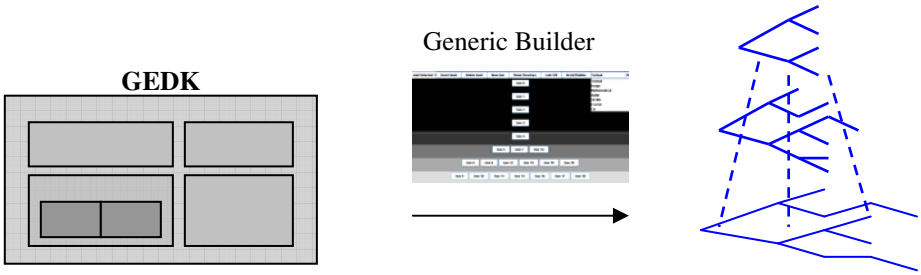


Figure 5.8: A GED hierarchy created with the GED Builder.

b) Manually editing or auto-evolving the initial evolutionary element (with the attached evolutionary mechanisms) located at the highest level of abstraction of the GED hierarchy, as shown in Figure 5.9.

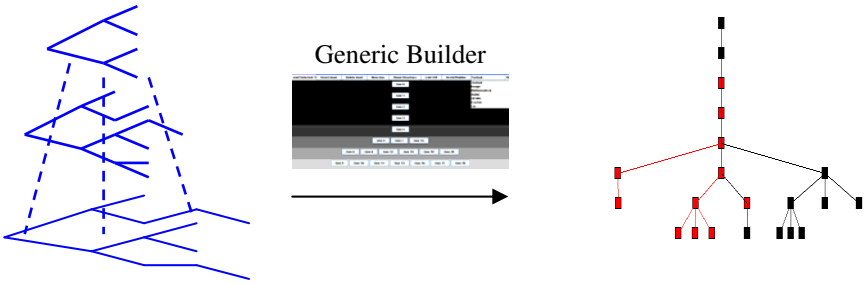


Figure 5.9: Manipulating data to evolve the hierarchy

c) Linking the instantiated GED to an interface for user manipulation or autonomous module interactions (either internal or external interfaces), and converting design representations, such as 2D graphical or 3D, as shown in Figure 5.10.

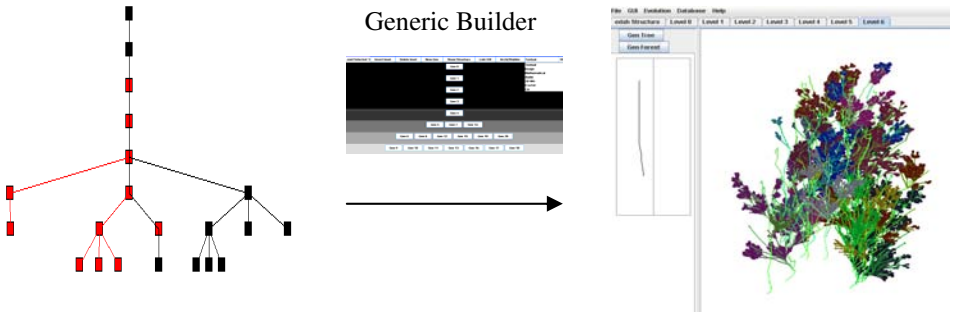


Figure 5.10: Linking interface and representation to a GED Hierarchy.

Figure 5.11 shows a typical GED application system for a specific design task. Three examples of applying the kernel to computational design tasks are to be presented next in order to illustrate in more details how the GED can be used to support generative and evolutionary design applications. These examples demonstrate how the kernel supports design with enhanced explorative and adaptive ability. The first example illustrates how the GED kernel (GEDK) automatically builds a GED hierarchy and explores different forms of plant

structures from a single "self-replicating" evolutionary root element. The second demonstration shows how the GED kernel (GEDK) supports a simple design adaptation, in the form of knowledge reconstruction in design. The third design demonstration works on a more complex design application of generating wineglasses, after being integrated with a commercial CAD tool.

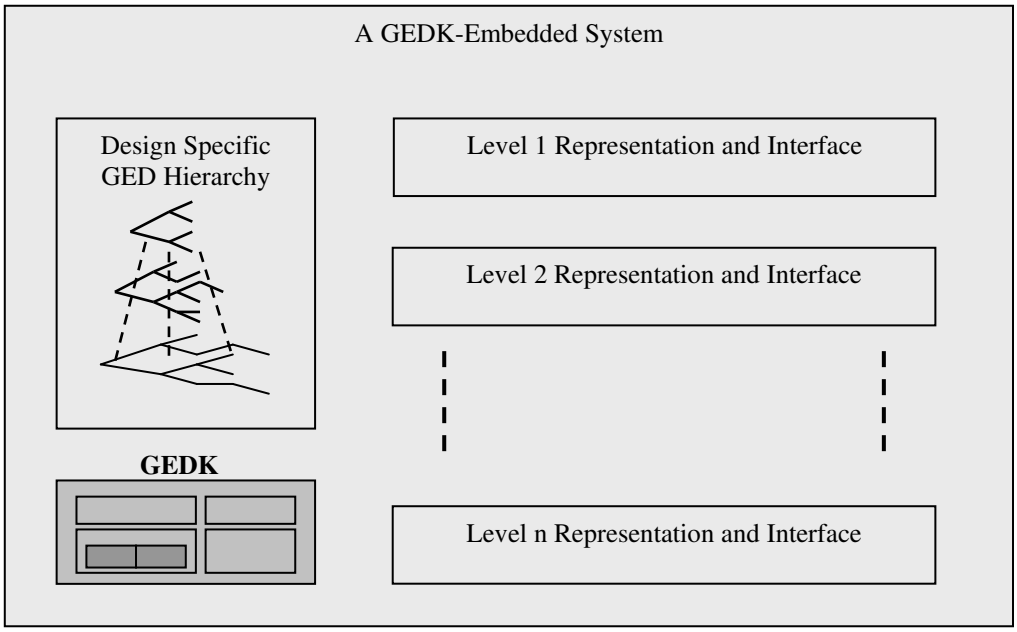


Figure 5.11: A GEDK-embedded application system.

5.5 Summary

This chapter has presented a formal representation of the GED kernel. This formal representation articulates the relation between evolutionary elements and evolutionary mechanisms in a timely manner so that the process of designing using this kernel can be formulated as instantiating and specializing evolutionary elements with the associated evolutionary mechanisms. In this process, a change caused by an evolutionary mechanism operating on an evolutionary element may propagate a chain of actions through the hierarchy. In a real design, a propagation top town represents a process of exploration and specialization, while a propagation bottom up represents a process of backtracking or generalization. The system architecture adopts changes to update the system so that the

consistency of the knowledge it holds is maintained. In a design practice, such a hierarchy allows a designer to create an initial design concept, explore the alternatives of it, and then select one for specialization by going down a level in the hierarchy. If a satisfactory solution cannot be derived at a particular level, the designer may wish to get back to an upper level which is a more abstractive level, to explore more. This process continues until a satisfactory result is derived.

This chapter has further introduced an architecture for generative and evolutionary design. This architecture provides a basis for linking evolutionary elements (representing design objects) and evolutionary mechanisms (representing inference methods or design methods). The evolutionary mechanisms develop and specialize evolutionary elements within a hierarchical structure with which a design concept is evolved to a desirable detail. This represents a process of design exploration and adaptation, during which a design solution is generated, explored, evaluated and selected with the alternatives and justification data are retained for possible back tracking and review. The proposed kernel provides key functions for building an application within this architecture.

Part III:

Application of the GED Kernel in Design Examples

Artificial Plant Generation

With the GED kernel, implementation of a design application system can be simplified. Creating an abstract design concept and then exploring this concept in the hierarchy with evolutionary mechanisms attached to the hierarchy, a design exploration and adaptation task can be supported. The kernel provides key support for the implementation of a design application system. Additional work to make a design application can focus on building the initial objects and providing proper interfaces. The last chapter has given a formal representation of the kernel and a detailed description of building application system with it, the coming three chapters in this Part III present the application of the GED kernel to three different design examples. The detailed implementation and coding issues of developing these applications can be referred to Appendix A.

In the first example presented in this Chapter 6, the patterns of plant structure are generated with the developed GED kernel as a supporting tool. Initially the system starts with a structural gene or seed, attached with a self-replication (SR) mechanism. This root seed then self-replicates itself to a number of children at a lower level of abstraction of the hierarchy. This self-replication proceeds automatically generation by generation. The elements at higher abstraction levels represent the macro structure of the visualized plants, while the lower ones affect the micro details. Different visual effects with artificial plant structures can be generated, with manipulation of the internal evolutionary elements and external interference. The generation and exploration of the patterns can be further enhanced with an evolutionary self-replication mechanism.

6.1 Artificial Life and Plant with Dynamical Hierarchies

This artificial plant example can be related to the research area of Artificial Life. There are different issues in the field of Artificial Life. Some of these are raised in Lenaerts, Gross and Watson's paper (Lenaerts et al., 2002). Bedau et al. (Bedau et al., 2000) also introduced some common issues. One of these issues is to study and simulate the self-organization of dynamical hierarchies in artificial living systems.

The importance of studying the dynamical hierarchies in artificial life is to simulate, explore and adapt the possible patterns, behaviours and outcomes of the simulated systems. This is also close to a generative and evolutionary design approach, in which exploration and adaptation are considered an important issue. To develop computational systems for supporting dynamical hierarchies of artificial lives, the evolutionary behaviour has to be handled. In this example it is shown that the proposed kernel provides potential support to tackling these issues.

This application is developed with the GED kernel and the design is evolved from a single self-replicating cell. After a few generations of evolution, a multi-level hierarchy is generated. When the design parameters of each evolutionary element are mapped to a geometrical structure, various structural patterns are obtained. In this example, a manual single freehand stroke is also treated as an evolutionary mechanism and is seen as an external influence. The GED hierarchy can then be represented and realized as:

a) a single plant, in which the leaves of the hierarchy become the leaves of the plant, or
b) a group of plants in which a path from the root to each leaf becomes a plant of the group.

6.2 Manipulation of Evolutionary Elements in the GED

The specific GED hierarchy in this demonstration can be formulated as:

- e_11_t = ($structuralPara\_11_t$, {SR_11_t }, {}, $\boldsymbol{El\_11_t}$),
  where SR = *Self-Replication Mechanism*
- e_ik_t = ($structuralPara\_ij_t$, { SR_ik_t }, { $e\_(i-1)j_t$ }, $\boldsymbol{El\_ik_t}$),
  where e_11_t is the root seed, while e_ik_t is the k^{th} evolutionary element situated at the *i* level of the hierarchy.

All elements (except the root) have only one parent. SR_t is a self-replicating evolutionary mechanism. Each self-replicating mechanism, SR_t, attached to an evolutionary element, e_t, at any time frame, *t*, can be in the state of 1) inactive, $e_{t+1} = SR_t\ (e_t) = e_t$, or 2) actively replicating a fixed number of new elements and linking them to a lower abstraction level of the element, such that

- $e_t = SR_{t-1}\ (e_{t-1}) = SR_{t-1}$ ($structuralPara\_e_{t-1}$, {SR_{t-1}}, {eH_{t-1}}, {})
  = ($structuralPara\_e_{t-1}$, {SR_{t-1}}, {eH_{t-1}}, $\boldsymbol{El_t}$)
- el_i_t = ($structuralPara\_el\_i_t$, {SR_i_t }, {e_t}, {}), where $el\_i_t \in \boldsymbol{El_t}$

In this example, the evolutionary element e_t becomes inactive if it has one or more replicated elements at a lower level of abstraction. In other words, only the elements without any lower level elements can replicate themselves.

6.2.1 From a Simple Seed to a More Complex Hierarchical Structure

At time *t=0*, there is only one evolutionary element seed in the GED application, i.e., the root seed e_11_t, which has no linked higher or lower abstract elements, as shown in Figure 6.1(a). The attached evolutionary mechanism is a self-replicating mechanism.

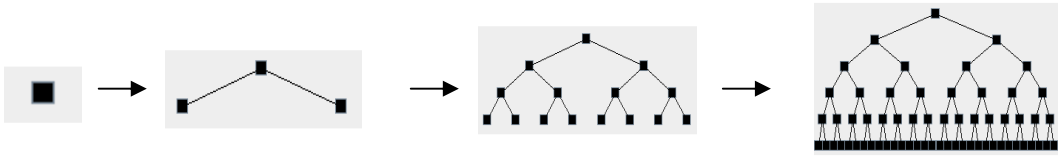
- e_11_0 = ($structuralPara\_11_0$, { SR_11_0}, {}, {})

At time *t=1*, the root seed attempts to replicate itself with the attached self-replicating mechanism, and a number of lower level images are produced as shown in Figure 6.1(b).

- e_11_1 = ($structuralPara\_11_1$, { SR_11_1}, {}, $\mathbf{El\_11_1}$)
- el_2i_1 = ($structuralPara\_el\_2i_1$, {SR_2i_1 }, {e_11_1}, {}),

  where $el\_2i_1 \in \mathbf{El\_11_1}$

If more than one root images have been replicated, the attached self-replicating mechanism of the root seed, SR_11_1, at this time frame, *t=1*, becomes inactive. Such replication will then be propagated to the lower abstraction levels and the process will continue at the following time frames, and a dynamically evolving hierarchical structure will be formed, as shown in Figure 6.1(c) and 6.1(d).



(a) (b) (c) (d)

Figure 6.1: An evolving GED hierarchy, at time (a) t = 0, (b) t = 1, (c) t = 3, and (d) t = 5.

6.2.2 Manipulating Internal Design Parameters of Evolutionary Elements

This evolving hierarchy can be interacted with the interfaces supported by the kernel and represented with a plant form structure, as shown in Figure 6.2. Based on the hierarchy generated with the SR mechanism having a fixed number (2) of replications as shown in Figure 6.1, a regular pattern of plants can be generated as shown in Figure 6.2(a), as the design parameter of the root seed has a line pattern and this line pattern is propagated towards the leaves. Design parameters *structuralPara* of each element can be further manipulated by the users, and different forms of desired plants can be generated as shown in Figure 6.2(b). In Figure 6.2(b) a less regular pattern is generated when the root seed pattern is mapped to a curve, which is further propagated to the leaves.

However, for generating more realistic, flexible and seemingly natural form of plant structures, more features should be supported. Some degree of randomness may be applied. External influences such as user defined features and evolving SR mechanisms are further introduced in the next two subsections for extending the possible patterns generated.

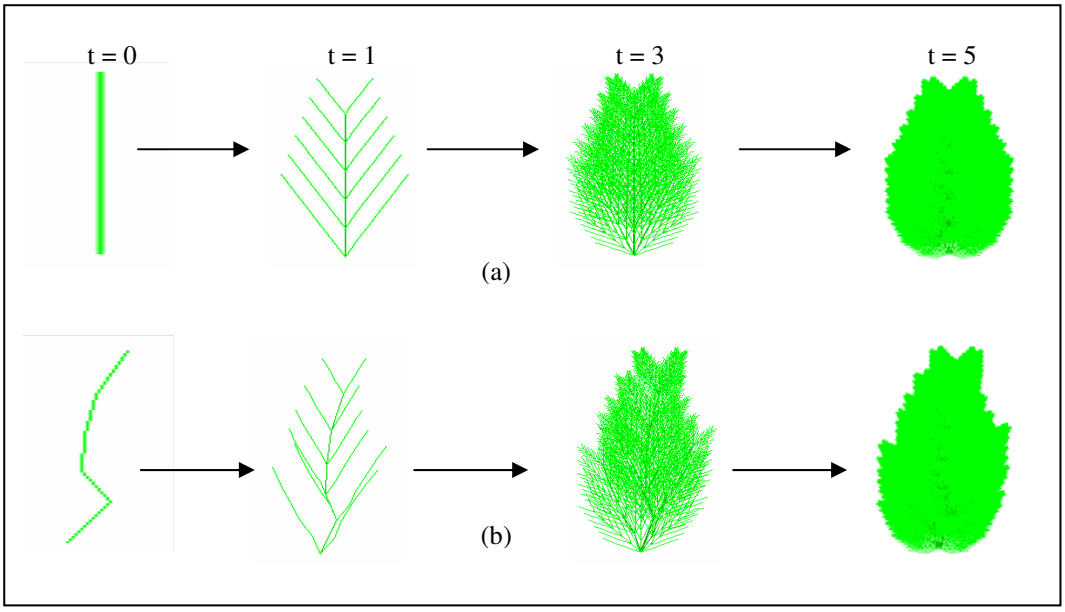


Figure 6.2: Artificial plants generated with (a) a straight line, and (b) an irregular curve, at t = 0, t = 1, t = 3 and t = 5.

6.3 External Influences

For supporting user interaction in the system, manual drawn curve is applied as an external influence of the GED hierarchy in this example. When external influence is added as a generative mechanism, the formal representation of this example is formulated as below:

- e_11_t = { $structuralPara\_11_t$, {SR_11_t , $ExInf\_11_t$}, {}, $\mathbf{El\_11_t}$ }
- e_ik_t = { $structuralPara\_ij_t$, {SR_ik_t , $ExInf\_ik_t$}, { $e\_(i\text{-}1)j_t$ }, $\mathbf{El\_ik_t}$ }

where *SR* = Self-Replication Mechanism, and *ExInf* = External Influence at level *i*.

In the implemented kernel, external influence *ExInf* is supported in the form of geometric deviation through 2D curve points as shown in Figure 6.3. The GED hierarchy in this case is exactly the same as the hierarchy used in the Figures 6.1 and 6.2, with a static SR mechanism of 2 children. With this external influence and some degree of randomness, more natural plant patterns can be manipulated and produced.

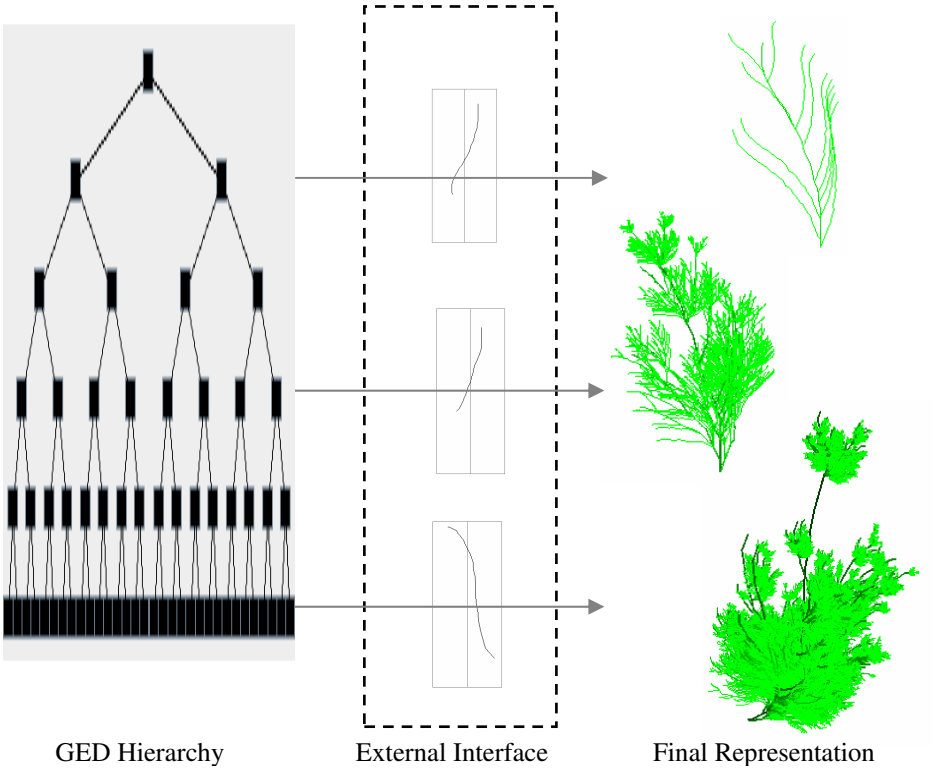


Figure 6.3: More natural plants generated with external influences and some randomness.

6.4 Enhancing Exploration with a Self-Replication (SR) Mechanism

The results of the GED system developed for generating plant-like structural pattern have shown how the GED hierarchy can be automatically evolved from a single root seed to a more complicated structure. The hierarchy can be further represented and visualized in a 2D image of plant form. Human interaction with

internal evolutionary elements and the external influences can act as a manual evolution mechanism that supports interactively exploring and generating variety of outcomes. The results have shown that different effects are obtained when the changes are made at different levels of the hierarchy.

Generation and exploration of the output can be further enhanced when the SR mechanism can be evolved instead of being fixed in a specific form or type. SR mechanism used in the last examples has a fixed number of replications and reproduces two "exact" elements. When an evolutionary SR mechanism is applied such that the number of replication is evolved over time instead of a fixed number, much wider possible outcomes can be obtained and more naturally realistic plants can be generated as shown in Figure 6.4.

The left GED hierarchy in Figure 6.4 shows that the overall hierarchy is not a simple binary tree as those in previous examples and the numbers of lower elements attached at each element are different. At any time frame t, an evolutionary element can be 1) inactive, $e_{t+1} = SR_t\ (e_t) = e_t$, or 2) actively replicating zero, one, or more new elements and link them to a lower abstraction level element. Furthermore, the design parameters of the replicated ones may be deviated from those of the original ones.

As discussed earlier, the hierarchical structure generated with the kernel alone is application-independent. Different forms of representation, visualization and realization may be applied to the framework for interacting with external users or intelligent software agents. The application so far generated one plant from the hierarchy. In fact the same hierarchy can be realized differently, such as a group of plants as shown in Figure 6.3. Instead of representing a level in the hierarchy as a branch of a plant, each path from the root seed to a leave becomes an individual plant when the hierarchy is presented as a group of plants.

