一個以 XML 爲基礎的 DESIGN PATTE R 設計工具[1]

戴佑恩、鄭有進[2]、謝金雲　　周忠信

國立台北科技大學 電子工程系　　東海大學 資訊科學系

{yccheng,hsieh}@en.ntut.edu.tw　　jwo@mail.thu.edu.tw

摘要

Design Patter 爲物件導向程式設計中一個常用的方法，本文描述如何利用 XML（eXtendable Markup Language），定義一套描述 Design Patter 中文件說明與結構圖的語法，並以 JAVA 語言與相關工具設計及實作一個協助程式設計師定義、註解及記錄使用經驗的 Design Patter 的設計工具。

關鍵詞：Design Pattern, XML, CASE tool, JAVA

1. 簡介

Design Patter 在近幾年來已漸漸地成爲主要軟體設計方法之一，它對物件導向設計中物件模型的結構與行爲有非常詳盡的分析，並有一套說明完整且結構化的描述文件，使用者在瞭解需求之後，便可將合適的 Design Pattern 套用至其程式設計中。Design Pattern 套用的成功與否與程式設計師的訓練與經驗（如熟悉的 Pattern 數目、瞭解的深度及過去應用的經驗等）有密切的關係，這樣的知識固然可以自然的儲存在程式設計師的腦子裡，然一方面隨 pattern 數目的增加，另一方面顧及重複使用的便利性，一個協助程式設計師定義、註解及記錄使用經驗的 Design Patter 的設計工具顯然有其必要。因此，曾有研究者提出以 SGML（Standard Generalized Markup Language）制定出一套 Design Pattern 的描述語法及設計工具[1]。然而，SGML 的複雜度與支援工具的昂貴使得這樣的工具無法有效的被採用。

爲了避免 SGML 所遭遇到的困難且同時擁有 SGML 相同的彈性，我們以新一代的 XML 替代 SGML，定義出一套描述 Design Patter 的語法，並以 JAVA 語言與相關工具設計及實作一個協助程式設計師定義、註解及記錄使用經驗的 Design Pattern 設計工具；透過它以圖形介面（GU - Graphic User Interface）爲主的整合開發環境，程式設計師將可：

1. 製作使用者自行定義的Design Patter 結構圖與說明文件，並且完全以 XML 的方式描述。
2. 套用現成內建的 Design Pattern 結構圖與說明文件，並在圖形化介面下進行修改與程式設計。

關於 Design Pattern 與 XML 這兩個技術，我們請讀者直接參考相關書籍及文件[2][3]。本文將針對 Design Pattern 設計工具的系統規畫、設計、實作、測試與驗證的部分作詳盡的說明。

2. 系統規劃

2.1 系統目標

目前市場上的物件導向開發工具普遍提供非常完整的結構化圖形編輯環境，其原因在於人類對於圖形的理解能力較文字描述強。再觀察 Design Pattern 的文件描述格式與使用者的閱讀方式，發現使用者通常在使用 Design Pattern 時皆是套用書中結構圖，然後根據其架構設計自己的程式，故在使用者介面上，以結構圖的圖形化編輯環境爲主。

事實上，本系統所期望提供的是一套整合的程式開發工具，所謂的整合是指使用者可經由此系統，進行 Design Pattern 的套用、說明的撰寫，並可透過圖形化的

[1] 本研究承國科會大專生參與專題研究計劃獎助（計劃編號：NSC88-2815-C-027-001-E）特此銘謝

[2] 主要聯絡人

編輯環境編寫原始程式碼，然後產生部分原始程式。故結構圖的編輯必須與程式碼的撰寫有適當的結合。再者，由於考慮到 Pattern 的儲存與電腦處理需定義一套完整且標準化的格式來描述，故需選用一套標準化的文件描述語言來定義 Design Pattern 的文件。

