私 立 東 海 大 學

資 訊 工 程 與 科 學 所

碩 士 論 文

指導教授：朱正忠 博士

實作一個電腦輔助軟體工程工具(CASE Tool)以提昇軟體發展效率及軟體可維護性

A CASE Tool Implementation to Improve Software Development and Maintenance

研 究 生：連文達

中 華 民 國 九 十 一 年 六 月

摘 要

由於各種環境架構的快速改變，例如硬體方面像是個人數位助理(PDA)、無線應用協定(WAP)、行動電話、資訊家電…等。軟體技術如物件導向、Java、仲介架構(Middleware)、群組軟體(groupware)…等，新興的服務，像是電子商務(E-commerce)、行動商務(Mobile-commerce)、應用服務提供者(Application Service Provider)、網際網路內容提供者(Internet Content Provider, ICP)…等等的出現，使得軟體系統目前正面臨比以往更多的挑戰。使得軟體需要能快速的time-to-market、在多種不同環境中相互運作、可再使用、跨平台…等。由於現有軟體的高複雜性，使得軟體開發往往必須採用團隊工作，因此軟體發展以及維護程序的標準化變得更加重要。在過去幾年，許多軟體標準紛紛被提出，像是通用模型語言(UML)、設計樣板(Design Patterns)、CORBA 及 COM/DCOM 等，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，用來改善軟體產能與降低軟體開發成本。

個別的標準在軟體開發過程中提供了特定的功能與好處，如 UML 提供了標準的模型表示法、而 Design Pattern 則提供了標準的設計規範。雖然這些標準的確提昇了軟體開發的速度及效率，但也帶來了許多問題。最大的問題是標準與標準間的溝通、交換及同步性。由於這些問題的發生將導致軟體維護的成本提昇、及錯誤的發生率。

在 XML-based Unified Model [1]的理論裡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。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。本論文利用此理論方法並以實作一個統一模型軟體工具(XUM software environment)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。

關鍵詞：*CASE tool、軟體標準、軟體模型、UML、設計樣版、XML、模型整合、軟體維護、軟體再使用*

Abstract

Software systems are now facing more challenges than before due to the rapid changes of hardware, such as PDA, WAP mobile phone, Information Appliance, .. etc, software technologies, such as OO, Java, Middleware, groupware, ..etc, and new services, such as E-commerce, Mobile-commerce, Application Service Provider, Internet Content Provider(ICP), .. etc. Software is now required to be fast time-to-market, evolutional, interoperable, reusable, cross-platform, and many more. Due to the high complexity of current software systems, it usually involves teamwork. Therefore, standardization for the software development and maintenance becomes very important. In the past few years, many software standards, such as Unifying Modeling Language (UML), Design Patterns (DPs), CORBA, and COM/DCOM, and commonly accepted standard mechanisms, such as component based approaches, have been proposed and advocated to improve the software productivity, and then reduce the high cost of software.

Software standards are highly recommended because they promise faster and more efficient ways for software development with proven techniques and standard notations. Designers who adopt standards like UML and design patterns to construct models and designs in the processes of development suffer from a lack of communication and integration of various models and designs. Also, the problem of implicit inconsistency caused by making changes to components of the models and designs will significantly increase the cost and error for the process of maintenance.

An XML-based unified model (XUM)[1] is proposed to solve the problems and to improve both software development and maintenance through unification and integration. In this paper, we providing a XUM software environment that supports XUM approach and provides automation, tractability , and validation to the activities of software development and maintenance.