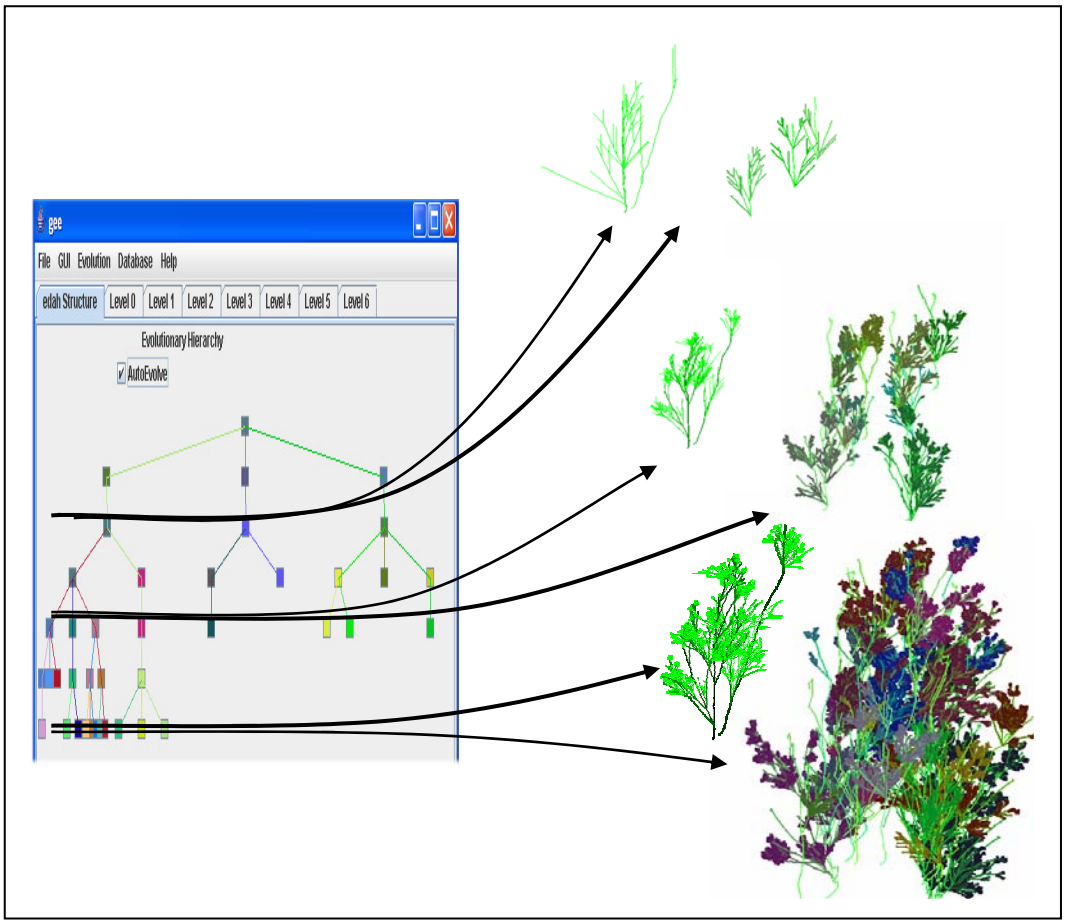


Figure 6.4: More flexible generation effect obtained with an evolutionary SR mechanism.

6.5 Issues and Discussions

This example has shown the exploration ability of the kernel in generating structural patterns with a simple self-replication generative mechanism. The GED hierarchy developed with the kernel and its builder for this example can automatically be evolved from a single seed to a complex plant or a group of plant structures. This GED system also supports manual intervention as an external interference of the evolutionary elements situated at different abstraction levels in the hierarchy to produce different generation effects when needed. More generic structural patterns can be explored if changes are made at higher abstraction levels while evolution of elements at lower abstraction levels

produces detailed modifications near the leave level of the plant. Generation and exploration can be enhanced with a dynamically evolving SR mechanism. Regular structural patterns as well as more natural realistic plants with certain randomness are produced.

All domains at different levels are represented with geometrical structures. Higher levels can be realized as more geometric abstractions containing information of basic overall structural patterns, while lower levels obtain finer or detailed information of the final plant structure. The GED hierarchy itself is application-independent. While the elements of this hierarchy is mapped and realized to a plant-like structure, it can also be mapped to other representations such as image or audio. This mapping of different representations may provide the similar pattern in some other forms of design outputs, but it needs further work to represent this hierarchy in a more generic manner.

6.6 Summary

This chapter has described the GED kernel and explained how it can support the development of generative and evolutionary design. The application of the kernel in an example of generating plant-like structures is presented. This simple example illustrates mainly how the developed GED kernel supports and enhances a more flexible exploration of generating potential patterns with the manipulation of design representations at different abstractions and evolutionary generative mechanisms. However, without proper adaptation ability supported in the system, such generation activity may behave as an aimless and inefficient exploration. The demonstration example presented in the next chapter in generating and matching 2D digital image patterns with a simple 1D binary Cellular Automata presents how the kernel supports such adaptation.

2D Pattern Generation and Matching

The last example is built with the kernel as a supporting tool without any additional external software. With some additions of problem specific supporting systems, more complex applications can be developed with adaptation ability. With such a system the potential problems of design can be explored more efficiently. In this chapter, an example using Cellular Automata as an evolutionary mechanism is introduced to provide a demonstration.

This example illustrates how design knowledge is evolved and adapted for generating and matching desired patterns with the GED system. Genetic Algorithm (GA) is used as an evolutionary mechanism that handles the global exploration and adaptation of a set of elementary elements, each of which is attached with another evolving generative mechanism – Cellular Automata (CA). GA provides a mechanism for adaptation and exploration through its objective function. CA contains the design knowledge of generating 2D pattern formation with its transition rules and seeds (initial cell states). Furthermore, a constraint mechanism (CM) is used to restructure the knowledge embedded in CA in terms of design objects and their generative design process.

7.1 Generating 2D Patterns with Cellular Automata (CA)

Cellular Automata (CA) is firstly introduced by von Neumann (1966). It is a specific computational model with a simple self-organizing mechanism. Much attention is particularly given to its simple self-organizing mechanism applied locally, which can seemingly produce complex global behaviours or patterns. The cells (elements) of CA behave in a self-organizing manner with their

neighbours according to a transition rule. A well-known 2 dimensional CA is Conway's game of life (Berlekamp et al., 1982).

In the case of using CA to generate 2D patterns in this example, knowledge is referred to as

1) how 1D spatial information (the 1D CA seed) is evolved (with the transition rules), and
2) how the history of this evolution is used to form the 2D pattern.

To simplify this illustration the second issue is kept static with sequentially packing 1D cell arrays together to form the 2D image. The first issue is directly related to the transition rules and initial states (seeds) of CA.

One-dimensional (1D) CA can generate a 2D image pattern based on initial states (or seed) and the transition rules. It can be regarded as a generative mechanism, and can generate a more complex 2D pattern with its evolutionary rules (the transition rules) and its states. Figure 7.1 shows 5 sample patterns generated with different transition rules (TR) and initial states (S_0) of a simple 1D binary CA.

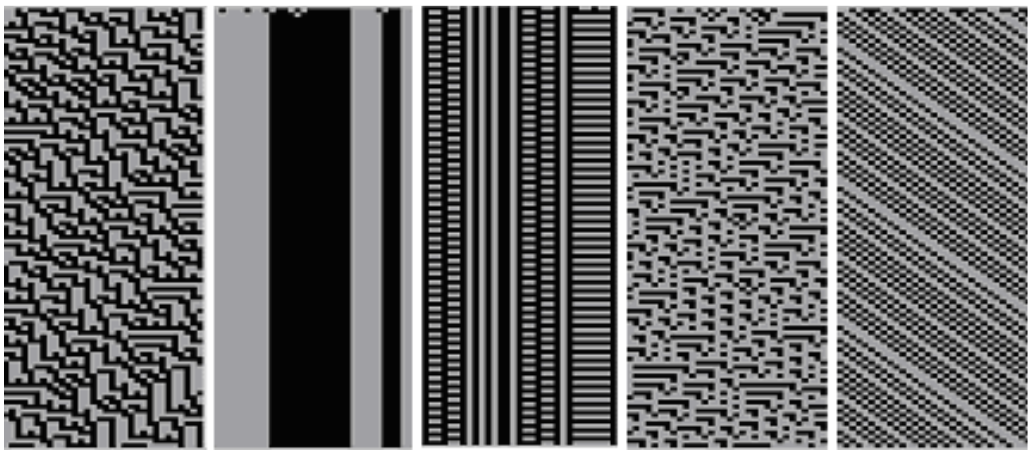


Figure 7.1: Patterns generated with a simple 1D binary CA.

In the example, CA plays the central role on generating seemingly complex 2D patterns with simple transition rules and seeds, which are regarded as another

representation of the 2D patterns at a higher abstraction level. As introduced in an earlier chapter, CA can be categorized into different types:

1) totalistic,
2) semi-totalistic,
3) non-totalistic 1D binary CA with 2-neighbourhood.

A totalistic CA has transition rules that can be realized as density functions. The next state of a cell in the transition is solely dependent on the density of its local region, and thus the sum of the states of its neighbourhood. Furthermore, if the transition rules lead the next state of a cell to a new state that depend not only on the sum of the states of neighbouring cells but also on the state of its own, such CA and their rules are called semi-totalistic. The transition rules of a non-totalistic CA are not related to the density of its local region, and each next state of a cell depends on itself and the state patterns of its neighbours. The possible patterns produced by a totalistic CA is a subset of those by semi-totalistic, which in turn is a subset by non-totalistic.

Therefore, the next state (s') of a cell (c) having current state (s), of a totalistic CA is

- $s' = TR(Sum(All\_NeighbourStates(c)))$

For a semi-totalistic CA, it is

- $s' = TR(Sum(All\_NeighbourStates(c), s))$

For a non-totalistic CA, it is

- $s' = TR(All\_NeighbourStates(c), s),$

where *All_NeighbourStates(c)* is the list of the states of the cell c neighbours.

In the case of 2-neighbourhood:

- $s' = TR(Sum(s+1, s-1, s)),$

A simple CA can be modelled with the GED kernel in the form of

- $eCA1D_t = (1DPattern_t, \{CA_t\}, \{ \}, \{eCA2D_t\})$
- $eCA2D_t = (2DPattern_t, \{ \}, \{eCA1D_t\}, \{ \})$
- $CA_t(eCA1D_t) = eCA1D_{t+1}$

such that

- $1DPattern_{t+1} = 1DPattern_t$
- $CA_{t+1} = CA_t$
- $2DPattern_{t+1} = TR\_of\_CA_t (1DPattern_t)$

where *TR_of_CA* represents the transition rules of CA applied to the initial 1D pattern.

In a design application, designers or users manipulate data at a high level of abstraction in terms of design features or characteristics. For example, instead of inputting the actual checkerboard pattern or specifying the initial seed and transition rules of CA, a designer may wish to characterize his/her own desired pattern with a feature stated as "no adjacent cells have the same colour in the digitized space". However, CA itself has no mechanism to control and reflect this characteristic. Furthermore, designers sometimes may not be able to specify what a desired feature is in explicit terms. In this case, genetic algorithm is a better choice to handle this problem, as will be explained below.

7.2 Integration of CA with Genetic Algorithm (GA)

Holland's Genetic Algorithm (1975) can be realised as a stochastic searching method, which seeks optimal solution(s) from a pool of possible candidates (the population). From the initial population, which is often generated randomly, the candidates/individuals/chromosomes of the population go through evaluation, selection, crossover and mutation from one generation to another.

In this example, the goal of the Genetic Algorithm is reflected in its selection process, either through a formulated objective function or through artificial

selection. GA also provides the main interaction with designers as an external influence to the system.

Constructing a mini GA-CA system with GA and CA mechanisms to solve the problem just mentioned is a case of GA controlling CA. This can be related to the studies of applying GA to CA for solving the problems of density classification and synchronization (Das et al., 1995). Instead of finding a universal rule that can be applied to every random seed for obtaining a final goal or a pattern, the main goal here is to find the right transition rules that can be applied to the right seeds to produce the right patterns. The global environment for producing the right pattern is governed by the GA. This GA controls the transition rule of each CA instead of the final pattern, while the transition rule of CA influences the final pattern.

In this GA-CA system, GA captures the characteristics or features of the desired patterns. When these features are formulated and embedded into the objective function of the GA, the GA-CA system can then run automatically without human interaction to achieve the goal. However, design problems are often ill defined, and the problem itself is to be evolved during the design process together with the solutions. Furthermore design knowledge is often implicit in designers' mind and this tacit knowledge cannot be formulated easily in computational systems. In this situation, GA can offer an artificial selection mechanism to let designers select the candidates with desired features without explicitly specifying what those features are. This implicit human objective function guides the system to produce the similar effect of selecting and reproduce fitter candidates with a computational objective function.

The evolution of the CA transition rules and the seed, governed by the GA, exhibits a way of knowledge evolution, which explores the possible solutions at a different level of abstraction in terms of the representation of 2D patterns. As shown in Figure 7.2, a population of three CA is governed by the GA. As the

checkerboard feature can be formulated and programmed as an objective function, this GA-CA system can automatically evolve the right transition rules and the seed. Furthermore, the client user can also rate the CA candidate(s) or artificially select the best-matched CA according to his/her desired pattern. With this external influence to GA, the checkerboard pattern can be obtained after a few generations with the GA seeking the right seed and transition rules for the CA to run.

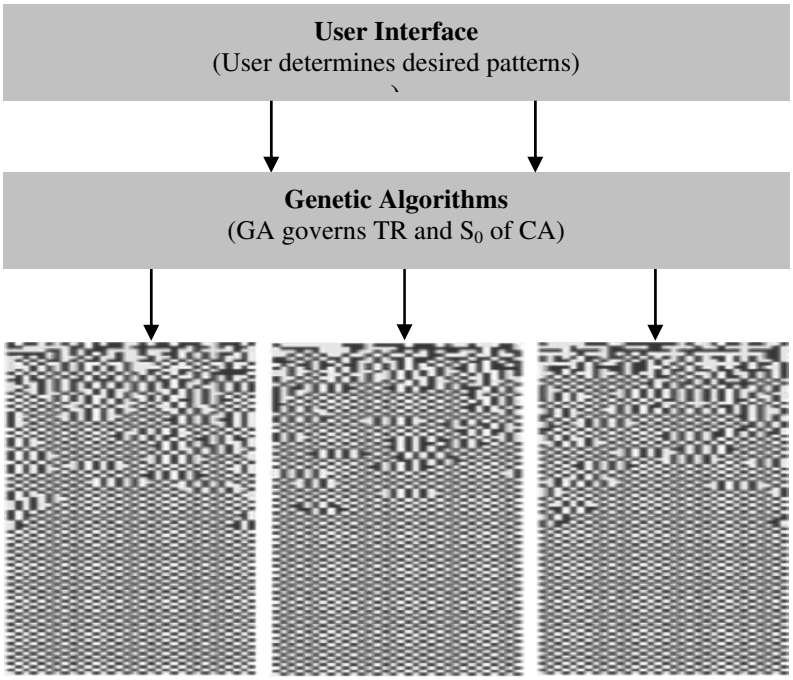


Figure 7.2: Patterns generated with a GA-CA system, having a checkerboard as the goal.

With the GED kernel, this GACA application can be structured as:

- eGA_11_t = { $gaPara_t$, **{ GA_t }**, {}, **El_11_t** }
- $eCA1D\_2j_t$ = { $1D_t$, { CA_t }, { eGA_11_t }, { $eCA2D\_3j_t$ } },

  where $eCA1D\_2j_t \in El\_11_t$
- $eCA2D\_3j_t$ = { $2D_t$, { }, { $eCA1D\_2j_t$ }, {} }

With GA, the right pattern can be generated with the given CA and the explored pairs of initial states (s_0) and transition rules (TR). This CA may properly function if the given problem is fixed. However when the problem is dynamically changing or still not well-defined, this static CA may not be working efficiently.

Without knowing what possible 2D image patterns are wanted, prematurely fixing the type of CA may cause 1) no acceptable solution can be found if a simple CA is used to try to generate a complex pattern, or 2) wasting time and computational power if a complex CA is used for generating a simple pattern.

For the example of a simple CA mechanism, different types of CA have different lengths of TR and the total numbers of different TR, as shown the table 1 below. Acceptable solutions for complex patterns cannot be found with such a simple totalistic binary 2-neighbourhood CA, which has only 8 possible TRs. Conversely it is inefficiency (and sometimes infeasible) to explore a simple checkerboard pattern, through a comparative complex 4-state 4-neighbourhood non-totalistic CA searching within the domain of 4^{1024} possible TR solutions. As design is considered to be an exploration of feasible candidates rather than searching for the optimal solutions, therefore for an efficient exploration of acceptable solutions, some mechanisms for adapting the right CA at specific times are needed. Table 1 compares CA of three types, each having different numbers of states (S) and numbers of neighbourhoods (N).

	General (sS, nN)		2S, 2N		2S, 4N		4S,4N	
	TR length	Total TR	TR length	Total TR	TR length	Total TR	TR length	Total TR
(1) Totalistic(T)	n.(s-1)+1	$s^{n.(s-1)+1}$	2(1)+1 = 3	$2^3 = 8$	4(1)+1 = 5	$2^5 = 32$	4(3)+1= 13	$4^{13} > 6.7x10^7$
(2) Semi-T	s.n.(s-1)+1	$s^{s.n.(s-1)+1}$	2.3=6	$2^6=64$	2.5=10	$2^{10}=1024$	4.13=52	4^{52}
(3) Non-T	$s^{(n+1)}$	$s^{s^{(n+1)}}$	$2^3=8$	$2^8 = 256$	$2^5=32$	$2^{32} > 4.2x10^9$	$4^5=1024$	4^{1024}

Table 1: Comparison of transition rules (TR) length and Total possible TR of different CA types: (1) Totalistic, (2) Semi-Totalistic, and (3) Non-Totalistic, having different states (S) and different numbers of neighbourhood.

Although Cellular Automata has been studied extensively in different aspects including the variant versions from conventional CA (Sarkar, 2000), there is not much work on studying how an evolving CA can further enhance design

exploration and knowledge. A tempting approach is applying the concept of hierarchical GA and Genetic Programming (Koza 1989, 1992) to the problem. However, this hierarchical approach deviates from the problem of abstracting knowledge in design generation.

To tackle this problem, a new approach is introduced in this study to evolve the generative mechanism, the CA type, apart from changing the parameters of a static CA, the transition rules and the initial states. For evolving generative mechanisms, adaptation needs to be supported by the kernel.

There are two approaches to integrate this adaptation ability in a GED-based system. An adaptive ability can be embedded in the generation mechanism to form an adaptive mechanism such that the GA guides how CA evolves. However this limits the possible future extension of the system and restricts the employable evolutionary mechanisms only to CA. Instead, in this study, a separate evolutionary mechanism is used as an adaptive mechanism to evolve the generative mechanism. This approach can be more flexible and extensible for further modification and enhancement of the system such that different internal generative modules and external adaptive modules can be interacted.

In the next section, Piaget's concept of knowledge reconstruction is applied to the CA example, followed by an illustration of how a constraint management (CM) concept can be used as a possible adaptation mechanism.

7.3 Design Knowledge Reconstruction

In this study, the kernel supports the handling of generative mechanisms in design as the core of a knowledge evolution and reconstruction process. Generative mechanisms evolve and adapt to the new situations by adjusting their internal rules, and even the goals. The emphasis in this study is on knowledge development in terms of design objects and design process through Piaget's

concept of assimilation and accommodation. From a cognitive perspective related to Piaget's view discussed earlier on design knowledge, the evolution of knowledge may become possible in the form of assimilation for exploring potential design solutions given the existing knowledge, or of accommodation for synthesising possible design outcomes through proper knowledge reconstruction. Both are often needed in order to tackle new design problems.

In the CA case, certain solutions are obtained based on the knowledge obtained from the past information. Although it is impossible to encounter all possible solutions that can be produced by this specific CA, it is possible to derive from the existing knowledge (of a specific CA) to obtain a domain of all possible solutions. When this situation is adequately managed, an equilibrating state of Piaget's knowledge construction can be reached. There are four possible cases when further new problems are encountered.

First, a new problem is identical to an existed one. In this case old knowledge can be used to handle this new problem without altering the knowledge. Therefore, a new CA is the same as one of those encountered before. What is needed to do is to get the right S_0 and TR back. In the rest of the cases, the new problem is not the same as any one encountered before.

In the second case, an assimilation concept can be applied to re-organising the knowledge with predefined logical structures. Relating to the CA cases, the right S_0 and TR can be searched and obtained, and correspondingly evolved into a new schemata, to generate the right (acceptable) 2D patterns.

In the third and fourth cases, the accommodation concept can be applied to re-constructing the knowledge with some form of restructuring. Relating to the CA cases in the third case, the search domain can be dynamically modified by altering the CA structure, or the type of CA in the later example, as well as its right S_0 and

TR in order to generate the final 2D patterns, while correspondingly evolving the CA structure.

Finally, if all the above mentioned cases still cannot handle the problem with the existing knowledge or the new knowledge generated with the above methods, then it is necessary to turn to other knowledge domain for solving the problem. It may need to re-construct the existing knowledge with some new external knowledge sources. For example, besides CA some other 1D to 2D generative mechanisms exist which might be able to do the job better. It may even be necessary to change the concept, by asking if producing 2D pattern from 1D CA is the right one choice for the new problem. Figure 7.3 illustrates this CA case with the block diagram. Based on this concept, the example of this CA demonstration has been further extended to work with constraint management as an adaptive mechanism.

7.4 Combining GA and CA with Constraint Mechanism (CM)

The term "constraint management" is often referred to the theory of constraints (Goldratt, 1986) in the field of organizational management, for guiding management actions in reaching a goal upon certain constraints imposed. However, in this thesis the constraint mechanism (CM) that handles design constraints is not defined from this perspective. Instead, it is specifically related to handling knowledge constraints that produce design candidates.

In the kernel, transition rules of CA constrain the dynamics of the cells, and thus limit their possible output patterns. Different types of transition rules have different levels of constraints. Totalistic, semi-totalistic and non-totalistic CA are all considered, with the tightest constraints being in the case of totalistic CA, less tight in semi-totalistic CA and further relaxed constraints in non-totalistic CA respectively. A CM module is designed in the kernel to relax or tighten the constraints (the transition rules) of these CA for adapting and exploring the right desired patterns.

