A CODE GENERATOR TOOL FOR THE GAMMA DESIGN PATTERNS

NOVIA INDRIATY ADMODISASTRO

SUBMITTED TO FULFILL THE
PARTIAL REQUIREMENTS FOR THE DEGREE OF
MASTER OF SOFTWARE ENGINEERING

FACULTY OF COMPUTER SCIENCE AND
INFORMATION TECHNOLOGY
UNIVERSITY MALAYA
KUALA LUMPUR

APRIL 2003

ABSTRACT

Software reuse has been recognized 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, it was found difficult to put into practice. Since 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 a leap from the pattern description into a particular implementation. This complexity can overcome by using code generator tool that assist the developers to transform a design pattern into code automatically. There are a few existing tools in the market however most of them have limitations. Therefore, this thesis describes an attempt to automate the design patterns implementation into its concrete form and utilizing the WWW as its communication infrastructure. γ-CGT is introduces to implement the main features of the existing tools and tackles some of their shortcomings. The tool has been evaluated and the results were reported to be comparable and competitive to other pattern code generator tools.

ACKNOWLEDGEMENT

First and foremost I would like to express my gratitude to Allah S.W.T that gave me the possibility to complete this thesis. A very sincere and special thanks to my supervisor Prof. Dr. P. Sellapan, for his invaluable advice, guidance and for his subtle direction of my efforts throughout the preparation of this thesis.

Secondly, I would like to thank my colleagues, lecturers and technical staffs from the Department of Software Engineering for their endless assistance, technical advice and co-operation. I would also like to thank Universiti Putra Malaysia (UPM) for providing me a financial support throughout my study.

Last but not least, I would like to forward my deepest love and gratitude to my beloved parents, Rezhan S. Ashraff and Admodisastro family members for their continuous support, encouragement and previous love during the years of her studies and putting their heart and soul into her life.

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

Page

LIST OF ABBREVIATION

γ-CGT	Gamma Code Generator Tool
HTTP	Hyper Text Transfer Protocol
IAS	Inprise Application Server
JSP	Java Server Pages
OMT	Object Modeling Technique
UI	User Interface
UML	Unified Modeling Language
URL	Unified Resource Locator
WWW	World Wide Web

Chapter 1

INTRODUCTION

Over the past few decades, the software industry has grown dramatically. This proliferation has led to a vast growing demand on successful software systems among customers. Here, success refers to software systems that are faster, better, and cheaper (Jacobson et al., 1997). Software reuse has been recognized as an attractive idea with an obvious payoff to produce successful software systems.

The basic concept of software reuse is simple (McIlroy, 1969), develop systems components of a reusable size and reuse them. The idea of systems component is not only focused on code alone but also on requirements, analysis models, designs, and tests. All the software development processes stages are subjected to be "reuse". Reuse helps developers minimize problem-solving effort and redundant work. It also enhances reliability, which lead to reduced development time.

It is reported that the reuse process allows the management to expect substantial gains, time to market: reductions of 2 to 5 times, defect density: reductions of 5 to 10 times, maintenance cost: reductions of 5 to 10 times, and overall software development cost: reduction of around 15% to as much as 75% (Jacobson et al., 1997).

One important component to be highlighted in designing reusable object-oriented software is expert knowledge. Expertise is an intangible but unquestionably valuable commodity (Budinsky et al., 1996). 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 patterns is a promising step for capturing, communicating, and assimilating expertise. As a result, patterns enable expertise to be tangible so it does not only reside in the experts' minds. These patterns provide proven solutions that are certainly useful for designers to solve new design problems. This is because recurring design problems always occur in object-oriented software or known as design déjá-vu. This makes object-oriented designs more flexible, elegant and ultimately reusable.

Patterns are grouped into three categories namely architectural patterns, design patterns, and idioms (Buschmann et al., 1996). The difference between these three kinds of patterns is in their corresponding levels of abstraction and detail. Architectural patterns represent the highest-level patterns that concern large-scale components. Whereas, design patterns (Gamma et al., 1995) are medium-scale patterns and have a strong influence on the structure of subsystems. Lastly, idioms represent low-level patterns, which are specific to a programming language (Coplien, 1992). Generally, design patterns are scenes the far most well elaborated published patterns. Furthermore, design patterns describe many examples of basic patterns and fundamental frameworks structure that novice object-oriented designers should become familiar with (Brown, 1996).

1.1 Design Patterns

Design patterns are popularized by Erich Gamma et al. through the published book of (Gamma et al., 1995), which present a catalog consisting of 23 design patterns. The design pattern catalog are found in *Appendix C*. Gamma et al. or frequently referred as the Gang of Four (GoF) was much influenced by the work of a building architect, Christopher Alexander. According to him,

> "*Each pattern describes a problem which occurs over and over again in our environment, and then describe the core solution to that problem, in such a way that you can use this solutions a million times over, without ever doing it the same way twice*" (*Alexander et al., 1977; Alexander, 1979).*

The idea as suggested by Alexander is relevant for object-oriented patterns. A design pattern describes a commonly recurring structure of communicating components that solve a general design problem in a particular context (Gamma et al., 1995). An important property of all design patterns is that they are independent on a particular application domain and programming paradigm. Besides that, design patterns must have a name as the design vocabulary and be represented using a consistent format called template. The template lends a uniform structure to the information, making the design pattern easier to learn, comparable, and useful. It contains the following parts:

- The intent of the pattern
- The design forces that motivate the pattern
- The solution to these forces
- The structure and roles of classes in the solution
- The responsibilities and collaborations among classes
- The trade-offs and results using the pattern.
- Implementation guidance
- Example source code
- References to related patterns

The template shows that a design pattern captures both static (intent, design force and solutions, trade-offs and results, implementation guidance, source code, and references to related patterns), and dynamic (structure, participants, and collaboration) structures of successful solutions to solve problem arise during building the applications in a particular domain. Therefore, design patterns aid the development of reusable software by expressing the structure and collaboration of components to designers at a level higher than source code or object-oriented design models that focus on individual objects and classes (Schmidt, 1996). Figure 1.1 illustrates the area of design pattern in object-oriented design methodologies.

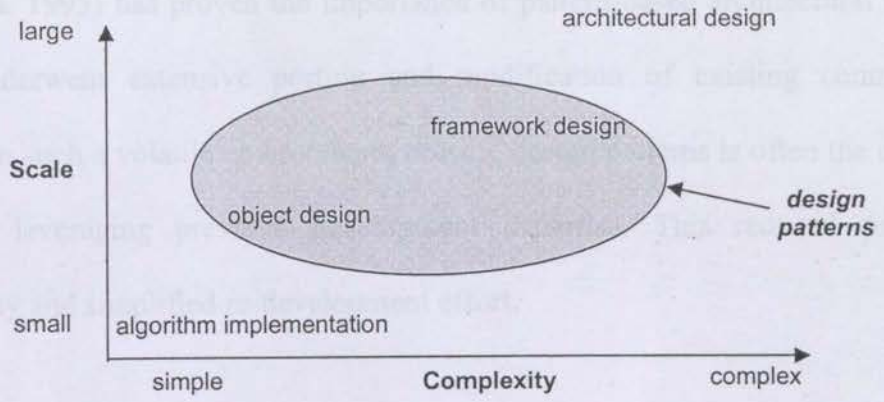

Figure 1.1: Design patterns reside in object-oriented design methodologies (Buschmann et al., 1996)

1.2 Design Patterns Objectives

The goal of design patterns within software engineering is to create a body of literature to assist software designers to resolve recurring problems encountered throughout the software development (Brad, 2000). Design patterns help create a shared language communication insight and experience about these problems and their solutions.

1.3 Benefits Of Design Patterns

There are a number of pragmatic benefits in using design patterns. Firstly, design patterns enable the widespread reuse of software architecture (Beck and Johnson, 1994; Gamma, 1991; Johnson, 1992; Keller and Lajoie, 1994; Pree, 1994; Schmid, 1995), even when other forms of reuse are infeasible. This infeasibility could be caused by the fundamental differences in operating system mechanism or programming language features. The Ericsson telecommunication switch management project (Schmidt and

Stephenson, 1995) has proven the importance of pattern-based architectural reuse. The project underwent extensive porting and modification of existing communication software. In such a volatile environment, reusing design patterns is often the only viable means of leveraging previous development expertise. This reduced project risk significantly and simplified re-development effort.

Secondly, design patterns improve communication within and across software development teams because they provide developers with shared vocabulary and concepts (Gamma et al., 1995; Beck et al., 1996; Cline, 1996). In addition, design patterns helped to bridge the communication gap that exists among software developer, managers, and non-technical team members in marketing and sales (Helm, 1995; Schmidt, 1995a). Managers and non-technical team members often failed to understand the system at the detailed object models or source code level. However, they frequently could understand and evaluate the consequences and trade-offs among software architecture concepts that are expressed as design patterns. Their feedbacks are valuable to ensure that the technical solutions do not drift away from the overall system requirements.

Design patterns explicitly capture knowledge that experienced designers already understand implicitly (Alexander et al., 1977; Buschmann et al., 1996; Coad, 1992; Coplien, 1992; Fowler, 1996; Gamma et al., 1995). In this case, the use of patterns permits experts document, discuss, and reason systematically about sophisticated architectural concepts. Furthermore, explicitly capturing expertise through design

patterns helps to impart this knowledge to less experienced designers. Research conducted by Tao (2000), showed that the learning curve flatted and the framework was used more effectively when design patterns were used to facilitate new designers. Likewise, design patterns descriptions explicitly record engineering trade-offs and design alternatives. These can be used to record why certain design choices were selected and others rejected. If this rationale is not captured explicitly, it may be lost over time (Schmidt, 1995a).

Schmidt (1995a) also stated that design patterns help to transcend "programming language centric" viewpoints. This is beneficial because design patterns enable experienced developers from different language communities, such as Lisp, Smalltalk, Ada, Eiffel, C++, C, and Erlang to share design insights of mutual interest without the barrier of language wars. Once the experience developers moved beyond language syntax and semantic differences, remarkable commonality of successful software solutions can be shared.

Finally, at CEE a cellular network management and engineering system project, Helm (1995) reported that designs based on design patterns seemed to be more robust in requirement changes, and minimize the need for class-refactoring and re-design during later implementation. This is mainly because requirements variability is often factored out by the design patterns.

Many other companies implementing design patterns in real world environment reported similar pragmatic benefits. Some of the companies involved are Motorola Iridium (Schmidt, 1995b), Kodak Health Imaging Systems (Blaine et al., 1994), and Phoenix-based AG Communications Systems (Goldfedder and Rising, 1996).

1.4 Design Patterns Difficulties

Despite the fact that design patterns have tangible benefits, however, it is found that it is difficult to put into practice (Schmidt, 1995a; Sommerville, 2001). This is due to struggles that the novice object-oriented designers have to go through to learn about design patterns. According to Schmidt (1995a) they usually misunderstood that design pattern can solve design problems. This postulation is erroneous because integrating design patterns into a software development process is a human-intensive activity. Like other software reuse technologies, reuse of patterns does not come without cost (Fayad et al., 1996). Design patterns are no silver bullet that absolves designers from having to wrestle with complex analysis, design, and implementation issues. There is simply no substitute for creativity, experience, and diligence on the designers' parts.

Furthermore, design pattern only describes solutions to a particular design problem and it does not lead to direct code reuse. Some designers found it difficult to make the leap from the pattern description to a particular implementation, even though the pattern includes code fragments in the Sample Code section. Other experienced designers may have no trouble translating the pattern into code but they still find it a hassle, especially when they have to do it repeatedly. A design change might require

substantial re-implementation because different design choices in the pattern can lead to vastly different codes. Moreover, in the abstract form, patterns cannot be used directly by designers in their implementations and this make custom implementation vital. The mechanics of implementing design patterns is left to the designers (Budinsky et al., 1996).

These difficulties and many more (Cline, 1996; Levine and Schmidt, 1999; Helm, 1995) make it arduous for some designers to reap the advantages of design patterns.

1.5 Easing Design Pattern Difficulties

Perhaps the main obstacle in implementing design patterns is caused by their abstract form. Although when actually 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 in actual code (Hedin et al., 1998). The first step in relaxing this complexity is to provide an explicit way of recording the design patterns implementation in the code. This can be achieved by using code generator tool that aids designers to transform design patterns into code automatically (Schmidt, 1995a). It might also be useful to visualize the design pattern implementation to show the relationships between the design pattern classes (Hedin et al., 1998).

Besides being able to generate code and diagram, a well-designed web-based tool can be much more efficient in helping designers to find and use design patterns than a fully integrated "pattern-supporting" software development environment ever could be (Buschmann et al., 1996). The web-based tool allows designers to access materials and hypertext links to navigate information quickly through multiple levels of abstraction. It is capable to execute on different operating systems such as Windows, Linux, and Unix. It does not have working limits to allow anyone to access it at their preferences. Moreover, the tool is globally distributed. In addition, the Web is an ideal tool for disseminating, sharing, and communicating information (Berners-Lee, 1996).

Generally, a web code generator tool for design patterns has the following effect:

- Patterns have achieved the status as a must-have or must-do both in object-oriented circles and among software architects (Coplien, 1997). The tool aid in learning process and is a valuable approach to make the knowledge and implementation of design patterns widely accessible.
- Enable less experienced designers to acquire the knowledge and understanding the design patterns faster while more expert designers can modify or enhance the design patterns.
- Ease, encourages and speed-up the process of design patterns implementation in the object-oriented software development. The tool also plays an important role in introducing patterns as first-class citizens in an integrated object-oriented development environment (Florijn et al., 1997).

1.6 Research Objectives

This research is conducted to facilitate the design patterns implementation process. There are two main aims of this project:

1. To build a web-based design pattern (Gamma et al., 1995) code generator tool that distributes the design patterns' knowledge and ease designer's task to transform design patterns into concrete form. The prototype is called γ-CGT (Gamma Code Generator Tool). It is intended to solve some of the major problems found in using design pattern by:

   - Simplifying the process of generating source code by requiring only small amount of customization from designers.
   - Depicting the implemented design pattern into class diagram for visualization support.
   - Providing necessarily tips for designers while they are using the tools. This is to make sure the less experienced designers will be able to cope with the implementation process.
   - Expressing design patterns at a higher level of abstraction for reviews before customizing process.

2. To evaluate the implemented prototype. The tool was evaluated against the above objectives and a few of existing similar tools features.

1.7 Project Methodology

There are a few pattern code generator tools in the market. This study focuses on building a design pattern code generator tool that uses the web as a distributed infrastructure. The strategy to achieve this task involves the following steps.

1. Conducting a study on the organization and methodology for applying design pattern. This is to acquire a good understanding and to determine the steps in applying design pattern into its concrete form. This offers a clear idea about the structure of γ-CGT.

2. Scrutinising the existing on-line tools in order to capture their good features and to tackle their shortcomings in building γ-CGT.

3. Measuring the success of γ-CGT by conducting a pilot study involving a few participants. γ-CGT is going to be evaluated by participant's feedback via a questionnaire.

1.8 Thesis Organization

This study is organized in the following way:

Chapter 2, Literature Review

This chapter is divided into two main sections. The first section elaborates on the design patterns, explained the organization and methodology for applying design pattern. The second section provides a review of the existing

online pattern code generator tools. A number of main features in the existing tools revealed the basic features for γ-CGT.

Chapter 3, Research Framework

This chapter identifies the framework for this study and is divided into three section namely the structure of design pattern code generator tool's methodology, the construction of web-based design pattern code generator tool known as γ-CGT and the evaluation of γ-CGT.

Chapter 4, Analysis and Design

This chapter presents analysis and the design of γ-CGT. It includes the requirement analysis, γ-CGT object-oriented analysis and design and some aspects of the user interface design.

Chapter 5, Implementation and Execution

This chapter presents the implementation of γ-CGT. It also shows how to use γ-CGT in executing the design patterns implementation process. The enhancements that γ-CGT provide are also highlighted.

Chapter 6, Evaluation and Results

This chapter describes the evaluation process and the results recorded from the pilot study.

Chapter 7, Conclusion

This chapter summarizes the content and the contribution of this study. This is followed by a conclusion and suggestions for future study.

Chapter 2

LITERATURE REVIEW

This chapter reviews design patterns, which explains the organization and methodology for applying design patterns. These reviews are important to understand the design pattern before any further study can be conducted.

In addition, this chapter also discussed the pattern code generator tools found in the market. This study shapes the main features of the pattern code generator tool.

2.1 Review of the Design Patterns

2.1.1 Organizing Design Patterns

Design patterns vary in their granularity and level of abstraction. This section classifies design patterns as described in (Gamma et al., 1995) by grouping the design patterns into families of related patterns. The classification helps designers to learn patterns in the catalog faster and it can direct efforts to find new patterns as well.

Table 2.1: Pattern catalog (Gamma et al., 1995)

		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory	Adapter	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Façade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

As shown in table 2.1 design patterns are classified into two criteria. The first criterion is called 'Purpose' that reflects what a pattern does. Patterns can have creational, structural or behavioral purposes. Creational patterns are focused with the process of object creation while structural patterns deal with the composition classes or objects. Behavioral patterns characterized the way each classes or objects interacts and distributes responsibility.

The second criterion is called 'Scope', specifies whether the pattern applies primarily to classes or objects. Class patterns deals with relationships between classes and their subclasses. These relationships are established through inheritance thus, they are static-fixed at compile time. Object patterns deals with object relationships, which can be changed at run-time and more dynamic. Almost all patterns uses inheritance to some extent hence, the only patterns labelled "class patterns" are those that focused on class relationships.

Creational class patterns defer some parts of object creation to subclasses, while Creational object patterns defer object to another object. The Structural class patterns use inheritance to compose classes, while the Structural object patterns describe ways to assemble objects. The Behavioral class patterns use inheritance to describe algorithms and control flow, whereas the Behavioral object patterns describe how a group of objects cooperate to perform a task that no single object can carry out alone.

2.1.2 Design Pattern Implementation Methodology

A study conducted in this literature found only one methodology to guide designers in implementing design pattern into their design problems. The methodology presented in (Gamma et al., 1995) describes the systematic approach required to apply a design pattern effectively. It consists of seven important steps and Figure 2.1 depicts the methodology in the form of diagram.

1. Read the pattern once through for an overview.

Designers must pay particular attention to *Intent*, *Applicability,* and *Consequences* sections to ensure the pattern is right for their design problem.

2. The study of Structure, Participants, and Collaborations

Designers must make sure that they understand the classes and objects in the pattern and how it relates to one another.

3. The review of sample code

Studying the code helps designers to learn how to implement the pattern.

4. Selection of names for pattern participants

The names for participants in design pattern are usually too abstract to appear directly in an application. Nevertheless, it is useful if designers incorporate the participants' names into the name that appears in the application. This helps to make the pattern more explicit in the implementation.

5. Define classes.

It is important to declare pattern interfaces, establish the inheritance relationships and define the instance variables that represent data and object references. After that, designers identify existing classes in the application that may affect the pattern and modified them accordingly.

