2003

Unified modeling language class diagram translator for the online design pattern library system

Kaiyan Li

UNIFIED MODELING LANGUAGE CLASS DIAGRAM TRANSLATOR FOR THE ONLINE DESIGN PATTERN LIBRARY SYSTEM

A Project Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Kaiyan Li

March 2003

UNIFIED MODELING LANGUAGE CLASS DIAGRAM TRANSLATOR FOR THE ONLINE DESIGN PATTERN LIBRARY SYSTEM

A Project

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Kaiyan Li

March 2003

Approved by:

[redacted]

Dr. Arturo Concepcion, Chair, Computer Science — 24 Jan 2003 Date

[redacted]

Dr. Richard Botting

[redacted]

Dr. Josephine Mendoza

ABSTRACT

This project, UML Class Diagram Translator (UMLCDT), is a software program designed to work in the Online Design Pattern Library (DPL) system [12], which is an implementation of a tool in Applying Designs And Patterns (ADAP) Project [3].

UMLCDT project is designed to provide a useful tool that helps designers to use design patterns for software design. This project stores all the design patterns from Gamma's book, including Creational Design Patterns, Structural Design Patterns and Behavioral Design Patterns. The pattern structures are written in XML.

UMLCDT translates from class diagrams to XML files and vice versa. The server side allows the user to download the structures in XML and browse the UML class diagrams and description of each design patterns. A Graphic User Interface (GUI) was developed to provide better visual and operational convenience for the client side. For novice designers, UMLCDT is a good tool as it helps them get familiar with design patterns and use them for designing their software applications. But its main contribution is providing an XML file, which is a translation of class diagrams. Future directions can

utilize these XML files for automated design analysis and source code generation capabilities.

ACKNOWLEDGMENTS

As I look back at the years I spent in CSUSB, I really feel thankful to all my instructors, relatives and friends. Without their support, I could never go this far.

First of all, I would like to thank Dr. Concepcion. His inputs to the project were invaluable. No matter how hard it gets, he was always willing to help find a solution, discuss new ideas and provide practical help. I really appreciate that he was there to encourage me whenever I was down, which made me get through the difficulties and reach where I am today.

Secondly, I want to thank my other two Committee members, Dr. Botting and Dr. Mendoza. Dr. Botting's creative ideas lead me to a new area of study. Dr. Mendoza was there to help me get familiar with all the processes involved in the graduating from here.

I would like to thank my parents, my sister, and my other relatives. During these years of study, my parents supported me very much both financially and emotionally. My sister was there to help me get through many difficulties in my life.

Moreover, I would like to thank my friends from school, church and other places who encouraged me, loved me and never lost hope in me. Without all this love, encouragement and help, this project would never have been done.

The support of the National Science Foundation under the award 9810708 and the CSUSB Associated Student Research grant are gratefully acknowledged.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

1.1 Background

The UML Class Diagram Translator (UMLCDT) is a project under the Applying Designs and Patterns (ADAP) Project, which is a model for applying design patterns. Yun Ji in her Master's thesis initially developed the ADAP Project in 2000 [3]. The model incorporates three more layers of abstraction in order to ease the transition from abstract layer to implementation layer as well as to provide additional guidance to software developers in the use of design patterns. According to Yun Ji, ADAP is not only a model, but also forms a basis for automating design pattern applications. ADAP model defines five levels when applying design patterns: generic/domain-specific, concrete, specific, integrated, and implemented design patterns (see Figure 1).

The generic design patterns are descriptions of solutions in natural languages. Domain specific design patterns are core solutions to problems in a specific domain. The concrete design pattern, which is the first step in applying design patterns, eliminates the

ambiguities in the generic/domain-specific level. The specific design patterns apply the concrete pattern in the specific applications. The integrated design pattern combines two or more specific design patterns in the design. Finally, the system is implemented based on the integrated design pattern.

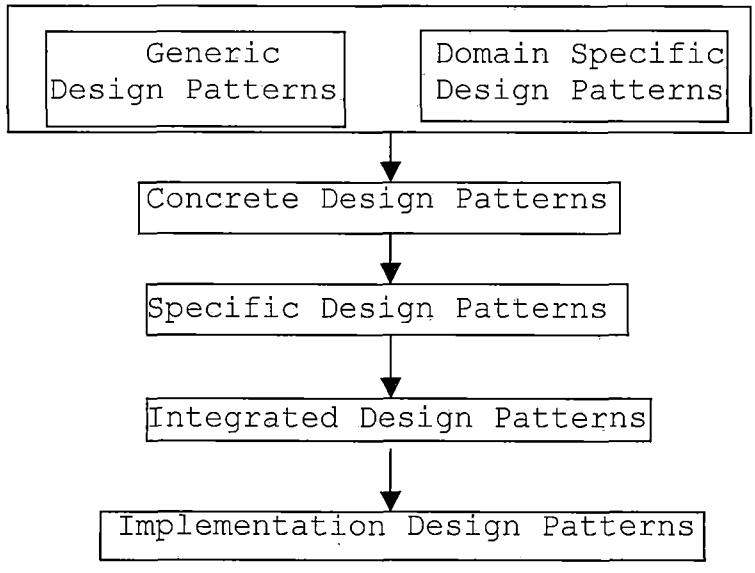

Figure 1. Applying Design and Pattern Model

Based on Ji's ADAP thesis, Weichung Wu, one of the ADAP team members, implemented the Online Design Patterns Library (DPL) [12]. The Online DPL system implements the concrete design pattern layer. Some of the most useful design patterns of Gamma et al [5] are posted on the Online DPL in order to assist a novice software developer to use the library of design patterns. Online DPL allows

a user to retrieve and modify a limited number of design patterns in order to make them more applicable to the given applications. These design patterns are grouped into four major groups: Creational Design Pattern, Structural Design Pattern and Behavioral Design Pattern.

1.2 Purpose of the Project

UMLCDT project adds new features to Online DPL. It reads and writes UML class diagrams into XML files and vice versa.

As we all know, there are many UML drawing tools available, but only a few of them allow a user to generate a UML class diagram directly from a text input file instead of spending lots of time drawing each class. The UMLCDT Project builds such a software tool that not only enables user to achieve the above function, but also allows the user to modify the diagram and write back the changes into a new text file, which is in XML.

The model of text file chosen for this project is written in XML. XML stands for Extensible Markup Language. XML itself has no predefined element types or meaning. Each XML document type defines its own structures, element types, and meanings. The reasons of

choosing XML to be the input and output files are listed as follow:

- ✓ XML schemas can be the starting point of a design and the UML class diagrams can be derived from the XML schemas;
- ✓ UML class structure can be written in certain style so that it is straightforward to generate the XML schema;
- ✓ Both XML schemas and UML class diagrams are defining sets of objects;
- ✓ XML schema is useful to separate the structure of the design from the implementation.

Table 1. Comparison of XML and UML

XML schema	UML class diagram
Processing of Objects: Programs (Java and etc.)	Processing of Objects: Programs (Java and etc.)
Defines: Languages, Structure of object	Defines: Structure of objects
Summarized program: All classes, interfaces	

Class diagram is selected to be the UML expression for specific design patterns based on the concrete design patterns in the project. The reasons are listed as follows:

The Unified Modeling Language (UML) is the successor to the wave of object-oriented analysis and design (OOA&D) methods that appeared in the late '80s and early '90s. The class diagram is one of the most useful diagrams in UML. The class diagram technique has become truly central within object-oriented methods. It describes the types of objects in the system and the various kinds of static relationships that exist among them. Also, class diagrams show the attributes and operations of a class and the constraints that apply to the way objects are connected. All these features make it one of the best ways to express the object-oriented programming structure.

It is also necessary to mention the usefulness and importance of design patterns. A design pattern describes a problem that occurs over and over again in the development of an application. It describes the core of the solution to that problem, in such a way that you can use this solution several times over, without ever doing it the same way twice. In software engineering, it simply means software reusable solutions, that is, the design patterns are descriptions of communicating objects and classes that are customized to solve a general design

problem in a particular context. For a complete discussion of the most useful kinds of design patterns, see Gamma et al "Design Patterns". [5]

In the UMLCDT Project, some of the most useful patterns are chosen from Creational, Structural and Behavioral patterns to be the base patterns. The user can implement and create more features in them since those patterns only offer basic ideas and structures.

1.3 Limitation of the Project

The limitations of UMLCDT Project are listed as follow:

- UMLCDT currently handles only limited functions in class diagram;
- UMLCDT does not allow user to make modifications on the server layer;
- UMLCDT handles single direction relationship between two class object only;
- UMLCDT does not handle all types of structures and complicate relationships among class objects;
- UMLCDT uses the Algorithm of Automatic Layout of Object-Oriented Software Diagram to generate the

class diagram, so the objects position cannot be changed manually;

- There is so far no perfect solution for this algorithm.

However, these are only temporary limitations which can be taken care of over time. This project can be taken over by other students who wish to improve and expand the original.

1.4 Organization of the Document

The chapters of this document will be organized as follows: Chapter One documents the goal of this project and some important techniques used in the project. Chapter Two presents the Software Requirements Specification (SRS), which includes the overview of software functions, hardware specifications and software specifications for UMLCDT Project. Chapter Three contains the design of this project, where the detailed design of every component of the system is discussed. Chapter Four is testing, which shows how the system has been tested and how the performance has been measured. Chapter Five is the maintenance manual. It describes how to maintain the system and how to develop the UML class diagrams

using UML translator. Chapter Six is the last chapter of this project report. It gives the conclusion as well as suggestions for future development.

CHAPTER TWO

SOFTWARE REQUIREMENTS SPECIFICATION

2.1 Introduction

This Software Requirement Specification (SRS) is for the UML Class Diagram Translator (UMLCDT) Project, which is an extension of Yun Ji's master thesis ADAP [3]. While the product primarily targets a specialized technical audience, this specification is presented in a form suitable for technical and semi-technical personnel.

This SRS provides the following functionalities:

- Provides a basis for requirement and resource estimation
- Provides a baseline for validation and verification

2.1.1 Scope

ADAP project is an on-going design pattern project in the Department of Computer Science, California State University San Bernardino (CSUSB). Ms. Yun Ji first proposed it as a Model of Applying Design Patterns in her Master's thesis in 1998 [3]. Following that, Mr. Weichung

Wu for his Master's Project, Online DPL, added some features to it [12].

The UMLCDT Project is an implementation of Weichung Wu's Online DPL [12], which belongs to the concrete design pattern level. The differences between Online DPL and UMLCDT are listed as follow:

Table 2. Online Library and UMLCDT

Online DPL	UMLCDT
UML class diagram is saved as a JPEG file	UML class diagram is saved as a XML file
Used DisCo language	Uses XML
Diagram cannot be modified once saved	Diagram can be modified after been saved
Seven of Gamma's design patterns structures are posted online	All of Gamma's design pattern structures are posted online
Combination of design patterns cannot be done	Able to combine design patterns
Only for concrete design pattern layer	Provide a bridge for the rest of the layers under concrete design pattern layer

The purpose of UMLCDT is to provide a UML Drawing tool that translates XML files into UML class diagrams. It enables the user to obtain the UML class diagrams directly from an input text file and write back to the same kind of file after modifications.

The following is a list of descriptions that UMLCDT can and cannot do. The UMLCDT is able to:

- ✓ Run on top of both Windows and Unix operating systems;
- ✓ allow users to retrieve a list of XML files generated from some of the most useful design patterns;
- ✓ Parse the data from XML files to UML class diagram translator by using an XML parser written in Java;
- ✓ Allow users to generate class diagrams from the input XML file;
- ✓ Allow users to make modifications, write or save back files, and exit the program;
- ✓ Allow users to browse through the class diagrams generated by UMLCDT and download the XML files for the design patterns from the server;

The UMLCDT is not able to:

Allow users to make modifications on the server level;

Handle complicated relationships;

2.1.2 Definitions, Acronyms, and Abbreviations

The acronyms and abbreviations used in the project are listed as follow:

Design Patterns - Abstracted level software development model. Design patterns do not give solutions to a specific application environment, but provide detail descriptions of where, when, why and how the pattern should be used.

Unified Modeling Language (UML) - Used in software designs, particularly in the object-oriented paradigm.

Class Diagram - A diagram that describes the types of objects in the system and various kinds of static relationships that exist among them.

Extensible Markup Language (XML) - used to create custom information structures for any domain or application.

Hypertext Markup Language (HTML) - used to provide information at an abstract level without providing specific details.

Document Type Definition (DTD) - used to declare all the element types and attributes and their constraints.

Java - An object-oriented language developed by Sun Microsystems. Java programs are capable of running on

most popular computer platforms without the need of recompilation.

Graphical User Interface (GUI) - Uses buttons, text components, combo box and menus to design a user-friendly environment.

2.1.3 Overview

Section 2 and the rest of this chapter follow the outline for a typical SRS, according to the IEEE Recommended Practice of Software Requirements Specifications. This section provides the product perspective, summary of product functions, description of the characteristics of the expected users, list of the development constraints and a list of assumptions and dependencies involved in the project.

Section 3 of this chapter presents the external interface requirements, functional requirements, performance requirements, design constrains, software system attributes and other requirements for this project.

2.2 Overall Description

2.2.1 Product Perspective

ADAP model facilitates the development of new tools in software design [3]. By applying this design pattern, software engineers could keep up with new technologies. This advantage makes design patterns very useful in building applications relating to E-commerce or internet in areas such as network security, transaction processing, mobile agents, instant messaging, online advertisements and so on. As a result, ADAP is designed to provide a step-by-step process of applying the solutions from the abstract level to the implementation level.

Based on the above design, UMLCDT Project provides a translator that translates the XML file to a UML diagram. The process of translation is coded in Java so that this project is able to run under both Windows and Linux operating systems with Java 1.2 or greater runtime environment or version installed.

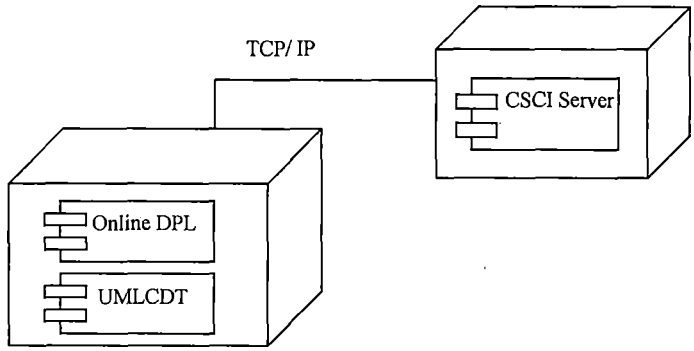

Figure 2. Deployment Diagram

2.2.1.1 System Interfaces. This project uses the server in the Computer Science Department at CSUSB as the server base. User can browse the Online DPL home page as well as the UMLCDT pages. The latest versions of browsers are suggested. On the client side, which is where the UMLCDT application is, the user can actually translate design pattern structures from XML to UML class diagram. The connection used between server and client is TCP/IP.

2.2.1.2 User Interfaces. The user interfaces for this project can be divided into two groups: server side and client side.

Server side functions:

- ✓ Allow the user to download the basic design patterns structures into his/her local machine;
- ✓ Allow the user to browse all design pattern structures from Gamma's book generated by the GUI;

- ✓ Allow user to view the combination of some different design pattern structures.

Client side functions:

- ✓ Allow user to retrieve design pattern XML file from his local directory;
- ✓ Allow user to see the UML class diagram based on the user input XML class structure;
- ✓ Allow user to modify the generated class diagram;
- ✓ Allow user to build new design patterns;
- ✓ Allow user to combine different design patterns;
- ✓ Allow user to save the modified class diagram into a XML file.

2.2.1.3 Hardware Interface. The server operation system in the Computer Science Department at CSUSB is used as a server site for this project and manages the hardware interface issues. There is no specific hardware related interface issues. However, it is suggested that this software run on any existing Linux platform or Windows platform with Intel processors Pentium II or higher. The suggested RAM is 128 MHz.

2.2.1.4 Software Interface. The project mainly consists of four parts. The first part includes a list of XML files generated based on Gamma's design pattern book. The second part generates Linked-List from the data in XML files. Apache's XML parser is selected to be the tool used for the process [11]. The third part of UMLCDT Project generates UML diagrams based on the data received from the XML parser. The last part is the GUI application that is written in Java 2. The Java version used for this project is JDK 1.3. The GUI allows the user to select one of the existing XML files or design his/her own. If the user chooses the existing file, it is then read by the compiler (written in Java) and converted into UML diagrams. Also, this GUI application can allow users to modify the existing or self-designed XML files or UML diagrams. The whole procedure is written in XML, Java and HTML.

2.2.1.5 Communications Interfaces. UMLCDT offers a sophisticated Graphic User Interface (GUI) that provides a friendly interface for the client side. The GUI is written in Java. The server side pages can be seen using the lastest Netscape or Internet Explorer. TCP/IP handles the connection between server and client side.

2.2.1.6 Operations. The system server should operate 24 hours a day, 7 days a week excluding maintains.

2.2.2 Product Functions

This project concentrates on the concrete design pattern. It includes the following functions:

- Allow user to browse the design pattern files and structures on the server
- Translate certain design pattern into UML class diagram from files
- Rename elements in the concrete design pattern, such as class, method and attributes
- Delete elements in the concrete design pattern, such as class, method and attributes
- Write back the concrete design pattern to files after modification
- Add classes for certain design pattern
- Add more details to the class

2.2.3 User Characteristics

This project is primarily designed for the ADAP project. The following capabilities are assumed to be true of the various users:

2.2.3.1 Novice Software Engineers. Designers have a background in design pattern. They are able to access and modify the XML and understand the usage of UML.

2.2.3.2 Professors and the Students Interested in Design Pattern. It is required that the students should have some knowledge of design pattern, background in UML. Knowledge of Java and XML is optional.

2.2.4 Constraints

Since this project only concentrates on the process of translating XML files into UML frame, the correctness of the design pattern is ignored here.

2.2.5 Assumptions and Dependencies

There is no specific assumption for this project.

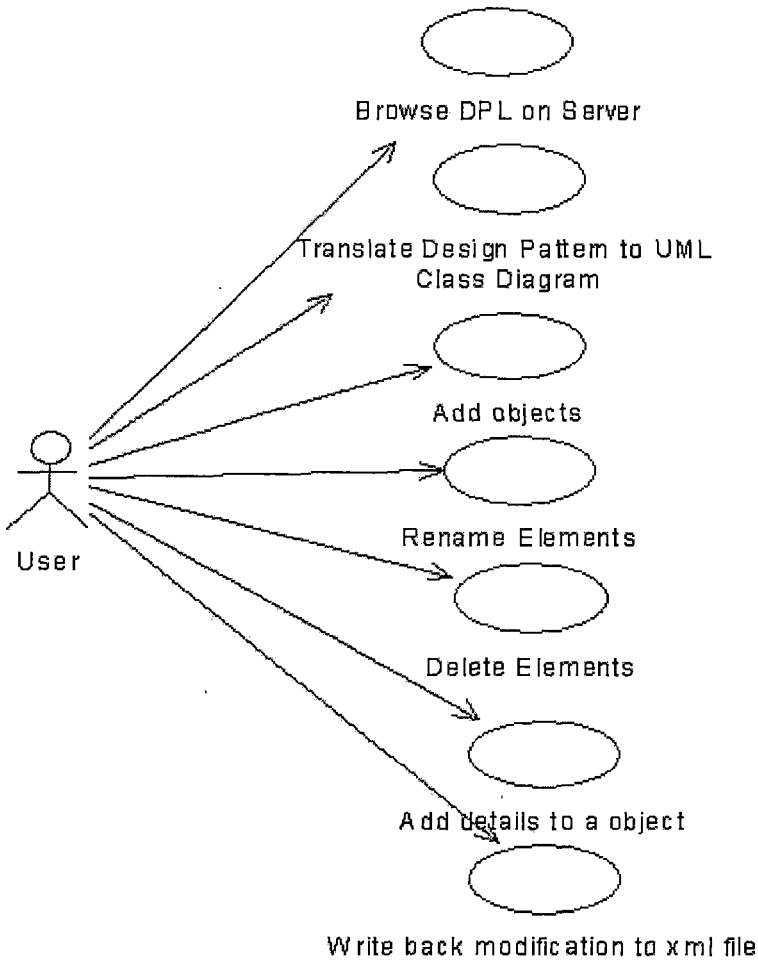

Figure 3. Use Case Diagram

2.3 Specific Requirements

2.3.1 External Interface Requirements

In this project, the design pattern is stored in XML format and is generated into a UML diagram (using compiler written in Java) in order to give the designers a complete picture of the design result. Users are able

to modify the diagrams either from XML files or the GUI application.

A list of links based on the design pattern from Gamma's book is listed on the UMLCDT main page. It allows the user to select a specific design pattern, view its UML diagram, its description and download its structure written in XML.

The generating process from XML file to class diagram is handled by the graphic user interface (GUI).

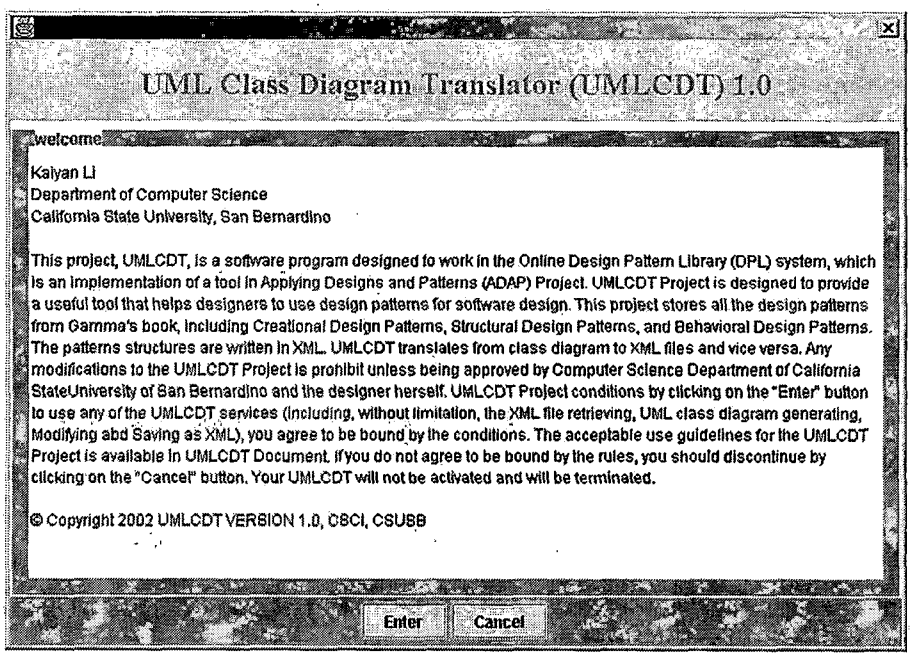

Figure 4: UMLCDT Welcome Page

The Welcome page will be the first page for the UMLCDT GUI. It includes the general information and

acknowledgment for UML class diagram translator. On the same page, there are two buttons for a user to choose. Choose "Enter" to get into the system panel; choose "Cancel" to exit and terminate the system.

After User chooses "Enter", he/she is able to see the following page.

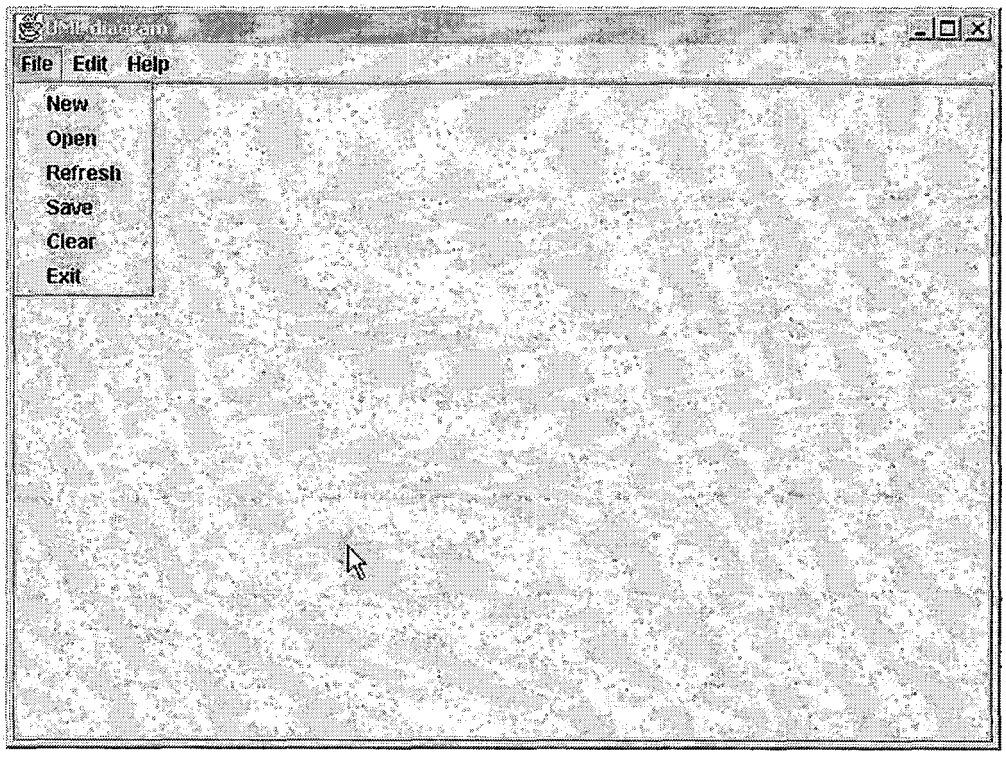

Figure 5. UMLCDT File Menu

As shown in Figure 5, the file menu in UMLCDT menu bar has the following functions:

New - Opens a new blank page that allows the user to design a design pattern.