***Keywords**: Case Tool, software standards, software models, UML, design patterns, XML, model unification and integration, software maintenance, software reuse.*

章 節 目 錄

圖 目 錄

第一章 緒論

1.1 前言

在 e 世代的今天，軟體系統是企業造就成功事業的關鍵因素。但顯而易見的，軟體系統的開發比過去更加的複雜，原因包含了電子商務(E-Commerce)的盛行，許多服務被要求要 Time to Market。再加上無線商務(M-Commerce)的興起，硬體平台不斷的推陳出新，而許多產品常常是硬體已成型，軟體開發整合速度慢造成延遲，再加上現今的軟體被要求跨系統、平台、地域及具有移動性(mobility)，並且能夠隨著嶄新的硬體、OS 及軟體技術的更新腳步，如此一來軟體開發及維護的複雜度及困難度就越來越高。

軟體系統執行效率及彈性固然重要，但軟體的可維護性更是重要。所謂軟體維護是指軟體從出貨後，錯誤的修正、效率的提昇，及因應外界需求的修改而修改。在早期(1950 年至 1960 年)軟體維護的重要性在軟體的生命週期裡較不為人所重視，直到 1960 至 1970 年間，愈來愈多的軟體被開發完成並上線使用，人們開始了解到舊的軟體(legacy system)不容易被置換，而軟體的維護性才便成軟體生命週期中主要的過程[4]。一直到目前為止，當初使用舊的語言，舊的設計方法所開發出來的舊系統數量相當的多，這些軟體中模組間的高偶合度，功能與資料的高相依性提高了該軟體本身的複雜度。到最後，花費在軟體系統的維護比較於當初開發軟體所耗費的時間與成本來的高出許多[6]。而隨著軟體的愈來愈複雜，花費在軟體維護的成本也就愈來愈高。

為了提昇軟體開發的效率與速度及軟體可維護性，許多的軟體標準(software standard)被建議實用於軟體開發流程以提昇軟體的品質，例如 UML(Unified Modeling Language)[34]及 XML(eXtensible Markup Modeling Language)[2，12] 提供了標準的模型表示法及資料格式，使得許多軟體模型的描述得以統一。而 Design Pattern[15]則對於特定的設計問題提供了可再使用的標準設計規範，使軟體設計有一個標準規範可循。Framework 則針對一特定領域的問題提供了一個標準系統架構，並提供了相對的一組程式碼提供程式開發人員客制化的空間，加快程式開發的速度。而就軟體開發的觀點而言，這些軟體標準確實提供了不少的貢獻。

1.2 研究動機

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。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。

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。

1.3 研究目的

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)。本論文利用此理論方法並以實作一個統一模型軟體工具(XUM software environment)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。

1.4 章節安排

在本論文的章節安排中，第二章是描述本論文相關的背景知識及相關研究。第三章則是描述 XUM 的理論方法。第四章說明了 XUM 軟體環境的系統設計。第五章安排了一個使用情節用以說明本論文系統如何落實 XUM 的理論，達成自

動化、可追蹤性及驗證性以提昇軟體開發及軟體可維護性。而結論、未來工作則描述於第六章節。

第二章 背景知識及相關研究

在本章節將簡單的描述軟體開發及軟體維護的背景知識，並說明導入軟體標準所帶來的問題。接著參考了許多相關於軟體開發及維護的文獻以突顯出 XUM 所提出的協統整合模型。

2.1 標準化軟體開發與維護的問題描述

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，其目的就是提昇軟體的品質。但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。

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[40]。

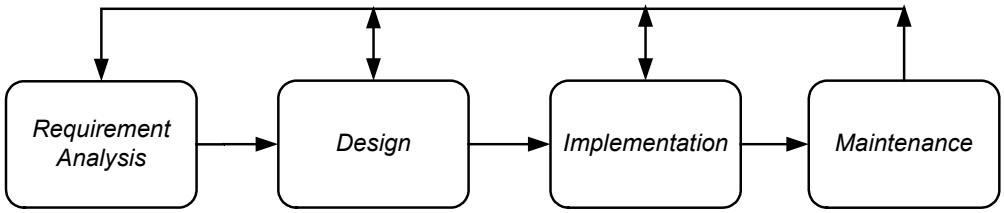

圖 1 軟體生命週期

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。

軟體維護可分為四大類[4]，corrective maintenance 在於導正軟體不符合其既

定規格(specification)的錯誤。adaptive maintenance 為因應軟體本身以外的環境如作業系統、資料庫系統之修改而作的變更。perfective maintenance 發生的情況在於當軟體使用者需求改變時，軟體本身所需做的改變。preventive maintenance 則是在考慮到未來軟體可能會發生問題時所做的預防性修改。