6. Define application-specific names for operations in the pattern.

The names generally depend on the application. Designers can use the responsibilities and collaborations associated with each operation as a guide. The naming conventions must be consistent.

7. Implementation of the operations in carrying out the responsibilities and collaborations in the pattern.

The implementation section offers hints to guide designers in the implementation.

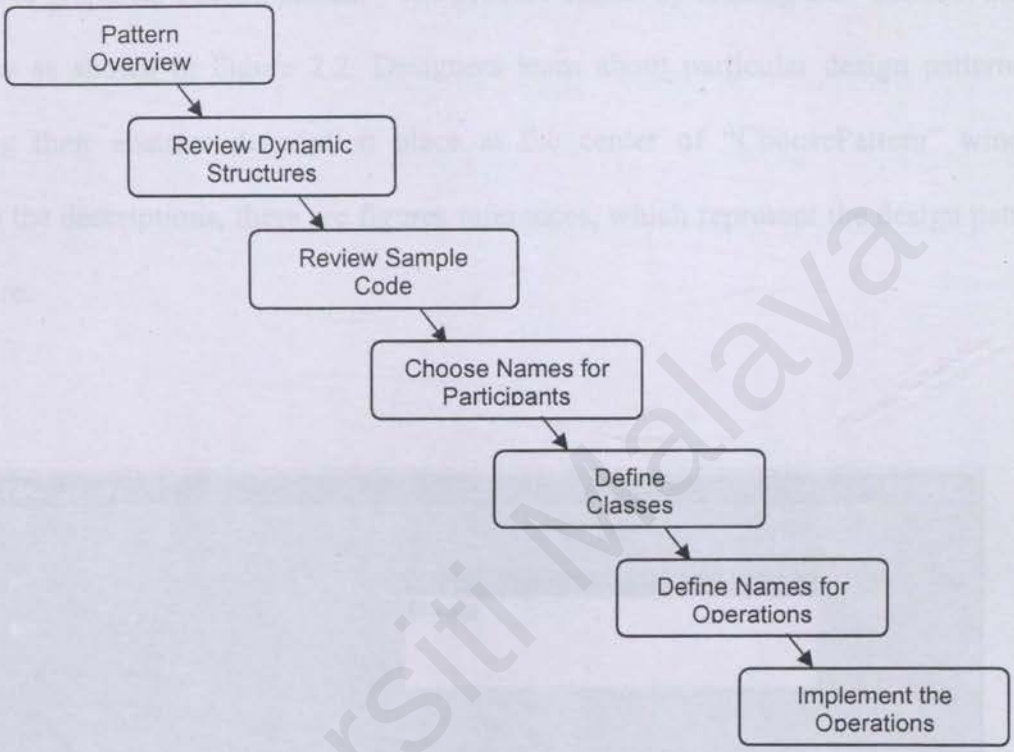

Figure 2.1: The design pattern methodology

2.2 Pattern Code Generator Tools

This section describes the existing pattern code generator tools. It is divided into two sub-sections. In the first sub-section, a review of non-web based pattern code generator tools was made. In the second sub-section, web-based pattern code generator tools are described. Based on this review, the main features for pattern code generator tool are identified.

2.1.1 Non-Web Based Pattern Code Generator Tools

2.2.1.1 S.C.U.P.E

S.C.U.P.E (Santa Clara University Pattern Editor) was developed by Mendoza & Hall (1998). This is the only tool that generates code for design patterns (Gamma et al., 1995) via graphical customization. The process begins by loading the "ChoosePattern" window as shown in Figure 2.2. Designers learn about particular design patterns by viewing their abstract description place at the center of "ChoosePattern" window. Within the descriptions, there are figures references, which represent the design patterns structure.

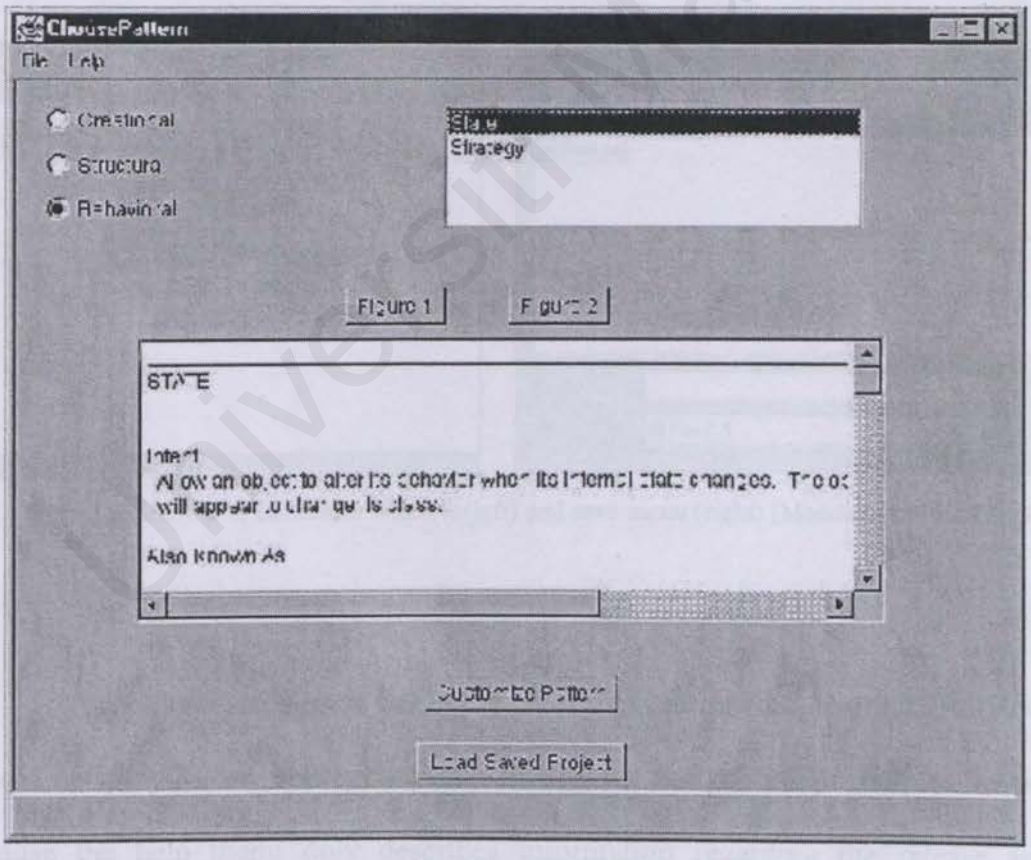

Figure 2.2: S.C.U.P.E Main window (Mendoza and Hall, 1998)

Subsequently, designers can choose to customize a predefined pattern or load pattern that they working on. Designers can graphically customize a design pattern that they have chosen. Design patterns are shown in a UML format on the screen as depicted in Figure 2.3 (left). Designers customize the pattern by changing class name, add or delete methods, method expressions, instance variables, and delete classes. These changes can be made by clicking on the class name, instance variable names or method names text boxes and edit the information directly. Once customization is completed, designers can choose either to generate Java source code for the customized design patterns or not. Then, designers need to save their design patterns as shown in Figure 2.3 (right).

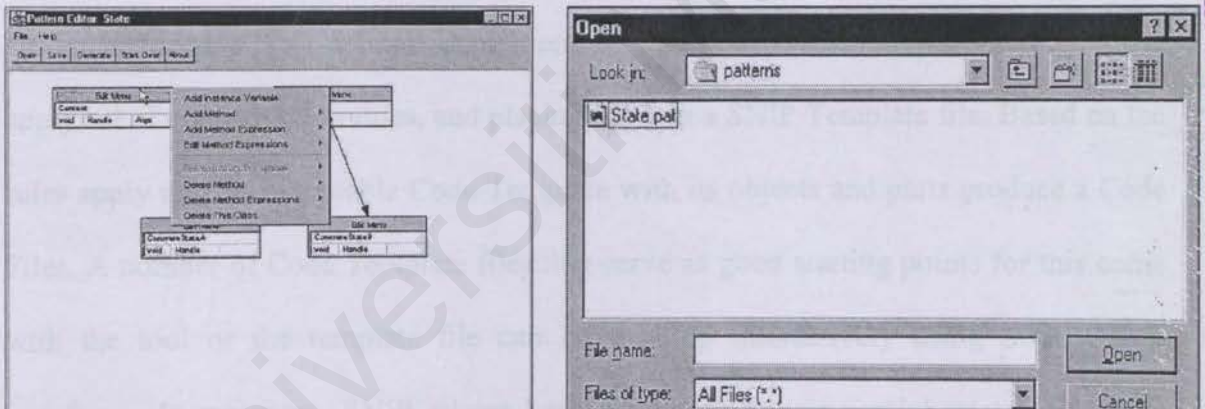

Figure 2.3: S.C.U.P.E customize window (left) and save menu (right) (Mendoza and Hall, 1998)

This tool provides great help in local design patterns implementation. However, it does not provide any appropriate help facility for designers especially to the novices because the help menu only describes information regarding the release and the

copyright. Moreover, this tool does not support distributed design patterns implementation.

2.2.1.2 SNIP

SNIP is a tool for instantiated patterns of code that can be derived from object models. It was developed by Wild (1996) in order to instantiated C++ code for patterns and these codes are defined as code patterns. Design patterns are logic in nature, whereas code patterns are physical in nature. Code patterns focus on how a particular structure or sequence of action is accomplished using the specific mechanisms of a programming language. It is also known as an idiom (Buschmann et al., 1996). SNIP generates code patterns by using object model as an instantiation context to be defined. Designers need a well-defined implementation strategy, which requires designers to apply set of code-creation rules, and placing them in a SNIP Template file. Based on the rules apply into an executable Code Template with its objects and parts produce a Code Files. A number of Code Template files that serve as good starting points for this come with the tool or the template file can be develop interactively using SNIP's user interface. In summary, SNIP allows both objects and their part characteristic to be defined, and how these objects and their parts are mapped onto code elements. The interconnection of object model and Code Template in SNIP to instantiated C++ code is illustrated in Figure 2.4.

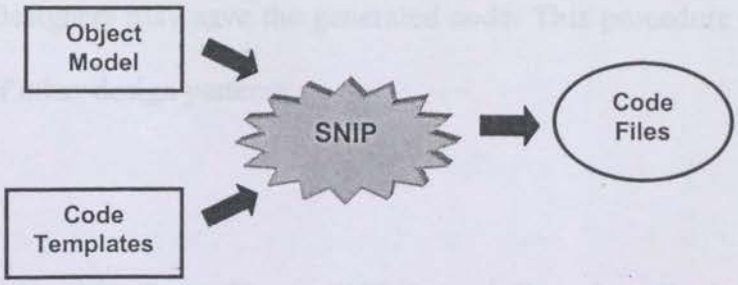

Figure 2.4: SNIP Model (Wild, 1996)

SNIP can run either interactively as a Windows MDI application or as a batch command. As a batch command, SNIP can run from inside other tools such as MS-Visual C++ or as a translation rule within a make file. Apparently, the tool supports different types of patterns.

2.1.2 Web-Based Pattern Code Generator Tools

2.2.2.1 Designer's Assistant Tool

Designer's Assistant Tool was developed by Huang (1996) in order to generate code for design patterns (Gamma et al., 1995). The "Design Pattern Space" page serves as a central focus while navigating through the tool. This page represents the 23 design patterns in a well-organized catalog as depicted in (Gamma et al., 1995). Designers can select a design pattern according to its name from the catalog in order to view the pattern abstract description page. These sections are intended to remind designers of the patterns. Apart from the customizing the design pattern, designers need to define names for class participants based on the problem domain. Upon completing the customization form, they have an option either to generate the customize pattern in C++, Java, or

Smalltalk code. Designers may save the generated code. This procedure is repeated for implementation of other design patterns.

Unlike other researchers, Huang (1996) used formal method as the process language to describe the application of the design patterns. In addition, he introduced a language based on a schema to describe objects and their interconnections. These schemas do not restrict the designers' ideas, however restricts the way the designs are expressed. The abstract schemas for objects specification and interconnection are based on Abstract Design View Model (ADV). The ADV model uses two basic object types, the Abstract Design View (ADV), and the Abstract Design Object (ADO). They represent respectively, interface objects (views and interactions) and application objects, which are interface independent. ADV can detect the identity of its corresponding ADO. However, an ADO could not identify the identity of its interface. Table 2.2 shows the schema structure used in the specification of the ADVs and explains briefly the intent of each section.

Table 2.2: A descriptive schema for an ADV (Huang, 1996)

ADV ADV_Name [*For_ADO* ADO_Name]
Declarations
Data Signatures..... .....sorts and functions
Attributes.....observable properties of the object
Input Actions. .....input actions of this view
Effectural Actions. .....actions triggered as effects of input events
Relationships Actions ..... actions triggered as effects of input events
Effectual Actions ..... aggregation, inheritance, and association
Static Properties
Constraints .....constraints on the attributes values
Derived Attributes .....*non-primitive attribute descriptions*
Dynamic Properties
Interconnection ..... description of communication among objects
Valuation ..... pre- and post-conditions for actions
Behavior ..... sequence of action occurrences in the object
End Money

ADO schemas have a structure similar to an ADV schema except that ADOs does not support input actions and does not know the identity of its corresponding ADVs. Whereas, to demonstrate the process language, design patterns have been factored into a descriptive section, and a process description. The descriptive part of the structure briefly describes the characteristics of a design pattern. The process descriptions use a simple primitive constructors language and a template or product text specification to show how objects based on ADV and ADO schemas can be interconnected in a specific design pattern. The process description part of the pattern constructors indicates the result of its application in which the interconnection between ADV and ADO schemas is indicated by the use of a template. The template encapsulates the interconnection specified in the design pattern in terms of the ADV and ADO schemas. Based on these elements, a design pattern specification meta-schema has been developed and this is shown in Table 2.3.

Table 2.3: Development constructor structure for a design pattern (Huang, 1996)

Operator **ADV_Based Design Pattern Name**	
Objective	- description of the intent of the pattern
Consequences	- how pattern in primitive constructors
Process Steps	- description of pattern in primitive constructors
Product Text	- language dependent specification of pattern
End Operator	

Based on Table 2.3 the objective section introduces the problem statement in natural language. The primitive constructors for the process of applying a design pattern are identified in the Process Steps section. The expected results are then reported in the Consequences section. Product Text describes the interconnection structure.

This tool has the ability to generate codes in different programming languages. This is a good web-based pattern code generator tool even though it uses formal method specifications. However, it neither supports visualization after the design pattern is customized nor it provides help facility.

2.2.2.2 Automatic Code Generation

Automatic code generation is a tool developed by Budinsky et al. (1996) for generating design patterns code in C++. It is often referred as Budinsky's Tool. This tool has similarity as the Designer's Assistant tool in which the customization of design pattern requires designers to define names for class participants. Although the customization is made along with choices for the design trade-offs, yet designers are not forced to accept these trade-offs. The choice of trade-offs for Composite design pattern is shown in Figure 2.5.

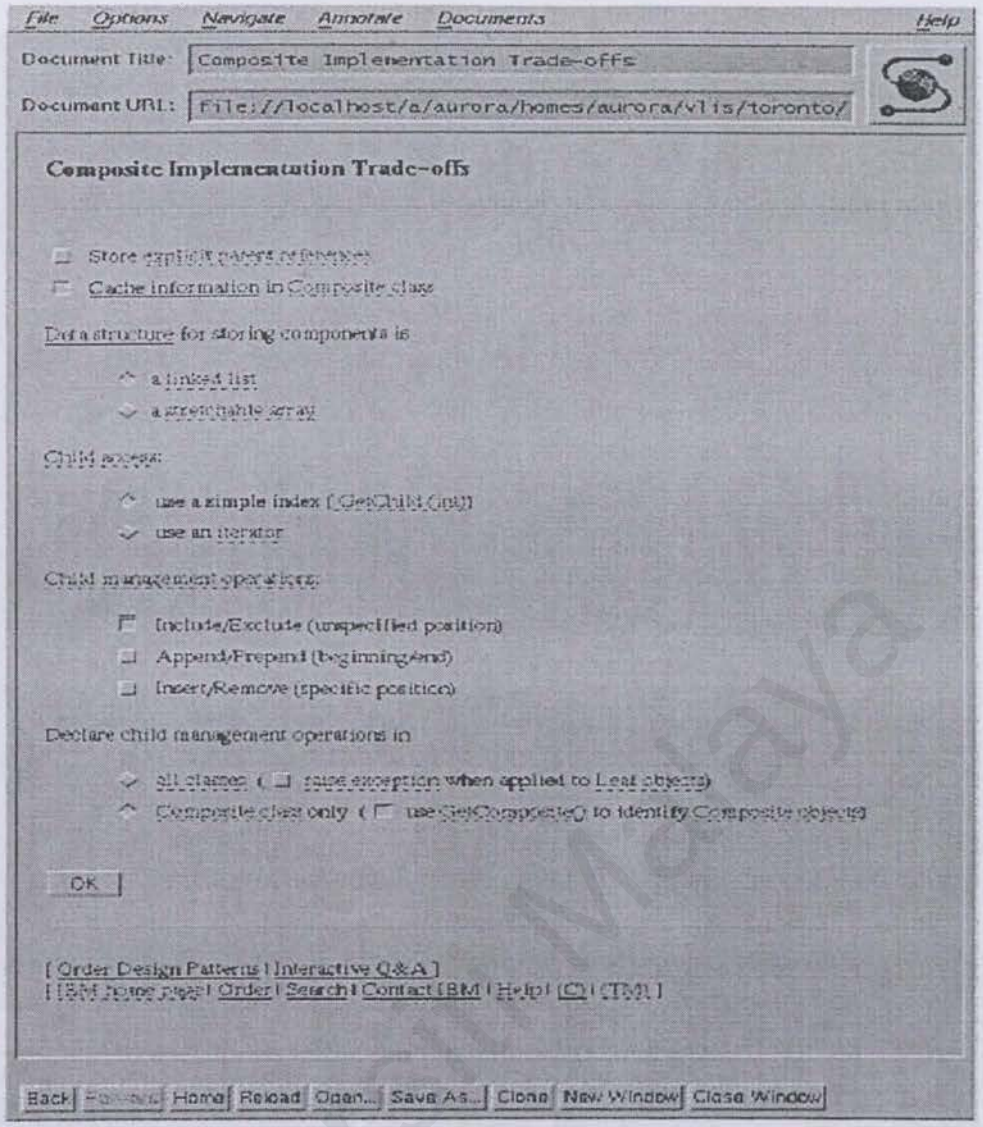

Figure 2.5: Budinsky's implementation trade-offs page (Budinsky et al., 1996)

