行政院國家科學委員會專題研究計畫　成果報告

以 XML 與統一模型整合並正規化軟體發展程序之研究與實做

計畫類別：個別型計畫
計畫編號：NSC92-2218-E-164-003-
執行期間：92 年 12 月 01 日至 93 年 10 月 31 日
執行單位：修平技術學院資訊管理系

計畫主持人：盧志偉
共同主持人：朱正忠
計畫參與人員：陳駿閎、游紀強、吳逸杰、陳明澤、李安純

報告類型：精簡報告

處理方式：本計畫可公開查詢

中　華　民　國 93 年 12 月 6 日

中文摘要：

軟體系統，尤其是當前熱門的 Internet 應用領域，由於逐漸成為企業活動不可或缺的一部份，而又面臨來自各種不同方面的嚴峻挑戰，例如：硬體平台的快速變遷、新軟體技術的發展、新型態服務需求的出現等等，因此，就其趨勢而論，必須於開發與維護的過程中，表現出更具有彈性與效率的本質。軟體系統的開發與維護，通常無可避免地涉及團對分工與不斷的溝通作業。然而，大多數的系統在這些過程中，一般都以個人特殊方式、缺乏標準規範的方式完成作業，而也就造成了整個開發與維護工作的繁複困難與高度的成本障礙。

而軟體標準就是用來解決上述的問題。所謂的軟體標準，例如通用模型語言(UML)、CORBA、COM/DCOM、設計樣板(Design Patterns)，以及其他已被公認的標準機制如以元件為基礎的軟體建構方式等，是用來提供標準高階抽象符記與程序規範，以期改善軟體產能與降低軟體開發成本。不過，他們都存在共同的問題，就是現有的標準方法往往只包涵軟體開發局部程序，無法涵蓋整個軟體開發流程，而且現有的標準之間也無法相互支援，設計者往往需要花費大量的人力去對應與整合這些標準方法，缺乏一個軟體開發標準的協同與整合機制，使得再使用程序中的自動化程度無法提昇，導致軟體開發成本的居高不下。在這個專案計畫中，我們將秉承技職院校應用實作的本職，與朱正忠教授合作，承接其「協同與整合各種標準於再使用導向的軟體開發之研究(NSC91-2213-E-029-004)」的理論基礎，利用以 XML 為基礎的整合式系統模型，利用此整合模型可以將不同開發階段中使用的各種軟體標準加以整合，以降低軟體模型間跨標準、跨階段的整合問題，並以 XML 為基礎的 open source 作為依循途徑，實作一個統一模型軟體工具以達成軟體開發及維護中自動化(automation)、可追蹤(tractability)及驗證(validation)等特性，並串連所有於開發過程中設計出的抽象與實做模型，而達成一個正規劃、整合性的系統文件規範，以協助解決長久以來存在軟體系統整合發展與維護作業上的問題。

關鍵字：軟體開發程序、可延伸標記語言、模型整合、自由軟體

1. 前言

軟體系統，尤其是當前熱門的 Internet 應用領域，由於逐漸成為企業活動不可或缺的一部份，而又面臨來自各種不同方面的嚴峻挑戰，例如：硬體平台的快速變遷、新軟體技術的發展、新型態服務需求的出現等等，因此，就其趨勢而論，必須於開發與維護的過程中，表現出更具有彈性與效率的本質。軟體系統的開發與維護，通常無可避免地涉及團對分工與不斷的溝通作業。然而，大多數的系統在這些過程中，一般都以個人特殊方式、缺乏標準規範的方式完成作業，而也就造成了整個開發與維護工作的繁複困難與高度的成本障礙。而軟體標準就是用來解決上述的問題。所謂的軟體標準，例如通用模型語言(UML)、CORBA、COM/DCOM、設計樣板(Design Patterns)，以及其他已被公認的標準機制如以元件為基礎的軟體建構方式等，是用來提供標準高階抽象符記與程序規範，以期改善軟體產能與降低軟體開發成本。不過，他們都存在共同的問題，就是現有的標準方法往往只包涵軟體開發局部程序，無法涵蓋整個軟體開發流程，而且現有的標準之間也無法相互支援，設計者往往需要花費大量的人力去對應與整合這些標準方法，缺乏一個軟體開發標準的協同與整合機制，使得再使用程序中的自動化程度無法提昇，導致軟體開發成本的居高不下。

在這個專案計畫中，我們將秉承技職院校應用實作的本職，與朱正忠教授合作，承接其「協同與整合各種標準於再使用導向的軟體開發之研究(NSC91-2213-E-029-004)」以 XML 為基礎的整合式系統模型的理論基礎，利用此整合模型可以將不同開發階段中使用的各種軟體標準加以整合，以降低軟體模型間跨標準、跨階段的整合問題，並以 XML 為基礎的 open source 作為依循途徑，實作一個統一模型軟體工具以達成軟體開發及維護中自動化(automation)、可追蹤(tractability)及驗證(validation)等特性，並串連所有於開發過程中設計出的抽象與實做模型，而達成一個正規劃、整合性的系統文件規範與實際可行的應用環境與工具，以協助解決長久以來存在軟體系統整合發展與維護作業上的問題。

