Malaysian Journal of Computer Science, Vol. 15 No. 2, December 2002, pp. 94-101

A CODE GENERATOR TOOL FOR THE GAMMA DESIGN PATTERNS

Novia Indriaty Admodisastro
Faculty of Science Computer and Information Technology
Universiti Putra Malaysia
Tel. No.: 03-89466517
email: novia@fsktm.upm.edu.my

Sellappan Palaniappan
Department of Information Technology
Malaysia University of Science and Technology
Tel. No.: 03-78801777
email: sell@must.edu.my

ABSTRACT

Software reuse has been recognised as an attractive idea with an obvious payoff to achieve software that is faster, better and cheaper. One important component to be highlighted in designing reusable object-oriented software is design patterns. Design patterns describe a commonly recurring structure of communicating components that solve a general design problem in a particular context. An important property of design patterns is that they are independent of a particular application domain and programming paradigm. As a result, design patterns facilitate reuse of software architecture, even when other forms of reuse are infeasible. Despite the fact that design patterns have tangible benefits, they have been found difficult to use. Since a design pattern only describes a solution to particular design problem, it does not lead to direct code reuse. Some developers have found it difficult to make the leap from pattern description to a particular implementation. The step in relaxing this complexity can be achieved using a code generator tool that aids developers to transform design patterns into code automatically. There are several pattern code generator tools currently available, but they have several shortcomings. This paper describes an attempt to automate design patterns implementation into a concrete form that takes advantage of WWW as a communication infrastructure. It includes the main features implemented by the existing tools and tackles some of their shortcomings. The tool has been evaluated and results were reported to be comparable and even better than other pattern code generator tools.

Keywords: ***Software reuse, Design patterns, Pattern code generator tool***

1.0 INTRODUCTION

Over the last few decades, the software industry has grown dramatically. This proliferation has led to an urgent and growing demand from customers for developing successful software systems. Here, success means, products and services that rest upon a software base to produce software faster, better and cheaper [1]. Software reuse has been recognized as an attractive idea with an obvious payoff to achieve these goals.

One important component to be highlighted in designing reusable object-oriented software is expert knowledge. Expertise is an intangible but unquestionably valuable commodity [2]. People acquire it slowly through hard work and perseverance. Expertise distinguishes a novice from an expert and it is difficult for experts to convey their expertise to novices. Emerging field of design patterns is a promising step for capturing, communicating and assimilating expertise. As a result, design patterns enable expertise to become tangible and not just reside in the experts' minds. These design patterns provide proven solutions that are certainly useful for designers to solve new design problems [3].

Despite the fact that design patterns have tangible benefits, they have been found difficult to use [4]. The main obstacle implementing design patterns is due to their abstract form. When recording a design, it is usually done at the more primitive level of individual classes and objects either in the form of class diagrams or actual code [5]. The first step in relaxing this complexity is to provide an explicit way of recording the design patterns application in the code. This can be achieved using a code generator tool that can aid developers to transform design patterns into code automatically [4].

The paper is organised as follows. Section 2 gives background studies on existing pattern code generator tools. In Section 3, the pattern code generator framework is presented. Section 4 discusses the evaluation done to validate the pattern code generator tool. Finally, Section 5 discusses the contribution of this research and future work.

2.0 BACKGROUND STUDIES

This section discusses the pattern code generator tools that are currently available in the market. The existing pattern code generator tools are divided into web-based online pattern code generator tools and non-web based pattern code generator tools.

2.1 Non-Web Based Pattern Code Generator Tools

2.1.1 S.C.U.P.E

S.C.U.P.E (Santa Clara University Pattern Editor) is developed by David Mendoza and Michael Hall [6]. This is the only tool that generates code for design patterns via graphical customisation [3]. The process begins by loading the "ChoosePattern" window. Software designers learn about particular design patterns by viewing their abstract description place at the center of "ChoosePattern" window. In the descriptions, there are references to figures, which represent the design patterns structure.