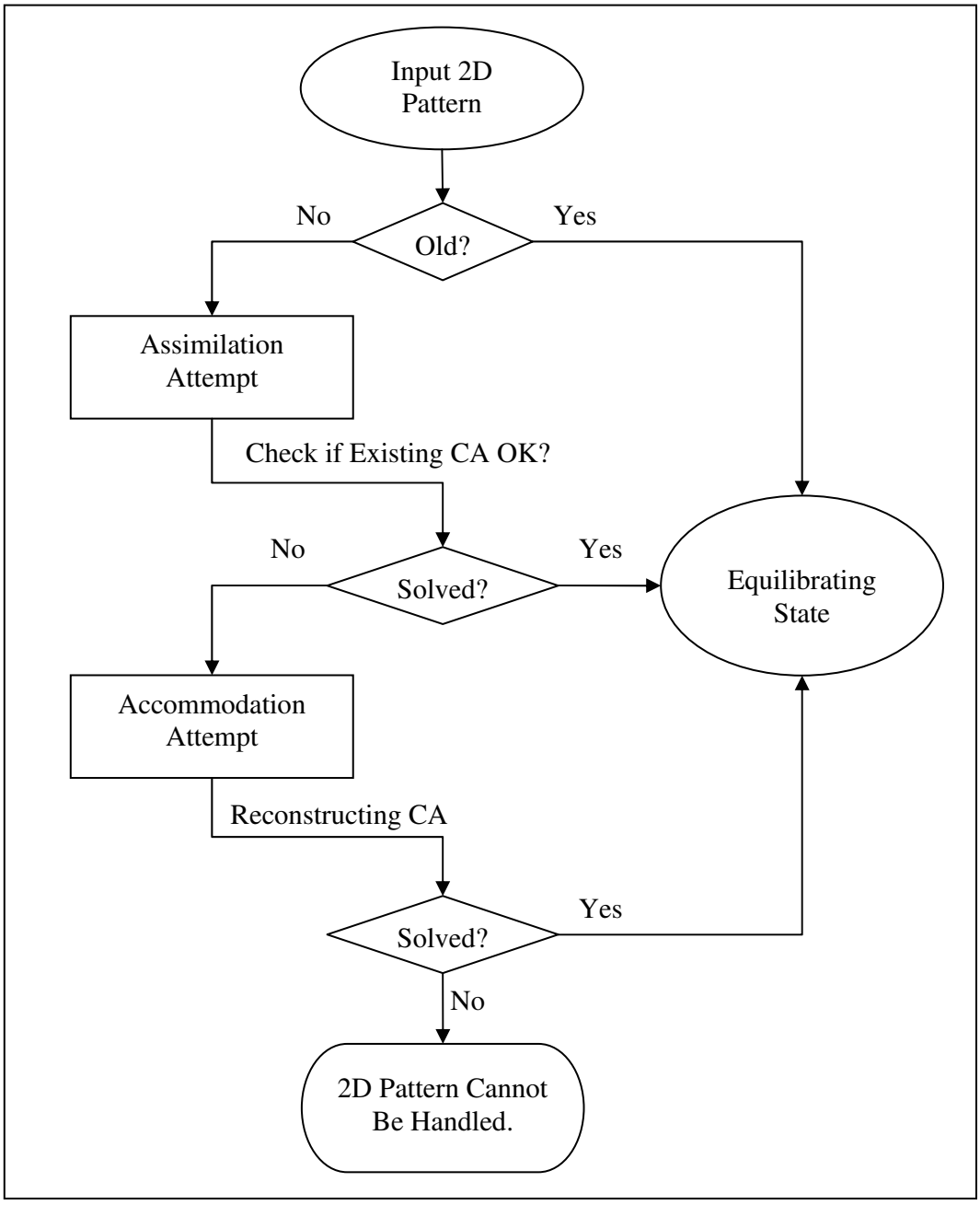


Figure 7.3: A Cellular Automata (CA) Version of Piaget's knowledge reconstruction.

In the above GA-CA case, the knowledge representing the right transition rules and initial seeds to produce the right patterns is evolved and the knowledge of the generating process with GA-CA is emerged. However, the basic structure of the 1D CA (the binary totalistic transition rules) is not changed. The possible 2D images produced by the GA-CA system are still the same within the searchable domain of the original CA.

When the desired patterns are out of this original domain, e.g. some seemly chaotic patterns, no solution can be found no matter how the GA-CA system evolves. In this case, further knowledge evolution is required to handle this. When the knowledge is related to a set of constraints adding upon a set of objects under a certain context, reconstructing the knowledge may be obtained after relaxing and tightening the constraints.

A constraint mechanism can be applied with GA-CA for evolving design generation knowledge to solve the new problem. The role of CM here is to relax and tighten the constraints. In this example the constraints are related to the types of CA. Tightening constraints is related to using a CA that produces limited patterns, while relaxing constraints attempts to obtain a CA that can produce more patterns. The evolutionary process of this CM-GA-CA system can be realized in the psychological perspective, in particular in Piaget's knowledge reconstruction process as discussed in the last section. Figure 7.4 is the schematic diagram showing how this application works and how the knowledge of this CM-GA-CA is developed through the assimilation and accommodation process. The system is operated in the following way:

- *Equilibrating state*

In equilibrating state, the CA is generating and matching the known patterns (the top right corner in Figure 7.4) with the known transition rules and seeds in advance according to the current condition of the evolving knowledge pool (the top left circle in the figure).

- *Assimilation for new patterns*

When a new unknown input pattern is required and causes a dis-equilibrating state, assimilation process will take place. The GA supports this assimilation, by exploring new patterns within the existing knowledge domain. GA attempts to govern the transition rules of a set of CA candidates and their initial seeds and

find the right transition rules and seeds for producing the desired patterns. While the right CA is found, GA returns the finding back to the knowledge pool and the system is then back to the equilibrating state.

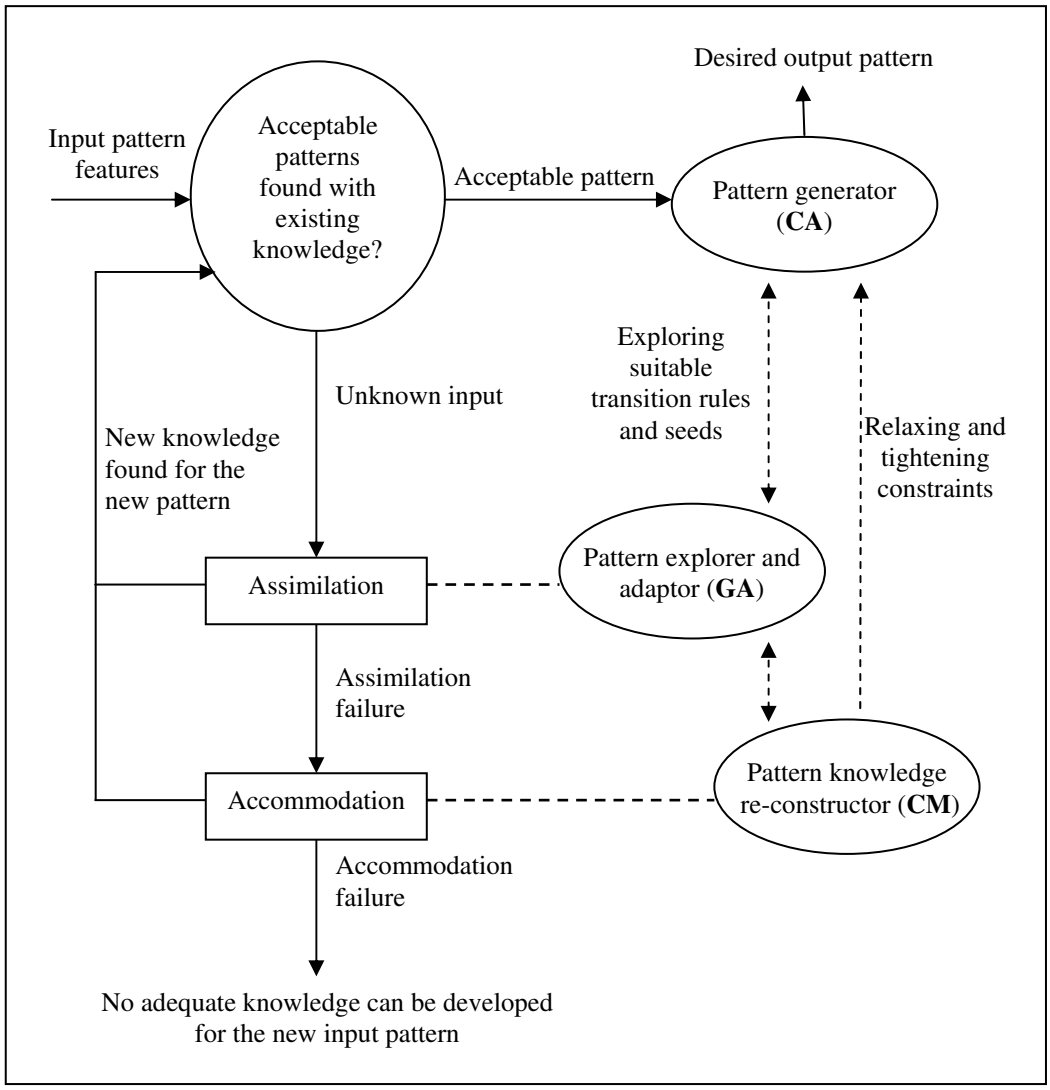


Figure 7.4: The knowledge development of a CM-GA-CA system, through the assimilation and accommodation process with the CA, GA and CM

- *Accommodation for patterns which are out of existing domain*

However, when acceptable CA cannot be found in the existing domain, reconstructing the current design knowledge and its domain of CA is required. Through GA information related to the matching error will be used by CM. CM

relaxes or tightens CA corresponding to GA. The process is repeatedly running until right solutions are found and a right reconstructed CA is done, or the right solution is not found after all possible knowledge reconstruction attempts. When the solutions are found, the system gets back to its equilibrating state. However when they are not, this CM-GA-CA system will not be able to solve the new problems, and no adequate knowledge can be developed for the new input pattern as shown in the bottom left in Figure 7.4. If there is a computational system having more intelligent modules, some potential modification of the existing knowledge may be reconstructed with these modules to solve this problem.

As mentioned earlier, CA is relaxed and tightened from totalistic, semi- totalistic, to non-totalistic. Furthermore, together with relaxing the number of cell states of CA from 2-neighbours to 4-neighbours, more complex patterns are handled as shown in Figures 7.5 and 7.6. In this example, the population of GA is 30 and the probabilities of crossover and mutation are preset as 0.2 and 0.05 respectively.

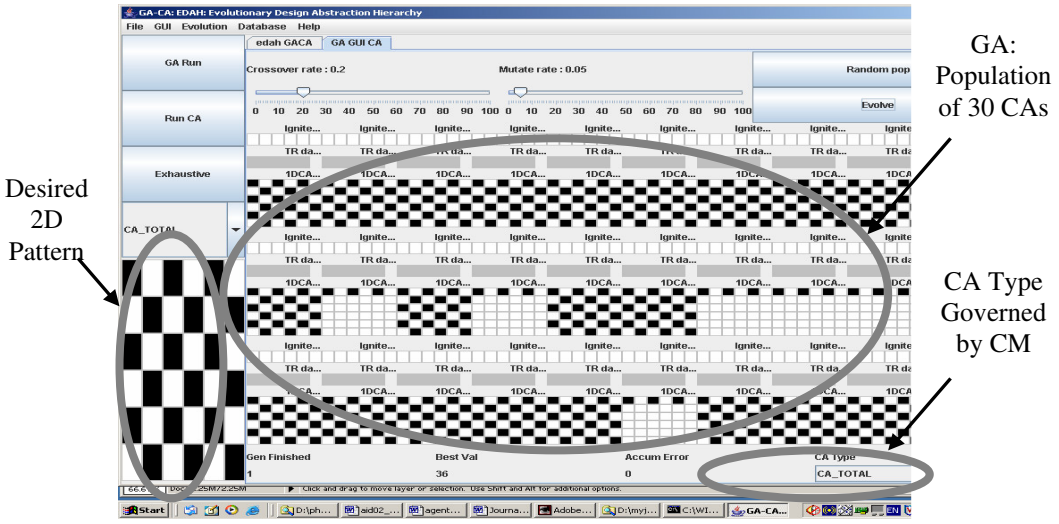


Figure 7.5: Successful matching with constraint tightening in the CM-GA-CA system.

The GACA system has been modified and further enhanced to explore and adapt the right CA form with a prototype CM-GA-CA system based on the kernel developed in this thesis. The system relaxes or tightens CA constraints according to the accumulated errors for finding the best results. In the bottom left of the

Figure 7.5, a checkerboard pattern is to be matched or approximated. When an acceptable CA has been found for closely matching the input pattern, the system will be in its equilibrating state and stop further assimilation or accommodation.

There are two cases that CA will have to be restructured. When the matching is close to a perfect one, the CA will be tightened from a more complex to a simpler one to check if the simpler one can sufficiently and efficiently handle the given matching, as shown in Figure 7.5. Conversely, the second case restructures CA from a simpler one to a more complex one if the matching error is over the acceptable limit. In this case, attempts will be made to see if CA reconstruction with constraint relaxation can lead to a perfect matching or an acceptable approximation, as shown in Figure 7.6 for matching a more irregular pattern (like a character 'A').

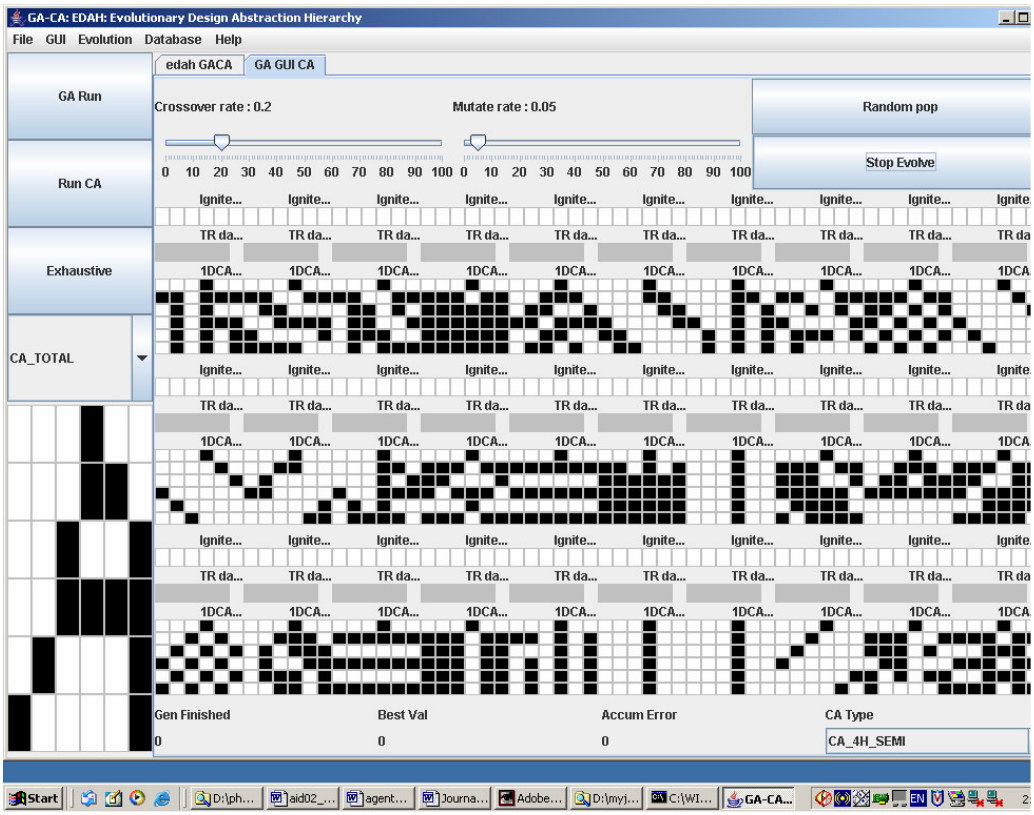


Figure 7.6: Failed pattern matching, even with the most relaxed constraints.

7.5 Issues and Discussions

Although the illustrated example is simple, it shows how simple evolutionary mechanisms (CA, GA, and CM) adapt to new requirements with the evolution of the system knowledge. There are, however, some issues which need to be further discussed.

In this example a constraint relaxation approach is used. With this approach, the generative mechanism is related to a set of constraints. Three types of CA are related to how to relax the constraints. However, in term of computational implementation, further constraints may need to be introduced. Besides the perspective of density (totalistic, semi-totalistic, and non-totalistic) and the number of cell states, there are other perspectives such as 1) static versus dynamic transition rules, 2) synchronous versus asynchronous transition of cell states, and 3) size of neighbourhood.

- *Decomposing constraints*

To make this approach work in a computational system, the evolutionary mechanisms should be constructed and programmed in an appropriate way. In this case, decomposing CA for constraint manipulation is needed and CA should then be implemented in a way that the constraints are partitioned into program segments for CM to split and merge for relaxing or tightening those constraints. In fact, this split-and-merge (or decomposition-and-composition) approach not only works in the natural world such as the splitting and merging of chromosomes, but also in many design tasks. Further study in applying Genetic Programming (GP) in modularising and re-composing generative mechanisms is worth to be investigated.

- *Efficiency and resource*

The knowledge evolution in the system is further closely related to the issues of resource and efficiency. It would be wasting resource with low efficiency when

producing very simple patterns with complex systems having complicated generation mechanisms, while these simple patterns can be generated with very simple rules. In the CA case, seeking the right transition rules and seeds in a totalistic CA is much faster than that in a non-totalistic one although all the patterns generated with totalistic CA can also be generated by non-totalistic CA.

For a 1D binary totalistic CA with 2 neighbouring cells, the number of possible transition rule sets is $2^3 = 8$ while that of a 1D binary non-totalistic one is $2^8 = 256$. The difference between the two types of CA having one relaxing constraint feature is large even in this extremely simple CA, and more constraint differences require huge computational resources in terms of memory and time. Therefore further study is needed on modifying the domain so that not only the newly needed sub-domains can be added but also the unused sub-domains can be subtracted out, for a much more efficient exploration to be achieved.

- *Exact versus approximate solutions*

CA cannot produce all the desired images, and exact solutions are not often found with the limited knowledge. However, with the limited resources, it is almost always the case that knowledge kept by generative techniques, such as CA in this case, can only produce approximated results to some desired optimal patterns which are out of the knowledge domain. Despite of the inexact outcomes, like lossy compression techniques in image compression applications, design exploration is not necessarily to be exact. In fact, one basic criterion for creativity is the unexpectedness. When applied appropriately, this defect can be used as a driving force for achieving surprising results.

- *Forms of adaptation*

In this example both constraint management (CM) and Genetic Algorithms (GA) have different adaptation abilities. While CM has a clear and explicit mechanism to guide how the constraint is to be changed according to the feedback matching error from GA, GA conversely implicitly directs the changes of CA parameters to

achieve the goal (pattern matching) through selection with certain degree of randomness. In comparison, the form of adaptation supported by CM can be treated as an explicit or hard adaptation while that by GA an implicit or soft adaptation. Similar to evolutionary design concept, soft adaptation approach may be more appropriate for handling highly dynamic and diverse problems while hard adaptation for managing comparatively simple and direct tasks.

- *Other representations of structural patterns*

Although this example generates and handles only some simple 2D digital pattern images, the adaptive exploration and generation mechanisms provide a suitable infrastructure to be further extended to other pattern domains. With the kernel, the same 2D pattern can be used to form some structural patterns of plant-form as shown in Figure 7.7. Further work is needed for generating and matching other forms of pattern.

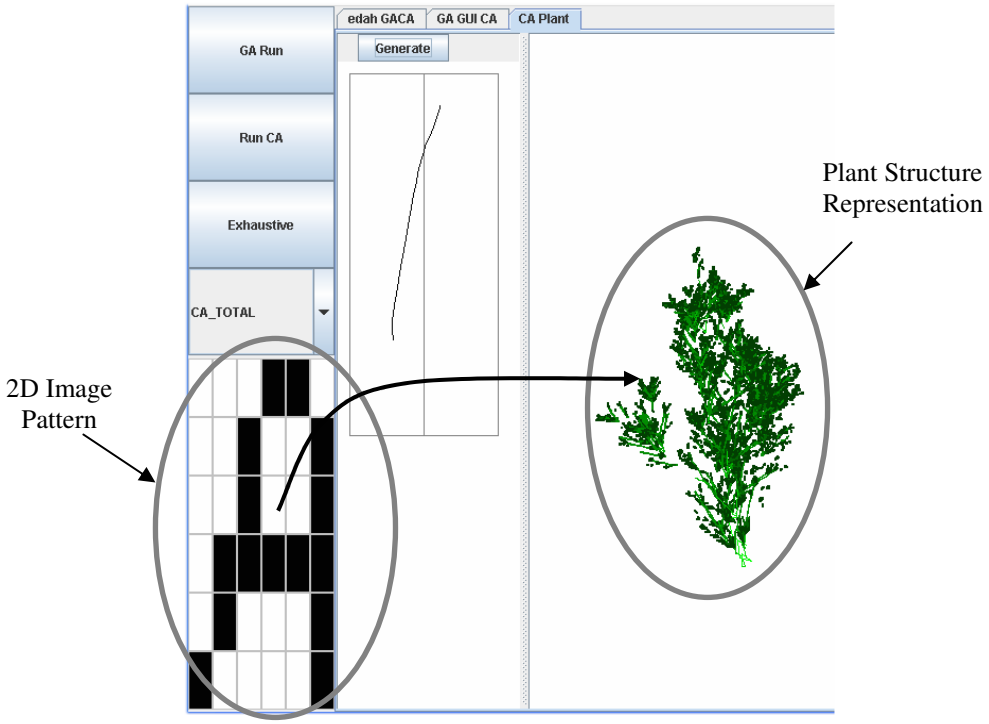


Figure 7.7: Plant-form structural patterns generated with the CM-GA-CA system.

This CM-GA-CA example has shown the exploration and adaptation ability of a computational system with the GED kernel. The simple adaptation is based on

tightening or relaxing CA constraints with CM for dynamically changing the types of CA such that exploration of acceptable CA for a desired pattern can be done more efficiently with GA.

The earlier artificial plant and this CA example can be run automatically without much user intervention. However for practical design tasks, more works should be given to system integration and user interaction. An example in the next chapter demonstrates how the developed GED kernel is integrated with external commercial CAD tool and database to form a more sophisticated system, with more emphasis on human interaction.

7.6 Summary

The issue of design knowledge reconstruction is discussed with the simple example presented in this chapter. Given a specific 1D CA, with specific initial cell states (S_0) and transition rules (TR), it can be shown that specific 2D digital maps or images can be generated. Such S_0 and TR can be treated as a schemata referred by Piaget, while a 1D-generating-2D approach can be regarded as the underneath concept (abstraction and relation). There are different types of CA, including totalistic, semi-totalistic, non-totalistic, neighbourhood, and synchronization. While a fixed form of such structure is used in most research works in generative design, this study emphasizes the evolution of such generation structure or mechanism.

Most conventional research studies use a fixed form of CA in generative design. Design generation is thus based on the transition rules and seeds within this fixed form of CA to produce the desired patterns. However, the basic structure of CA is not changed as well as the possible design domain. This study demonstrates how the GED kernel supports capturing the knowledge of how a design object is generated through evolving generative mechanisms, and thus actually changing

the form of CA in this example so that design generation can be achieved more effectively and efficiently.

In real design, however, more attention should be paid to how a design problem can be converted to a formulation with the necessary information, knowledge and user interaction to allow the problem being explored with the kernel functions. In the next chapter, a more substantial design problem is used to demonstrate the applicability of the generative and evolutionary approach with the necessary components of the GED kernel from a more designer oriented perspective.

Wineglass Design with the GED Kernel

Research on generative and evolutionary design has advanced in the last decade but the application of such technique in a real product design has been limited to isolated reports on experiments with simplified or abstracted examples. To develop a software kernel enabling the application of this technique in a larger and more realistic scale, a design oriented approach is needed. In this chapter, an application of the developed kernel to the design of wine glasses is presented from a designer's perspective. Non-computational issues related to design activities are taken into account in an attempt to examine the feasibility and the limitations of the technological solution proposed in this issue to the problem of fully supporting design activities with generative and evolutionary techniques.