2. 研究目的

軟體的開發與維護相當的不易，也因此至目前為止已有許多的組織及相關領域的專家制定並提出了軟體標準，例如通用模型語言(UML)、CORBA、COM/DCOM、設計樣板(Design Patterns)，以及其他已被公認的標準機制如以元件為基礎的軟體建構方式等，其目的就是提昇軟體的品質。軟體標準雖然提昇了軟體的開發效率，但個別的軟體標準往往只有支援軟體開發的部分過程，沒有一種軟體標準及模型可以涵蓋軟體開發過程中的每個階段。例如 UML 提供了需求分析及系統設計的軟體模型。而設計樣板(Design Pattern)則只有提供設計階段的軟體模型。至於元件技術(Component-based technology)主要是強調在實作階段。

軟體的生命週期一般而言可以圖 1 作說明，包含了分析、設計、實作及維護四各階段

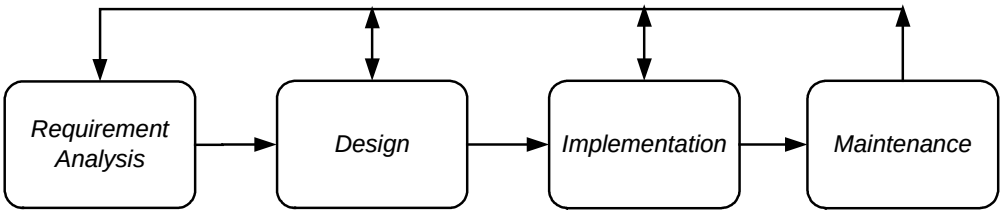

圖 1 軟體生命週期

在這些階段裡可以利用許多的軟體標準以降低軟體開發本身的複雜度。然而當我們使用某個標準於某階段時所造成的改變，可能會帶來其他階段所使用的標準也連帶的需要作改變的問題。在整個軟體開發週期裡，開發團隊裡的成員會根據特殊的需求或個人的經驗法則導入不同的軟體標準。但是所引發的結果卻是標準與標準之間的不協調(incompatibility)，這樣的結果往往會帶來許多問題，例如由一個軟體階段跨越至另一個軟體階段時可能會有部分的資訊因此流失。

要建置一個高維護性的軟體必須包含軟體生命週期的每個階段包含需求分析、軟體設計、系統實作及軟體測試。單純的利用軟體標準以支援特定軟體開發階段往往是不夠的，而是必須要將所有階段的所有軟體模型整合起來以達成軟體模型間的同步性，才能真正的達成軟體可維護性的需求。要整合所有軟體模型則是等於要整合所有的軟體標準，但以目前而言，各個標準的軟體模型往

往需要倚賴人為的思考從中穿針引線而得以完成，這樣的過程也容易發生人為的疏失而降低的軟體可維護性。

有鑑於上述的動機，在 XUM[1]的理論中提出一個方法用以整合所有的標準於一個基礎於 XML 的整合模型。所整合的軟體模型包含了 UML、Design Pattern，基礎於元件技術的框架結構及 XML。在該方法中描述軟體模型如何整合以達成彼此的對映關係及交換性。而經過整合的軟體模型即 XML-based unified model(XUM)。本計畫利用此理論方法並以實作一個統一模型軟體工具以達成軟體開發及維護中自動化(automation)、可追蹤(tractability)及驗證(validation)等特性。

3. 文獻探討

3.1 物件導向技術及 UML

物件導向是軟體工程領域的一項重要的技術，物件導向的觀點在於將現實世界人、事、物等許多概念的靜態資料及動態行為表示成為物件，而這些概念之間的互動也會對映至物件之間的交互關係。一個物件往往被定義成為一個角色，擁有各自的特徵與行為，而所謂的商業模型(Business Model)就是將某個領域的資訊定義成個別的物件。利用物件導向技術可以使軟體的開發更快、成本更低、及得到較高的品質[5]。物件導向分析與設計(OOA/D) [1]基礎於物件導向的概念成為了軟體開發方法的主流。

Unified Modeling Language (UML)[3]是用來將一個軟體系統描述成視覺化軟體模型的標準表示法。UML 提供了許多既定的圖示用來描述軟體開發過程不同階段的抽象概念，例如使用案例圖(Use case diagram)、類別圖(Class diagram)、合作圖(Collaboration diagram)等等。利用這些圖示法可以讓使用者很容易的了解軟體的分析與設計過程。也因此 UML 已成為物件導向軟體開發的一項重要的技

術。但 UML 的使用往往過於彈性而缺乏整合性，不同的 diagram 有不同的適用範圍，但不同的 diagram 卻缺乏一個串連的機制，在系統分析規劃時一旦需要跨不同 diagram，就有賴程式設計師以人工方式轉換、驗證，導致需求分析與系統設計的模型容易產生不完整及不一致性。