Software designers can choose to customise a predefined pattern or load a pattern that they were previously working on. They can graphically customise a design pattern that they have chosen to load. Design patterns are shown in the UML format on screen. They customise the pattern by changing class name, adding and deleting methods, method expressions, instance variables and deleting classes. The changes can be made by clicking on the class name, instance variable names or method names text boxes and editing the information directly. Once customisation is done, software designers can use the option feature to generate Java source for the design pattern displayed on screen. At the end of the process, the users have the option to save their customised design pattern diagram and the generated Java source code for later use.

The tool provides much assistance in design patterns implementation. However, it does not provide any useful help facility to software designers, especially to novices because the help menu only describes information on release and the copyright. Moreover, the tool does not support distributed implementation.

2.1.2 SNIP

SNIP [7] is a tool for instantiated patterns of code that can be derived from object models. It is developed by Fred Wild in order to instantiate C++ code for patterns defined by him as code patterns. Design patterns are logical in nature, but code patterns are physical in nature. Code patterns focus on how a particular structure or sequence of actions is accomplished using specific mechanisms of a programming language. It is also known as an idiom [8].

SNIP generates code patterns by using Object Model as an instantiation context to be defined. Thereafter users also need a well-defined implementation strategy where it requires users to apply a set of code-creation rules and place them in a SNIP Template file. It applies the rules called out in an executable Code Template to the objects and their parts and produces Code Files based on those rules. A number of Code Template files that serve as good starting points for this come with the tool or the template file can be developed interactively using SNIP's user interface. To summarise, SNIP allows defining both objects and their parts characteristics and shows how the object and their parts are mapped onto code elements.

2.2 Web-Based Pattern Code Generator Tools

2.2.1 Designer's Assistant Tool

Designer's Assistant Tool [9] is developed by Dr. J. Q. Huang in order to generate code for design patterns [3]. The "Design Pattern Space" page serves as a central focus while navigating through the tool. This page represents 23 design patterns in a well-organised catalogue as depicted in [3]. The users can select a design pattern according to its name from the catalogue to be able to view the pattern abstract description page. These sections are intended to remind users of the patterns. Next, to customise the design pattern, users need to define class participant names based on problem domain. After completing the customisation form, they have an option to generate C++, Java or Smalltalk code for the customised pattern. The users may save the generated code in a file. This procedure is repeated for the application of each design pattern.

Unlike other tools, Huang has used formal method to describe the design patterns where process language is introduced to describe the application of a design pattern. In addition, a language for describing objects and their

interconnections is introduced based on a schema. The use of these schemas does not restrict the designer, but does restrict the way in which designs are expressed.

The tool has the ability to generate code for different programming languages. This is a good web-based pattern code generator tool though it uses formal method specification. However, it does not support visualisation after the design patterns have been customised, nor does it provide help facility.

2.2.2 Automatic Code Generation

Automatic code generation is a tool developed by Budinsky et al. [2] for generating design patterns code in C++. For the sake of simplicity, the tool is referred as Budinsky's Tool. The tool has the same similarity as the Designer's Assistant tool where the customisation of design pattern requires users to define class participant names. However, although the customisation is done along with choices for the design trade-offs, the user is not forced to accept these trade-offs. The user then adds this code to the rest of the application, often enhancing it with other application specific functionality. In addition, this tool also supports design patterns abstract description, which mirrors the corresponding section in [3]. The information content in Section Pages displays the section of a pattern, e.g. Intent, Motivation and others in separate pages. The user can access the other sections of a design pattern either randomly or in sequence. Finally, users may save the generated code in a file using the browser's "Save As…" command.

Although Budinsky's tool offers the users the choice of selecting design trade-offs during the implementation of design pattern, it does not provide the ability to visualise the customise design pattern.

2.3 Identifying Main Features

The above tool review has revealed the most important features for a code generator tool. Seven features have been identified, supporting abstract description, diagram representation, participants and graphical customisation, customise pattern visualisation, trade-offs constraints and help facility. These features are listed in Table 1.