8.1 Wineglass Design

The evolution of a product is closely related to the social, historical, and technological development as the life styles of people change and improve over a long period of time. The design of wine glasses has not been a simple matter and it is an established business in the west. The revolution in drinking glasses had emanated from the two southern English workshops of George Ravenscroft, who in 1675 had discovered how to make LEAD crystal. This gave rise to a whole new style of English glassware quite distinct from intricate Venetian fashions. Increasingly, different glasses were designed and produced to be used specifically for certain wines, and by the end of the 18th century the concept of a uniformly decorated glass service was well established throughout Europe.

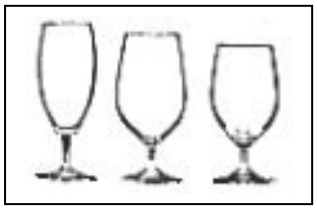
The picture in Figure 8.1 is a typical catalogue in London's Army & Navy store in 1902, after the Victorians further developed the notion of complete range of matching glasses, including finger bowls (Robinson, 1994). After centuries of evolution, techniques and knowledge in wine glass productions have been improved and a variety of wineglass types can be produced. Figure 8.2 shows 3 traditional series of modern wine-glass families from a well-known wine-glass manufacturer (Riedel, 2006). Figure 8.3 shows some unusual wineglasses (Johnson, 1993).

Figure 8.1: Some historical drinking vessels.

8.1.1 A Computational Approach

Computationally, forms of wine glass can be generated and evolved relatively more easily than other products which have complex functional and aesthetic requirements. This example of wineglass generation was originally motivated by the earlier works of Frazer's (top image in Figure 8.4). A simple demonstration program was later implemented in Visual Basic (VB) programming language to

show how conventional GA can be applied to generate a number of 3D models of wineglasses (shown at the bottom of Figure 8.4).



The "Basic" stemware series

The "Wine" stemware series

Two sample wineglasses

The "Sommeliers" stemware series

Figure 8.2: Different series of wineglass families.

Figure 8.3: Some unusual wineglasses.

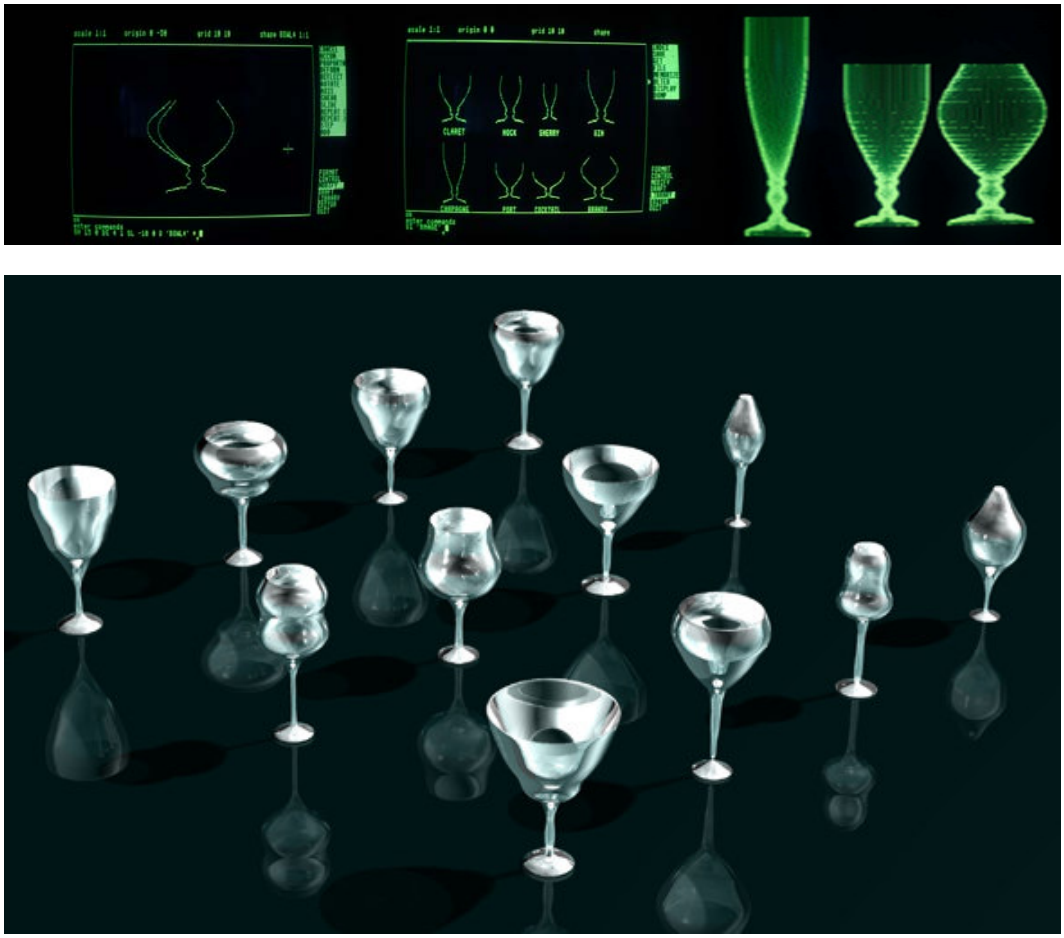


Figure 8.4: An early attempt of 3D modelling wineglasses.

However, this program, once developed like most conventional evolutionary design systems, provided little flexibility for reconstructing the generating process of the wineglass. Its exploration ability is highly restricted to what had been preset by the programmer. With the support of the GED kernel, a better version of this program can be developed which offers stronger support to designers in order for them to explore their designs.

8.2 A Wineglass Design System with the GED Kernel

With little modification to the GED kernel software, a new version of wineglass generation system is developed. This new system with the kernel alone generates wineglasses in a 2D profile representation as shown in Figure 8.5, without integrated to external CAD tools.

8.2.1 Wineglass Design with the GED Kernel

This simple version is hierarchically structured as a demonstration of how different GUI can be linked to interact with each level of the hierarchy. In the middle layer of the hierarchy, a set of templates or seeds represented in a 2D profile form is provided. The geometrical form of these seeds is based on a wineglass series of a famous manufacturer, Riedel.

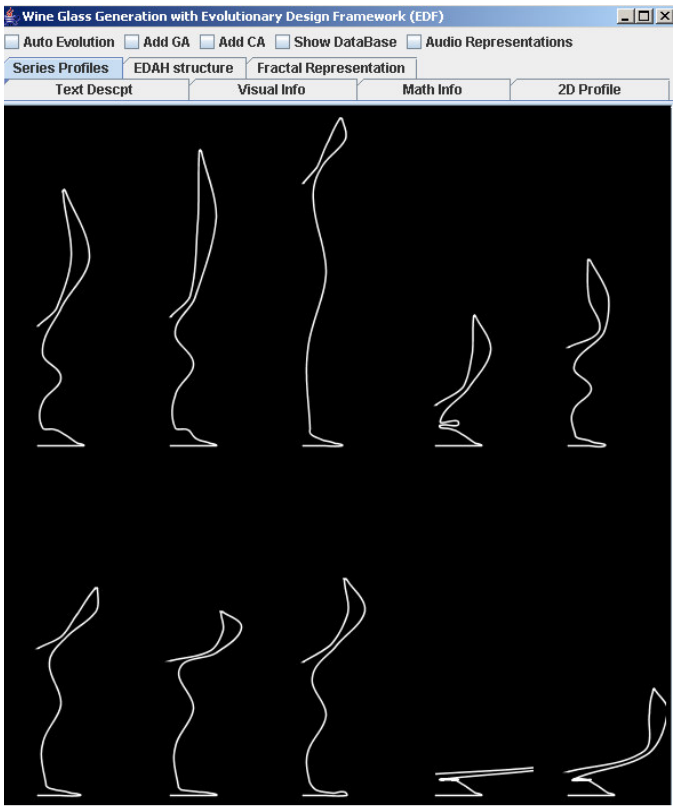


Figure 8.5: The wineglass design system without integrated to external CAD tools, in simple 2D profile representation.

Representation of this example can be formulated as

- e_11_t = { $textualInfo_t$, { }, { }, {e_21_t } }
- e_21_t = { $ImageInfo_t$, {mm_21_t }, { e_11_t }, {e_31_t } }
- e_31_t = { $MathInfo_t$, {mm_31_t }, { e_21_t }, {e_41_t } }
- e_41_t = { $2Dprofile\_41_t$, {mm_41_t , $Dbtemplate\_41_t$ }, { e_31_t }, {e_51_t , e_52_t , ..} }
- e_5i_t = { $2Dprofile\_5i_t$, {mm_5i_t , $Dbtemplate\_5i_t$ }, { e_41_t }, { } }

where *mm* represents a manual mechanism in a form of manual evolutionary mechanisms

8.2.2 Limitations without 3D Manipulations

However, functions supported by this simple system are limited, without the support from commercial 3D solid modelling tools. For example, the representation of wineglass in this generic GED kernel system is limited to 2D geometrical graphical representation. To improve this, the kernel can be either further developed to a more complex software system that is resource consuming, or it can be integrated with external tools to make use of their advanced features such as 3D geometrical modelling. In the following example, the kernel is integrated with an external CAD tool and shows that it is feasible and applicable to product design applications.

8.3 An Improved Version

In this improved version the GED kernel (GEDK) is integrated to a commercial CAD tool (MicroStation) and external CAD functions are utilized. The block diagram of this GEDK-embedded system is shown in Figure 8.6 and a captured image of the system is shown in Figure 8.7.

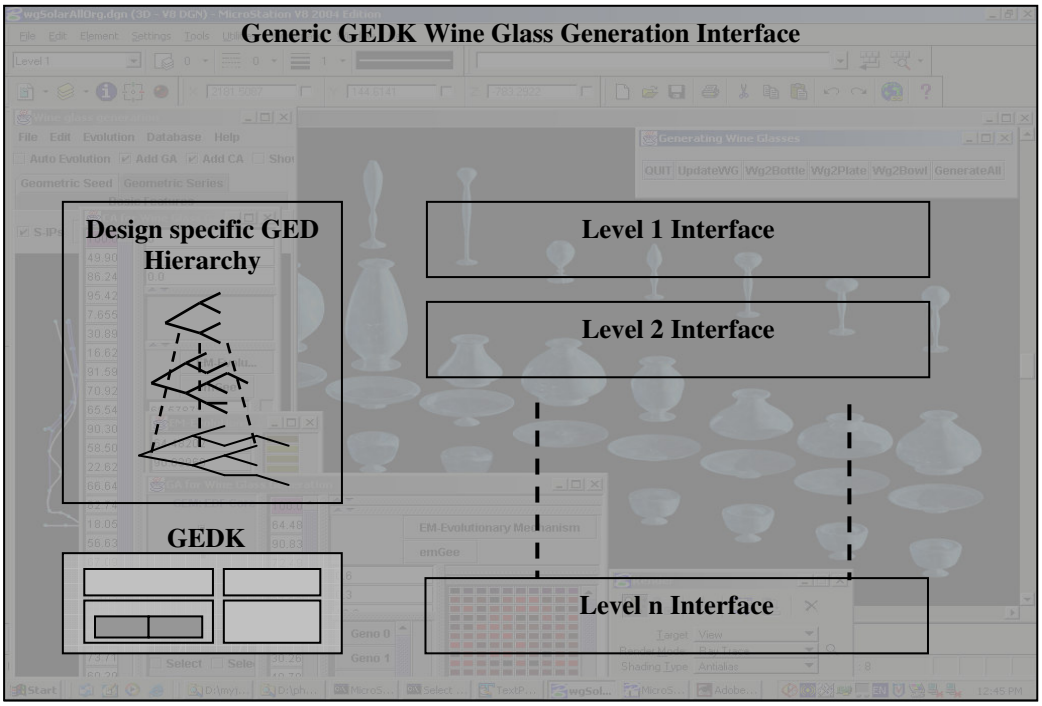


Figure 8.6: The block diagram of the wineglass generation system.

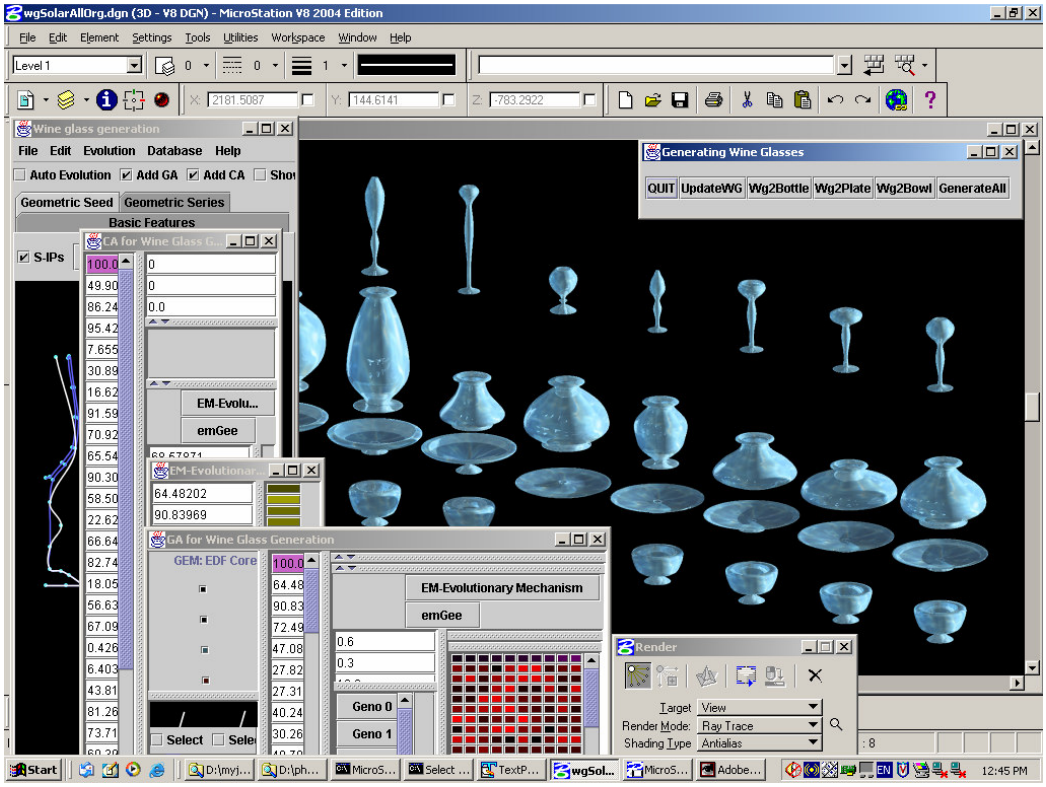


Figure 8.7: The GEDK-embedded system for wineglass generation.

With such GED kernel (GEDK) embedded, the application system can be interacted with designers or users at different abstraction levels and different results can be explored as shown in Figure 8.8. In this application, there are five levels in the GED hierarchy. The element in the first layer represents the root seed of the whole family, while that at the second layer captures the basic geometric feature of desired wineglasses. The element in the third layer consists of more than one mechanism, including 2D profile interface and a GA. At the moment, only artificial or manual selection is used in this GA process, while more research work are required to study what objective functions are needed and can be formulated for fully-automating the GA selection process.

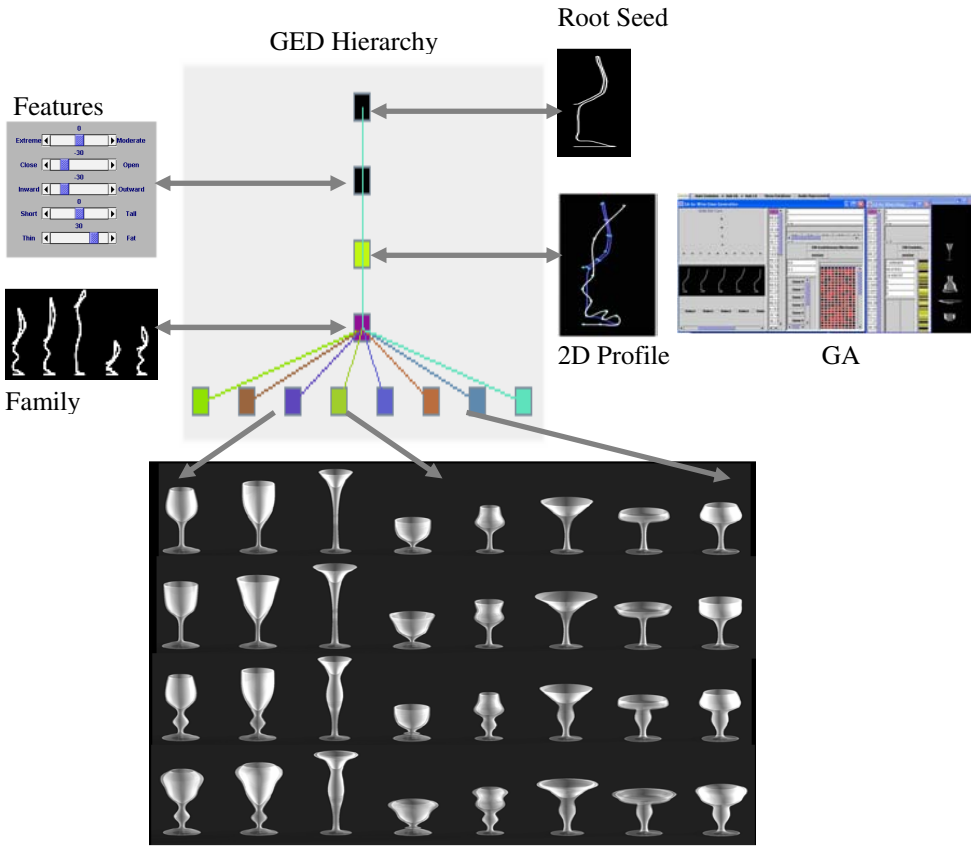


Figure 8.8: Design Generation, evolution and interaction with the GEDK-embedded system.

The best candidate from GA is generated, selected and directed to the fourth level. The evolutionary element at the fourth level is embedded with a family variants

loaded from a database. The family includes eight members for serving different wines: Riesling, Champagne, Underberg, Water, Cognac XO, Martini, Moscato and Sauternes. The final family members, retaining all inherited styles and features from higher abstractions, are generated and fed to the fifth layer, at which they can be transformed to 3D models by the integrated CAD tool.

In this example, the GED kernel is integrated to an external CAD tool and the system can then be formally represented and formulated as

- e_11_t = { $WG\_ProfileSeed_t$, { mm_11_t }, { }, { e_21_t } }
- e_21_t = { $FRFeatures_t$, {mm_21_t }, { e_11_t }, { e_31_t } }
- e_31_t = { $2Dprofile_t$, {mm_31_t, GA_32_t }, { e_21_t }, {e_41_t } }
- e_41_t = { $Best2Dprofile_t$, { $Dbtemplate\_41_t$ }, { e_31_t }, { e_51_t , e_52_t , ..} }
- e_5k_t = { $WG\_FamilyProfile\_5k_t$, { CAD_tools }, { e_41_t }, { } }

With this system, designers can manipulate wineglasses at various abstractions: from the abstract descriptive features of wineglasses in the top-right window, the 2D profile of the wineglasses, the add-on evolutionary mechanisms including a Genetic Algorithm, to the final 3D models of the generated series. A large number of alternative design solutions can then be explored and generated, as shown in Figure 8.9.

8.3.1 From Seeds to Relatives

When further modification of this system is made for generating wineglasses with different genes or "seeds", various design solutions ("relatives" or "species") can be generated with the same evolutionary hierarchy developed for wineglass generation. Figure 8.10 shows the results obtained from an extended example of this case. With a simple modification, corresponding series of bottles (the second row in the figure), plates (the third row) and bowls (the last row) having same inheritance from a wineglass family (the first row) can be generated.

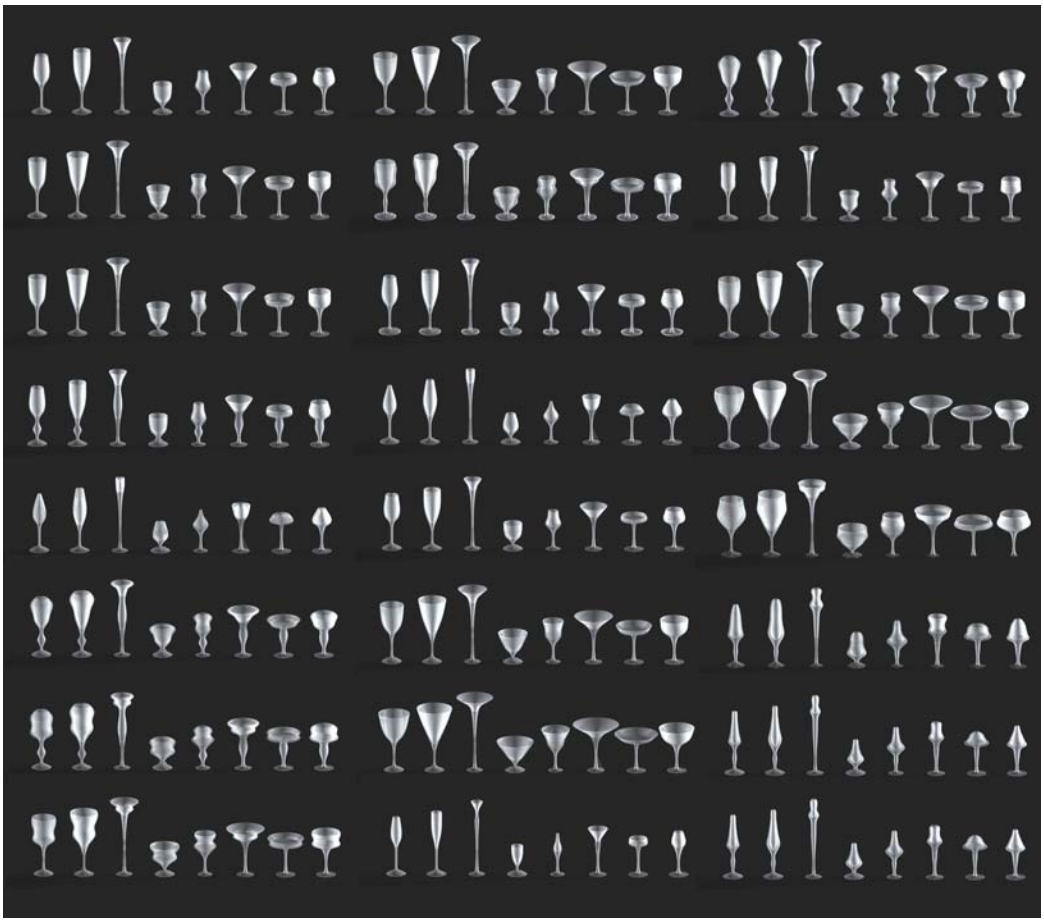


Figure 8.9: Different wineglasses can be generated with the system.