根據以上幾點考量，本系統所需達成的目標，大致可分爲以下幾個重點：

1. 提供一個圖形化的編輯環境， 讓使用者可依 Design Pattern 中結構圖的部分進行其特定系統結構圖的建置。
2. 根據所建置之系統結構圖，使用者可對特定的模組或物件進行原始程式碼的編輯。
3. 制定一套標準化的 Design Patter 描述語言，以方便電腦化處理，以及資訊交換。

2.2 開發工具

本系統共使用了下列幾個開發工具：

1. **XML**：我們選用 XML 來爲 Design Pattern 制定一套標準化的描述語言，主要因爲 XML 可讓使用這自行定義文件描述格式，且在透過 DTD 與 XSL 等相關技術的支援後，將可讓我們所定義的 Pattern 描述格式定義標準化，且能直接利用 Web browse 瀏覽我們所描述的 Pattern。
2. **JAVA Development Kit 1.2**：在考慮到本系統希望能以跨平台的方式呈現，並且提供完整而易於使用的 GUI（Graphic User Interface），我們以 JAVA 開發本系統的核心，期望運用新版 JAVA 1.2 所提供的強大 GUI 處理能力（Swing packages [4]），與其語言本身的可攜性，製作能在多種平台上執行又擁有強大功能且具備友善的圖形化編輯能力的應用軟體。
3. **Sun Project X Technology Release 1（XML Parser for JAV ）**：這是一套由昇陽公司所開發的 XML Parser [5]，其功能在檢查使用者自行撰寫的 XML 檔案是否符合語法規則，並可建立出 XML 文件的物件模型（Object Module），由於這套 Parser 完全以 JAVA 開發，故可直接將它與其 JAVA 程式整合。所以我們利用這個工具作爲本程式中 Design Pattern 之 XML 檔案與系統核心的連結，其實作於下節將有詳細說明。

3. 系統設計與架構說明

3.1 Design Patte r 的 XML 文件設計

在設計 Design Pattern 的 XML 文件之前，首先要對 Design Pattern 文件本身作結構上之分析，然後才能進行 XML 文件設計。以下即由分析至設計作完整的說明。

1. **Design Pattern 文件結構分析**：我們根據參考文獻[2]對於 Design Pattern 的結構化描述，將每個 Pattern 分爲文件說明與結構圖兩部分，其中文件說明部分包括一個 Pattern 的製作意圖（Intent）、動機（Motivation）、應用（Applicability）、結果（Consequences）與實作（implementation）：這個部份主要是針對這個 Pattern 的使用時機、應用範圍、使用方法、以及套用後之結

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE pattern SYSTEM "pattern.dtd">

<pattern>

        <title>
        Iterator
        </title>

        <intent>
        Provide a way to access the elements of an aggre ate
        object sequentially without exposing its underlyin
        representation.
        </intent>

        <motivation>
        User define…
        </motivation>

        <applicability>
        User define...
        </applicability>

        <consequences>
        User define...
        </consequences>

        <implementation>
        User define...
        </implementation>

        <structure>
                見圖二…
        </structure>
</pattern>
```

Figure 1

果作說明。至於結構圖的部分，則是針對這個 Pattern 的圖形化表現。這兩部分的差異在於文件說明部分皆以文字敘述的方式表現，而結構圖則是利用類似 UML 的圖形繪製方式表現；故針對文件說明部份，只需將各個說明的部份作適當的區分，而結構圖則需設計各種標籤來定義圖形中的各元素（包括類別（class）、繼承關係（inheritanc ）以及各種平行關係），並利用各標籤之屬性描述這些圖形元素的大小與位置。

2. DTD 文件設計：在 XML 文件設計中 DTD（Document Type Definition）扮演著格式定義的角色，根據上述文件結構分析的結果，先將文件依照各個標題定義出基本的 elements，然後再分別定義分屬文件說明部分及結構圖部分之 element 的內含元件與屬性。其詳細的 DTD 描述由於內容過於繁瑣，在此不多做描述，而原始的定義描述檔則歸列在附錄中供參考。

3. 文件說明部分的 XML 文件設計：這個部分的設計較