不管所考慮的軟體維護是屬於哪一類，一般而言，軟體維護的流程包含了以下幾點:

- 找出軟體所需修改的地方
- 修改軟體系統
- 處理因軟體修改所引發的連漪效應(ripple effect)
- 軟體測試

軟體維護的重點之一，為達成軟體模型的一致性(consistency)。為了達成這個目的，處理因軟體修改所以發的連漪效應(ripple effect)是最重要的工作。解決連漪效應的問題就是作衝擊分析(impact analysis)，而目前要做衝撞分析必須基礎於程式語言，例如資料流程(data flow)、控制流程(control flow)等等。但是這些方式對於導入軟體標準的系統而言幫助並不大。也因此提高了軟體維護的風險性。

在以下的章節裡，本章節將會針對各種軟體技術及標準作個別的探討，以了解當中的問題。

2.2 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[37, 45]。物件導向分析與設計(OOA/D) [6]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) [34]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嚴謹，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[7]。然而不只 UML，包含所有現有的模型建置技術(modeling techniques)都必須要有一個正規化的方式以達成軟體開發與維護過程中軟體模型的高度整合性。

2.3 物件導向分析與設計的正規化

所謂正規化方法(formal method)乃是利用系統化或數學的方式描述軟體的模型[7]。而其描述的方式則是利用正規化語言(formal language)建立出軟體的需求及設計模型。利用正規化的方法提供軟體開發人員在軟體被實作前，嚴謹的針對軟體模型進行驗證，已確認其完整性及一致性。正規語言所使用的表示法都是相的明確及可驗證，也因此利用正規方法的確加強軟體開發自動化的可能性。

在正規方法的議題上已有相當多的研究，例如以 Z 表示法[41]。Z 表示法利用數學的方式描述模型的資料型態，並利用 predicate logic 描述系統運作的邏輯。而目前為止已經有部分的工具軟體落實了 Z 表示法的機制，例如 ZTC[23]與 ZEN[24]。利用 Z 表示法可以結合軟體分析設計與正規化方法，但是由許多觀點來看軟體分析與設計模型仍然需要非正規或半正規的方式描述例如資料及運作規格[22]。非正規方法的描述方式可以利用圖形化或自然語言的方式提供開發人員描述軟體的模型，不但較容易上手，也使軟體開發的速度加快。然而所帶來的缺點就是軟體模型較容易使人產生誤解[9]。

因為正規語言較不容易使用，因此有些研究試圖提供使用者先以圖形化的方式描述軟體模型，接著將此圖形的描述轉換成正規化的描述方式[9]。而其他相

關的研究則是探索如何在非正規的軟體模型中建立正規的描述方式[14, 17, 18, 31, 38]。在[25]中提出一個雛形的軟體工具-Venus，用以整合 UML 及 Z 表示法[22]。Wang 與 Cheng 提出了一個方式可以有系統的由需求分析至系統設計建立出軟體模型[44]，但是這些方式只有對每個軟體開發階段作局部性的考量，因此往往只適用於開發較特定的系統。

2.4 其他標準化軟體模型建置技術

除了物件導向分析與設計的技術以外，還有其他的軟體技術及標準用以提昇軟體開發的效率。在以下的章節裡將描述設計樣板(Design Pattern)、框架(framework)及元件化軟體開發的技術(Component-Based Software Engineering)。

2.4.1 設計樣板(Design Pattern)

在系統設計的過程中經常可以發現相同的設計概念一而再的被重複使用，而這些設計概念被稱為設計樣板(Design Pattern) [15]。一般而言，設計樣板乃是由一些專家在經過成功經驗的累積，而針對許多經常發生的設計問題所歸納出各類可重複使用的解決方式。因此設計樣板可以說是專家的智慧結晶，而導入設計樣板無疑的可以加強軟體的設計品質，免除了許多人為的疏失。詳細的描述各個設計樣版的特性、使用動機、使用範例可以使系統設計人員更容易使用設計樣板。另外，由於設計樣板明白的定義出該樣板有哪些需求的角色，及其角色間的關係。在物件導向分析設計的過程中，套用了設計樣版時決定了該系統存在了哪些物件，及其合作的關係，如此一來將可以使設計模型的建置更有效率。綜合以上所述，設計樣板可以使系統設計更快且更正確[15]。而設計樣版的相關研究也是相當熱門的議題。