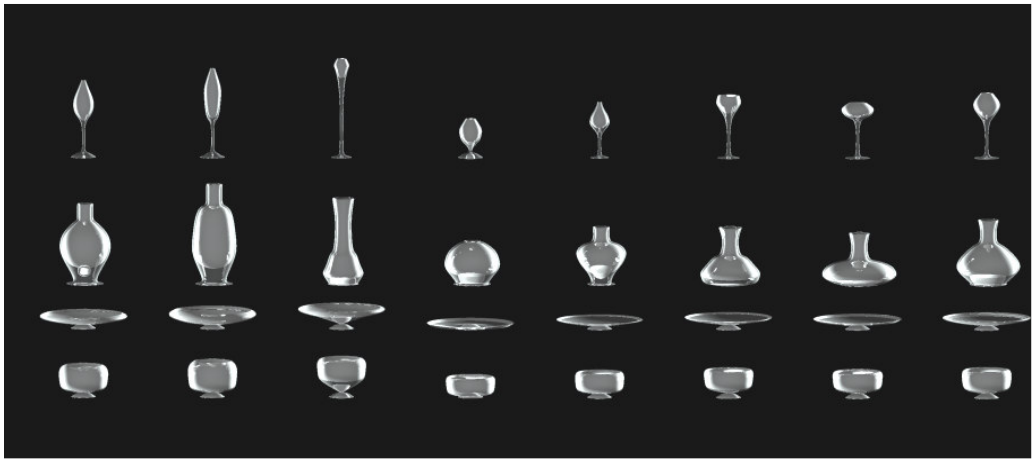


Figure 8.10: Seeds produce different species with the GEDK-embedded system.

In this case the GED hierarchy can been formulated as

- e_11_t = { $Relative\_ProfileSeed_t$, { mm_11_t }, { }, {e_21_t } }
- e_21_t = { $FRFeatures_t$, {mm_21_t }, { e_11_t }, {e_31_t } }
- e_31_t = { $2Dprofile_t$, {mm_31_t, GA_32_t }, { e_21_t }, {e_41_t } }
- e_41_t = { $Best2Dprofile_t$, { $Dbtemplate\_41_t$ }, { e_31_t }, {e_51_t , e_52_t , ..} }
- e_5k_t = { $Relative\_FamilyProfile\_5k_t$, { CAD_tools }, { e_41_t }, { } }

Close "relatives" (bottles, plates and bowls) of the wineglass "family" can easily be generated with the GED system. Many potential design solutions of these close relatives can then be explored as shown in Figure 8.11.

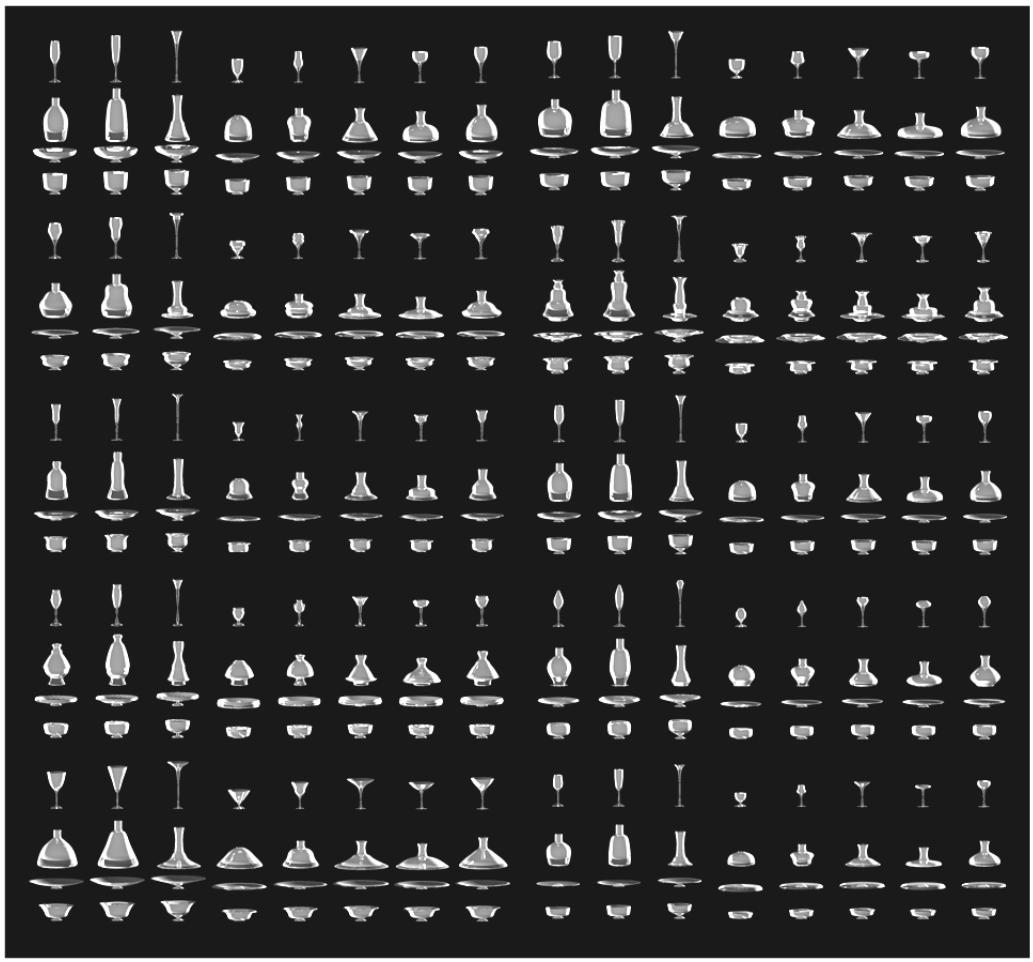


Figure 8.11: Some example results, generated with the GEDK-embedded system.

This design example presents an application for generating a family of wineglasses and their "relative" utensils when the GED kernel is integrated with commercial CAD tools. The emphasis in this example has been given on developing a design system in a designer's perspective with the GED kernel and integrating the kernel with external design tools available for supporting practical design tasks. Together with the previous two design examples in Chapters 6 and 7, these three examples have shown the feasibility and applicability of the GED kernel for supporting design in various aspects which will be discussed further in the next session.

8.4 Discussion and Evaluation

Three chapters in this Part III have described and illustrated the generative and evolutionary design (GED) kernel on how it supports the development of generative and evolutionary design systems, through the introduction of three different design examples. In the first example presented in Chapter 6, patterns of plant structure are generated with the GED kernel as a supporting tool. The system starts with a structural root gene or seed, which self-replicates itself to a number of children at a lower level of abstraction in the GED hierarchy through an attached self-replication (SR) mechanism. This self-replication proceeds automatically generation by generation. The elements at lower abstraction levels represent the micro details of the artificial plants, while the higher ones affect their macro structure.

This first example illustrates how the developed GED kernel supports and enhances a more flexible exploration of generating potential patterns with manipulation of design representations at different abstractions and evolutionary generative mechanisms. However, without proper adaptation ability supported in the system, such generation activity may behave as an aimless and inefficient exploration.

The second demonstration example presented in Chapter 7 shows how the kernel supports such adaptation. In this example Genetic Algorithm (GA) is used as an evolutionary mechanism that handles the global exploration and adaptation of a set of elementary elements, each of which is attached with another evolving generative mechanism – Cellular Automata (CA). GA provides a mechanism for adaptation and exploration through its objective matching function. CA contains the design knowledge of generating 2D pattern formation with its transition rules and seeds. Furthermore, a constraint mechanism (CM) is used to restructure such knowledge of generating 2D pattern formation embedded in the system, in terms of design objects and their generative design process.

This example focuses on showing how design knowledge is evolved and adapted for generating and matching desired patterns with the GED system. However, in real design tasks much attention is required for knowing how to convert a design problem to a formulation with the necessary information, knowledge and user interaction to allow the problem being explored with the kernel functions. In Chapter 8, a more substantial design problem is used to demonstrate the applicability of the generative and evolutionary approach with the necessary components of the GED kernel from a more designer oriented perspective.

To develop a software kernel enabling the application of this technique in a larger and more realistic scale, a design oriented approach is needed. An application of the developed kernel to the design of wineglasses is presented from a designer's perspective in Chapter 8. Non-computational issues related to design activities are taken into account in an attempt to examine the feasibility and the limitations of the technological solution proposed in this issue to the problem of fully supporting design activities with generative and evolutionary techniques.

These three application examples have demonstrated the feasibility and applicability of the GED kernel in supporting computer-based design systems in various aspects, particularly in the issues of dynamics of GED hierarchies,

knowledge of design generation, abilities in design exploration and adaptation, design representation and interaction, and the system development and integration with external design tools.

8.4.1 Dynamics of GED Structural Hierarchy

The three examples have shown that the GED kernel supports an evolving structure with multiple design representations in a hierarchical form, and provides mechanisms for manipulating different design abstractions at different design stages. Compared to traditional evolutionary design approach in which design representation is fixed and lack of flexibility once preset, such evolving structure changes dynamically according to the generative mechanisms attached to the evolutionary design elements and supports a wider exploration of potential design candidates. Three examples have different degrees of evolutionary dynamics, which can be related to design cases that require a higher degree of innovation and creativity (e.g. the first plant example) to a lower degree one (e.g. the wineglass example).

In the first example the artificial plant generation system supports a fully dynamic GED hierarchy that forms an evolutionary structure. This simple example illustrates how the developed GED kernel supports and enhances a more flexible exploration of generating potential patterns with a fully-automated system. The system keeps evolving the design elements as well as the overall hierarchical representation of design objects at different abstractions. Although this automatic evolving structure exhibited in an aimless way without human intervention, this example demonstrates the ability of the kernel in supporting structural evolution of the multiple design representations in a hierarchical form. As discussed in earlier chapters in Part II, alternation of design representations at more abstraction forms often leads to much radical and innovative design, which is particularly important in new design tasks that emphasize creativity and new design features.

Compared to the first example, the second 2D image pattern example emphasizes on how the GED kernel based system supports knowledge adaptation in design generation. In this example the overall hierarchical representation is a static structure as a whole, while the generative and evolutionary mechanisms demonstrated with Cellular Automata, Genetic Algorithms, and Constraint Management techniques are manipulated to improve the effectiveness and efficiency in design exploration. Therefore, each evolutionary element attached with a CA mechanism changes while the overall GED hierarchy does not.

In the last example, the system supports a fixed GED hierarchy of wineglass design with external design tools. While our internal GED hierarchy itself in this example can be dynamic, the system for this example is developed for handling a comparatively mature practical design case with a static GED design representation. The integration of the kernel with external CAD tool, database, and other computational modules imposes difficulties to produce a flexible system which can operate in a much dynamic way as the last two examples.

8.4.2 Major Generative Mechanisms

In this study much emphasis is also given to the ability in exploring and generating designs, through various computationally generative mechanisms. These generative mechanisms are used to retain the knowledge of how design objects are generated, from a more abstract form to a less one.

In the first plant example, a simple self-replication (SR) mechanism is used. This SR mechanism supports simple reproduction process having a mechanism of generating details similar to a "fractal" way. The system then generates less abstract forms from a more abstract one at different levels of abstraction in the GED hierarchy in a self-symmetric manner. This simple SR mechanism illustrates the ability of our GED kernel in dynamically evolving the GED hierarchy which represents design in multiple abstractions.

In the second 2D pattern matching example, 1D binary Cellular Automata (CA) is applied for producing seemingly much complex pattern with a set of simple transition rules. This generative mechanism in this example shows how the knowledge adaptation of design generation is achieved and affects the effectiveness and efficiency of exploration, through adapting suitable transition rules and initial states of the CA within the GED kernel.

In practical design cases as the third wineglass example, manual manipulation and designer interaction are important. In this example human interfaces for manipulating 2D curve and geometrical features of the wineglass shape are provided. Generative mechanisms then rely on human interaction, and computational generative modules are comparatively fixed. For further enhancements, appropriate and suitable computational generative mechanisms should be investigated and applied to specific design tasks, as discussed in next chapter.

8.4.3 Exploration and Adaptation Abilities

Three examples also demonstrate how the GED kernel (GEDK) enhances the exploration and adaptation ability of GEDK-embedded system for supporting generative design in an evolutionary manner. In the exploration aspect, the GEDK-embedded systems provide a suitable mechanism to explore potential candidates without limiting to a fixed or preset domain. However, without proper knowledge adaptation such exploration would be aimless, inefficient and ineffective. The three examples presented in this Part III have illustrated how the GED kernel supports design with different degrees of explorative and adaptive abilities.

In the plant example, artificial plant structures are generated with pure exploration without adaptation ability, although implicit human adaptation is supported through manual interactions. In this example, exploration can be exhibited more freely to produce different hierarchical forms, while the underneath meaning of why these forms should be is not retained or captured without any adaptation mechanism.

In the second 2D pattern example, the exploration mechanism for seeking appropriate CA to generate and match the right 2D patterns is governed by a GA, while a Constraint Management module further controls the adaptation of the CA types and improves the effectiveness and efficiency of the exploration. Even though such mechanism is not complicated, it shows how the GED kernel provides the fundamental feature in collaborating exploration and adaptation activities within a design supporting system, and enhances the design generation more effectively and efficiently.

In the third wineglass example, the importance of the involvement of designers in the system has been shown. Without undermining the ability of the GEDK-embedded systems, human and designer interaction is considered in the exploration and adaptation process through human selection in a GA process. Such semi-automatic exploration and adaptation process can be effective and efficient when the computational module is carefully tailored.

In practical design tasks, formulating appropriate computational modules for exploration and adaptation incurs a lot of works and problems to be solved. Without awaiting such tremendous works and limiting the application of GED kernel, designer exploration and adaptation with human interaction are supported. The wineglass example shows how this issue is tackled and how the integrated system makes use of sophisticated functions provided by these externally developed design supporting tools, such as 3D modelling and rendering.

8.4.4 Design Representation and Interaction

The examples have further demonstrated how the GED kernel supports design objects to be represented with multiple forms which are situated at different abstraction levels of the GED hierarchy. In this representation and interaction issue, the GED kernel provides a basic generic representation and interface of design objects. This can be used to directly develop computational systems for supporting some simple design tasks such as the first example, while it can also be integrated easily with external CAD tools which further support more sophisticated design representations and interactions.

In the plant example, 2D geometric structure in a plant-like form is generated with the GED kernel based system. With human interaction that alternates the geometrical shape of the plant at different levels of details, different forms can be produced. In the 2D pattern example, the final representation is in a form of 2D digital grids, which is then directly mapped to 2D image patterns. The generated 2D image patterns can also be visualized and represented as artificial plants.

In the wineglass example, it shows how the GED kernel alone supports designing wineglasses and represents design products such as wineglasses with simple 2D geometric profiles. With the GED kernel alone, these design objects can also be mapped to and represented with 2D plants or audio sound patterns. Furthermore much sophisticated representation can be provided when the GED kernel is integrated with external commercial CAD tools. When the GED kernel is integrated with external tools, much sophisticated design representation such as 3D geometric models and interactions provided by external tools can be used in the integrated design system.

8.4.5 System Development and Integration

The degrees of sophistication of three example design systems are deviated from a simple and generic one (the first artificial plant) to a much sophisticated one (the wineglass design system). While the simple plant example has shown how a generic GED kernel (GEDK) alone can develop a general system which supports designing in a generative and evolutionary approach, the comparatively sophisticated wineglass example has demonstrated the feasibility and applicability of applying our kernel for developing a more sophisticated system with external computational design tools when needed.

In the first plant example, the generation and exploration of artificial plant-like patterns is supported with the GED kernel alone. The graphical user interface (GUI) of the system is solely developed with the generic GED builder of the kernel such that the GED hierarchy of the system is initially structured with a single root seed attached to a simple evolutionary self-replication (SR) mechanism. The generic interface provided by the kernel is used for visualizing, manipulating and interacting with the evolutionary elements (and their attached SR mechanisms) through the generic interfaces of the kernel.

With some further development works, a more application-specific system for the 2D pattern generation and matching application with adaptation ability is developed in the second example. As this example concentrates on demonstrating the adaptation ability of the GED kernel, interfaces for comparatively in-depth manipulation of computational modules involved in this adaptation are developed.

For developing design systems in a much design-oriented approach, the sophistication of the systems in supporting designer manipulation and interaction in different design aspects should be emphasized, including making use of and integrating with externally developed CAD tools. The third wineglass example illustrates how the GED kernel is integrated with external design tools to develop

a design supporting system. This integrated system supports designing tasks with external design tools more effectively and efficiently, and makes use of those functions supported by these external computational tools such as sophisticated 3D geometric modelling and rendering functions.

However, further issues in this integration approach need to be handled. Before using this integration approach, it is important to know the availability of technical supports in integrating external design tools with the GED kernel, such as the source code and software development kits of the external tools. It is also worth to question if the efficiency and effectiveness of this integration approach is higher than those of developing a new system based on the GED kernel alone such as the first two examples.

8.5 Summary

Results of three demonstration examples presented in this and the last two chapters show that it is feasible and applicable to use the kernel as the core architecture of computational systems for supporting generative and evolutionary design. The kernel further improves the generative, explorative and adaptive ability of the computational design supporting systems with the kernel in producing potential design solutions efficiently. Discussion and evaluation of these three examples in various aspects have been given in the last section. The comparison of three examples discussed in the last section is summarized as given in table 2.

Despite of those discussed issues in how the GED kernel supports in various aspects, there are many other properties and factors to be considered in practical designing, including design science, psychology, culture, material, production technology, aesthetics, and cost. For example, the wineglass example concentrates on its geometrical form, in terms of how this form can be generated and manipulated. In this case, abstract representations at different hierarchical

levels are mainly related to geometrical issues. Supporting other representations of wineglasses, such as those related to aesthetics and cultural issues, can further sophisticate the system so that wineglass design can be represented and manipulated with other important features in different aspects.

Examples / Issues	Plant	2D Image Pattern	Wineglasses
GED Hierarchy	• Dynamic structure	• Static structure	• Static structure
Major Generative Mechanism	• Self-Replication (SR) • Static	• 1D Binary CA • Dynamic	• 2D curve manipulation • geometric feature manipulation
Exploration Ability	• Auto aimless-exploration • Manual intervention	• GA (pattern matching and approximation) • Manual intervention	• GA (human judgement) • Manual intervention
Adaptation Ability	• None	• GA (GM para: TR, S0) • CM (GM types: CA)	• GA: human selection
Representation and Interaction	• 2D geometric structure • Plants	• 2D digital grid • 2D image pattern	• 3D geometric model • 2D Plants • Audio
System Development & Integration	• GED Kernel • GED Builder • Generic GUI	• GED Kernel • GED Builder • Problem specific GUI	• GED Kernel • GED Builder • Additional GUI • External CAD

Table 2: Comparison of three demonstrative examples.

Furthermore only artificial or manual selection is used in the GA process of the wineglass design system at the moment, while more research works are required to study what objective functions are needed and can be formulated for further automating this GA process. More investigations and studies are also worth to be taken, and some further directions will be discussed in the next conclusion chapter.

Conclusions

A computational Generative and Evolutionary Design (GED) kernel has been formulated and developed for supporting design in this study. Design objects are represented with multiple abstractions, and are evolved from a more abstract form to a less abstract one through the exploration process supported with the kernel. Knowledge of design generation can also be adapted in the form of generative mechanisms in the kernel, so that possible generation process of potential design candidates can be captured.

When computational design supporting systems are integrated with this GED kernel (GEDK), a large number of potential designs can be explored more efficiently and effectively, through evolving the initial abstract design objects together with their generative mechanisms. With the adaptation methods attached to the kernel, knowledge of design object generation can be reconstructed with those evolving generative mechanisms. Explorative and adaptive abilities of these GEDK-embedded applications can then be enhanced.

To demonstrate the feasibility and applicability of the kernel for supporting computational design tasks, three example of GEDK-embedded systems are developed and evaluated in various aspects. The results show that it is feasible and applicable to use the kernel as an architectural core of computational systems for supporting generative and evolutionary design. The generative, explorative and adaptive ability of such GEDK-embedded computational design applications is improved in producing potential design solutions effectively and efficiently. Before concluding the thesis through revisiting the main objectives and

significance of the study, the coming section recaptures and summarizes the main topics of this thesis.

9.1 A Summary of Research Conducted

This study focuses on how the evolutionary design process can be more efficiently handled, with the GED kernel that supports the evolution of design objects in a design oriented manner as presented in Chapter 4. In particular, the GED kernel is used for exploring and adapting different potential design objects with different ways of generating them through the evolving generative mechanisms attached to a hierarchical representation scheme.

With the formulation of this hierarchical representation and evolutionary mechanisms, the knowledge that generates design can be captured and adapted with different levels of abstraction in order for designers to explore design solutions in a gradual and general-to-specific manner. Therefore the kernel and its underlying model concern themselves not only with what the design solutions are, but also with how they are explored. The formal representation of the GED model has been presented and discussed in Chapter 5 which provided the basis for the implementation, testing and evaluation.

In the thesis, the details of how the kernel supports modelling design in a form of evolutionary hierarchy, constructed with interlinked evolutionary elements and mechanisms are described. The GED kernel has been examined with three examples which have shown the feasibility and applicability of the kernel for design tasks. In Chapter 6 the first example illustrated how a simple self-replicating seed can automatically evolve into a more complex plant-like structure in a multi-level hierarchical form with the kernel alone.

Example two in Chapter 7 showed how the kernel enhanced the explorative and adaptive ability of a 2D pattern generation and matching application with Cellular

Automata, Genetic Algorithms, and Constraint Management as the main evolutionary mechanisms. Finally the kernel has been used to support a product design task. In the last example presented in Chapter 8, the kernel was integrated with an external CAD tool to generate a variety of wineglasses in a short time with user interaction. Other utensils having similar styles and features of wineglasses can also be generated by replacing different product seeds.

The results of these examples have been evaluated and have shown that the kernel can improve the flexibility and efficiency of generating, exploring and adapting potential designs. To conclude this study, the main objectives and the significance as outlined in Chapter 1 are revisited in the next section in order to give a detailed account of how the research contributes to the field of generative and evolutionary design.

9.2 Objectives and Significance Revisited

The major objective of this research study is to examine the feasibility and applicability of a generic computational kernel, i.e., the Generative and Evolutionary Design (GED) kernel, to be used as an architectural core of computer-based design supporting systems. The main focus of this research is on the formulation, implementation and evaluation of this computational GED kernel (GEDK) which supports 1) modelling design object and design process in a generative and evolutionary manner with a structured representation, 2) capturing and adapting knowledge on how design objects can possibly be generated, and 3) simplifying the process of mapping design applications to a generative evolutionary system. In an integrated way, the GEDK-embedded systems can enhance the exploration ability on potential designs more efficiently and effectively than normal CAD systems which do not have a generative and evolutionary kernel.