Open - Opens an XML file input by user and displays its UML class diagram.

Refresh - Allows user to see the change after modifying a certain class diagram.

Save - Allows the user to save the current class diagram as .XML format based on user input file name.

Clear - Clears the data in the buffer and the current class diagram.

Exit - Allows the user to close the current window and exit the UMLCDT application.

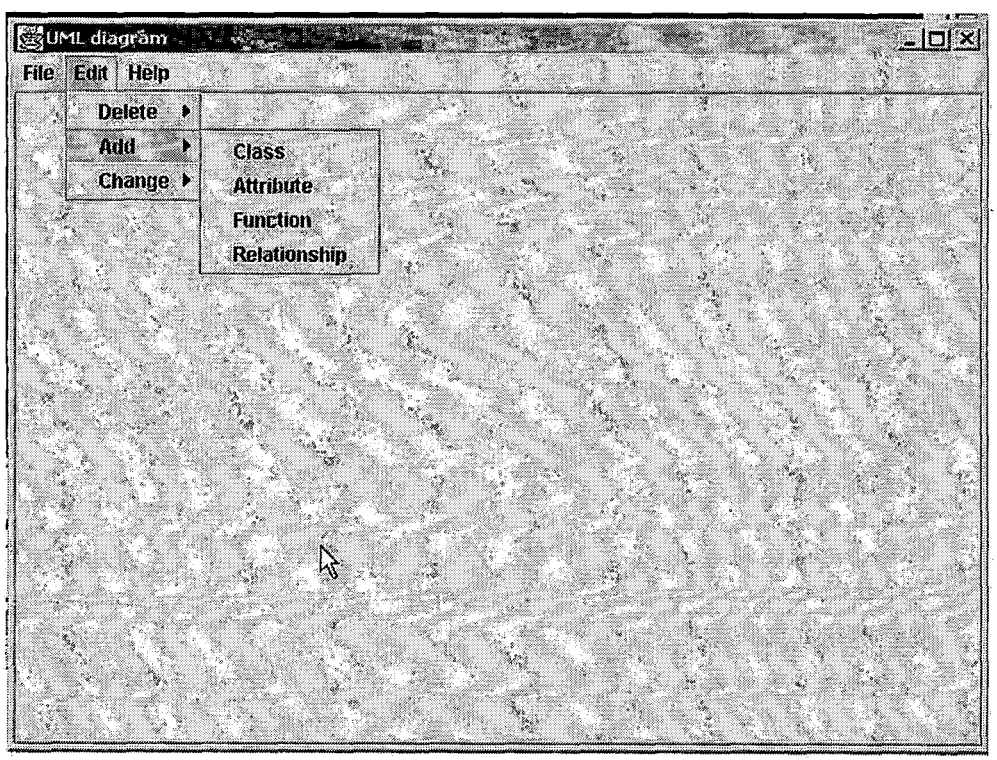

Figure 6. UMLCDT Edit Menu

As shown in Figure 6, the edit menu mainly contains three parts: Delete, Add and Change. Each of them has its own submenu that allows the user to modify certain part of the UML class diagram: Class, Attribute, Function and Relationship:

Class - A new class can be added based on the user input class name. An existing class can be deleted if the user wants to. Also, the class name can be changed according to user's choice.

Attribute - Based on a specific class selected by user input class name, its attributes can be added, deleted or changed. To change the existing attribute, the user can either choose to change its type, name or both.

Functions - Based on a specific class selected by user input class name, its functions can be either added, deleted or changed. The modification for argument for certain function is handled in this part.

Relationship - Based on a specific class, the specific relationship for that class can be added, removed or changed. Such as add a new relationship for the class, or delete a relationship for the class.

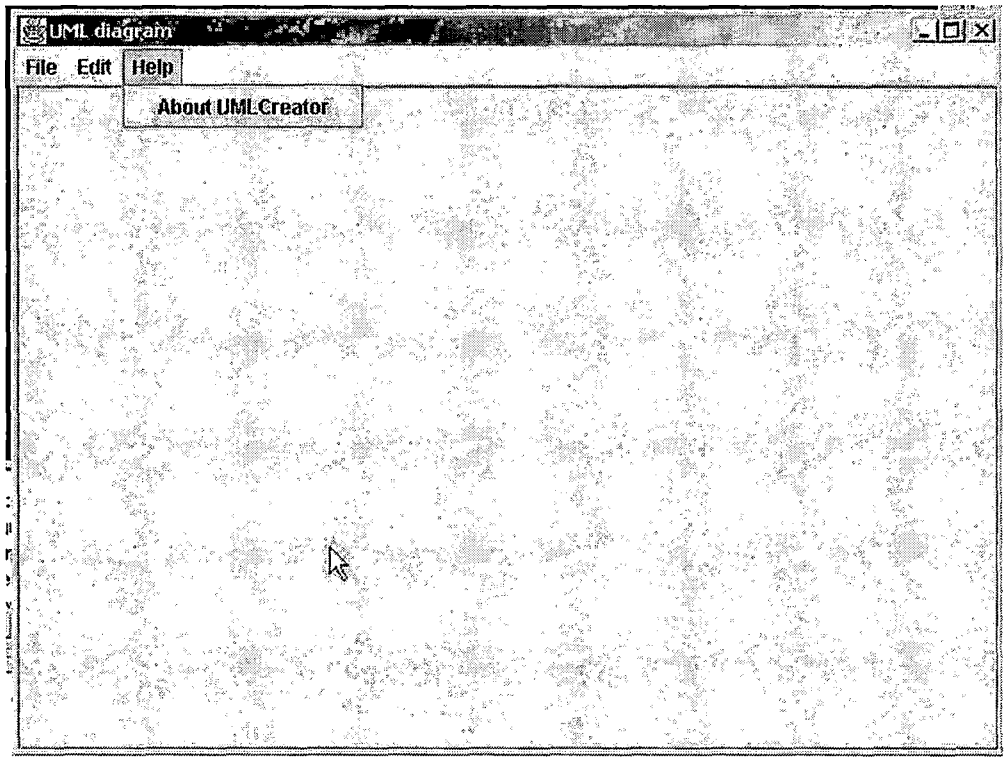

Figure 7. UMLCDT Help Menu

The Help menu in the UMLCDT menu bar is used for assisting the user to use the UMLCDT application. When it is clicked, a pop up window will show the guid for every menu choice from the menu bar.

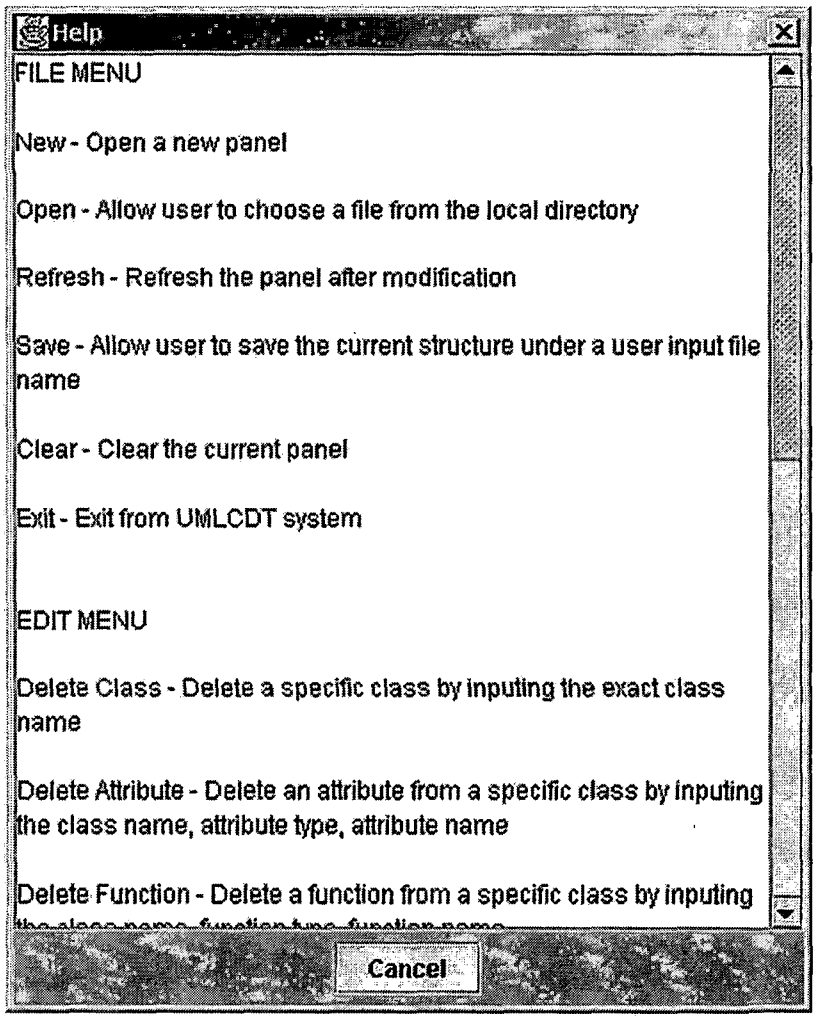

Figure 8. UMLCDT Help Page

2.3.2 Functional Requirements

The GUI of UMLCDT contains a menu panel and a graphic panel. All the functions provided by this GUI can be selected from the menu. It not only allows the user to select an existing design pattern and translate it to a class diagram, but also allows the user to design his/her own pattern based on new or existing ones.

Based on the whole picture of the class diagram, classes can be added or removed. Also, based on a single class, attributes, functions, arguments and relationships can be added, removed or modified.

2.3.3 Performance Requirements

The project requires only one terminal and one user. This project supports the particular design pattern provided by the Gamma's book.

2.3.4 Software System Attributes

2.3.4.1 Reliability. The system should be running 24 hours a day, 7 days a week except when the server is running maintenance schedule.

2.3.4.2 Maintainability. The entire project is written in XML, HTML and Java. Object-oriented programming is an approach used for design, so that maintenance can be easy. The development process has been documented to make further modifications easier.

2.3.4.3 Portability. All computers that have Linux or Windows installed are able to run this project. XML and Java are able to run on any workstation that has browser.

CHAPTER THREE

SOFTWARE DESIGN

3.1 Architecture

The UMLCDT Project implements the concrete design pattern layer in ADAP project. It reads in the class structures specification as a XML file, parses the data using XML parser, then analyzes the transformed data and generates a UML class diagram based on the structures. The diagram is shown on the Java GUI panel. User can modify the diagram using GUI and save back as a XML file. (see Figure 9.)

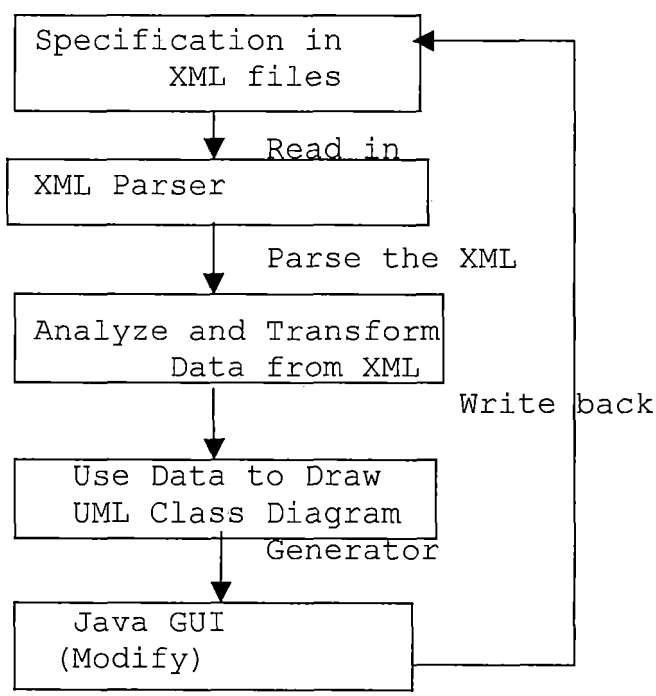

Figure 9. UMLCDT Structure

The basic strategy used for generating the UML class diagram from XML file is to parse the elements from XML file into several linked lists. The system then retrieves the data from the linked lists to generate the class diagram.

The XML file is based on the following structures: The XML file stores the class structure of a class diagram in text format. Each class element defines a structure for a class object. Each function element defines a function structure, which includes the argument structure. Each attribute element defines an attribute structure. All the function structures and attribute structures are included in the class structure where they belong. All the class structures are included in the UML structure. (See Figure 10)

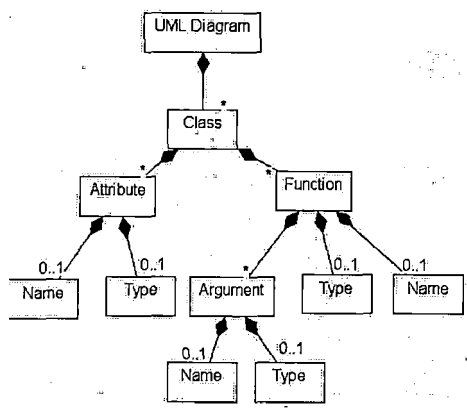

Figure 10. XML File Structure

There are a total of ten Java classes written for the UMLCDT project. The class diagram shows the structure and relationships for these classes. (See Figure 11.)

As shown in Figure 11, there are ten Java classes: ArgumentStructure, FunctionStructure, AttributeStructure, RelationshipStructure, ClassStructure, ClassDiagram, UMLDiagram, Connection, XMLHandler and UMLPanel.

The ArgumentStructure, FunctionStructure, AttributeStructure and RelationshipStructure classes are all in charge of creating the LinkedLists for each starting element in the XML file. They retrieve the data from XMLHandler class and put them into linked lists. All the functions used to set and get specific elements are designed in these classes.

The XMLHandler class works as a parser that parses the data from an XML file into the structures of each element. Since the structure of each start element has been created already, this class places all the needed data into those linked lists for each structure.

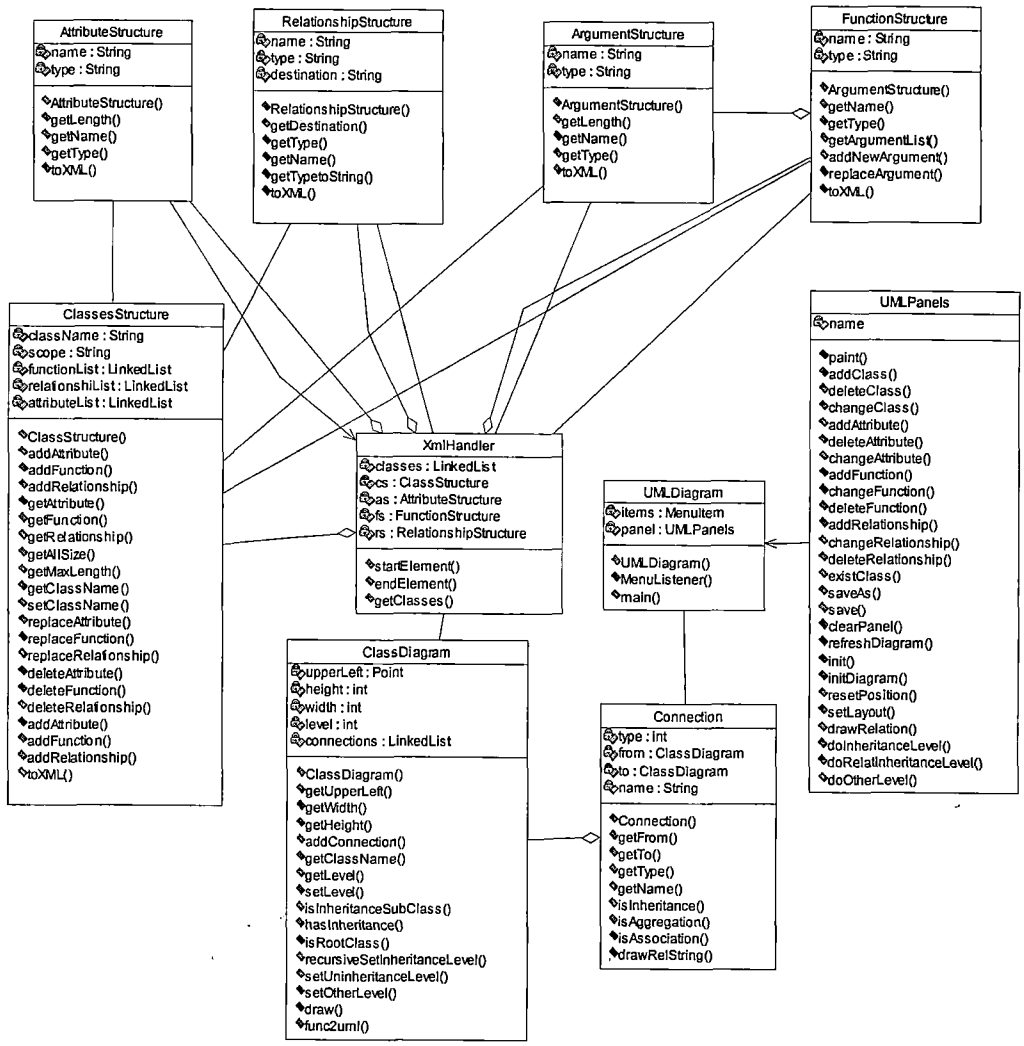

Figure 11. UMLCDT Class Diagram

The class that handles the drawing of class objects is the ClassDiagram class. This class has a function that calculates the width and height of each object.

The Connection class draws the relationships among the objects. It detects the position of two related

objects and draws the specific type of relationship between them.

Finally, the UMLDiagram and UMLPanel classes put all the above structures and drawing functions together to make them work as a whole application. UMLDiagram class creates the menu bars on the frame and links each menu to a specific function written in UMLPanel class. UMLPanel class calls the draw object classes and draw relationships functions from ClassDiagram class and Connection class. Also, all the modifications functions for GUI are written in this class.

The above files are all on the client side. On the server side, there are the same XML files as well as HTML files. The HTML files are linked to each other in order to create a convenient way. (See Figure 12.)

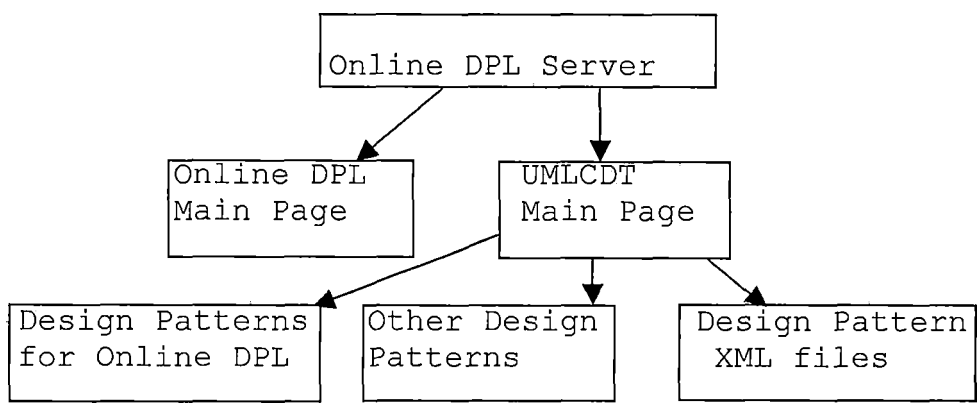

Figure 12. Connection Among the HTML Files

3.2 Detailed Design

The detailed design of each class is listed as follow:

Table 3. ArgumentStructure Class Design

ArgumentStructure.java

import libraries;
class ArgumentStructure {
function getLength() {
if ((name exists) and (type exists))
return string length of name and type with one more space;
else if (only name exists)
return string length of name with one more space;
else if (only type exists)
return string length of type with one more space;
else //there is no argument
return 0;
// getLength functions are used for counting the object width
}
 	
function toXML(PrintStream ps) {
write "<argument " to out put file;
if (function type exits) {
write "type = \"" + type + "\" ";
}
if (function name exists) {
write "name = \"" + name + "\" ";
}
write "/>" to close the argument element;
}
//to XML functions are used for writing the modified structure into XML format;
}

Table 4. AttributeStructure Class Design

ArtributeStructure.java

import libraries;
class ArttrbuteStructure {
function getLength() {
if ((name exists) and (type exists))
return string length of name and type with one more space;
else if (only name exists)
return string length of name with one more space;
else if (only type exists)
return string length of type with one more space;
else
return 0;
// getLength functions are used for counting the object width
}
 	
function toXML(PrintStream ps) {
write "<artribute " to out put file;
if (function type exits) {
write "type = \"" + type + "\" ";
}
if (function name exists) {
write "name = \"" + name + "\" ";
}
write "/>" to close the artribute element;
}
//to XML functions are used for writing the modified structure into XML format;
}

The above class builds a structure for the attributes in a specific class.