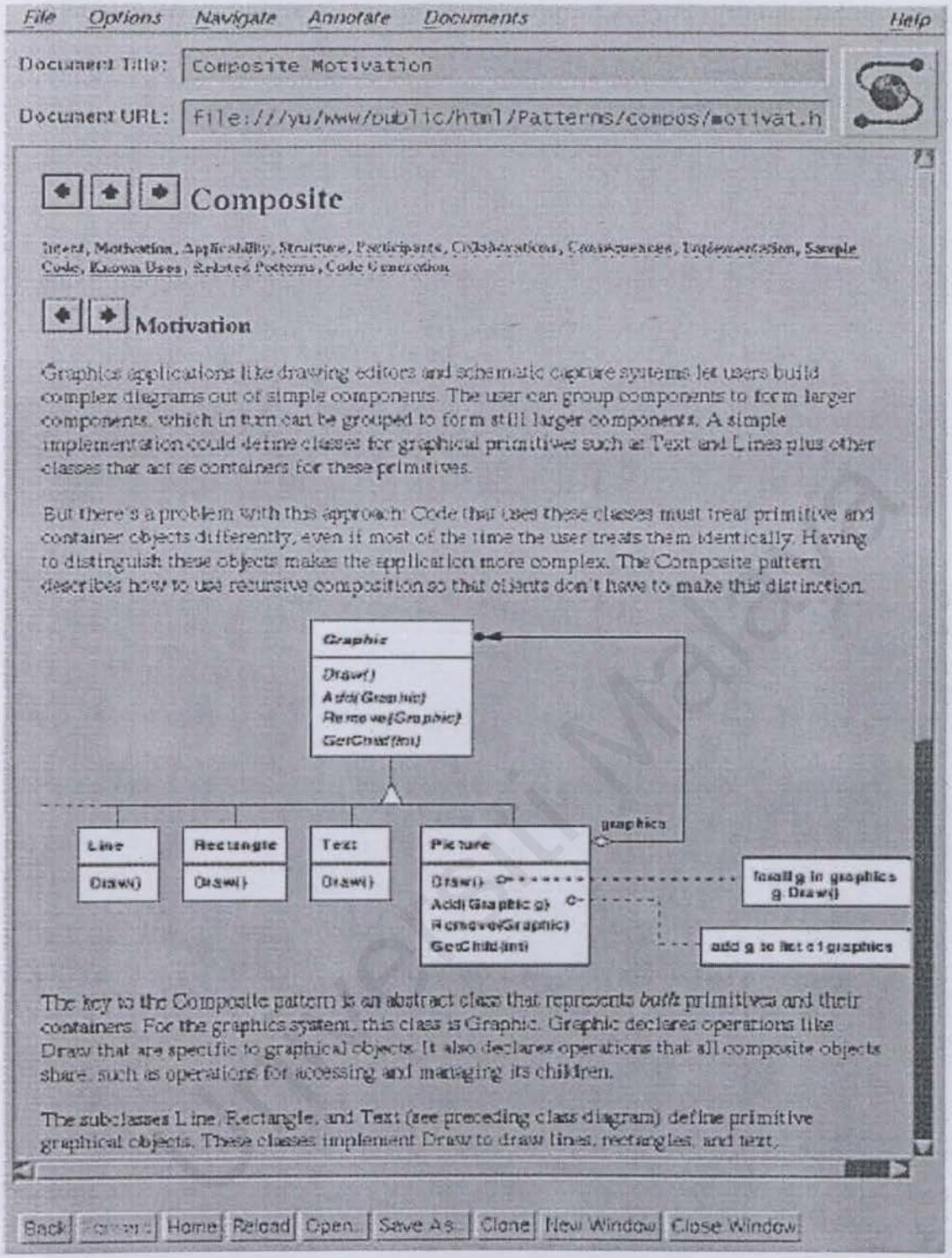

Figure 2.6: Budinsky's tool pattern description page (Budinsky et al., 1996)

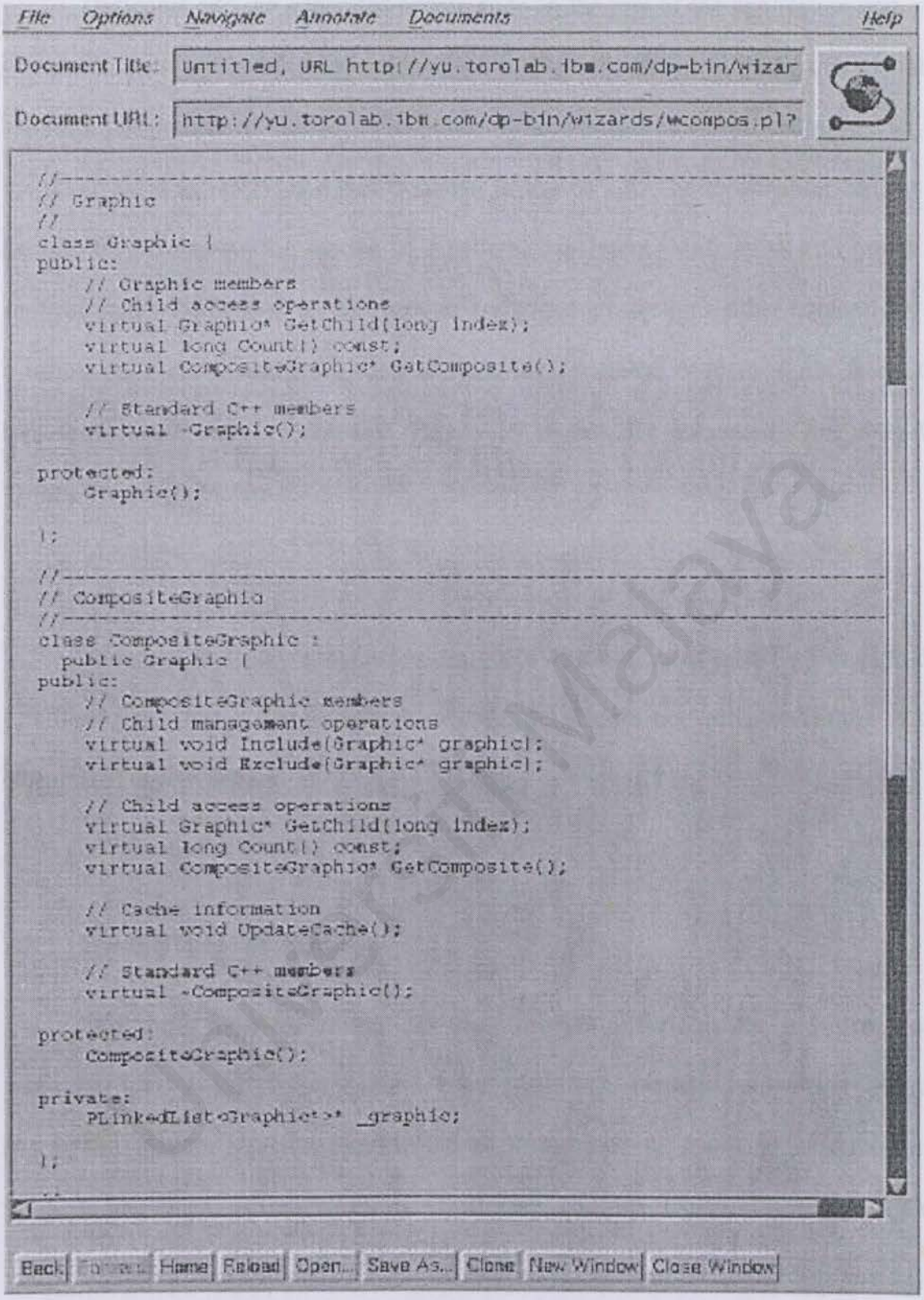

Figure 2.7: Budinsky's generated C++ code (Budinsky et al., 1996)

Designers add the generated source code to the rest of the application and will often enhance it with other application specific functionality. In addition, this tool also supports design patterns abstract descriptions, which mirror the corresponding section in the (Gamma et al., 1995) and this is shown in Figure 2.6. The information content in Section Pages displays the section of a pattern, e.g. Intent, Motivation, and others in separate pages. Designers can access other design pattern sections either randomly or in a sequence. Finally, designers need to save the generated code in a file using the browser's "Save As..." command. Figure 2.7 shows the generated C++ code for Composite design pattern.

Although Budinsky's tool offers designers with choices of selecting design trade-offs during the design pattern implementation, yet it does not provide viewing of the customized design pattern.

2.3 Identifying Main Features

The tools reviewed reveal the most important features for a pattern code generator tool. Seven features have been identified, namely: supporting abstract description, pattern structure representation, form-based or graphical customization, visualize implement pattern, trade-offs constraints, and help facility. These features are listed in Table 2.4.

Table 2.4: Summary features of pattern 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	√	√	√	√
Pattern Structure Diagram	√		√	√
Form-based Customization		√	√	√
Graphical Customization	√			
Visualize Implement Pattern	√			
Trade-offs Constraints		√	√	
Help Facility	√	√	√	

1. Supporting Abstract Descriptions

Supporting abstract descriptions contain the descriptions of static and dynamic part 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 should lead designers to make the right choice of pattern used in their design problems. The importance of these materials is proven when all reviewed tools support these features.

2. Pattern Structure Diagram

Most of the tools provide pattern structure diagram. Only SNIP does not support this feature. Pattern structure diagram is 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).

3. Form-based Customization

Form-based customization requires designers to define names for the pattern participants in a form. SNIP, Designer's Assistant and Budinsky's Tool have used this type of customization before generating codes for the pattern. The names for pattern class participants defined by designers must be meaningful in the application context.

4. Graphical Customization

S.C.U.P.E is the only tool that supports design pattern code generation via graphical customization. The diagrams are represented in UML format where designers can directly edit the diagram that will be displayed on the screen.

5. Visualize Implement Pattern

It is useful to view the pattern implementation to show the relationships between the pattern classes (Hedin et al., 1998). S.C.U.P.E is the only tool that provides this feature by visualizing the implemented design pattern in the "Customize Pattern Screen".

6. Trade-off Constraint

Trade-off constraints permit designers with substitute conditions in 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 this rules in a SNIP Template file.

7. Help Facility

Tools providing this feature is S.C.U.P.E, SNIP, and Budinsky's Tool. This feature is considered as one of the most important elements in software usability principles.

2.4 Summary

This chapter reviewed design patterns and existing pattern code generator tools thoroughly. In the first section, patterns organization and methodology for implementing design pattern were discussed. The second section reviewed the existing pattern code generator tools. It is divided into two subsections: non-web based pattern code generator tools and web-based pattern code generator tools. Finally, based on the study conducted in these two subsections, the main features for pattern code generator tools have been identified.

Chapter 3

RESEARCH FRAMEWORK

This chapter describes the framework of this research and is divided into three sections. The first section begins with an explanation of the structure design pattern code generator tool's methodology. The second section presents the structure of web-based design pattern code generator tool known as γ-CGT. Finally, the third section describes the evaluation of γ-CGT. The framework identified determines the clear boundaries for this research.

3.1 The Design Pattern Code Generator Tool Methodology

In order to build a pattern code generator tool, a methodology for the development of design pattern transformation into its concrete form must first be presented. The purpose of this methodology is to ensure that the design pattern transformation preserves the proposed tool behaviour. This methodology adapts several steps of the Gamma et al. methodology in applying the design pattern as discussed in section 2.2.2 and some new steps based on the study conducted on existing pattern code generator tools. The methodology is depicted in Figure 3.1 and its justifications are pointed out in section 3.1.1.

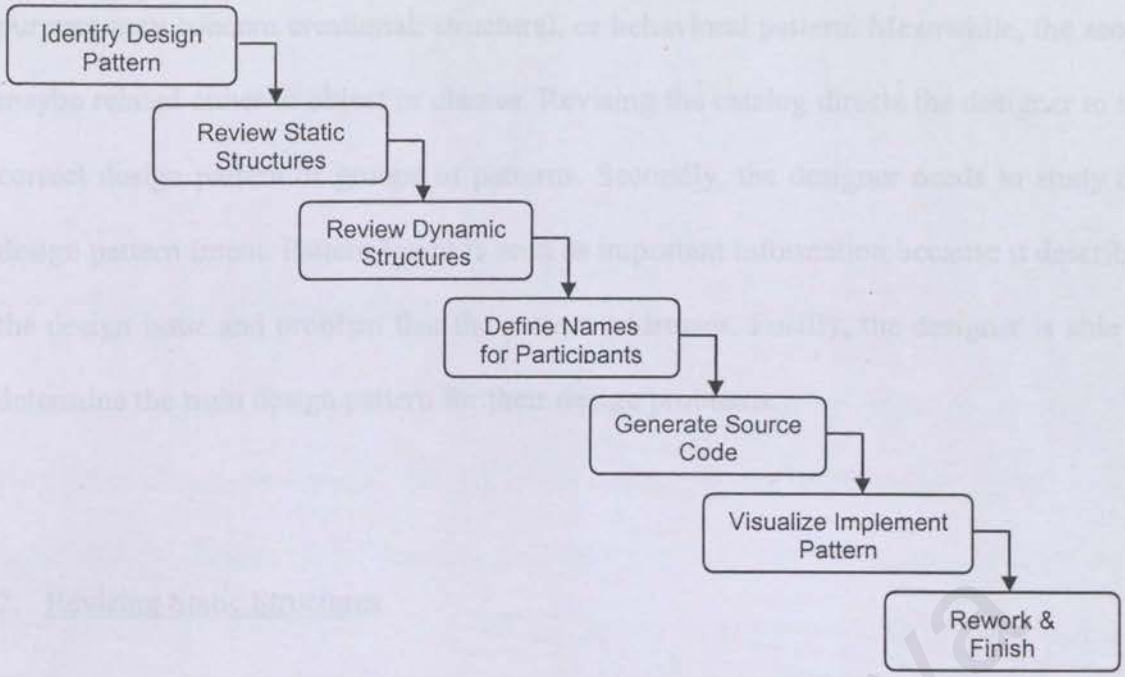

Figure 3.1: Pattern code generator tool's methodology

3.1.1 The Methodology Outline

This methodology consists of seven important steps. The justification of each steps are pointed out below:

1. Identifying design pattern

It might be a daunting task to identify a right design patterns from over 20 design patterns available in the catalog. This would be a difficult task if the catalog were new and unfamiliar to the designer. Here, two approaches are addressed to guide the designer to find the suitable design pattern for their problem. First, the selection of design pattern is made through design pattern catalog. The designer is able to narrow down the search by specifying the problem 'Purpose' and 'Scope' of the design pattern. The problem

purpose may concern creational, structural, or behavioral pattern. Meanwhile, the scope maybe related either to object or classes. Revising the catalog directs the designer to the correct design pattern or groups of patterns. Secondly, the designer needs to study the design pattern Intent. Pattern Intent is seen as important information because it describes the design issue and problem that the pattern addresses. Finally, the designer is able to determine the right design pattern for their design problems.

2. Revising Static Structures

Revise static structures allows designer to study the static structure of the design pattern. Although the design pattern template contains several parts of static structure, only two segments that are considered most important are described. These are the pattern Applicability, and Consequences. Applicability explains the situation in which the design pattern can be applied, and the examples of poor designs that the pattern can address. Meanwhile, Consequences describes the pattern trade-offs and the results of using the pattern.

3. Revising Dynamic Structures

Besides the two segments of static structure that have been examined, the designer needs to study the dynamic parts of the 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 the design pattern and their responsibilities.

4. Defining Names for Participants

Once designer has decided that the design pattern is suitable for their design problem, they proceed with the implementation of the design pattern. In this phase design pattern in its abstract form is transform into concrete form that is seen as code. Designer needs to customize the design pattern by defining names for the class participants. The names for participants in design patterns are usually too abstract to appear directly in an application. Therefore, the designer must define names that are meaningful for the participants in the application context of design problem.

5. Generation of Source Code

The source code for the design pattern generated after the design pattern has been implemented. The generated pattern code can be later integrated by the designer into their own code of application that they wish to develop. The generated code is usually presented by using object-oriented language, for example Java, C++, and Smalltalk.

6. Visualizing the Implemented Pattern

Apart from the generated code, visualization of the implemented design pattern is useful for the designer. Visualization model the implemented design pattern into class diagram.

The purpose of modelling is to lessen the difficulty for the designer to understand the complexity of the relationship in classes, or objects in the design pattern.

7. Rework & Finish

After the completion of the process of implementing the design pattern, the designer can therefore reiterate the same process for other design problem they might have. On the contrary, the designer closes the browser after using the pattern code generator tool.

3.1.2 The People

The designer is someone who intends to implement design patterns in the software development design phase. The designer may range from beginner to expert, who has different level of skills and knowledge about design patterns and its implementations.

3.2 The Implementation of a Web-Based Design Pattern Code Generator Tool

Based on past studies, a tool named γ-CGT (Gamma Code Generator Tool) was implemented. The tool includes the basic features required for the pattern code generator tool and an additional new feature that may benefit the tool. The following features are identified to constitute γ-CGT.

1. The Support of the Abstract Description

γ-CGT provides descriptions about the design patterns using a consistent format. The information includes the description of static and dynamic structures of a design pattern. It also includes the design pattern description of its Intent, Applicability, Participants, and Consequences.

2. Pattern Structure Diagram

γ-CGT illustrates the graphical representation of classes in the design pattern using a notation based on the Object Modelling Technique (OMT).

3. Form-based Customization

γ-CGT used form-based customization mechanism to customize the chosen design pattern. This type of customization requires the designer to define the names for pattern class participants using a form.

4. Visualize Implement Pattern

γ-CGT visualizes the implemented design pattern in class diagram by using Unified Modelling Language (UML) notations. The design pattern's class diagram given earlier is changed based on the customization done by the designer.

5. Help Facility

γ-CGT provides an online help throughout the design pattern implementation process.

Feature suggestion:

6. Alert Message

γ-CGT provides an alert message to the designer if an illegitimate action is committed while defining names for the pattern participants. This is to ensure that the designer has provided participants with valid names.

3.2.1 Limitation

Although γ-CGT provides some good features, it has some limitations since γ-CGT is only a prototype tool. γ-CGT supports the abstract descriptions for all design patterns in the catalog but it provide only five design patterns to be implemented by the designer. In addition, the designer is allowed to define minimum participants for design pattern.

3.3 Evaluating γ-CGT

γ-CGT evaluation is carried out by using some experimental materials. The only purpose behind this experiment is to access the feasibility of executing the implementation of the design pattern using γ-CGT.

3.4 Summary

This chapter specified the framework of this study, which is divided into three sections. In the first section, the structure of design pattern code generator tool's methodology was explained. The second section presented the structure of web-based design pattern code generator tool known as γ-CGT. Finally, the evaluations of γ-CGT were described.

Chapter 4

ANALYSIS AND DESIGN

This chapter presents the analysis and the design of the γ-CGT. In the analysis section, both the functional and the non-functional requirements are identified. The object-oriented analysis is also addressed. In the design section, the architecture of γ-CGT, the object-oriented design and some aspects of the user interface design are presented.

4.1 γ-CGT Analysis

4.1.1 Requirements Analysis

There are two main concerns of the γ-CGT requirements: functional and non-functional requirements. Functional requirements describe the functionality of γ-CGT. Meanwhile, non-functional requirements describe aspects such as usability, portability, and other run time properties.