3.2XML

XML [6]是由 World Wide Web Consortium (W3C)所制定的標準，一種延伸式標記語言，具有擴展性(Extensibility)、結構性(Structure)、描述性(Description)、確認性(Validation)等特性。同時 XML 具有跨平台的功能，對於不同的作業系統、硬體設備、應用軟體、多元的輸入模式，開發者可以自行制定符合己身需求的標記(tag)，做結構性的描述，促使相同的一份文件呈現不同的規格，適用於不同的軟體，符合不同的設備、滿足多重的輸入方式。

3.3XUM (XML-based Unified Model)

XUM[7]是一個利用 XML 為基礎，將軟體生命週期中，不同階段產出物加以串連、整合，XUM 印證了不同型態的軟體模型、程式碼能有效的加以整合成為具相同表示格式，卻又不失各軟體模型原本特質，並且透過 XML 的特性，可以呈現出多重不同樣式。本篇論文主要就是以 XUM 的研究為基礎，進一步將系統文件加以正規化、整合成為一個一致的模式，並具有支援後續相關處理的表現模型。

4. 研究方法

本計畫主要是採用 Java 程式語言所實作之應用程式，實作的技術包含以下

- 圖形界面的框架—Java Swing、JHotDraw

- Java API for XML Binding 及 XML 相關技術
- Java Regular Expressions

一般而言，許多的軟體標準例如 UML、設計樣版提供了軟體模型的標準表示法，或提供了加速軟體開發的技術。但是並沒有任一種軟體標準可以包含整個軟體開發流程，而標準間也存在著整合不易或資訊交換的問題。一旦個別的導入這些軟體標準，將造成軟體維護的不易。

先前朱正忠教授在「協同與整合各種標準於再使用導向的軟體開發之研究(NSC91-2213-E-029-004)」研究中提出了一個基礎於 XML 的統一模型。XUM 可以整合不同階段各種使用的軟體標準於一個統一軟體模型。本計畫預計將此方法實作於一個以 JAVA 為開發語言的統一模型工具軟體。工具軟體的實作目標為利用 XUM 的理論基礎，並以自動化、可追蹤性及可驗證性等方向來加強軟體開發及維護的特性

在自動化方面透過抽象連結(Abstraction link)達成了需求分析至系統設計上某程度的自動化。當使用者建立或修改了需求模型的資訊，系統會將該資訊傳導至設計階段以提供設計人員建立設計模型。而透過程式碼鏈結(Source code link)則促進了設計模型與實作模型的同步機制。

另外軟體維護的最大問題在於解決軟體修改時產生的漣漪效應，解決的方法是做衝擊分析。但一般的衝擊分析只做到程式碼的階段，對於更上層的設計模型甚至需求模型則很難能有效的追蹤。再加上各階段導入不同的標準，使得軟體維護的問題更為複雜。由於 XUM 整合了軟體開發的各個階段包含需求分析、軟體設計及軟體實作之軟體模型及模型中各種不同得軟體標準格式。一旦某階段某種軟體標準的模型發生改變，透過鏈結的機制—抽象鏈結(Abstraction link)、整合鏈結(Integration link)及程式碼鏈結(Source code link)。可以很快的追蹤到直接及間接受影響的軟體模型。可以輔助開發人員提昇軟體維護的速度及正確性，而本工具軟體也實現了漣漪效應的可追蹤特性。