```
<structure>
        <roles>
                <role id="aggregate" virtual="true" x="50" y="50" width="100"
                length="100" subClass="concreteAggregate">
                        <name>Aggregate</name>
                        <participant>
                        defines an interface for creating an Iterator object
                        </participant>
                        <memberFunction virtual="true">
                                <name>CreateIterator()</name>
                                <code></code>
                        </memberFunction>
                </role>
                <role id="concreteAggregate" virtual= "false" x="50" y="250"
                width="100" length="100" superClass="aggregate">
                        <name>ConcreteAggregate</name>
                        <participant>
                        implements the Iterator creation interface to return an instance of the proper
                        ConcreteIterator
                        </participant>
                        <memberFunction virtual="false">
                                <name>CreateIterator()</name>
                                <code>return new ConcreteIterator(this)</code>
                        </memberFunction>
                </role>
                <role id="iterator" virtual="true" x="250" y="50" width="100"
                length="100" subClass="concreteIterator">
                        <name>Iterator</name>
                        <participant>
                        defines an interface for accessing and traversing elements
                        </participant>
                        <memberFunction virtual="true">
                                <name>First()</name>
                                <code></code>
                        </memberFunction>
                        <memberFunction virtual="true">
                                <name>Next()</name>
                                <code></code>
                        </memberFunction>
                        <memberFunction virtual="true">