4.1.1.1 γ-CGT Functional Requirements

The first major requirement for γ-CGT is to implement the methodology that has been presented in chapter 3. Basically, each step is considered as one phase and the functionality of the tool is different with one another. The functional requirements of each phase are stated in table 4.1:

Table 4.1: γ-CGT functional requirements

PHASE	FUNCTIONAL REQUIREMENTS	
Step 1	**Identify Design Pattern**	
	F.R 1.1	The designer loads web client.
	F.R 1.2	The web client displays the mainDoc.
	F.R 1.3	The designer retrieves the design pattern catalog.
	F.R 1.4	The design pattern catalog is displayed in catalogBrowser.
	F.R 1.5	The designer revises design pattern catalog.
	F.R 1.6	The designer chooses a design pattern
	F.R 1.7	The catalogBrowser retrieves the design pattern abstract descriptions.
	F.R 1.8	The design pattern abstract description is displayed in patternBrowser.
	F.R 1.9	The designer revises design pattern Intent.
Step 2	**Revise Static and Dynamic Structures**	
	F.R 2.1	The designer loads web client.
	F.R 2.2	The client displays mainDoc.
	F.R 2.3	The designer retrieves design pattern catalog.
	F.R 2.4	The design pattern catalog is displayed in catalogBrowser.
	F.R 2.5	The designer chooses a design pattern.
	F.R 2.6	The catalogBrowser retrieves the design pattern details.
	F.R 2.7	The design pattern abstract description is displayed in patternBrowser.
	F.R 2.8	The designer revises the design pattern static structures.
	F.R 2.8	The designer revises the design pattern dynamic structures.
Step 3	**Define Names for Participants**	
	F.R 3.1	The designer loads web client.
	F.R 3.2	The client displays patternBrowser.
	F.R 3.3	The designer retrieves wizardBrowser.
	F.R 3.4	The wizardBrowser displays list of design patterns based on pattern names.
	F.R 3.5	The designer chooses a design pattern from the list.
	F.R 3.6	The wizardBrowser retrieves the design pattern participant form.
	F.R 3.7	The design pattern participant form is displayed in formBrowser.
	F.R 3.8	The designer defines names for design pattern participants.

Table 4.1, continue

Step 4	**Generate Source Code**	
	F.R 4.1	The designer loads web client.
	F.R 4.2	The client displays wizardBrowser.
	F.R 4.3	The designer chooses a design pattern from a list.
	F.R 4.4	The wizardBrowser retrieves the design pattern participant form.
	F.R 4.5	The design pattern participant form is displayed in formBrowser.
	F.R 4.6	The designer defines names for design pattern participants.
	F.R 4.7	The formBrowser passes the names to the server.
	F.R 4.8	The server generates source code for the design pattern.
	F.R 4.9	The server passes the source code to the sourcecodeBrowser.
	F.R 4.10	The sourcecodeBrowser displays the generated source code.
	F.R 4.11	The designer views the generated source code.
Step 5	**Visualize Implement Pattern**	
	F.R 5.1	The designer loads web client.
	F.R 5.2	The client displays wizardBrowser.
	F.R 5.3	The designer chooses a design pattern from a list.
	F.R 5.4	The wizardBrowser retrieves the design pattern participant form.
	F.R 5.5	The design pattern participant form is displayed in formBrowser.
	F.R 5.6	The designer defines names for design pattern participants.
	F.R 5.7	The formBrowser passes the names to the server.
	F.R 5.8	The server generates class diagram for the design pattern.
	F.R 5.9	The server passes the class diagram to the diagramBrowser.
	F.R 5.10	The diagramBrowser displays the generated class diagram.
	F.R 5.11	The designer views the generated class diagram.
Step 6	**Rework & Finish**	
	F.R 6.1	The designer repeats the process to implement design pattern, or
	F.R 6.2	The designer exit from γ-CGT.

4.1.1.2 γ-CGT Non-Functional Requirements

In addition to the above functional requirements, some other specific non-functional requirements have been taken into account when building γ-CGT. These non-functional requirements are:

1. Usability

γ-CGT is built in a way that portrays its functionality. One main issue, which reflects the functionality behind software tools, is the user interface. Therefore, the user interface is designed in a way that mimics the pattern catalog as in (Gamma et al., 1995).

2. Portability

In order to distribute the knowledge and implementation of design patterns, γ-CGT is accessible from the normal web-browser and it runs on multiple platforms.

3. Maintainability and expandability

γ-CGT is developed to be easily updated, maintained, and expanded.

4.1.2 γ-CGT Object-Oriented Analysis (OOA)

Object-oriented analysis technique has been used to identify the different aspects of γ-CGT. The Unified Modelling Language (UML) notations are used to represent the γ-CGT diagrams (UML, 2001). For the sake of simplicity, only one of the diagrams is presented in this chapter. The complete diagrams are presented in *Appendix A*.

4.1.2.1 The Unified Modelling Language

UML is a common notation that is used to specify, visualize and document the artefacts of an object-oriented system under development (Quatrani, 2000). It is developed based on the unification of three most popular analysis and design methodologies. There are Object Modelling Techniques (OMT), Booch technique, and Object Oriented Software Engineering (OOSE). There are two main steps that followed through the analysis phase:

- Use-Case Modelling

Describe the use of the system and show the courses of events that can be performed. This information is presented in the form of use-case diagrams and associated scenarios. This step is sometimes referred to as functional modelling.

- Class Modelling

Determine the classes and their attributes and the relationship between the classes. This information is usually presented in the form of class diagrams.

4.1.2.2 Identifying γ-CGT Use-Cases

The use-cases for γ-CGT are shown in Figure 4.1. There are six identified use-cases: identify design pattern, revise static and dynamic structures, defines names for class participants, generate source code, visualize the customize pattern, and lastly reiterate the above process or exit from the tool.

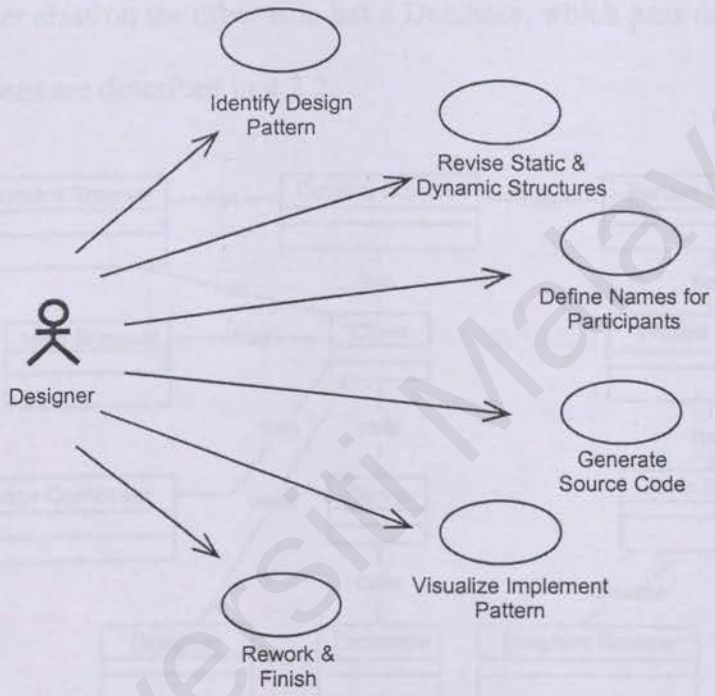

Figure 4.1: γ-CGT Use-cases

4.1.2.3 Class Modelling

From the requirement statement and the above use-cases diagram, the class diagram is depicted in Figure 4.2.

It appears from the class diagram that there are two main classes: Client class, and the Server class. Each uses a number of classes to accomplish its task. The Client class for example, has a main document browser, help browser, message composer, catalog browser, pattern browser, wizard browser, form browser, diagram browser, and code browser. The designer class uses the client side to access the entire browser.

The Server class on the other side has a Database, which pass data to client class. The detailed classes are described in 4.2.2.

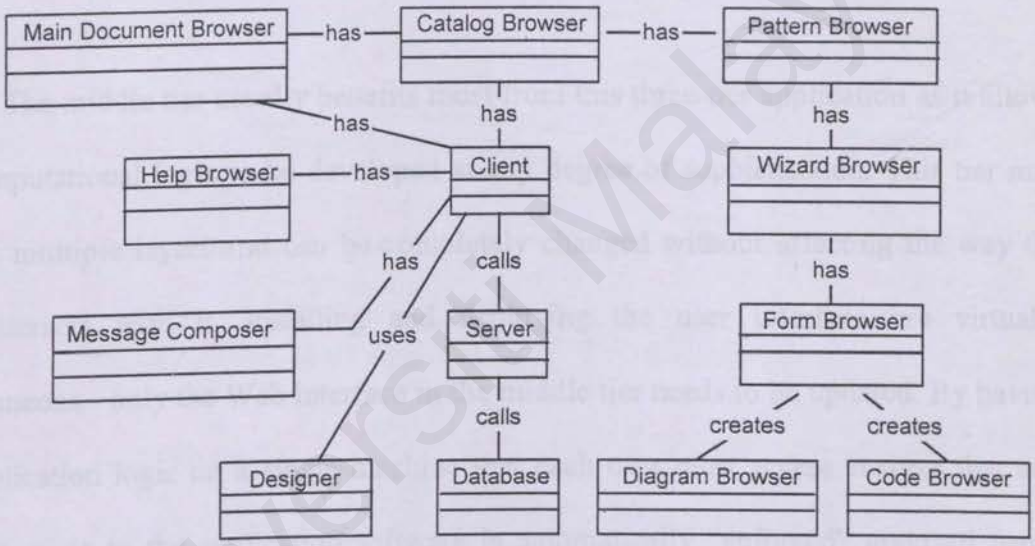

Figure 4.2: γ-CGT class diagram

4.2 γ-CGT Design

In order to satisfy the requirements identified in the analysis section as well as some of the features identified in chapter two, the following considerations were taken into account while designing the γ-CGT:

4.2.1 γ-CGT Architecture

γ-CGT is designed based on the multi-tier architecture, specifically the three-tiers application and this is depicted in Figure 4.3. This multi-tier architecture provides many benefits over traditional (two-tiered) client/server architecture (Nakhimovsky and Myers, 1999). By splitting an application across three tiers, three logical components of the application can be separated: user interface, computational logic, and data storage. Each logical unit can then be developed separately from the other, thus introducing an enormous degree of flexibility into the design of application.

The middle tier usually benefits most from this three-tier application as it allows the computational logic to be developed at any degree of sophistication. This tier may contain multiple layers and can be completely changed without affecting the way the user interacts with it. Installing and deploying the user interface are virtually instantaneous - only the Web interface in the middle tier needs to be updated. By having the application logic on a single machine that each user must access ensures that any upgrade made to the application software is automatically "enforced" upon all users. This will avoids the nightmare of maintaining different versions of the same application. Without a "thick" client interface, it is easier to deploy, maintain, and modify applications, no matter where the client is located.

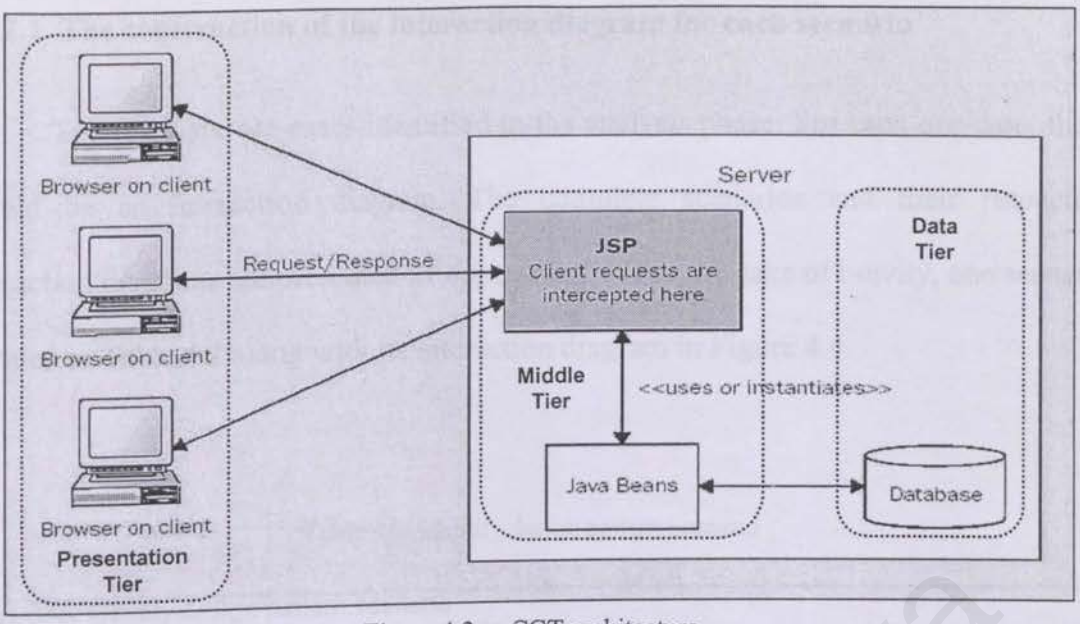

Figure 4.3: γ-CGT architecture

4.2.2 γ-CGT Object-Oriented Design (OOD)

In the object-oriented design phase, the following steps are followed:

- The construction of interaction diagrams for each scenario

This can be represented using sequences diagram or collaboration diagrams. Both diagrams show the different objects and the messages passed between them.

- The construction of the detailed class diagram

In the analysis phase, the class diagram depicts the classes and some of their attributes only. In the design phase, some other attributes and methods are added to the class diagram. These methods are usually derived from the interaction diagrams of all scenarios.

4.2.2.1 The construction of the interaction diagram for each scenario

There are six use-cases identified in the analysis phase. For each use-case, there should be an interaction diagram. The complete scenarios and their respective interaction diagrams are presented in *Appendix A.2*. For the sake of brevity, one scenario is stated in Table 4.2 along with its interaction diagram in Figure 4.4.

Table 4.2: Identify design pattern scenario

A designer identify design pattern scenario
1- The designer loads web client. 2- The client displays mainDoc. 3- The designer retrieves design pattern catalog. 4- The design pattern catalog is displayed in catalogBrowser. 5- The designer revises the design pattern catalog. 6- The designer chooses a design pattern. 7- The catalogBrowser retrieves the design pattern details. 8- The design pattern abstract description is displayed in patternBrowser. 9- The designer revises the design pattern Intent.

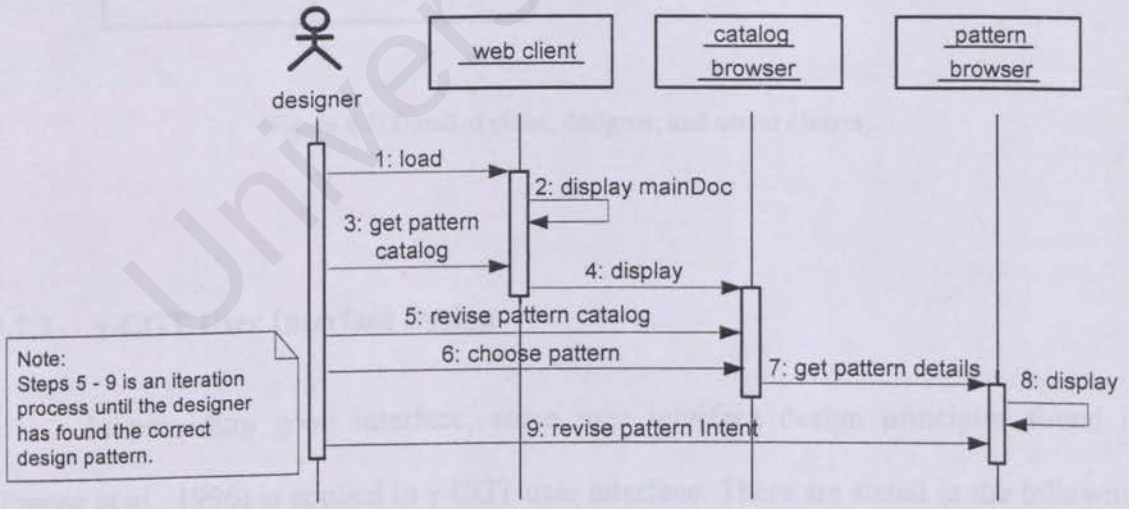

Figure 4.4: Identify design pattern interaction diagram

4.2.2.2 The construction of the detailed class diagram

By using the scenarios and their respective diagrams, the detailed attributes and methods are derived. The detailed class diagrams are similar to the diagram shown in Figure 4.2. However, the attributes and methods have been added. Some of the main classes with their full attributes and methods are presented in Figure 4.5. The rest of the classes and their complete attributes are found in *Appendix A.3*.

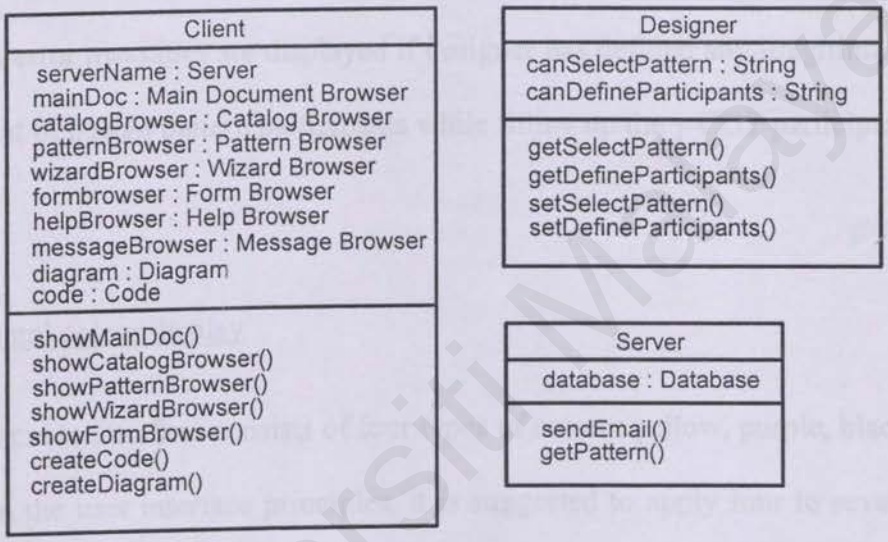

Figure 4.5: Detailed client, designer, and server classes

4.2.3 γ-CGT User Interface Design

In providing good interface, some user interface design principles found in (Preece et al., 1996) is applied in γ-CGT user interface. There are stated in the following section:

- Maintaining uniformity and consistency

γ-CGT consistency emerges from standard operations and representations. For example, in γ-CGT every page has the same basic "look and feel" to ensure a consistent format and intuitive interface.

- Error messages for unacceptable values

γ-CGT displays an error messages on any erroneous input given by the designer. For instance, error messages are displayed if designer has defined any illegitimate names for the format of design pattern participants while filling up the γ-CGT participant form.