9.2.1 GED Model for Dynamic Design Object and Process

Before the Generative and Evolutionary Design (GED) model can be formulated and the associated GED kernel is computationally implemented, the nature of design and how to support design with computational techniques have been examined. Such GED model focuses on providing a foundation for the application of generative and evolutionary techniques in design domains, which is verified with the examples of realistic scales. In particular, much emphasis is given to evolutionary computing and structured representation that improve the efficiency in using computer-based design supporting systems. Without examining and understanding the generative and evolutionary nature of design, formulating and implementing a computationally kernel of such model will not be feasible.

Following the initial findings and literature reviews presented in Part I, a GED model was developed and the GED kernel was then formulated in the way as introduced in Part II. Such modelling and formulation of the GED kernel particularly focused on the issues of dynamics of GED hierarchies that support representing design objects and their generation process, knowledge of design, abilities in design exploration and adaptation, design representation and interaction, and the system development and integration with external design tools.

As discussed in the evaluation section 8.4 with three demonstration examples, it has shown that the GED kernel supports an evolving structure with multiple design representations in a hierarchical form. Compared to traditional evolutionary design approach in which design representation is inflexibly preset and fixed, such evolving structure dynamically changes the design data according to the attached generative mechanisms. This dynamic nature of evolving design reflects an exploration process in which potential design candidates are generated. The kernel further provides the mechanisms for manipulating different design abstraction forms at different design stages.

Several difficult issues were encountered during the research, given the complexity as demonstrated in the three examples used to verify the system kernel and its underlying representation. The formation of a suitable GED hierarchy for different design tasks involves a generalization at both data structure level and design process level. While generating artificial plants can be achieved with a fully dynamic GED hierarchy, the overall hierarchical representation has to be adjusted when a comparatively more constrained data structure is involved in the example of product design applications such as the wineglass example.

9.2.2 Knowledge Exploration and Adaptation of Design Generation

The main objective of this study is also associated with the issue of how to capture the knowledge of design generation in which design solutions are explored, captured and adapted. Through a knowledge adaptation mechanism supported by the kernel, possible generative methods of design can be utilized to retain the data as well as the process of evolving them. To achieve this, the emphasis is given to the ability of the system in exploring and generating designs, through various computationally generative mechanisms. These mechanisms can retain the knowledge of how design objects are generated, from a more abstract form to a less one. Some evolutionary computation methods were integrated into the GED kernel so that design assistance in terms of adapting design solutions and exploring design alternatives can be provided to designers.

Three demonstration examples presented in Part III have shown how the GED kernel (GEDK) enhanced the exploration and adaptation ability of GEDK-embedded system for supporting generative design in an evolutionary manner. In the exploration aspect, the GEDK-embedded applications provide suitable mechanisms to explore potential candidates without limiting to a fixed or preset domain. With such adaptation ability, the design exploration process is more focused, efficient and effective.

The second pattern matching example presented in Chapter 7 in particular has demonstrated how the kernel supports this adaptive activity. However, formulating fully-automated computational modules for design exploration and adaptation in product design tasks, such as the third wineglass example, are more difficult to formulate and support without the integration with external systems that deal with the 3D construction of components and assemblies. The wineglass example was developed to find out how the kernel can be integrated with external kernel to evolve complex designs. However, developing such complex computational modules in specific design domains requires better understanding of designers' knowledge and the way in which they explore design solutions, especially at the conceptual design stages. This research has provided a foundation for further exploring this issue of generalizing product design process with extensive design specific knowledge, in order to fully utilizing the generative and evolutionary mechanisms built into the kernel and the hierarchical representation.

9.2.3 GEDK-Embedded System Development

Besides the issues of knowledge adaptation, the development issue of GED kernel-embedded systems has also been studied. For simplifying the process of mapping design applications to a generative evolutionary system and enhancing the exploration ability to explore potential designs more efficiently and effectively, the feasibility and applicability of the developed GED kernel to computational design systems are tested and evaluated.

As discussed earlier, the sophistication degrees of three example design applications presented in Part III deviate from a simple generic one (the first artificial plant) to a much sophisticated one (the wineglass design system). While the simple plant example has shown how a generic GED kernel (GEDK) alone can develop a fully-automated system that supports designing in a generative and

evolutionary approach, the comparatively sophisticated wineglass system has shown the feasibility and applicability of applying the kernel for developing a more complex system when the kernel is integrated with external computational design tools. This integrated approach supports designing tasks with external design tools more effectively and efficiently, and makes use of those functions supported by these external tools such as 3D geometric modelling and rendering functions.

However this sophisticated system cannot be functioning in a fully-automated way as the first example. For product design applications, it is necessary to involve designer interactions at various key stages of the design process. Therefore being a way to overcome the limitation of a fully automated system for evolving and exploring design solutions, this study also focused on how to find a balance between a fully automated evolutionary system controlled by nature selection criteria and an interactive system that provides ways for the designers to intervene in the process of evolving and exploring design by providing data or decisions on the evaluation of the candidate solutions or the directions of the evolution. Given the nature of complexity in product design and with the three examples tested in this study, it is concluded that the kernel is used as a framework for developing generative and evolutionary design applications at which further domain specific knowledge and control strategies are to be worked together by the system developers and designers.

The results of these examples show that the kernel can improve the flexibility and efficiency of generating, exploring and adapting potential design candidates. Some further issues are worth to be investigated, particularly those issues discussed in section 8.4 relating to dynamics of GED hierarchies, knowledge of design, abilities in design exploration and adaptation, design representation and interaction, and the system development and integration with external design tools.

9.3 Contributions

A computational Generative and Evolutionary Design (GED) kernel offers an opportunity to tackle design problems by using computational techniques in a generic and scalable manner, for achieving better designing more efficiently and shortening the process of building an application. In the process of developing this kernel, knowledge and strategies were discovered for a unified representation of design objects related to their process of being explored and adapted. This approach provides insight on how the knowledge outside the discipline of design can be utilized and integrated to the theories and methodologies of design which by its nature is a multidisciplinary activity and process.

From a perspective of design, it is also necessary to know exactly what the prospective is and where the opportunities are for using computational techniques in improving design in terms of supporting the tasks achievable by human designers more quickly, and more importantly, supporting the designers in deriving better design solutions which would be difficult to achieve by designers themselves without the support of such kernel and its related computational techniques.

The developed GED kernel in this thesis provides an alternative and potentially more interactive and efficient way of exploring design problems. The implementation and evaluation of the kernel involving its applications in three different design examples has provided a foundation for the development of a new generation of design tools which are generative and evolutionary. This offers considerable advantages over other systems for the development of 3D product forms and structures which are normally supported in a certain degree partially with parametric technology.

Formulation, implementation and integration of the GEK kernel (GEDK) to these three demonstration systems have shown that it is feasible and applicable to use

the GEDK as a computational core of design supporting systems. These GEDK-embedded example systems have been evaluated and further demonstrated that the GED kernel can improve the flexibility and efficiency of generating, exploring and adapting potential solutions in design. This research has demonstrated the potentials of three different evolutionary mechanisms in different design applications that involved generalization and specialization of design representation as well as design exploration process. As such this study contributes to the field of computational design by the formulation of the evolutionary kernel which has the potentials to be further studied and enhanced as an alternative and potentially more powerful design tool than those systems without generative and evolutionary mechanisms. This new tool can be integrated with existing design systems at a proper level for the designers to interact and evolve a large number of design solutions.

9.4 Future Work and Directions

One potential work is related to the formation of a suitable GED hierarchy for a specific design task. In the example of generating wineglasses, geometric structure is the main abstraction property of the GED hierarchy. Different hierarchies are constructed based on different abstraction aspects used, such as cultural and aesthetic aspects. It is worth to further investigate the issues relating to constructing a more general GED network, in which different abstraction aspects are handled within one GED system, as shown in the illustration in Figure 9.1. The figure shows how the GED hierarchies of the three examples may be merged to form a new network for including different abstraction aspects in one GED system which is more generic.

In fact, based on the last two examples in 2D pattern matching and wineglass generation, some experimental works were conducted in this research for producing wineglasses with a hybrid system formed by directly integrating these two examples without much of additional programming. As expected the

outcome is largely distorted as shown in the image of figure 9.2. This shows clearly that in a product design domain where the form of a design object is more constrained than a naturally growing plant or a random generated graphic pattern, additional knowledge or control mechanisms are needed in order to generate the design solutions which fit more with the functional and aesthetic design requirements rather than merely giving surprised and over diversified results.

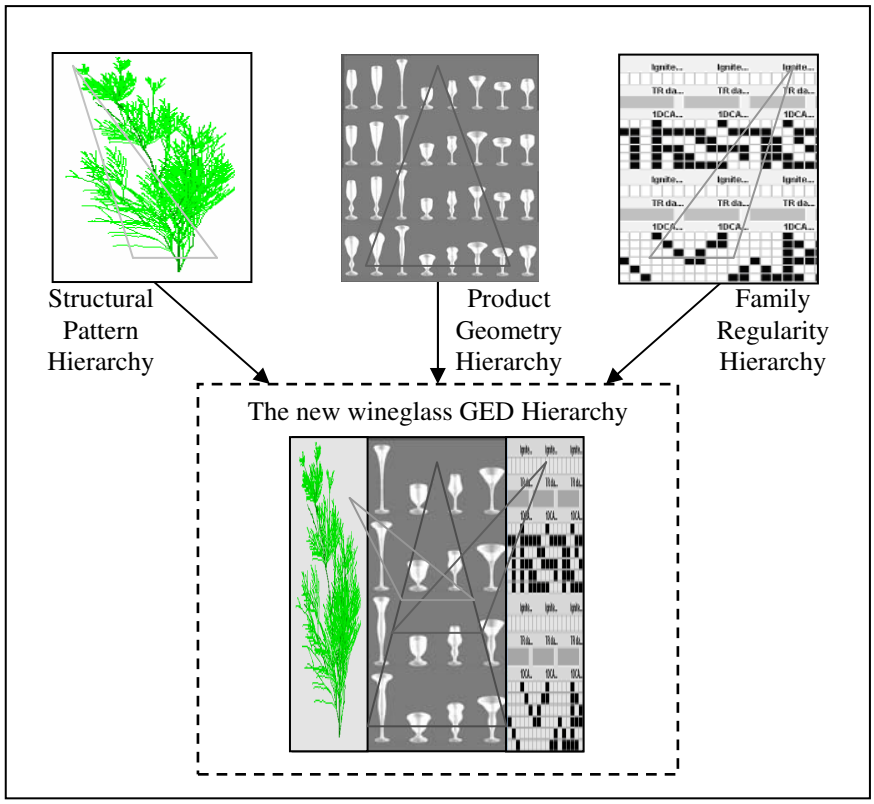


Figure 9.1: An example of merging different GED hierarchies to a general GED network.

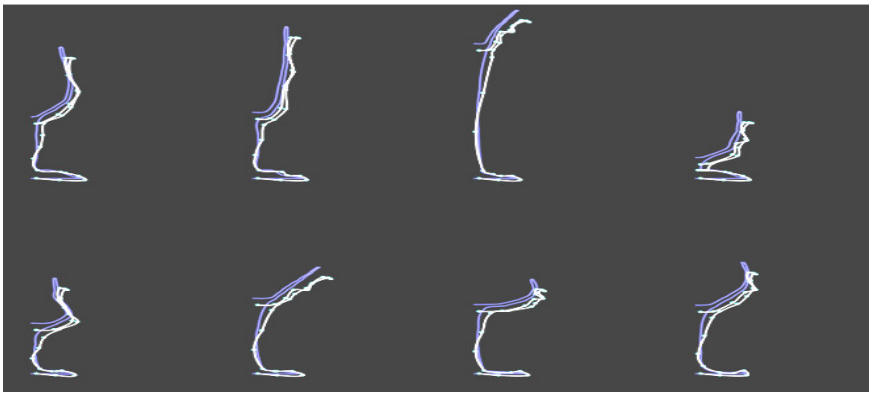


Figure 9.2: Highly distorted wineglass profiles in an attempt to apply simple GA-CA.

Another work is to get the right generative design mechanisms for potential design solutions at specific abstractions. For example, CA is used in the second demonstration example as the generative mechanism to generate a more complex form (2D image) from a simple one (1D data). It is shown that there are a variety of 1D CA types that can produce different 2D patterns. However there are other generative mechanisms, such as Shape Grammars and L-Systems, which may also produce other potential design patterns that cannot be generated by CA. It is necessary to compare their effectiveness in a unified system such as the kernel developed in this thesis, and through experimental studies to show how a similar approach with appropriate techniques can generate more potential solutions in a wider dynamic domain.

The kernel has not been implemented at a level at which three examples of different complexity can be tested and evaluated. As such it is not intended to be a fully automated system kernel to support any design applications, which is out of the scope of this study. However, more investigations in applying the developed kernel to other design application can certain enrich the knowledge and generic properties of the kernel. Further investigations are needed in order to maximize the potentials of the kernel and its associated generative and evolutionary mechanisms.

References

Akin, O. and Akin, C. (1996) Frames of reference in architectural design: analyzing the hyperacclamation (A-h-a!). *Design Studies*, 17(4), pp.341-361.

Andrews, P.T.J. and Sivaloganathan, S. (1998) A Variant Model for Storing Families of Mechanical Designs. In: S. Sivaloganathan and T.M.M. Shahin (eds.) *Engineering Design Conference, '98: Design Reuse*. London: Professional Engineering Pub., pp.361-369.

Aspray, W. and Burks, A. (eds.)(1987) *Papers of John von Neumann on computing and computer theory*. Cambridge: MIT Press.

Back, T. (1996) *Evolutionary Algorithms in Theory and Practice*. England: Oxford University Press.

Batty, M., Couclelis, H. and Eichen, M. (1997) Urban systems as cellular automata. *Environment and Planning B: Planning and Design*, 24, pp.159-164.

Bedau, M., McCaskill, J., Packard, N., Rasmussen, S., Adami, C., Green, D., Ikegami, T., Kaneko, K. and Ray, T. (2000) Open Problems in Artificial Life. *Artificial Life*, 6(4), pp.363-376.

Bentley, P.J. (ed.) (1999) *Evolutionary Design by Computers*. San Francisco: Morgan Kaufmann.

Berlekamp, E., Conway, J.H., and Guy, R. (1982) *Winning Ways for Your Mathematical Plays*. Volume 2. New York: Academic Press.

Black, R. (1996) *Design and Manufacture: An Integrated Approach*. London: Macmillan Press Ltd.

Blessing, L.T.M. (1994) *A Process-Based Approach to Computer-Supported Engineering Design*. Thesis, University of Twente, Enschede, Netherlands.

Bliek, C. (1995) Set Based Hierarchical Design: a Constraint Satisfaction Approach. *Proceedings of the 1995 Design Engineering Technical Conferences*, 17-20 September, Boston, pp.437-446.

Braha, D. and Maimon, O. (1998) *A mathematical theory of design: foundations, algorithms, and applications*. Boston: Kluwer.

Brazier, F.M.T., Jonker, C.M., Treur, J. and Wijngaards, N.J.E. (2001) Compositional design of a generic design agent. *Design Studies*, 22(5), pp.439-471.

Brazier, F.M.T. and Wijingaards, N.J.E. (2002) Designing Creativity. *AID 02 Learning and Creativity in Design Workshop*, July, Cambridge University, UK.

Brown, D.C. and Grecu, D.L. (2000) Always Expect the Unexpected! *AID 00 Machine Learning in Design Workshop*, June, Worcester Polytechnic Institute, USA.

Burton, R.M. and Obel, B. (eds.) (1995) *Design models for hierarchical organizations: computation, information, and Decentralization*. Boston: Kluwer.

Caldas, L. (2008) Generation of energy-efficient architecture solutions applying GENE_ARCH: An evolution-based generative design system. *Advanced Engineering Informatics*, 22(1), pp.59-70.

Ceccato, C. (1999) The Architect as Toolmaker. *Proceedings of the Fourth Conference on Computer Aided Architectural Design Research in Asia CAADRIA '99*, 5-7 May, Shanghai, pp.294-304.

Chakrabarti, A. and Bligh, T.P. (1996) An Approach to Functional Synthesis of Solutions in Mechanical Conceptual Design, Part III: Spatial Configuration. *Research in Engineering Design*, 8(2), pp.116-124.

Chakrabarti, A., Langton, P., Liu, Y. and Bligh, T. (2002) An approach to compositional synthesis of mechanical design concepts using computers. In: A. Chakrabarti (ed.) *Engineering Design Synthesis: Understanding, Approaches, and Tools*. New York: Springer, pp.179-198.

Chase, S.C. (2005) Generative design tools for novice designers: Issues for selection. *Automation in Construction*, 14(6), pp.689-698.

Chien, S. and Flemming, U. (2002) Design space navigation in generative design systems. *Automation in Construction*, 11(1), pp.1-22.

Cliff, D. (2003) Explorations in evolutionary design of online auction market mechanisms. *Electronic Commerce Research and Applications*, 2(2), pp.162-175.

Coyne, R., Finger, S., Konda, S., Monarch, I., Prinz, F.B., Siewiorek, D.P., Subrahmanian, E., Tenenbaum, J.M., Weber, J., Cutkosky, M., Leifer, L.J., Bajcsy, R., Koivunen, V. and Birmingham, W. (1994) Creating an Advanced Collaborative Open Resource Network. *Design Theory and Methodology*, 68, pp.375-380.

Cross, N. (1994) *Engineering design methods: strategies for product design*. New York: Wiley.

Cross, N. (1997) Descriptive models of creative design: application to an example. *Design Studies*, 18, pp.427-455.

Cross, N. (1999) Natural Intelligence in Design. *Design Studies*, 20, pp.25-39.

Cross N, Christiaans, H. and Dorst, K. (1996) *Analysing design activity*. Chichester: Wiley.

Crozier, R. (1994) *Manufactured pleasures: psychological responses to design*. Manchester: Manchester University Press.

Das, R., Crutchfield, J.P., Mitchell, M. and Hanson, J.E. (1995) Evolving globally

synchronized Cellular Automata. In: L.J. Eshelman (ed.) *Proceedings of the Sixth International Conference on Genetic Algorithms*, July, Pittsburgh. San Francisco: Morgan Kaufmann, pp.336-343.

de Vries, E. (2006) Students' construction of external representations in design-based learning situations. *Learning and Instruction*, 16(3), pp.213-227.

Dhillon, B.S. (1996) *Engineering design: a modern approach*. Chicago: Irwin.

Dhillon, B.S. (1998) *Advanced design concepts for engineers*. Lancaster: Technomic Pub. Co.

Dimopoulos, C. (2006) Multi-objective optimization of manufacturing cell design. *International Journal of Production Research*, 44(22), pp.4855-4875.

Duffy, A.H.B. (1997) The what and how of learning in design. *IEEE Expert*, 12(3), pp.71-76.

Eiben, A.E. (1996) Evolutionary exploration of search spaces. In: Z. Ras and M. Michalewicz (eds.) *Proceedings of Foundations of Intelligent Systems: 9th international symposium, ISMIS '96*, 9-13 June, Zakopane, Poland. New York: Springer, pp.178-188.

Ferreira, C. (2001) Gene Expression Programming: A New Adaptive Algorithm for Solving Problems. *Complex Systems*, 13(2), pp.87-129.

Fikes, R.E. and Nilsson, N.J. (1971) STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2(3-4), pp.189-208.

Fischer, T., Burry, M. and Frazer, J. (2005) Triangulation of generative form for parametric design and rapid prototyping. *Automation in Construction*, 14(2), pp.233-240.

Flake, G.W. (1998) *The computational beauty of nature: computer explorations of fractals, chaos, complex systems, and adaptation*. Cambridge: MIT Press.

Fogel, D.B. (1995) *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. New York: IEEE press.

Frazer, J. (1995) *An Evolutionary Architecture*. London: Architecture Association.

French, M.J. (1994) *Invention and evolution: design in nature and engineering*. New York: Cambridge University Press.

French, M.J. (1999) *Conceptual design for engineers*. Berlin: Springer-Verlag.

Gero, J. (1990) Design Prototypes: A Knowledge Representations Schema for Design. *AI Magazine*, 11(4), pp.26-36.

Gero, J.S. (1996) Creativity, emergence and evolution in design: concepts and framework. *Knowledge Based Systems*, 9(7), pp.435-448.

Gero, J.S., Kazakov, V.A. and Schnier, T. (1997) Genetic Engineering and Design Problems. In: D. Dasgupta and Z. Michalewicz (eds.) *Evolutionary algorithms in engineering applications*. Berlin: Springer, pp.47-69.

Gero, J.S. and Reffat, R. (1997) Multiple representations for situated agent-based learning. In: B. Varma, B. and X. Yao (eds.) *Proceedings of the International Conference on Computational Intelligence and Multimedia applications (ICCIMA-97)*, 10-12 February, Griffith University, Gold Coast, pp.81-85.

Gero, J.S. and Tyugu, E. (eds.) (1994) *Formal Design Methods for CAD*. New York: Elsevier.

Giunchiglia, F., Villafiorita, A and Walsh, T. (1997) Theories of Abstraction. *AI Communications*, 10(3-4), pp.167-176.

Giunchiglia, F. and Walsh, T. (1992) A Theory of Abstraction. *Artificial Intelligence*, 56(2-3), pp.323-390.

Goel, A.K. (1997) Design, analogy, and creativity. *IEEE Expert*, 12(3), pp.62-70.

Goldratt, E.M. and Fox, R.E. (1986) *The Race*. New York: The North River Press.

Gong, D., Guo, G., Lu, L. and Ma, H. (2008) Adaptive interactive genetic algorithms with individual interval fitness. *Progress in Natural Science*, 18(3), pp.359-365.

Graf, J. (1995) Interactive Evolutionary Algorithms in Design. *International Conference on Artificial Neural Networks and Genetic Algorithms ICANNGA '95*, Ales, France, pp.227-230.

Grecu, D.L. and Brown, D.C. (1998) Dimensions of machine learning in design. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AIEDAM)*, 12, pp.117-121.

Gu, Z. (2006) *Towards capturing aesthetic intent of design in an interactive evolutionary system using neural networks*. Thesis (PhD.), School of Design, Hong Kong Polytechnic University.

He, K., Zheng, L., Dong, S. and Tang, L. (2007) PGO: A parallel computing platform for global optimization based on genetic algorithm. *Computers & Geosciences*, 33, pp.357–366.

Heisserman, J., Callahan, S. & Mattikalli, R. (2000) A design representation to support automated design generation. In: J.S. Gero and F. Sudweeks (eds.) *Artificial Intelligence in Design '00*. Boston: Kluwer Academic, pp.545-566.