Table 5. ClassDiagram Class Design

ClassDiagram.java

```
import libraries;

public class ClassDiagram {
        function isInheritanceSubClass() {
                create iterator for connections;
                while (there is next iterator) {
                        Connection conn = (Connection) next iterator;
                        if (from class object is inheritance)
                                return true;
//find the classes that are involved in inheritance relationship but not the root}
        return false;}
        function hasInheritance() {
                create iterator for connections;
                while (there is next iterator) {
                        Connection conn = (Connection) next iterator;
                        if (there is inheritance relationship)
                                return true;}
                return false;
//find classes that are involved in inheritance relationship                }
        function setRelatLevel() {
        find class objects that already have a level greater than the initial level;
                set their related object level to the same levels;}
        function setOtherLevel(int l) {
                find objects that still on the initial level;
        Set level to object related with aggregation, composition and association;}
function draw (Graphics g) {
    set color to black;
    get left and top point;
    draw rectangle using left, top, width, height;
    draw string for class name;
    retrieve attribute list;
    if (attribute list is not empty) {
       draw line;
       create iterator for attribute list;
       while (there is attribute) {
         draw string for attribute;}
    retrieve function list;
    if (function list is not empty) {
    draw line;
        create iterator for function list;
       while (there is function) {
         draw string for function;}}
//draws the entire class object
  }}
```

Table 6. ClassStructure Class Design

ClassStructure.java

import libraries;
class ClassStructure {
public int getAllSize() {
if (attributeList is not empty)
get size of attributeList;
else
attribute size = 0;
if (functionList is not empty)
get size of functionList;
else
function size = 0;
return attribute size + function size with five more spaces;
//use the number of total attributes and functions to be the class object height }
function getMaxLength() {
set the string length of class name to be the maxLength;
set x to be the longest string in attributes and functions;
if (x is greater than maxLength)
set value of x to maxLength;
return maxLength;
//find the longest string length to be the class object width; }
public void replaceAttribute ( ) {
create new attributeList listIterator;
while (there is next iterator)
{ if same attribute found;
delete old attribute;
add new attribute;
return;}}
//if no match attribute
show error message "replace: No match attr found!";}
public void deleteAttribute(String type, String name) {
create new attributeList iterator;
while (there is next iterator)
{ AttributeStructure as = next AttributeStructure;
if the same attribute found {
delete that attribute;
return;}}
//if not match attribute
show error message "delete: No match attr found!";
}
}

Table 7. Connections Class Design

Connection.java

```
import libraries;
class Connection {
        create graphic pointers for different kind of relationship

  if (type is association) {
   // draw association relationship
     set color to green;
     if (from class and to class are on the same level) { //same level
     draw relationship line and pointers between two classes;
    else if (from class and to class are on the different level) {
                draw relationship line and pointers between two classes;
}
   else if (type == aggregation)
          // draw aggregation relationship
    set color to red;
     if (from class and to class are on the same level) { //same level
     draw relationship line and pointers between two classes;
    else if (from class and to class are on the different level) {
                draw relationship line and pointers between two classes;
}
  else if (type == inheritance) {
       if (from class and to class are on the same level) { //same level
     draw relationship line and pointers between two classes;
    else if (from class and to class are on the different level) {
                draw relationship line and pointers between two classes;
}
   else if (type == composition) {
     if (from class and to class are on the same level) { //same level
     draw relationship line and pointers between two classes;
    else if (from class and to class are on the different level) {
                draw relationship line and pointers between two classes;
}
     else //when the type is not one of above 4 types
    show message "The relationship type should be inheritance, aggregation or
association!";
 }
}
```

Table 8. FunctionStructure Class Design

FunctionStructure.java

```
import libraries;

class FunctionStructure {
  function getLength() {
   if (argument exists)
     get argument length;
     if ((name exists) and (type exists))

          return string length of name and type with one more space;
     else if (only name exists)
          return string length of name with one more space;
     else if (only type exists)
          return string length of type with one more space;
     else
          return 0;
// getLength functions are used for counting the object width
  }

  function toXML(PrintStream ps) {
     write "<function " to output file;
     if (function type exits) {
          write "type = \"" + type + "\" ";
     }
     if (function name exists) {
          write "name = \"" + name + "\" ";
     }
     write "/>" to close the artribute element;
  }
//to XML functions are used for writing the modified structure into XML format;
}
```

Table 9. Panels Class Design

Panels.java

import libraries;
 	
class WelcomePanel extends JDialog {
set border to "welcome";
type text and set it to uneditable;
put panel in the center;
create "Enter" button; //enter into the GUI
create "Cancel" button; //exits from the system
}
 	
class AboutPanel extends JDialog {
type text and set it as uneditable;
create "Cancel" button; //unshow the dialog
}

Table 10. RelationshipStructure Class Design

RelationshipStructure.java

import libraries;
public class RelationshipStructure {
public void toXML(PrintStream ps) {
write ("<relationship type = \"") into output file;
write (types[type]);
write ("\" destination = \"");
write (destination);
if (name is not empty) {
write ("\" name = \"");
write (name);
}
write ("\"/>") to close the relationship element;
//write the modified relationship into a XML file}}

Table 11. UMLDiagram Class Design

UMLDiagram.java

import libraries;
 	
class UMLDiagram extends JFrame {
create menu "File", "Edit", "About";
function MenuListener implements ActionListener {
public void actionPerformed(ActionEvent event) {
//acts according to the menu that was selected
if (user choose "exit")
exit from system;
public static void main(String args[]) {
call WelcomePanel class;
set it to visible; }}
 	
class UMLPanel extends JPanel
{ function paint(Graphics g) {
if (there is noclass diagram)
return;
else //class diagram exists
create new iterator for the values of diagrams;
while (there is diagram) {
draw class diagram;}
call drawRelation(g) function; }
function init() {
go to local directory;
choose a file name;}
function initDiagrams() {
set default position for every object;}
function resetPosition() {
reset position for object according to their relationships; }
function setLayout() {
//assign a level to each class object according to its relationship
call doInheritanceLevel();
call doNotInheritanceLevel();
call doOtherLevel();
call resetPosition();}
function drawRelation(Graphics g) {
For each class
Get its positon and its relationship
call drawConnection( ) function; } }

Table 12. XMLHandler Class Design

XMLHandler.java

import libraries;
 	
class XmlHandler extends DefaultHandler
{
check start element in XML file;
if (localName equals ("class")) {
get value from ("className");
get value from ("scope");
add this class to class structure list;}
 	
else if (localName equals ("attribute")) {
get value for ("type"),
get value for ("name"));
add this attribute to attribute structure list;}
 	
else if (localName equals ("function")) {
get value for ("type"),
get value for ("name");
add this function to function structure list;}
 	
else if (localName equals ("argument")) {
get value for ("type"),
get value for ("name");
add this argument to argument list;}
 	
else if (localName.equals("relationship")) {
get value for ("type"),
get value for ("destanation");
get value for ("name");
add this relationship to relationship list;}}
 	
function endElement( ) {
if (localName equals ("class")) {
put all above data into a class structure;}}
 	
public LinkedList getClasses() {
return classes;}}

The detailed design of XML file is listed below:

Define XML file version, name, author;

Table 13. XML File Design

XML File
<start element UML>   <start element Class> define class name   <start element attribute> define attribute type, name <end element attribute>   <start element function> define function type, name   <start element argument> define argument type, name   <end element argument>   <end element function>   <end element class> <end element UML> // The XML file stores the class structure of a class diagram in text format. It has a start element and an end element for each class content element.

3.3 Algorithm and Error Handling

The Algorithm of Automatic Layout of Object-Oriented Software Diagram for UMLCDT Project is listed below:

- Set all object levels to -1;
- Search the design pattern structure for inheritance
- relationship. Recursively set level for the objects involved in inheritance relationship, start with 0;
- In case there is no inheritance in the structure, recursively set level for the object involved in other types of relationship, start with 0;
- Check if the level of every object is greater than -1. If no, get the level from or to which it is related to and assign to it.

The errors handled by UMLCDT Project are listed as follow:

- User input file should be a XML file. In case it is not a XML file or there is no such file in the selected directory, error message pops up;
- Class name cannot be duplicate. In case the user inputs a class name that already exists, display error message;
- Relationship type can only be aggregation, association, inheritance or composition. Error message appears if the user inputs relationship type other than above four;
- The design pattern structure should be complete. In case it is not, such as missing class, error message pops up;
- In case of modify an exist design structure, error message appears if the input class, attribute or function is not in the current structure.

CHAPTER FOUR

TESTING AND EXAMPLES

There are two scenarios for the UMLCDT project: server side and client side. For the server side, this project uses HTML pages to build the user interface in order to allow users to access different design patterns. For the client side, Java graphic user interface (GUI) is used.

4.1 Server Side

The following test cases are from the server side:

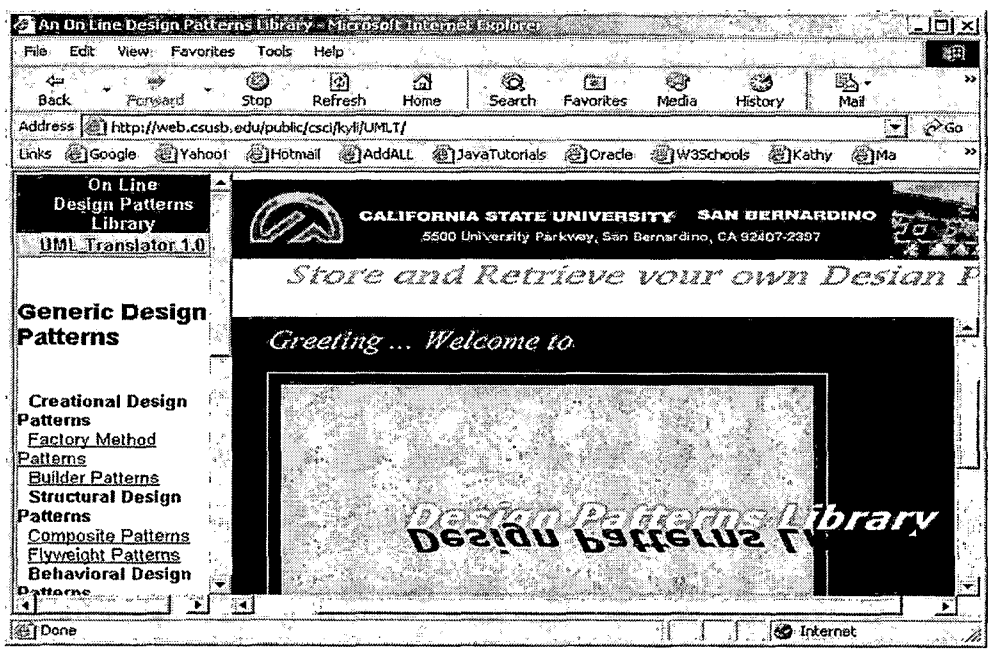

Figure 13. Online Library Welcome Page

This page is the Online DPL welcome page, which is the first page users see when browsing the web site for this project. At the upper left side, you can find the link for UMLCDT project. Click on that line, it will lead you to the following page:

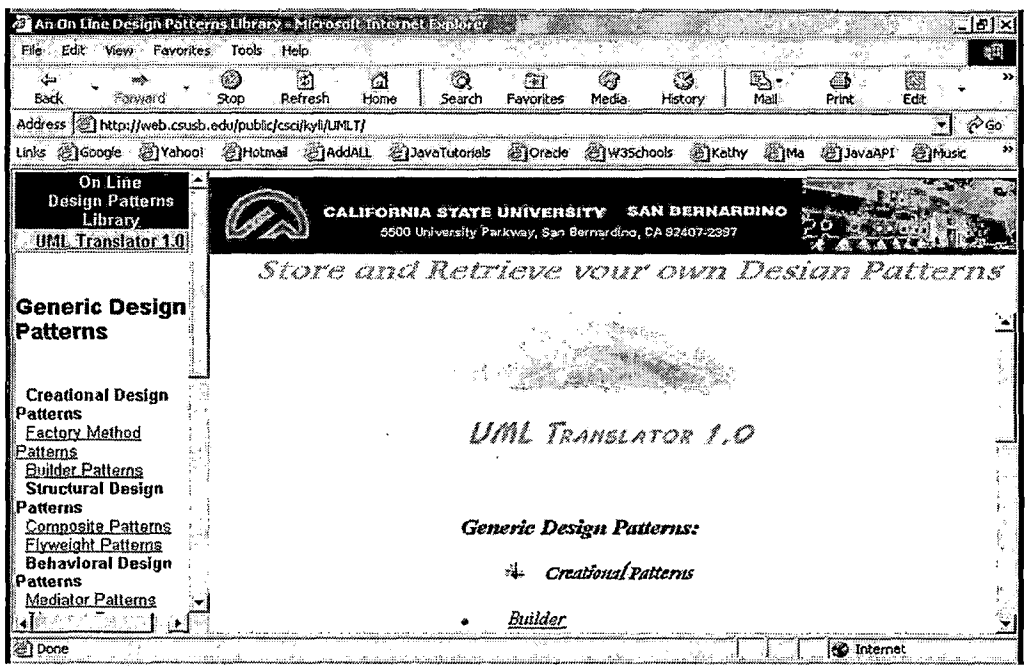

Figure 14. UMLCDT Main Page (First Half)

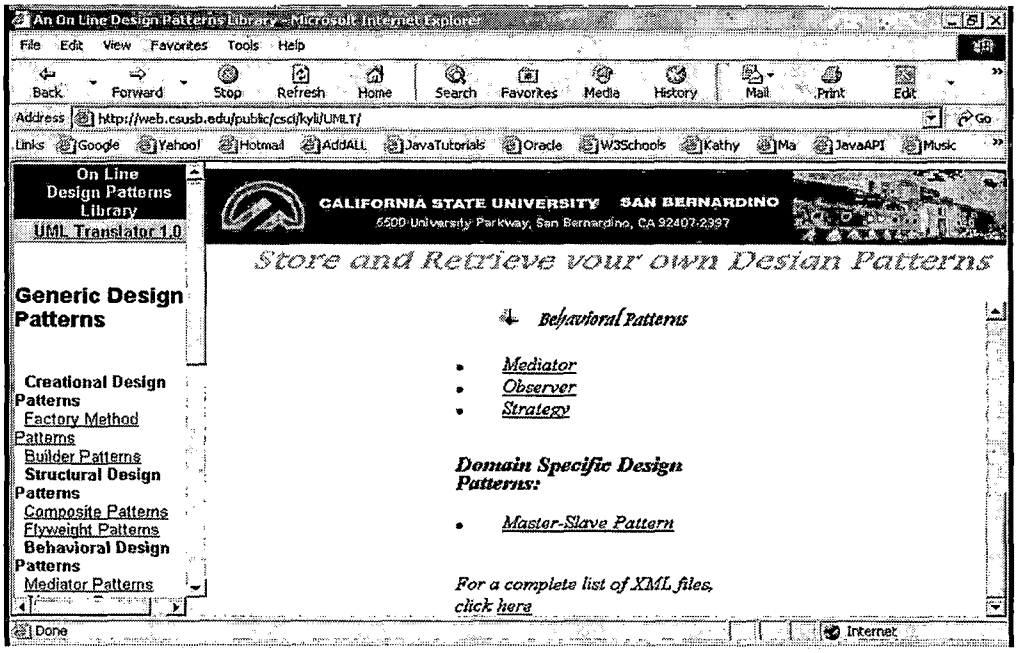

Figure 15. UMLCDT Main Page (Second Half)

The above page is the home page of the UMLCDT project. This page displays a list of links based on all the design patterns used in Online DPL. They have been grouped into two major groups: Generic Design Pattern and Domain Specific Design Pattern. Each link leads to a specific design pattern page. For testing purpose, Builder pattern is chosen here. Click on the Builder pattern link, and the user will see the next page:

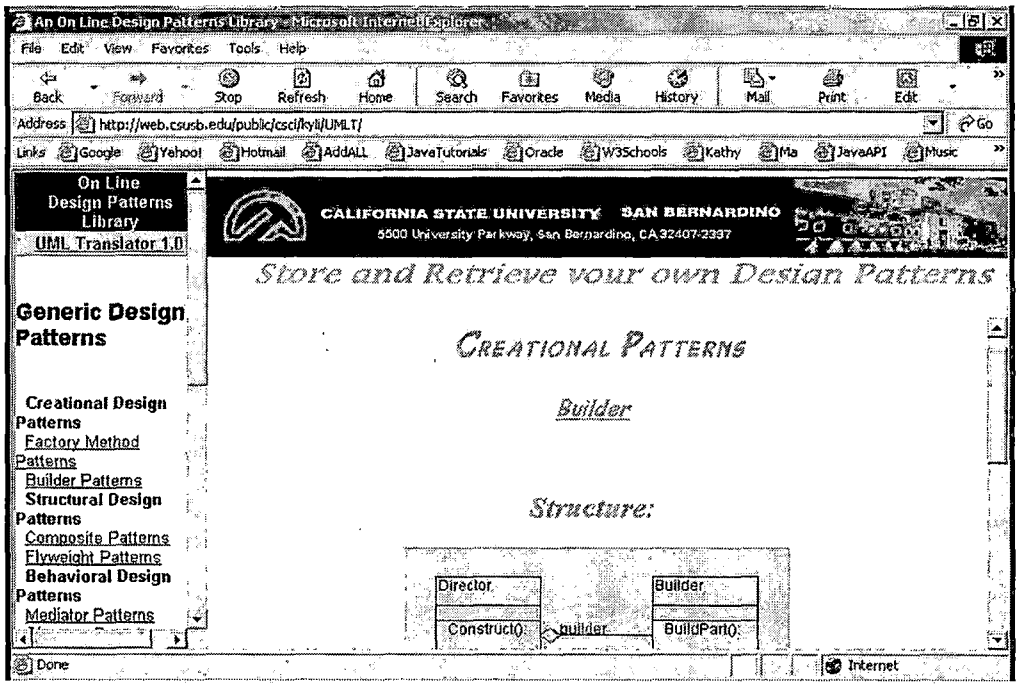

Figure 16. Specific Pattern (First Half)

The specific pattern pages are used as examples for the UMLCDT GUI design result. The figures displayed in each specific pattern page are all generated by the UMLCDT client side. The class objects are displayed in black rectangles. The different relationship lines are displayed by using different colors.

Followed by this figure is the second half of the same page.

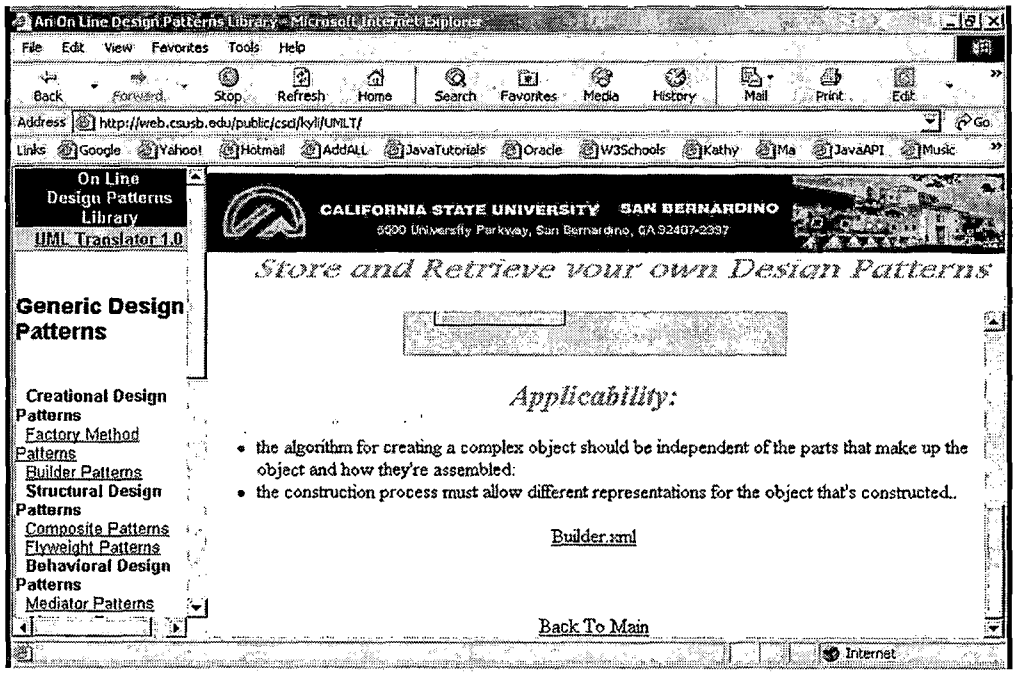

Figure 17. Specific Pattern (Second Half)

As mentioned above, the specific design pattern page displays after the user selects a certain link of the design pattern name. This page includes the UML class diagram for this design pattern generated from UMLCDT, a XML file for this design pattern (available to be downloaded) and a description of this design pattern.

In order to see the XML file for certain design pattern, click on the pattern_name.XML link. For this example, after clicking Builder.XML link, the following page appears:

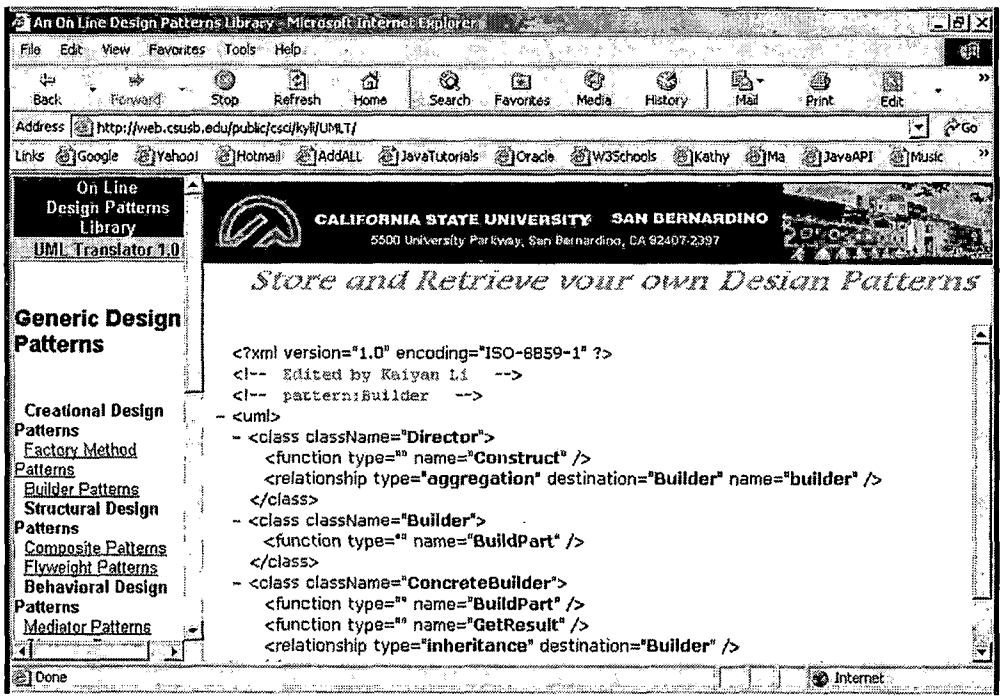

Figure 18. Design Pattern XML Page

This page is the XML source code for the certain design pattern structure. The user can either download this file from this page or right click the XML link on previous page to download.

4.2 Client Side

For the Client side, which is the Java Graphic User Interface (GUI) application, there is the UMLCDT GUI page. It allows the user to select one of the existing design patterns written in XML, or design his/her own patterns. The newly designed or modified patterns can be saved into a XML file based on user's preference.

There are several example pages listed as follows, which display some basic functions that UMLCDT offers:

The following pop up window (See Figure 20) appears after the user clickes "Open" under File menu in the UMLCDT menu bar. It requires the user to input an existing .XML file name along with its path into the input area in order to let the system know which file should be read and parsed in. For this particular example, design pattern named Builder grouped in Creational Patterns is selected.