                                <name>IsDone()</name>
                                <code></code>
                        </memberFunction>
                        <memberFunction virtual="true">
                                <name>CurrentItem()</name>
                                <code></code>
                        </memberFunction>
                </role>
                <role id="concreteIterator" virtual= "false" x="250" y="250"
                width="100" length="100" superClass="iterator">
                        <name>ConcreteIterator</name>
                        <participant>
                        implements the Iterator interface.
                        keeps track of the curent position in the traversal of the aggregate.
                        </participant>
                </role>
        </roles>
        <relations>
                <inheritance>
                        <from roleID="aggregate" x="100" y="150"/>
                        <to roleID="concreteAggregate" x="100" y="250"/>
                </inheritance>
                <inheritance>
                        <from roleID="iterator" x="300" y="150"/>
                        <to roleID="concreteIterator" x="300" y="250"/>
                </inheritance>
                <instantiate from="concreteAggregate" to="concreteIterator">
                        <positionX>150,250</positionX>
                        <positionY>285,285</positionY>
                </instantiate>
                <delegate from="concreteIterator" to="concreteAggregate">
                        <positionX>250,150</positionX>
                        <positionY>315,315</positionY>
                </delegate>
        </relations>
</structure>
```

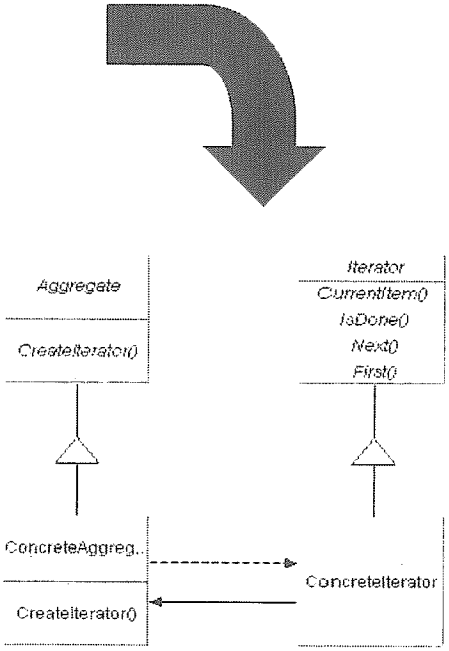

Figure 2

爲容易，只需將各個說明的部份視作個別的元素（Element），而對每一個部分定義一組 tag 作爲這些說明部分的區隔其結果如 【Figure 1】，此圖是以 Iterator Pattern 作爲例子，其中包含了<title> </title 、<intent> </intent 、<motivation> </motivati o n 、<consequences> </consequences>、<implementation> </implementation>與<structure> </structure>等 tag。這些 tag 中(除了<structure> tag)皆未再定義任何的 sub tag，且其內含資料爲字元的格式，至於<structure> tag 則內含了結構圖的圖形結構描述。