- Minimal colour display

γ-CGT screen interface consists of four types of colour: yellow, purple, black, and grey. Based on the user interface principles, it is suggested to apply four to seven colours in the interface in order to avoid problems of distractions and confusions.

- Default command

γ-CGT provides a reset button in the γ-CGT wizard to ensure that the designer is able to retrieve back any default value in the wizard. For example, the default names for the design pattern participants.

4.3 Summary

This chapter presents γ-CGT analysis and the design. In the first part of this chapter, γ-CGT functional and non-functional requirements have been stated. This was followed by object-oriented analysis in which the different use-cases with their respective scenarios were identified. The main classes were also modelled and presented in the form of class diagram. In the second part, γ-CGT architecture has been presented. This has been followed by the object-oriented design in which the uses-cases identified in the analysis part have been transformed into interaction diagrams. These interaction diagrams uncovered the behaviour of γ-CGT classes. Finally, some aspects of γ-CGT user interface have been presented.

Chapter 5

IMPLEMENTATION AND EXECUTION

This chapter describes the different aspects of γ-CGT implementation and its execution. The first section begins by describing the implementation of environment in which the communication infrastructure, the implementation programming language, the database option and the web server are described. In the second part of this section, γ-CGT main objects and their implementation are presented. In the following sub-section, the implementation of γ-CGT 's main features is shown.

5.1 γ-CGT Implementation

Many elements are used to implement γ-CGT. Some of these elements are related to the development environment and some others are related to the technical aspects of γ-CGT's functions. These elements are described below:

5.1.1 Implementation Environment

5.1.1.1 The Communication Infrastructure

The most important motivation behind building γ-CGT is the exploration of the WWW as a medium of communication in building web tools. The WWW offers a number of characteristics over other available communication medium. These are as follow:

- The WWW is highly platform-independent. This allows the same code to be run on different programming operating systems such as Windows and UNIX.

- The WWW is global.

- No accessing time limits. Information stored on the Web pages can be accessed at any time. This allows users to work on their individual preferences instead of being confined within working hours.

5.1.1.2 The Programming Language

Java Server Pages (JSP) has been chosen to implement γ-CGT. JSP is part of Java 2 Platform Enterprise Edition (J2EE) architecture and is considered as a Java programming language at the server side as shown in Figure 5.1. As part of the JavaTM family, the JSP technology enables rapid development of web-based applications that are platform independent. JSP pages are efficient since the JSP loads into the web server memory on receiving the request at very first time and the subsequent calls are served within a very short period.

In addition, JSP technology separates the user interface from content generation enabling overall page layout to change without altering the underlying dynamic content. The application logic residing in server-based resources are known as JavaBeans component architecture that the JSP page accesses with tags and scriptlets. By separating the page logic from its design and display may supports a reusable

component-based design. JSP technology makes it faster and easier than ever to build and maintain information-rich and dynamic web-based applications.

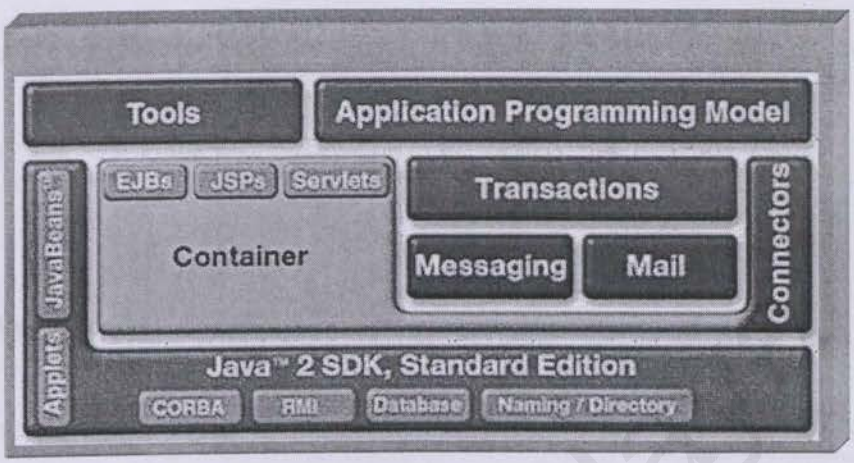

Figure 5.1: JSPs and JavaBeans resides in the Java™ 2 Platform Enterprise Edition (J2EE) architecture (Sun, 2001)

Besides JSP, Java Applet has been used to generate visualization support for the design patterns. A Web browser executes Java Applet when the browser loads a JSP pages that contains an applet tag. The applet source code is stored in *.java file and its bytecode is stored in *.class file. It is included in the JSP pages. The applet tag defines the width and height of the applet window within the JSP pages. The applet tag has numerous attributes to enhance its placement within the JSP pages.

5.1.1.3 The Database

The DataSet is used as data storage that resides in data tier of γ-CGT architecture. The DataSet class is an abstract class that provides basic editing, view, and

cursor functionality for accessing two-dimensional data. This component or any other component that extends from StorageDataSet can be used to directly access tables stored in a DataStore database file. This component can be attached to any User Interface (UI) control in the same way that other subclasses of StorageDataSet connect to a UI control. It thereby, mimics single-user SQL server functionality although no database connection is involved.

5.1.1.4 The Web Server

The Inprise Application Server (IAS) has been chosen to host γ-CGT. IAS is a suite of development and runtime facilities that allow users to build dynamic, scalable and high performing web applications. One of the major components of the AppServer is a Web container, designed to support development and deployment of web applications. The web container contains a Tomcat container, which provides the JSP compiler and engine. The first big advantage of IAS over most of its competitors is that IAS is built on CORBA and RMI-IIOP, which has already been proven to be reliable and scalable (Borland, 2001).

5.1.2 γ-CGT Objects

γ-CGT is composed of several objects used to accomplish the design pattern implementation process. The most important objects are:

1. Main document browser.

The main document browser is the initial screen layout for γ-CGT. The browser has been implemented to offer some important information about γ-CGT to the designer before they begin using the tool. This description includes a brief overview and available features of γ-CGT. After going through the information, the designer proceeds to the next browser that is the catalog browser to start the design pattern implementation process. By clicking the link from γ-CGT icon name it direct the designer to that browser. The browser is shown in figure 5.2.

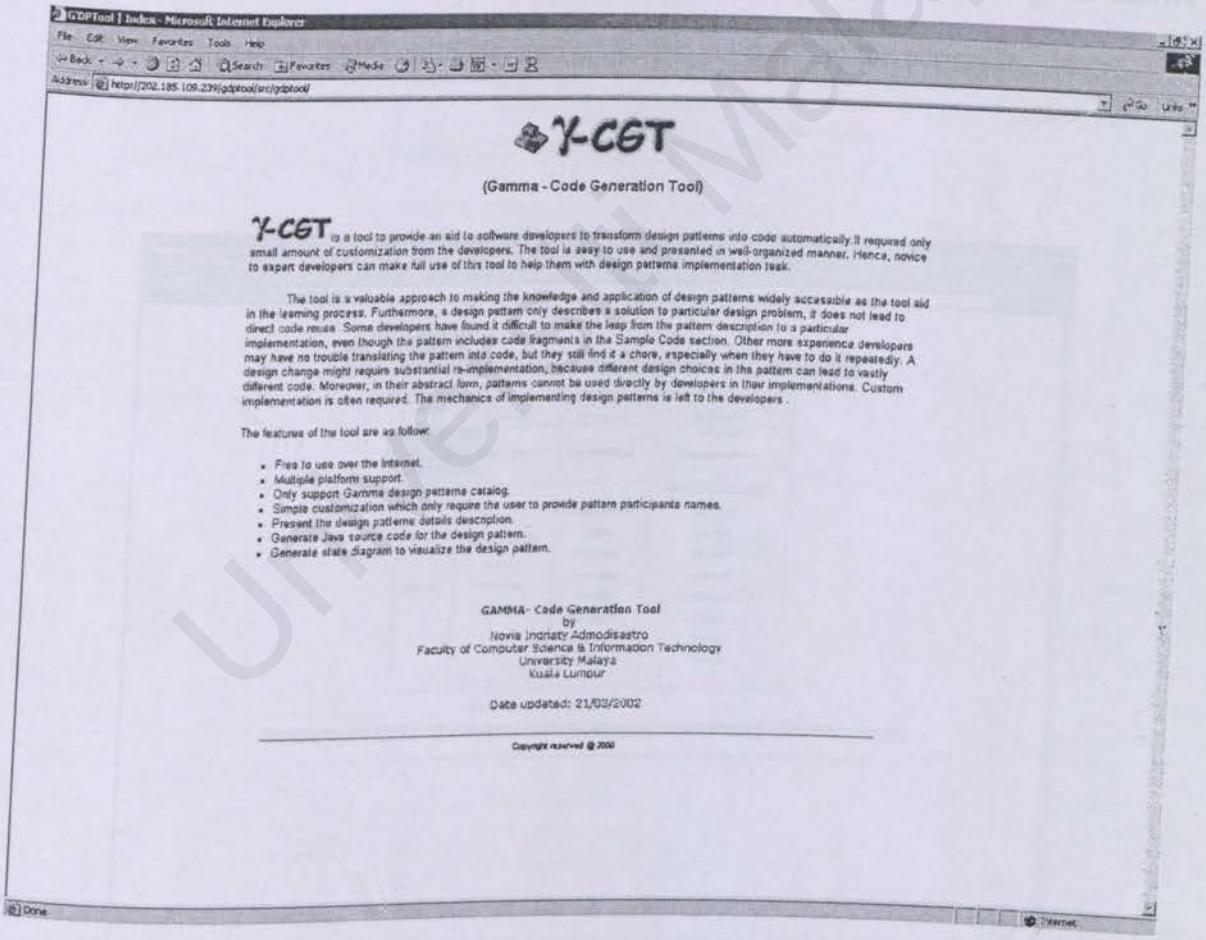

Figure 5.2: γ-CGT main document browser

2. Catalog browser

The catalog browser serves as the central focus while it can navigate through the tool and this is shown in Figure 5.3. The catalog browser is implemented to facilitate the design pattern identification stage in the implementation of the design pattern process. The design pattern catalog mimics the pattern catalog found in Gamma et al. (1995). The purpose of having the same layout is to flatten the designer's learning process through familiarization of the Gamma et al. (1995) book they might have referred to before. Moreover, the organization of the catalog is proven to enable the designer identify the required design pattern quickly and easily. In addition, the browser provides links to the main document browser, help menu, and email composer. The email composer allows the designer to send their comments and enquiries via email to the administrator.

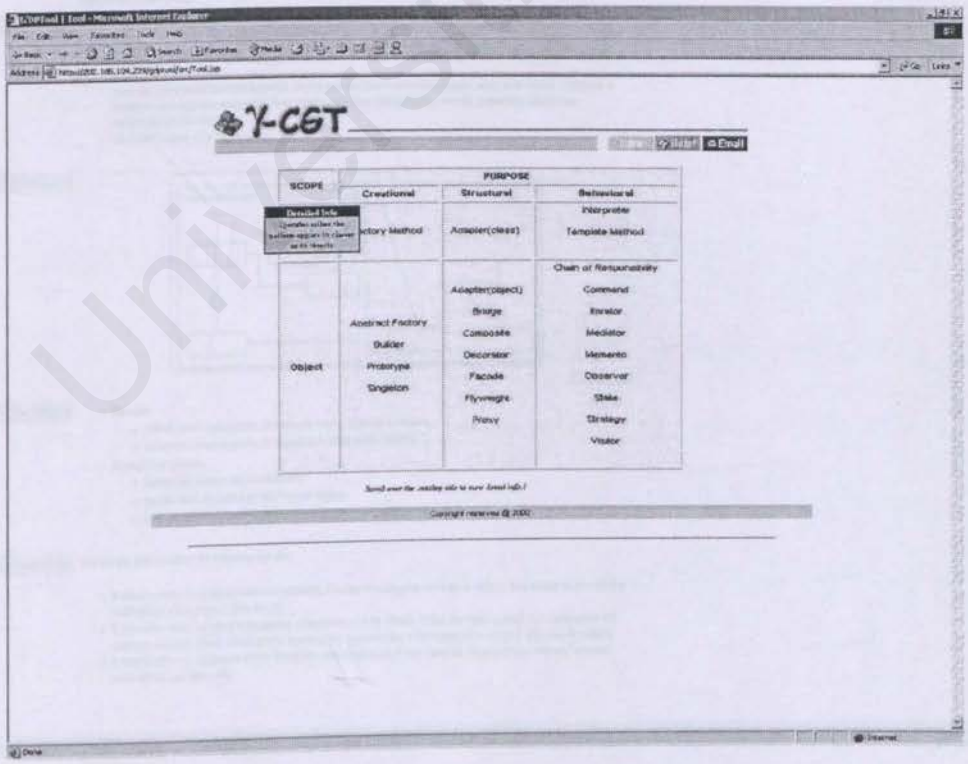

Figure 5.3: γ-CGT catalog browser

3. Pattern browser

The pattern browser is used to provide the designer with the detail descriptions of the design pattern that they have chosen. The descriptions include the design pattern Intent, Applicability, Consequences, Structure, and Participants. The top of the template shows the design pattern's name with its scope and purpose. Besides the template assists the designer to ascertain if whether they have chosen the right design pattern. On the contrary, the designer can use the Home link to re-visit the catalog browser to choose other patterns if the design pattern is not suitable for their design problem. The pattern browser is depicted in Figure 5.4.

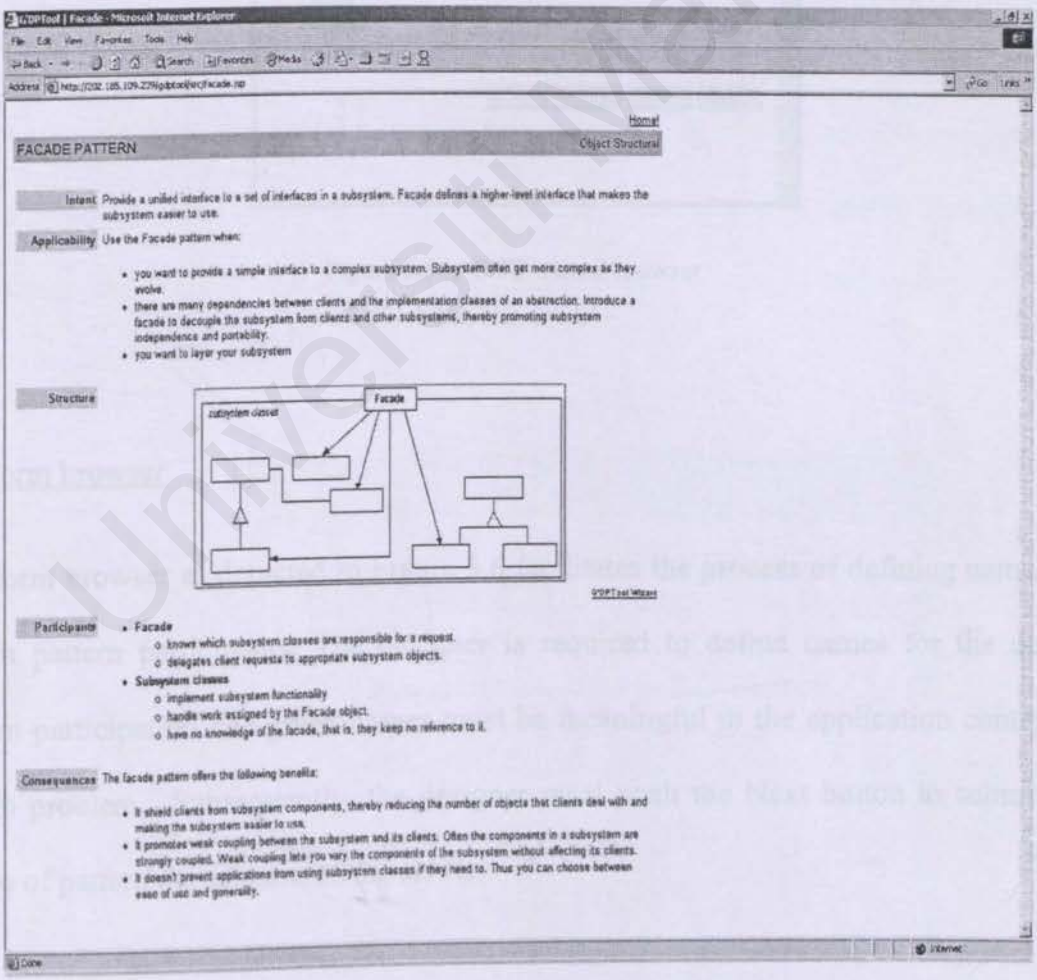

Figure 5.4: γ-CGT pattern browser

4. Wizard browser

The wizard browser shown in Figure 5.5 provides the designer with the list of design pattern names. The designer must tick the bullet box containing the design pattern's name once they have decided the design pattern that they want to implement. Thereafter, the designer needs to click the Next button to invoke the form browser that is related to the design pattern.

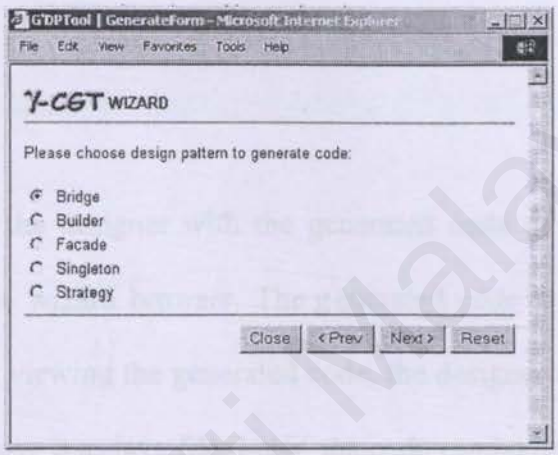

Figure 5.5: γ-CGT wizard browser

5. Form browser

The form browser as depicted in Figure 5.6 facilitates the process of defining names for design pattern participants. The designer is required to define names for the design pattern participants. The given names must be meaningful in the application context of design problem. Subsequently, the designer must push the Next button to submit the names of pattern participants to the server.

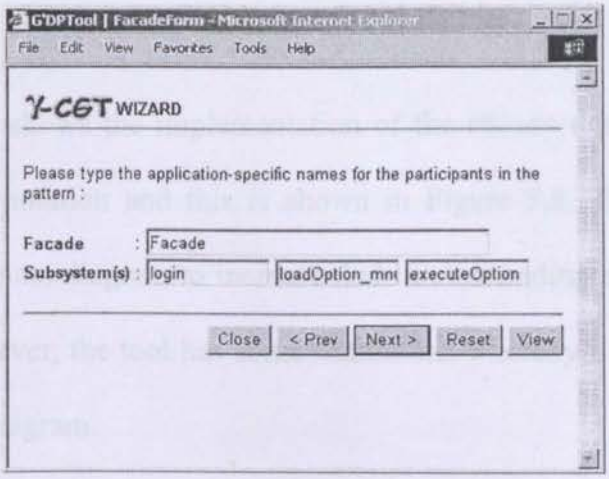

Figure 5.6: γ-CGT form browser

6. Code Browser

This browser provides the designer with the generated code of design pattern, which they have chosen in the wizard browser. The generated code is in Java programming language form. Besides viewing the generated code, the designer can copy and paste the code into notepad and save it as java file. Later, the code can be integrated into their own application code they wish to develop. Example of C++ generated code for design pattern is shown in Figure 5.7.