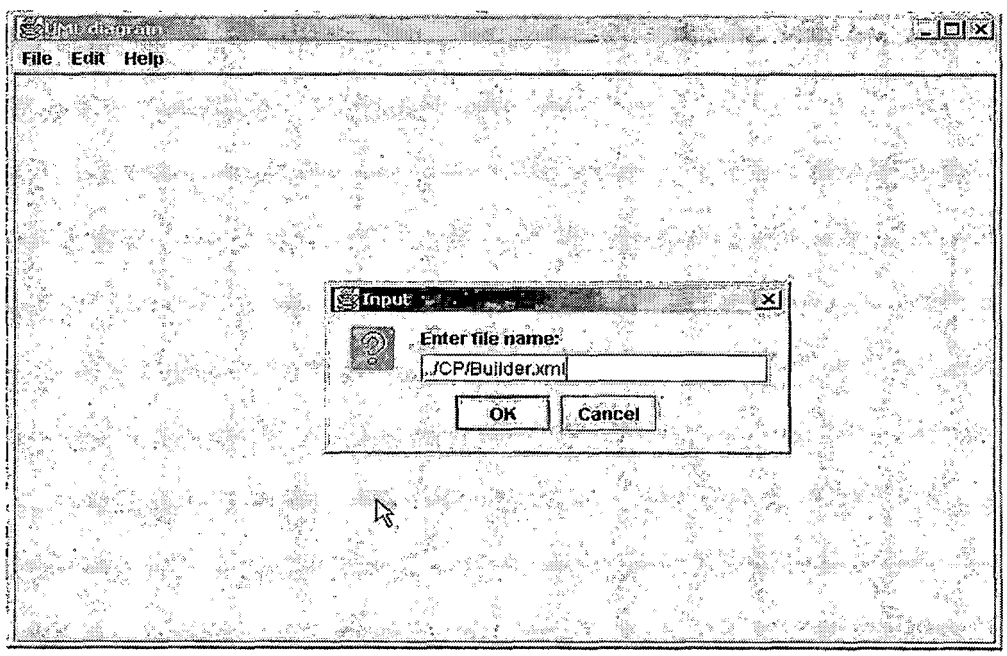

Figure 19. Accepting User Input

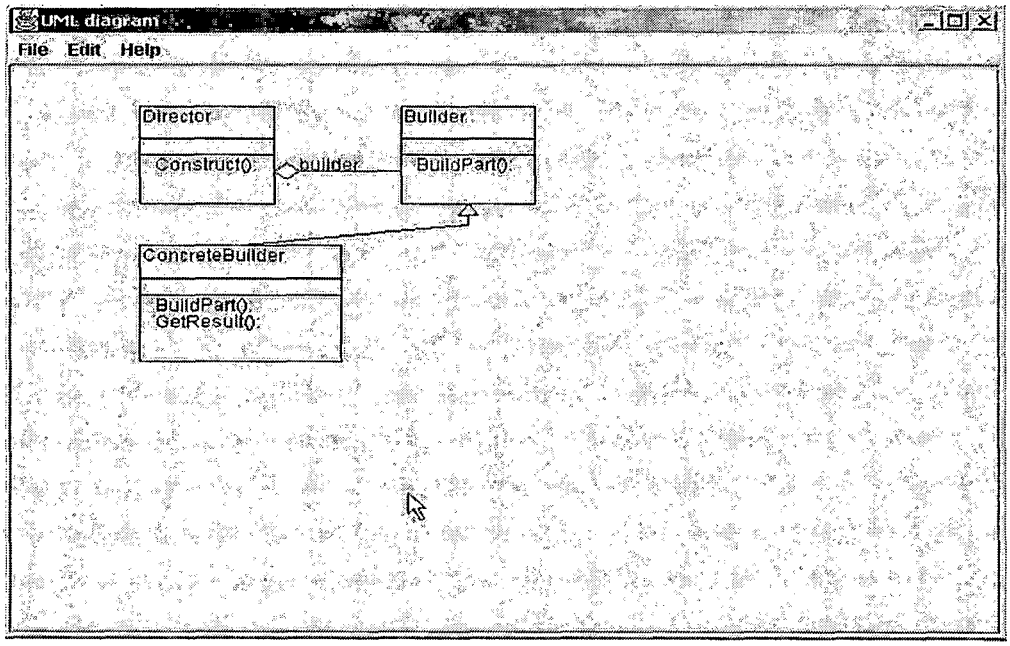

Figure 20. UMLCDT Class Diagram

The above figure displays the system drawn UML class diagram for the Builder pattern, which is selected from the previous page. This page appears only if there is no error during compiling and generating. The user can choose to modify this diagram if he/she prefers. Here, let us see what happens when the "Director" class is deleted from this structure:

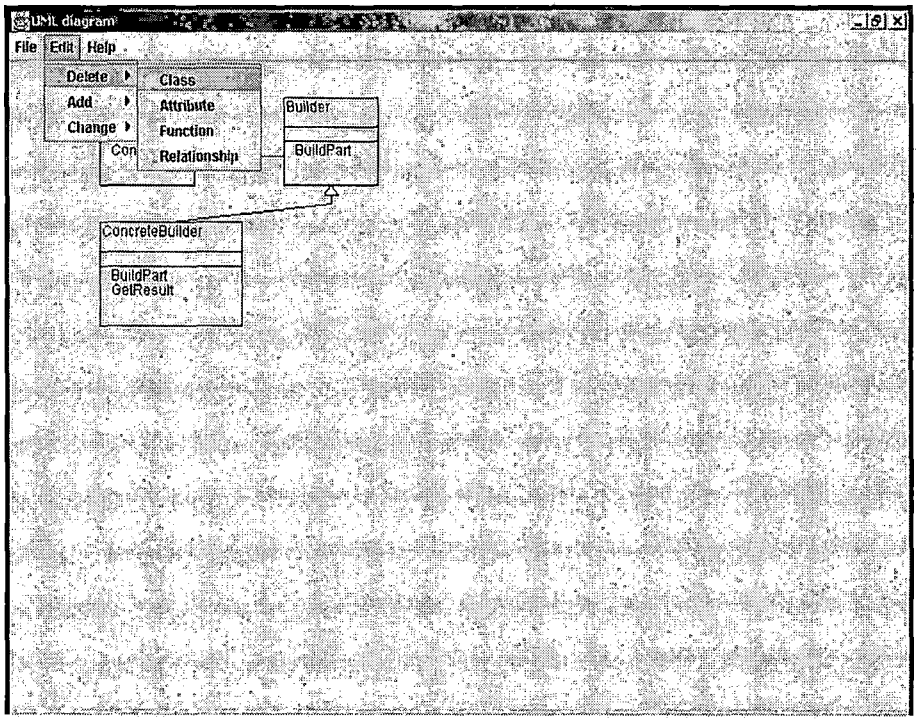

Figure 21. Choose Delete Class

As shown above, to delete a class from the existing structure, choose Edit -> Delete -> Class.

After clicking on the Class menu, a pop up window will ask which class needs to be deleted:

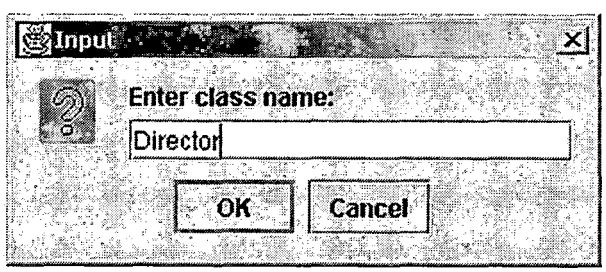

Figure 22. Pop Up Window for User Input

Here choose class Director. Click on "OK", then click File -> Refresh. The following page displays the result.

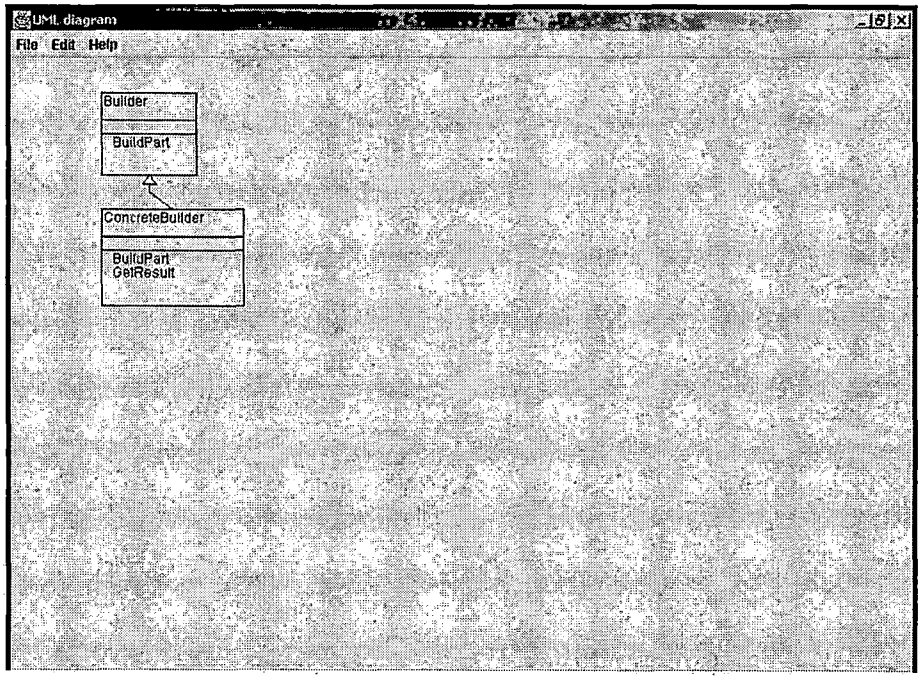

Figure 23. Result After Delete

After each modification, the refresh button under the file menu must be clicked in order to see the change. At the end of the modification, the user can choose to save all the changes as an .XML file. The newly saved file name is input by the user.

In addition to the above functionalities, UMLCDT can combine two or more different design patterns into a new one. To do so, simply go to the XML file, copy the class

structures of one design pattern to another design pattern. This function has also been tested.

First, a combination of Builder and Chain of Responsibility Design Patterns is tested.

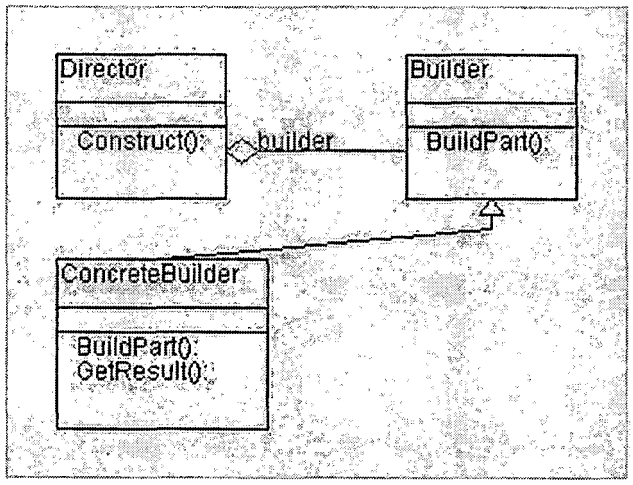

Figure 24. Builder Design Pattern

The Builder design pattern's XML file is listed below:

Table 14. Builder Design Structure

Builder.xml
<?XML version="1.0" encoding="ISO-8859-1"?> <uml><class className="Director"> <function type="" name="Construct"></function> <relationship type="aggregation" destination="Builder" name="builder"></relationship></class> <class className="Builder"><function type="" name="BuildPart"></function> </class><class className="ConcreteBuilder"> <function type="" name="BuildPart"></function> <function type="" name="GetResult"></function> <relationship type="inheritance" destination="Builder"></relationship> </class></uml>

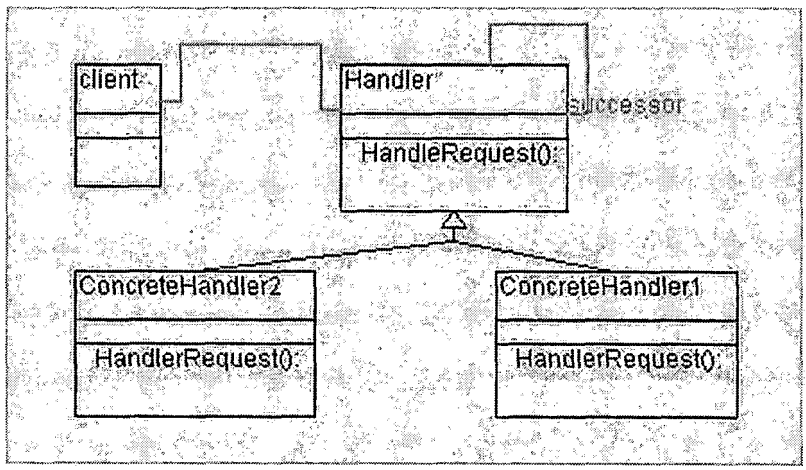

Figure 25. Chain of Responsibility

The Chain of Responsibility design pattern's XML file is listed below:

Table 15. Chain_of_Responsibility Design

Chain_of_Responsibility.xml
<?XML version="1.0" encoding="ISO-8859-1"?> <uml> <class className="Handler"> <function type = "" name = "HandleRequest"></function> <relationship type = "association" destination="Handler" name="successor"> </relationship> </class> <class className="client"> <relationship type="association" destination="Handler"></relationship> </class> <class className="ConcreteHandler1"> <function type = "" name="HandlerRequest"></function> <relationship type="Inheritance" destination="Handler"></relationship> </class> <class className="ConcreteHandler2"> <function type = "" name="HandlerRequest"></function> <relationship type="Inheritance" destination="Handler"></relationship> </class> </uml>

After combining the class structure of above two design patterns, UMLCDT generates the following class diagram based on the new combined design pattern:

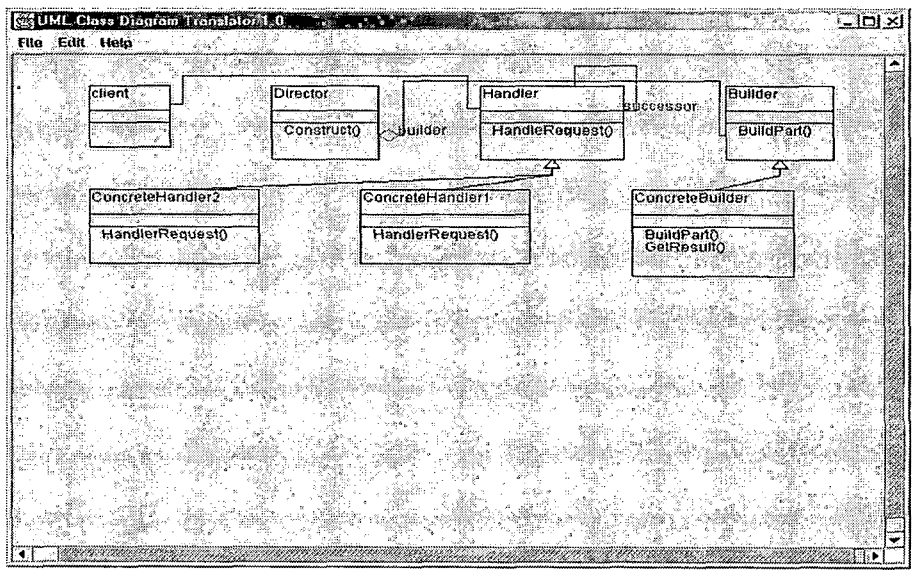

Figure 26. Builder and Chain of Responsibility

As the figure displayed, the Builder design pattern and the Chain of Responsibility design pattern are merged together as a new design pattern. The exchange of the positions of these class objects are done by the UMLCDT client side GUI system. Generally speaking, the positions are generated by their level and relationships.

The combining design pattern's XML file is shown below:

Table 16. Builder and Chain Design

BuilderAndChain_of_Responsibility.xml

```
<?XML version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by Kaiyan Li -->
<!-- pattern:Combine -->
<uml>
 <class className="Handler">
  <function type = "" name = "HandleRequest"></function>
  <relationship type = "association" destination="Handler" name="successor">
</relationship>
  </class>
  <class className="client">
  <relationship type="association" destination="Handler"></relationship>
</class>
<class className="ConcreteHandler1">
<function type = "" name="HandlerRequest"></function>
<relationship type="Inheritance" destination="Handler"></relationship>
</class>
<class className="ConcreteHandler2">
<function type = "" name="HandlerRequest"></function>
<relationship type="Inheritance" destination="Handler"></relationship>
</class>
<class className="Director">
<function type="" name="Construct"></function>
<relationship type="aggregation" destination="Builder"
name="builder"></relationship>
</class>
<class className="Builder">
<function type="" name="BuildPart"></function>
</class>
<class className="ConcreteBuilder">
<function type="" name="BuildPart"></function>
<function type="" name="GetResult"></function>
<relationship type="inheritance" destination="Builder"></relationship>
</class>
</uml>
```

The design patterns used for testing are all from the Gammas's book [5]. The functions of UMLCDT worked properly for these design patterns.

CHAPTER FIVE

MAINTENANCE MANUAL

5.1 Source Directory

There are twenty-four HTML pages, a GUI application that offers more than 18 functions, twenty-three XML files, and ten Java Classes in the UMLCDT Project. All these files are located under the XMLProject directory. Also, jdk1.3 from Java.sun [10] and Apache parser xerces-2_0_1 [11] are placed under the same directory in order to support the development. There are four folders under the XMLProject directory: src, XML, web and xerces-2_0_1.

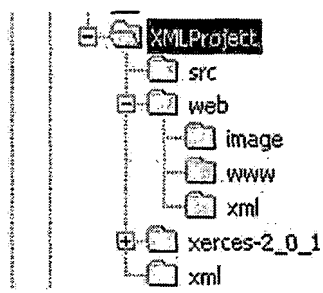

Figure 27. XMLProject Folder

The /src folder contains all the Java source code for the UMLCDT application; the /web folder contains all the sources for the server side, including HTML pages, images and XML files; the /XML directory contains all the

XML files and the /xerces-2_0_1 directory contains the necessary libraries for the XML Parser.

The following is a list of all the XML files in /XML directory:

- Abstract_Factory.XML
- Builder.XML
- Factory_Method.XML
- Prototype.XML
- Singleton.XML
- Adapter.XML
- Bridge.XML
- Composite.XML
- Decorator.XML
- Facade.XML
- Flyweight..XML
- Proxy.XML
- Chain_of_Responsibility.XML
- Command.XML
- Interpreter.XML
- Mediator.XML
- Observer.XML

- State.XML
- Strategy.XML
- Template_Method.XML
- Visitor.XML
- MasterSlave.XML

All the Java files under /src directory are listed as follow:

- ArgumentStructure.Java
- FunctionStructure.Java
- AttributeStructure.Java
- RelationshipStructure.Java
- ClassStructure.Java
- XMLHandler.Java
- ClassDiagram.Java
- Connection.Java
- UMLDiagram.Java
- UMLPanel.Java

The following is a list of all the HTML files under /web/www directory:

- Abstract_Factory.html
- Builder.html

- Factory_Method.html
- Prototype.html
- Singleton.html
- Adapter.html
- Bridge.html
- Composite.html
- Decorator.html
- Facade.html
- Flyweight.html
- Proxy.html
- Chain_of_Responsibility.html
- Command.html
- Interpreter.html
- Mediator.html
- Observer.html
- State.html
- Strategy.html
- Template_Method.html
- Visitor.html
- MasterSlave.html
- UMLMain.html

5.2 Installation Process

The following information is used for installation:

1. In order to run the GUI application, JDK 1.3 needs to be installed from Java.sun.com (The user does not need to install the Apache XML parser since it is included in the xerces-2_0_1 folder).
2. Retrieve /XMLProject directory from CD and place it directly under C: drive.
3. Double click the XMLProject folder, make sure all five folders are under the XMLProject directory: src, XML, classes, web and xerces-2_0_1.
4. Set the environment variable for XML Parser library. Go to Start -> Control Panel -> System, choose Advanced, click on Environment Variables, choose New under System Variables.
5. Type *classpath* for the variable name, type *c:\XMLProject\xerces-2_0_1\xercesImpl.jar; c:\XMLProject\xerces2_0_1\XMLParserAPIs.jar;* as the variable values.

5.3 Recompile

After creating the environment variables, the application can be run. In order to do that, click start

-> run, type in *cmd* to open the command window. In the command window, go to the /src directory where all the Java source code located, type *Javac *.Java* to compile all the source programs available, then, type *Java UMLDiagram* to run the application. The Java GUI will appear after this step.

In order to go to the server side, the user must have at least one of these two browsers available: Internet Explorer or Netscape. The URL for the server side is

http://web.csusb.edu/public/csci/kyli/UMLT/

5.4 Modifications

If XML file is to be changed to other type of text file, modify XMLHandler.Java class.

To modify the algorithm used to generate the position of class objects, the user needs to change both ClassDiagram.java and UMLDiagram.java.

Modify the Connection.Java to change to algorithm for drawing the relationships.

After the modification, user has to recompile the whole application.

CHAPTER SIX

CONCLUSIONS AND FUTURE DIRECTIONS

6.1 Conclusions

UMLCDT Project is an extention project of ADAP. It implemented the concrete design pattern layer in ADAP [3]. A UML generating tool is built that allows users to translate a UML class diagram directly from XML files and vice versa.

There are several benefits of UMLCDT. First, it does not only includes the design patterns stored in the Online DPL Project, but also has the complete design patterns from Gamma's book [5]. The list includes each design pattern's usage, functions, structures and descriptions. This change makes all Gamma's design patterns available online for users to browse and download.

Secondly, UMLCDT assists a novice software engineer in using design patterns. The software engineer can get a complete descroption of what design patterns are used for and how to use them by browsing the UMLCDT web site. Also, from the client site, the novice software engineer

can modify the downloaded design patterns or to create new design patterns.

Third, UMLCDT saves time for software engineers since this system draws the diagram automatically instead of manually. This feature solves the problem that most novice design pattern designers had. The designers do not have to worry about where to place the newly added objects or how the relationship lines should be drawn. Also, he/she does not have to reorganize each object after removing one. Adding, changing and removing attributes and functions can be easy also. The user can just type in the class name and make the changes.

Moreover, UMLCDT allows the user to save the diagram as an XML file that clearly lists the hierarchy level and relationship among all the classes.

Also, since the user can choose to modify the diagram from either the XML files or the class diagram, the range of choices for modification is enlarged. For a novice designer who feels uncomfortable with XML, the UMLCDT GUI can be used to design the design patterns. If the user prefers the text file, the user can go to the XML file to modify the structure, which offers another functionality - combining different design patterns into

a new one. This function makes it easy for software engineers to create combinations of design patterns.

All these features make the UMLCDT project a very useful tool for software designers.

6.2 Future Directions

There are some major directions for further development. First, the algorithm of Automatic Layout of Object-Oriented Software Diagram can be further modified in order to handle more complicated class diagram structures. The algorithm for layout diagrams can be modified to allow user to drag and move the class object.

For detailed design, drop-down combo boxes can be created for user input text boxes. Also, the object-oriented structure of the source code can be further modified to increase reusability.

Next, it is possible to do more types of UML diagrams instead of just class diagrams. Also, more functions could be added into this project, such as box types and class scope. Other types of relationships included in class diagram could be added too.

By using XML files as input and output, the XML files can be further used as the source for a compiler

that detects inconsistencies in the edited design pattern. This will provide a syntax compiler that is responsible for the correctness of user created design patterns before generating the class diagram.

Also, since the current implementation has numerous repetitious codes, it is possible to improve the reusability of the classes.

Last, it is possible to generate XML files into C++ or Java file, which enlarges the range of object-oriented design for software engineers. All these directions can lead the UMLCDT Project to a new level.

APPENDIX A

JAVA SOURCE CODE

Source Code of ArgumentStructure.java

```
import java.util.LinkedList;
import java.io.PrintStream;

public class ArgumentStructure {
  String type;
  String name;

  public ArgumentStructure(String atype, String aname) {
     type = atype;
     name = aname;  }
  public int getLength() {
     if ((name!=null && name!="") && (type!=null && type!="")
       return name.length() + type.length() + 1; //1 for space
     else if ((name!=null && name!="") && (type==null || type==""))
       return name.length() + 1;
     else if ((name==null || name=="") && (type!=null && type!=""))
       return type.length() + 1;
     else
       return 0;}
  public String getName() {
     return name;}
  public String getType() {
     return type;}
  public String toString() {
     String s = "argument name:\t" + name + "\n";
     s += "argument type:\t" + type + "\n";
     return s;}
  public void prettyPrint() {
     System.out.print(type + " " + name);}
  public void toXML(PrintStream ps) {
     ps.print("<argument ");
     if (type != null && type != "") {
ps.print("type = \"");        ps.print(type);              ps.print("\" ");
                        }      if (name != null && name != "") {
                       ps.print("name = \"");
                       ps.print(name);
                       ps.print("\" ");
ps.println("/>");          } }
```

Source Code of AttributeStructure.java

```
import java.util.LinkedList;
import java.io.PrintStream;

public class AttributeStructure {
  String name;
  String type;

  public AttributeStructure(String atype, String aname) {
     type = atype;
     name = aname;
  }
  public int getLength() {
     if ((name!=null && name!="") && (type!=null && type!=""))
       return name.length() + type.length() + 1; //1 for space
     else if ((name!=null && name!="") && (type==null || type==""))
       return name.length() + 1;
     else if ((name==null || name=="") && (type!=null && type!=""))
       return type.length() + 1;
     else
      return 0;
  }
  public String getName() {
     return name;}
  public String getType() {
     return type;}
  public String toString() {
     String str = "attribute name:\t" + name + "\n";
     str += "attribute type:\t" + type + "\n";
     return str;}
  public void prettyPrint() {
     System.out.println(type + " " + name);}
  public void toXML(PrintStream ps) {
     ps.print("<attribute ");
     if (type != null && type != "") {
                    ps.print("type = \"");
                    ps.print(type);
                    ps.print("\" ");              }
                    if (name != null && name != "") {
                    ps.print("name = \"");
                    ps.print(name);
                    ps.print("\" ");}
     ps.println("/>");
}
  }}
```