Table 1: Summary of features of on-line patterns code generator tools

Features	**Non Web-based**		**Web-based Tool**	
	S.C.U.P.E	SNIP	Budinsky's Tool	Designer's Assistant
Supporting Abstract Description	√	√	√	√
Diagram Representation	√		√	√
Participants Customisation		√	√	√
Graphical Customisation	√			
Customise Pattern Visualisation	√			
Trade-offs Constraints		√	√	
Help Facility	√	√	√	

- ***Supporting Abstract Descriptions***
  Supporting abstract descriptions contain the description of static and dynamic structure of a design pattern. The static parts are the pattern *Intent, Motivation, Applicability and Consequences* whereas the dynamic parts are pattern *Participants and Collaboration.* These descriptions lead designers in making a correct choice of pattern to be used in their design problem. The importance of these materials is proven when all review tools support this feature.

- ***Diagram Representation***
  Most of the tools provide pattern diagram representation. Only SNIP does not support this feature. Diagram representation is also the dynamic part of a pattern that illustrates a graphical representation of the classes in the pattern using a notation based on the Object Modelling Technique (OMT).

- ***Participants Customisation***
  The participants' customisation of a pattern requires users to define the pattern participants' names. SNIP, Designer's Assistant and Budinsky's Tool have used this type of customisation before generating code for the pattern. The pattern participant names selected by the designers must be meaningful in the application context.

- ***Graphical Customisation***
  S.C.U.P.E is the only tool that supports code generation for the pattern via graphical customisation. The diagram is represented in the UML format and edited directly on screen.

- ***Customise Pattern Visualisation***
  It is useful to visualise the pattern application to show the relationships between the pattern classes [5]. S.C.U.P.E is the only tool providing this feature by visualising the customised design pattern in the "Customise Pattern Screen".

- ***Trade-off Constraint***
  Trade-off constraint allows users to provide substitute conditions to the pattern implementation. Budinsky's Tool allows designers to choose pattern trade-offs. SNIP also acclaimed this feature by setting of code-creation rules and placing these in a SNIP Template file.

- ***Help Facility***
  The tool providing this feature is S.C.U.P.E, SNIP and Budinsky's Tool. This feature is considered one of the most important elements for supporting software usability.

3.0 GAMMA CODE GENERATOR TOOL (γ-CGT) ARCHITECTURE

This section presents the pattern code generator tool that has been developed, known as γ-CGT. The architecture explains the methodology and the implementation of γ-CGT.

3.1 γ-CGT Methodology

In order to build a pattern code generator tool, a methodology for the development of design pattern transformations into its concrete form is presented. The purpose of this methodology is to make sure the design pattern transformation preserves the proposed tool behaviour. This methodology adapts few steps of Gamma et al. methodology for applying design pattern [3] and some new steps based on the study conducted on existing pattern code generator tools.