4. 結構圖部分的 XML 文件設計：這個部分的設計是將結構圖上所有的元件定義成個別的 element(如<role> 爲 class node、<inheritance>爲繼承關係、<instantiate> 爲產生的關係、<delegate>爲指向的關係等)，而各元件依據其特性又有其內含的 element 及屬性，其中值得一提的屬性就是各元件的定位屬性，由於結構圖是圖形化的表現方式，故必須對於圖中元件的大小與位置作描述，以供程式讀入作爲座標轉換，進而繪製出結構圖，【Figure 2】即爲結構圖的描述以及經由程式讀取後所繪出之結構圖。

3.2 XML Parser 與系統核心

本應用程式的核心，就是對於 XML 文件做讀取、修改與建立資訊的工作。【Figure 3】所示爲實作方式。

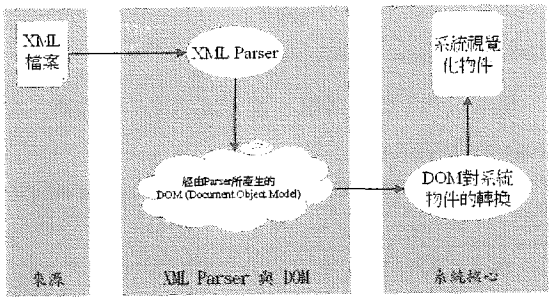

Figure 3

圖中，橢圓形的部分爲處理單元，DOM 與系統視覺化物件皆爲程式執行中的物件模型，而 XML 檔案則爲輸入的來源資料。其各部份的說明如下：

1. **XML Parser**：其動作爲讀入 XML 檔案，並根據 XML 文件的結構產生出一組 JAVA 程式物件的集合，這些物件可以透過 DO 的介面作讀取與修改的動作。
2. **DOM (Document Object Model)**：即一組包含了所有 XML 檔案內容的物件，這些內容也就是一個 Pattern 文件的所有資訊。
3. **DOM 對系統物件的轉換**：此部份位於系統核心，系統核心程式透過 DOM 的介面將 Pattern 文件中的各部分資訊取出，並產生系統視覺化的物件，如結構圖中的各個元件即根據這些文件中的資訊（包括座標、大小等）產生實際的 JAVA 程式的視覺化元件。
4. **系統視覺化物件**：這些物件包含了 Pattern 文件中個別不同元素的資訊，根據其性質分類，將屬於結構圖的部份作成個別的圖形元件（如 node、inheritance、delegate 等都有相對應的圖形元件），而文件說明則透過 Text Area 展現。稍後在使用者圖形介面設計的部份將有較詳細的介紹。

3.3 使用者圖形介面的設計與實作

由於本應用程式強調提供一個完整的圖形化編輯環境，讓使用者能透過對 Pattern 中結構圖的操作來達成程式撰寫的目的，故在 XML 檔案中特別強調對結構圖的描述，而使用者圖形介面也以提供圖形的繪製與修改爲主，配合個別的屬性設定浮動視窗，達成程式編寫的目的。如【Figure 4】所示。

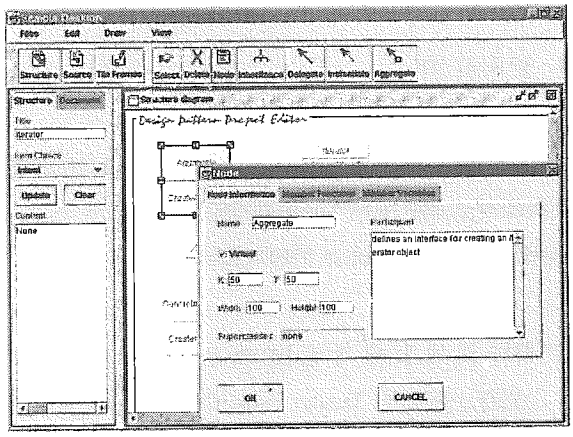

Figure 4

圖中上排的工具列爲圖形編輯工具列，可供使用者切換圖形繪製與編輯的功能；左列的分頁標欄爲說明文件編輯的部份；中間的主要視窗爲結構圖編輯的視窗(目前所展現的爲 Iterator Pattern)，使用者可利用圖形編輯工具列上的功能鈕，點選、刪除或繪製各種結構圖元件；而右下角的浮動式視窗則爲點選結構圖中 Aggregate Node 後，所呈現的元件屬性編輯視窗，使用者可透過此視窗進行屬性的編輯 (Node Informatio 頁面) 與程式碼

的撰寫（Member Functions 與 Member Variables 頁面）。

除此之外，本程式亦提供了 XML 原始碼的編輯環境，如【Figure 5】所示。

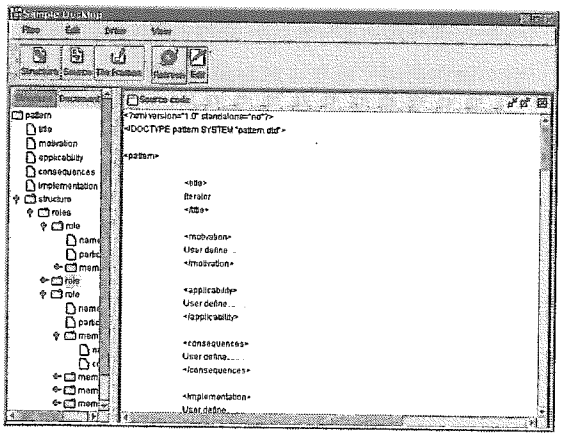

Figure 5

圖中左列之分頁標欄爲 XML 文件的樹狀結構展開，使用者可透過此樹狀結構做階層式的展開，進而了解此 Pattern 的各個元素在 XML 文件中的階層關係；而主視窗則爲 XML 原始碼的編輯視窗，藉此使用者可完整地看到這個 Pattern 的 XML 描述檔案，並可對其內容做編輯，再利用上方工具列的 Refresh 功能鈕來進行結構圖以及說明文件的更新。

4. 使用範例

在本節中，我們由結構圖的繪製、說明文件的編輯至實際程式碼的撰寫，按部就班製作一個套用 Design Pattern 中的 State Pattern 的應用程式 － 小計算機。

4.1 利用圖形編輯功能繪State Patte r 的套用範本

開啓一個新的 Design Pattern 範本設計視窗，如【Figure 6】所示。

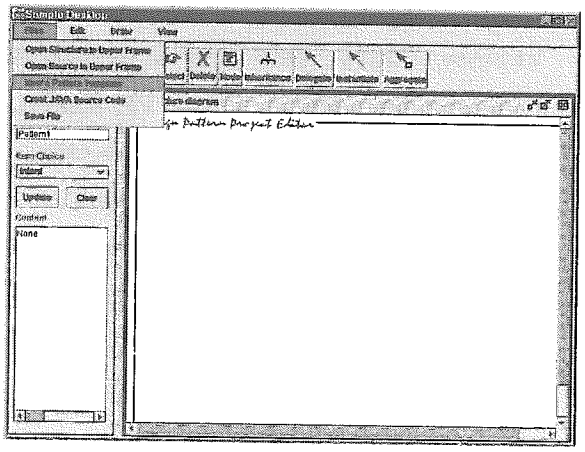

Figure 6