Herr, C.M. and Kvan, T. (2007) Adapting cellular automata to support the architectural design process. *Automation in Construction*, 16(1), pp.61-69.

Holland, J.H. (1975) *Adaptation in Natural and Artificial Systems*. 2nd edition. Ann Arbor: the University of Michigan Press.

Hornby, H.S. (2003) *Generative Representations for Evolutionary Design Automation*.

Thesis (PhD.), Department of Computer Science, Brandeis University.

Jain, S. and Gea, H.C. (1998) Two-Dimensional Packing Problems Using Genetic Algorithms. *Engineering with Computers*, 14, pp.206-213.

Janssen, P., Frazer, J. and Tang, M. (2002) Evolutionary Design Systems and Generative Processes. *Applied Intelligence*, 16(2), pp.119-128.

Janssen, P.H.T. (2005) *A design method and computational architecture for generating and evolving building designs*. Thesis (PhD.), School of Design, Hong Kong Polytechnic University.

Jenkins, D.L. and Martin, R.R. (1993) The Importance of Free-Hand Sketching in Conceptual Design: Automatic Sketch Input. *Design Theory and Methodology*, 53, 155-128.

Johnson, H. (1993) *How to enjoy your wine : understanding, storing, serving, ordering, enjoying every glass to the full*. London: Chancellor Press, p.58-59.

Jones, J.C. (1992) *Design Methods*. 2nd edition. New York: Van Nostrand Reinhold.

Kakihara, M. and Sorensen, C. (2002) Exploring knowledge emergence: from chaos to organizational knowledge. *Journal of Global Information Technology Management*, 5(3), pp.48-66.

Khuri, S., Schutz, M. and Heitkotter, J. (1995) Evolutionary Heuristics for the Bin Packing Problem. *International Conference on Artificial Neural Networks and Genetic Algorithms, ICANNGA '95*, Ales, France, pp.286-288.

Kicinger, R., Arciszewski, T. and Jong, K.D. (2005) Evolutionary computation and structural design: A survey of the state-of-the-art. *Computers & Structures*, 83(23-24), pp.1943-1978.

Kim, H. and Yoon, W.C. (2005) Supporting the cognitive process of user interface design with reusable design cases. *International Journal of Human-Computer Studies*, 62(4), pp.457-486.

Knight, T.W. (1994) *Transformations in design: a formal approach to stylistic change and innovation in the visual arts*. Cambridge: Cambridge University Press.

Knoblock, C.A. (1994) Automatically Generating Abstractions for Planning. *Artificial Intelligence*, 68(2), 243-302.

Kondo, T. (2007) Evolutionary design and behavior analysis of neuromodulatory neural networks for mobile robots control. *Applied Soft Computing*, 7(1), pp.189-202.

Koza, J.R. (1989) Hierarchical genetic algorithms operating on populations of computer programs. In: N.S. Sridharan (ed.) *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence IJCAI-89*, 20-25 August, Michigan, pp 768--774.

Koza, J.R. (1992) *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge: The MIT Press.

Lawson, B.R. (1979) Cognitive strategies in architectural design. *Ergonomics*, 22(1), pp.59-68.

Lawson, B. (1990) *How designers think*. 2nd edition. London: Butterworth Architecture.

Lee, L.H., Chew, E.P., Teng, S. and Chen, Y. (2008) Multi-objective simulation-based evolutionary algorithm for an aircraft spare parts allocation problem. *European Journal of Operational Research*, 189(2), pp.476-491.

Lenaerts, T., Gross, D. and Watson, R. (2002) On the Modelling of dynamical hierarchies: Introduction to the Workshop WDH 2002. *Workshop Proceedings Alife VIII: Modeling Dynamical Hierarchies in Artificial Life*, University of New South Wales, Sydney, pp.37-44.

Limbourg, P. and Kochs, H. (2008) Multi-objective optimization of generalized reliability design problems using feature models—A concept for early design stages. *Reliability Engineering and System Safety*, 93(6), pp.815–828

Lindenmayer, A. (1968) Mathematical models for cellular interaction in development. *Journal of Theoretical Biology*, 18, pp.280-315.

Liu, H. and Tang, M. (2006) Evolutionary design in a multi-agent design environment. *Applied Soft Computing*, 6(2), pp.207-220.

Liu, H., Tang, M.X. and Frazer, J. (2000) Supporting learning in a shared design environment. *Advances in Engineering Software*, 32(4), pp.285-293.

Liu, X., Tang, M. and Frazer, J.H. (2005) An eco-conscious housing design model based on co-evolution. *Advances in Engineering Software*, 36(2), pp.115-125.

Medland, A.J. (1992) *The computer-based design process*. London: Chapman & Hall.

Menon, U., O'Grady, P.J., Gu, J.Z. and Young, R.E. (1994) Quality Function Deployment: an overview. In: C.S. Syan and U. Menon (eds.) *Concurrent Engineering: Concepts, Implementation, and Practice*. London: Chapman & Hall, pp.91-99.

Michalewicz, Z., Xiao, J. and Trojanowski, K. (1996) Evolutionary computation: one project, many directions. In: Z. Ras and M. Michalewicz (eds.) *Proceedings of Foundations of Intelligent Systems: 9th international symposium, ISMIS '96*, 9-13 June, Zakopane, Poland. New York: Springer, pp.189-201.

Miller, J.G. (1978) *Living systems*. New York: McGraw-Hill.

Mitchell, C.T. (1996) *New Thinking in Design: Conversations on Theory and Practice.* New York: Van Nostrand Reinhold.

Mühlenbein, H., Zinchenko, L., Kureichik, V. and Mahnig, T. (2007) Effective mutation rate for probabilistic evolutionary design of analogue electrical circuits. *Applied Soft Computing*, 7(3), pp.1012-1018.

Mukesh, J.P., Honavar, V. and Balakrishnan, K. (2001) *Advances in the Evolutionary*

Synthesis of Intelligent Agents. Cambridge: MIT Press.

Nariman-Zadeh, N., Darvizeh, A., Jamali, A. and Moeini, A. (2005) Evolutionary design of generalized polynomial neural networks for modelling and prediction of explosive forming process. *Journal of Materials Processing Technology*, 164-165, pp.1561-1571.

Navinchandra, D. (1991) *Exploration and innovation in design: towards a computational model.* New York: Springer-Verlag.

Newell, A. and Simon, H. (1972) *Human Problem Solving*. New Jersey: Prentice-Hall.

O'Sullivan, D. (2001) Graph-cellular automata: a generalised discrete urban and regional model. *Environment and Planning B: Planning and Design*, 28, pp.687-705.

Pahl, C. (2004) Adaptive development and maintenance of user-centric software systems. *Information and Software Technology*, 46(14), pp.973-986.

Pahl, G. and Beitz, W. (1996) *Engineering design: a systematic approach*. London: Springer.

Park, H., Pedrycz, W. and Oh, S. (2007) Evolutionary design of hybrid self-organizing fuzzy polynomial neural networks with the aid of information granulation. *Expert Systems with Applications*, 33(4), pp.830-846.

Petroski, H. (1996) *Invention by design: how engineers get from thought to thing*. London: Harvard University Press.

Piaget, J. (1970) *Structuralism*. New York: Harper & Row.

Piaget, J. (1971) *Biology and Knowledge*. Chicago: University of Chicago Press.

Piaget, J. (1983) Piaget's theory. In: P. Mussen (ed.) *Handbook of Child Psychology.* 4th edition. New York: Wiley.

Pierreval, H., Caux, C., Paris, J. L. and Viguier, F. (2003) Evolutionary approaches to the design and organization of manufacturing systems. *Computers & Industrial Engineering*, 44(3), pp.339-364.

Poon, J. and Maher, M.L. (1996) Emergent Behaviour in Co-Evolutionary Design. In: J.S. Gero and F. Sudweeks (eds.) *Artificial Intelligence in Design '96*. Netherlands: Kluwer Academic, pp.703-722.

Pugh, S. (1991) *Total design: integrated methods for successful product engineering*. Wokingham: Addison-Wesley.

Raton, B. (1999) *Fusion of neural networks, fuzzy sets, and genetic algorithms: industrial applications*. FL: CRC press.

Riedel (2006) *The Wine Glass Company*. Available from http://www.riedel.com/ [accessed December 2006].

Robinson, J. (ed.) (1994) *The Oxford companion to wine*. New York: Oxford University

Press, pp.449.

Rodgers, P.A. (1998) The Role of Artificial Intelligence as 'text' within Design. *Design Studies*, 19, pp.143-160.

Rosenman, M. and Gero, J. (1999) Evolving Designs by Generating Useful Complex Gene Structures. In: P.J. Bentley (ed.) *Evolutionary Design by Computers*. San Francisco: Morgan Kaufmann, pp.345-364.

Rowbottom, A. (1999) Evolutionary Art and Form. In: P.J. Bentley (ed.) *Evolutionary Design by Computers*. San Francisco: Morgan Kaufmann, pp.261-277.

Sacerdoti, E.D. (1974) Planning in a Hierarchical Abstraction Spaces. *Artificial Intelligence*, 5(2), pp.115-135.

Sarkar, P. (2000) A Brief History of Cellular Automata. *ACM Computing Systems*, 32(1), pp.80–107.

Sekimoto, S. and Ukai, M. (1994) A Study of Creative Design Based on the Axiomatic Design Theory. *Design Theory and Methodology*, 68, pp.71-77.

Shahin, T.M.M., Andrews, P.T.J. and Sivaloganathan, S. (1998) A Design Reuse System. In: S. Sivaloganathan and T.M.M. Shahin (eds.) *Engineering Design Conference, '98: Design Reuse*. London: Professional Engineering Pub., pp.163-172.

Shahin, T.M.M. and Sivaloganathan, S. (1998) Representing Conceptual Designs. In: S. Sivaloganathan and T.M.M. Shahin (eds.) *Engineering Design Conference, '98: Design Reuse*. London: Professional Engineering Pub., pp.569-576.

Shea, K., Aish, R. and Gourtovaia, M. (2005) Towards integrated performance-driven generative design tools. *Automation in Construction*, 14(2), pp.253-264.

Sim, S.K. and Duffy, A.H.B. (1998) A foundation for machine learning in design. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AIEDAM)*, 12, pp.193-209.

Sim, S.K. and Duffy, A.H.B. (2002) Knowledge transformers – a link between learning and creativity. *AID 02 Learning and Creativity in Design Workshop*, July, Cambridge University, UK.

Simon, H.A. (1996) *The Sciences of the Artificial*. 3rd edition. London: MIT Press.

Sims, K. (1991) Artificial Evolution for Computer Graphics. *Computer Graphics Proceedings (SIGGRAPH '91)*, 28 July-2 August, Las Vegas, pp.319-328.

Sims, K. (1994) Evolving Virtual Creatures. *Computer Graphics Proceedings (SIGGRAPH '94)*, 24-29 July, Florida, pp.15-22.

Stiny, G. and Gips, J. (1972) Shape grammars and the generative specification of painting and sculpture. In: C.V. Freiman (ed.) *Information Processing 71*. Amsterdam: North-Holland, pp.1460-1465.

Stouffs, R. (2008) Constructing design representations using a sortal approach. *Advanced Engineering Informatics*, 22(1), pp.71-89.

Suh, N.P. (1990) *The Principles of Design*. New York: Oxford University Press.

Sun, J., Frazer, J.H. and Tang, M. (2007) Shape optimisation using evolutionary techniques in product design. *Computers & Industrial Engineering*, 53(2), pp.200-205.

Tang, M.X. and Wallance, K.M. (1997) A Knowledge-Based Approach to CAD System Integration. *International Conference on Engineering Design ICED '97*, August, Finland, pp.185-190.

Todd, S. and Latham, W. (1999) The Mutation and Growth of Art by Computers. In: P.J. Bentley (ed.) *Evolutionary Design by Computers*. San Francisco: Morgan Kaufmann, pp.221-250.

Tomiyama, T. (1995) A Design Process Model that Unifies General Design Theory and Empirical Findings. *Proceedings of the 1995 Design Engineering Technical Conferences, ASME*, Boston, pp.329-340.

Ueda, K. (2001) Synthesis and emergence – research overview. *Artificial Intelligence in Engineering*, 15, pp.321-327.

Van Dijk, C.G.C. (1994) Evaluation of a Surface Modeler for Conceptual Design. *Proceedings of the 1994 Lancaster International Workshop on Engineering Design (CACD '94)*, Lancaster, pp.11-13.

von Neumann, J (1966) *Theory of self-reproducing automata*. Edited and completed by Arthur W. Burks. Urbana: University of Illinois Press.

Vries, M.J. (1993) Design methodology and relationships with science: introduction. In: M.J. Vries, N. Cross, and D.P. Grant (eds.) *Design methodology and relationships with science*. Boston: Kluwer, pp.2.

Westney, D.E. (2001) Multinational enterprises and cross-border knowledge creation. In: I. Nonaka and T. Nishiguchi (eds.) *Knowledge Emergence: Social, Technical, and Evolutionary Dimensions of Knowledge Creation*. New York: Oxford University Press, pp.147-175.

Witbrock, M. and Neil-Reilly, S. (1999) Evolving Genetic Art. In: P.J. Bentley (ed.) *Evolutionary Design by Computers*. San Francisco: Morgan Kaufmann, pp.251-260.

Wu, Z. and Duffy, A. (2002) Mutual Knowledge Evolution in Team Design. *AID 02 Learning and Creativity in Design Workshop*, July, Cambridge University, UK.

Yoshioka, M., Nakamura, M., Tomiyama, T. and Yoshikawa, H. (1993) A Design Process Model with Multiple Design Object Models. *Design Theory and Methodology,* 53, pp.7-14.

Youssef, A.M.A. and ElMaraghy, H.A. (2006) Modelling and optimization of multiple-aspect RMS configurations. *International Journal of Production Research*, 44(22), pp.4929-4958.

Appendix A: GED Kernel Implementation

In order to demonstrate the feasibility and applicability of the kernel for supporting different computational design tasks, the kernel has been implemented as a software package (or library) in Java programming language. It can be integrated with other software that supports Java application interfaces, including external commercial CAD tools. This appendix introduces the major Java program components implemented and provides information for possible further development based on this study. The implemented programs include the GED kernel and three example systems presented in chapters 5, 6, 7 and 8 of this thesis.

A.1 Implemented Java Classes and Packages for the GED Kernel

An objective of this research study is to examine the feasibility and applicability of a generic computational kernel. In this study, Java programming language is used to implement the GED kernel as a software library or Java package, which can be used as the core framework of design supporting systems. There are four main Java packages implemented in this study:

1. The generic GED kernel (Chapter 5), **gedah**, includes the basic interface (**edah_IF**) and GUI representation (**edah_GUI**). The GED Kernel **gedah** alone is used in the first example (Chapter 6) without additional modification.
2. The 2D pattern generation system (Chapter 7), **edah_gaca**, uses the generic kernel to generate and match image patterns for demonstrating the kernel ability of knowledge adaptation in design generation.
3. The simple wine-glass system (Chapter 8), **edah_wg**, is developed with the generic kernel that generates wine-glasses in a 2D profile form.

4. The enhanced wine-glass system (Chapter 8), **wgmedah**, is developed by embedding the kernel with an external CAD tool (MicroStation).

Figure A.1 shows the block diagram of these four packages.

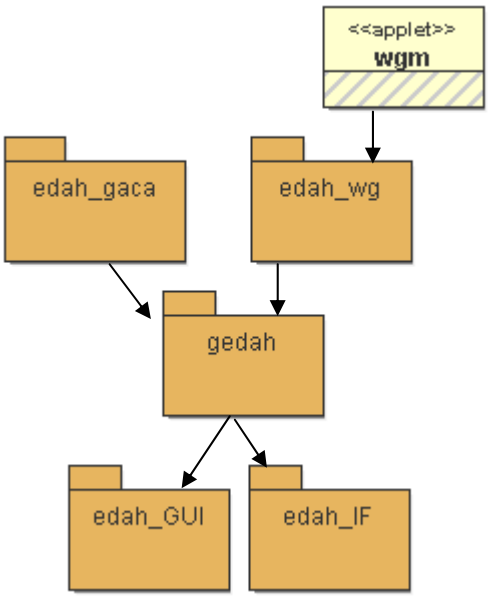


Figure A.1: A block diagram of the implemented Java packages for the GED kernel.

A.2 First Example: Artificial Plant Generation

The Java package **gedah** is the implementation of the GED kernel presented in this study. It consists of the major generic classes: evolutionary elements and mechanisms, together with basic interfaces (in **edah_IF**) and GUI representations (in **edah_GUI**) as shown in figure A.2. The first application example, the artificial plant generation, is also demonstrated with this kernel alone.

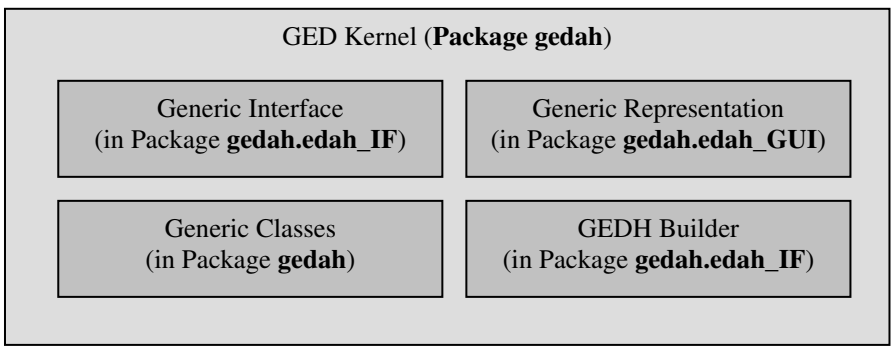


Figure A.2: Block diagrams of the implemented GED kernel.

A.2.1 Package **gedah***: the GED Kernel*

Figure A.3 shows the block diagram of the implemented **gedah** generic classes, followed by listing the summary of this package (including the classes implemented in the package, the class inheritance in Java class hierarchy, and the list of major variables, constructors and methods).

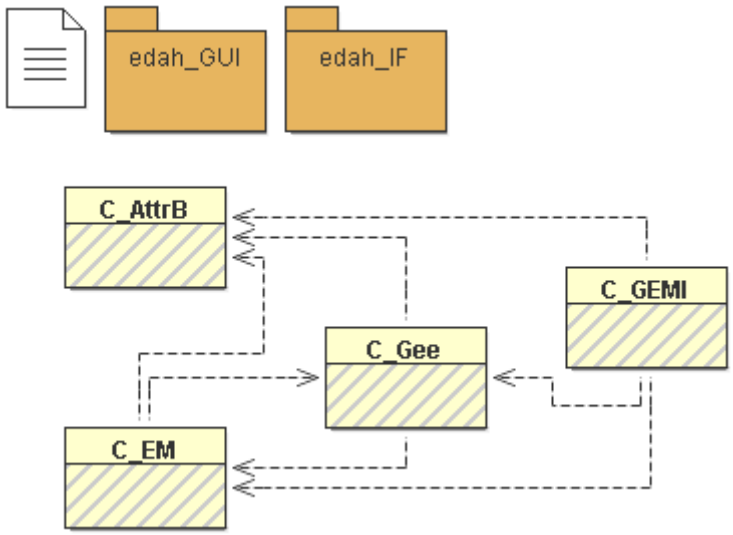


Figure A.3: Block diagrams of the implemented **gedah** package.

*Classes in **gedah***	
C_AttrB	**C_Gee**
C_EM	**C_GEMI**

*Class Hierarchy For Package **gedah***

- java.lang.Object
  - gedah.**C_AttrB**
  - gedah.**C_EM**
  - gedah.**C_Gee**
  - gedah.**C_GEMI**

*List of major variables, constructors and methods in **gedah***

A
allGeeV - Variable in class gedah.C_GEMI
allGeeVSize - Variable in class gedah.C_GEMI
AppendGCVGee(C_Gee) - Method in class gedah.C_Gee
AppendGPVGee(C_Gee) - Method in class gedah.C_Gee

A.2.2 Package **gedah.edah_GUI***: the Generic Graphical User Interface (GUI)*

Figure A.4 shows the block diagram of the implemented **gedah.edah_GUI** package, followed by listing the summary of this package (including the classes implemented in the package, the class inheritance in Java class hierarchy, and the list of major variables, constructors and methods).

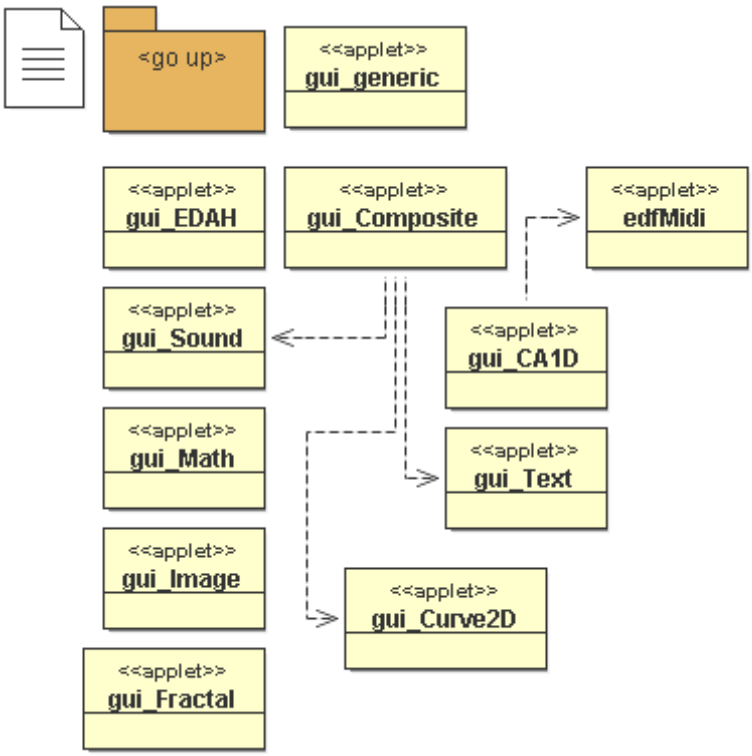


Figure A.4: Block diagrams of the implemented **gedah.edah_GUI** package.