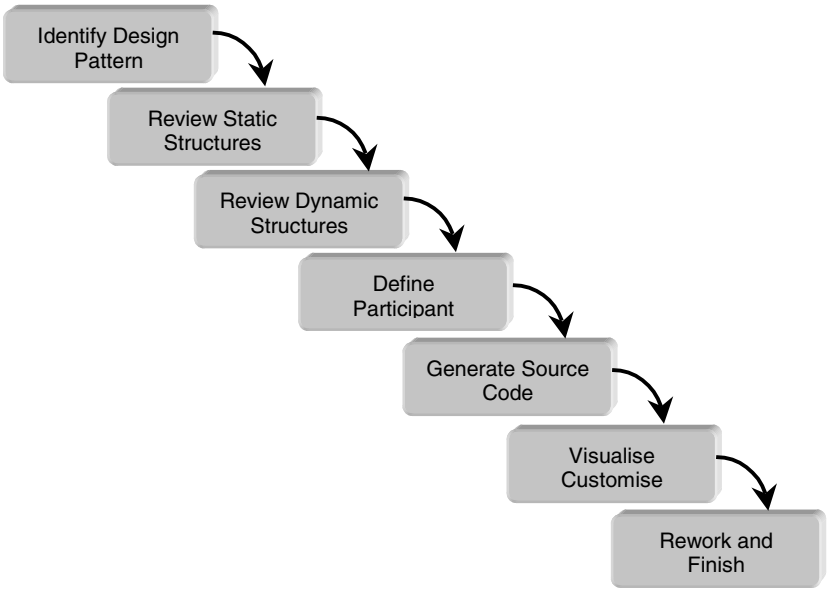

Fig 1: Pattern code generator tool methodology

3.1.1 Identify Design Pattern

With more than 20 design patterns in the catalogue to choose from, it might be hard to find the one that addresses a particular design problem, especially if the catalogue is new and unfamiliar to the designers. Here two approaches

are addressed to guide the designers to find the design pattern suitable for their problem. First, the designers have to study the pattern catalogue carefully.

Referring to the catalogue, the designers are able to narrow down the search by specifying the problem statement and scope. The problem statement may be concerned with creational, structural or behavioural pattern and the scope maybe related either to objects or classes. Studying the catalogue directs the designers to the right pattern or group of patterns. Next, the designers should study the pattern *Intent*. This step becomes easier due to the number of patterns to be reviewed becomes less after the designers have specified the pattern group. Pattern *Intent* is very important information because it describes the design issue and problem that the pattern addresses. Finally, the designers are able to determine the right design pattern for their problem.

3.1.2 Review Static Structures

Once the designers have chosen a design pattern, the static structures of the pattern must be reviewed. The pattern template contains several parts of static structures; however, two segments that are considered most important will be described. These are the pattern *Applicability* and *Consequences*. *Applicability* explains the situation where the design pattern can be applied and the examples of poor designs that the pattern can address. Whilst C*onsequences* describes the pattern trade-offs and the result of using the pattern.

3.1.3 Review Dynamic Structures

Besides the two segments of static structure that have been examined, the designers need to study the dynamic parts of design pattern as well. The dynamic parts are the pattern *Structure* and *Participants*. *Structure* shows a graphical representation of classes or objects in the design pattern using a notation based on Object Modelling Technique (OMT). *Participants* on the other hand explain the classes or objects participating in design pattern and their responsibilities.

3.1.4 Define Participant Names

After the designers have confirmed that the pattern is right for their design problem, they can proceed with the next step, which is to implement design pattern. The implementation will transform the design pattern from its abstract form to concrete form that is code. Before going further the designer needs to customise the pattern by defining pattern participants' names. The names for participants in the design pattern usually are too abstract to appear directly in an application. Therefore, the designers must define names for the participants that are meaningful in the application context.

3.1.5 Generate Source Code

Eventually, the design pattern is being implemented and the code for the customised design pattern is generated. The generated pattern code can be later integrated by the designers into their own application code they wish to develop. The generated code usually is presented using an object-oriented language such as Java, C++ or Smalltalk.

3.1.6 Visualise Customise Pattern

In addition to the generated code, visualisation of the design pattern application is really useful for the designers. Modelling helps to lessen the difficulty for the designers to understand the complexity of the relationship in classes or objects in the design pattern.

3.1.7 Rework and Finish

After the designers have finished implementing the design pattern, they may repeat the process for other design problems they have. Or, the designer may close the browser after using the pattern code generator tool.

3.2 Implementing γ-CGT

The following features are identified to constitute γ-CGT.

- **Supporting Abstract Description**. γ-CGT provides design patterns static and dynamic descriptions that are considered important for the designers. These include the pattern Intent, Applicability, Participants and Consequences description.
- **Diagram Representation**. γ-CGT illustrates the graphical representation of classes in the pattern using a notation based on the Object Modelling Technique (OMT).
- **Participant Customisation**. γ-CGT implements participant customisation mechanism before generating code for the design pattern. This requires the designers to define the pattern participants' names.
- **Customise Pattern Visualisation**. γ-CGT visualises the design pattern application by using UML notations.
- **Help Facility**. γ-CGT provides an online help throughout the process of generating code for the design pattern.
- **Alert Message**. γ-CGT provides an alert message while defining pattern participant names to assure that designers have given legitimate names for the participants.