5. 結果

由於時間關係，本計畫「以 XML 與統一模型整合並正規化軟體發展程序之

研究與實做 (NSC92-2218-E-164-003)」第一階段對應於整體系統架構之關係如下圖 2 所示

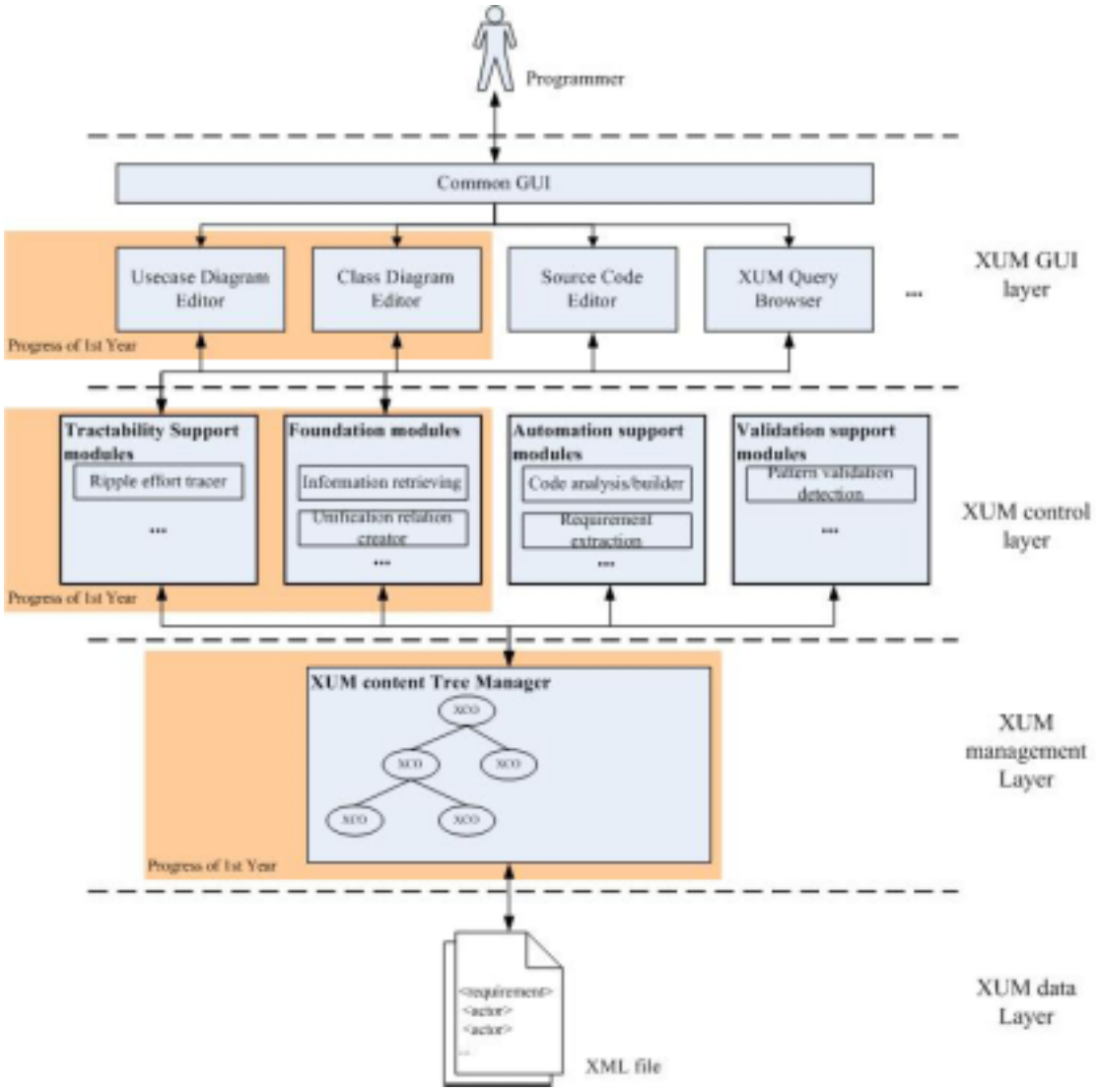

圖 2 系統架構圖

以下是第一階段成果與系統雛形簡略說明：

- XUM 使用者介面

(1) 使用模型工具的初始畫面

