

University of Benghazi
Faculty of Information Technology
Computer Science Department



***Xml – based parser to extract design pattern
in xml files***

A thesis Submitted to faculty of information technology in partial
fulfillment of the requirement of the Master's Degree
in Software Engineering

By

Samah Ali Naief

Supervisor

Dr.Omar Mustafa El-sallabi

March – 2012

Abstract

Much research has been carried out on the importance of design patterns as a tool that saves time in the design of a software program and establishes helpful results in different platforms. However, the methods for pattern representation, organization, saving and restoring have not been stated clearly. This study attempts to find an easy way to turn data into an XML (extensible markup language) file that shows the component pattern as a text, and in order to help the designer to organize and save such patterns. We begin by applying a design pattern that has been described and documented by a catalog, published on a web page and stored in xml format. The store operation consists of saving the page, taking the content and using it in an application. Most the time these patterns are reduced, because no clear methods for saving and returning the data when it is needed are available. A tool is required to perform this task. The tool will be applied by using an xml-based parser that compiles an xml file and transforms it into a formula understandable by all designers. The proposed tool will use the HTML (Hypertext Markup language) technology to display the xml-file as an acceptable text, desired by the designer, and suitable language such as vb.net and asp.net will be selected to implement the tool. The aim of the present work is to provide a work mechanism for designers. This mechanism will allow information and experience to be exchanged among designers in different programming environments, and also reduce time and effort in the design of reliable solutions to problems.

TABLE OF CONTENTS

Chapter 1. Introduction

1.1	Introduction	1
1.2	Problem Statement	3
1.3	Objectives	5
1.4	Thesis structure	7

Chapter 2. Background

2.1	Introduction	6
2.2	Design pattern representation	7
2.3	The Extensible Markup Language (XML)	7
2.4	Xml advantages	7
2.5	Document object model (DOM)	8
2.6	Related work	9
2.6.1	The CO2PS Tool	9
2.6.2	The Meta-CO2PS Tool	11
2.6.3	The Design Pattern Framework TM 3.5	12
2.6.4	The OMT Design Tool	13
2.6.5	The Tool of Florijn et al	13
2.6.6	Framework of Balanyi and Ferenc	14
2.7	Evaluation of the literature tools	15
2.8	Summary	16

Chapter 3. An overview of xml mapping of the EPT

3.1	Introduction	18
3.2	Presenting Data in XML Documents	19

3.3	The Tree Structure of XML Documents	21
3.4	Presented Design Patterns in XML Documents	23
3.5	The Suggested Structure of XML Files	23
3.6	The Structure of the XML mapping of the EPT	35
3.7	The XML Generated Files	36
Chapter 4. The EP Tool Architecture		
4.1	Introduction	37
4.2	The documentation of the classes of the client side	39
4.3	The Documentation of Classes for the Patterns Stored	45
4.4	Tool Interface System	46
Chapter 5. Case Study		56
Chapter 6. Conclusion and future work		
6.1	Conclusion	82
6.2	Scope for future work	84
References		85
Appendix		87

LIST OF FIGURES

Figure 2.1	The main functions in the CO2P2S tool	10
Figure 2.2	Results from the CO2P2S tool	11
Figure 3.1	A general view of an XML structure	21
Figure 3.2	The structure of pattern information in XML files	23
Figure 3.3	The input files for the proposed tool	34
Figure 3.4	General conceptual overview for the proposed tool	35
Figure 4.1	UML class diagram for the EPT	38
Figure 4.2	UML class diagram for patterns stored server side	44
Figure 4.3	The main functions for EPT	46
Figure 4.4	List of patterns available in EPT	47
Figure 4.5	List of classes for the determined pattern	47
Figure 4.6	summery of methods and fileds for the determined pattern	48
Figure 4.7	View details about methods	49
Figure 4.8	Load information xml file of patterns into EPT	50
Figure 4.9	Load xml files of classes for patterns into EPT	50
Figure 4.10	Backup patterns stored in EPT	51
Figure 4.11	Finding pattern availability in EPT	51
Figure 4.12	The result of a search operation	52
Figure 4.13	List of patterns available in EPT	53
Figure 4.14	Details about shapes pattern	54
Figure 4.15	Download files of shapes pattern	55

Figure 5.1	A general view of the proposed graphics pattern classes	57
Figure 5.2a	Generating XML files for the Circle class	58
Figure 5.2b	Generating XML files for the Square class	59
Figure 5.2c	Generating XML files for the triangle class	60
Figure 5.2d	Generating XML files for the canvas class	61
Figure 5.2e	Generating XML files for all patterns embedded into the tool	62
Figure 5.3	The extraction operation by EPT	71
Figure 5.4	The main functions for EPT	72
Figure 5.5	Load XML file to memory by an XML document	72
Figure 5.6	The list of patterns available in EPT	73
Figure 5.7	List of classes for the shapes pattern	74
Figure 5.8	Load canvas XML file to memory by an xml document	74
Figure 5.9	List of fields and methods of canvas class	75
Figure 5.10	View details about methods	76
Figure 5.11	Load information xml file of pattern into EPT	76
Figure 5.12	Load xml files of classes for pattern into EPT	77
Figure 5.13	Backup patterns that store in PET	77
Figure 5.14	Find pattern available in EPT	78
Figure 5.15	The result of search operation	78
Figure 5.16	List of patterns available in EPT	79
Figure 5.17	details about shapes pattern	80
Figure 5.18	download files of shapes pattern	81

LIST OF TABLES

Table 3.1	Description tags displaying the general information of patterns	25
Table 3.2	General information about a certain pattern	27
Table 3.3	The description attributes of general info for a shape pattern	28
Table 3.4	description of Fields for a certain class	30
Table 3.5	The filed attributes for a certain class	30
Table 3.6	Tags for methods and functions for a certain class	32
Table 3.7	Attributes of methods and functions for a certain class	33
Table 5.2a	Documentation of fields and methods for the Circle class	63
Table 5.2b	Documentation of fields and methods for the Canvas class	66

Chapter 1

Introduction

1.1 Introduction

The process of saving pattern data, in order to be shared or to be published with other users this became relatively simple with the appearance and widespread use of the standard file format, which is known as XML (extensible markup language). An XML is a file format containing a data pattern generated and published on a server. The user may take this file and make use of the pattern through the use of a tool or mechanism that facilitates the process of reaching the data, extracting what

it contains in an understandable way, and then saving the data so that it can be retrieved whenever necessary.

Most of the suggested tools for saving data and information associated with a specific pattern depend on the user. More specifically, they depend on where the user gathers all the information and data associated with the pattern data that is fed into the tool so that the components can be displayed as an actual drawing. An example is the tool CO2P2S (Correct Object-Oriented Pattern-based Programming System), which is confined to taking all information from the user and then presenting it as a drawing illustrating the components that belong to the pattern generated. If a pattern in the shape of an XML were made available and published on one of the servers, one would not be able to save it or make use of it later if one wanted to make a high-quality work from it.

Since a pattern represents an active mechanism or tool that saves time in the preparation and achievement of several solutions, It represents a general reusable solution to commonly occurring problems in software design; however, as such, It is not a finished design that can be transformed directly into code language. It is a description or a template for how to solve a problem that can be used in many different situations. Object oriented design patterns typically show relationships and interactions between classes and objects, without specifying the final application of the classes or objects that are involved [1]. They are informal descriptions of tested solutions to recurring problems.

Most design tools have little or no support for documenting the presence and usage of patterns in code[5,6,7]. This tool used DOM technology (Document Object Model) to extract data of design pattern from xml files .

1.2 Problem Statement

The process of representing, organising and preserving these design patterns, and the process of retrieving them when needed has not been fully determined. Many studies have investigated these issues ; however, most of them, if not all, have been restricted to taking all information from the user and then transferring this information to an XML file from where it can be accessed by the user as a drawing displaying the components of the pattern generated. This research goes a step further in attempting to provide a tool that will organise and preserve data patterns in a way that can be easily referenced for future use [5,6,7].

Some researchers [5,6,7] have attempted to provide tools that describe generative design patterns embedded in XML files so that classes, relationships and interactions between patterns can be defined. In the proposed work, the author attempts design such a tool, namely, a tool that will enable a design pattern described by an XML file to be used as a standard for storing, sharing and transferring variant data and information between different applications and to variant users[5,6,7]. This tool will be applied by using an XML-based parser that compiles an XML file and transfers it into a formula that can be understood by all

users. The proposed tool will use HTML (Hypertext Markup language) technology to display the XML-file as an acceptable text desired by a user; further, suitable language such as ASP.NET will be selected to implement the tool.

1.3 Objectives

The aim of the present work is to provide a work mechanism for exchanging information and experience among users in different programming environments. A corollary to this goal is to create a tool will provide standard feedback to all designers so that reliable and efficient solutions to problems can be found that reduce time and effort

1.4 Thesis Structure

The remaining chapters of this thesis are organized as follows: Chapter 2 presents the literature background. This chapter describes some of the important methods ,which that have contributed to these research. presents a description of the proposed XML file format and explains all fields included in it in Chapter 3. presents the architecture of the tool in Chapter 4. presents a case study to show how the proposed tool is used throughout its stages of operation in Chapter 5.draws some concluding remarks and draws the scope for future work in Chapter 6.

Chapter 2

Background

2.1 Introduction

In this chapter, the author provides background information on some important subjects related to XML files that contribute to the present work. Much research has been carried out on the importance of design patterns as a tool that saves time in the design of a program and establishes helpful results in different platforms. However, the procedures for pattern representation, organization and saving and restoring have not been clearly stated. The current chapter will summarize this information by presenting the various procedures available.

2.2 Design pattern representation

Template solutions (Design Patterns) can be saved in two ways:

- On a web page in a UML Diagram display or in specific language such as VB or C#.
- In an XML file, where the tag structure of pattern display carries the data about the design pattern.

2.3 The Extensible Markup Language (XML)

XML is a general-purpose specification for creating custom languages. It is classified as an extensible language, because it allows the user to define the mark-up elements. XML's purpose is to aid information systems in sharing structured data, especially via the Internet [2]. The development of XML is carried out by an XML working group[2].

2.4 XML Advantages

- The user has a wide range of access to the data through the use of simple and varied applications.
- Data can be exchanged, edited and shared among users without complications.

- The file format is an easily-constructed open standard that allows the user to add tags to the data without interruption to the application.
- The format supports various applications in coping with different platforms.
- The format is internet oriented.

2.5 Document Object Model (DOM)

The Document Object Model (DOM) is an application programming interface (API) for valid HTML and well-formed XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated. In the DOM specification, the term "document" is used in the broad sense; increasingly, XML is being used as a way of representing many different kinds of information that may be stored in diverse systems, and much of this would traditionally be seen as data rather than as documents. Nevertheless, XML presents this data as documents, and the DOM may be used to manage this data [3].

The DOM is separated into 3 different parts / levels:

- Core DOM – the standard model for any structured document
- XML DOM – the standard model for XML documents
- HTML DOM – the standard model for HTML documents

The DOM defines the objects and properties of all document elements, and also the methods (interface) to access them [4].

2.6 Related Work

There are many tools used to parse XML file content data or information about design patterns. Some of these tools are listed below:

2.6.1 The CO2PS Tool

J. Anvik, S. Bromling, S. MacDonald, J. Schaeffer, D. Szafron, and K. Tan have designed and implemented a tool called the CO2P2S (Correct Object-Oriented Pattern-Based Programming System) that combines design patterns and object-oriented frameworks into a process for writing high-performance object-oriented programmes. The tool contains design pattern templates, which represent a family of solutions to a design problem. To select the most appropriate solution, the designer specializes the pattern template by specifying values for design pattern template parameters. The tool also has an interface to take values of the parameters and methods about the pattern from the user so as to generate an XML file within these parameters. Information about the design pattern is stored in this file; then, after generating a code, the user can modify the method or code line in

the code that has been generated, save this change, and make a back-up for its contents.

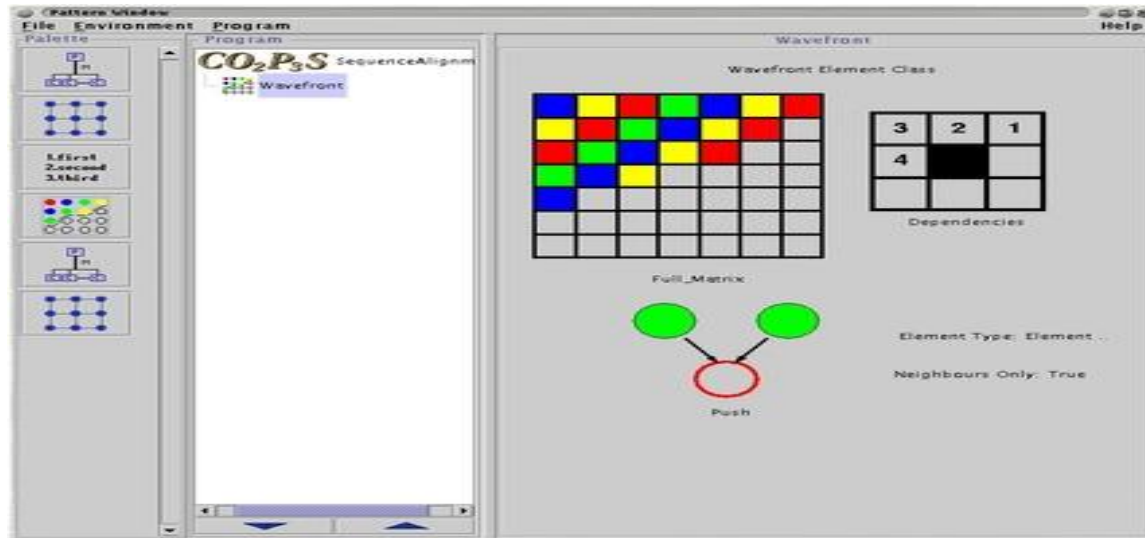


Figure 2.1: The main functions in the CO₂P₂S tool

Users have successfully run CO₂P₂S using native-threaded Java virtual machines (VMs) on the following operating systems:

- Linux
- Solaris
- SGI Irix

The result of this pattern are shown in Figure 2.2.

```

/**
    *Iteration op for a top edge node in a 4 point mesh.
    * @param east the node to the right
    * @param south the node below
    * @param west the node to the left
    * @parameter numNeighbours 4
    * @parameter boundary Non
    * @parameter boundary Horizontal
    * @editable
    */
    Public void topEdge(sp_meshElement east,sp_MeshElement
    south,sp_meshElement south , south,sp_meshElement west)
    {
    }

```

Figure 2.2 : Results from the CO2P2S tool

There is no option in this tool for the user to create an individualized pattern. When opening a programme, the user finds three pattern templates already installed: the Two-Dimensional Mesh, the Phases, and the Distributor. These three pattern templates, which can be used in various programmes [5][6][7], will appear whenever the user creates a new program or opens an existing one.

2.6.2 The Meta-CO2PS Tool

The same group of researchers who developed the CO2PS tool added a new feature to it that allows users to add new pattern templates, and remove and modify

them; this modified tool is called the Meta-CO2P2S. A limitation of this tool relates to the generation code options button that the user presses to generate a code about selected pattern templates. To address this limitation, the researchers have advised separating the tool generation design patterns from the framework that generates the code[5].

The same problem was found when users attempted to use the tool with the Windows operating system; the tool not successfully run on it [5][6].

2.6.3 The Design Pattern Framework TM 3.5

Released from Microsoft, this framework contains the most famous design patterns, such as the Gang of Four, Enterprise and service-oriented architecture(SOA). “SOA is a service-oriented architecture is essentially a collection of services. These services communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. Some means of connecting services to each other is needed”, in two editions: C# and VB.NET. The framework provides a multi-package filled with information and source codes. However, it does not accept any external design pattern from the user; instead, the templates are included in the tool [8].