Source Code for ClassDiagram.java

```
import java.awt.*;
import java.util.*;

public class ClassDiagram {
                        Point upperleft;
                        int height;
                        int width;
                        int level = -1;
                        ClassStructure cs;
                        LinkedList connections;

                        public ClassDiagram(Point upperleft, ClassStructure cs) {
                                this.upperleft = upperleft;
                                this.cs = cs;
                           width = cs.getMaxLength() * 7;
      height = cs.getAllSize() * 12;
      connections = new LinkedList();
                        }

                        public Point getUpperleft() {
                                return upperleft;    }
   public ClassStructure getClassStructure() {
                        return cs;}
                        public int getWidth() {
                                return width;}
                        public int getHeight() {
                                return height;}
                        public void addConnection(Connection conn) {
                                connections.add(conn);}
                        public String getClassName() {
                                return cs.getClassName();}
                        public int getLevel() {
                                return level;}
                        public void setLevel(int level) {
                                this.level = level;}
                        public void setPosition(int x, int y) {
                                upperleft = new Point(x, y);}
                        public boolean isInheritanceSubClass() {
                                ListIterator iter = connections.listIterator();
                                while (iter.hasNext()) {
                                        Connection conn = (Connection) iter.next();
                                if (conn.isInheritance() && conn.getFrom() == this)
                                                return true;}
                                return false;}
public boolean hasInheritance() {
                                ListIterator iter = connections.listIterator();
                                while (iter.hasNext()) {
                                        Connection conn = (Connection) iter.next();
                                        if (conn.isInheritance())
                                                return true;
```

```
                }
                return false;
        }

        public boolean isRootClass() {
                if (isInheritanceSubClass())
                        return false;
                if (hasInheritance())
                        return true;
                return false;
        }

        public void recursiveSetInheritanceLevel(int l) {
                this.level = l;
                ListIterator iter = connections.listIterator();
                while (iter.hasNext()) {
                        Connection conn = (Connection) iter.next();
                if (conn.isInheritance() && conn.getTo() == this) {
                                conn.getFrom().recursiveSetInheritanceLevel(l+1);
                        }
                }
        }

        public void setRelatLevel() {
                 ListIterator iter = connections.listIterator();
                 while(iter.hasNext()) {
                Connection conn = (Connection) iter.next();
    if((conn.getTo() == this) && (conn.getFrom().getLevel()!=-1))
       this.setLevel(conn.getFrom().getLevel());
    else if ((conn.getFrom() == this) && (conn.getTo().getLevel()!=-1))
       this.setLevel(conn.getTo().getLevel());
    else
                 setOtherLevel(0);
                        }
        }
public void setOtherLevel(int l) {
                this.level = l;
                ListIterator iter = connections.listIterator();
                while (iter.hasNext()) {
                        Connection conn = (Connection) iter.next();
                if (conn.isAggregation() && conn.getTo() == this)
                                conn.getFrom().setOtherLevel(l+1);
                else if (conn.isAssociation() && conn.getTo() == this)
                                conn.getFrom().setOtherLevel(l+1);
                else if (conn.isComposition() && conn.getTo() == this)
                                conn.getFrom().setOtherLevel(l+1);
                }
        }

        public void draw (Graphics g) {
          System.out.println("class: " + cs.getClassName());
          System.out.println("level: " + level);
   g.setColor(Color.black);
```

```
  int left = (int)upperleft.getX();
  int top = (int)upperleft.getY();
  g.drawRect(left, top, width, height);
  int line = 1;
  g.drawString(cs.getClassName(), left+2, top + (line++) * 12);
  int temp = top + (line++) * 12;
  LinkedList al = cs.getAttributeList();
  if (al != null) {
                        g.drawLine(left, temp, left + width, temp);

    ListIterator iter = al.listIterator();
    while (iter.hasNext()) {
      AttributeStructure as = (AttributeStructure) iter.next();

      if ((as.getName()!=null && as.getName()!="") &&
      (as.getType()!=null && as.getType()!=""))
              g.drawString(as.getName() + ": " +
              as.getType(), left + 10, top + (line++) * 12);

                  else if ((as.getName()!=null && as.getName()!="") &&
                  (as.getType()==null || as.getType()==""))
                   g.drawString(as.getName(), left+10, top+(line++)*12);

                    else if ((as.getName()==null || as.getName()=="") &&
                          (as.getType()!=null && as.getType()!=""))
                   g.drawString(as.getType(), left+10, top+(line++)*12);
    }
   }
  temp = top + (line++) * 12;
  LinkedList fl = cs.getFunctionList();
  if (fl != null) {
  g.drawLine(left, temp, left + width, temp);

    ListIterator iter = fl.listIterator();
    while (iter.hasNext()) {
      FunctionStructure fs = (FunctionStructure) iter.next();
      g.drawString(func2uml(fs), left + 10, top + (line++) * 12);
    }
  }
}

String func2uml(FunctionStructure fs) {
  StringBuffer sb = new StringBuffer();

  if (fs.getName()!=null && fs.getName()!="") {
              sb.append(fs.getName());
              sb.append("(");

  LinkedList al = fs.getArgumentList();
  if (al != null) {
    ListIterator iter = al.listIterator();
    while (iter.hasNext()) {
      ArgumentStructure as = (ArgumentStructure) iter.next();
```

```
        if ((as.getName()!=null && as.getName()!="")
                &&(as.getType()!=null && as.getType()!="")) {
                        sb.append(as.getName());
                        sb.append(":");
                        sb.append(as.getType());
        }
        if ((as.getName()==null || as.getName()=="")
                &&(as.getType()!=null && as.getType()!=""))
                        sb.append(as.getType());
        if ((as.getName()!=null && as.getName()!="")
                &&(as.getType()==null || as.getType()==""))
                        sb.append(as.getName());
        if (iter.nextIndex() != al.size())
           sb.append(", ");
      }
     }
     sb.append(")");
    }
    if (fs.getType() != null && fs.getType() != "") {
                        sb.append(": ");
                        sb.append(fs.getType());
    }
    return sb.toString();
  }
```

Source Code for ClassStructure.java

```
import java.util.LinkedList;
import java.util.ListIterator;
import java.io.PrintStream;
import javax.swing.JOptionPane;

public class ClassStructure {

  String className;
  String scope;
  LinkedList attributeList;
  LinkedList functionList;
  LinkedList relationshipList;
  LinkedList levelOne;

  public ClassStructure(String n) {
                        className = n;
                        attributeList = new LinkedList();
                        functionList = new LinkedList();
   relationshipList = new LinkedList();
  }

  public void setScope(String s) {
                        scope = s;
  }
```

```
public void addAttribute(AttributeStructure a) {
 if (attributeList == null)
   { attributeList = new LinkedList();}
 attributeList.add(a);
}

public void addFunction(FunctionStructure f) {
 if (functionList == null)
   { functionList = new LinkedList();}
 functionList.add(f);
}

public void addRelationship(RelationshipStructure r) {
 if (relationshipList == null)
   { relationshipList = new LinkedList(); }
 relationshipList.add(r);
}

public LinkedList getAttributeList() {
  return attributeList;
}

public LinkedList getFunctionList() {
  return functionList;
}

public LinkedList getRelationshipList() {
  return relationshipList;
}

public int getAllSize() {

 int aSize = 0;
 int fSize = 0;

                    if (attributeList != null)
   aSize = attributeList.size();
 else
   aSize = 0;

 if (functionList != null)
   fSize = functionList.size();
 else
   fSize = 0;

 return aSize + fSize + 5; //5 for class line, and each line for uml class
}

public int getMaxLength() {

  int maxLength = className.length()+3;
```

```
                    if (functionList != null) {
    ListIterator f = functionList.listIterator();
    while (f.hasNext()) {
                        FunctionStructure fs = (FunctionStructure)f.next();
                    if (fs.getLength() > maxLength)
                    maxLength = fs.getLength();
                     }
  }

  if (attributeList != null) {
                    ListIterator a = attributeList.listIterator();
                        while (a.hasNext()) {
                        AttributeStructure as = (AttributeStructure)a.next();
                    if (as.getLength() > maxLength)
      maxLength = as.getLength();
                    }
  }
  return maxLength;
}

public String getClassName() {
   return className;
}

public void setClassName(String name) {
                     className = name;
}

public String toString() {
                    String s = "class name:\t" + className + "\n";
                    s += "class scope:\t" + scope + "\n";
   return s;
}

public void replaceAttribute (String oldType, String oldName, String newType, String newName) {
   ListIterator a = attributeList.listIterator();

   while (a.hasNext())
   { AttributeStructure as = (AttributeStructure)a.next();
    if (as.getType().equals(oldType) && as.getName().equals(oldName))
    { a.remove();
      as = new AttributeStructure(newType, newName);
      a.add(as);
      return;
    }
   }
                        JOptionPane.showMessageDialog(null, "replace: No match attr found!",
    "Error", JOptionPane.PLAIN_MESSAGE);
}

public void replaceFunction (String oldType, String oldName, String newType, String newName) {
   ListIterator f = functionList.listIterator();
```

```
        while (f.hasNext())
        { FunctionStructure fs = (FunctionStructure)f.next();
          if (fs.getType().equals(oldType) && fs.getName().equals(oldName))
          { f.remove();
            fs = new FunctionStructure(newType, newName);
            f.add(fs);
            return;
          }
        }
JOptionPane.showMessageDialog(null, "replace: No match function found!",
        "Error", JOptionPane.PLAIN_MESSAGE);
    }

    public void replaceRelationship (String oldType, String oldDestination,
    String oldName, String newType, String newDestination, String newName) {
        ListIterator r = relationshipList.listIterator();

        while (r.hasNext())
        { RelationshipStructure rs = (RelationshipStructure)r.next();
          if (rs.getTypeToString().equals(oldType) &&
          rs.getDestination().equals(oldDestination) &&
          rs.getName().equals(oldName))
          { r.remove();
            rs = new RelationshipStructure(newType, newDestination, newName);
            r.add(rs);
            return;
          }
        }
JOptionPane.showMessageDialog(null, "replace: No match relationship found!",
        "Error", JOptionPane.PLAIN_MESSAGE);
    }

    public void deleteAttribute(String type, String name) {
        ListIterator a = attributeList.listIterator();

        while (a.hasNext())
        { AttributeStructure as = (AttributeStructure)a.next();
          if (as.getType().equals(type) && as.getName().equals(name)) {
            a.remove();
            return;
          }
        }
JOptionPane.showMessageDialog(null, "delete: No match attr found!",
        "Error", JOptionPane.PLAIN_MESSAGE);
    }


    public void deleteFunction(String type, String name) {
        ListIterator f = functionList.listIterator();

        while (f.hasNext())
        { FunctionStructure fs = (FunctionStructure)f.next();
          if (fs.getType().equals(type) && fs.getName().equals(name)) {
```

```
        f.remove();
        return;
        }}
JOptionPane.showMessageDialog(null, "delete: No match function found!",
        "Error", JOptionPane.PLAIN_MESSAGE); }
  public void deleteRelationship(String type, String destination, String name) {
      ListIterator r = relationshipList.listIterator();
      while (r.hasNext())
      { RelationshipStructure rs = (RelationshipStructure)r.next();
       if (rs.getTypeToString().equals(type) &&
       rs.getDestination().equals(destination) &&
       rs.getName().equals(name)) {
        r.remove();
        return;
       }      }
JOptionPane.showMessageDialog(null, "delete: No match relationship found!",
        "Error", JOptionPane.PLAIN_MESSAGE);
  }
  public void addAttribute(String type, String name) {
      AttributeStructure as = new AttributeStructure(type, name);
                        attributeList.add(as);   }
  public void addFunction(String type, String name) {
      FunctionStructure fs = new FunctionStructure(type, name);
                        functionList.add(fs);}
  public void addRelationship(String type, String destination, String name) {
      RelationshipStructure rs =
      new RelationshipStructure(type, destination, name);
      relationshipList.add(rs);  }
   public void toXML(PrintStream ps) {
     ps.print("<class className = \"");
     ps.print(className);
     ps.println("\">");
     if (attributeList != null) {
        ListIterator iter = attributeList.listIterator();
        while(iter.hasNext()) {
          AttributeStructure as = (AttributeStructure)iter.next();
          as.toXML(ps);
        }}
     if (functionList != null) {
        ListIterator iter = functionList.listIterator();
        while(iter.hasNext()) {
          FunctionStructure fs = (FunctionStructure)iter.next();
          fs.toXML(ps);
        }}
     if (relationshipList != null) {
        ListIterator iter = relationshipList.listIterator();
        while(iter.hasNext()) {
          RelationshipStructure rs = (RelationshipStructure)iter.next();
          rs.toXML(ps); }      }
     ps.println("</class>");
  }
```

Source Code for Connections.java

```
import java.awt.*;
import java.awt.Graphics;
import javax.swing.JOptionPane;
public class Connection {
                    int type;
                    ClassDiagram from;
                    ClassDiagram to;
                    String name
                    public Connection (int type, ClassDiagram from,
                    ClassDiagram to, String name) {
                            this.type = type;
                            this.from = from;
                            this.to = to;
                            this.name = name;
                    }
                    public ClassDiagram getFrom() {
                            return from;
                    }
                    public ClassDiagram getTo() {
                            return to;
                    }
                    public int getType() {
                            return type;
                    }
                    public String getName() {
                            return name;
                    }
                    public boolean isInheritance() {
                            return type == RelationshipStructure.INHERITANCE;
                    }
                    public boolean isAggregation() {
                            return type == RelationshipStructure.AGGREGATION;
                    }
                    public boolean isAssociation() {
                            return type == RelationshipStructure.ASSOCIATION;
                    }
                    public boolean isComposition() {
                            return type == RelationshipStructure.COMPOSITION;
                    }

public void drawRelString(String name, Point startPoint, Graphics g) {
                            //draw the name of the relationship
g.drawString(name, (int)startPoint.getX(), (int)startPoint.getY());
                    }
//the following are the pointers for different kind of relationships
                    public void drawToTopTriangle(int x, int y, Graphics g) {
      g.drawLine(x, y, x-6, y+8);
      g.drawLine(x, y, x+6, y+8);
      g.drawLine(x-6, y+8, x+6, y+8);
                    }
```

```
                    public void drawToButtomTriangle(int x, int y, Graphics g) {
    g.drawLine(x, y, x-6, y-8);
    g.drawLine(x, y, x+6, y-8);
    g.drawLine(x-6, y-8, x+6, y-8);
                    }
                    public void drawToLeftTriangle(int x, int y, Graphics g) {
    g.drawLine(x, y, x+8, y-6);
    g.drawLine(x, y, x+8, y+6);
    g.drawLine(x+8, y-6, x+8, y+6);
                    }
                    public void drawToRightTriangle(int x, int y, Graphics g) {
    g.drawLine(x, y, x-8, y-6);
    g.drawLine(x, y, x-8, y+6);
    g.drawLine(x-8, y-6, x-8, y+6);
                    }
          public void drawTopDiamend(int x, int y, Graphics g) {
    g.drawLine(x, y, x-6, y+8);
    g.drawLine(x, y, x+6, y+8);
    g.drawLine(x-6, y+8, x, y+16);
    g.drawLine(x+6, y+8, x, y+16);
  }
    public void drawTopFilledDiamend(int x, int y, Graphics g) {
                    int xcoords[] = {x, x-6, x, x+6};
                            int ycoords[] = {y, y+8, y+16, y+8};
                    g.fillPolygon(xcoords, ycoords, 4);
  }
  public void drawLeftDiamend(int x, int y, Graphics g) {
    g.drawLine(x, y, x-8, y+6);
    g.drawLine(x, y, x-8, y-6);
    g.drawLine(x-8, y-6, x-16, y);
    g.drawLine(x-8, y+6, x-16, y);
  }
public void drawLeftFilledDiamend(int x, int y, Graphics g) {
                    int xcoords[] = {x, x-8, x-16, x-8};
                            int ycoords[] = {y, y-6, y, y+6};
                    g.fillPolygon(xcoords, ycoords, 4);
  }

  public void drawToTopAngle(int x, int y, Graphics g) {
    g.drawLine(x, y, x-6, y+8);
    g.drawLine(x, y, x+6, y+8);
    g.drawLine(x, y, x, y+8);
  }

  public void drawToButtomAngle(int x, int y, Graphics g) {
    g.drawLine(x, y, x-6, y-8);
    g.drawLine(x, y, x+6, y-8);
    g.drawLine(x, y, x, y-8);
  }

                    public void drawToLeftAngle(int x, int y, Graphics g) {
    g.drawLine(x, y, x+8, y-6);
    g.drawLine(x, y, x+8, y+6);
```

```
        g.drawLine(x, y, x+8, y);
    }
    public void drawToRightAngle(int x, int y, Graphics g) {
        g.drawLine(x, y, x-8, y-6);
        g.drawLine(x, y, x-8, y+6);
        g.drawLine(x, y, x-8, y);
    }
blic void drawConnection(ClassDiagram from, ClassDiagram to,
String name, Graphics g) {

    //draw relationship lines among the diagrams

        from = getFrom();
        to = getTo();
        name = getName();

        Point startPoint = null;

        //define position that can use for relationships

        int fromWidth = (int)from.getWidth();
        int fromHeight = (int)from.getHeight();
        int toWidth = (int)to.getWidth();
        int toHeight = (int)to.getHeight();

        int fromLevel = (int)from.getLevel();
        int toLevel = (int)to.getLevel();

        int fromLeftX = (int)from.getUpperleft().getX();
        int fromCenterX = fromLeftX + from.getWidth()/2;
        int fromOneThirdX = fromLeftX + from.getWidth()/3;
        int fromTwoThirdX = fromLeftX + from.getWidth()/3*2;
        int fromRightX = fromLeftX + from.getWidth();

        int fromUpperY = (int)from.getUpperleft().getY();
        int fromCenterY = fromUpperY + from.getHeight()/2;
        int fromOneThirdY = fromUpperY + from.getHeight()/3;
        int fromTwoThirdY = fromUpperY + from.getHeight()/3*2;
        int fromLowerY = fromUpperY + from.getHeight();

        int toLeftX = (int)to.getUpperleft().getX();
        int toCenterX = toLeftX + to.getWidth()/2;
        int toOneThirdX = toLeftX + from.getWidth()/3;
        int toTwoThirdX = toLeftX + from.getWidth()/3*2;
        int toRightX = toLeftX + to.getWidth();

        int toUpperY = (int)to.getUpperleft().getY();
        int toCenterY = toUpperY + to.getHeight()/2;
        int toOneThirdY = toUpperY + to.getHeight()/3;
        int toTwoThirdY = toUpperY + to.getHeight()/3*2;
        int toLowerY = toUpperY + to.getHeight();
```

```
    //ASSOCIATION = 0
    //AGGREGATION = 1
    //INHERITANCE = 2
    //COMPOSITION = 3

/*************************************************
drawString(String str, int x, int y)
Draws the text given by the specified string,
using this graphics context's current font and color.
*************************************************/

                    if (type == 0) {
  // draw association relationship
    g.setColor(Color.green);
    if (fromUpperY == toUpperY) { //same level

     if (fromLeftX == toLeftX) {
                    // associate to itself
      startPoint = new Point(fromRightX, fromOneThirdY);
      g.drawLine(fromRightX, fromOneThirdY,
                                    fromRightX+10, fromOneThirdY);
      g.drawLine(fromRightX+10, fromOneThirdY,
                                    fromRightX+10, fromUpperY-20);
      g.drawLine(fromRightX+10, fromUpperY-20,
                                    fromTwoThirdX, fromUpperY-20);
      g.drawLine(fromTwoThirdX, fromUpperY-20,
                                    toTwoThirdX, toUpperY);
     }

     else if (fromLeftX == toRightX+80) {
                    // to is on the right side of from
      startPoint = new Point(fromLeftX-(name.length()*6), fromOneThirdY);
      g.drawLine(fromLeftX, fromOneThirdY,
            toRightX, toOneThirdY);
     }

     else if (toLeftX == fromRightX + 80) {
     // to is on the left side of from
                    startPoint = new Point(fromRightX, fromOneThirdY);
      g.drawLine(fromRightX, fromOneThirdY,
                                    toLeftX, toOneThirdY);
     }

     else if(toLeftX > fromRightX+80) {
     // to is not adjust from, there are objects between them
                    startPoint = new Point (fromRightX, fromOneThirdY-2);
                    g.drawLine(fromRightX, fromOneThirdY-2,
                                          fromRightX+10, fromOneThirdY-2);
      g.drawLine(fromRightX+10, fromOneThirdY-2,
                                    fromRightX+10, fromUpperY-10);
      g.drawLine(fromRightX+10, fromUpperY-10,
                                    toLeftX-10, toUpperY-10);
      g.drawLine(toLeftX-10, toUpperY-10,
```

```
                                    toLeftX-10, toOneThirdY-2);
        g.drawLine(toLeftX-10, toOneThirdY-2,
                                    toLeftX, toOneThirdY-2);
        }


        else if (toRightX < fromLeftX-80) {
        // to is not adjust from, there are objects between them
         startPoint = new Point(fromLeftX-(name.length()*6), fromOneThirdY-2);
         g.drawLine(fromLeftX, fromOneThirdY-2,
                                    fromLeftX-10,fromOneThirdY-2);
         g.drawLine(fromLeftX-10, fromOneThirdY-2,
                                    fromLeftX-10, fromUpperY-10);
         g.drawLine(fromLeftX-10, fromUpperY-10,
                                    toRightX+10, toUpperY-10);
         g.drawLine(toRightX+10, toUpperY-10,
                                    toRightX+10, toOneThirdY-2);
         g.drawLine(toRightX+10, toOneThirdY-2,
                                    toRightX,toOneThirdY-2);
        }
    }

    else if (fromUpperY != toUpperY) {

    if (fromLevel == toLevel+1) {
                        startPoint = new Point(fromLeftX+5, fromUpperY);
                        g.drawLine(fromLeftX+5, fromUpperY,
                                            toLeftX+5, toLowerY);
    }

    else if (fromLevel > toLevel+1) {
                        startPoint = new Point(fromLeftX+4, fromUpperY);
                        g.drawLine(fromLeftX+4, fromUpperY,
                        fromLeftX+4, fromUpperY-2);
                        g.drawLine(fromLeftX+4, fromUpperY-2,50, fromUpperY-2);
                        g.drawLine(50, fromUpperY-2, 50, toLowerY+8);
                        g.drawLine(50, toLowerY+8, toLeftX+4, toLowerY+8);
                        g.drawLine(toLeftX+4, toLowerY+8, toLeftX+4, toLowerY);
    }

    else if (fromLevel == toLevel-1) {
                        startPoint = new Point(fromLeftX+3,fromLowerY+16);
                        // 16 is the biggest size for relationship symble
                        g.drawLine(fromLeftX+3, fromLowerY,
                                toLeftX+3, toUpperY);
    }

    else if (fromLevel < toLevel-1) {
                        startPoint = new Point(fromLeftX+2, fromLowerY+16);
g.drawLine(fromLeftX+2, fromLowerY, fromLeftX+2, fromLowerY+2);
 g.drawLine(fromLeftX+2, fromLowerY+2, 55, fromLowerY+2);
                        g.drawLine(55, fromLowerY+2, 55, toUpperY-2);
                        g.drawLine(55, toUpperY-2, toLeftX+2, toUpperY-2);
```

```
    g.drawLine(toLeftX+2, toUpperY-2, toLeftX+2, toUpperY);
   }
  }

  if(name != "" && name != null) {
                    System.out.println("name : " + name);
                    drawRelString(name, startPoint, g);
  }
 }

 else if (type == 1) {

                    // draw aggregation relationship

 g.setColor(Color.red);

                    if (fromUpperY == toUpperY) {
  if (fromLeftX == toLeftX) {
  //point to self
                    startPoint = new Point(fromRightX, fromOneThirdY);
                    g.drawLine(fromRightX, fromOneThirdY,
                                fromRightX+10, fromOneThirdY);
                    g.drawLine(fromRightX+10, fromOneThirdY,
                                fromRightX+10, fromUpperY-20);
                    g.drawLine(fromRightX+10, fromUpperY-20,
                          fromTwoThirdX, fromUpperY-20);
    g.drawLine(fromTwoThirdX, fromUpperY-20,
                          toTwoThirdX, toUpperY-16);
    drawTopDiamend(toTwoThirdX, toUpperY-16, g);
   }

   else if (toLeftX == fromRightX+80) {
startPoint = new Point(toLeftX-16-(name.length()*6), toTwoThirdY);
                    g.drawLine(fromRightX, fromTwoThirdY,
          toLeftX-16, toTwoThirdY);
                    drawLeftDiamend(toLeftX, toTwoThirdY,g);
                     }

                     else if (fromLeftX == toRightX+80) {
    startPoint = new Point(toRightX+16, toTwoThirdY);
    g.drawLine(fromLeftX, fromTwoThirdY,
          toRightX+16, toTwoThirdY);
    drawLeftDiamend(toRightX+16, toTwoThirdY,g);
   }

                     else if (toLeftX > fromRightX + 80) {
 startPoint=new Point(toLeftX-16-(name.length()*6), toTwoThirdY);
                      g.drawLine(fromRightX, fromTwoThirdY,
                                fromRightX+4, fromTwoThirdY);
                          g.drawLine(fromRightX+4, fromTwoThirdY,
                                fromRightX+4, fromUpperY-5);
                          g.drawLine(fromRightX+4, fromUpperY-5,
                                toLeftX-16-4, toUpperY-5);
```

```
                        g.drawLine(toLeftX-16-4, toUpperY-5,
                                        toLeftX-16-4, toTwoThirdY);
                        g.drawLine(toLeftX-16-4, toTwoThirdY,
                                        toLeftX-16, toTwoThirdY);
                        drawLeftDiamend(toLeftX, toTwoThirdY,g);
                    }

    else if (toRightX < fromLeftX+80) {
     startPoint = new Point(toRightX+16, toTwoThirdY);
                    g.drawLine(fromLeftX, fromTwoThirdY,
                                        fromLeftX-4, fromTwoThirdY);
                        g.drawLine(fromLeftX-4, fromTwoThirdY,
                                        fromLeftX-4, fromUpperY-5);
                        g.drawLine(fromLeftX-4, fromUpperY-5,
                                        toRightX+16+5, toUpperY-5);
                        g.drawLine(toRightX+16+5, toUpperY-5,
                                        toRightX+16+5, toTwoThirdY);
                        g.drawLine(toRightX+16+5, toTwoThirdY,
                                        toRightX+16, toTwoThirdY);
     drawLeftDiamend(toRightX+16, toTwoThirdY,g);
     }

   }

   else if (fromUpperY != toUpperY) {

   if (fromLevel == toLevel+1) {
                    startPoint = new Point(toCenterX-6, toLowerY+16);
                    g.drawLine(fromCenterX-6, fromUpperY,
                                    toCenterX-6, toLowerY+16);
                    drawTopDiamend(toCenterX-6,toLowerY, g);
   }

                    else if (fromLevel > toLevel+1) {
                        startPoint = new Point(toCenterX-9, toLowerY+16);
                        g.drawLine(fromCenterX-9, fromUpperY,
                        fromCenterX-9, fromUpperY-2);
g.drawLine(fromCenterX-9, fromUpperY-2, 60, fromUpperY-2);
                        g.drawLine(60, fromUpperY-2, 60, toLowerY+16+5);
g.drawLine(60, toLowerY+16+5, toCenterX-9, toLowerY+16+5);
g.drawLine(toCenterX-9, toLowerY+16+5, toCenterX-9, toLowerY+5);
                        drawTopDiamend(toCenterX-9, toLowerY, g);
                    }

   else if (fromLevel == toLevel-1) {
                    startPoint = new Point(toCenterX+6, toUpperY-16);
                    g.drawLine(fromCenterX+6, fromLowerY,
                        toCenterX+6, toUpperY-16);
                    drawTopDiamend(toCenterX+6, toUpperY-16, g);
   }

   else if (fromLevel < toLevel-1) {
                    startPoint = new Point(toCenterX+9, toUpperY-16);
```

```
                    g.drawLine(fromCenterX+9, fromLowerY,
                                                fromCenterX+9, fromLowerY+2);
g.drawLine(fromCenterX+9, fromLowerY+2, 65, fromLowerY+2);
                    g.drawLine(65, fromLowerY+2, 65, toUpperY-16-5);
                    g.drawLine(65, toUpperY-16-5, toCenterX+9, toUpperY-16-5);
g.drawLine(toCenterX+9, toUpperY-16-5, toCenterX+9, toUpperY-16);
                    drawTopDiamend(toCenterX+9, toUpperY-16, g);
  }
 }

  if(name != "" && name != null) {
                    System.out.println("name : " + name);
                    drawRelString(name, startPoint, g);
  }
 }

 else if (type == 2) {
 //draw inheritance relationship

                    g.setColor(Color.blue);

                        if (fromUpperY == toUpperY) {
   if (fromLeftX == toLeftX) {
   //point to self
                    startPoint = new Point(fromRightX, fromOneThirdY+3);
                    g.drawLine(fromRightX, fromOneThirdY+3,
                                fromRightX+7, fromOneThirdY+3);
                    g.drawLine(fromRightX+7, fromOneThirdY+3,
                                fromRightX+7, fromUpperY-18);
                    g.drawLine(fromRightX+7, fromUpperY-18,
                          fromOneThirdX, fromUpperY-18);
    g.drawLine(fromOneThirdX, fromUpperY-18,
                                toOneThirdX, toUpperY-8);
    drawToButtomTriangle(toCenterX, toUpperY, g);
   }

   else if (toLeftX == fromRightX+80) {
                    startPoint = new Point(fromRightX, fromTwoThirdY+3);
                    g.drawLine(fromRightX, fromTwoThirdY+3,
         toLeftX-8, toTwoThirdY+3);
                    drawToRightTriangle(toLeftX, toTwoThirdY+3,g);
                     }

                      else if (fromLeftX == toRightX+80) {
    startPoint = new Point(fromLeftX, fromTwoThirdY+3);
    g.drawLine(fromLeftX, fromTwoThirdY+3,
          toRightX+8, toTwoThirdY+3);
    drawToLeftTriangle(toRightX, toTwoThirdY+3,g);
   }

else if ((toLeftX > fromRightX + 80)||(toRightX < fromLeftX+80)) {
                      startPoint=new Point(fromCenterX, fromUpperY);
                      g.drawLine(fromCenterX, fromUpperY,
```

```
                                        fromCenterX, fromUpperY-12);
                        g.drawLine(fromCenterX, fromUpperY-12,
                                        toCenterX, toUpperY-12);
                        g.drawLine(toCenterX, toUpperY-12,
                                        toCenterX, toUpperY-8);
                        drawToButtomTriangle(toCenterX, toUpperY,g);
                    }
}

else if (fromUpperY > toUpperY) {
                startPoint = new Point(fromCenterX, fromUpperY);
                g.drawLine(fromCenterX, fromUpperY,
                                                toCenterX, toLowerY+15);
                        g.drawLine(toCenterX, toLowerY+15,
                                                toCenterX, toLowerY+8);
                drawToTopTriangle(toCenterX, toLowerY, g);
}

else if (fromUpperY < toUpperY) {

                startPoint = new Point(fromCenterX, fromLowerY);
                g.drawLine(fromCenterY, fromLowerY,
                                        toCenterX, toUpperY-15);
  g.drawLine(toCenterX, toUpperY-15,
                                        toCenterX, toUpperY-8);
                drawToButtomTriangle(toCenterX, toUpperY, g);
}

if(name != "" && name != null) {
                System.out.println("name : " + name);
                drawRelString(name, startPoint, g);
}
}

  else if (type == 3) {

                // draw composition relationship

g.setColor(Color.darkGray);

                if (fromUpperY == toUpperY) {
 if (fromLeftX == toLeftX) {
 //point to self
                startPoint = new Point(fromRightX, fromOneThirdY);
                g.drawLine(fromRightX, fromOneThirdY,
                                        fromRightX+10, fromOneThirdY);
                g.drawLine(fromRightX+10, fromOneThirdY,
                                        fromRightX+10, fromUpperY-20);
                g.drawLine(fromRightX+10, fromUpperY-20,
                        fromTwoThirdX, fromUpperY-20);
   g.drawLine(fromTwoThirdX, fromUpperY-20,
                        toTwoThirdX, toUpperY-16);
   drawTopFilledDiamend(toTwoThirdX, toUpperY-16, g);
```

```
    }

    else if (toLeftX == fromRightX+80) {
startPoint = new Point(toLeftX-16-(name.length()*6), toTwoThirdY);
                    g.drawLine(fromRightX, fromTwoThirdY,
          toLeftX-16, toTwoThirdY);
                    drawLeftFilledDiamend(toLeftX, toTwoThirdY,g);
                     }

                     else if (toLeftX > fromRightX + 80) {
  startPoint=new Point(toLeftX-16-(name.length()*6), toTwoThirdY);
                      g.drawLine(fromRightX, fromTwoThirdY,
                                        fromRightX+4, fromTwoThirdY);
                          g.drawLine(fromRightX+4, fromTwoThirdY,
                                        fromRightX+4, fromUpperY-5);
                          g.drawLine(fromRightX+4, fromUpperY-5,
                                        toLeftX-5-16, toUpperY-5);
                          g.drawLine(toLeftX-5-16, toUpperY-5,
                                        toLeftX-5-16, toTwoThirdY);
                          g.drawLine(toLeftX-5-16, toTwoThirdY,
                                        toLeftX-16, toTwoThirdY);
                          drawLeftFilledDiamend(toLeftX, toTwoThirdY,g);
                     }

                     else if (fromLeftX == toRightX+80) {
     startPoint = new Point(toRightX+16, toTwoThirdY);
     g.drawLine(fromLeftX, fromTwoThirdY,
           toRightX+16, toTwoThirdY);
     drawLeftFilledDiamend(toRightX+16, toTwoThirdY,g);
    }

    else if (toRightX < fromLeftX+80) {
     startPoint = new Point(toRightX+16, toTwoThirdY);
                    g.drawLine(fromLeftX, fromTwoThirdY,
                                        fromLeftX-4, fromTwoThirdY);
                          g.drawLine(fromLeftX-4, fromTwoThirdY,
                                        fromLeftX-4, fromUpperY-5);
                          g.drawLine(fromLeftX-4, fromUpperY-5,
                                        toRightX+5+16, toUpperY-5);
                          g.drawLine(toRightX+5+16, toUpperY-5,
                                        toRightX+5+16, toTwoThirdY);
                          g.drawLine(toRightX+5+16, toTwoThirdY,
                                        toRightX+16, toTwoThirdY);
     drawLeftFilledDiamend(toRightX+16, toTwoThirdY,g);
    }

  }

  else if (fromUpperY != toUpperY) {

  if (fromLevel == toLevel+1) {
                    startPoint = new Point(toRightX-2, toLowerY+16);
                    g.drawLine(fromRightX-2, fromUpperY,
```

```
                                        toRightX-2, toLowerY+16);
                    drawTopFilledDiamend(toRightX-2,toLowerY, g);
  }

                    else if (fromLevel > toLevel+1) {
                        startPoint = new Point(toRightX-5, toLowerY+16);
                        g.drawLine(fromRightX-5, fromUpperY,
                        fromRightX-5, fromUpperY-5);
g.drawLine(fromRightX-5, fromUpperY-5, 70, fromUpperY-5);
                        g.drawLine(70, fromUpperY-5, 70, toLowerY+8+16);
g.drawLine(70, toLowerY+8+16, toRightX-5, toLowerY+8+16);
g.drawLine(toRightX-5, toLowerY+8+16, toRightX-5, toLowerY+16);
                        drawTopFilledDiamend(toRightX-5, toLowerY, g);
                    }

  else if (fromLevel == toLevel-1) {
                    startPoint = new Point(toRightX, toUpperY-16);
                    g.drawLine(fromRightX, fromLowerY,
                        toRightX, toUpperY-16);
                    drawTopDiamend(toRightX, toUpperY-16, g);
  }

  else if (fromLevel < toLevel-1) {
                    startPoint = new Point(toRightX-3, toUpperY-16);
                    g.drawLine(fromRightX-3, fromLowerY,
                                            fromRightX-3, fromLowerY+5);
g.drawLine(fromRightX-3, fromLowerY+5, 75, fromLowerY+5);
                    g.drawLine(75, fromLowerY+5, 75, toUpperY-8-16);
                    g.drawLine(75, toUpperY-8-16, toRightX-3, toUpperY-8-16);
 g.drawLine(toRightX-3, toUpperY-8-16, toRightX-3, toUpperY-16);
                    drawTopFilledDiamend(toRightX-3, toUpperY-16, g);
  }
  }

  if(name != "" && name != null) {
                    System.out.println("name : " + name);
                    drawRelString(name, startPoint, g);
  }
 }

 else
  JOptionPane.showMessageDialog(null,
  "The relationship type should be inheritance, aggregation or association!",
    "Error", JOptionPane.PLAIN_MESSAGE);
}
```