根據 State Pattern 基本的結構圖，繪製出 State Pattern 的套用範本，如【Figure 7】所示。

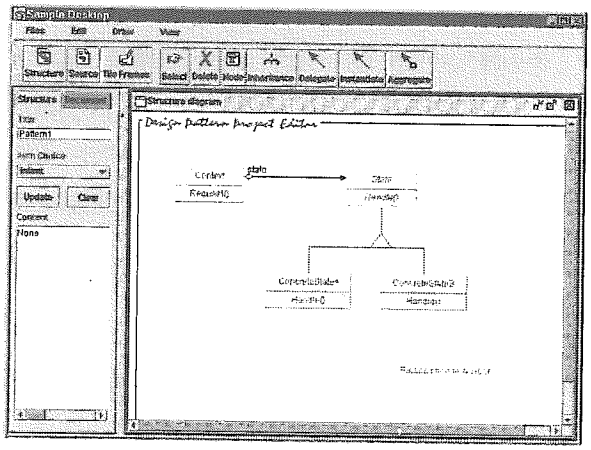

Figure 7

4.2Design Patte r 說明文件的編輯

點選右列下拉式選單，進行說明文件的編輯，如【Figure 8】所示。

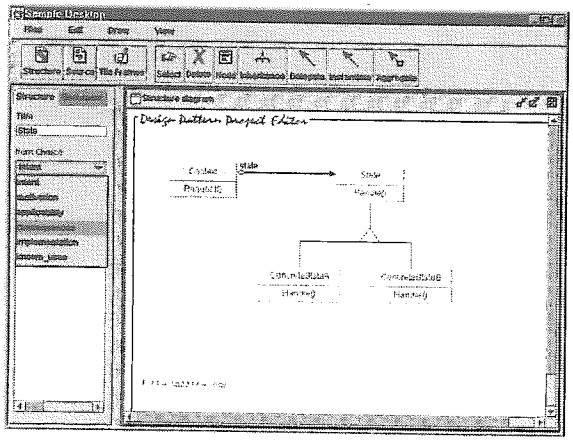

Figure 8

4.3 JAVA 原始程式碼的編輯與產生

套用以上所設計好的範本，建立出小計算機的核心計算模型，如【Figure 9】所示。

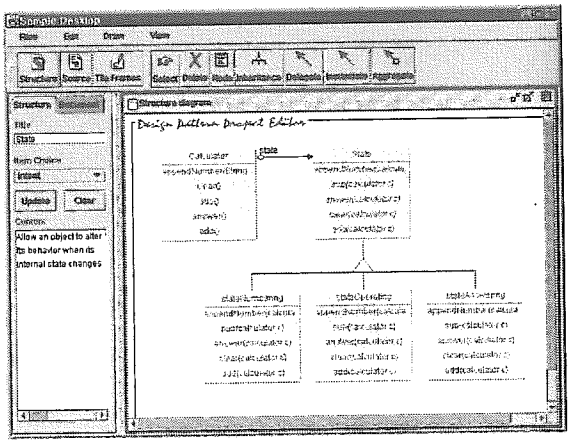

Figure 9

點選各個 class nod ，進行 member functio 及 member variable 的編輯，如【Figure 10】所示。

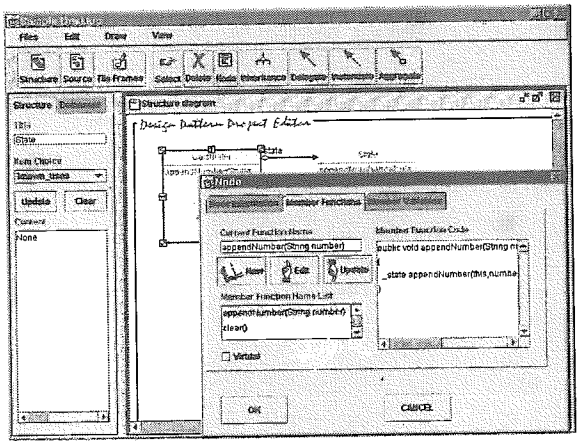

Figure 10

利用 JAVA 原始碼產生功能，產生相對於各個 class node 的.java 檔，如【Figure 11】所示。

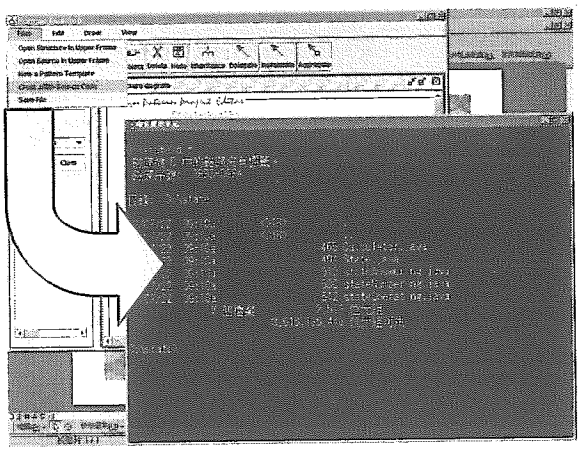

Figure 11

利用編輯軟體編輯小計算機的 GUI JAV A程式，並將由 Design Pattern Project Editor 所產生之.java 程式做小部份的修改。最後，整合編譯成一組.class 檔，如【Figure 12】即爲利用 JAVA interpreter 所執行的小計算機程式。

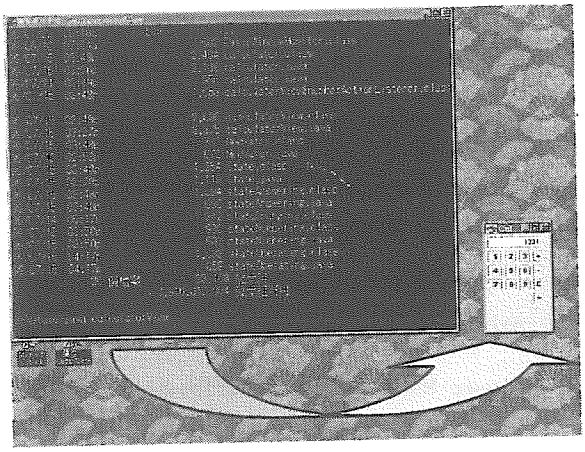

Figure 12

5. 結論與展望

本文介紹一個以 XML（eXtensible Markup Language）定義一套描述 Design Pattern 的語法，並以 JAVA 語言與相關工具設計及實作一個協助程式設計師定義、註解及記錄使用經驗的 Design Pattern 的設計工具。

在未來展望上，由於本系統之各元件從 XML Parser、Application Ker n e 至 GU 皆以 JAVA 開發，故可輕易與 WW 結合。再者，微軟的 Internet Explorer 5.0 以及即將推出的 Netscape Navigator 5.0 都支援 XML，這意味著 Design Pattern 的 XML 文件將可在 web browser 上依據統一的瀏覽介面進行資訊交換。我們將結合 XSL 的技術，使 browser 能有統一的介面，瀏覽各地運用 Design Pattern Project Edito 所製作出用同樣一種定義格式定義出的 Design Pattern XML 文件，達到遠距資訊交換的目的。

國科會大專生參與專題研究計劃（計劃編號：NSC88-2815-C-027-001-E）

參考文獻

[1] Ohtsuki, M.; Segawa, J.;Yoshida, N.; Makinouchi, A. "Structured document framework for design patterns based on SGML," *Computer Software and Applications Conference, 1997. COMPSAC '97. Proceedings., The Twenty-First Annual International*, pp. 320 – 323, Aug. 1997.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vliss ides, "Design Patterns: Eelments of Reusable Object-Oriented Software," *Addison-Wesley Professional Computing Series*, 1994.

[3] Elliotte Rusty Harold, "XML: Extensible Markup Language, " *IDG Books Worldwide, Inc.*, 1998.

[4] Robert Eckstein, "JAVA Swing," *O'Reilly & Associates, Inc.*, 1998.

[5] "The Source for JAVA Technology/XML by java.sun.com" http://java.sun.com/xml/

附錄：Design Pattern XM　文件格式定義檔