2.6.4 The OMT Design Tool

The OMT tool represents an early work by Kramer and Prechelt [9] in which patterns are drawn and translated into Prolog rules. In the design and testing of this tool, the source code was parsed using the Paradigm Plus tool and converted into Prolog facts. Then, queries were run to determine what facts matched the rules, i.e., what patterns were present in the code. The parsing tool had significant limitations. It did not extract information that would have been useful, such as whether a method is a constructor, or whether a class is abstract or concrete. Further, the tool looked only at header files, and thus contained no information on the function call hierarchy. This made the recovery of patterns more difficult, since essential parts of the signature of a pattern that depend on these concepts could not be expressed. Nevertheless, the tool achieved reasonable recall and precision rates on source codes of moderate size (150–350 classes) [10].

2.6.5 The Tool of Florijn et al.

Florijn et al. [11] constructed a tool that was integrated in a Smalltalk environment . “ Smalltalk is an object-oriented, dynamically typed, reflective programming language. Smalltalk was created as the language to underpin the "new world" of computing exemplified . It was designed and created in part for

educational use [13] ”.The tool supported development at several abstraction levels, including that of Design Pattern. With this tool it is possible to create new classes as instances of patterns, connect existing classes with patterns and roles, and check whether pattern invariants are being upheld by classes in the code. The real-life test example cited involves about 150 classes.

2.6.6 The Reverse Engineering Framework of Balanyi and Ferenc

Balanyi and Ferenc [12] used a reverse engineering framework to convert C++ code into metadata (termed Abstract Semantic Graph), and express patterns in an XML-based language. They then performed a multi-step algorithm to identify candidate class structures, match them to the pattern descriptions, and filter out mismatches. One of their major contributions was to look at information from function bodies, such as function calls and object creations, in addition to the more traditional static structure.

2.7 Evaluation of the literature tools :

J. Anvik, S. Bromling, S. MacDonald, J. Schaeffer, D. Szafron, and K. Tan have designed and implemented a tool called the CO2P2S (Correct Object-Oriented Pattern-Based Programming System) that combines design patterns and object-oriented frameworks into a process for writing high-performance object-oriented programmes. The tool contains fixed design pattern templates, which represent a family of solutions to a design problem.

The problem was found when users attempted to use the tool with the Windows operating system; the tool not successfully run on it.

The same group of researchers who developed the CO2PS tool added a new feature to it that allows users to add new pattern templates, and remove and modify them; this modified tool is called the Meta-CO2P2S. A limitation of this tool relates to the generation code options button that the user presses to generate a code about selected pattern templates. To address this limitation, the researchers have advised separating the tool generation design patterns from the framework that generates the code[5].

The same problem was found when users attempted to use the tool with the Windows operating system; the tool not successfully run on it [5][6].

Released from Microsoft Framework TM 3.5 , this framework contains the most famous design patterns, in two editions: C# and VB.NET. The framework provides a multi-package filled with information and source codes. However, it does not accept any external design pattern from the user; instead, the templates are included in the tool [8].

From the literature, most of the previous presented tools have little support of patterns documentations or incompatible with Windows operating systems, while our proposed tool focuses on the documentation of design patterns, and to be compatible tool with all operating systems with no limit number of patterns to be stored in it.

2.8 Summary

A review of the available literature shows that most previous tools have little or no support for documenting the presence and usage of patterns in code. They also lack a method of retrieving these solutions when they are needed. In addition, because they depend on limited operating systems. The proposed tool will enable the user to save, document and return many designs that have already been produced by another user or designer. Sharing and exchanging these designs and

experiences will thus be possible, and will save the user much time and effort. These designs will also guarantee the affectivity and quality of the tool in real working fields.

Chapter 3

An overview of XML mapping of the Pattern Extractor Tool

3.1 Introduction

This chapter outlines the shape of the proposed XML file and describes how the main access to this tool will be set. The tool is based on presented data patterns, all of which can be stored in what is called a tag. This tag is the main base for any XML. It is also filed for downloading the data that is returned by using what is called the “document object model (DOM).” The DOM contains several

methods or functions that allow users access to any tag or any file from which they want to get information. The features associated with the model are defined below:

- **Name of pattern:** a meaningful name that reflects the knowledge embodied by the pattern.
- **Description of the problem:** the problem that the pattern addresses, i.e., the intent of the pattern.
- **Forces:** the constraints or issues that must be addressed by the solution.
- **Solution :** a description of the static and dynamic relationships among the components of patterns.

3.2 Presenting Data in XML Documents

The proposed method for saving any data in an XML file can be explained as follows: First, the information is saved in a node found in the XML file. Then, a tree structure of XML documents is formed that starts at "the root" and branches to "the leaves." Each node is placed in a field. This field will contain the data and have a particular name relating to the information it holds.

An example of an XML document is provided below. First, the self-describing and simple syntax is shown:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The first line is the XML declaration. It defines the XML version (1.0) and the encoding used (ISO-8859-1 = Latin-1/West European character set).

The next line describes the root element of the document (e.g., "this document is a note"):

```
<note>
```

The next four lines describe four child elements of the root (to, from, heading, and body):

```
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
```

Finally, the last line defines the end of the root element:

```
</note>
```

3.3 The Tree Structure of XML Documents

The elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree. All XML documents must contain a root element, which is "the parent" of all other elements, and all elements can have sub elements (child elements):

```

<root>

  <child>

    <subchild>.....</subchild>

  </child>

</root>

```

The terms "parent," "child," and "sibling" are used to describe the relationships between elements. Parent elements have children. Children on the same level are called siblings (brothers or sisters). All elements can have text content and attributes (just as in HTML). Example:

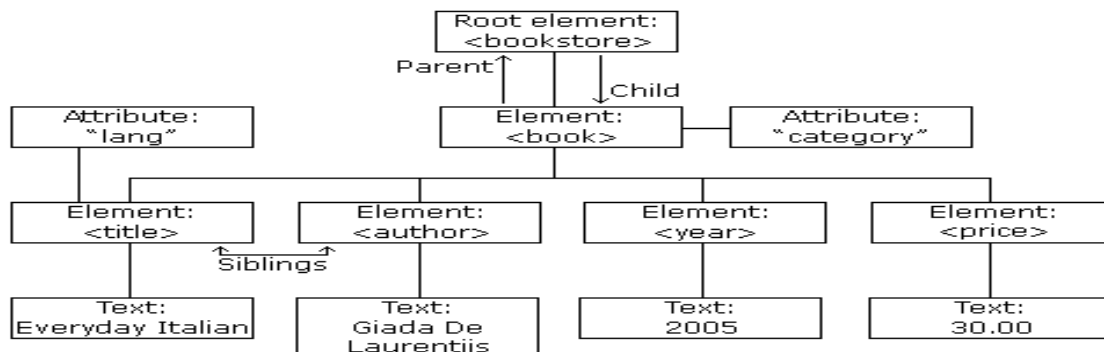


Figure 3.1 : A general view of an XML structure

Figure 3.1 represents one book in the XML below:

```
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

The root element in the example is `<bookstore>`. All `<book>` elements in the document are contained within `<bookstore>`. The `<book>` element has four children: `<title>`, `<author>`, `<year>`, `<price>` [3].

3.4 Presented Design Patterns in XML Documents

Any pattern presented as a solution to any programming problem must have some elements and specifications that define it. Some of these elements and specifications are defined in XML files. These are the elements and specifications already represented inside the nodes, which are the basic ingredient for XML files.

3.5 The Suggested Structure of XML Files

The suggested structure for presenting the pattern inside an XML file is divided into many levels as shown below:

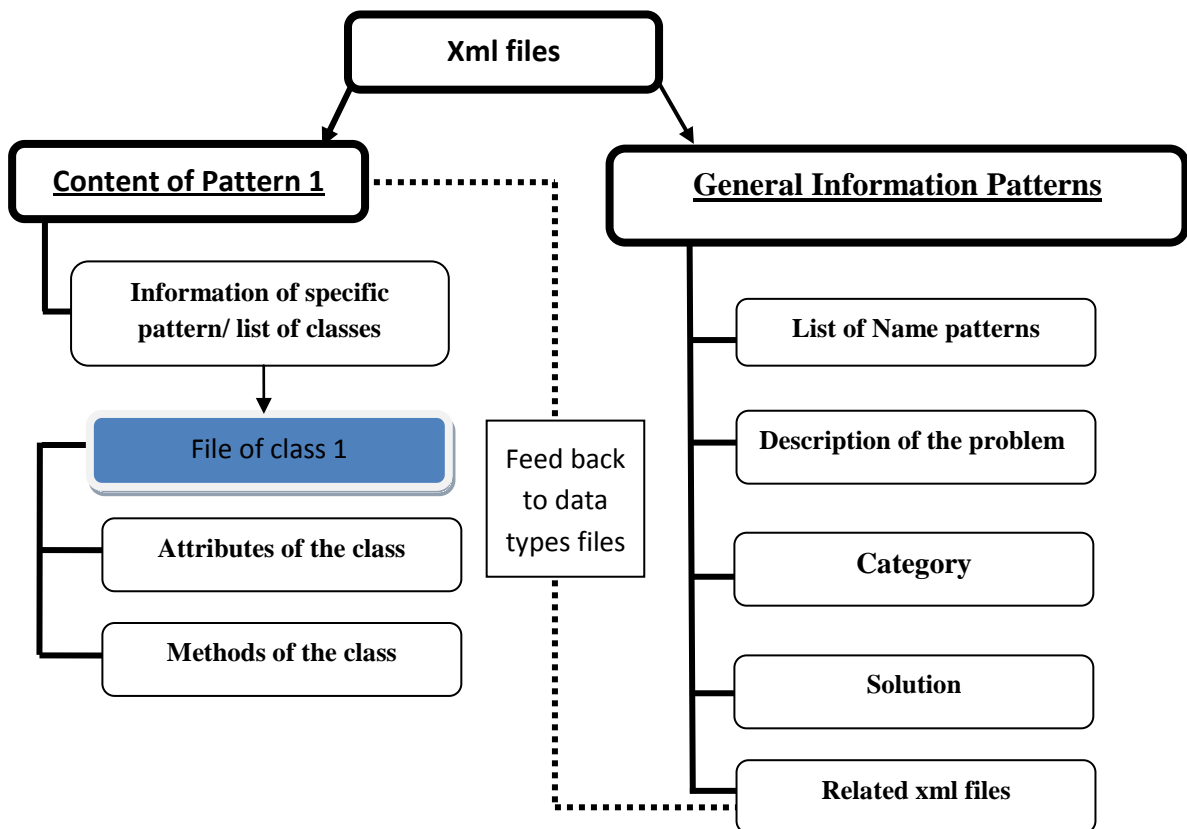


Figure 3.2 : The structure of pattern information in XML files

A number of XML files will be proposed one of which will represent all the patterns found in the tool. This file will be named the general pattern data file and it will involve the following :

- 1) the digital identity for defining the pattern that will be related to the name saved on the hard desk;
- 2) the classification of the pattern;
- 3) a description of how the pattern works.

These fields in the xml file have been represented as follow:

```
<?xml version="1.0" encoding="utf-8" ?>
<pattern number="">
<pattern1>
<id></id>
<name> </name>
<description> </description>
<category> </category>
</pattern1>
<pattern2>
<id></id>
<name> </name>
<description> </description>
<category> </category>
</pattern2>
</pattern>
```

These fields in the XML file are illustrated by description tags, as shown in the table below:

Table 3.1 Description tags displaying the general information of patterns

Name of tag	Description
Id	Describes the physical name of the class on the hard disk
Name	Indicates the name of the pattern
Description	Describes the pattern
Category	Indicate the category of pattern
Number	Indicate the number of patterns in the tool

There are three patterns embedded in the tool, as shown in the xml file below:

```
<?xml version="1.0" encoding="utf-8" ?>
<pattern number="3" >
<pattern1>
<id>1</id>
<name>Drawing shape Pattern</name>
<description>This Pattren to drawing different Shapes</description>
<category>Graghical Pattern</category>
</pattern1>
<pattern2><id>2</id>
```



```

<name>www</name>

<description>ww111w</description>

<category>weeeee</category>

</pattern2>

<pattern3>

<id>3</id>

<name>zzzz</name>

<description>zzzz</description>

<category>xxxxx</category>

</pattern3>

</pattern>

```

The shape of the represented data inside this file will be presented first. Then, the data belonging to each pattern will be represented separately and will be provided with a proposed file containing that pattern's special data. Table 3.2 illustrates the description tags of general information in an XML file for a certain pattern (i.e., a shape pattern).

These fields in the xml file have been represented as follow:

```

<?xml version="1.0" encoding="utf-8" ?>

<pattern>

<namepatt> </namepatt>

<description> </description>

```

```

<solution>

</solution>

<catpattern> </catpattern>

<classess number="" >

<class1 id=""> </class1>

<class2 id=""> </class2>

<class3 id=""> </class3>

<class4 id=""> </class4>

</classess>

</pattern>

```

Table 3.2: General information about a certain pattern

Name of tag	Description
namepatt	Describes the name of pattern
description	Describes of the problem
solution	The solution to this problem by this pattern
catpattern	Category of pattern
classes	The list of classes that contain this patterns

For example, if the user wants to represent special data for a pattern for geometric shapes, the XML file will be presented as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<pattern>
<namepatt>Dreawing Shapes Pattern </namepatt>
<description> The problem is summerised in drawing different shapes and change it
colors and sizes</description>
<solution>Attach additional responsibilities to an object dynamically. Decorators
provide a flexible alternative to subclassing for extending functionality.drawing alot of
shapes in different colors and different size</solution>
<catpattern> The Decorator Pattern </catpattern>
<classess number="4" >
<class1 id="1"> Class Canvas </class1>
<class2 id="2">Class Circle</class2>
<class3 id="3">Class Square</class3>
<class4 id="4">Class Triangle</class4>
</classess>
</pattern>
```

Table 3.3 : The description attributes of general information for a shape pattern

Attributes	Description
Number	Displays the number of classes
id	Displays the name of the class on the hard disk

Afterwards, a special file will be created for each pattern that represents the pattern method and another file will be created for each field. The conception of the XML file is shown in the tables below. The first table, Table 3.4, illustrates the description tags of XML fields for a certain class. Presented fields in XML file :

```
<?xml version="1.0" encoding="utf-8" ?>
<filedclass>
  <classname> </classname>
  <description>
    </description>
  <fields>
    <filed1 type=" " access=" " dtype=" "> </filed1>
    <filed2 type="" access=" " dtype=""> </filed2>
    <filed3 type=" " access=" " dtype=""> </filed3>
    <filed4 type="" access=" " dtype=""> </filed4>
    Description   Name of tag
    Describes the name of the class   Classname
    Describes the class job           Description
    Displays the list of fields       Fields
    <filed5 type=" " access=" " dtype=""> </filed5>
    <filed6 type="" access=" " dtype=""> </filed6>
  </fields>
</filedclass>
```

Table 3.4 : description of Fields for a certain class

Name of tag	Description
Classname	Describes the name of the class
Description	Describe the class job
Fileds	Displays the list of fileds

Table 3.5 : the filed attributes for a certain class

Attributes	Description
Type	Displays the field's data type
Access	Displays the permission to access this field
Dtype	Displays the type of variant

The data can be represented as follow:

```
<?xml version="1.0" encoding="utf-8" ?>
<filedclass>
<classname>Class Caven </classname>
<description>
  Class Canvas - a class to allow for simple graphical drawing on a canvas. This is a
  modification of the general purpose Canvas
</description>
<fileds>
<filed1 type="Canvas.CanvasPane" access="private" dtype="static">canvas</filed1>
<filed2 type="JFrame" access="private" dtype="">frame</filed2>
<filed3 type="Graphics2D" access="private" dtype="">graphic</filed3>
<filed4 type="CanvasPane" access="public" dtype="">canvas</filed4>
<filed5 type="Color" access="private" dtype="">backgroundColour</filed5>
<filed6 type="Image" access="private" dtype="">canvasImage</filed6>
</fileds>
</filedclass>
```