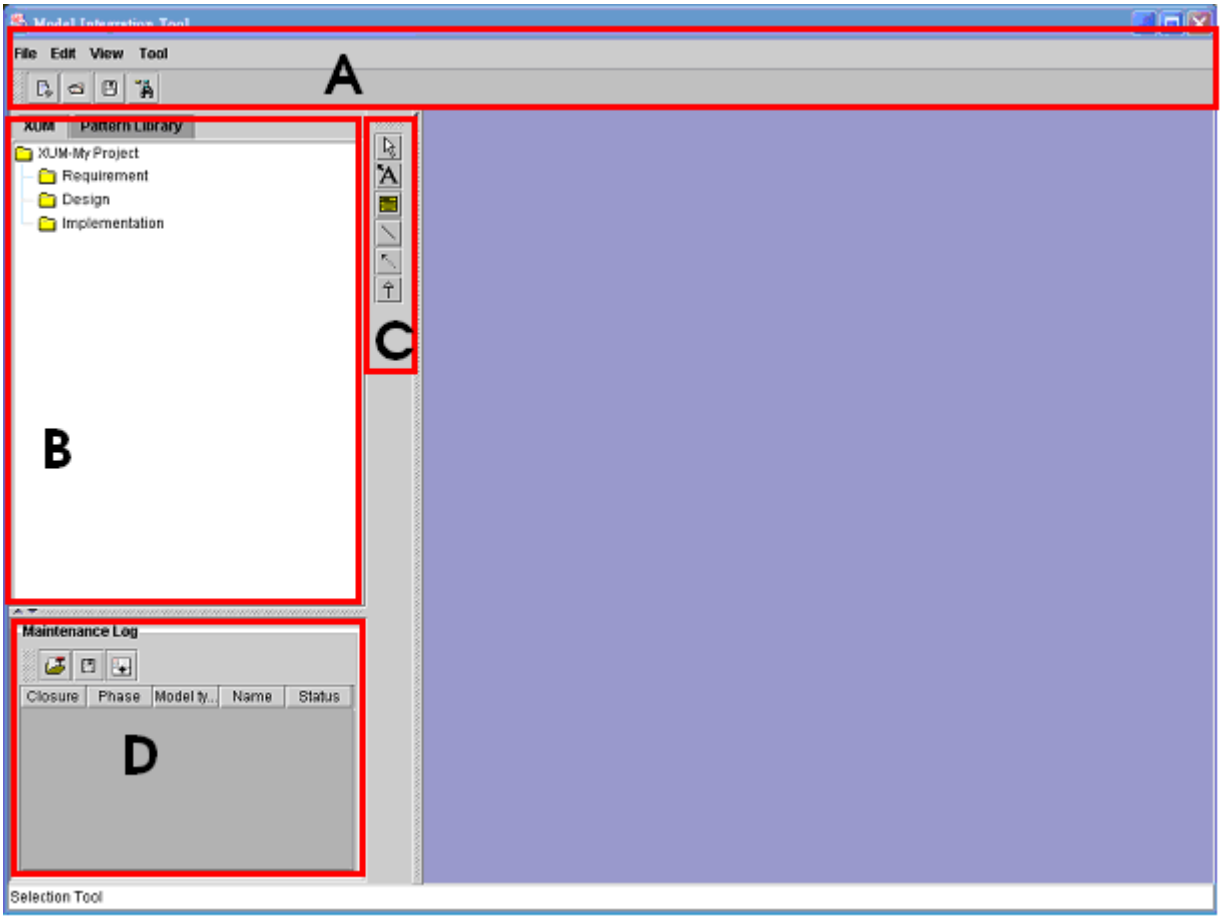

圖 3、使用模型工具的初始畫面

使用者執行工具軟體時所出現的初始畫面，如圖 3 所示。圖中 A 區塊為工具列，B 區塊為軟體開發的各個階段，C 區塊為編輯使用者案例或類別圖的工具區，D 區塊的功能為條列出各個經過修改的模型及其相關資訊包含該模型名稱、模型類型、名稱、該模型所屬階段(phase)及該模型作何種修改。