Classes in ***gedah.edah_GUI***	
edfMidi	gui_generic
gui_CA1D	gui_Image
gui_Composite	gui_Math
gui_Curve2D	gui_Sound
gui_EDAH	gui_Text
gui_Fractal	

Class Hierarchy For Package ***gedah.edah_GUI***

- java.lang.Object
  - java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - java.awt.Container
      - java.awt.Panel (implements javax.accessibility.Accessible)
        - java.applet.Applet
          - javax.swing.JApplet (implements javax.accessibility.Accessible, javax.swing.RootPaneContainer)
            - gedah.edah_GUI.**edfMidi** (implements java.lang.Runnable)
            - gedah.edah_GUI.**gui_CA1D** (implements java.awt.event.ActionListener, java.lang.Runnable)
            - gedah.edah_GUI.**gui_Composite** (implements java.awt.event.ActionListener, java.awt.event.ItemListener, java.lang.Runnable)
            - gedah.edah_GUI.**gui_Curve2D** (implements java.awt.event.ActionListener, java.awt.event.ItemListener, java.lang.Runnable)
            - gedah.edah_GUI.**gui_EDAH** (implements java.awt.event.ActionListener, java.awt.event.ItemListener, java.lang.Runnable)
            - gedah.edah_GUI.**gui_Fractal** (implements java.awt.event.ActionListener, java.awt.event.ItemListener, java.lang.Runnable)
            - gedah.edah_GUI.**gui_generic** (implements java.awt.event.ActionListener, java.awt.event.ItemListener, java.lang.Runnable)
            - gedah.edah_GUI.**gui_Image** (implements java.awt.event.ActionListener, java.awt.event.ItemListener, java.lang.Runnable)
            - gedah.edah_GUI.**gui_Math** (implements java.awt.event.ActionListener, java.awt.event.ItemListener, java.lang.Runnable)
            - gedah.edah_GUI.**gui_Sound** (implements java.awt.event.ActionListener, java.awt.event.ItemListener, java.lang.Runnable)
            - gedah.edah_GUI.**gui_Text** (implements java.awt.event.ActionListener, java.awt.event.ItemListener, java.lang.Runnable)

List of major variables, constructors and methods in ***gedah.edah_GUI***

A

actionPerformed(ActionEvent) - Method in class gedah.edah_GUI.gui_CA1D

actionPerformed(ActionEvent) - Method in class gedah.edah_GUI.gui_Composite

actionPerformed(ActionEvent) - Method in class gedah.edah_GUI.gui_Curve2D

actionPerformed(ActionEvent) - Method in class gedah.edah_GUI.gui_EDAH

actionPerformed(ActionEvent) - Method in class gedah.edah_GUI.gui_Fractal

actionPerformed(ActionEvent) - Method in class gedah.edah_GUI.gui_generic

actionPerformed(ActionEvent) - Method in class gedah.edah_GUI.gui_Image

actionPerformed(ActionEvent) - Method in class gedah.edah_GUI.gui_Math

actionPerformed(ActionEvent) - Method in class gedah.edah_GUI.gui_Sound

actionPerformed(ActionEvent) - Method in class gedah.edah_GUI.gui_Text

B

butStr - Static variable in class gedah.edah_GUI.gui_CA1D

C

close() - Method in class gedah.edah_GUI.edfMidi

close() - Method in class gedah.edah_GUI.gui_Sound

D

DatabaseAction(int) - Method in class gedah.edah_GUI.gui_EDAH

def_selectNo - Static variable in class gedah.edah_GUI.gui_CA1D

def_selectNoStr - Static variable in class gedah.edah_GUI.gui_CA1D

DrawForest() - Method in class gedah.edah_GUI.gui_Fractal

DrawForestBranch(Vector, int, double, double, double, double, double, Random, int, int) - Method in class gedah.edah_GUI.gui_Fractal

DrawForestTree(Vector, Random) - Method in class gedah.edah_GUI.gui_Fractal

DrawForestTree_OLD(Vector, Random) - Method in class gedah.edah_GUI.gui_Fractal

DrawImage() - Method in class gedah.edah_GUI.gui_Image

DrawTree() - Method in class gedah.edah_GUI.gui_Fractal

DrawTree_Determine() - Method in class gedah.edah_GUI.gui_Fractal

E

edahMouseListener - Variable in class gedah.edah_GUI.gui_Composite

edahMouseListener - Variable in class gedah.edah_GUI.gui_Curve2D

edahMouseListener - Variable in class gedah.edah_GUI.gui_generic

edahMouseListener - Variable in class gedah.edah_GUI.gui_Image

edahMouseListener - Variable in class gedah.edah_GUI.gui_Math

edahMouseListener - Variable in class gedah.edah_GUI.gui_Sound

edahMouseListener - Variable in class gedah.edah_GUI.gui_Text

edfAllNotesOff(int) - Method in class gedah.edah_GUI.edfMidi

edfAllNotesOff(int) - Method in class gedah.edah_GUI.gui_Sound

edfMidi - Class in gedah.edah_GUI

edfMidi() - Constructor for class gedah.edah_GUI.edfMidi

edfNoteOn(int, int, int) - Method in class gedah.edah_GUI.edfMidi

edfNoteOn(int, int, int) - Method in class gedah.edah_GUI.gui_Sound

edfProgChange(int, int) - Method in class gedah.edah_GUI.edfMidi

edfProgChange(int, int) - Method in class gedah.edah_GUI.gui_Sound

EvolutionAction(int) - Method in class gedah.edah_GUI.gui_EDAH

EvolveOnce() - Method in class gedah.edah_GUI.gui_CA1D

threadInterval - Variable in class gedah.edah_GUI.gui_generic
threadInterval - Variable in class gedah.edah_GUI.gui_Image
threadInterval - Variable in class gedah.edah_GUI.gui_Math
threadInterval - Variable in class gedah.edah_GUI.gui_Sound
threadInterval - Variable in class gedah.edah_GUI.gui_Text
topP - Variable in class gedah.edah_GUI.gui_Composite
topP - Variable in class gedah.edah_GUI.gui_Curve2D
topP - Variable in class gedah.edah_GUI.gui_generic
topP - Variable in class gedah.edah_GUI.gui_Image
topP - Variable in class gedah.edah_GUI.gui_Math
topP - Variable in class gedah.edah_GUI.gui_Sound
topP - Variable in class gedah.edah_GUI.gui_Text
treeNum - Variable in class gedah.edah_GUI.gui_Fractal
trStr - Static variable in class gedah.edah_GUI.gui_CA1D
U
UpdateResult() - Method in class gedah.edah_GUI.gui_EDAH
UpdateStates() - Method in class gedah.edah_GUI.gui_CA1D
W
WaitSound() - Method in class gedah.edah_GUI.edfMidi

A.2.3 Package **gedah.edah_IF***: The Basic Interface*

Figure A.5 shows the block diagram of the implemented **gedah.edah_IF** package, followed by listing the summary of this package (including the classes implemented in the package, the class inheritance in Java class hierarchy, and the list of major variables, constructors and methods).

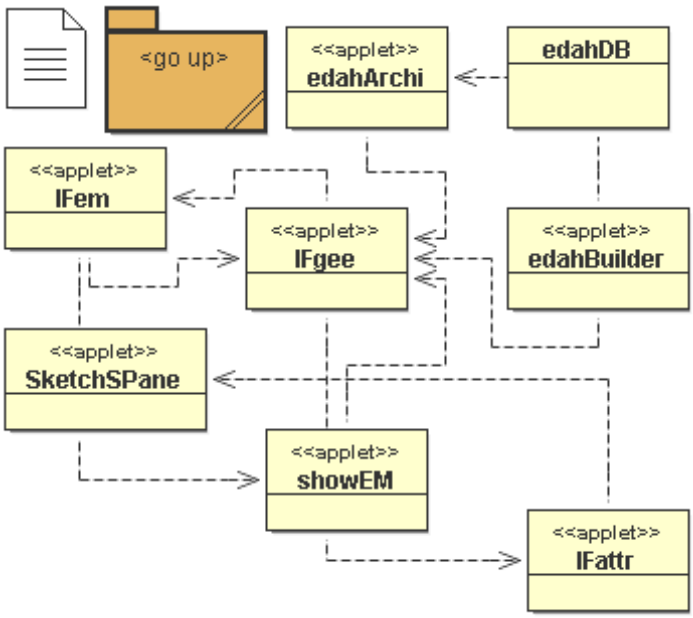


Figure A.5: Block diagrams of the implemented **gedah.edah_IF** package.

*Classes in **gedah.edah_IF***	
edahArchi	**IFem**
edahBuilder	**IFgee**
edahDB	**showEM**
IFattr	**SketchSPane**

*Class Hierarchy For Package **gedah.edah_IF***

- java.lang.Object
  - java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - java.awt.Container
      - java.awt.Panel (implements javax.accessibility.Accessible)
        - java.applet.Applet
          - javax.swing.JApplet (implements javax.accessibility.Accessible, javax.swing.RootPaneContainer)
            - gedah.edah_IF.**edahArchi** (implements java.awt.event.ActionListener, java.awt.event.ItemListener, java.lang.Runnable)
            - gedah.edah_IF.**edahBuilder** (implements java.awt.event.ActionListener, java.awt.event.ItemListener, java.lang.Runnable)
            - gedah.edah_IF.**IFattr** (implements java.awt.event.ActionListener, java.awt.event.ItemListener)
            - gedah.edah_IF.**IFem** (implements java.awt.event.ActionListener, java.awt.event.ItemListener)
            - gedah.edah_IF.**IFgee** (implements java.awt.event.ActionListener)
            - gedah.edah_IF.**showEM** (implements java.awt.event.ActionListener)
            - gedah.edah_IF.**SketchSPane** (implements java.awt.event.ActionListener, java.awt.event.MouseListener, java.awt.event.MouseMotionListener)
    - gedah.edah_IF.**edahDB**

*List of major variables, constructors and methods in **gedah.edah_IF***

A
actionPerformed(ActionEvent) - Method in class gedah.edah_IF.edahArchi
actionPerformed(ActionEvent) - Method in class gedah.edah_IF.edahBuilder
actionPerformed(ActionEvent) - Method in class gedah.edah_IF.IFattr

SketchSPane - Class in gedah.edah_IF
SketchSPane() - Constructor for class gedah.edah_IF.SketchSPane
start() - Method in class gedah.edah_IF.edahArchi
start() - Method in class gedah.edah_IF.edahBuilder
stop() - Method in class gedah.edah_IF.edahArchi
stop() - Method in class gedah.edah_IF.edahBuilder

T

tablesStr - Static variable in class gedah.edah_IF.edahDB
TestGEM(int) - Method in class gedah.edah_IF.edahArchi
threadInterval - Variable in class gedah.edah_IF.edahArchi
threadInterval - Variable in class gedah.edah_IF.edahBuilder
titleNameStr - Static variable in class gedah.edah_IF.edahArchi
titleNameStr - Static variable in class gedah.edah_IF.edahBuilder
topButLeng - Variable in class gedah.edah_IF.edahArchi
topButStr - Static variable in class gedah.edah_IF.edahArchi
topButStr - Static variable in class gedah.edah_IF.IFgee
topCheckB - Variable in class gedah.edah_IF.edahArchi
topP - Variable in class gedah.edah_IF.edahArchi
topP - Variable in class gedah.edah_IF.edahBuilder

U

UpdateBound(int, int, int, int) - Method in class gedah.edah_IF.SketchSPane
UpdateFromStructure() - Method in class gedah.edah_IF.edahBuilder
UpdateIFem() - Method in class gedah.edah_IF.IFem
UpdateIFgee(C_Gee) - Method in class gedah.edah_IF.IFgee
UpdateShowEM() - Method in class gedah.edah_IF.showEM
UpdateShowEM_CA() - Method in class gedah.edah_IF.showEM
UpdateShowEM_GA() - Method in class gedah.edah_IF.showEM

W

width - Variable in class gedah.edah_IF.edahArchi
wMin - Variable in class gedah.edah_IF.edahArchi
wP1 - Variable in class gedah.edah_IF.edahArchi

X

xP1 - Variable in class gedah.edah_IF.edahArchi
xStroke - Variable in class gedah.edah_IF.SketchSPane

Y

yInterval - Variable in class gedah.edah_IF.edahArchi
yOffset - Variable in class gedah.edah_IF.edahArchi
yP1 - Variable in class gedah.edah_IF.edahArchi
yStroke - Variable in class gedah.edah_IF.SketchSPane
yTopOffset - Variable in class gedah.edah_IF.edahArchi

A.3 Second Example: The 2D Pattern Generation System

The second application example, a 2D Pattern Generation and Matching system, is implemented in the Java package **edah_gaca** and is embedded with the GED

kernel package **gedah**. Figure A.6 shows the block diagram of this implemented package, followed by listing the summary of this package (including the classes implemented in the package, the class inheritance in Java class hierarchy, and the list of major variables, constructors and methods).

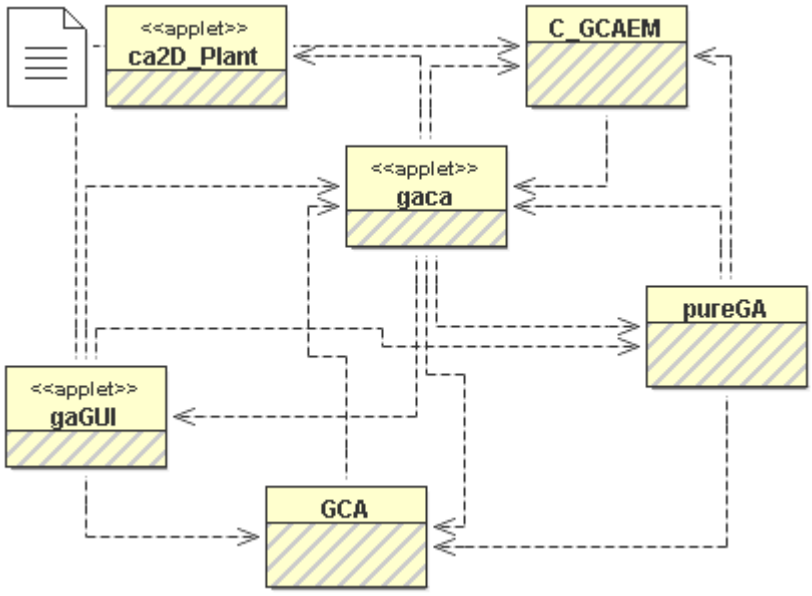


Figure A.6: Block diagrams of the implemented **edah_gaca** package.

Classes in ***edah_gaca***	
C_GCAEM	**gaGUI**
ca2D_Plant	**pureGA**
gaca	

Class Hierarchy For Package ***edah_gaca***

- java.lang.Object
  - C_EM
    - edah_gaca.**C_GCAEM**
  - java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - java.awt.Container
      - java.awt.Panel(implements javax.accessibility.Accessible)
        - java.applet.Applet
          - javax.swing.JApplet(implements javax.accessibility.Accessible, javax.swing.RootPaneContainer)
            - edah_gaca.**ca2D_Plant** (implements java.awt.event.ActionListener,

java.awt.event.ItemListener, java.lang.Runnable)
    - edah_gaca.**gaca** (implements java.awt.event.ActionListener, java.awt.event.ItemListener, java.lang.Runnable)
    - edah_gaca.**gaGUI** (implements java.awt.event.ActionListener, java.awt.event.ItemListener)
- edah_gaca.**pureGA**

*List of major variables, constructors and methods in **edah_gaca***

A.4 Third Example: The Wine-Glass Design System

The third application example, a wine-glass design system, is implemented in the package **edah_wg** and is embedded with the GED kernel package **gedah**. The wine-glass design example is further enhanced by integrating an external CAD tool, MicroStation. This enhanced system is implemented in a Microstation jmdl package **wgmedah** within the CAD tool MicroStation platform and is embedded with both the package **gedah** and the pure java wine glass package **edah_wg**.

A.4.1 Package **edah_wg***: the Pure Java-Based Wine-Glass System*

Figure A.7 shows the block diagram of this implemented package **edah_wg**, followed by listing the summary of this package (including the classes implemented in the package, the class inheritance in Java class hierarchy, and the list of major variables, constructors and methods).

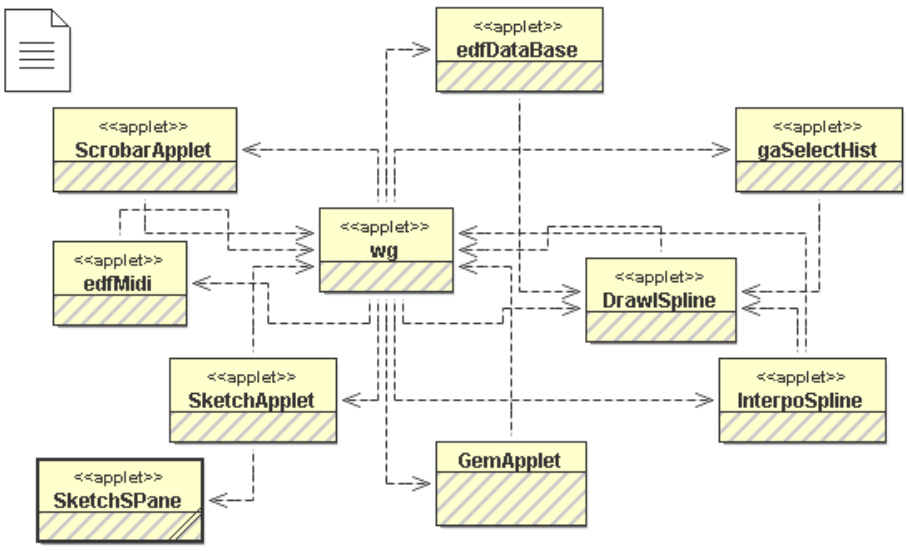


Figure A.7: Block diagrams of the implemented **edah_wg** package.

Classes in **edah_wg**	
DrawISpline	InterpoSpline
edfDataBase	ScrobarApplet
edfMidi	SketchApplet
gaSelectHist	wg
GemApplet	

*Class Hierarchy For Package **edah_wg***

- java.lang.Object
  - java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - java.awt.Container
      - java.awt.Panel(implements javax.accessibility.Accessible)
        - java.applet.Applet
          - javax.swing.JApplet(implements javax.accessibility.Accessible, javax.swing.RootPaneContainer)
            - edah_wg.**DrawISpline**
            - edah_wg.**edfDataBase**
            - edah_wg.**edfMidi** (implements java.lang.Runnable)
            - edah_wg.**gaSelectHist** (implements java.awt.event.ItemListener)
            - edah_wg.**InterpoSpline** (implements java.awt.event.ActionListener, java.awt.event.ItemListener)
            - edah_wg.**ScrobarApplet** (implements java.awt.event.AdjustmentListener)
            - edah_wg.**SketchApplet**
            - edah_wg.**wg** (implements java.awt.event.ActionListener, java.awt.event.ItemListener, java.lang.Runnable)
  - edahArchi
    - edah_wg.**GemApplet** (implements java.awt.event.ActionListener, java.awt.event.ItemListener, java.lang.Runnable)

*List of major variables, constructors and methods in **edah_wg***

SketchApplet - Class in edah_wg
SketchApplet() - Constructor for class edah_wg.SketchApplet
SketchApplet(wg, C_Gee) - Constructor for class edah_wg.SketchApplet
start() - Method in class edah_wg.edfMidi
start() - Method in class edah_wg.wg
stop() - Method in class edah_wg.edfMidi
stop() - Method in class edah_wg.wg
subWinStr - Static variable in class edah_wg.wg
U
UpdateDB() - Method in class edah_wg.edfDataBase
UpdateResult() - Method in class edah_wg.wg
W
wg - Class in edah_wg
wg() - Constructor for class edah_wg.wg
wgDbOut - Variable in class edah_wg.wg
wgIs2 - Variable in class edah_wg.wg

A.4.2 Package **wgmedah***: the Wine-Glass System with External CAD Tool*

Figure A.8 shows the block diagram of this implemented package **wgmedah**, followed by listing the summary of this package (including the classes implemented in the package, the class inheritance in Java class hierarchy, and the list of major variables, constructors and methods).

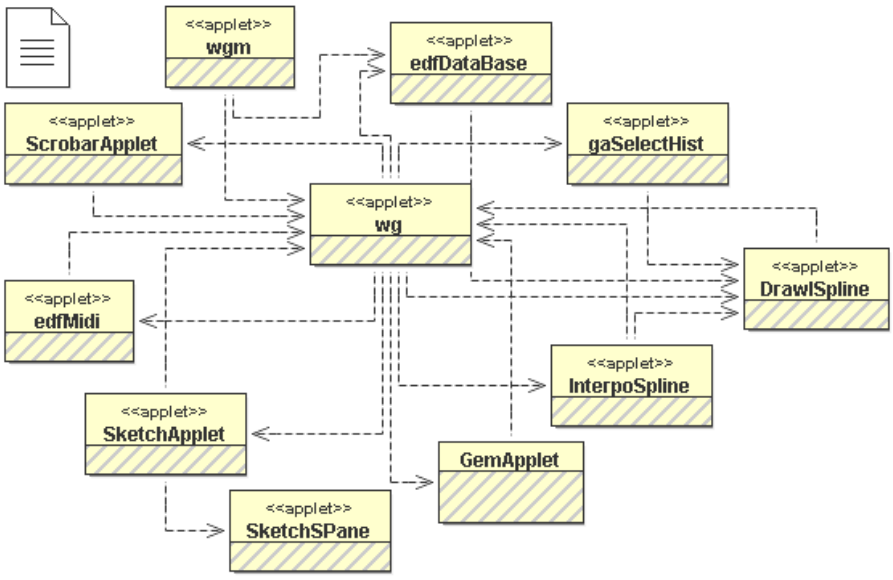


Figure A.8: Block diagrams of the implemented **wgmedah** package.

Class in ***wgmedah***	
wgm	

Class Hierarchy For Package ***wgmedah***

- java.lang.Object
  - java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - java.awt.Container
      - java.awt.Panel (implements javax.accessibility.Accessible)
        - java.applet.Applet
          - javax.swing.JApplet (implements javax.accessibility.Accessible, javax.swing.RootPaneContainer)
            - wgmedah.**wgm** (implements java.awt.event.ActionListener, java.lang.Runnable)

List of major variables, constructors and methods in ***wgmedah***

A
actionPerformed(ActionEvent) - Method in class wgmedah.wgm
D
DrawMStation(int[][]) - Method in class wgmedah.wgm
DrawWineGlassMStation(Point[][], boolean) - Method in class wgmedah.wgm
G
GenerateAllOnce() - Method in class wgmedah.wgm
I
InvertingPts(Point[][]) - Method in class wgmedah.wgm
M
main(String[]) - Static method in class wgmedah.wgm
msTBarStr - Static variable in class wgmedah.wgm
N
NewToolBar() - Method in class wgmedah.wgm
R
run() - Method in class wgmedah.wgm
S
start() - Method in class wgmedah.wgm
stop() - Method in class wgmedah.wgm
U
UpdateBottleOnce(boolean) - Method in class wgmedah.wgm
UpdateBowlOnce(boolean) - Method in class wgmedah.wgm
UpdatePlateOnce(boolean) - Method in class wgmedah.wgm
UpdateWGOnce(boolean) - Method in class wgmedah.wgm
W
wgm - Class in wgmedah
wgm() - Constructor for class wgmedah.wgm
wgmedah - package wgmedah

~ END ~