Presented methods in XML file :

```
<?xml version="1.0" encoding="utf-8" ?>
<methods>
<method1 ftype=" " access=" " return=" " name=" ">
<desc> </desc>
<descreturn> </descreturn>
<parameter value=""></parameter>
</method1>
<method2 ftype="" access=" " return="" name=" ">
<desc></desc>
<descreturn></descreturn>
<parameter value="">
```

```

<par1 ptype=" " desc=" "></par1>
</parameter>
</method2>
<method3 ftype="0" access="public" return="boolean" name="drawImage">
<desc></desc>
<descreturn>
</descreturn>
<parameter value="">
<par1 ptype=" " desc=" "> </par1>
<par2 ptype=" " desc=" "> </par2>
<par3 ptype=" " desc=" "> </par3>
</parameter>
</method3>
</methods>

```

Table 3.6: Tags for methods and functions for a certain class

Name of tag	Description
method1,2,3,--	Describes the start of method information
desc	Describes the job method
descreturn	Displays the return variable
parameter par1, 2,...	Display the start information about parameters Par1 - the name of parameter

Table 3.7 : Attributes of methods and functions for a certain class

Attributes	Description
method1 ftype	Displays the type of method, e.g., static
Access	Displays the permission to access of this method: public or private
Return	Displays the type data of returned; if empty, assume no returned data.
name	Displays the name of patterns
parameter value=""	Value=0, the no parameters
par1 ptype=""	Displays the type of parameter
desc	Describes the parameters

The data can be represented as follow:

```
<?xml version="1.0" encoding="utf-8" ?>
<methods>
<method1 ftype="static" access="public" return="Canvas" name="getCanvas">
<desc>Factory method to get the canvas singleton object.</desc>
<descreturn>return object the created</descreturn>
<parameter value="0"></parameter>
</method1>
<method2 ftype="0" access="public" return="0" name="draw">
<desc>Draws a given shape onto the canvas.</desc>
<descreturn></descreturn>
<parameter value="1">
<par1 ptype="java.awt.Shape" desc="the shape object to be drawn on the
canvas">shape</par1>
</parameter>
```



```

</method2>
<method3 ftype="0" access="public" return="boolean" name="drawImage">
<desc>Draws an image onto the canvas.</desc>
<descreturn>returns boolean value representing whether the image was completely
    loaded</descreturn>
<parameter value="1">
<par1 ptype="java.awt.Image" desc="the Image object to be displayed">image</par1>
<par2 ptype="int" desc="co-ordinate for Image placement">x</par2>
<par3 ptype="int" desc="co-ordinate for Image placement">y</par3>
</parameter>
</method3>
</methods>

```

The final shape of the XML file will be as follows:

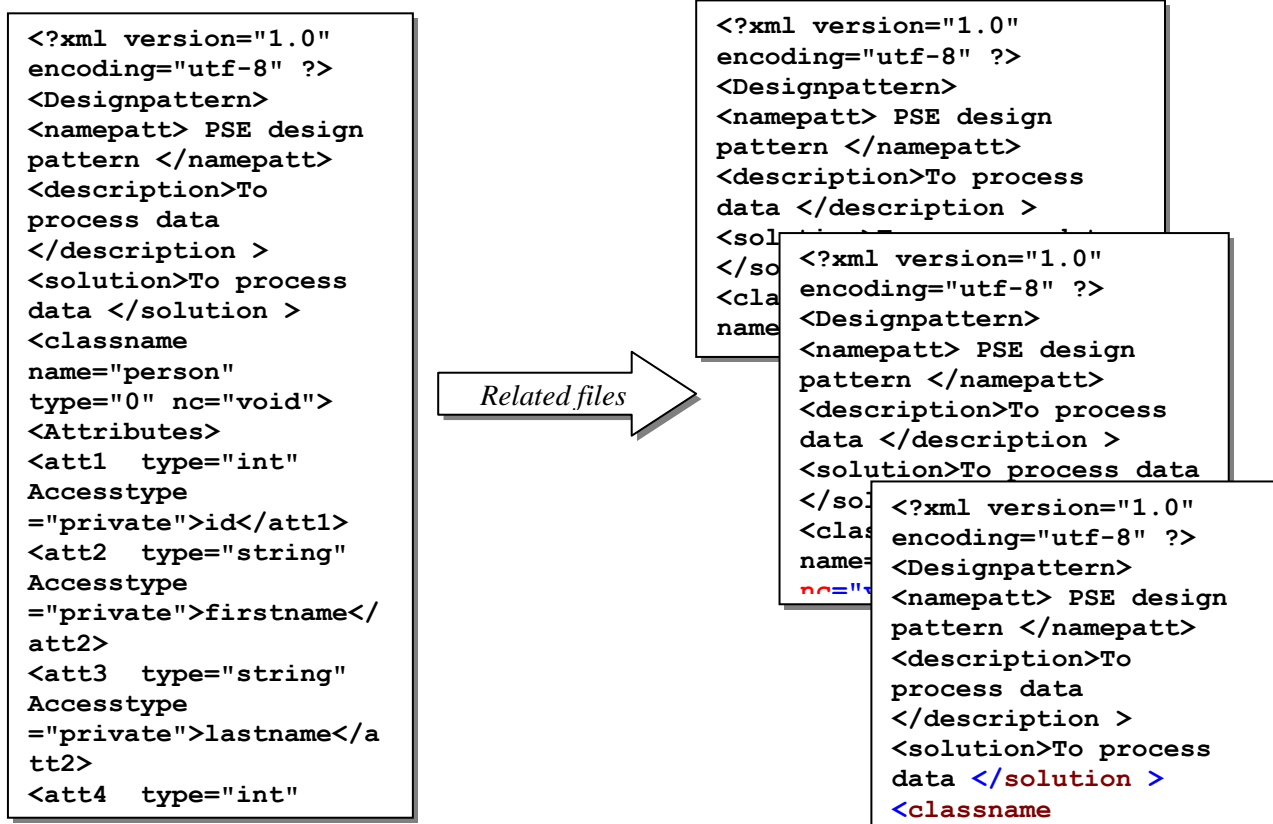


Figure 3.3: the input files for the proposed tool

3.6 The Structure of the XML mapping of the proposed function

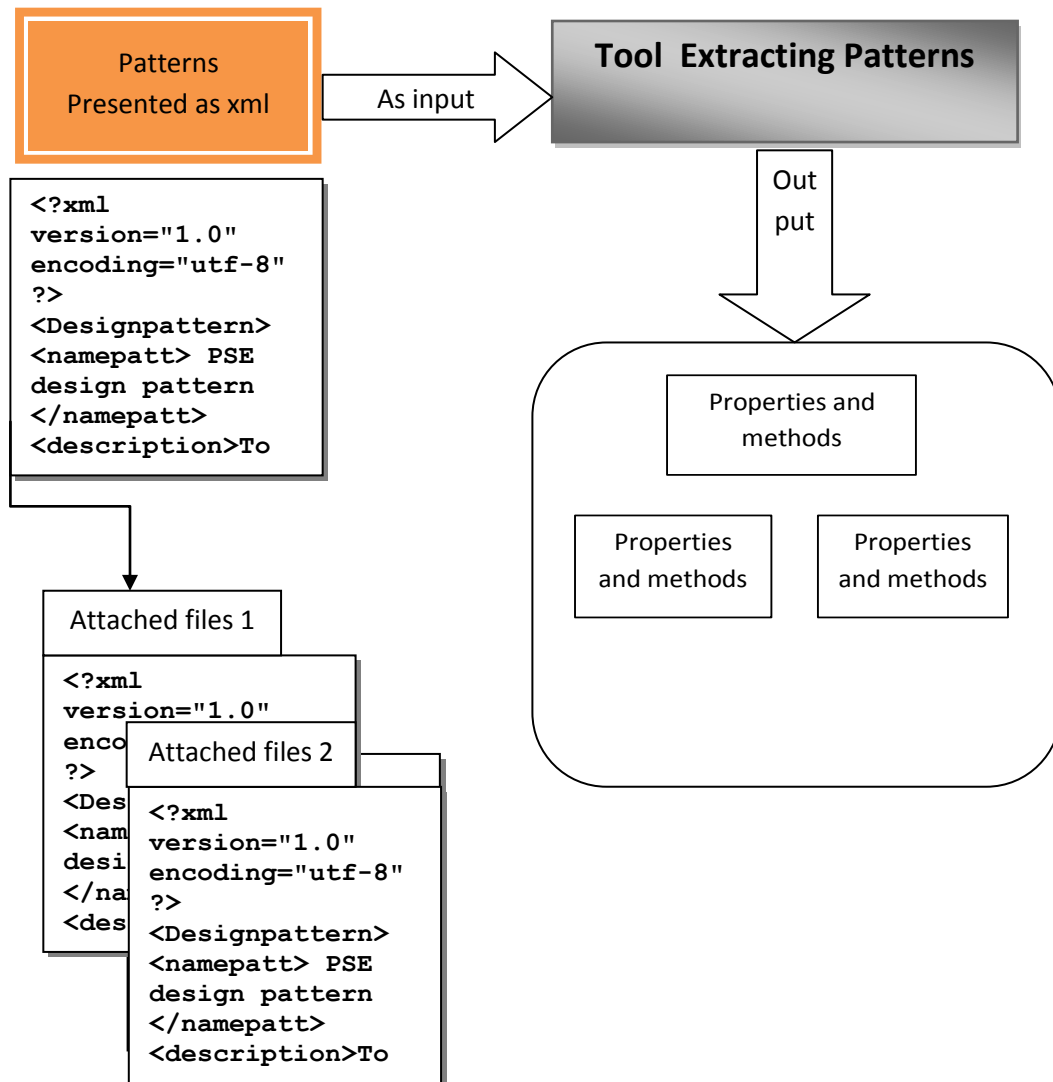


Figure 3.4: General conceptual overview for the proposed tool.

3.7 The XML Generated Files

The proposed XML files have been generated (produced) by using an editor, available from Microsoft, for editing XML files included in Visual Studio.NET 2003. There are many editors that can be used for editing and producing XML files such as front page and visual studio 2003. Also, an access program with converting capability is available that can convert tables created by Microsoft Access into XML files.

Chapter 4

The EP Tool Architecture

4.1 Introduction

In this chapter, Extracting Patterns Tool (EPT) will be construct which have two application. The first one belongs to the user and its function ,in order to connect to a server to return and save the data the user needs. The second application belongs to the server itself and it runs the process of saving and returning the data.

Figure 4.1 illustrates the UML (Unified Modeling Language) class diagram

for the first application, and Figure 4.2 illustrates the UML class diagram for the second application.

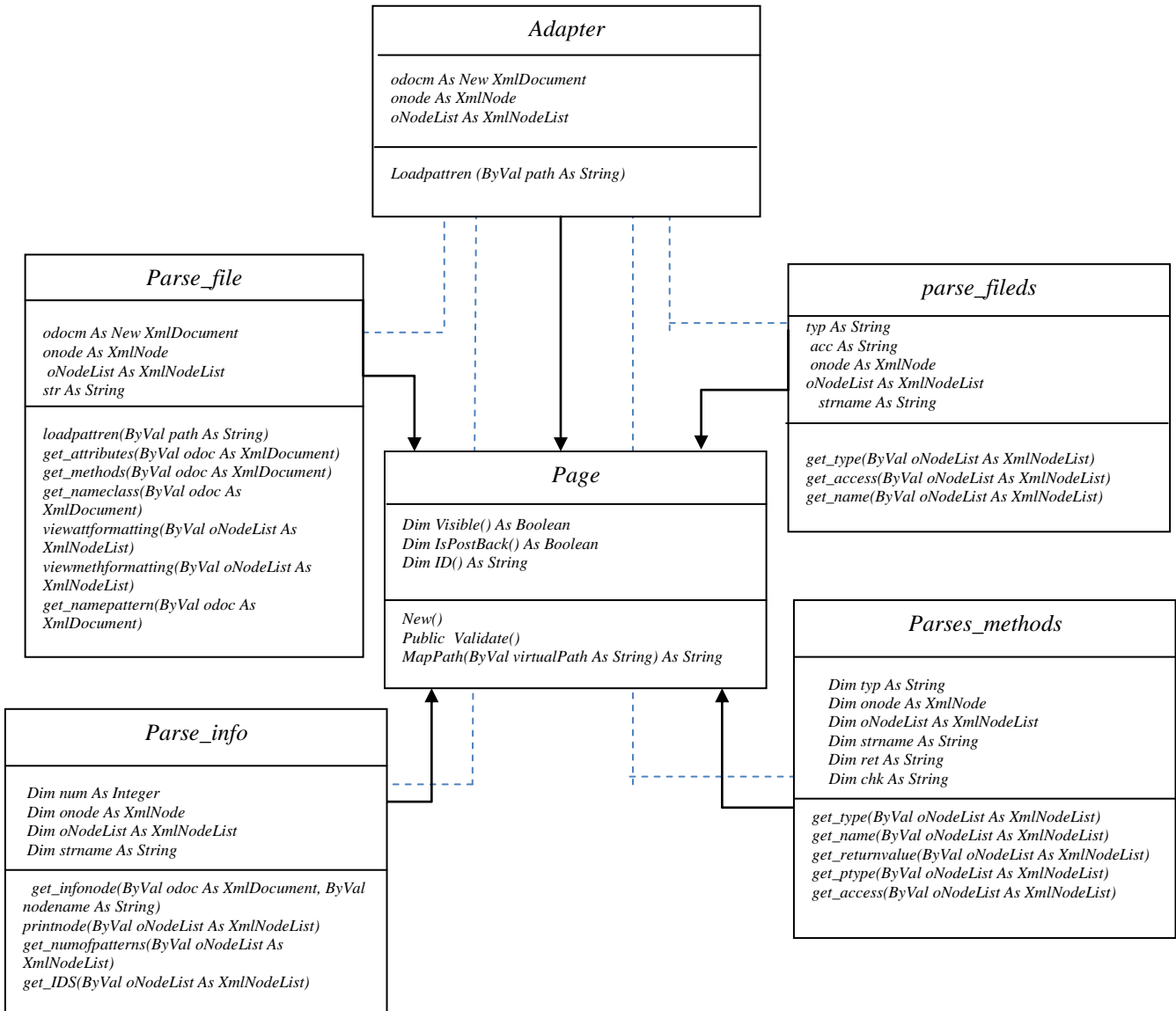


Figure 4.1: UML class diagram for the EPT.

In the Appendix, we describe the continent table, which contains the following information:

- **Class name:** identifies the class.
- **Field summary:** lists the attributes that the class contains and their types.
- **Methods contained:** lists the methods that the class contains and the types of their returned values.

4.2 The Documentation of Classes for the Client Side

a. class name: adapter

This class contain methods that load the XML file to an XML data document to start the processing. The adapter class is related to the page class through the “inherits” attribute, and it contains two methods. For more details, view Appendix A1 which illustrates the documentation of the adapter class.

1. ***Loadpatteren:*** *this method will receive the path of any xml file and load it to start the process operation.*
2. ***Mappath:*** *this method gets the path of xml files to be returned, and send it to load and to start the processing operation.*

b. class name: parse_file

This class contains some methods for parsing XML files on the client side, saving patterns and retrieving files according to the request of the user. The parse_file class is related to the adapter class through the “use” attribute and to the page class by the “inherits” attribute. The following list display some of methods in this class. For more details, see Appendix A2 which illustrates the documentation of the parse_file.

1. *get_attributes*: this method retrieves the attributes from the xml attributes file and stores the returned values in an xml node list.
2. *get_methods*: this method retrieves the methods from the xml methods file and stores the returned values in an xml node list.
3. *viewattformatting*: this method displays the final formatting of the display, the shape of methods and the attributes. The XML is returned as string and printed in the frame view.

d. class name: parse_info

This class contains some methods for parsing the XML information file on the client side and retrieving the information about all patterns saved in the tool. The following list displays some of methods in this class. For more details, see

Appendix A3 which illustrates the documentation of the parse_info.

1. *get_IDS*: this method returns the sequence number of pattern.
2. *get_infonode*: this method returns the data stored in a specific node.
3. *get_numofpatterns*: this method returns the information stored in a node about the number of patterns in the tool.
4. *get_nameclass*: this class retrieves the class name presented in the xml file.

e. class name: parse_fields

This class contains some methods for parsing the XML fields file of a specific class and retrieving the information about these fields. This class is related to adapter class through the “use” attribute and to the page class through the “inherits” attribute. The following list displays some of methods in this class. For more details, see Appendix A4 which illustrates the documentation of the parse_fields.