(2) 編輯使用者案例圖的執行畫面

本子系統提供了圖形化界面以提使用者建立統一模型，如圖 4 所示。圖中 A 區塊是模型編輯的工具區， B 區塊則為編輯的結果。

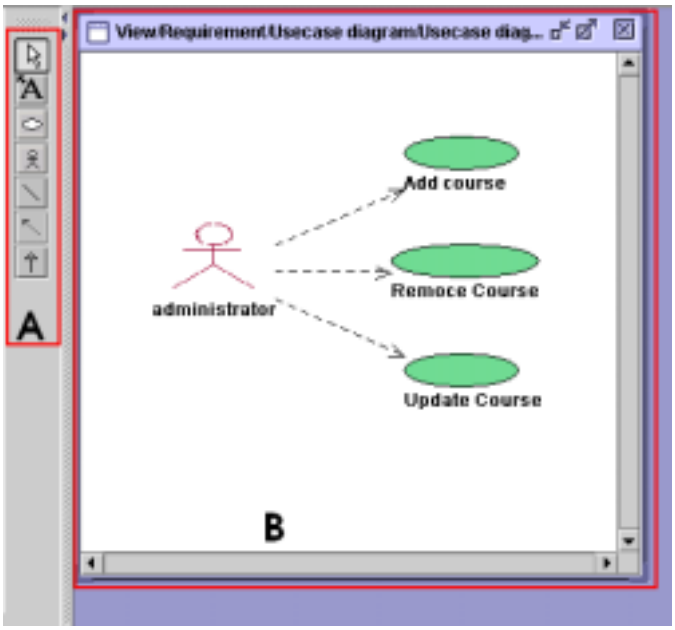

圖 4、編輯使用者案例圖的執行畫面

(3) 編輯類別圖的執行畫面

本子系統提供了圖形化界面以提使用者建立統一模型，如圖 5 所示。圖中 A 區塊是模型編輯的工具區， B 區塊則為編輯的結果。

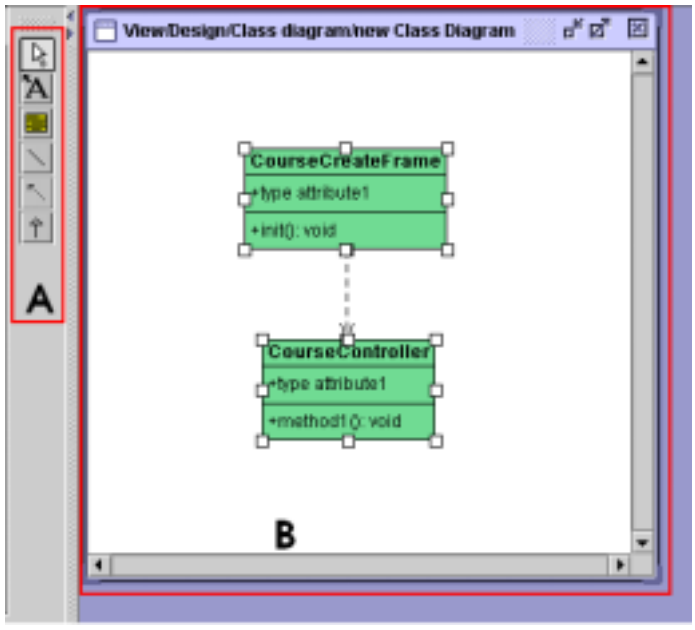

圖 5、類別圖之圖形化界面

- 統一模型的關聯機制

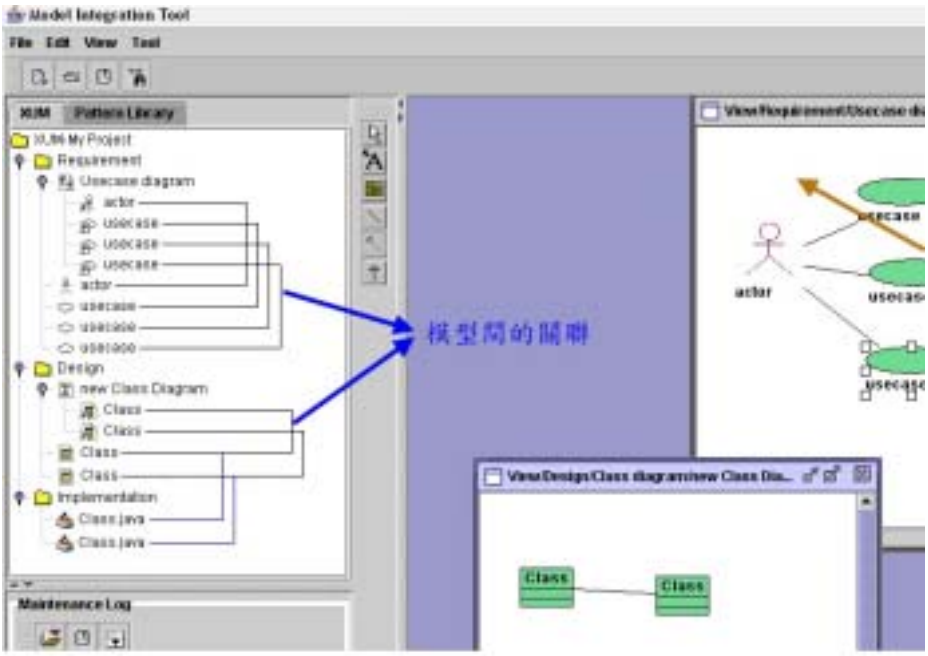

圖 6、統一模型的關聯使用者介面

統一模型的關聯機制是本系統實作出模型整合的方法，可分為三種類型的關聯，包含抽象鏈結(Abstraction link)，整合鏈結(Integration link)及程式碼鏈結(Source code link)。而此三種鏈結都基礎於模型的反映機制。

- 漣漪效應的追蹤

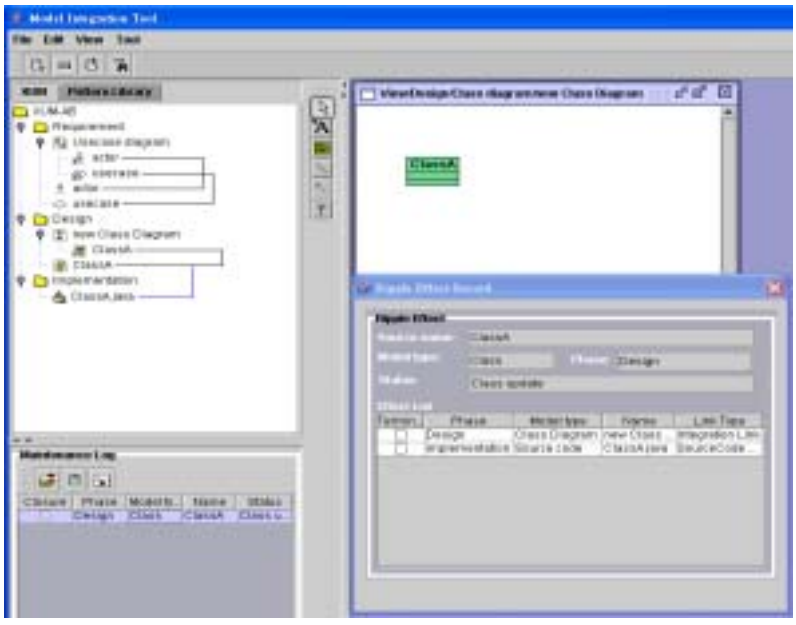

圖 7、漣漪效應的追蹤使用者介面

漣漪效應的產生在於當一個模型改變之後，其相關聯的模型也受到影響。本系統的目的就是在於追蹤漣漪效應，並記錄成歷史資訊，以輔助系統開發人員提昇系統的可維護性。

- **模型編輯工具子系統**

本子系統提供圖形化之介面供使用者建立各類 UML 標準的模型如使用案例圖、類別圖，並將 UML 圖形轉成 XML 文字格式，也可將存放好的 XML 文字格式的資料還原為 UML 圖形。