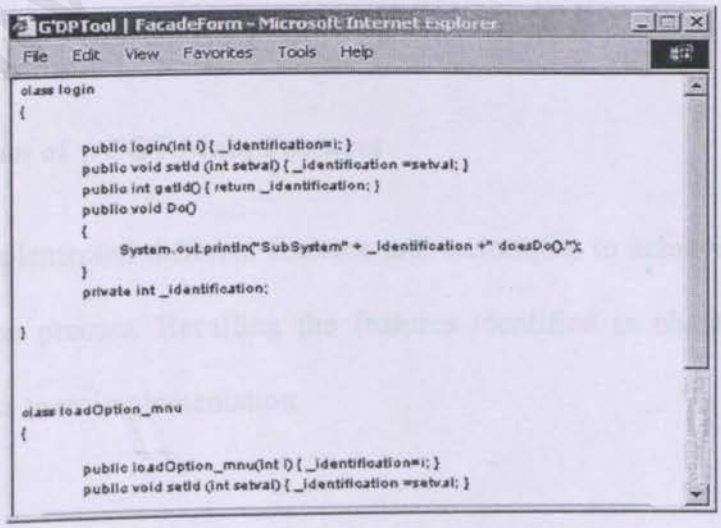

Figure 5.7: γ-CGT pattern generated code browser

7. Diagram Browser

The diagram browser shows the implementation of the chosen design pattern in class diagram using UML notation and this is shown in Figure 5.8. The designer is only allowed to view this class diagram to increase their understanding of the design pattern implementation. However, the tool has some limitations whereby it does not permit the designer to save this diagram.

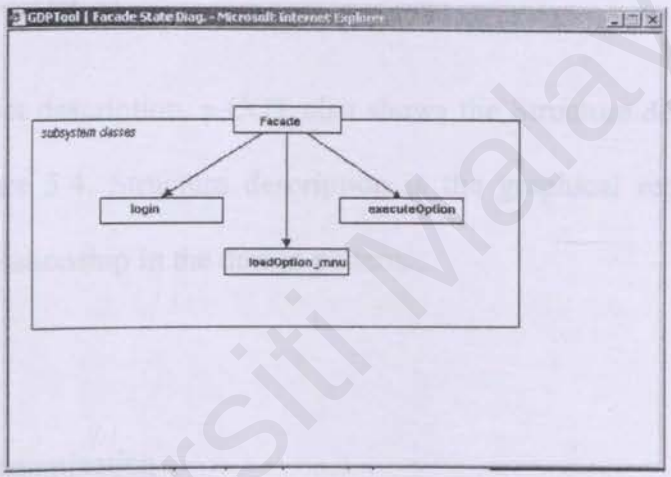

Figure 5.8: γ-CGT diagram browser

5.1.3 Implementation of γ-CGT Main Features

γ-CGT has implemented different features and techniques to achieve the design pattern implementation process. Recalling the features identified in chapter two, the following section show their implementation.

1. Supporting Abstract Description

Supporting abstract description is vital in the pattern code generator tool. This feature leverage the material found in Gamma et al. (1995) to help the designers to refer to this material while using the γ-CGT. To be most effective, therefore, the book's content especially the precise topic of interest to the designer is accessible from γ-CGT. The description includes the pattern Intent, Applicability, Participants, and Consequences.

2. Diagram Representation

Besides, the abstract description, γ-CGT also shows the Structure description of design pattern as in Figure 5.4. Structure description is the graphical representation of the classes or object relationship in the design pattern.

3. Form-based Customization

γ-CGT allows the designer to customize design pattern by defining names for design pattern participants. This requires the designer to define names for the design pattern participants in text fields that have been displayed in the pattern form. This method of customization is quite simple and easy to be done. Figure 5.9 shows pattern participants form for bridge design pattern.

Figure 5.9: Participants customization wizard

4. Customize Pattern Visualization

γ-CGT has supporting visualization for the design pattern implementation in class diagram based on UML notations. The class diagram is displayed in the diagram Browser and this is shown in Figure 5.10.

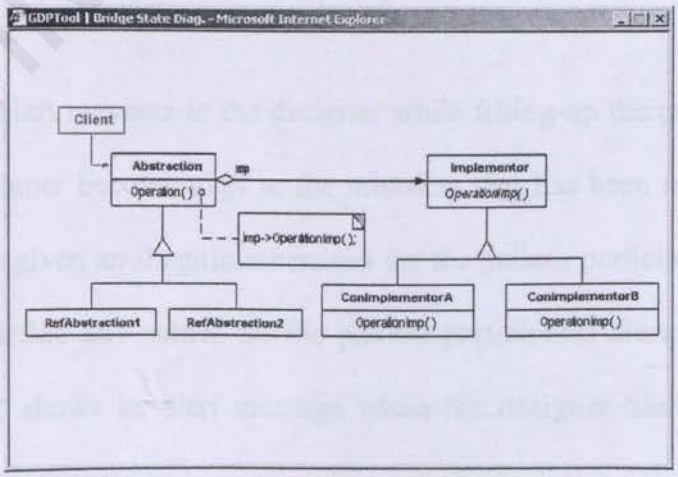

Figure 5.10: γ-CGT customize design pattern visualization

5. Help Facility

Help facility is important to assist the designer during the design pattern implementation process. γ-CGT provides an on-line help facility throughout the design pattern implementation process. The help window allows the designer to view the design pattern template details as shown in Figure 5.11.

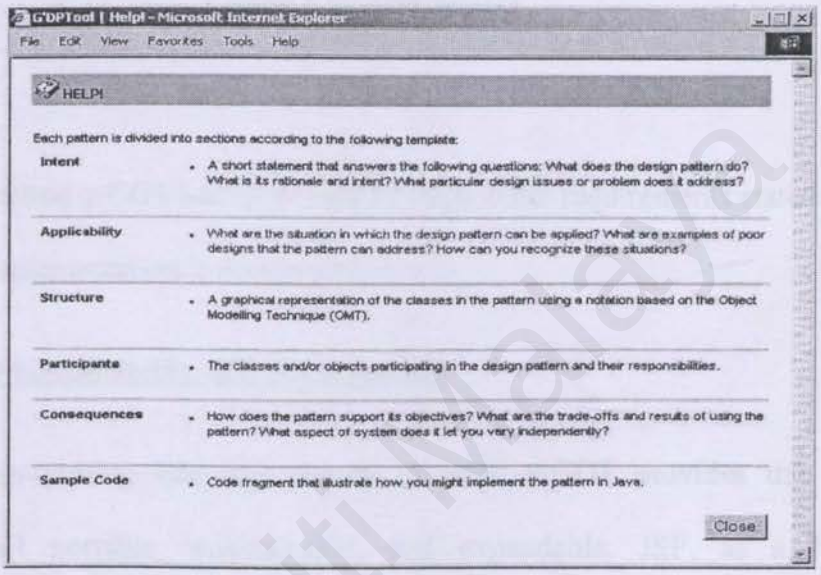

Figure 5.11: Help facility explain the pattern template

6. Alert Message

γ-CGT provides an alert message to the designer while filling-up the pattern form. This ensures that the designer become alert to the mistakes that has been made. In the case that the designer has given an illegitimate names for the pattern participants or when the designer forget to define any names for the pattern participants alert message will be pop-up. Figure 5.12 shows an alert message when the designer has not defined any names for pattern participants.

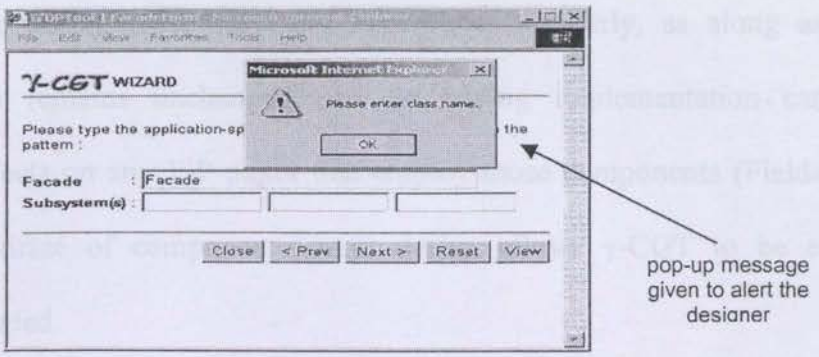

Figure 5.12: Alert message

Implementing γ-CGT has incorporated some other requirements stated in section 4.1.1.2. Their implementation is presented below:

- Portability, Maintainability, and Expendability

The programming language chosen to build γ-CGT provides the means of making the tool portable, maintainable, and expandable. JSP, as a Java-based technology, enjoys all the advantages that the Java language provides with respect to development and deployment. JSP supports the portability by not locking the deployment using specific hardware platform, operating system or server software. If a switch in any one of these components becomes necessary, all JSP pages and associated class can be migrated over as it is.

By taking advantage of JSP's built-in support for JavaBeans, it becomes possible to maintain a strict separation between presentation and the program implementation. The benefit of decoupling these two aspects is that changes in one can be made without requiring any changes in the other. For example, the way data is displayed can be

revised without ever having to modify any Java code. Similarly, as along as the component interface remains unchanged, the underlying implementation can be rewritten with no effects on any JSP pages that employ those components (Fields and Kolb, 2000). This virtue of component-centric design allows γ-CGT to be easily maintained and expanded.

5.2 γ-CGT Execution

The processes of design pattern implementation using γ-CGT begin with identifying design pattern phase, which the designer tries to find the correct design pattern. Based on the design pattern catalog, the designer is thus able to narrow down the search by specifying the design problem purpose and scope. The design problem 'Purpose' may concern creational, structural or behavioral pattern. Meanwhile the 'Scope' might be related either to object or classes. Studying the catalog directs the designer to the right design pattern or group of patterns. Next, the designer studies the pattern Intent. This step becomes easier owing to the number of patterns to be reviewed becomes less after the designer has specified the pattern group. Pattern Intent is a very important description because it describes the design issue and problem that the pattern addresses. Finally, the designer is able to determine the right design pattern for their design problems.

After the designer has chosen a design pattern, the next step is the review static structure phase. Here, the designer reviews the static structures, there are the design pattern Applicability and Consequences. Besides examining the static structures of a

design pattern, the designer need to undergo the review dynamic structure phase. The dynamic parts are the pattern Structure and Participants. Structure shows graphical representation of classes or objects in the design pattern using a notation based on Object Modelling Technique (OMT).

Once the designer has decided that the pattern is suitable for their design problem, they proceed with the following phase that is defining names for design pattern participants. The names for participants in design pattern are usually too abstract to appear directly in an application. Therefore, the designer must define names for the participants that are meaningful in the application context of design problem. Finally, the designer must submit the names of the design pattern participants to the server. In the generate code phase, the code for the design pattern is generated and displayed in the code browser. The generated code is presented in Java programming language. The designer views and able to save the codes. This codes can be use to be integrated with the designer own application code that they wish to develop. The subsequent phase is visualization of the customized pattern. The designer should click "View button" in the γ-CGT wizard to view the visualization of the design pattern implementation in class diagram. In the last phase of rework and finish, the designer should have completed the implementation of the design pattern and thus, repeat the same process for other design problems they may have. On the contrary, the designer should quit after they using the γ-CGT.

5.3 Summary

Two main things were described in this chapter: γ-CGT implementation and execution. In the implementation section, the communication infrastructure, the database option and web server was described. This is followed by the exploration of the γ-CGT main browsers. Each object was described, the implementation of its functions was presented and its required and necessary graphical representation was shown. The final part of the implementation was the implementation of the main features identified in the previous chapters. The second section of this chapter described the execution of the pattern implementation using γ-CGT.

Chapter 6

EVALUATION AND RESULTS

One way of assuring the validation of software system is to evaluate its compliance with its requirements (Boehm, 1984). This chapter presents γ-CGT evaluation process. It describes the pilot study conducted to evaluate its features and requirements, compares γ-CGT with other pattern code generator tools and presents the results obtained from within this study.

6.1 γ-CGT Evaluation

γ-CGT was evaluated by students from the Faculty of Computer Science & Information Technology, University of Malaya. The pilot study conducted is explained in section 6.1.1.

6.1.1 Pilot Study

The primary goal behind carrying this study is to assess the feasibility of executing the design pattern implementation using γ-CGT. The study is not intended to address any validation for the implemented pattern code generator tool nor its methodology. The pilot study includes a small number of postgraduate students with moderate to extensive knowledge on software engineering and reuse process. Nevertheless, all the participants have null to moderate knowledge about design patterns and their implementations. Thus, this study does not provide any scientific evidence for

its effectiveness. The study includes both quantitative and qualitative measurements. The quantitative measurements include the quantity measurement observed from this study and the qualitative measurements include the responses of γ-CGT participants. The information is gathered using a questionnaire.

6.1.1.1 Participants

The study involves five graduate student volunteers. The students are postgraduate students in Software Engineering at the Department of Software Engineering, University of Malaya. All participants have moderate to extensive knowledge on software engineering and reuse process. On the contrary, all the participants have null to moderate knowledge on design patterns and their implementations.

6.1.1.2 Experiment Material

The experimental material was adopted from (Cheng, 2002) and adapted to suit the requirements of this experiment. Although the pattern catalog consists of 23 design patterns yet only five design patterns will be involve. In which cover all the patterns in a short period of time usually blurring together or participants spending may more time on understanding the details of many different examples that on the pattern themselves (Goldfedder and Rising, 1996). Therefore, this material only seeded with four requirements with each of the requirements the participants have to specify the design pattern related.

6.1.1.3 Environment

The participants used PC with the hardware specification of Pentium II with 128 Megabyte RAM. The browser they used was either Internet Explorer ver.5 or Netscape ver. 4.7.

6.1.1.4 Methodology

Firstly, each participant was briefed on the design pattern implementation methodology and the experiment materials as in *Appendix B.1* before using γ-CGT. At the end of the design pattern application process, each participant was given a questionnaire as enclosed in *Appendix B.2*. The questionnaire is categorized into 3 parts. The first part gathered the participant knowledge about design pattern and their application. Meanwhile, the second part consists of questions on the implementation aspect and the third part includes questions about the overall usage of γ-CGT.

6.1.1.5 Quantitative Measurement

From the four requirements seeded in this experimental material, the quantitative measurement results are described. Three of the participants have performed well since they have managed to identify all the design patterns related to the requirements given. Meanwhile, two participants only managed to identify three and two design patterns respectively. Those that have identified the design patterns continued with the design patterns transformation into its concrete form and eventually view the visualization of the design patterns implementation. Although not all of the participants were able to

identify the whole design patterns for the given requirements, it is still a fine response gathered from them. This is due to the experiment is actually some of the participant first attempt to explore design patterns and their implementation.

6.1.1.6 Qualitative Measurement

Participants were asked to fill in the questionnaire at the end of the evaluation session. Table 6.1 shows the summary of the questionnaire and its results and Table 6.2 stated the questionnaire answers scores. Generally, participants scored an above average mean score for all the questions.

The first part of the questionnaire, contained questions related to the participants' knowledge about design patterns and their implementation. It seems that three of the participants have average knowledge about design patterns but only two of them have attempted using pattern code generator tool before. On the contrary, two of the participants have zero knowledge about design patterns but they have knowledge about software reuse. Hence, this evaluation assesses how γ-CGT helps beginners in gaining understanding about design patterns and their implementation.

In the second section, participants were asked to rate the effectiveness of the design pattern implementation process. The mean answer was between average and accurate. The highest mean scores are found in Q5 and Q6, which were above 3.00. This shows that the static and dynamic description of the design patterns have helped the

participants to gain understanding of the design pattern. The lowest mean score was 2.51 for the Q4. It seems that the participants were tolerable with the pattern catalog facilities.

Table 6.1: Summary of the questionnaire and its results

Q	QUESTIONS	AVE.
Section 1: Knowledge Level		
1	Have you ever encountered the terms of design patterns?	Yes/No
2	Do you have any understanding about the contribution of design patterns in software development?	Yes/No
3	Have you ever used any specific tool or software related to design patterns?	Yes/No
Section 2: Implementation Session		
4	Does the pattern catalog ease the search of pattern that is right for your design problem?	2.51
5	Does the pattern abstract description (static parts) help you to apply the design pattern?	3.02
6	Does the pattern abstract description (dynamic parts) help you to apply the design pattern?	3.10
7	Is it easy to customize the design pattern?	2.90
8	Does the pattern implementation visual help you to understand the relationship between the design pattern classes more clearly?	2.64
9	What do you think of the help facilities given?	2.84
Section 3: Overall Implementation Session		
10	How satisfied are you with γ-CGT in terms of its overall performance?	2.86
11	How satisfied are you with γ-CGT in terms of its reliability?	2.20
12	Overall, how would you rate the usability of γ-CGT?	2.65
13	Do you feel that γ-CGT improves any aspect of your performances? Is so, please state which aspect(s) and reason(s)?	Free form
14	Are there any features you think will enhance γ-CGT?	Free form
15	Do you feel that γ-CGT has in any way or other slow down your performance? If so, please state how?	Free form

Table 6.2: Questionnaire answers scores

0	1	2	3	4
Very difficult	Difficult	Average	Easy	Very easy
Very dissatisfied	Dissatisfied	Average	Satisfied	Very satisfied
Totally unhelpful	Unhelpful	Average	Helpful	Very helpful
Extremely unusable	Fairly unusable	Average	Fairly usable	Extremely usable

The last section contains questions about the overall view of γ-CGT. Q10 required the participants to rate the overall performance of γ-CGT. The mean scores record 2.86 and this shows that the participants faired between moderate to satisfied responses with γ-CGT. Participants were also asked about the reliability of γ-CGT and the response was between average and satisfied 2.20. Next, the participants were asked about the overall usability of γ-CGT. The mean score was 2.65 in which the participants showed between moderate and satisfied responses in γ-CGT usability. Finally, the questionnaire includes free-form questions. The first free-form question asked participants how γ-CGT has improved the different aspects of their performances. Some of them have stated that the implementation process of the design pattern has becomes easier and the reuse process will certainly improve. The participants were also asked about the additional features that they like to see in γ-CGT. Some of them wanted the generated code to be in other object-oriented programming such as C++. In the last question, subjects were asked if γ-CGT decrease their work performance. Some stated that the design pattern was quite new to employ into their working environment. Therefore, the process of using design patterns certainly required more time to adapt in their software development process.