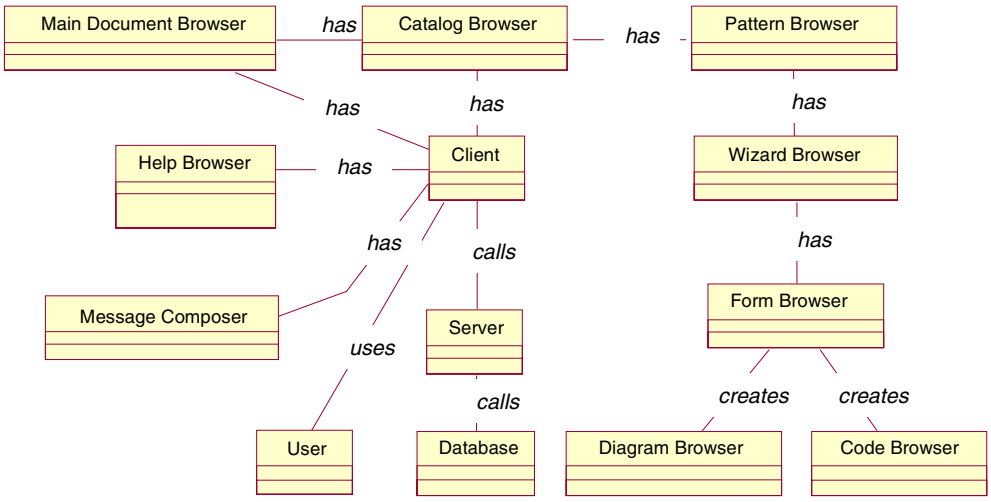

Fig. 2: γ-CGT class diagram

4.0 EVALUATION

This section discusses the evaluation done to validate γ-CGT. The evaluation is carried out by first conducting a pilot study with some graduate students, and second, by evaluating γ-CGT by comparing it with the current similar available tools.

4.1 Experimenting γ-CGT

This experiment describes the pilot study conducted to evaluate γ-CGT features and requirements. The pilot study includes a small number of postgraduate students with moderate-to-extensive knowledge on software engineering and reuse processes. The experimental material was taken from [10] with some modification to suit the requirements of this experiment. Although pattern catalogue consists of 23 design patterns, covering all the patterns in a short period of time usually blurs the participants spending more time together in understanding the details of many different examples on the pattern themselves [11]. Therefore, this material is only seeded with four requirements where each requirement requires the participants to specify the design pattern related to using γ-CGT.

At the end of this experiment, the participants were given a questionnaire to respond and the results showed that γ-CGT have met the proposed features and requirements [12].

4.2 Comparison of γ-CGT with Other On-line Patterns Generator Tool

This section compares γ-CGT with other on-line pattern code generator tools. Table 2 includes γ-CGT within the listed pattern code generator tools.

It is clear that γ-CGT has covered the most important features. The only two features that γ-CGT has not covered were the graphical customisation and trade-offs constraints. Compared to Budinsky's Tool, γ-CGT has supported 2 additional features; it offers help facility and provides alert messages. Whereas compared to Designer's Assistant Tool, γ-CGT offered three more features. It supports customised pattern visualisation, offers help facility and provides alert message. On the other hand, compared to non-web based pattern code generator tools, namely S.C.U.P.E and SNIP, γ-CGT appears to offer more features. Besides, γ-CGT has better features in terms of distributing the design pattern and their implementation knowledge.