細部設計 (Detailed Design)

一般而言，將 XML 以物件的方式存取可以利用 DOM，但是 DOM 的處理方式是將 XML 中各個元素(Element)表示成 Node 物件，屬性則表示成 Attribute 物件，Node 物件可包含其他的 Node 物件以處理 XML 的巢狀結構。這樣的處理方式往往太過低階，無法根據該 XML 的資訊定義特定的類別以提供較高階的處理方式。因此本子系統利用 Java Architectural for XML Binder(JAXB)的技術，根據 XML 的定義即 XML Schema 產生一組對應的 Java 類別，稱為 binding class。因此只要對該類別做存取就可以間接存取的 XML 中的資訊。圖 8 為產生 binding class 的過程。圖 8 中的 binding schema 是用來定義出哪些元素會對應至何種類別。

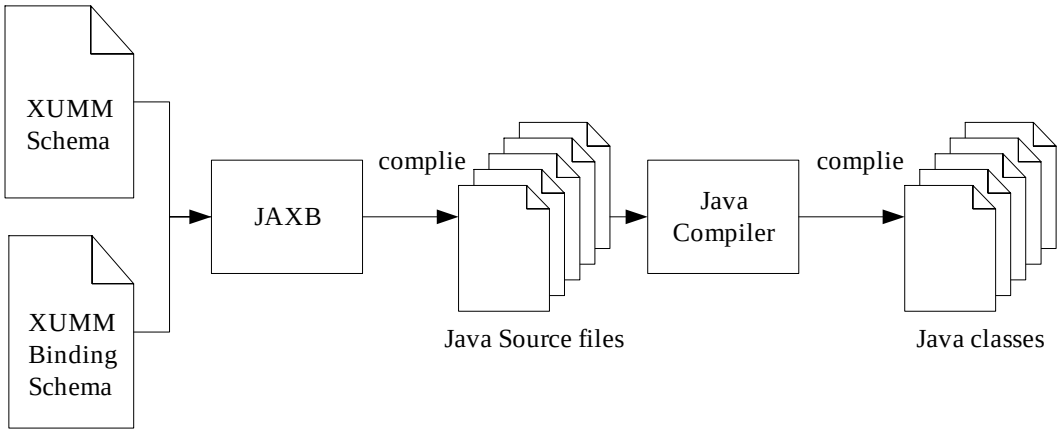

圖 8、產生 binding class 的過程

本子系統提供了圖形化的界面以提供使用者建立統一模型，如圖 9 所示。

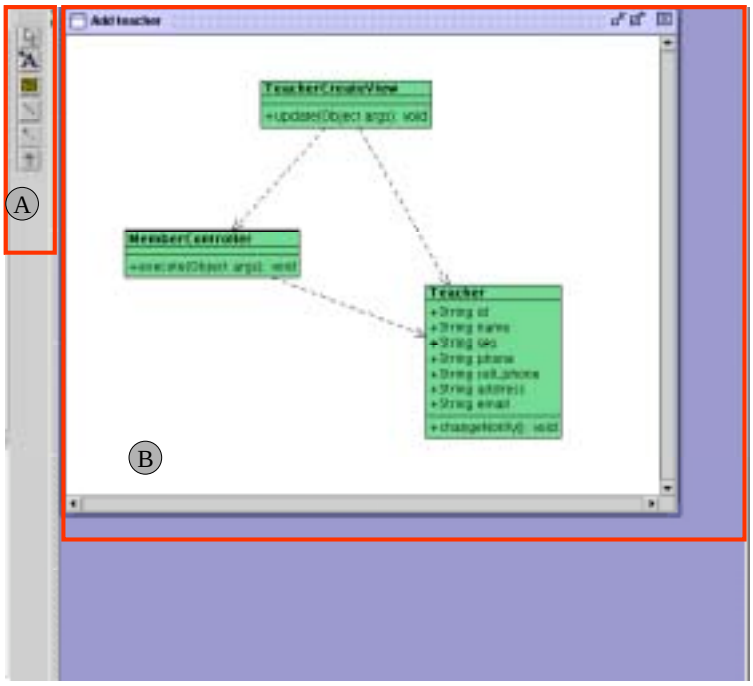

圖 9、圖形化界面

圖 9 中的 A 區塊是模型編輯的工具區，B 區塊則是編輯的結果。以類別圖(Class diagram)為例。圖形介面、圖形介面類別與模型資訊的關係可由圖 10、