Source Code of FunctionStructure.java

```
import java.util.LinkedList;
import java.util.ListIterator;
import java.io.PrintStream;
```

```
public class FunctionStructure {
  String name;
  String type;
  LinkedList argumentList;

  public FunctionStructure(String ftype, String fname) {
     type = ftype;
     name = fname;
  }
  public int getLength() {
     int length = 0;
     if (argumentList != null) {
                    ListIterator fl = argumentList.listIterator();
                    while (fl.hasNext())
                          { ArgumentStructure ag = (ArgumentStructure)fl.next();
                          length += ag.getLength() + 1; //1 for each comma
                    }     }
     if ((name!=null && name!="") && (type!=null && type!=""))
return name.length() + type.length() + length + 3; //3 for "(", ")" and space
else if ((name!=null && name!="") && (type==null || type==""))
                                    return name.length() + length + 3;
else if ((name==null || name=="") && (type!=null && type!=""))
                                    return type.length() + length + 3;
                              else
                              return 0;}
  public void addArgument(ArgumentStructure a) {
     if (argumentList == null) {
                         argumentList = new LinkedList();}
     argumentList.add(a);
  }
  public String getName() {
     return name;
  }
  public String getType() {
     return type;
  }
  public LinkedList getArgumentList() {
     return argumentList;
  }  public void addNewArgument(String type, String name, ArgumentStructure as) {
     as = new ArgumentStructure(name ,type);
                    argumentList.add(as);
  }
  public void replaceArgument (String oldType, String oldName, String newType, String newName) {
     ListIterator a = argumentList.listIterator();
     while (a.hasNext())
     { ArgumentStructure as = (ArgumentStructure)a.next();
      if (as.getType().equals(oldType) && as.getName().equals(oldName))
      { a.remove();
       as = new ArgumentStructure(newName, newType);
       a.add(as);
      }
      else
       System.out.println("No match found!");
```

```
    }
}
public String toString() {
    String s = "function name:\t" + name + "\n";
    s += "function type:\t" + type + "\n";
    return s;
}
public void prettyPrint() {
    System.out.print(type + " " + name);

    System.out.print("(");
    if (argumentList != null) {
    ListIterator li = argumentList.listIterator();
      while (li.hasNext()) {
        ArgumentStructure as = (ArgumentStructure) li.next();
        as.prettyPrint();
        if (li.nextIndex() != argumentList.size())
          System.out.print(", ");
      }
    }
    System.out.println(")");
}
public void toXML(PrintStream ps) {
   ps.print("<function ");
   if (type!=null && type!="") {
                        ps.print("type = \"");
                        ps.print(type);
                        ps.print("\" ");
   }
   if (name!=null && name !="") {
                        ps.print("name = \"");
                        ps.print(name);
                        ps.print("\" ");
   }
   ps.println(">");
   if (argumentList != null) {
      ListIterator iter = argumentList.listIterator();
      while(iter.hasNext()) {
        ArgumentStructure as = (ArgumentStructure)iter.next();
        as.toXML(ps);
      }
   }
   ps.println("</function>");
}}
```

Source Code of Panels.java

```
import java.awt.*;
import java.awt.event.*;
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.*;
import javax.swing.JOptionPane;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JMenuBar;
import javax.swing.border.*;
import javax.swing.JDialog;
import javax.swing.JComboBox;
import java.util.*;
import java.util.LinkedList;
import java.util.ListIterator;
import java.util.Hashtable;
import java.util.Iterator;
class WelcomePanel extends JDialog {
                    public WelcomePanel() {
                            JPanel TitlePanel = new JPanel();
TitlePanel.add(new JLabel("<html><center><h1><b>UML Class" +
" Diagram Translator (UMLCDT) 1.0</b></h1></center></html>"));
                            Container contentPane = getContentPane();
                            contentPane.add(TitlePanel, BorderLayout.NORTH);

                            JPanel welPanel = new JPanel();
welPanel.setBorder(new TitledBorder(new LineBorder(Color.gray, 12),
                            "welcome"));
                            welPanel.setLayout(new GridLayout(1, 1));
                            //welPanel.add(new JLabel("Acknowledgment"));
                            JTextArea j = new JTextArea("Kaiyan Li"+"\n" +
                            "Department of Computer Science"+"\n" +
                    "California State University, San Bernardino"+"\n" +"\n"+
"This project, UMLCDT, is a software program designed to work"
                            +" in the Online Design Pattern Library (DPL)" +
                            " system, which is an" +
" implementation of a tool in Applying Designs and Patterns (ADAP)" +
" Project. UMLCDT Project is designed to provide a useful tool that" +
                    " helps designers to use design patterns for software design." +
                            " This project stores all the design patterns from" +
" Gamma's book, including Creational Design Patterns, Structural" +
                            " Design Patterns, and Behavioral Design Patterns."+
                            " The patterns structures are written in XML." +
" UMLCDT translates from class diagram to XML files and vice versa."+
" Any modifications to the UMLCDT Project is prohibit unless being" +
```

```
                    " approved by Computer Science Department of California State" +
                "University of San Bernardino and the designer herself." +
" UMLCDT Project conditions by clicking on the \"Enter\"" +
" button to use any of the UMLCDT services (including, without " +
"limitation, the XML file retrieving, UML class diagram generating, " +
"Modifying abd Saving as XML), you agree to be bound by the " +
"conditions. The acceptable use guidelines for the UMLCDT Project is " +
                "available in UMLCDT Document. " +
                "If you do not agree to be bound by the rules, you should" +
" discontinue by clicking on the \"Cancel\" button. Your UMLCDT" +
" will not be activated and will be terminated. " + "\n" + "\n" +
"© Copyright 2002 UMLCDT VERSION 1.0, CSCI, CSUSB",20, 2);
                        j.setEditable(false);
                        j.setLineWrap(true);
                        j.setWrapStyleWord(true);
                        welPanel.add(j);
                        welPanel.setBackground(Color.white);

                        contentPane.add(welPanel, BorderLayout.CENTER);

                        JPanel buttonPanel = new JPanel();
                        buttonPanel.setBackground(Color.gray);

                        JButton okButton = new JButton("Enter");
                        okButton.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent event)
                        {       UMLDiagram u = new UMLDiagram();
                                        u.setVisible(true);
                        u.setTitle("UML Class Diagram Translator 1.0");
                                        u.pack();
                                        setVisible(false);
                                }
                        });
                        JButton cancleButton = new JButton("Cancel");
                        cancleButton.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent event)
                                {       setVisible(false);}
                        });
                        buttonPanel.add(okButton);
                        buttonPanel.add(cancleButton);
                        contentPane.add(buttonPanel, BorderLayout.SOUTH);

                        setSize(700, 500);
                }
}
class AboutPanel extends JDialog {
                public AboutPanel() {
                        JPanel TitlePanel = new JPanel();
                        Container contentPane = getContentPane();
                        JPanel aboutPanel = new JPanel();
                        aboutPanel.setLayout(new GridLayout(1, 1));
                JTextArea j = new JTextArea("FILE MENU" + "\n" + "\n"
                        + "New - Open a new panel" + "\n" + "\n"
```

```
+ "Open - Allow user to choose a file from the local directory" + "\n" + "\n"
+ "Refresh - Refresh the panel after modification" + "\n"+ "\n"          + "Save - Allow user to save the
current structure under er " +"input file name" + "\n" + "\n"
                + "Clear - Clear the current panel" + "\n" + "\n"
                + "Exit - Exit from UMLCDT system" + "\n" + "\n" + "\n"
                    + "EDIT MENU" + "\n" + "\n"      + "Delete Class - Delete a specific class
by inputing " +"the exact class name" + "\n" + "\n"
+ "Delete Attribute - Delete an attribute from a specific class by " +
"inputing the class name, attribute type, attribute name" + "\n" + "\n"
+ "Delete Function - Delete a function from a specific class by " +
"inputing the class name, function type, function name" + "\n" + "\n"
+ "Delete Relationship - Delete a relationship from a specific class by " +
"inputing the class name, relationship type, destination and name" + "\n" + "\n"
+ "Add Class - Add a new class by inputing a new class name" + "\n" + "\n"
+ "Add Attribute - Add a new attribute to a specific class by " +
"inputing the class name, attribute type, attribute name" + "\n" + "\n"
+ "Add Function - Add a new function to a specific class by " +
"inputing the class name, function type, function name" + "\n" + "\n"
+ "Add Relationship - Add a new relationship to a specific class by " +
"inputing the class name, relationship type, destination and name" + "\n" + "\n"
+ "Change Class - Inform user to go to the XML code to change " +
"a class name" + "\n" + "\n"
+ "Change Attribute - Changed an exist attribute to a new one by " +
"inputing the attribute type, name of the exist one and the attribute " +
"type and name of new one" + "\n" + "\n"
+ "Change Function - Changed an exist function to a new one by " +
"inputing the function type, name of the exist one and the attribute " +
"type and name of new one" + "\n" + "\n"
+ "Change Relationship - Changed an exist relationship to a new one by " +
"inputing the relationship type, destination and name of the exist one and the attribute " +"type,
destination and name of new one");
                j.setEditable(false);
                    j.setLineWrap(true);
                    j.setWrapStyleWord(true);
                    aboutPanel.add(j);
                    JScrollPane scrollbar = new JScrollPane( j,
                    JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
                JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED );
                    aboutPanel.setBackground(Color.white);
                    contentPane.add(aboutPanel, BorderLayout.CENTER);
                    contentPane.add(scrollbar);
                    JPanel buttonPanel = new JPanel();
                    buttonPanel.setBackground(Color.gray);
                    JButton cancleButton = new JButton("Cancel");
                    cancleButton.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent event)
                            {       setVisible(false);}
                    });
                    buttonPanel.add(cancleButton);
                    contentPane.add(buttonPanel, BorderLayout.SOUTH);
                    setSize(400, 500);
                }}
```

Source Code of RelationshipStructure.java

```
import java.util.LinkedList;
import java.awt.Point;
import java.awt.*;
import java.io.PrintStream;
import javax.swing.JOptionPane;
public class RelationshipStructure {
  int type;
  String destination;
  String name;
  static final int ASSOCIATION = 0;
  static final int AGGREGATION = 1;
  static final int INHERITANCE = 2;
  static final int COMPOSITION = 3;
  static final String [] types = { "association", "aggregation", "inheritance", "composition" };
  public RelationshipStructure(String rtype, String rdestination, String rname) {
                        destination = rdestination;
                        name = rname;
                            for (int i = 0; i < types.length; i++) {
                                    if (types[i].equalsIgnoreCase(rtype)) {
                            type = i;
                            return;      }
     JOptionPane.showMessageDialog(null,
     "the relastionship type should be inheritance, aggregation or association!",
     "Error", JOptionPane.PLAIN_MESSAGE);   }
  public String getDestination() {
     return destination;   }
  public void setDestination(String dis) {
                            destination = dis;   }
                              public int getType() {
                    return type;}
    public String getName() {
                    return name;}
    public String getTypeToString() {
      return types[type];}
  public String toString() {
     String str = "type:\t" + types[type] + "\n";
     str += "destination:\t" + destination + "\n";
     str += "name:\t" + name + "\n";
     return str;   }
  public void toXML(PrintStream ps) {
     ps.print("<relationship type = \"");
     ps.print(types[type]);
     ps.print("\" destination = \"");
     ps.print(destination);
     if (name != null && name != "") {
     ps.print("\" name = \"");
     ps.print(name);
  }      ps.println("\"/>");
  }}
```