```
<!ELEMENT pattern (title,intent, motivation, applicability,
consequences, implementation, sample_code, known_uses,
related_patterns, structure)?>
    <!ELEMENT title (#PCDATA)>
    <!ELEMENT intent (#PCDATA)>
    <!ELEMENT motivation (#PCDATA)>
    <!ELEMENT applicablity(#PCDATA)>
    <!ELEMENT consequences (#PCDATA)>
    <!ELEMENT implementation (#PCDATA)>
    <!ELEMENT sample_code (#PCDATA)>
    <!ELEMENT known_uses (#PCDATA)>
    <!ELEMENT related_patterns (#PCDATA)>
    <!ELEMENT structure (notes?, roles, relations)>
    <!ELEMENT notes (#PCDATA)>
    <!ELEMENT roles (role*)>

    <!ELEMENT role (name, participant?,
memberFunction*,memberVariable*)>
    <!ATTLIST role id ID #REQUIRED>
    <!ATTLIST role virtual (true | false) "false">
    <!ATTLIST role x CDATA #REQUIRED>
    <!ATTLIST role y CDATA #REQUIRED>
    <!ATTLIST role width CDATA #REQUIRED>
    <!ATTLIST role length CDATA #REQUIRED>
    <!ATTLIST role superClass IDREF #IMPLIED>
    <!ATTLIST role subClass IDREF #IMPLIED>
    <!ELEMENT participant (#PCDATA)>
    <!ELEMENT memberFunction (name, code?)>
    <!ATTLIST memberFunction virtual (true | false) "false">
    <!ATTLIST memberFunction parameterCDATA #IMPLIED>
    <!ATTLIST memberFunction return CDATA #IMPLIED>
    <!ELEMENT code (#PCDATA)>