圖 11 及圖 12 所示。

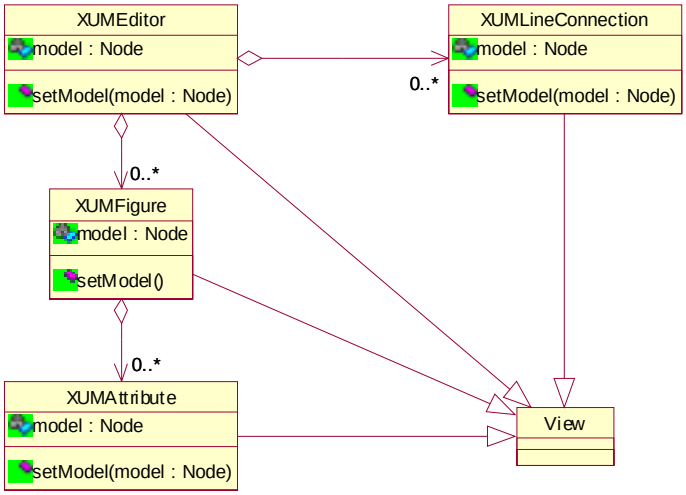

圖 10、圖形介面類別關係

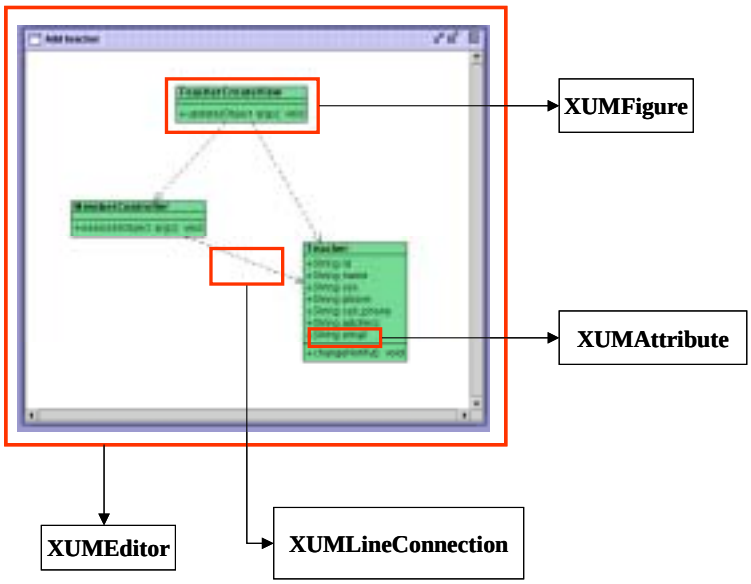

圖 11、圖形介面類別與圖形介面的對應關係

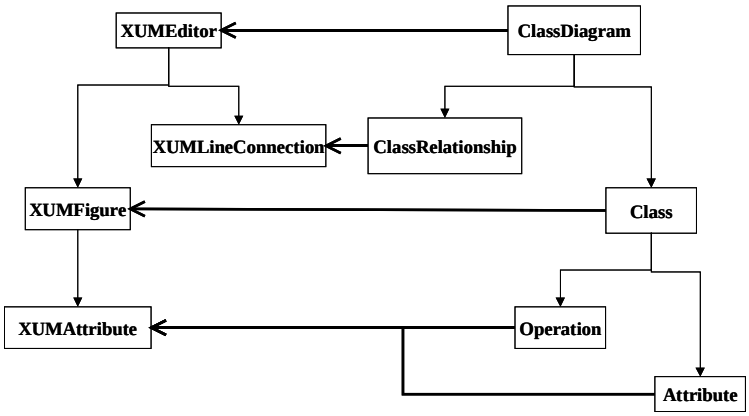

圖 12、圖形介面類別與模型資訊的關係

參考資料：

[1] Booch, G. (1994). *Object-oriented analysis and design with applications* 2nd ed. Redwood City, Calif. : Benjamin/Cummings Pub. Co., 3-25.

[2] Deitel, H., Deitel, P., Nieto, T., Lin, T., & Sadhu, P. (2001). *XML how to program*. Upper Saddle River, NJ : Prentice Hall.

[3] Object Management Group. (2001, August). *OMG unified modeling language specification*. Version 1.4, Retrieved July 16, 2001 from http://www.omg.org/technology/documents/recent/omg_modeling.htm

[4] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Reading, MA.: Addison-Wesley.

[5] Wirfs-Brock, R. J., & Johnson, R. E. (1990). Surveying Current Research in Object-Oriented Design. *Communications of the ACM*, 33(9),105-124.

[6] World Wide Web Consortium (W3C), Extensible Markup Language (XML), Retrieved July 16, 2003 from http://www.w3.org/XML/

[7] Lu C.W., Chu C.W., Chang C.H., Lian W.D., Yang D.L. (2003). Integrating Divers Paradigms in Evolution and Maintenance by an XML-based Unified Model , Journal of Software Maintenance and Evolution

[8] Chu W.C., Chang C.H., Lu C.W., Lain W.D., Chiu H.C. (2002). A CASE Tool Implementation to Improve Software Development and Maintenance. In *Proceedings of the Thirteen Workshop on Object-Oriented Technology and Applications*, pp.193 - 200.