Source Code of UMLDiagram.java

```
import org.apache.xerces.parsers.SAXParser;
import org.xml.sax.InputSource;
import java.awt.*;
import java.awt.event.*;
import java.awt.Color;
import java.awt.Container;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.*;
import java.io.FileReader;
import javax.swing.*;
import javax.swing.filechooser.*;
import javax.swing.JOptionPane;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JMenuBar;
import javax.swing.border.*;
import javax.swing.JDialog;
import java.util.*;
import java.util.LinkedList;
import java.util.ListIterator;
import java.util.Hashtable;
import java.util.Iterator;

class UMLDiagram extends JFrame {

  private JMenuItem newMenuItem;
  private JMenuItem openMenuItem;
  private JMenuItem refreshMenuItem;
  private JMenuItem saveMenuItem;
  private JMenuItem saveAsMenuItem;
  private JMenuItem exitMenuItem;
  private JMenuItem aboutMenuItem;

  //private JMenuItem mergeMenuItem;
  private JMenuItem deleteClassMenuItem;
  private JMenuItem addClassMenuItem;
  private JMenuItem changeClassMenuItem;

  private JMenuItem deleteAttrMenuItem;
  private JMenuItem addAttrMenuItem;
  private JMenuItem changeAttrMenuItem;
```

```
private JMenuItem deleteFuncMenuItem;
private JMenuItem addFuncMenuItem;
private JMenuItem changeFuncMenuItem;

private JMenuItem deleteRelationMenuItem;
private JMenuItem addRelationMenuItem;
private JMenuItem changeRelationMenuItem;

private JMenuItem moveItem;
private JMenuItem separator;
private JMenuItem clearMenuItem;
private UMLPanel panel;

public UMLDiagram() {

addWindowListener(new WindowAdapter() {
public void windowClosing(WindowEvent e) {
  System.exit(0);
   }
});

//add drawing panel to content panel


panel = new UMLPanel();
Color background = new Color(215, 193, 255);
panel.setBackground(background);

                    JScrollPane scroller = new JScrollPane(panel);
                    scroller.setPreferredSize(new Dimension(700, 500));
Container contentPane = getContentPane();
                    contentPane.add("Center", scroller);

JMenuBar menuBar = new JMenuBar();
setJMenuBar(menuBar);

MenuListener listener = new MenuListener();

JMenu fileMenu = new JMenu("File");
menuBar.add(fileMenu);

newMenuItem = new JMenuItem("New");
fileMenu.add(newMenuItem);
newMenuItem.addActionListener(listener);

openMenuItem = new JMenuItem("Open");
fileMenu.add(openMenuItem);
openMenuItem.addActionListener(listener);

refreshMenuItem = new JMenuItem("Refresh");
fileMenu.add(refreshMenuItem);
refreshMenuItem.addActionListener(listener);
```

```
saveAsMenuItem = new JMenuItem("Save As");
fileMenu.add(saveAsMenuItem);
saveAsMenuItem.addActionListener(listener);

clearMenuItem = new JMenuItem("Clear");
fileMenu.add(clearMenuItem);
clearMenuItem.addActionListener(listener);

exitMenuItem = new JMenuItem("Exit");
fileMenu.add(exitMenuItem);
exitMenuItem.addActionListener(listener);

JMenu editMenu = new JMenu("Edit");
menuBar.add(editMenu);

JMenuItem deleteMenu = new JMenu("Delete");
editMenu.add(deleteMenu);

deleteClassMenuItem = new JMenuItem("Class");
deleteMenu.add(deleteClassMenuItem);
deleteClassMenuItem.addActionListener(listener);

deleteAttrMenuItem = new JMenuItem("Attribute");
deleteMenu.add(deleteAttrMenuItem);
deleteAttrMenuItem.addActionListener(listener);

deleteFuncMenuItem = new JMenuItem("Function");
deleteMenu.add(deleteFuncMenuItem);
deleteFuncMenuItem.addActionListener(listener);

deleteRelationMenuItem = new JMenuItem("Relationship");
deleteMenu.add(deleteRelationMenuItem);
deleteRelationMenuItem.addActionListener(listener);

JMenuItem addMenu = new JMenu("Add");
editMenu.add(addMenu);

addClassMenuItem = new JMenuItem("Class");
addMenu.add(addClassMenuItem);
addClassMenuItem.addActionListener(listener);

addAttrMenuItem = new JMenuItem("Attribute");
addMenu.add(addAttrMenuItem);
addAttrMenuItem.addActionListener(listener);

addFuncMenuItem = new JMenuItem("Function");
addMenu.add(addFuncMenuItem);
addFuncMenuItem.addActionListener(listener);

addRelationMenuItem = new JMenuItem("Relationship");
addMenu.add(addRelationMenuItem);
addRelationMenuItem.addActionListener(listener);
```

```
JMenuItem changeMenu = new JMenu("Change");
editMenu.add(changeMenu);

changeClassMenuItem = new JMenuItem("Class");
changeMenu.add(changeClassMenuItem);
changeClassMenuItem.addActionListener(listener);

changeAttrMenuItem = new JMenuItem("Attribute");
changeMenu.add(changeAttrMenuItem);
changeAttrMenuItem.addActionListener(listener);

changeFuncMenuItem = new JMenuItem("Function");
changeMenu.add(changeFuncMenuItem);
changeFuncMenuItem.addActionListener(listener);

changeRelationMenuItem = new JMenuItem("Relationship");
changeMenu.add(changeRelationMenuItem);
changeRelationMenuItem.addActionListener(listener);

JMenu helpMenu = new JMenu("About");
menuBar.add(helpMenu);

aboutMenuItem = new JMenuItem("About UMLCreator");
helpMenu.add(aboutMenuItem);
aboutMenuItem.addActionListener(listener);
}

private class MenuListener implements ActionListener {
public void actionPerformed(ActionEvent event) {
  //find the menu that was selected

  Object source = event.getSource();

  if (source == exitMenuItem)
    System.exit(0);

  else if (source == openMenuItem) {
    panel.init();
    panel.repaint();
  }

  else if (source == refreshMenuItem) {
                        panel.refreshDiagram();
                        panel.repaint();
  }

  else if (source == newMenuItem) {
                        panel.clearPanel();
                        panel.repaint();
  }

  else if (source == saveMenuItem)
                    panel.save();
```

```
else if (source == saveAsMenuItem)
                    panel.saveAs();

else if (source == clearMenuItem) {
                    panel.clearPanel();
                    panel.repaint();
}

else if (source == aboutMenuItem) {
                     JDialog ap = new AboutPanel();
                                ap.setVisible(true);
                                ap.setTitle("About UMLCDT 1.0");
                          }

else if (source == deleteClassMenuItem) {
                          panel.deleteClass();
                          panel.save();
                          panel.refreshDiagram();
                          panel.repaint();
}

else if (source == addClassMenuItem) {
                          panel.addClass();
                          panel.save();
                          panel.refreshDiagram();
                          panel.repaint();
}

else if (source == changeClassMenuItem) {

                          panel.changeClass();
                          panel.save();
}

else if (source == deleteAttrMenuItem) {
                                panel.deleteAttr();
                                panel.save();
                                panel.refreshDiagram();
                          panel.repaint();
                    }

else if (source == addAttrMenuItem) {
                          panel.addAttr();
                          panel.save();
                          panel.refreshDiagram();
                          panel.repaint();
}

else if (source == changeAttrMenuItem) {
                          panel.changeAttr();
                          panel.save();
                          panel.refreshDiagram();
```

```
                    panel.repaint();
                }

    else if (source == deleteFuncMenuItem) {
                    panel.deleteFunc();
                    panel.save();
                    panel.refreshDiagram();
                    panel.repaint();
                }

    else if (source == addFuncMenuItem) {
                    panel.addFunc();
                    panel.save();
                    panel.refreshDiagram();
                    panel.repaint();
                }

    else if (source == changeFuncMenuItem) {
                    panel.changeFunc();
                    panel.save();
                    panel.refreshDiagram();
                    panel.repaint();
                }

    else if (source == deleteRelationMenuItem) {
                    panel.deleteRelation();
                    panel.save();
                    panel.refreshDiagram();
                    panel.repaint();
                }

    else if (source == addRelationMenuItem) {
                    panel.addRelation();
                    panel.save();
                    panel.refreshDiagram();
                    panel.repaint();
                }

    else if (source == changeRelationMenuItem) {
                    panel.changeRelation();
                    panel.save();
                    panel.refreshDiagram();
                    panel.repaint();
                }

   }
  }

  public static void main(String args[]) {
   JDialog wp = new WelcomePanel();
                wp.setVisible(true);
  }
}
```

```
class UMLPanel extends JPanel
{
  LinkedList classes = null;
  Hashtable diagrams = new Hashtable();
  int top = 0;
  int left = 0;
  String fileName;
  Dimension size = new Dimension (0,0);

  public UMLPanel() {}

  public void paint(Graphics g) {
                        super.paint(g);

                        if (diagrams.isEmpty())
                              return;

                        Iterator iter = diagrams.values().iterator();

                        while (iter.hasNext()) {
                              ClassDiagram dia = (ClassDiagram) iter.next();
                              dia.draw(g);
                        }

                        drawRelation(g);
  }

                       public void addClass() {
     String className;
     className = JOptionPane.showInputDialog("Enter class name:");

     if (classes == null) {
                          ClassStructure cs = new ClassStructure(className);
                              classes = new LinkedList();
                                    classes.add(cs);
                              }
                              else {
                                    ListIterator c = classes.listIterator();
                                while (c.hasNext()) {
                          ClassStructure cs = (ClassStructure)c.next();
                          if (cs.getClassName().equals(className)) {
                           JOptionPane.showMessageDialog(null,
                                                "This class name already exists!",
                            "Error", JOptionPane.PLAIN_MESSAGE);
                            return;
                            }
                          }
                          ClassStructure cs = new ClassStructure(className);
                                    classes.add(cs);
                              }
  }
```

```
                public boolean isParent(String classname) {
                        ListIterator c = classes.listIterator();
                        while (c.hasNext()) {
    ClassStructure cs = (ClassStructure)c.next();
    LinkedList rl = cs.getRelationshipList();
                                if (rl != null && !rl.isEmpty()) {
                                        ListIterator rlIter = rl.listIterator();
                                        while (rlIter.hasNext()) {
                                                RelationshipStructure rs =
                                        (RelationshipStructure) rlIter.next();
                                                if(rs.getDestination().equals(classname))
                                                return true; //is a parent
                                        }
                                }
                        }
                        return false; //not a parent
                }

public void deleteClass() {
  ListIterator c = classes.listIterator();

   String className;

   className = JOptionPane.showInputDialog("Enter class name:");

   while (c.hasNext()) {
      ClassStructure cs = (ClassStructure)c.next();
      if (cs.getClassName().equals(className)) {
                if (isParent(className)==true) {
                                        JOptionPane.showMessageDialog(null,
                                        "This class is a parent class and cannot be deleted!",
                        "Error", JOptionPane.PLAIN_MESSAGE);
                        return;
                        }
                                else {
                                 c.remove();
                                 return;
                                }
                        }
   }
   JOptionPane.showMessageDialog(null, "delete class: No match class found!",
   "Error", JOptionPane.PLAIN_MESSAGE);
}

public void changeClass() {
                JOptionPane.showMessageDialog(null,
   "To change a class name, go to the xml code.",
   "Error", JOptionPane.PLAIN_MESSAGE);
}

public void deleteAttr() {
  ListIterator c = classes.listIterator();
```

```
    String className, attrType, attrName;

    className = JOptionPane.showInputDialog("Enter class name:");
                        attrType = JOptionPane.showInputDialog("Enter attr. type: ");

    attrName = JOptionPane.showInputDialog("Enter attr. name: ");

    while (c.hasNext()) {
       ClassStructure cs = (ClassStructure)c.next();
       if (cs.getClassName().equals(className)) {
                              cs.deleteAttribute(attrType,attrName);
                              return;
                    }
    }
    JOptionPane.showMessageDialog(null, "delete attr: No match class found!",
    "Error", JOptionPane.PLAIN_MESSAGE);
}

public void deleteFunc() {
  ListIterator c = classes.listIterator();
    String className, funcType, funcName;

    className = JOptionPane.showInputDialog("Enter class name:");
                        funcType = JOptionPane.showInputDialog("Enter func. type: ");

    funcName = JOptionPane.showInputDialog("Enter func. name: ");

    while (c.hasNext()) {
       ClassStructure cs = (ClassStructure)c.next();
       if (cs.getClassName().equals(className)) {
                              cs.deleteFunction(funcType,funcName);
                              return;
       }
    }
    JOptionPane.showMessageDialog(null, "delete func: No match class found!",
    "Error", JOptionPane.PLAIN_MESSAGE);
}

public void deleteRelation() {
  ListIterator c = classes.listIterator();

    String className, relaType, relaDestination, relaName;

    className = JOptionPane.showInputDialog("Enter class name:");
                        relaType = JOptionPane.showInputDialog("Enter relation type: " +
                        "(choose from association, aggregation, composition and inheritance)");

    relaDestination =
    JOptionPane.showInputDialog("Enter relation destination class: ");
                        relaName = JOptionPane.showInputDialog("Enter relation description: ");

    while (c.hasNext()) {
       ClassStructure cs = (ClassStructure)c.next();
```

```
        if (cs.getClassName().equals(className)) {
                        cs.deleteRelationship(relaType, relaDestination, relaName);
                                return;
        }
    }
    JOptionPane.showMessageDialog(null, "delete: No match class found!",
    "Error", JOptionPane.PLAIN_MESSAGE);
}

public void addAttr() {
  ListIterator c = classes.listIterator();

   String className, attrType, attrName;

   className = JOptionPane.showInputDialog("Enter class name:");
                attrType = JOptionPane.showInputDialog("Enter attr. type: ");

   attrName = JOptionPane.showInputDialog("Enter attr. name: ");


   while (c.hasNext()) {
      ClassStructure cs = (ClassStructure)c.next();
      if (cs.getClassName().equals(className)) {
                        cs.addAttribute(attrType,attrName);
                        return;
      }
                }
   JOptionPane.showMessageDialog(null, "add: No match class found!",
   "Error", JOptionPane.PLAIN_MESSAGE);
}

public void addFunc() {
  ListIterator c = classes.listIterator();
   String className, funcType, funcName;

   className = JOptionPane.showInputDialog("Enter class name:");
                funcType = JOptionPane.showInputDialog("Enter func. type: ");

   funcName = JOptionPane.showInputDialog("Enter func. name: ");

   while (c.hasNext()) {
      ClassStructure cs = (ClassStructure)c.next();
      if (cs.getClassName().equals(className)) {
                        cs.addFunction(funcType,funcName);
                        return;
      }
   }
      JOptionPane.showMessageDialog(null, "add: No match class found!",
     "Error", JOptionPane.PLAIN_MESSAGE);
}

public void addRelation() {
  ListIterator c = classes.listIterator();
```

```
   String className, relaType, relaDestination, relaName;

   className = JOptionPane.showInputDialog("Enter relation from class:");
                     relaType = JOptionPane.showInputDialog("Enter relation type:" +
                     "(choose from association, aggregation, composition and inheritance)");

   relaDestination =
   JOptionPane.showInputDialog("Enter relation destination class: ");
                     relaName = JOptionPane.showInputDialog("Enter relationship description: ");

   while (c.hasNext()) {
      ClassStructure cs = (ClassStructure)c.next();
      if (cs.getClassName().equals(className)) {
       if (existClass(relaDestination)) {
                        cs.addRelationship(relaType, relaDestination, relaName);
                  return;
       }
      }
  }
      JOptionPane.showMessageDialog(null, "add: No match class found!",
     "Error", JOptionPane.PLAIN_MESSAGE);
}

public boolean existClass(String className) {
                  //check if the input class exists
                  ListIterator c = classes.listIterator();
                  while (c.hasNext()) {
                        ClassStructure cs = (ClassStructure)c.next();
                        if (cs.getClassName().equals(className))
                              return true;
 }
return false;
}

public void changeAttr() {
  ListIterator c = classes.listIterator();

   String className, oldAttrType, oldAttrName, newAttrType, newAttrName;
   className = JOptionPane.showInputDialog("Enter class name:");
oldAttrType = JOptionPane.showInputDialog("Enter old attr. type: ");
   oldAttrName = JOptionPane.showInputDialog("Enter old attr. name: ");
newAttrType = JOptionPane.showInputDialog("Enter new attr. type: ");
   newAttrName = JOptionPane.showInputDialog("Enter new attr. name: ");

   while (c.hasNext()) {
      ClassStructure cs = (ClassStructure)c.next();
      if (cs.getClassName().equals(className)) {
 cs.replaceAttribute(oldAttrType,oldAttrName,  newAttrType,newAttrName);
                        return;
      }
   }
      JOptionPane.showMessageDialog(null, "replace: No match class found!",
     "Error", JOptionPane.PLAIN_MESSAGE);
```

```
}

public void changeFunc() {
  ListIterator c = classes.listIterator();

   String className, oldFuncType, oldFuncName, newFuncType,newFuncName;

   className = JOptionPane.showInputDialog("Enter class name:");
oldFuncType = JOptionPane.showInputDialog("Enter old Func. type: ");
   oldFuncName = JOptionPane.showInputDialog("Enter old Func. name: ");
 newFuncType = JOptionPane.showInputDialog("Enter new Func. type: ");
   newFuncName = JOptionPane.showInputDialog("Enter new Func. name: ");

   while (c.hasNext()) {
      ClassStructure cs = (ClassStructure)c.next();
      if (cs.getClassName().equals(className)) {
                        cs.replaceFunction(oldFuncType,oldFuncName,
                                                        newFuncType,newFuncName);
                        return;
      }
   }
      JOptionPane.showMessageDialog(null, "replace: No match class found!",
     "Error", JOptionPane.PLAIN_MESSAGE);
}

public void changeRelation() {
  ListIterator c = classes.listIterator();

   String className, oldRelType, oldRelDestination, oldRelName,
                                newRelType,newRelDestination, newRelName;

   className =
   JOptionPane.showInputDialog("Enter class name:");
                    oldRelType =
                    JOptionPane.showInputDialog("Enter old relation type: " +
"(choose from association, aggregation, composition and inheritance)");
   oldRelDestination =
   JOptionPane.showInputDialog("Enter old relation destination class: ");
   oldRelName =
   JOptionPane.showInputDialog("Enter old relation description: ");

                    newRelType =
                    JOptionPane.showInputDialog("Enter new relation type: " +
"(choose from association, aggregation, composition and inheritance)");
   newRelDestination =
   JOptionPane.showInputDialog("Enter new relation destination class: ");
   newRelName =
   JOptionPane.showInputDialog("Enter new relation description: ");


   while (c.hasNext()) {
      ClassStructure cs = (ClassStructure)c.next();
      if (cs.getClassName().equals(className)) {
```

```
    cs.replaceRelationship(oldRelType,oldRelDestination,oldRelName,
                           newRelType,newRelDestination, newRelName);
                           return;
       }
    }
       JOptionPane.showMessageDialog(null, "replace: No match class found!",
      "Error", JOptionPane.PLAIN_MESSAGE);
}

public void saveAs() {
  ListIterator c = classes.listIterator();
  PrintStream ps = null;
  //String fileName;

  JFileChooser chooser = new JFileChooser();
                    if (chooser.showSaveDialog(null)
                       == JFileChooser.APPROVE_OPTION) {
                      File fileName = chooser.getSelectedFile();
  try {
     ps  = new PrintStream(new FileOutputStream(fileName));
  } catch (Exception e) {
     e.printStackTrace();
     System.exit(0);
  }

  ps.println("<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?>");
  ps.println("<!-- Edited by Kaiyan Li -->");
  ps.println("<uml>");

   while (c.hasNext()) {
       ClassStructure cs = (ClassStructure)c.next();
       cs.toXML(ps);
   }

  ps.println("</uml>");
}

public void save() {
  ListIterator c = classes.listIterator();
  PrintStream ps = null;
   try {
     ps  = new PrintStream(new FileOutputStream("../saveback.xml"));
  } catch (Exception e) {
     e.printStackTrace();
     System.exit(0);
  }
  ps.println("<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?>");
  ps.println("<!-- Edited by Kaiyan Li -->");
  ps.println("<uml>");
   while (c.hasNext()) {
       ClassStructure cs = (ClassStructure)c.next();
       cs.toXML(ps);
   }
```

```
    ps.println("</uml>");
  }
  public void clearPanel() {
                    diagrams.clear();
                    classes = null;
  }
  public void refreshDiagram() {
                     clearPanel();
    fileName = "../saveback.xml";
    try {
                        SAXParser parser = new SAXParser();
                        XmlHandler handler = new XmlHandler();
                        parser.setContentHandler(handler);

InputSource is = new InputSource(new FileReader(fileName));
                        parser.parse(is);
                    classes = handler.getClasses();
                    initDiagrams();
                    setLayout();
    } catch (Exception e) {
                        e.printStackTrace();
     }
  }
\
  public void init() {
                     JFileChooser chooser = new JFileChooser();
                     if (chooser.showOpenDialog(null)
                       == JFileChooser.APPROVE_OPTION) {
                       File fileName = chooser.getSelectedFile();

                        try {
                                SAXParser parser = new SAXParser();
                                XmlHandler handler = new XmlHandler();
                                parser.setContentHandler(handler);

                    InputSource is = new InputSource(new FileReader(fileName));

                                parser.parse(is);
                        classes = handler.getClasses();
                        initDiagrams();
                        setLayout();
                        } catch (Exception e) {
                                e.printStackTrace();
                                JOptionPane.showMessageDialog(null,
                                "The System cannot open this file!",
                    "Error", JOptionPane.ERROR_MESSAGE);
                        }
                      }
  }
 private void initDiagrams() {
                        ListIterator iter = classes.listIterator();
                        Point start = new Point(30, 30);
```

```
                        while (iter.hasNext()) {
                                ClassStructure cs = (ClassStructure) iter.next();
ClassDiagram dia = new ClassDiagram(start, cs);
                                diagrams.put(cs.getClassName(), dia);

                        iter = classes.listIterator(
                        while (iter.hasNext()) {
                                ClassStructure cs = (ClassStructure) iter.next();
                                LinkedList rl = cs.getRelationshipList();

                                if (rl != null && !rl.isEmpty()) {
                                        ListIterator rlIter = rl.listIterator();
                                        while (rlIter.hasNext()) {
                                                RelationshipStructure rs =
                                (RelationshipStructure) rlIter.next();
                                                int type = rs.getType();
                                                String name = rs.getName();

                                                ClassDiagram from =
                        (ClassDiagram)diagrams.get(cs.getClassName());

                                                ClassDiagram to =
                (ClassDiagram)diagrams.get(rs.getDestination());

                        Connection conn = new Connection(type, from, to, name);
                                        from.addConnection(conn);
                                                to.addConnection(conn);

}}}}

public void swap(ClassDiagram one, ClassDiagram two) {
                        ClassDiagram temp = null;
                        temp = one;
                        one = two;
                        two = temp;
}

public void resetPosition() {
                Iterator iter = diagrams.values().iterator();
                int right = 0;
                int bottom = 0;
                int vGap = 30; //space between top and bottom class diagrams
                int hGap = 80; //space between left and right class diagrams
                int maxHeight = 0; //maxHight for each level
                ArrayList allLevels = new ArrayList();
                ArrayList sameLevel = new ArrayList();
                allLevels.add(sameLevel);
                System.out.println("height = " + this.getHeight());

                while (iter.hasNext()) {
                        ClassDiagram dia = (ClassDiagram) iter.next();
    int level = dia.getLevel();
```

```
    if (level >= allLevels.size()) {
                    sameLevel = new ArrayList();
                    allLevels.add(sameLevel);
    }

    else {
     sameLevel = (ArrayList)allLevels.get(level);
    }

    sameLevel.add(dia);
  }

  iter = allLevels.iterator();

  while (iter.hasNext()) {
                    sameLevel = (ArrayList) iter.next();
                    Iterator iter2 = sameLevel.iterator();

                    while (iter2.hasNext()) {
                       ClassDiagram dia = (ClassDiagram) iter2.next();
                       dia.setPosition(right + hGap, bottom + vGap);
                       right += dia.getWidth() + hGap;

                       if (maxHeight < dia.getHeight()) {
                          maxHeight = dia.getHeight();
                       }
                    }

                    bottom += maxHeight + vGap;
                    Dimension size = this.getSize();
                    int newHeight = (int)size.getHeight();
                    int newWidth = (int)size.getWidth();

                    if (bottom+vGap > newHeight) {
                          newHeight = bottom + vGap;
                    }

                    if (right+hGap > newWidth) {
                          newWidth = right + hGap;
                    }
    Dimension newSize = new Dimension(newWidth, newHeight);
                    this.setPreferredSize(newSize);
                    this.revalidate();
                    maxHeight = 0;
                    right = 0;
  }
}

public int getMaxHeight() {
                          ListIterator iter = classes.listIterator();
                          int maxHeight = 0;

                          while (iter.hasNext()) {
```

```
                                ClassStructure cs = (ClassStructure) iter.next();
                                int  height = cs.getAllSize() * 12;

        if (height > maxHeight)
          maxHeight = height;
      }
      System.out.println("max length" + maxHeight);
      return maxHeight;
  }

  public void setLayout() {
                        doInheritanceLevel();
                        doNotInheritanceLevel();
                        doOtherLevel();
                        resetPosition();
  }


  public void drawRelation(Graphics g) {
                ListIterator iter = classes.listIterator();

                while (iter.hasNext()) {
                        ClassStructure cs = (ClassStructure) iter.next();
                        LinkedList rl = cs.getRelationshipList();
//*****************************************
                        if (rl != null && !rl.isEmpty()) {
                                ListIterator rlIter = rl.listIterator();

                                while (rlIter.hasNext()) {
                                        RelationshipStructure rs =
                                        (RelationshipStructure) rlIter.next();
                                        int type = rs.getType();
                                        ClassDiagram from =
                                        (ClassDiagram) diagrams.get(cs.getClassName());
                                        ClassDiagram to =
                                        (ClassDiagram) diagrams.get(rs.getDestination());
                                        String name = rs.getName();
                System.out.println("from: " + from + " to: " + to);

                                        if (to == null) {
                                                JOptionPane.showMessageDialog(null,
"A parent class is missing or been deleted, check the structure then run again!",
                                "Error", JOptionPane.ERROR_MESSAGE);
                                clearPanel();
                                        repaint();
                                        }

                Connection conn = new Connection(type, from, to, name);
                        System.out.println("conn type: " + conn.getType() +
                        "conn from: " + conn.getFrom() + " to:" + conn.getTo()
                        + "name:" + conn.getName());
                        conn.drawConnection(from, to, name, g);
                          }
```

```
                    }
                }

    }

    public void doInheritanceLevel() {
                    Iterator iter = diagrams.values().iterator();

                    while (iter.hasNext()) {
                        ClassDiagram dia = (ClassDiagram) iter.next();

                        if (dia.isRootClass()) {
                            dia.recursiveSetInheritanceLevel(0);
                        }
                    }
    }

    public void doNotInheritanceLevel() {
                    Iterator iter = diagrams.values().iterator();
                    while (iter.hasNext()) {
                        ClassDiagram dia = (ClassDiagram) iter.next();
                        if (!dia.hasInheritance()) {
                            dia.setOtherLevel(0);
                        }
                    }
                }
    public void doOtherLevel() {
                    Iterator iter = diagrams.values().iterator();
                    while (iter.hasNext()) {
                      ClassDiagram dia = (ClassDiagram) iter.next();
        int level = dia.getLevel();
                if (level == -1)
                    dia.setRelatLevel();
} }}
```