    <!ELEMENT memberVariable (name)>
    <!ATTLIST memberVariable virtual (true | false) "false">
    <!ELEMENT name (#PCDATA)>

    <!ELEMENT relations (inheritance, instantiate, delegate,
aggregate)*>
    <!ELEMENT inheritance (from,to*)>
    <!ELEMENT from EMPTY>
    <!ATTLIST from roleID IDREF #REQUIRED>
    <!ATTLIST from x CDATA    #REQUIRED>
    <!ATTLIST from y CDATA #REQUIRED>
    <!ELEMENT to EMPTY>
    <!ATTLIST to roleID IDREF #REQUIRED>
    <!ATTLIST to x CDATA    #REQUIRED>
    <!ATTLIST to y CDATA #REQUIRED>

    <!ELEMENT instantiate (positionX,positionY)>
    <!ATTLIST instantiate from IDREF #REQUIRED>
    <!ATTLIST instantiate to IDREF #REQUIRED>
    <!ATTLIST instantiate oneToMany (true | false) "false">
    <!ATTLIST instantiate text CDATA #IMPLIED>

    <!ELEMENT delegate (positionX,positionY)>
    <!ATTLIST delegate from IDREF #REQUIRED>
    <!ATTLIST delegate to IDREF #REQUIRED>
    <!ATTLIST delegate oneToMany (true | false) "false">
    <!ATTLIST delegate text CDATA #IMPLIED>

    <!ELEMENT aggregate (positionX,positionY)>
    <!ATTLIST aggregate from IDREF #REQUIRED>
    <!ATTLIST aggregate to IDREF #REQUIRED>
    <!ATTLIST aggregate oneToMany (true | false) "false">
    <!ATTLIST aggregate text CDATA #IMPLIED>
    <!ELEMENT positionX(#PCDATA)>
    <!ELEMENT positionY(#PCDATA)>
```