6.2 Comparison of γ-CGT with Other Pattern Code Generator Tools

Another way of evaluating γ-CGT is by comparing it with the current available tools. This section compares γ-CGT with other pattern code generator tools. Table 6.3 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 are the graphical customization and trade-offs constraints. When compared to Budinsky's Tool, γ-CGT has supported two 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 customized pattern visualization, offers help facility and provides alert message. On the contrary, when 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 that, γ-CGT has more credit in terms of the distribution of the design pattern and their implementation knowledge via WWW.

Table 6.3: 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	√	√	√	√	√
Pattern Structure Diagram	√		√	√	√
Form-Based Customization		√	√	√	√
Graphical Customization	√				
Visualize Implement Pattern	√				√
Trade-offs Constraints	√				
Help Facility		√	√		√
Alert Message					√

6.3 Results

The above pilot study shows that implementing design patterns is feasible using γ-CGT tool. It also reveals that γ-CGT has achieved its objectives. This section describes how those objectives have been addressed.

1. Simplifying the process of generating source code

The process of generating code for the design patterns becomes simple via form-based customization. The participants are only required to supply legitimate names of design pattern participants before implementing the design pattern. The process is quite straightforward as the form provides default names of design pattern participants as reference.

2. Depicting the implemented design pattern

γ-CGT visualizes the design pattern implementation into class diagram and displays it in the diagram browser. Although the participants are not able to save the diagram, yet they are still able to view the class diagram in order to understand the complexity of the relationship in classes or objects in the design pattern.

3. Providing necessary helps to developers

γ-CGT provides help to the participants while they are using the tool. One of the help facilities given is the Windows help, which describes the design pattern template.

Besides Windows help, the tool also provides other help facilities, for example the alert message, and the automatic terms definition.

4. Expressing design patterns at a higher level

γ-CGT leverages the material found in (Gamma et al., 1995) to show the static and dynamic structure of the design patterns. The materials that are displayed in the pattern browser include the descriptions about the design pattern Intent, Applicability, Structure, Participants, and Consequences.

6.4 Summary

This chapter has evaluated γ-CGT against its requirements. It has described the pilot study conducted for the evaluation process. This pilot study resulted in both quantitative and qualitative measurements. These measures have been used to describe the overall satisfaction of γ-CGT. γ-CGT has also been compared with other existing patterns code generator tools. Finally, the evaluation results have been presented.

Chapter 7

CONCLUSION

7.1 Summary

Patterns provide proven solutions that solve a general design problem in a particular context that are certainly useful for designers to solve new designs problem. This is because recurring design problems always occurs in object-oriented software design or known as design déjá-vu. They are independent of a particular application domain and programming paradigm, thus enabling widespread reuse even when other forms of reuse components are infeasible. This definitely makes object-oriented designs more flexible, elegant, and ultimately reusable.

One of the main obstacles facing design patterns is that it does not lead to direct code reuse. Some developers have found it difficult to make the leap from the pattern description to a particular implementation, even though the pattern includes code fragments in the Sample Code section. Other more experience developers may have no trouble translating the pattern into code, but they still find it a chore, especially when they have to do it repeatedly. A design change might require substantial re-implementation, because different design choices in the pattern can lead to vastly different c odes. These problems and others were identified in the first chapter of this thesis. The main objectives have been put forward to tackle these difficulties.

There exists number of pattern code generator tools. Those have been surveyed in chapter two. This study has resulted in capturing the main features of a pattern code generator tool. On the light of the study conducted, the framework for this research was specified.

Based on the introduced tool, γ-CGT functionality has been drawn. An extensive analysis and design have been carried out to shape its functionality. Many techniques have been used. These techniques have been discussed in chapter three. The following chapter has shown how the design elements identified were implemented. It has also described the ways to execute the implementation of the design patterns using γ-CGT.

One way to validate a software system is to extensively run it. γ-CGT has been validated by a group of graduate students. The pilot study resulted in some qualitative and quantitative measurements. Although, these measures have revealed some limitations in γ-CGT, it proven that γ-CGT is usable and it has achieved its objectives.

7.2 Contributions

Two main contributions have been achieved:

- The first contribution is a methodology for the pattern code generator tool, which has been presented in 3.1.1. This methodology ensures that the design pattern transformation preserves the tool behaviour.

- The second contribution is a prototype of a web-based pattern code generator tool called γ-CGT has been designed and implemented. This tool applies the design patterns (Gamma et al., 1995) transformation into Java code. γ-CGT has been found to be comparable and competitive to other pattern code generator tools.

7.3 Future Work

The study can be extended in several ways:

Although the pilot study provided initial evidence about the effectiveness of the presented methodology, it was not enough to provide any explicit scientific evidence. The experimental materials were not that difficult because they were chosen to meet the average knowledge of the participants. For such reason, further research is needed to explore the effectiveness of the presented methodology.

γ-CGT has incorporated most of the main features that constitute a good pattern code generator tool. Some other features that increase the effectiveness of such tool should be integrated. γ-CGT can be improved to:

- Generate code for the pattern application into other programming language such as C++ and Smalltalk. This is beneficial to help designers from different programming backgrounds to transform the design pattern into its concrete form that they might prefer to use.

- Introduce some metrics into γ-CGT to measure the intended benefits of the decisions to apply specific design pattern based on intuition that are often made by designers. This metrics help to verify whether the designer intuition is right or not. Besides being able to measure the benefits of a specific pattern, metrics may measure the quality of the software that has been developed using design patterns.

Finally, design patterns as one of the most the successful reuse components, deserve further study to explore its full potential.

APPENDIX A: γ-CGT Object-Oriented Analysis and Design

These diagrams range from the use-cases in the analysis phase to the interaction diagrams in the design phase. This appendix presents these diagrams.

A.1 Use-Cases Diagram

These use-cases are depicted in Figure A.1.1. The scenarios for these use-cases are shown in section A.2 where the interaction diagrams are drawn based on them.

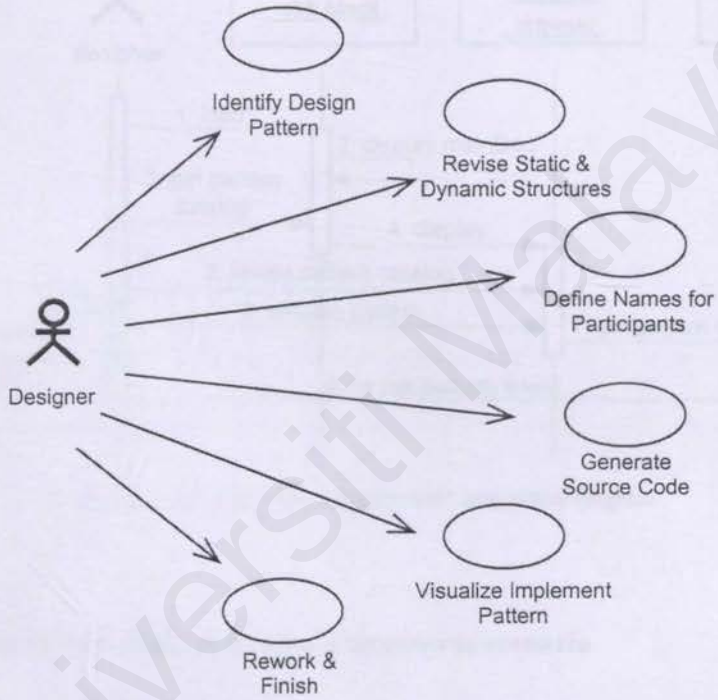

Figure A.1.1: γ-CGT use-cases

A.2 Scenarios and Interaction Diagrams

The above identified use-cases have their respective scenarios and interaction diagrams. These scenarios and diagrams are shown in this section.

A.2.1 A designer identify design pattern scenario

1- The designer loads web client.
2- The client displays mainDoc.
3- The designer retrieves design pattern catalog.
4- The design pattern catalog is displayed in catalogBrowser.
5- The designer revises the design pattern catalog.
6- The designer chooses a design pattern.
7- The catalogBrowser retrieves the design pattern details.
8- The design pattern abstract description is displayed in patternBrowser.
9- The designer revises the design pattern Intent.

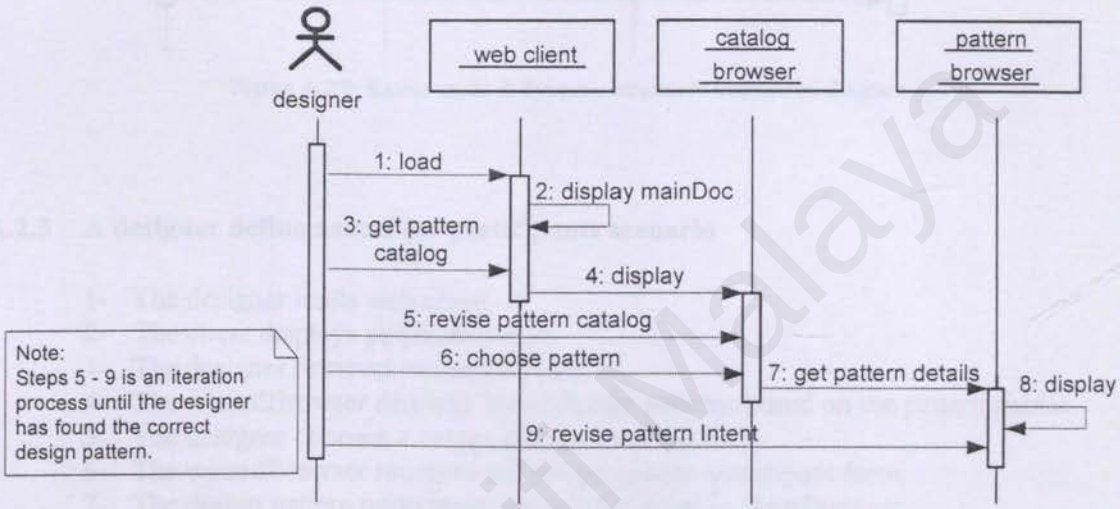

Figure A.2.1: Identify a design pattern interaction diagram

A.2.2 A designer revises static & dynamic structures scenario

1- The designer loads web client.
2- The client displays mainDoc.
3- The designer retrieves design pattern catalog.
4- The design pattern catalog is displayed in catalogBrowser.
5- The designer chooses a design pattern.
6- The catalogBrowser retrieves the design pattern details.
7- The design pattern abstract description is displayed in patternBrowser.
8- The designer revises the design pattern static structures.
9- The designer revises the design pattern dynamic structures.

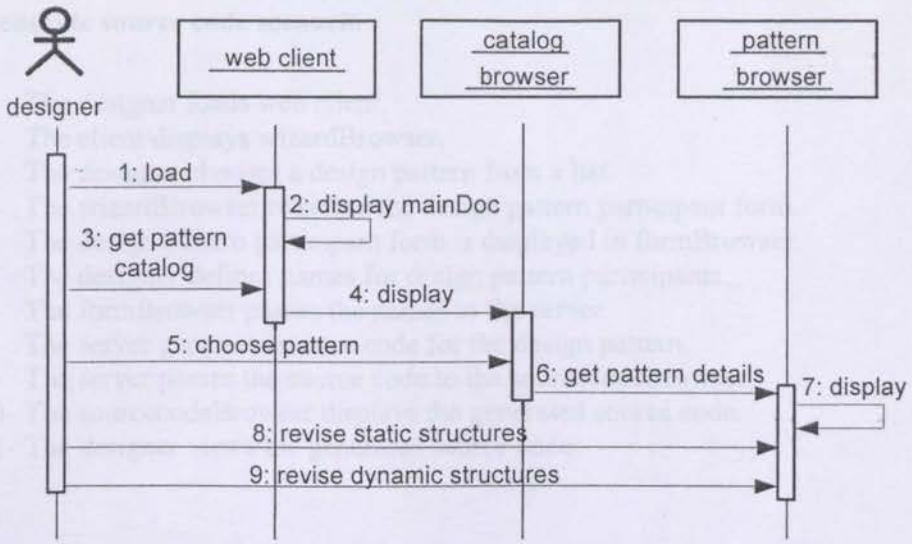

Figure A.2.2: Revise static & dynamic structures interaction diagram

A.2.3 A designer define names for participants scenario

1- The designer loads web client.
2- The client displays patternBrowser.
3- The designer retrieves wizardBrowser.
4- The wizardBrowser displays list of design patterns based on the pattern names.
5- The designer chooses a design pattern from the list.
6- The wizardBrowser retrieves the design pattern participant form.
7- The design pattern participant form is displayed in formBrowser.
8- The designer defines names for design pattern participants.

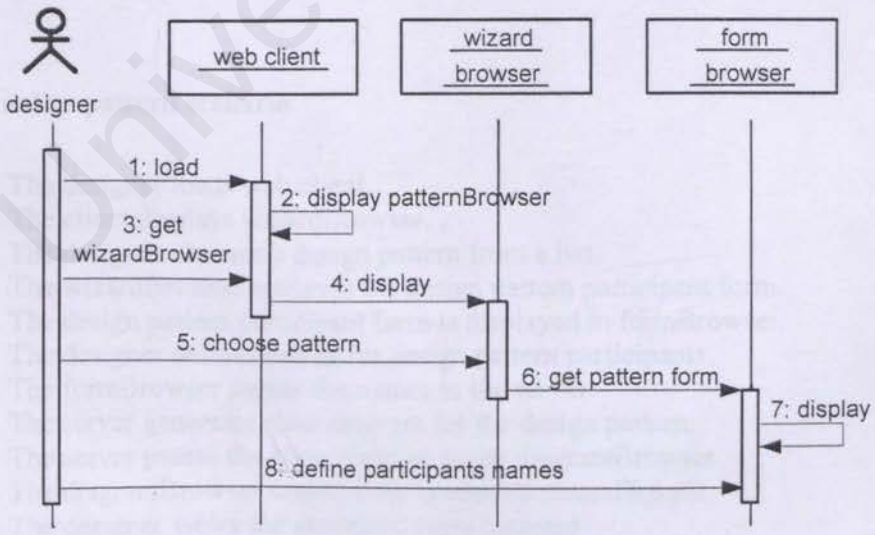

Figure A.2.3: Define names for participants interaction diagram

A.2.4 Generate source code scenario

1- The designer loads web client.
2- The client displays wizardBrowser.
3- The designer chooses a design pattern from a list.
4- The wizardBrowser retrieves the design pattern participant form.
5- The design pattern participant form is displayed in formBrowser.
6- The designer defines names for design pattern participants.
7- The formBrowser passes the names to the server.
8- The server generates source code for the design pattern.
9- The server passes the source code to the sourcecodeBrowser.
10- The sourcecodeBrowser displays the generated source code.
11- The designer views the generated source code.

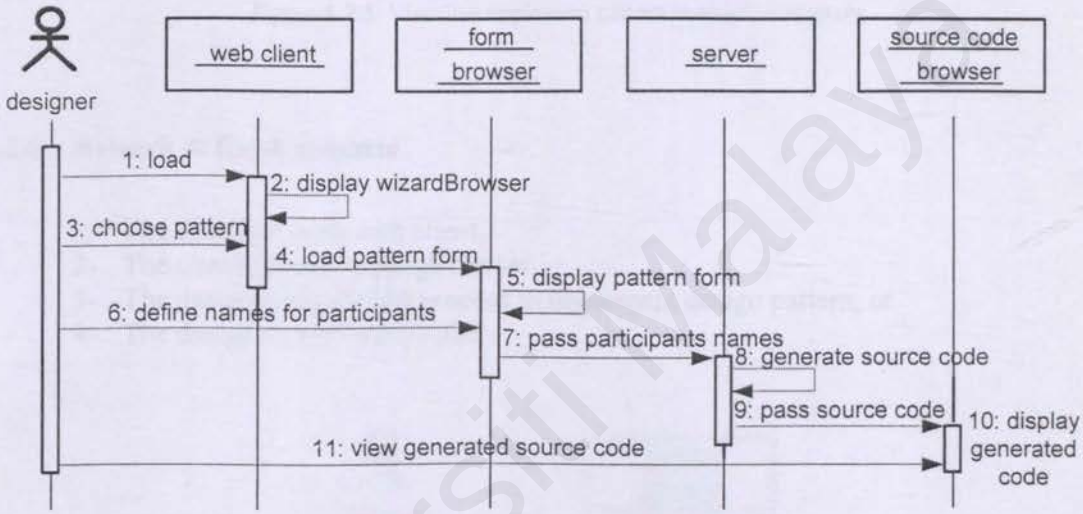

Figure A.2.4: Generate source code interaction diagram

A.2.5 Visualize pattern scenario

1- The designer loads web client.
2- The client displays wizardBrowser.
3- The designer chooses a design pattern from a list.
4- The wizardBrowser retrieves the design pattern participant form.
5- The design pattern participant form is displayed in formBrowser.
6- The designer defines names for design pattern participants.
7- The formBrowser passes the names to the server.
8- The server generates class diagram for the design pattern.
9- The server passes the class diagram to the diagramBrowser.
10- The diagramBrowser displays the generated class diagram.
11- The designer views the generated class diagram.

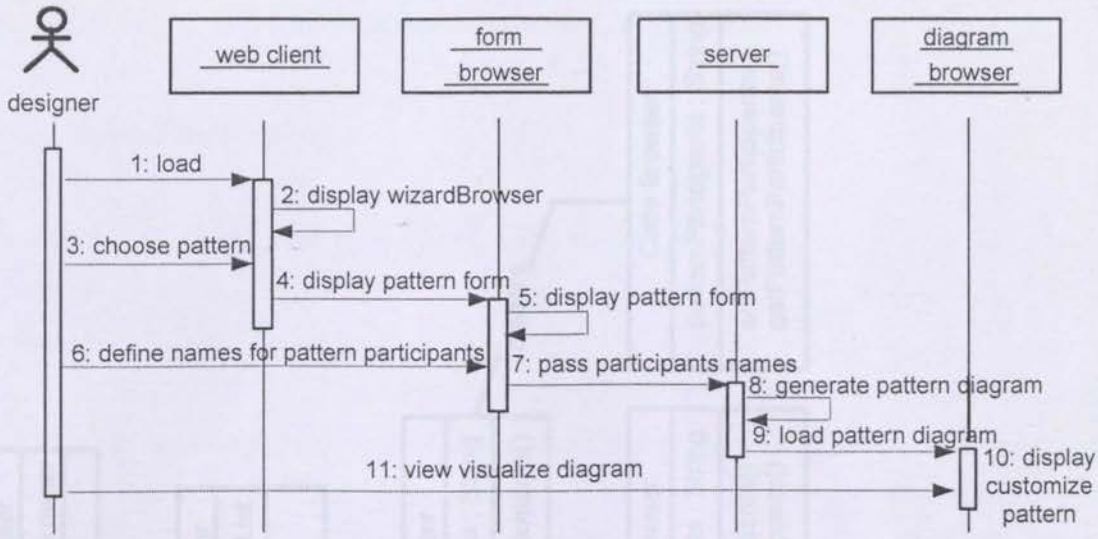

Figure A.2.5: Visualize implement pattern interaction diagram

A.2.6 Rework & finish scenario

1- The designer loads web client.
2- The client display catalogBrowser.
3- The designer repeats the process to implement design pattern, or
4- The designer exit from γ-CGT.