Source Code of XMLHandler.java

```
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import java.util.LinkedList;

public class XmlHandler extends DefaultHandler
{
 LinkedList classes = new LinkedList();
 ClassStructure cs;
 AttributeStructure as;
 FunctionStructure fs;
 ArgumentStructure ars;
 RelationshipStructure rs;
```

```
public void startElement(String uri, String localName,
String qName, Attributes attributes) throws SAXException {
                    if (localName.equals("class")) {
                        cs = new ClassStructure (attributes.getValue("className"));
                        cs.setScope(attributes.getValue("scope"));
                    }

                    else if (localName.equals("attribute")) {
                        as = new AttributeStructure(attributes.getValue("type"),
                        attributes.getValue("name"));
   cs.addAttribute(as);
                    }
  else if (localName.equals("function")) {
       fs = new FunctionStructure(attributes.getValue("type"),
       attributes.getValue("name"));
       cs.addFunction(fs);
  }
  else if (localName.equals("argument")) {
       ars = new ArgumentStructure (attributes.getValue("type"),
       attributes.getValue("name"));
       fs.addArgument(ars);
  }
  else if (localName.equals("relationship")) {
       rs = new RelationshipStructure (attributes.getValue("type"),
       attributes.getValue("destination"), attributes.getValue("name"));
       cs.addRelationship(rs);
 }}
 public void endElement(String uri, String localName, String gName) {
                    if (localName.equals("class")) {
                        classes.add(cs);
} }
 public LinkedList getClasses() {
                    return classes;
 }}
```

APPENDIX B

GAMMA'S DESIGN PATTERNS

Table 17. Builder.XML

Builder.XML

```
<?XML version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by Kaiyan Li -->
<!-- pattern:Builder -->
<uml>
        <class className="Handler">
                <function type = "" name = "HandleRequest"></function>
                <relationship type = "association" destination="Handler"
name="successor"> </relationship>
        </class>

        <class className="client">
                <relationship type="association"
destination="Handler"></relationship>
        </class>

        <class className="ConcreteHandler1">
                <function type = "" name="HandlerRequest"></function>
                <relationship type="Inheritance"
destination="Handler"></relationship>
        </class>

        <class className="ConcreteHandler2">
                <function type = "" name="HandlerRequest"></function>
                <relationship type="Inheritance"
destination="Handler"></relationship>
        </class>

        <class className="Director">
                <function type="" name="Construct"></function>
                <relationship type="aggregation" destination="Builder"
name="builder"></relationship>
        </class>

        <class className="Builder">
                <function type="" name="BuildPart"></function>
        </class>

        <class className="ConcreteBuilder">
                <function type="" name="BuildPart"></function>
                <function type="" name="GetResult"></function>
                <relationship type="inheritance"
destination="Builder"></relationship>
        </class>
</uml>
```

Table 18. Abstruct_Factory.XML

Abstract_Factory.XML
<?XML version="1.0" encoding="ISO-8859-1"?> <!-- Edited by Kaiyan Li --> <!-- pattern:Abstruct Factory --> <uml> <class className="AbstractFactory"> <function type="" name="CreateProductA"></function> <function type="" name="CreateProductB"></function> </class> <class className="ConcreteFectory1"> <function type="" name="CreateProductA"></function> <function type="" name="CreateProductB"></function> <relationship type="Inheritance" destination="AbstractFactory"></relationship> </class> <class className="ConcreteFactory2"> <function type="" name="CreateProductA"></function> <function type="" name="CreateProductB"></function> <relationship type="Inheritance" destination="AbstractFactory"></relationship> </class> <class className="AbstructProductA"> </class> <class className="ProductA2"> <relationship type="inheritance" destination="AbstructProductA"></relationship> </class> <class className="ProductA1"> <relationship type="inheritance" destination="AbstructProductA"></relationship> </class> <class className="AbstractProductB"> </class> <class className="ProductB2"> <relationship type="inheritance" destination="AbstractProductB"></relationship> </class> <class className="ProductB1"> <relationship type="inheritance" destination="AbstractProductB"></relationship> </class> <class className="Client"> <relationship type="association" destination="AbstractFactory"></relationship> <relationship type="association" destination="AbstructProductA"></relationship> <relationship type="association" destination="AbstractProductB"></relationship></class></uml>

Table 19. Adapter.XML

Adapter.XML

```
<?XML version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by Kaiyan Li -->
<!-- pattern:Adapter -->

<uml>
        <class className="Target">
                <function type="" name="Request"></function>
        </class>

        <class className="Adaptee">
                <function type="" name="SpecificRequest"></function>
        </class>

        <class className="Adapter">
                <function type="" name="Request"></function>
                <relationship type="inheritance"
destination="Target"></relationship>
                <relationship type="inheritance" destination="Adaptee"
name="(implementation)"></relationship>
        </class>

        <class className="Client">
                <relationship type="Association"
destination="Target"></relationship>
        </class>
</uml>
```

Table 20. Bridge.XML

Bridge.XML

```
<?XML version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by Kaiyan Li -->
<!-- pattern:Bridge -->
<uml>
        <class className="Abstraction">
                <function type="" name="Operation"></function>
                <relationship type="aggregation" destination="Implementor"
name="imp"></relationship>
        </class>

        <class className="RefinedAbstraction">
                <relationship type="inheritance"
destination="Abstraction"></relationship>
        </class>

        <class className="Implementor">
                <function type="" name="Operationimp"></function>
        </class>

        <class className="ConcreteImplementorA">
                <function type="" name="Operationimp"></function>
                <relationship type="inheritance"
destination="Implementor"></relationship>
        </class>

        <class className="ConcreteImplementorB">
                <function type="" name="Operationimp"></function>
                <relationship type="inheritance"
destination="Implementor"></relationship>
        </class>
</uml>
```

Table 21. Chain_of_Responsibility.XML

Chain_of_Responsibility.XML

```
<?XML version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by Kaiyan Li -->
<!-- patterns: Chain of Responsibility -->

<uml>
        <class className="Handler">
                <function type = "" name = "HandleRequest"></function>
                <relationship type = "association" destination="Handler"
name="successor"> </relationship>
        </class>

        <class className="client">
                <relationship type="association"
destination="Handler"></relationship>
        </class>

        <class className="ConcreteHandler1">
                <function type = "" name="HandlerRequest"></function>
                <relationship type="Inheritance"
destination="Handler"></relationship>
        </class>

        <class className="ConcreteHandler2">
                <function type = "" name="HandlerRequest"></function>
                <relationship type="Inheritance"
destination="Handler"></relationship>
        </class>
</uml>
```

Table 22. Command.XML

Command.XML

```
<?XML version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by Kaiyan Li -->
<!-- pattern: Command -->
<uml>
        <class className="Invoker">
            <relationship type="aggregation"
destination="Command"></relationship>
        </class>

        <class className="Command">
                <function type = "" name="Execute"></function>

        </class>

        <class className="Client">
                <relationship type="association"
destination="Receiver"></relationship>                          </class>

        <class className="Receiver">
                <function type="" name="Action"></function>
        </class>

        <class className="ConcreteCommand">
                <attribute type = "" name = "state"></attribute>
                <function type = "" name="Execute"></function>
        <relationship type="inheritance"
destination="Command"></relationship>
        <relationship type="association" destination="Receiver"
name="receiver"></relationship>
        </class>
                                        </uml>
```

Table 23. Composite.XML

Composite.XML

```
<?XML version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by Kaiyan Li -->
<!-- pattern:Composite -->
<uml>

        <class className="Component">
                <function type="" name="Operation"></function>
                <function type="" name="Add">
                <argument type="Component" name=""></argument>
                </function>
                <function type="" name="Remove">
                <argument type="Component" name=""></argument>
                </function>
                <function type="" name="GetChild">
                <argument type="int" name=""></argument>
                </function>
        </class>

        <class className="Leaf">
                <function type="" name="Operation"></function>
                <relationship type="inheritance"
destination="Component"></relationship>
        </class>

        <class className="Client">
                <relationship type="association"
destination="Component"></relationship>
        </class>

        <class className="Composite">
                <function type="" name="Operation"></function>
                <function type="" name="Add">
                <argument type="Component" name=""></argument>
                </function>
                <function type="" name="Remove">
                <argument type="Component" name=""></argument>
                </function>
                <function type="" name="GetChild">
                <argument type="int" name=""></argument>
                </function>
                <relationship type="inheritance"
destination="Component"></relationship>
                <relationship type="aggregation" destination="Component"
name="children"></relationship>
        </class></uml>
```

Table 24. Decorator.XML

Decorator.XML

```
<?XML version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by Kaiyan Li -->
<!-- pattern:Decorator -->

<uml>
        <class className="Component">
                <function type="" name="Operation"></function>
        </class>

        <class className="Decorator">
                <function type="" name="Operation"></function>
                <relationship type="inheritance"
destination="Component"></relationship>
        <relationship type="Aggregation" destination="Component"
name="component"></relationship>
        </class>

        <class className="ConcreteComponent">
                <function type="" name="Operation"></function>
                <relationship type="inheritance"
destination="Component"></relationship>
        </class>

        <class className="ConcreteDecoratorA">
                <attribute type="" name="addedState"></attribute>
                <function type="" name="Operation"></function>
                <relationship type="inheritance"
destination="Decorator"></relationship>
        </class>

        <class className="ConcreteDecoratorB">
                <function type="" name="Operation"></function>
                <function name="AddedBehavior"></function>
                <relationship type="inheritance"
destination="Decorator"></relationship>
        </class>
</uml>
```

Table 25. Façade.XML

Façade.XML

```
<?XML version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by Kaiyan Li -->
<!-- pattern:Facade -->
<uml>
        <class className="Facade">
                <relationship type="Association"
destination="null"></relationship>
        </class>

        <class className="Null">
        </class>

        <class className="null">
                <relationship type="Inheritance"
destination="Null"></relationship>
        </class>
                                        </uml>
```

Table 26. Factory_Method.XML

Factory_Method.XML

```
<?XML version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by Kaiyan Li -->
<!-- pattern:Factory_Method -->
<uml>
        <class className="Product">
        </class>
        <class className="ConcreteProduct">
                <relationship type="inheritance"
destination="Product"></relationship>
        </class>

        <class className="Creator">
                <function type="" name="FactoryMethod"></function>
                <function type="" name="AnOperation"></function>
        </class>
        <class className="ConcreteCreator">
                <function type="" name="FactoryMethod"></function>
                <relationship type="inheritance"
destination="Creator"></relationship>
        </class>
</uml>
```

Table 27. Flyweight.XML

Flyweight.XML

```
<?XML version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by Kaiyan Li -->
<!-- pattern:Flyweight -->

<uml>

        <class className="FlyweightFactory">
                <function type="" name="GetFlyweight"></function>
<relationship type="Aggregation" destination="Flyweight"
name="flyweights"></relationship>
        </class>

        <class className="Client">
                <relationship type="association"
destination="FlyweightFactory"></relationship>
                <relationship type="association"
destination="ConcreteFlyweight"></relationship>
        <relationship type="association"
destination="UnsharedConcreteFlyweight"></relationship>
        </class>

        <class className="ConcreteFlyweight">
                <attribute type="" name="intrinsicState"></attribute>
                <function type="" name="Operation">
                <argument type="" name="extrinsicState"></argument>
                </function>
                <relationship type="inheritance"
destination="Flyweight"></relationship>
        </class>

        <class className="UnsharedConcreteFlyweight">
                <attribute type="" name="allState"></attribute>
                <function type="" name="Operation">
                <argument type="" name="extrinsicState"></argument>
                </function>
                <relationship type="inheritance"
destination="Flyweight"></relationship>
        </class>

        <class className="Flyweight">
                <function type="" name="Operation">
                <argument type="" name="extrinsicState"></argument>
                </function>
        </class>
</uml>
```

Table 28. Interpretor.XML

Interpreter.XML

```
<?XML version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by Kaiyan Li -->
<!-- patterns: Interpreter -->
<uml>
        <class className="AbstractExpression">
                <function type = "" name="interpret"></function>
        </class>

        <class className="Context">
        </class>

        <class className="Client">
                <relationship type="association"
destination="Context"></relationship>
                <relationship type="association"
destination="AbstractExpression">
        </relationship>
        </class>

        <class className="TerminalExpression">
                <function type = "" name = "interpret">
                        <argument type = "" name = "COntext"></argument>
                </function>
                <relationship type="inheritance"
destination="AbstractExpression"></relationship>
        </class>

        <class className="NonTerminalExpression">
                <function type = "" name= "interpret">
                <argument type = "" name = "Context"></argument>
                </function>
                <relationship type="inheritance"
destination="AbstractExpression"></relationship>
                <relationship type="association"
destination="AbstractExpression"></relationship>
        </class>
</uml>
```

Table 29. Iterator.XML

Iterator.XML

```
<?XML version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by Kaiyan Li -->
<!-- pattern:Iterator -->

<uml>
        <class className="Aggregate">
                <function type = "" name="CreateIterator"></function>
        </class>

        <class className="ConcreteAggregate">
                <function type = "" name="CreateIterator"></function>
                <relationship type="inheritance"
destination="Aggregate"></relationship>
                </class>

        <class className="Iterator">
                <function type = "" name="First"></function>
                <function type = "" name="Next"></function>
                <function type = "" name="IsDone"></function>
                <function type = "" name="CurrentItem"></function>
        </class>

        <class className="ConcreteIterator">
                <relationship type="inheritance"
destination="Iterator"></relationship>
                <relationship type="association"
        destination="ConcreteAggregate"></relationship>
        </class>

        <class className="Client">
                <relationship type="association"
destination="Aggregate"></relationship>
                <relationship type="Association"
destination="Iterator"></relationship>
        </class>

                                        </uml>
```

Table 30. MasterSlave.XML

MasterSlave.XML
<?XML version="1.0" encoding="ISO-8859-1"?> <!-- Edited by Kaiyan Li --> <uml> <class className = "Master"> <relationship type = "association" destination = "Slave" name = "SendResult()"/> <relationship type = "association" destination = "ConcreteSlave" name = "Create()"/> </class> <class className = "Slave"> <function name = "InitializeTask" > </function> </class> <class className = "ConcreteSlave"> <function name = "InitializeTask" > </function> <relationship type = "inheritance" destination = "Slave"/> </class></uml>

Table 31. Mediator.XML

Mediator.XML
<?XML version="1.0" encoding="ISO-8859-1"?> <!-- Edited by Kaiyan Li --> <!-- pattern:Mediator --> <uml> <class className="Mediator"> </class> <class className="ConcreteMediator"> <relationship type="inheritance" destination="Mediator"></relationship> <relationship type="association" destination="ConcreteColleague1"></relationship> <relationship type="association" destination="ConcreteColleague2"></relationship> </class><class className="Colleague"> <relationship type="association" destination="Mediator" name="mediator"></relationship></class> <class className="ConcreteColleague1"> <relationship type="inheritance" destination="Colleague"></relationship></class> <class className="ConcreteColleague2"> <relationship type="inheritance" destination="Colleague"> </relationship> </class></uml>

Table 32. Memento.XML

Memento.XML

```
<?XML version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by Kaiyan Li -->
<!-- pattern:Memento -->

<uml>
        <class className="Originator">
                <attribute type="" name="state"></attribute>
                <function type="" name="SetMemento">
                <argument type="Memento" name="m"></argument>
                </function>
                <function type="" name="CreateMemento"></function>
                <relationship type="association"
destination="Memento"></relationship>
        </class>

        <class className="Memento">
                <attribute type="" name="state"></attribute>
                <function type="" name="GetState"></function>
                <function type="" name="SetState"></function>

        </class>

        <class className="Caretaker">
                <relationship type="aggregation" destination="Memento"
name="memento"></relationship>
        </class>
</uml>
```

Table 33. Observer.XML

Observer.XML

```
<?XML version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by Kaiyan Li -->
<!-- pattern:Observer -->
<uml>
        <class className="Subject">
                <function type="" name="Attach">
                <argument type="" name="Observer"></argument>
                </function>
                <function type="" name="Detach">
                <argument type="" name="Observer"></argument>
                </function>
                <function type="" name="Notify"></function>
                <relationship type="association" destination="Observer"
name="observers"></relationship>
        </class>

        <class className="ConcreteSubject">
                <attribute type="" name="subjectState"></attribute>
                <function type="" name="GetState"></function>
                <function type="" name="SetState"></function>

                <relationship type="Inheritance"
destination="Subject"></relationship>


        </class>

                <class className="Observer">

                <function type="" name="Update"></function>

        </class>

                <class className="ConcreteObserver">
                <attribute type="observerState" name=""></attribute>

                <function type="" name="Update"></function>

                <relationship type="Inheritance"
destination="Observer"></relationship>
<relationship type="association" destination="ConcreteSubject"
name="subject"></relationship>
</class></uml>
```

Table 34. Prototype.XML

Prototype.XML

```
<?XML version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by Kaiyan Li -->
<!-- pattern:Prototype -->
<uml>
        <class className="Prototype">
                <function type="" name="Clone"></function>
        </class>

        <class className="ConcretePrototype1">
                <function type="" name="Clone"></function>
                <relationship type="inheritance"
destination="Prototype"></relationship>
        </class>

        <class className="ConcretePrototype2">
                <function type="" name="Clone"></function>
                <relationship type="inheritance"
destination="Prototype"></relationship>
        </class>

        <class className="Client">
                <function type="" name="Operation"></function>
                <relationship type="association" destination="Prototype"
name="prototype"></relationship>
        </class>
</uml>
```

Table 35. Proxy.XML

Proxy.XML
<?XML version="1.0" encoding="ISO-8859-1"?> <!-- Edited by Kaiyan Li --> <!-- pattern:Proxy --> <uml> <class className="Subject"> <function type="" name="Request"></function> <function type="" name="..."></function> </class> <class className="RealSubject"> <function type="" name="Request"></function> <function type="" name="..."></function> <relationship type="Inheritance" destination="Subject"></relationship> </class> <class className="Proxy"> <function type="" name="Request"></function> <function type="" name="..."></function> <relationship type="Inheritance" destination="Subject"></relationship> <relationship type="association" destination="RealSubject" name="realSubject"></relationship> </class> </uml>

Table 36. Singleton.XML

Singleton.XML
<?XML version="1.0" encoding="ISO-8859-1"?> <!-- Edited by Kaiyan Li --><!-- pattern:Singleton --> <uml> <class className="Singleton"> <attribute name="uniqueinstance" type="static"></attribute> <attribute name="singletonData" type=""></attribute> <function name="instance" type="static"></function> <function type="" name="SingletonOperation"></function> <function type="" name="GetSingletonData"></function> </class></uml>

Table 37. State.XML

State.XML

```
<?XML version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by Kaiyan Li -->
<!-- pattern:State -->

<uml>
        <class className="Context">
                <function type="" name="Request"></function>
                <relationship type="Aggregation" destination="State"
name="state"></relationship>
        </class>

        <class className="State">
                <function type="" name="Handle"></function>
        </class>

        <class className="ConcreteStateA">
                <function type="" name="Handle"></function>
                <relationship type="Inheritance"
destination="State"></relationship>
        </class>

        <class className="ConcreteStateB">
                <function type="" name="Handle"></function>
                <relationship type="Inheritance"
destination="State"></relationship>
        </class>
</uml>
```

Table 38. Strategy.XML

Strategy.XML
<?XML version="1.0" encoding="ISO-8859-1"?> <!-- Edited by Kaiyan Li --> <!-- pattern:Strategy --> <uml> <class className="Context"> <function type="" name="ContextInterface"></function> <relationship type="aggregation" destination="Strategy" name="strategy"></relationship> </class> <class className="Strategy"> <function type="" name="AlgorithmInterface"></function> </class> <class className="ConcreteStrategeA"> <function type="" name="AlgorithmInterface"></function> <relationship type="inheritance" destination="Strategy"></relationship> </class> <class className="ConcreteStrategeB"> <function type="" name="AlgorithmInterface"></function> <relationship type="inheritance" destination="Strategy"></relationship> </class> <class className="ConcreteStrategeC"> <function type="" name="AlgorithmInterface"></function> <relationship type="inheritance" destination="Strategy"></relationship> </class> </uml>

Table 39. Templete_Method.XML

Template_Method.XML
<?XML version="1.0" encoding="ISO-8859-1"?> <!-- Edited by Kaiyan Li --> <!-- pattern:Template_Method --> <uml> <class className="AbstractClass"> <function type="" name="TemplateMethod"></function> <function type="" name="PrimitiveOperation1"></function> <function type="" name="PrimitiveOperation2"></function> </class> <class className="ConcreteClass"> <function type="" name="PrimitiveOperation1"></function> <function type="" name="PrimitiveOperation2"></function> <relationship type="inheritance" destination="AbstractClass"></relationship> </class> </uml>

Table 50. Visitor.XML

Visitor.XML
<?XML version="1.0" encoding="ISO-8859-1"?> <!-- Edited by Kaiyan Li --> <!-- pattern:Visitor --> <uml> <class className="ConcreteVisitor1"> <function type="" name="VisitConcreteElementA"> <argument type="" name="ConcreteElementA"></argument> </function> <function type="" name="VisitConcreatElementB"> <argument type="" name="ConcreteElementB"></argument> </function> <relationship type="inheritance" destination="Visitor"></relationship> </class>

Continues next page.

```
        <class className="Element">
                <function type="" name="Accept">
                <argument type="" name="Visitor"></argument>
                </function>
        </class>

        <class className="ObjectStructure">
                <relationship type="association"
destination="Element"></relationship>
        </class>
        <class className="ConcreteElementB">
                <function type="" name="Accept">
                <argument type="Visitor" name="v"></argument>
                </function>
                <function type="" name="OperationA"></function>
                <relationship type="inheritance"
destination="Element"></relationship>
        </class>

        <class className="ConcreteElementA">
                <function type="" name="Accept">
                <argument type="Visitor" name="v"></argument>
                </function>
                <function type="" name="OperationA"></function>
                <relationship type="inheritance"
destination="Element"></relationship>
        </class>

        <class className="ConcreteVisitor2">
                <function type="" name="VisitConcreteElementA">
                <argument type="" name="ConcreteElementA"></argument>
                </function>
                <function type="" name="VisitConcreatElementB">
                <argument type="" name="ConcreteElementB"></argument>
                </function>
                <relationship type="inheritance"
destination="Visitor"></relationship>
        </class>
        <class className="Visitor">
                <function type="" name="VisitConcreteElementA">
                <argument type="" name="ConcreteElementA"></argument>
                </function>
                <function type="" name="VisitConcreatElementB">
                <argument type="" name="ConcreteElementB"></argument>
                </function>        </class></uml>
```

REFERENCES

[1] IEEE STD 830- 1993 IEEE Recommended Practice of Software Requirements Specifications

[2] IEEE STD 830- 1998 IEEE Recommended Practice of Software Requirements Specifications- Annex A

[3] Katrina Yun Ji, ADAP: A Component-based Model, Master Thesis, Department of Computer Science, California State University San Bernardino, 2000

[4] Martin Fowler, UML Distilled (Second edition), Addison Wesley, 1999

[5] Erich Gamma, Design Patterns - elements of reusable object-oriented software, Addison Wesley, 1995

[6] J. Craig Cleaveland, Program generators with XML and JAVA, Prentice Hall PTR, 2001

[7] Cay Horstmann, Computing Concepts with JAVA 2 essentials (second edition), John Wiley & Sons, 2000

[8] David Carlson, Modeling XML Applications with UML, Addison Wesley, 2001

[9] Alfred V. Aho, Compilers - principles, techniques, and tools, Addison Wesley, 1986

[10] j2se1.4.0 Documentation, http://Java.sun.com/j2se/downloads.html

[11] Apache Xerces-2_0_1 XML Parser,

http://XML.apache.org/dist/xerces-j/

[12] Weichun Wu, Online DPL: Online Design Pattern Library, Master Project, Department of Computer Science, California State University, San Bernardino, 2002