1. *get_IDS*: this method returns the id of a pattern to load it and to start to return its information.
2. *get_infonode* :this method retrieves data of a specific node to return full information about this node.
3. *Printnode* : this method helps to print data of a specific node
4. *get_numofpatterns* : this method returns the total number of patterns.

f. class name: parse_methods

This class contains some methods for parsing the XML methods file for a specific class and retrieving the information about these methods. This class is related to the adapter class through the “use” attribute and to the page class through the “inherits” attribute. The following list displays some of methods in this class. For more details, see Appendix A5 which illustrates the documentation of the parse_methods.

1. *get_name* : this method returns the name of the method.
2. *get_access* : this method retrieves the permission of this method (private, public).
3. *get_returnvalue* : this method determines if the return contains data or is void.
4. *get_description* : this method gets a description of the work done.

g. class name: mfiles

This class contains some methods for manipulating the XML files that contain the class and information files. This class is related to the adapter class through the “use” attribute and to the page class through the “inherits” attribute. The following

list displays some of the methods in this class. For more details, see Appendix A6 which illustrates the documentation of mfiles.

1. *Copyclass* : this method saves the classes file in the new directory.
2. *Checkilenames* : this method returns the list of class names.
3. *Updateinfofile* : this method adds a new information pattern to the information file.
4. *Createnewdirec*: this method creates the new folder.

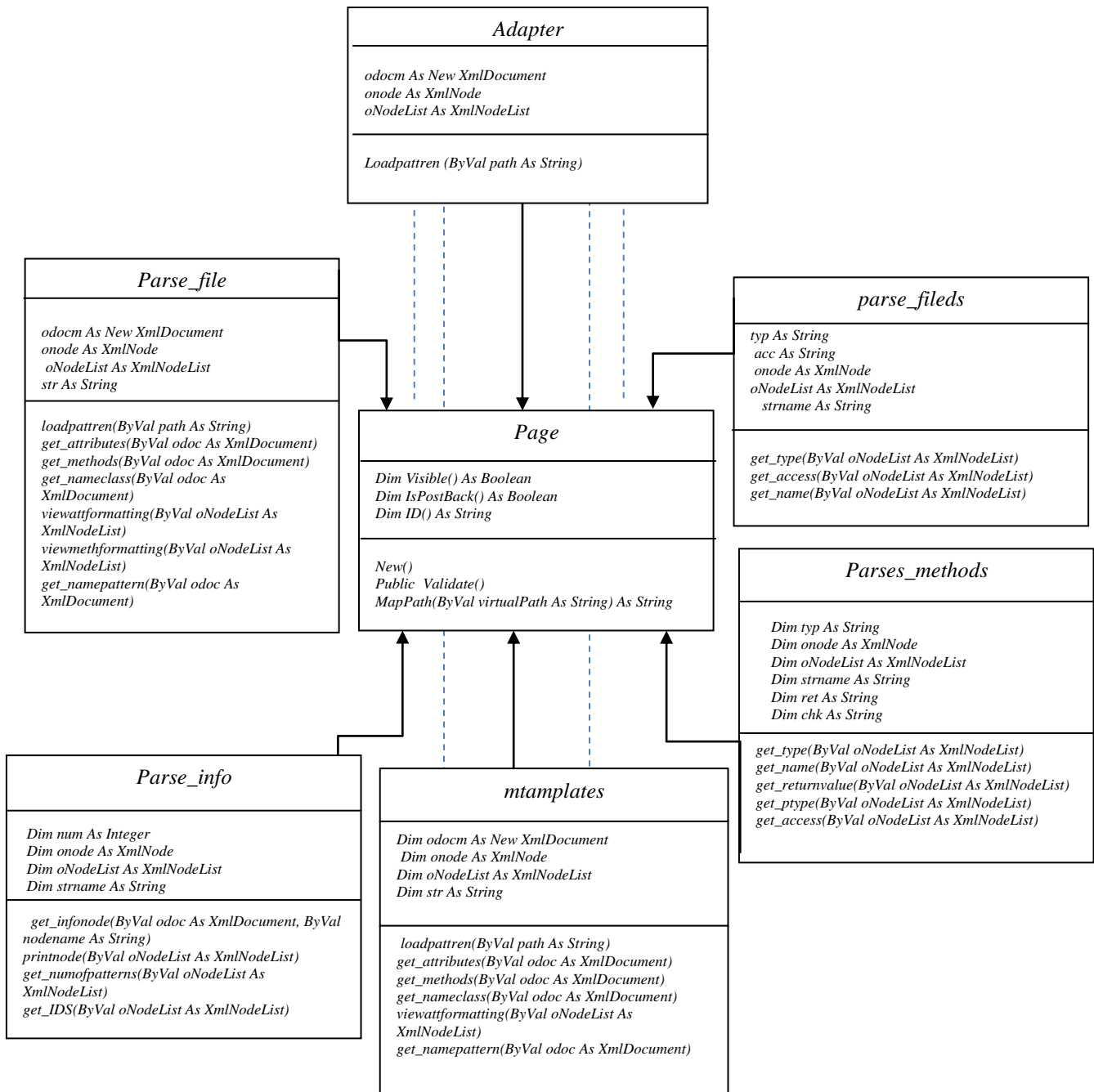


Figure 4.2: UML class diagram for patterns store server side.

4.3 The Documentation of Classes for the Patterns Stored

a. class name: `mtemplates` class

This class contains some methods for managing all server operations. These operations include receiving patterns from the user, checking and adding them to stored templates, making backup patterns, viewing available patterns, and deleting patterns. This class is related to the adapter class through the “use” attribute and to the page class through the “inherits” attribute. The following list display some of methods in this class. For more details, see Appendix A7 which illustrates the documentation of the `mtemplates`.

1. *Saverecivedpattern*: this method saves the pattern on the server.
2. *AddnewPattern*: this method adds new patterns to storage templates.
3. *BackupTemplatesonserver*: this method makes backup templates.
4. *Viewavalibaltemplates*: this method retrieves all templates available on storage templates.

4.4 Tool Interface System (tool description work)

This tool works to parse files containing patterns information presented in an xml file. The following description displays the steps of this tool and how patterns are parsed with this tool.

The main window is shown as Figure 4.3. As can be seen, this window contains the interface to navigate this tool. The interface displays the following functions:

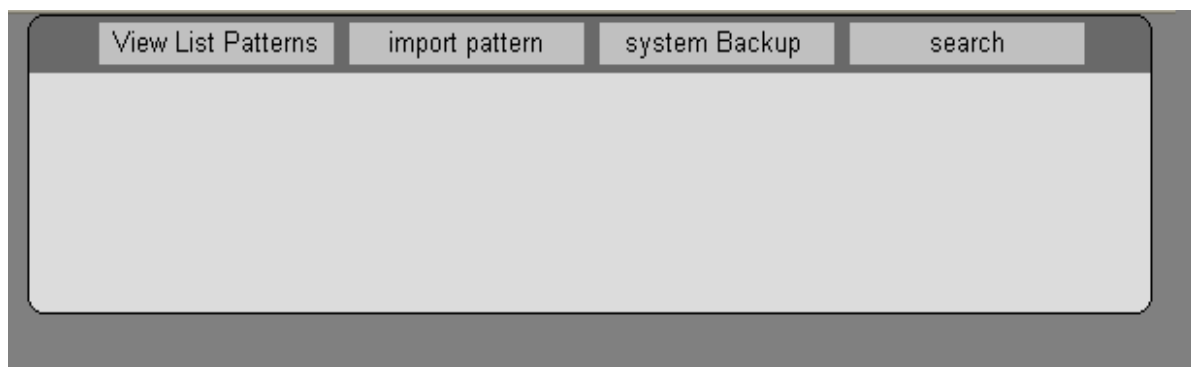


Figure 4.3 : The main functions for EPT

4.4.1- View list patterns:

When the user clicks this function, the following result is obtained:

The total number of patterns available in EPT.

Name of pattern.

Category of pattern.

Description of the pattern use.

Link to obtain all details about the selected patterns; when one is selected the result is the list of classes of this pattern. This operation is shown in Figure 4.4.

View List Patterns import pattern system Backup search			
The number of patterns avalibal in this tool = 4			
Name of Pattern	Description	Category	details..
Drawing shape Pattern	This Pattren to drawing different Shapes	Graghical Pattern	View More Details..
r	rest	r	View More Details..
t	t	t	View More Details..
e	e	e	View More Details..

Figure 4.4 : List of patterns available in EPT

name of pattern	number of classes	problem	soulution
Dreawing Shapes Pattern	4	The problem is summerised in drawing different shapes and change it colors and sizes	Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality. drawing alot of shapes in different colors and different size
List Of classes			
-	name of class		details..
1	Class Canvas		view full class..
2	Class Circle		view full class..
3	Class Square		view full class..

Figure 4.5: List of classes for the determined pattern

When the user presses “view full class,” he or she obtains a summary of the fields and methods included in this class. At this point, the user can start processing an XML document.

Class Caven

Class Canvas - a class to allow for simple graphical drawing on a canvas.

Field Summary :	
Canvas.CanvasPane	<u>canvas</u>
JFrame	<u>frame</u>
Graphics2D	<u>graphic</u>
CanvasPane	<u>canvas</u>
Color	<u>backgroundColour</u>
Image	<u>canvasImage</u>

[field Details](#)

Method Summary :	
Canvas	<u>getCanvas()</u> Factory method to get the canvas singleton object.
void	<u>draw()</u> Draws a given shape onto the canvas.
boolean	<u>drawImage()</u> Draws an image onto the canvas.

Figure 4.6: summary of methods and fields for the determined pattern

Clicking on “methods detail” or “fields detail” will obtain the result shown in Figure 4.7.

Methods Detail:	Back ...
<pre>getCanvas public Canvas getCanvas() Factory method to get the canvas singleton object. Returns : return object the created</pre> <hr/> <pre>draw public void draw() Draws a given shape onto the canvas.</pre> <p><u>Parameters:</u> shape-the shape object to be drawn on the canvas.</p>	

Figure 4.7 : View details about methods

4.4.2- Import patterns:

In this function, we will import our patterns to save them in EPT by downloading the patterns from the stored patterns and uploading them to EPT. Figure 4.8 shows the result.

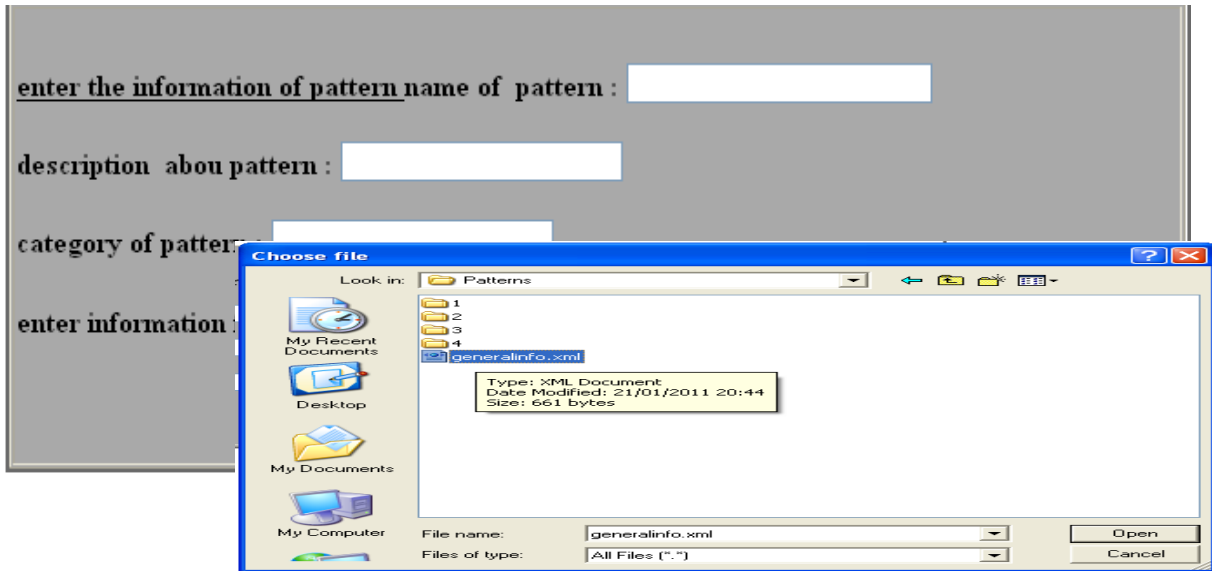


Figure 4.8 : Load information xml file of patterns into EPT

This tool divides the operation of uploading patterns into two parts:

1)- upload the pattern information file, as shown in Figure 4.8, and

include the name of the pattern, its description, and the

category of the pattern.

2)- upload xml files of classes (fields and methods xml files), as shown in Figure 4.9.

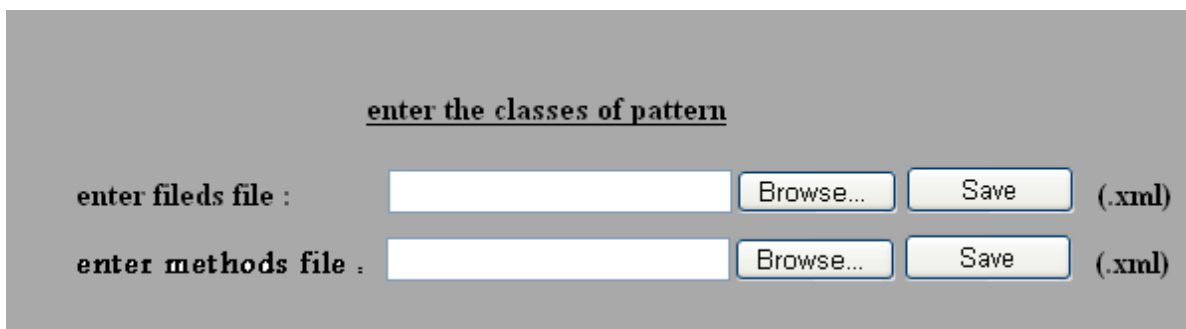
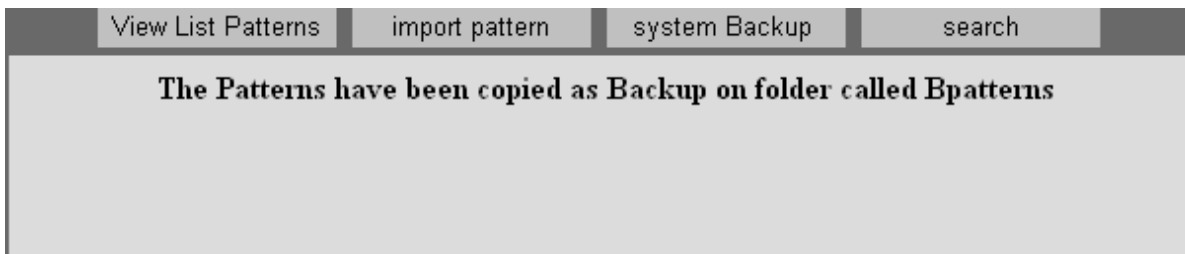


Figure 4.9 : Load xml files of classes for patterns into EPT

4.4.3 - System backup:

In this function, we will make a backup for all patterns available in EPT, and the new path of the backup will appear to the user after the operation is completed.

Figure 4.10 : Backup patterns stored in EPT



4.4.4- Search about pattern:

Use this step to find patterns saved by entering the name of the pattern and obtaining details about that pattern.