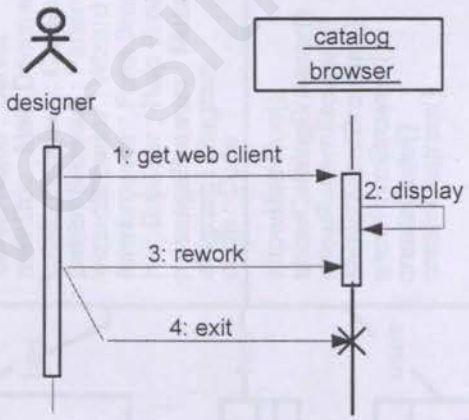

Figure A.2.6: Rework & finish interaction diagram

A.3 γ-CGT Detailed Classes

This section shows the detailed classes with their attributes and methods. The detailed classes diagram depicted in figure A.3.1.

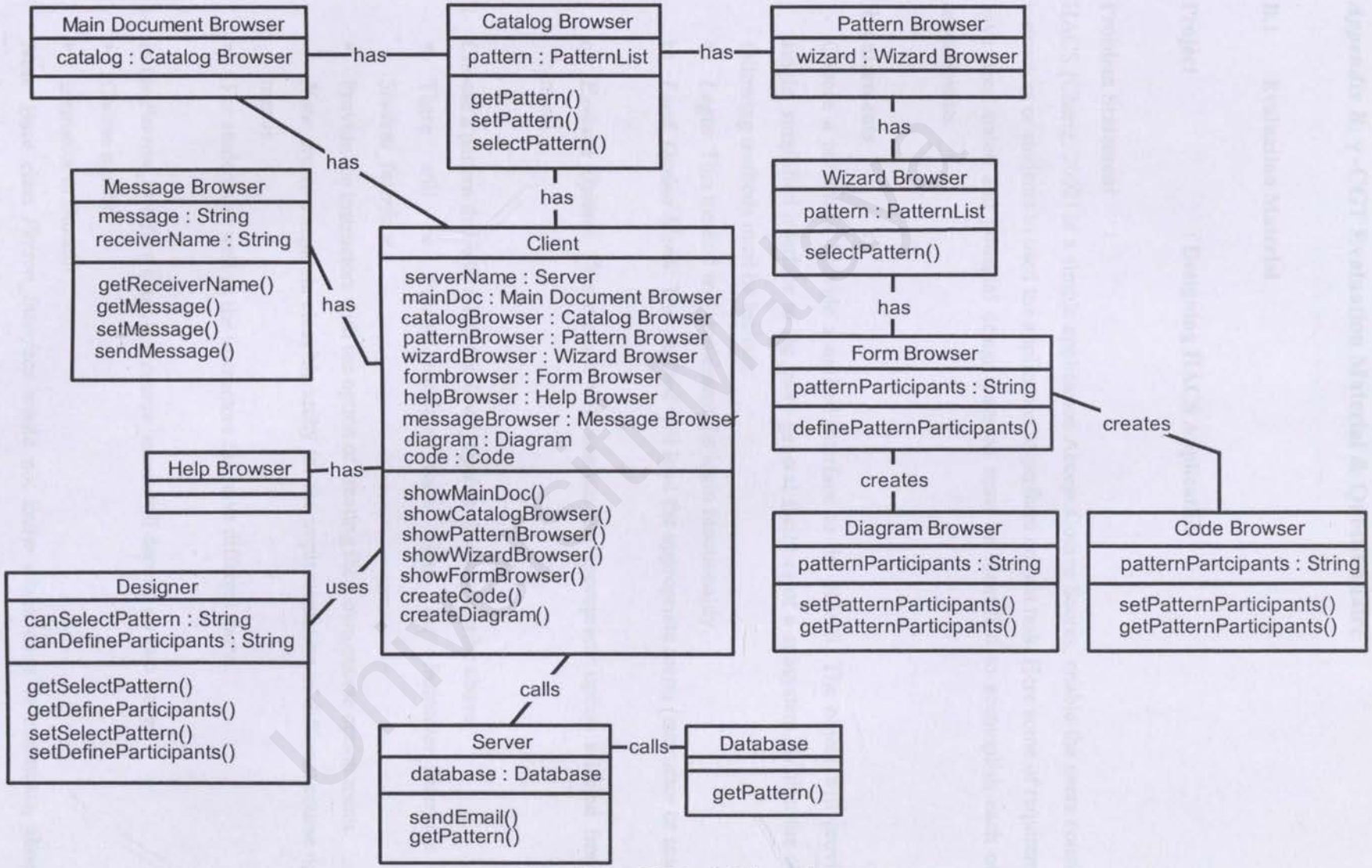

Figure A.3.1: γ-CGT detailed classes diagram

Appendix B: γ -CGT Evaluation Material & Questionnaire

B.1 Evaluation Material

Project : Designing HACS Application

Problem Statement :

HACS [Cheng, 2002] is a simple application Access Course Scores, enable the users consist of instructors or students to used the application to perform certain task. Here some of requirements have been stated and potential design patterns must be identified to accomplish each of the requirements.

Requirements :

1- Choose a pattern to provide a unified interface to the HACS. The object will provide a single, simplified interface to the more general facilities of a subsystem. Within the object following methods must be include:

a) ***Login***: This method will implement the login functionality.

b) ***Load_Option Menu***: This method will load the appropriate menu (instructor or student)

c) ***Execute Option***: This method will execute the appropriate option selected from the menu.

2- Choose a pattern that will affect the ***Load_Option Menu*** describe above:

- There will be *Person_Interface* base class for *Instractor_Interface and Student_Interface.*
- Provide the instructors with the option of creating their own course option menu.
  Note: create a separate class hierarchy for the implementation of different course option menus.
- The students as well as the instructors can have different menu.

3- In the *Person_Interface* class, the *course_menu* will depend on two things.

- Course number
- Instructor or Student

Note: Base class *Person_Interface* would not know which class to instantiate, choose a pattern that enable the subclasses decide which class to instantiate.

4- Implement a pattern as means for printing the grade report for the students.

Design Suggestion :

Try to identify the design pattern for each of the stated requirements.

Req. 1: Design Pattern Purpose : _____

Design Pattern Scope : _____

Design Pattern : _____

Class Participants : _____

Req. 2: Design Pattern Purpose : _____

Design Pattern Scope : _____

Design Pattern : _____

Class Participants : _____

Req. 3: Design Pattern Purpose : _____

Design Pattern Scope : _____

Design Pattern : _____

Class Participants : _____

Req. 4: Design Pattern Purpose : _____

Design Pattern Scope : _____

Design Pattern : _____

Class Participants : _____

QUESTIONNAIRE

A Code Generator Tool for the Gamma Design Patterns

Prepared by

Novia Indriaty Admodisastro

WGC 00005

Master of Software Engineering

Supervisor

Assoc. Prof. Dr. P. Sellapan

Faculty of Computer Science & Information Technology

University of Malaya

Questionnaire Objectives:

The objective of this questionnaire is to evaluate the implemented prototype of design pattern code generator tool known as γ-CGT. The tool was evaluated against the research objectives, and against its features that have been identified from the existing similar tool.

Instruction: Please mark ☒ appropriate box.

Section 1: Knowledge Level

1. Have you ever encountered the terms of design patterns?
   - ☐ Yes, give the definition of design pattern: _____
   - ☐ No

2. Do you have any understanding about the contribution of design patterns in software development?
   - ☐ Yes, explain _____
   - ☐ No

3. Have you ever used any specific tool or software related to design patterns?
   - ☐ Yes, give name of the tool/software: _____
   - ☐ No

Section 2: Implementation Level

4. Does the pattern catalog ease the search of pattern that's right for your design problem?
   - ☐ Totally unhelpful
   - ☐ Unhelpful
   - ☐ Average
   - ☐ Helpful
   - ☐ Very helpful

5. Does the pattern abstract description (static part) help you to apply the design pattern?
   - ☐ Totally unhelpful
   - ☐ Unhelpful
   - ☐ Average
   - ☐ Helpful
   - ☐ Very helpful

6. Does the pattern abstract description (dynamic part) help you to apply the design pattern?
   - ☐ Totally unhelpful
   - ☐ Unhelpful
   - ☐ Average
   - ☐ Helpful
   - ☐ Very helpful

7. Is it easy to customize the design pattern?
   - ☐ Very difficult
   - ☐ Difficult
   - ☐ Average
   - ☐ Easy
   - ☐ Very easy

8. Does the pattern implementation visual help you to understand the relationship between the design pattern classes more clearly?
   - ☐ Totally unhelpful
   - ☐ Unhelpful
   - ☐ Average
   - ☐ Helpful
   - ☐ Very helpful

9. What do you think of the help facilities given?
   - ☐ Totally unhelpful
   - ☐ Unhelpful
   - ☐ Average
   - ☐ Helpful
   - ☐ Very helpful

Section 3: Overall Level

10. How satisfied are you with γ-CGT in terms of its overall performance?
    - ☐ Completely dissatisfied
    - ☐ Dissatisfied
    - ☐ Average
    - ☐ Satisfied
    - ☐ Completely satisfied

11. How satisfied are you with γ-CGT in terms of its reliability?
    - ☐ Completely dissatisfied
    - ☐ Dissatisfied
    - ☐ Average
    - ☐ Satisfied
    - ☐ Completely satisfied

12. Overall, how would you rate the usability of γ-CGT?
    - ☐ Extremely unusable
    - ☐ Fairly unusable
    - ☐ Average
    - ☐ Fairly usable
    - ☐ Extremely usable

13. Do you feel that γ-CGT improve any aspect of your performances? Is so, please state which aspect(s) and reason(s)?

    _____

    _____

14. Are there any features you think will enhance γ-CGT?

    _____

    _____

15. Do you feel that γ-CGT has in any way or other slow down your performance? If so, please state how.

    _____

    _____

Note:

0: Very difficult	1: Difficult	2: Average	3: Easy	4: Very easy
0: Very dissatisfied	1: Dissatisfied	2: Average	3: Satisfied	4: Very satisfied
0: Totally unhelpful	1: Unhelpful	2: Average	3: Helpful	4: Very helpful
0: Extremely unusable	1: Fairly unusable	2: Average	3: Fairly usable	4:Extremely usable

- Finish -

Thank you.

Appendix C: The Catalog of Design Pattern

The catalog consists of 23 design patterns. The design patterns names and intents are listed below for an overview.

Abstract Factory: Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

Adapter: Convert the interface of a class into another interface client expects. Adapter lets classes work together that couldn't otherwise because of incompatible interface.

Bridge: Decouple an abstraction from its implementation so that the two can vary independently.

Builder: separate the construction of a complex object from its representation so that the same construction process can create different representations.

Chain of Responsibility: Avoid coupling the sender of a request to its receiver by giving more that one objects a chance to handle the request. Chain the receiving objects and pass the request along that chain until an object handles it.

Command: Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log request and support undoable operations.

Composite: compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

Decorator: Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to sub-classing for extending functionality.

Façade: Provide a unified interface to a set of interfaces in a subsystem. Façade defines a higher-level interface that makes the subsystem easier to use.

Factory Method: define an interface for crating an object, but let subclasses decide which class to instantiate. Factory method lets a class defer instantiation to subclasses.

Flyweight: Use sharing to support large numbers of fine-grained objects efficiently.

Interpreter: Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.

Iterator: provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

Mediator: define an object that encapsulates how a set of objects interacts. Mediator promotes loose coupling by keeping objects from referring to each other explicitly and it lets you vary their interaction independently.

Memento: without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.

Observer: define a one to many dependencies between objects so that when one object changes state, all its dependents are notified and updated automatically.

Prototype: specify the kinds of objects to create using a prototypical instance and create new objects by copying this prototype.

Proxy: Provide a surrogate or placeholder for another object to control access to it.

Singleton: Ensure a class only has one instance and provide a global point of access to it.

State: Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.

Strategy: define a family of algorithm, encapsulate each one and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

Template Method: define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.

Visitor: represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.

Bibliography

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King I. and Angel, S. (1977). *A Pattern Language: Towns, Building and Construction*. New York: Oxford University Press.

Alexander, C. (1979). *The Timeless of Building*. New York: Oxford University Press.

Beck, K. and Johnson, R. E. (1994). Patterns Generate Architectures. In *Proceedings of European Conference on Object-Oriented Programming (ECOOP '94)*, p. 139-149. Bologna, Italy.

Beck, K., Coplien, J. O., Crocker, R., Dominick, L., Meszaros, G., Paulisch, F. and Vlissides, J. (1996). Industrial Experience with Design Patterns. In *Proceedings of 18th International Conference on Software Engineering (ICSE '96)*, p. 103-114. Berlin, Germany.

Berners-Lee, T. (1996). WWW: Past, Present and Future. *IEEE Computer*, p. 69-77.

Blaine, G., Boyd, M. and Crider, S. (1994). Project Spectrum: Scalable Bandwidth for the BJC Health System. *HIMSS, Health Care Communications*, p.71-81.

Boehm, B. (1984). Verifying and Validating Software Requirements and Design Specifications. *IEEE Software*, p.75-88.

Borland (2001). *AppServer Documentation*. Available from Borland Software Corporation Homepage. URL: http://www.borland.com.

Brad, A. (2000). *Patterns and Software: Essential Concepts and Terminology*. Available from Brad Appleton Homepage. URL: http://www.enteract.com/~bradapp/.

Brown, K. (1996). *Design Reverse-Engineering and Automated Design Pattern Detection in Smalltalk*. Master's Thesis. University of Illinois.

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

Buschmann, F., Meunier, R., Rohnert, H., Sommerland, P. and Stal, M. (1996). *Pattern-Oriented Software Architecture- A System of Pattern.* New York: John Wiley & Sons.

Cheng, B. (2002). *Advance Software Engineering Notes*. Available from Michigan State University Homepage. URL: http://www.cse.msu.edu/~cse870/.

Cline, M.P. (1996). The Pros & Cons of Adopting and Applying Design Patterns in the Real World. *Communications of the ACM*, 39(10): 47-49.

Coad, P. (1992). Object-Oriented Patterns. *Communications of the ACM*, 35(9): 152-159.

Coplien, J.O. (1992). *Advanced C++ Programming Styles and Idioms.* Reading, MA: Addison-Wesley.

Coplien, J.O. (1997). Idioms and Patterns as Architectural Literature. *IEEE Software: Special Issue on Objects, Patterns and Architectures*.

Fayad, M., Tsai, W. and Fulghum, M. (1996). Transition to Object-Oriented Software Development. *Communications of the ACM,* 39(2): 108-121.

Fields, D.K. and Kolb, M.A (2000). *Web Development with Java Server Pages*. Greenwich, CT: Manning Publication.

Florijn, G., Meijers, M. and Winsen, P.v (1997). Tool Support for Object-Oriented Patterns. In *Proceedings of European Conference on Object-Oriented Programming (ECOOP'97).* Finland.

Fowler, M. (1996). *Analysis Patterns: Reusable Object Models.* Reading, MA: Addison-Wesley.

Gamma, E. (1991). *Object-Oriented Software Development based on ET++: Design Patterns, Class Library, Tools*. Ph.D. Thesis. University of Zurich.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley.

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

Hedin, G., Ive, A., Mughal, K., Normark, K., Ron, H. and Osterbye, K. (1998). Tools for Design Patterns. *NWPER'98 Subworkshop on Tools for Software Architecture.*

Helm, R. (1995). Patterns in Practice. In *Proceedings of 10th Object-Oriented Programming, Systems, Languages and Applications (OOPSLA '95)*. Austin, Texas.

Huang, J. Q. (1996). *Formal Specification and Tool Support for OO Design Patterns*. Master's Thesis. University of Waterloo.

Jacobson, I., Griss, M. and Jonsson, P. (1997). *Software Reuse: Architecture Process and Organization for Business Success*. Reading, MA: Addison-Wesley.

Johnson, R. E. (1992). Documenting Frameworks Using Patterns. In *Proceedings of Object-Oriented Programming, Systems, Languages and Applications (OOPSLA '91)*. Vancouver BC, Canada.

Keller, R.K. and Lajoie, R. (1994). Design and Reuse in Object-oriented Frameworks: Patterns, Contracts and Motifs in Concert. In *Proceedings of 62nd Congress of the Association Canadienne Française pour l'Avancement des Sciences*. Montreal, Canada.

Levine, D.L. and Schmidt, D.C. (1999). *Introduction to Patterns and Frameworks*. Notes for Object-Oriented for Software Development Lab. Washington University.

McIlroy, D. (1969). Mass Produced Software Components. *NATO Conference on Software Engineering*, p.138-155.

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

Nakhimovsky, A. and Myers, T. (1999). *Professional Java XML Programming with Servlets and JSP*. Chicago, Illinois: Wrox Press.

Pree, W. (1994). *Design Patterns for Object-Oriented Software Development*. Reading, MA: Addison-Wesley.

Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland and S., Carey, T. (1996). *Human-Computer Interaction*. Wokingham: Addison-Wesley.

Quatrani, T. (2000). *Visual Modeling with Rational Rose 2000 and UML*. New York: Addison-Wesley.

Schmid, H. A. (1995). Creating the Architecture of a Manufacturing Framework by Design Patterns. In *Proceedings of 10th Object-Oriented Programming, Systems, Languages and Applications (OOPSLA '95)*. Austin, Texas.

Schmidt, D.C. and Stephenson, P. (1995). Experiences Using Design Patterns to Evolve System Software Across Diverse OS Platform. In *Proceedings of the 9th European Conference Objet-Oriented Programming (ECOOP'95)*. Denmark.

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

Schmidt, D.C. (1995b). A System of Reusable Design Patterns for Communication Software. In *The Theory and Practice of Object System, Special Issue on Patterns and Pattern Language*, S.P. Berczuk edn. New York: John Wiley & Sons.

Schmidt, D.C. (1996). Using Design Patterns to Guide the Development of Reusable Object-Oriented Software. *ACM Computing Surveys (CSUR)*, 28(4es).

Sommerville, I. (2001). *Software Engineering*, 6th edn. Harlow, England: Addison-Wesley.

Sun Mircosystems (2001). *Java™ 2 Platform Enterprise Edition Documentation*. Available from Sun Mircosystems Homepage. URL: http://www.java.sun.com,.

Tao, Y. (2000). *Teaching Software Tools via Design Patterns*. In *Proceedings of the Australasian Computing Education (ACM) Conference*. Australia, Melbourne.

UML (2001). *Unified Modeling Language Documentation*. Available from Rational Software Corporation Homepage. URL: http://www.rational.com/uml.

Wild, F. (1996). Instantiating Code Patterns: Patterns Applied to Software Development, *Dr. Dobb's Journal: Patterns & Software Design*, June.