Table 2: Comparison γ-CGT with other pattern code generator tools

Features	**Non Web-based**		**Web-based Tool**		
	S.C.U.P.E	SNIP	Budinsky's Tool	Designer's Assistant	γ-CGT
Supporting Abstract Description	√	√	√	√	√
Diagram Representation	√		√	√	√
Participants Customisation		√	√	√	√
Graphical Customisation	√				
Customise Pattern Visualisation	√				√
Trade-offs Constraints	√				
Help Facility		√	√		√
Alert Message					√

5.0 CONTRIBUTIONS AND FUTURE WORK

In summary, this research has made two main contributions:

- The first contribution is a methodology for the pattern code generator tool. This methodology ensures that the design pattern transformation preserves the tool behaviour.

- The second contribution is a prototype of web-based pattern code generator tool called γ-CGT. It is designed and implemented such that it applies these design pattern transformation to Java code. γ-CGT has been found to be comparable and even better than some other pattern code generator tools.

γ-CGT has incorporated most of the main features that constitute a good pattern code generator tool. However, additional features that can further increase the effectiveness of this tool should be considered. γ-CGT can be improved to

- Generate code for the pattern application in other programming languages such as C++ and Smalltalk. This will help designers from different programming background to transform the design pattern into its concrete form that they prefer to use.

- Support other kinds of patterns, since currently γ-CGT has been developed only to support design patterns. Therefore, γ-CGT can assist in the implementation of other kinds of pattern such as architectural patterns or idioms.

- Introduce some metrics into γ-CGT to measure the intended benefits of the decisions to apply specific design patterns based on designer's intuition. This metrics can help verify whether this intuition is right

or not. Besides being able to measure the benefits of a specific pattern, metrics based on design patterns might say something about the quality of the software that has been developed using design patterns.

In conclusion, design patterns, as one of the most successful reuse components, deserves further research so as to explore their full potential.

REFERENCES

[1] I. Jacobson, M. Griss, P. Jonsson, *Software Reuse: Architecture Process and Organization for Business Success*. Addison-Wesley, 1997.

[2] F. J. Budinsky, M. A. Finnie, J. M. Vlissides, P. S. Yu, "Automatic Code Generation from Design Patterns". *IBM Systems Journal*, 35 (2): 151-171, 1996.

[3] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[4] D. C. Schmidt, "Using Design Patterns to Develop Reusable Object-Oriented Communication Software". *Communications of the ACM*, 38 (10): 65-74, October 1995.

[5] G. Hedin, A. Ive, K. Mughal, K. Normark, H. Ron, K. Osterbye, "Tools for Design Patterns". *NWPER'98 Subworkshop on Tools for Software Architecture*, 1998.

[6] D. Mendoza, M. Hall, "S.C.U.P.E. (Santa Clara University Pattern Editor)". *Master's Thesis*, Santa Clara University, California, 1998.

[7] F. Wild, "Instantiating Code Patterns: Patterns Applied to Software Development". *Dr. Dobb's Journal: Patterns & Software Design*, June 1996.

[8] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerland, M. Stal, *Pattern-Oriented Software Architecture - A System of Pattern*. Wiley, 1996.

[9] J. Q. Huang, "A Designer's Assistant Tool". *Ph.D. Thesis*, Department of Computer Science, University of Waterloo, 1996.

[10] B. Cheng, "Advanced Software Engineering", *available from Michigan State University Homepage*, URL: http://www.cse.msu.edu/~cse870/.

[11] B. Goldfedder, L. Rising, "A Training Experience with Patterns". *Communications of the ACM*, 39 (10) : 60-64, October 1996.

[12] Novia Indriaty Admodisastro, *Master's Thesis*, Department of Software Engineering, University of Malaya, 2000.

BIOGRAPHY

Novia Indriaty Admodisastro obtained her Bachelor in Computer Science from the Universiti Putra Malaysia in 1999. Currently, she is a Master candidate in the Software Engineering Department at the University of Malaya. Her research interests include object-oriented modelling, design patterns and component-based software development.

Sellappan Palaniappan obtained his PhD in Computer and Information Science from the University of Pittsburgh in 1978. Currently, he is Associate Professor in the Department of Information Technology, Malaysia University of Science and Technology. His research interests include web applications and services, object-oriented modelling, distributed database systems, user interfaces and decision support systems. He has a number of publications in these areas.