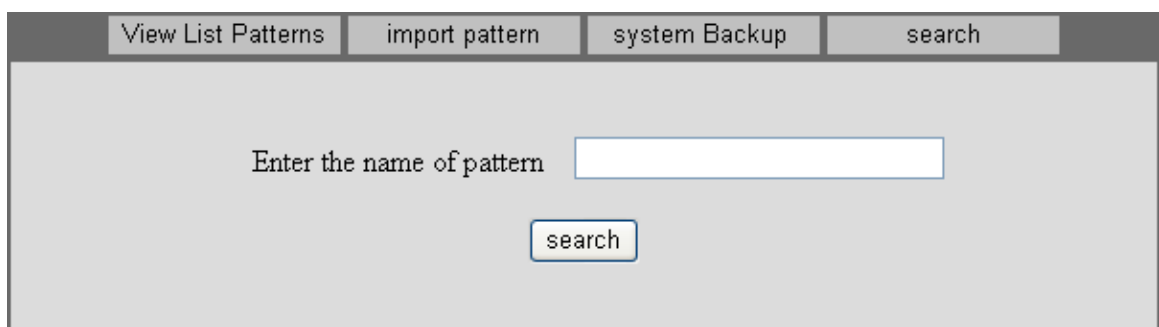


Figure 4.11 : Finding pattern availability in EPT

The result obtained from this function is as follows:

View List Patterns import pattern system Backup search			
the result of search =1			
Name of Pattern	Description	Category	details..
Drawing shape Pattern	This Pattren to drawing different Shapes	Graghical Pattern	View More Details..

Figure 4.12 : The result of a search operation

4.4.5- Dealing with stored patterns:

The storage for the proposed pattern will be as follows: The downloaded patterns by the users will be presented in the storage of the patterns.

This storage will enable the users to look at the classes , search for and download any patterns or a particular pattern. Figure 4.13 illustrates the result.

the number of patterns avalibal is : 3

Name of Pattern	Description	Category	details..	download files..
Drawing shape Pattern	This Pattren to drawing different Shapes	Graghical Pattern	View More Details..	download files..
wwww	ww11w	weeee	View More Details..	download files..
<i>zzzz</i>	<i>zzzz</i>	xxxxx	View More Details..	download files..

Figure 4.13: List of patterns available in EPT

To load any pattern from storage, press the related link and many choices will appear, as shown in Figure 4.14.

name of pattern	number of classes	problem	soulation
Drawing Shapes Pattern	4	The problem is summarised in drawing different shapes and change it colors and sizes	Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality. drawing alot of shapes in different colors and different size
List Of files			
<u>download information file</u>			
-	name of class	save class	
1	Class Canvas	<u>Save methods file ..</u> <u>Save fileds file ..</u>	
2	Class Circle	<u>Save methods file ..</u> <u>Save fileds file ..</u>	
3	Class Square	<u>Save methods file ..</u> <u>Save fileds file ..</u>	
4	Class Triangle	<u>Save methods file ..</u> <u>Save fileds file ..</u>	

Figure 4.14: Details about shapes pattern

To load the pattern files, we first download the information files for the chosen file and then download the files belonging to the classes involved in creating these patterns. Downloading can be done as shown in Figure 4.15.

name of pattern	number of classes	problem	soulution
Drawing Shapes Pattern	4	The problem is summarised in drawing different shapes and change it colors and sizes	Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality. drawing alot of shapes in different colors and different size

List Of files

[download information file](#)

-	name of class	save class
1	Class Canvas	Save methods file .. Save files file ..
2		Save methods file ..
3		
4		

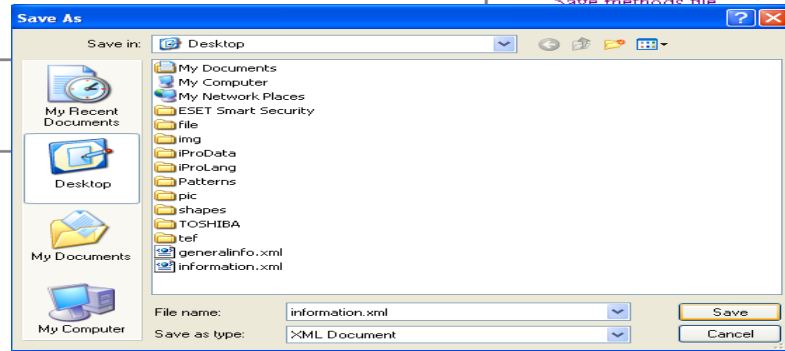


Figure 4.15: download files of shapes pattern.

Chapter 5

Case Study

This case study illustrates the concept of extracting design patterns in an XML file using the EP tool. The examples considered are geometric shapes patterns, such as “Circle,” “Square,” and “Triangle.” In the first stage, the author will present these patterns in an XML file, which will be imported into the proposed tool. The proposed pattern will be a Graphics_pattern and it will be used for drawing the geometric shapes (i.e., circle, square, and triangle) in different dimensions and places, and with different colors. Thus, four classes will be generated: . “Circle,” “Square,” “Triangle,” and “Canvas.”

In this first stage also, a method for representing these classes will be presented in XML files and then these files will be put into the proposed EP tool

whose purpose is to translate them and produce patterns that are easy to understand by users. The final shape for these classes, the proposed functions for each class, the relation between the classes and how to put these functions in their own files will be shown. Figure 5.1 illustrates the general view for the proposed graphics pattern classes.

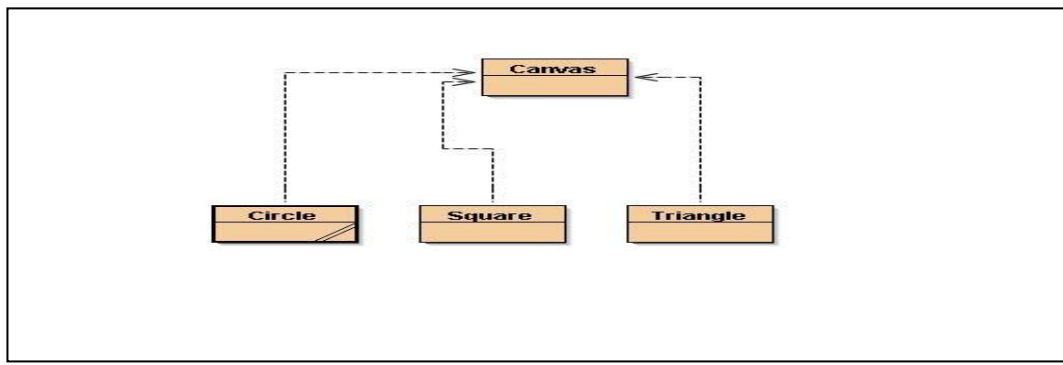


Figure 5.1: A general view of the proposed graphics pattern classes

5.1 - Generating XML Files

In the example above, the operation can be divided into three stages. The first stage involves generating the XML files. Each class will contain three XML files: the first one is for fields, the second is for methods and last one is for information.

Figure 5.2a illustrates the generation of special xml files for the Circle class.

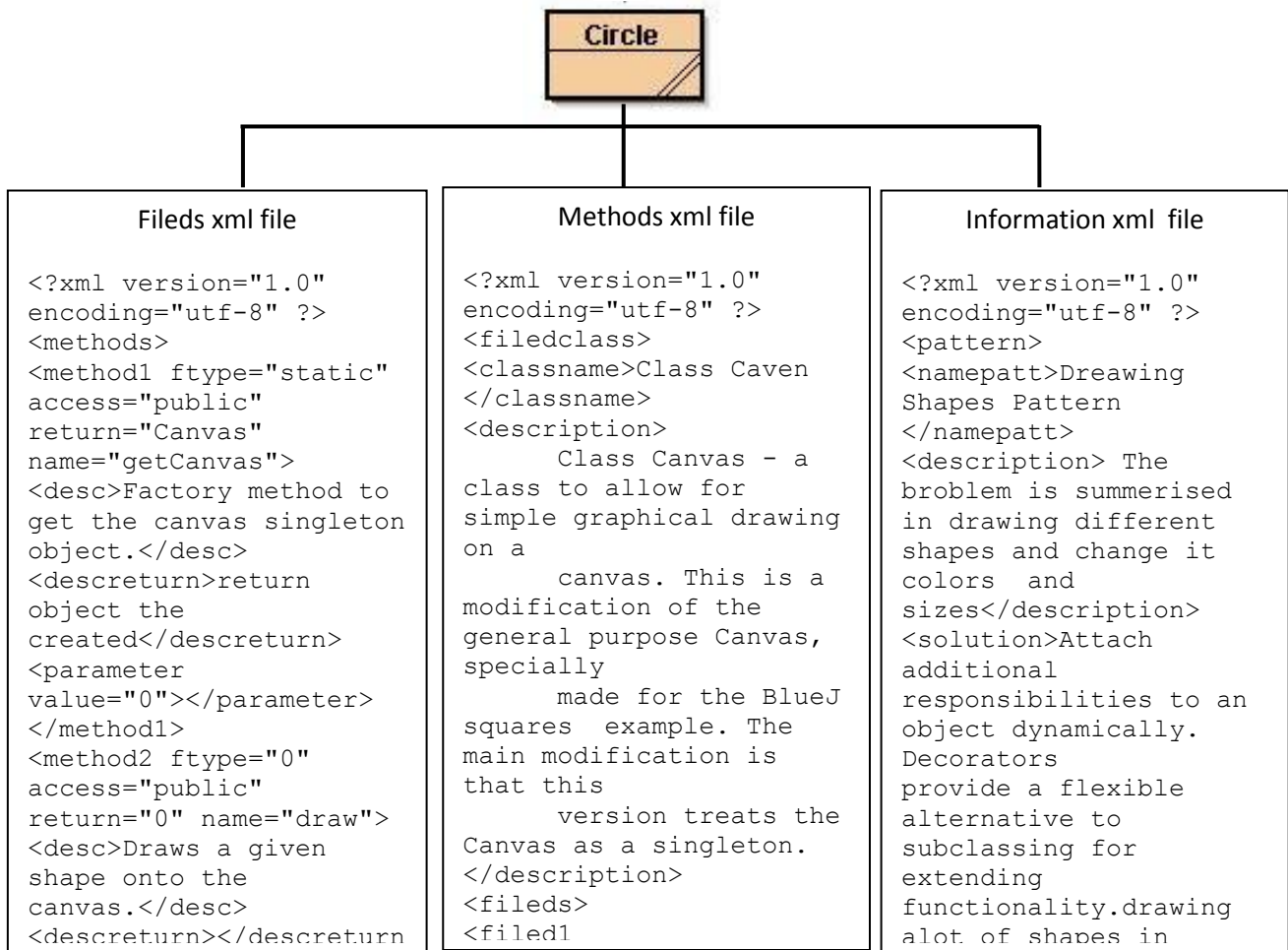


Figure 5.2a : Generating XML files for the Circle class

Figure 5.2b illustrates the generation of special XML files for the Square class.

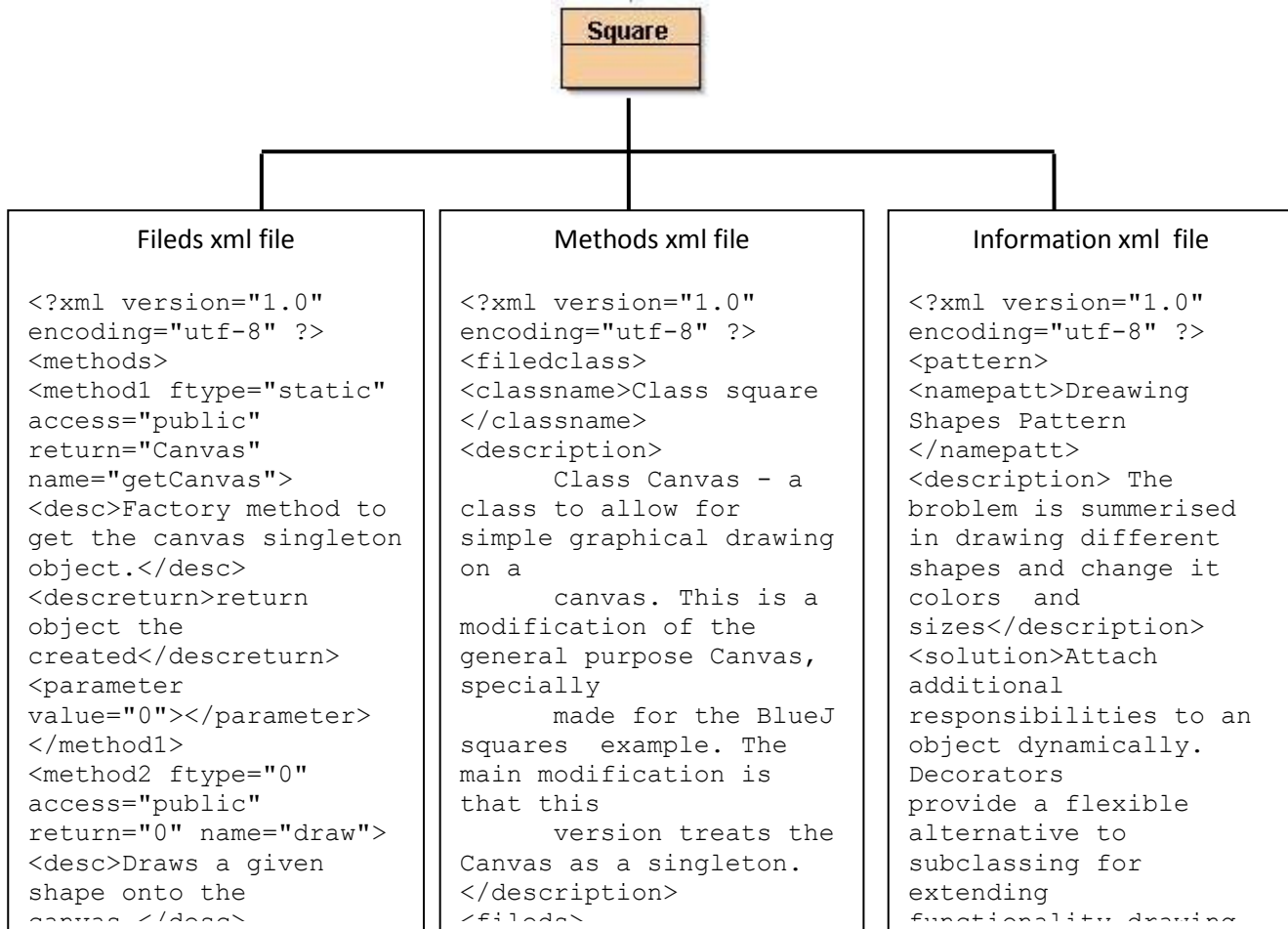


Figure 5.2b : Generating XML files for the Square class

Figure 5.2c illustrates the generation of special XML files for the triangle class.

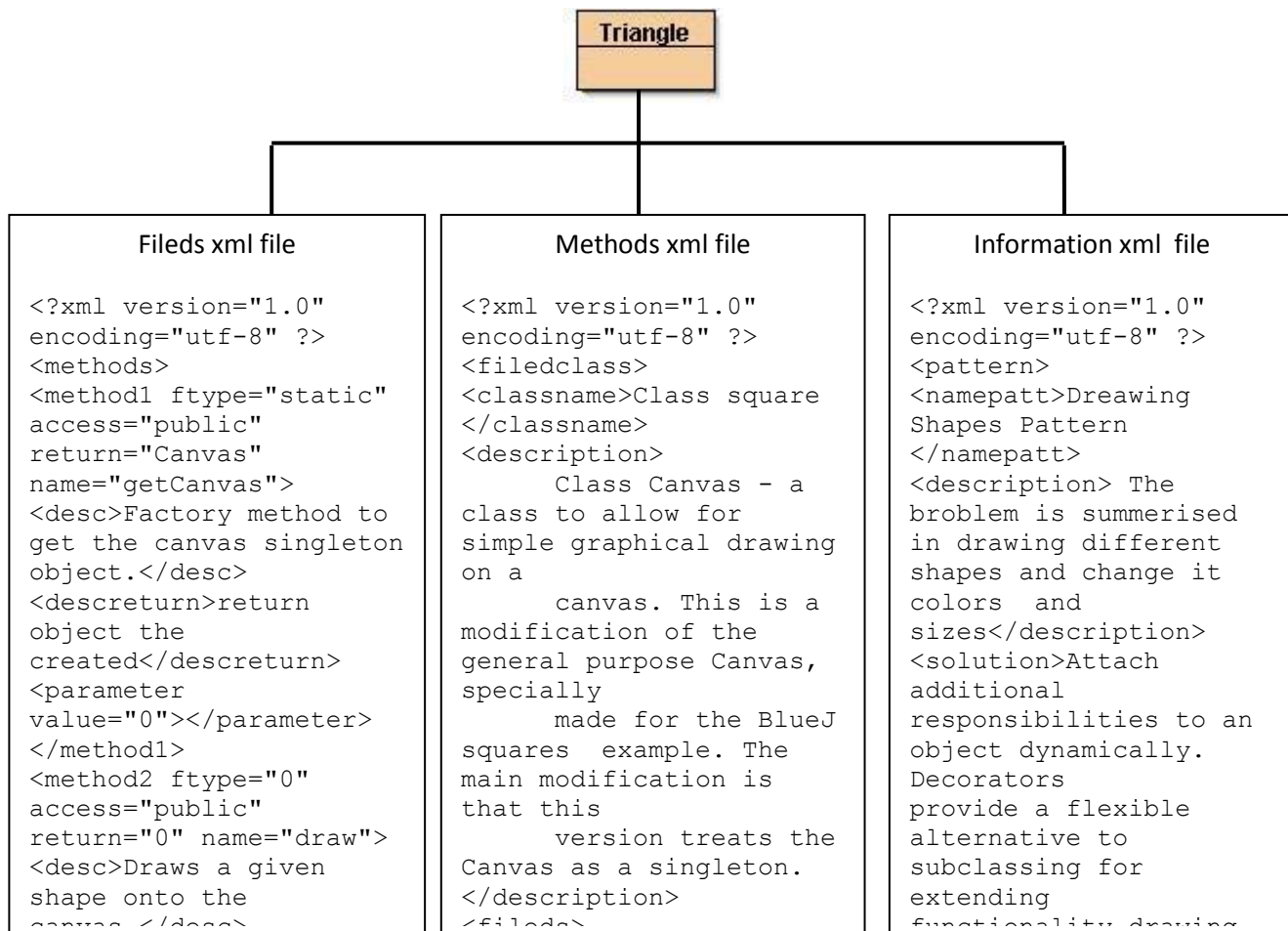


Figure 5.2c: Generating XML files for the triangle class

Figure 5.2d illustrates the generation of special XML files for the canvas class.

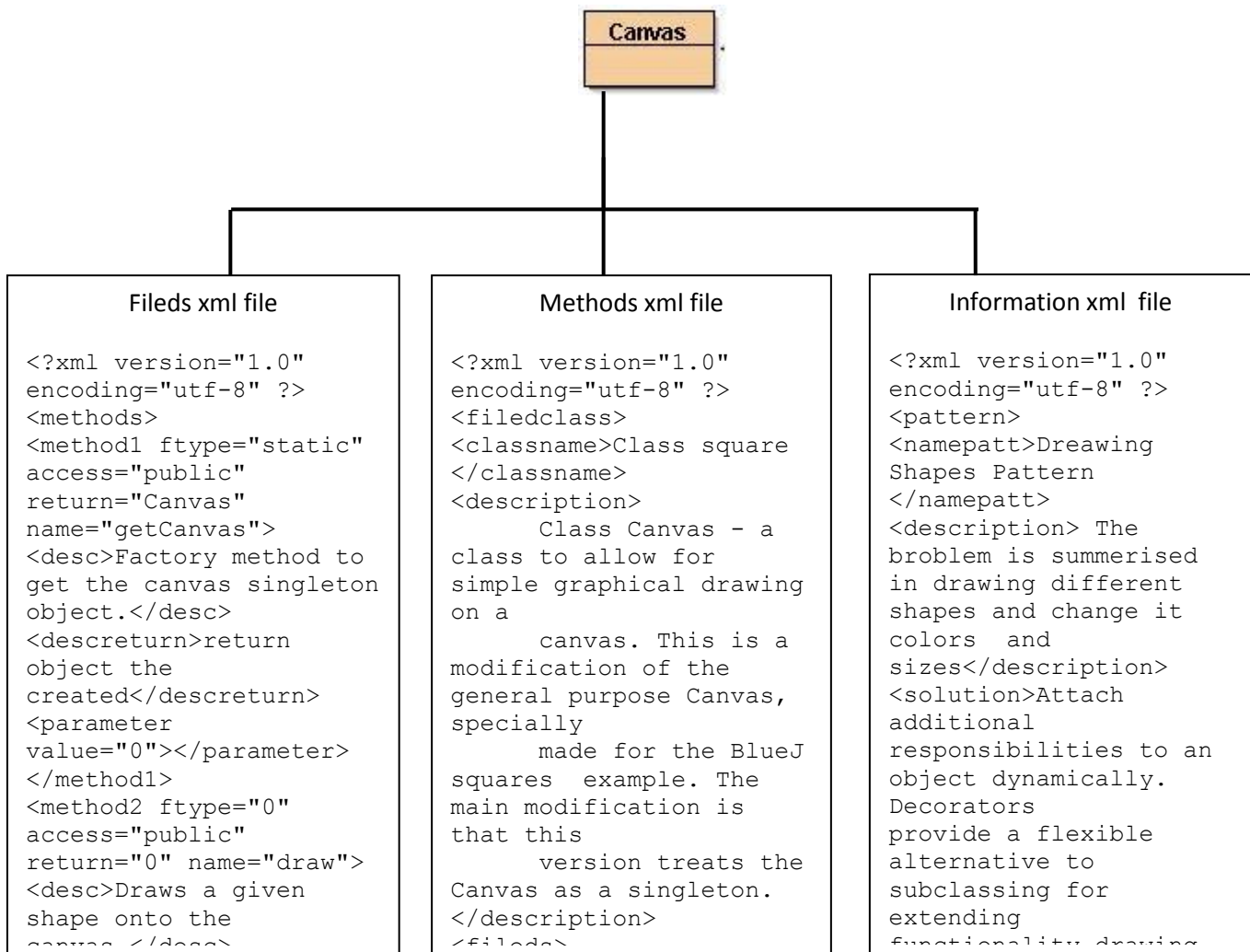


Figure 5.2d: Generating XML files for the canvas class

There is also a general file containing information about all the patterns available in this tool. Figure 5.2e illustrates how this general XML file of pattern information will be generated.

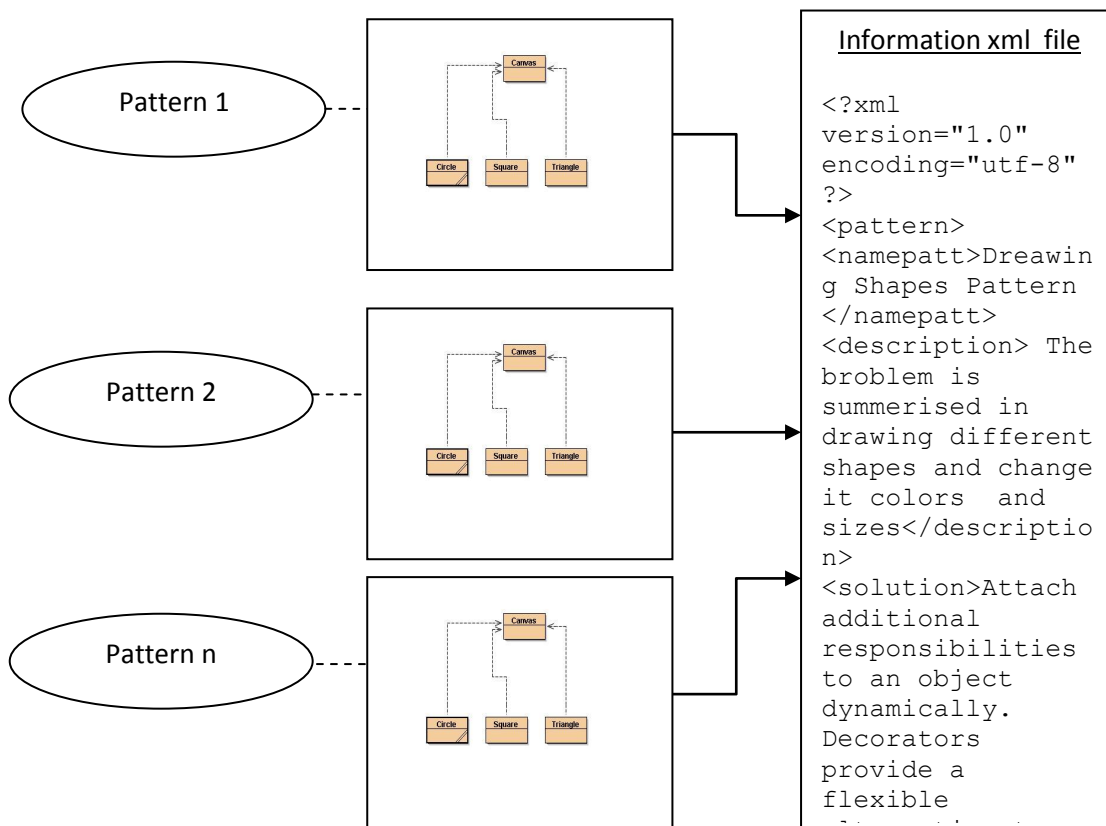


Figure 5.2e : Generating XML files for all patterns embedded into the tool

5.2 - Parsing and Documentation

In the second stage, the classes are documented as XML files, which are then parsed. The proposed mechanism is divided into two parts: the first one is fields and the second is methods.

First, all the fields and methods for the first class in this pattern, i.e., the Circle class, are represented.

Class Circle

public class **Circle**

A circle that can be manipulated and that draws itself on a canvas.

Table 5.2a Documentation of fields and methods for the Circle class

Constructor Summary	
Circle()	Create a new circle at default position with default color.
Method Summary	
void	changeColor(String newColor) Change the color.
void	changeSize(int newDiameter) Change the size to the new size (in pixels).
void	moveDown() Move the circle a few pixels down.
void	moveHorizontal(int distance) Move the circle horizontally by 'distance' pixels.
void	moveLeft() Move the circle a few pixels to the left.
void	moveRight() Move the circle a few pixels to the right.

64	
void	moveUp() Move the circle a few pixels up.
void	<u>moveVertical</u> (int distance) Move the circle vertically by 'distance' pixels.
void	<u>slowMoveHorizontal</u> (int distance) Slowly move the circle horizontally by 'distance' pixels.
void	<u>slowMoveVertical</u> (int distance) Slowly move the circle vertically by 'distance' pixels.

For files, the file format will be presented in XML file as follows:

```

<?xml version="1.0" encoding="utf-8" ?>
<filedclass>
<classname>Class circle </classname>
<description>
  A circle that can be manipulated and that draws itself on a canvas.
</description>
<fileds>
<filed1 type=" int " access="private" dtype=" "> diameter </filed1>
<filed2 type="int" access="private" dtype=""> xPosition </filed2>
<filed3 type="int" access="private" dtype=""> yPosition</filed3>
<filed4 type=" String " access=" private " dtype=""> color </filed4>
</fileds>
</filedclass>

```

For methods, the file format will be presented in XML file as follows:

```

<?xml version="1.0" encoding="utf-8" ?>
<methods>
<method1 ftype="String " access=" public" return=" 0" name="changeColor">
<desc>Change the color. Valid colors are "red", "yellow", "blue", "green", "magenta"
and "black".
</desc>
<descreturn> </descreturn>
<parameter value="1">
<par1 ptype=" String" desc="Change the color. "> newColor</par1>
</parameter>
</method1>
<method2 ftype="0" access="public" return="0" name="changeSize ">
<desc>Change the size to the new size (in pixels) .</desc>
<descreturn></descreturn>
<parameter value="1">
<par1 ptype="int " desc="Change the size to the new size (in pixels). Size must be >=
0.">newDiameter </par1></parameter>
</method2>
<method3 ftype="0" access="public" return="0" name="moveDown">
<desc> Move the circle a few pixels down.</desc>
<descreturn> </descreturn>
<parameter value="0"></parameter>
</method3>
<method4 ftype="0" access="public" return="0" name=" moveHorizontal ">
<desc> Move the circle horizontally by 'distance' pixels.</desc>
<descreturn> </descreturn></methods>

```


Second, all fields and methods for the second class in this pattern, i.e., the Canvas class, are represented.

Table 5.2b Documentation of fields and methods for the Canvas class

Field Summary	
Canvas.CanvasPane	<u>canvas</u>
Method Summary	
void	draw (Shape shape) Draws a given shape onto the canvas.
boolean	drawImage(Image image, int x, int y) Draws an image onto the canvas.
void	drawLine(int x1, int y1, int x2, int y2) Draws a line on the Canvas.
void	drawString(String text, int x, int y) Draws a String on the Canvas.
void	erase(Shape shape) Erases a given shape's interior on the screen.
void	eraseOutline(Shape shape) Erases a given shape's outline on the screen.
void	eraseString(String text, int x, int y) Erases a String on the Canvas.
void	fill(Shape shape) Fills the internal dimensions of a given shape with the current foreground colour of the canvas.
Color	getBackgroundColour() Returns the current colour of the background
static Canvas	getCanvas() Factory method to get the canvas singleton object.
Font	getFont() Returns the current font of the canvas.
Color	getForegroundColour() Returns the current colour of the foreground.
Dimension	getSize() Returns the size of the canvas.
boolean	isVisible()

	Provides information on visibility of the Canvas.
void	setBackgroundColour(Color newColour) Sets the background colour of the Canvas.
void	setFont(Font newFont) changes the current Font used on the Canvas
void	setForegroundColour(Color newColour) Sets the foreground colour of the Canvas.
void	setForegroundColour(String colourString) Sets the foreground colour of the Canvas.
void	setSize(int width, int height) Sets the size of the canvas.
void	setVisible(boolean visible) Sets the canvas visibility and brings canvas to the front of screen when made visible.
void	wait(int milliseconds) Waits for a specified number of milliseconds before finishing.

For fileds, the file format will be presented in XML file as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<filedclass>
<classname>Class Caven </classname>
<description>
  Class Canvas - a class to allow for simple graphical drawing on a
  canvas. This is a modification of the general purpose Canvas, specially
  made for the squares example. The main modification is that this
  version treats the Canvas as a singleton.
</description>
<fileds>
<filed1 type="Canvas.CanvasPane" access="private" dtype="static">canvas</filed1>
<filed2 type="JFrame" access="private" dtype="">frame</filed2>
<filed3 type="Graphics2D" access="private" dtype="">graphic</filed3>
<filed4 type="CanvasPane" access="public" dtype="">canvas</filed4>
<filed5 type="Color" access="private" dtype="">backgroundColour</filed5>
<filed6 type="Image" access="private" dtype="">canvasImage</filed6>
</fileds>
</filedclass>
```

For methods, the file format will be presented in XML file as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<methods>
<method1 ftype="static" access="public" return="Canvas" name="getCanvas">
<desc>Factory method to get the canvas singleton object.</desc>
<descreturn>return object the created</descreturn>
<parameter value="0"></parameter>
</method1>
<method2 ftype="0" access="public" return="0" name="draw">
<desc>Draws a given shape onto the canvas.</desc>
<descreturn></descreturn>
<parameter value="1">
<par1 ptype=" Shape" desc="the shape object to be drawn on the
canvas">shape</par1>
</parameter>
</method2>
</methods>
```

The proposed information file for this pattern can then be displayed.

```
<?xml version="1.0" encoding="utf-8" ?>
<pattern>
<namepatt>Dreawing Shapes Pattern </namepatt>
<description> The broblem is summerised in drawing different shapes and change it
colors and sizes</description>
<solution>Attach additional responsibilities to an object dynamically. Decorators
provide a flexible alternative to subclassing for extending functionality.drawing alot of
shapes in different colors and different size</solution>
<catpattern> The Decorator Pattern </catpattern>
<classess number="4" pathc="/Patterns/Shapes/">
<class1 id="1"> Class Canvas </class1>
<class2 id="2">Class Circle</class2>
<class3 id="3">Class Square</class3>
<class4 id="4">Class Triangle</class4>
</classess>
</pattern>
```

In this second stage, the XML files are parsed by entering them as input into the EP tool. This tool will take the information from the user, who will import the patterns downloaded from patterns stored and embed them into EPT. Figure 5.1.2 illustrates the extraction patterns from XML files.

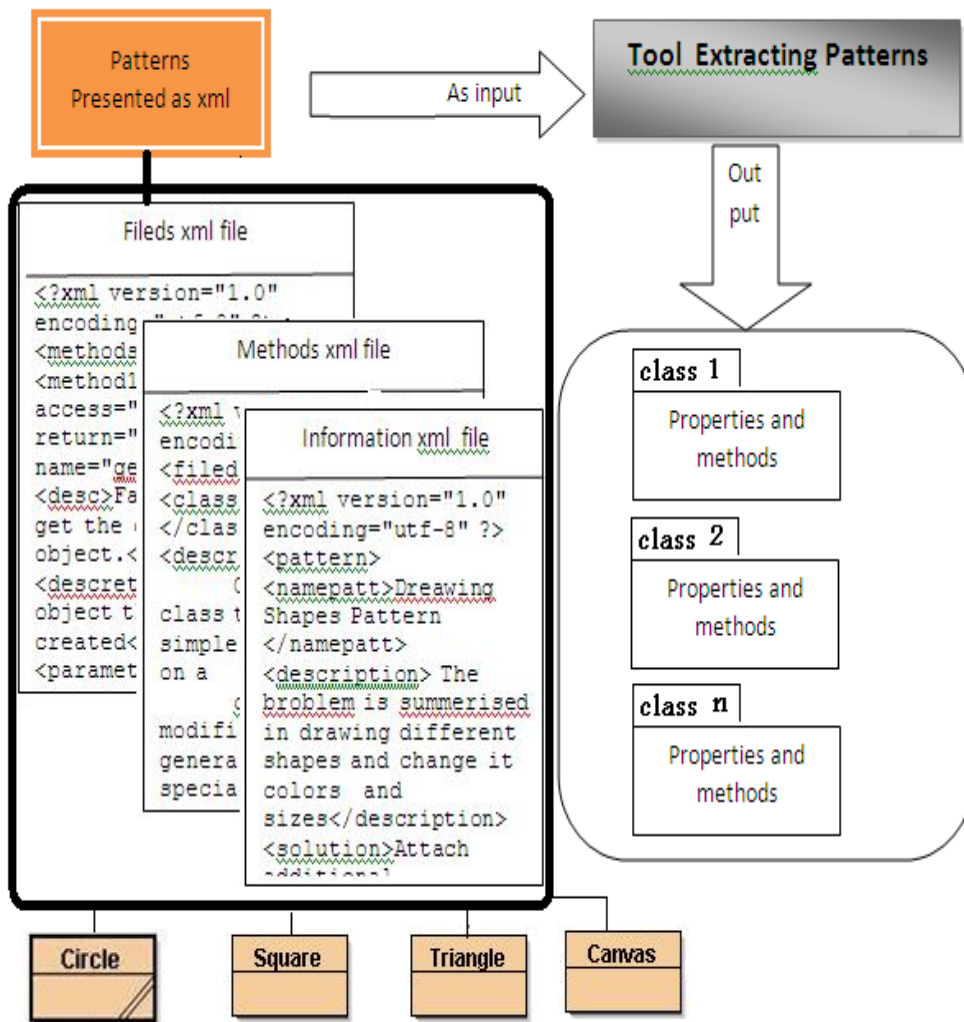


Figure 5.3: The extraction operation by EPT

The tool will parse XML files and display the data by the functions available in EPT. Figure 5.4 shows the main functions available in the EP tool.

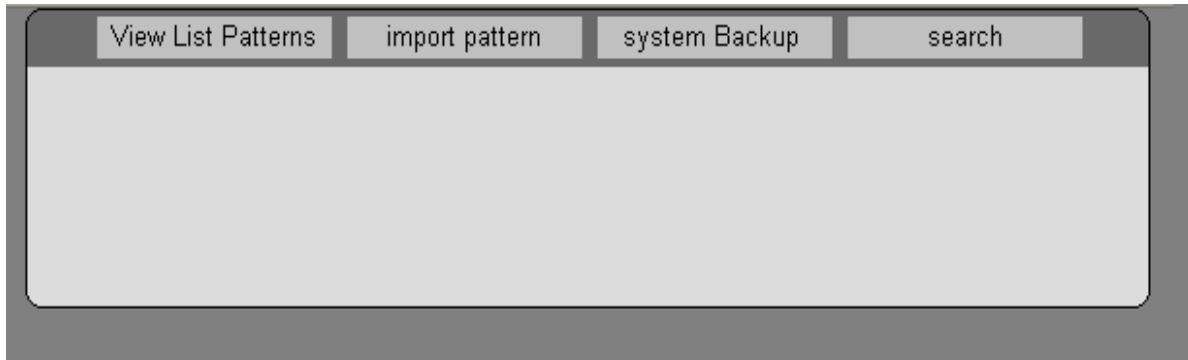


Figure 5.4 : The main functions for EPT

5.3- Loading and Displaying Pattern Data

In the third stage, all the patterns provided in this tool will be loaded and displayed to the user. These patterns will be saved in the general information file, and then loaded into an XML data document.

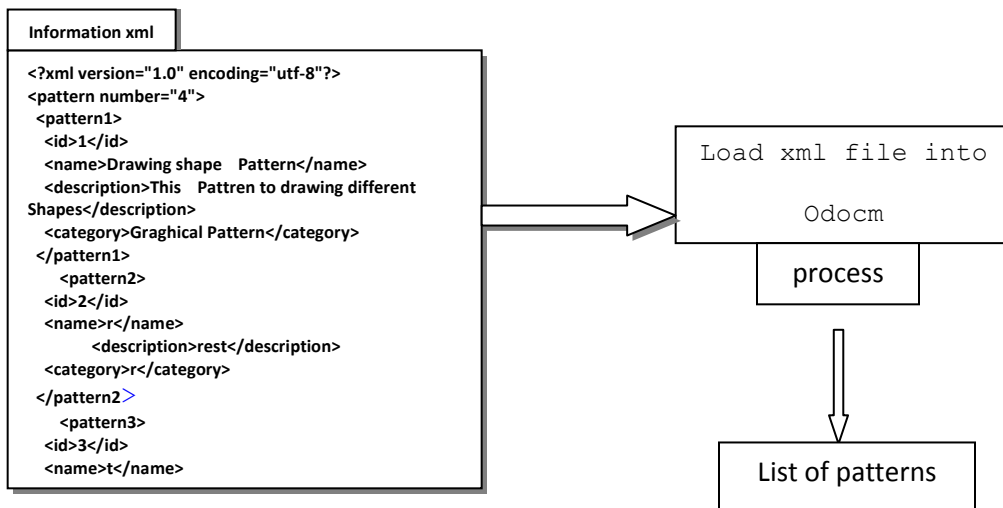


Figure 5.5: Load XML file to memory by an XML document

The result obtained from this operation is shown in Figure 5.6.

Name of Pattern	Description	Category	details..
Drawing shape Pattern	This Pattern to drawing different Shapes	Graphical Pattern	View More Details..

Figure 5.6: The list of patterns available in EPT

The result obtained from this function is explained as follows:

The total number of patterns available in EPT; in this case study, one pattern.

Name of pattern: drawing shapes pattern.

Category of pattern: graphical pattern.

Link to obtain all details about the selected patterns; when one is selected the result is list of classes of this pattern.

When the user presses “view full class,” obtains a summary of the fields and methods included in this class. At this point, the user can start processing an XML document.

name of pattern	number of classes	problem	soulution
Drawing Shapes Pattern	4	The problem is summarised in drawing different shapes and change it colors and sizes	Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality. drawing alot of shapes in different colors and different size
<u>List Of classes</u>			
-	name of class		details..
1	Class Canvas		view full class..
2	Class Circle		view full class..
3	Class Square		view full class..

Figure 5.7: List of classes for the shapes pattern

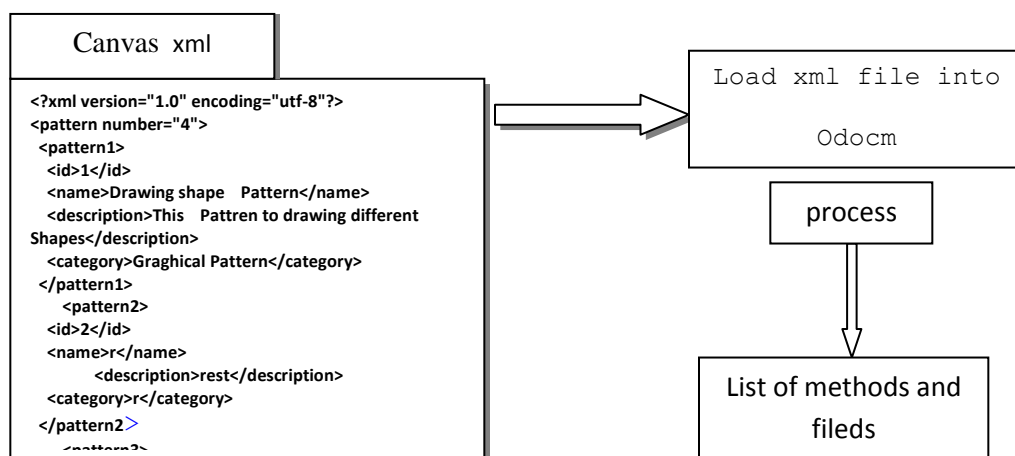


Figure 5.8 : Load canvas XML file to memory by an xml document

The result obtained from this operation is shown in Figure 5.9

Class Caven

Class Canvas - a class to allow for simple graphical drawing on a canvas.

Field Summary :

Canvas.CanvasPane	<u>canvas</u>
JFrame	<u>frame</u>
Graphics2D	<u>graphic</u>
CanvasPane	<u>canvas</u>
Color	<u>backgroundColour</u>
Image	<u>canvasImage</u>

[filed Details](#)

Method Summary :

Canvas	<u>getCanvas()</u> Factory method to get the canvas singleton object.
void	<u>draw()</u> Draws a given shape onto the canvas.
boolean	<u>drawImage()</u> Draws an image onto the canvas.

Figure 5.9: list of fileds and methods of canvas class

And when press methods details or fileds details, will obtain the follow result, shown in Figure 5.10

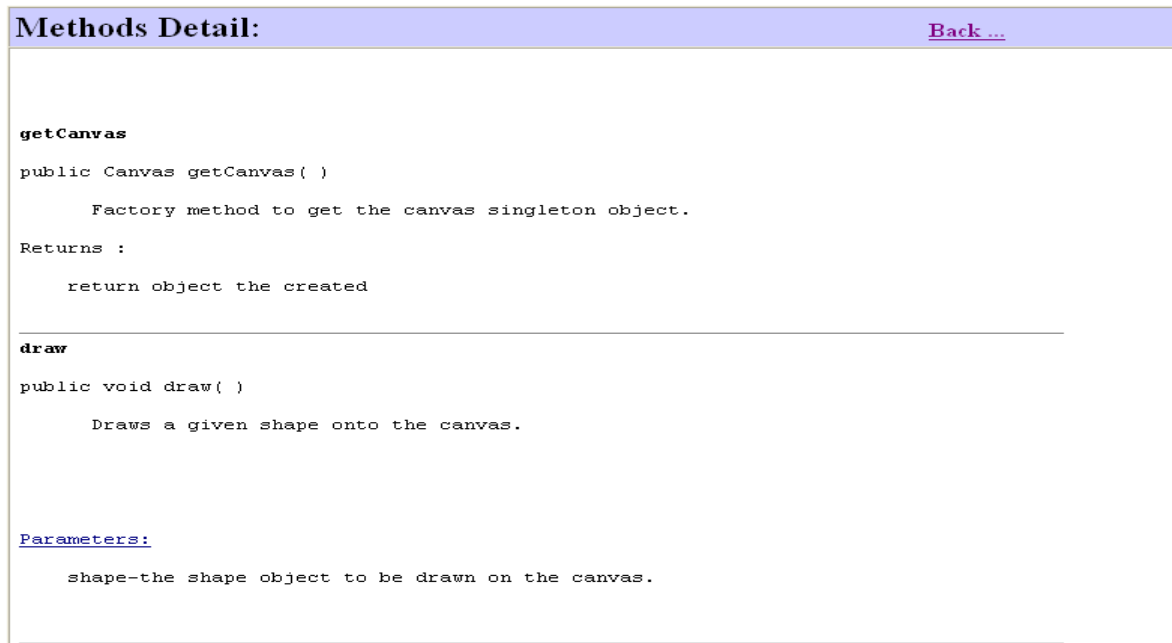


Figure 5.10: view details about methods

5.4- Import patterns

In this function we will import our patterns to save their in EPT by download pattern from store pattern and upload it to EPT, Figure 5.11 explain the result.

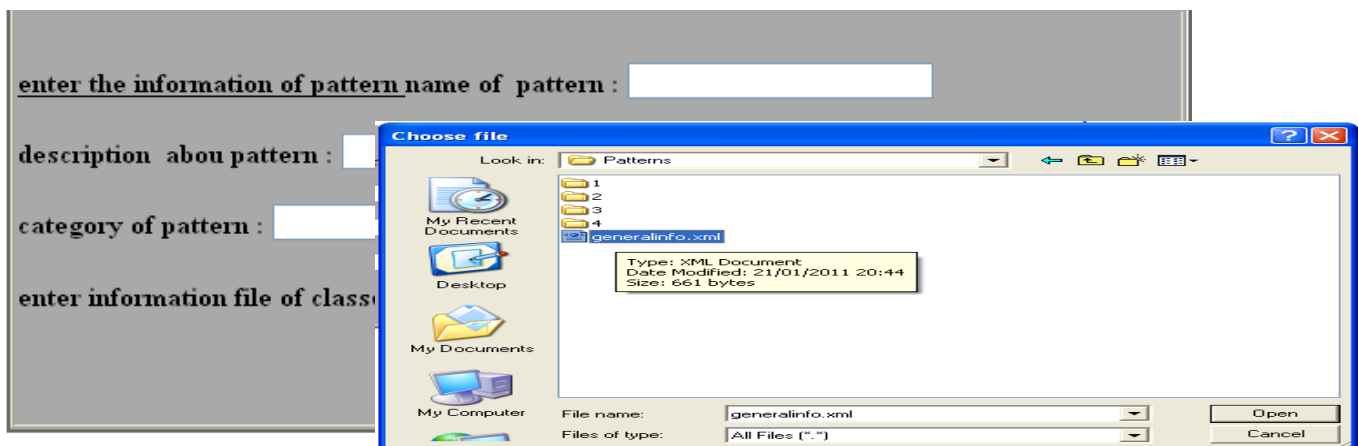


Figure 5.11: load information xml file of pattern into EPT

This tool divided the operation of upload pattern in two parts:

- 1)- upload the information file of pattern ,as Figure 5.11 and include (name of pattern, description, category of pattern) .
- 2)- upload xml files specially of classes (files and methods xml files) as follow ,Figure 5.12

enter the classes of pattern

enter files file : (.xml)

enter methods file : (.xml)

Figure 5.12: load xml files of classes for pattern into EPT

5.5- System Backup

In this function will make backup for all patterns available in EPT ,and the new path of backup will appear to the user after the operation is completely.

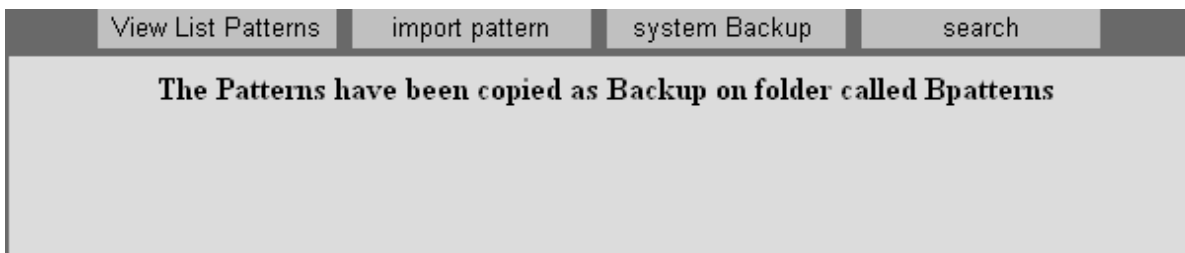


Figure 5.13: Backup patterns that store in PET

5.6- Search about pattern

Present this tool the practicability find pattern saved in it by enter name of pattern ,and obtain details about that pattern.

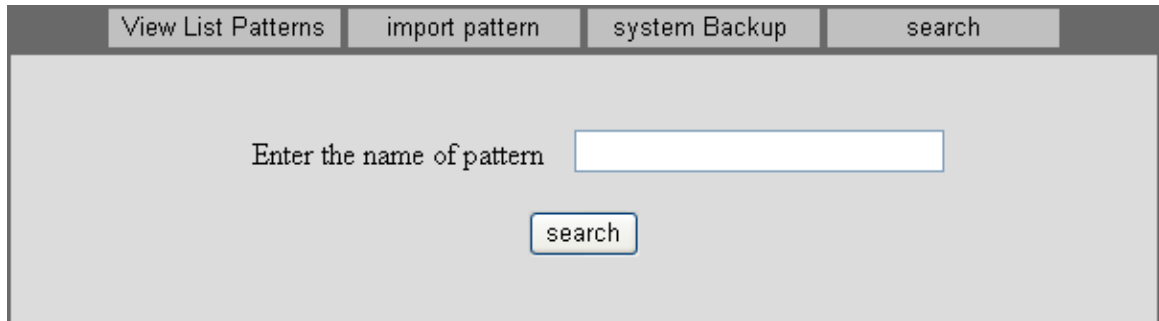
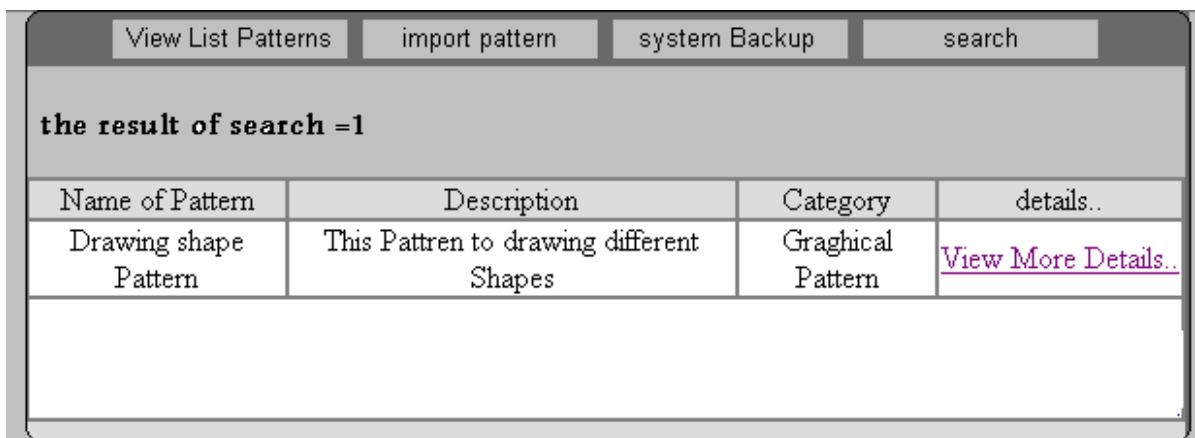


Figure 5.14: find pattern available in EPT

The result obtained from this function can be as follow:



the result of search =1			
Name of Pattern	Description	Category	details..
Drawing shape Pattern	This Pattren to drawing different Shapes	Graghical Pattern	View More Details..

Figure 5.15: the result of search operation

5.7- dealing with store patterns

The storage for the proposed pattern will be as follow:

The downloaded patterns by the users will presented in the storage of the pattern.

This storage enables the users to look at the classes , search for and download any patterns a particular pattern. Figure 5.16 illustrates the result.

the number of patterns avalibal is : 1				
Name of Pattern	Description	Category	details..	download files..
Drawing shape Pattern	This Pattren to drawing different Shapes	Graghical Pattern	View More Details..	download files..

Figure 5.16: list of patterns available in EPT

To load any pattern from the storage ,press the related link and many choices will appear as shown ,Figure 5.17 illustrates the result.

name of pattern	number of classes	problem	soulution
Dreawing Shapes Pattern	4	The problem is summarised in drawing different shapes and change it colors and sizes	Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality. drawing alot of shapes in different colors and different size
List Of files			
download information file			
-	name of class	save class	
1	Class Canvas	Save methods file .. Save fileds file ..	
2	Class Circle	Save methods file .. Save fileds file ..	
3	Class Square	Save methods file .. Save fileds file ..	
4	Class Triangle	Save methods file .. Save fileds file ..	

Figure 5.17 : details about shapes pattern

And To load the pattern files .First we download the information files for the chosen file and then we download the files belonging to the classes which are involved in creating these pattern. Downloading can be done as follow, Figure 5.18

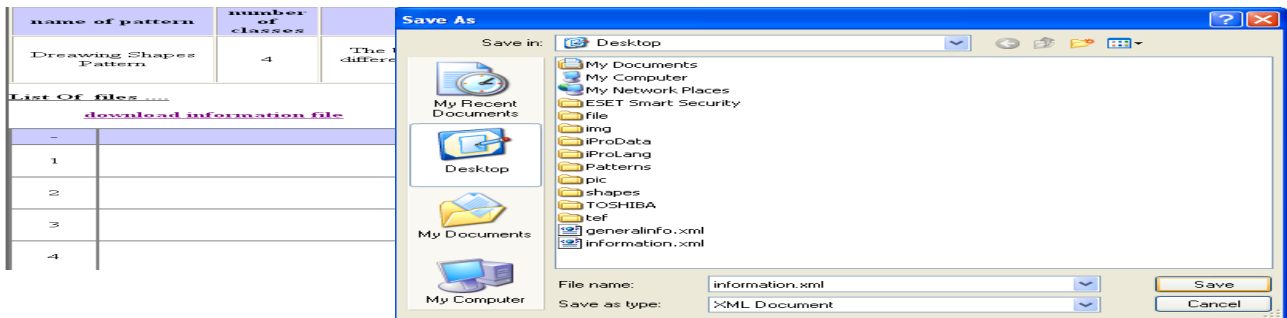


Figure 5.18 : download files of shapes pattern

Chapter 6

Conclusion and future work

6.1 Conclusion

Workers in the fields of computer design and development encounter many problems. A recurrent problem is how to save and document pattern templates and solutions to programming problems that have already been produced or obtained by other designers or developers. These patterns and solutions can also be the collective effort of a firm that needs to document them, so they can be exchanged

by developers and users inside the firm or across firms through the internet. Since the patterns and solutions are located either in a local computer or a web server, a way needs to be found to store them so they can be retrieved for development by firm members or by students in a university lab who use this data to improve their experience.

In this research, a tool is designed for solving this storage problem. The data are saved in an XML file containing information about patterns and class interactions, so that the content of the file can be analyzed to provide understandable information to developers and designers. The suggested tool has the following functions:

- On the user side, when the user imports the design pattern presented in an XML file and lists it inside the tool, this tool will parse the XML file, extract the information in it and display this information in an understandable format.
- The tool file will request the storage templates from the server or from a local computer.
- It will allow the user to manage the pattern templates, list them in the system, save them and have them returned whenever they are needed.

6.2 Scope of Future Work

This work can be used as a basis for designing patterns in XML or producing a special code or DLL for patterns. Much effort may be needed to make this tool generalizable to any kind of XML document carrying special data for different patterns.

References

- [1] - Wikipedia team. 2009. http://en.wikipedia.org/wiki/Design_pattern.
http://en.wikipedia.org. [Online] 6 5 2009. [Cited: 15 9 2009.]
- [2]- Bray, Tim; Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau. September 2006. *Extensible Markup Language (XML) 1.0 (Fourth Edition)-Origin and Goals*. s.l. : World Wide Web Consortium. <http://www.w3.org/TR/2006/REC-xml-20060816/#sec-origin-goals>, September 2006.
- [3]- w3schools team.<http://www.w3schools.com/html/ introduction.html> [Online][Cited: 25 06 2010]
- [4]- w3schools team .http://www.w3schools.com/html/ dom_ intro.asp [Online][Cited: 9 10 2010]
- [5]- S. MacDonald, D. Szafron, J. Schaeffer, J. Anvik, S. Bromling and K. Tan.
<http://www.cs.ualberta.ca/~systems/cops>. *http://www.cs.ualberta.ca*. [Online] [Cited: 9 10 2009.]
- [6] - S. MacDonald, D. Szafron, J. Schaeffer, J. Anvik, S. Bromling and K. Tan. 2006. *Generative Design Patterns*. canada : <http://www.cs.ualberta.ca>, 2006.
- . <http://www.cs.ualberta.ca/~systems/cops>. *http://www.cs.ualberta.ca*. [Online] [Cited: 9 10 2009.]
- [7] M. Massingill, T. Mattson, and B. Sanders. 1999. A pattern language for parallel application programs. Florida : Technical Report CISE TR 99-022, University of Florida, 1999, 1999.
- [8] Marek Vokáč, Software Engineering Department, Simula Research Laboratory, Oslo, Norway . 2006.An efficient tool for recovering Design Patterns from C++ Code
- [9] C Kramer and L. Prechelt. Design recovery by automated search for structural design patterns in object-oriented software. In *Reverse Engineering, 1996.*, Proceedings of the Third Working Conference on, pages 208–215, 1996.
- [10] Microsoft. C# programmer’s reference: foreach, in, 2004.

[11] G. Florijn, M. Meijers, and P. van Winsen. Tool support for object-oriented patterns. In *Ecoop'97: Object-Oriented Programming*, volume 1241 of *Lecture Notes in Computer Science*, pages 472–495. 1997.

[12] Zsolt Balanyi and Rudolf Ferenc. Mining design patterns from c++ source code. In *International Conference on Software Maintenance (ICSM'03)*, page 305, Amsterdam, The Netherlands, 2003. IEEE.

[13] - Wikipedia team. 2012. http://en.wikipedia.org/wiki/Design_pattern.
<http://en.wikipedia.org>. [Online] 4/ 7/ 2012 . [Cited: 10 /7/2012.]

Appendix

Table A1 Adapter class documentation

Field Summary

Name	Type
Odocm	XmlDocument

Methods Summary

Name	Description
Loadpattren (path As String)	Load xml file and retrieve it in Odocm object .
Mappath(string path)	To get path of files

Methods inherited from class System.Web.UI.Page , System.Xml, System. data

Server,mappath

Table A2 Parse_file class documentation

Field Summary

Name	Type
Odocm	XmlDocument
Onode	XmlNode
oNodeList	XmlNodeList
str	string

Methods Summary

Name	Description
loadpattren(ByVal path As String): XmlDocument	Load xml file and retrieve it in Odocm object .
get_attributes(ByVal odoc As XmlDocument)	Retrieve attributes that presented in xml file and return XmlNodeList.
get_methods(ByVal odoc As XmlDocument)	Retrieve methods that presented in xml file and return XmlNodeList.
get_nameclass(ByVal odoc As XmlDocument)	Retrieve class name that presented in xml file
viewattformatting(ByVal oNodeList As XmlNodeList) : string	Retrieve formatting ready to print of attributes

viewmethformatting (ByVal oNodeList As XmlNodeList) : string	Retrieve formatting ready to print of methods
--	---

Methods inherited from class System.Web.UI.Page , System.Xml, System. Data

Table A3 Parse_info class documentation

Field Summary

Name	Type
Odocm	XmlDocument
Onode	XmlNode
num	Integer
strname	String

Methods Summary

Name	Description
get_IDS(ByVal oNodeList As XmlNodeList)	Return the id of pattern
get_infonode(ByVal odoc As XmlDocument, ByVal nodename As String)	Retrieve data of specific node
printnode(ByVal oNodeList As XmlNodeList)	Print data of specific node
get_numofpatterns(ByVal oNodeList As XmlNodeList)	Return the total number of patterns
get_name(ByVal oNodeList As XmlNodeList)	Return the name of pattern
get_description (ByVal oNodeList As XmlNodeList)	Return the description of pattern
get_category (ByVal oNodeList As XmlNodeList)	Return the category of pattern

Methods inherited from class System.Web.UI.Page , System.Xml, System. data

Server,mappath

Table A4 Parse_fields class documentation

Field Summary	
Name	Type
Typ	String
Acc	String
onode	XmlNode
oNodeList	XmlNodeList
strname	String
Methods Summary	
Name	Description
get_type(ByVal oNodeList As XmlNodeList)	Return the type of fields (integer ,string,..)
get_access(ByVal oNodeList As XmlNodeList)	Retrieve the permission of this filed (private ,public)
get_name(ByVal oNodeList As XmlNodeList)	Return the name of filed
get_description(ByVal oNodeList As XmlNodeList)	Get description of class
get_dtype(ByVal oNodeList As XmlNodeList)	Get type of data (static,..)
Viewformatting(ByVal oNodeList As XmlNodeList)	Print the data in final format

Methods inherited from class System.Web.UI.Page , System.Xml, System. data

Table A5 Mfiles class documentation

Field Summary	
Name	Type
typ	string
onode	XmlNode
oNodeList	XmlNodeList
strname	String
ret	String
Methods Summary	
Name	Description
get_name(ByVal oNodeList As XmlNodeList)	Return name of method
get_access(ByVal oNodeList As XmlNodeList)	Retrieve the permission of this method (private ,public)
get_returnvalue(ByVal oNodeList As XmlNodeList)	determine if method return data or void
get_description(ByVal oNodeList As XmlNodeList)	Get description the work of method

get_type(ByVal oNodeList As XmlNodeList)	Get type of method
Checkhasparameters(ByVal oNodeList As XmlNodeList): boolean	Check if the method has parameters
Returnparameters(ByVal oNodeList As XmlNodeList)	Return parameters for specific method
Get_description(ByVal oNodeList As XmlNodeList)	Return the description of specific method
Get_namepara(ByVal oNodeList As XmlNodeList)	Return the name of parameter
Get_paradescr(ByVal oNodeList As XmlNodeList)	Return the description of parameter

Methods inherited from class System.Web.UI.Page , System.Xml, System. data

Table A6 Parse_methods class documentation

Field Summary

Name	Type
typ	String
onode	XmlNode
oNodeList	XmlNodeList
strname	String
Ret	String

Methods Summary

Name	Description
get_name(ByVal oNodeList As XmlNodeList)	Return name of method
get_access(ByVal oNodeList As XmlNodeList)	Retrieve the permission of this method (private ,public)
get_returnvalue(ByVal oNodeList As XmlNodeList)	determine if method return data or void
get_description(ByVal oNodeList As XmlNodeList)	Get description the work of method
get_type(ByVal oNodeList As XmlNodeList)	Get type of method
Checkhasparameters(ByVal oNodeList As XmlNodeList): boolean	Check if the method has parameters
Returnparameters(ByVal oNodeList As XmlNodeList)	Return parameters for specific method

Get_description(ByVal oNodeList As XmlNodeList)	Return the description of specific method
Get_namepara(ByVal oNodeList As XmlNodeList)	Return the name of parameter
Get_paradescr(ByVal oNodeList As XmlNodeList)	Return the description of parameter

Methods inherited from class System.Web.UI.Page , System.Xml, System. data

Table A7 Mtemplates class documentation

Field Summary

Name	Type
Found	boolean
PathS	string
docm	xmldocument

Methods Summary

Name	Description
Saverecivedpattern(xdoc as xmldocument) : void	Save pattern on the server .
AddnewPattern(xdoc as xmldocument) : void	Add new patterns to storage templates.
Viewavalibaltemplates() : void	Retrieve all templates available on storage templates
BackupTemplatesonserver() : void	Make backup templates
Deletepatterns(xpath as string) : void	Delete stored patterns from storage patterns

Methods inherited from class System.Web.UI.Page , System.Xml, System. Data