雖然設計樣板有利於提昇系統設計的效率，但它對於系統分析的問題卻無太多的著墨。也留下了許多急待解決的問題，例如設計樣板與其他軟體技術的協調性，及設計樣板間整合性的問題。

2.4.2 框架(Framework)

框架(Framework)是結合了軟體元件再使用及設計再利用的一種軟體技術，框架是基礎於某一存在的架構而用以開發某特定領域的軟體系統，並且大幅的提昇了可重複使用性及開發效率[32]。框架包含了一組抽象類別(abstract classes)、具體類別(concrete classes)及這些類別的介面(interface)定義[8]，而透過各些類別及介面定義的組合關係建立出一個系統架構。系統開發人員也可以利用這些類別及介面透過繼承及實作的方式將此架構課製化成需求的應用系統。基本上，框架提供了一個開發的環境可以根據使用者的需求課製成某應用軟體，但由許多的案例可見，往往是在有限的領域及規模下，框架的使用較為順利。

框架及設計樣板同樣是在達成軟體再利用的目標，但是它們之間仍有差異性。一個框架中往往包含了許多設計樣板以組成一個整體框架結構，除此之外，框架也包含許多既有的軟體元件，即程式碼。然而設計樣板僅止於一個抽象的設計架構，系統開發人員在了解此設計架構之後可以利用各種程式語言實作。就上述的比較可以看出設計樣版就使用上較框架來的彈性。

2.4.3 元件化軟體開發

傳統的軟體開發經常是直接為客戶的需求量身定作。所帶來的好處是此軟體將會完全的契合於客戶所需的商業模型。但是也帶來了許多的缺點。

正如[43]的參考文件所述，軟體元件是組成軟體系統的單位，每個軟體元件都有介面的定義，而介面的定義也正是該元件的使用契約，在該契約的限制下該元件方可被存取。而軟體的運作則是靠元件間的交互合作來完成。軟體元件透過介面定義出該元件所提供的功能(What to do)，但是並沒有限制該元件內部的運作(How to do)。換句話而言軟體元件中介面的定義獨立於元件的內部運作。這也正是軟體元件如何達成可置換性(replaceable)即 plug&play 的特性。

在介面的定義完成之後，元件內部的實作可以利用不同的技術或不同的程式語言完成實作，甚至同一組介面定義下，系統開發人員可以根據環境的需求使用各種外來的元件(third parties)組合出需求的應用系統。就上述的觀點可以比較出

元件與物件(object)之間的差異性。

元件化軟體開發帶了了許多好處，例如開發速度快、容易做到軟體修改、運作穩定等優點。軟體再利用是加快軟體產率的重要方法之一，而軟體重新使用包含了軟體元件的相關研究如元件的定義(identification)、元件的表示法(representation)、元件的擷取(retrieval)及元件的調整及(adaptation)整合(integration)。只有少數的研究[10]有包含全部的過程。而在軟體開發上要落實軟體元件仍然會面臨許多問題，例如軟體元件缺乏一個統一的規格、元件間也缺乏一個整合的標準、而對於商業元件的銷售也缺乏一個統一的標準[39]。因此要利用軟體元件技術達成軟體再利用仍有相當多努力的空間。

2.5 軟體模型的轉換及驗證

現階段軟體模型的技術及標準大都有明確的定義及表示法以提供軟體開發之用。但是對於軟體模型完整性(completeness)及一致性(consistency)則缺乏一個驗證的機制。尤其是當軟體開發人員在某階段的使用某標準要轉換至另一階段使用的另一種標準時，中間過程中往往需要經由人為的方式完成，不但效率低，而如果開發人員的經驗不足則容易破壞模型的完整性及一致性。然而有許多的研究已試圖去解決這方面的問題。在以下的內容中，本論文將針對模型驗證及轉換的一些主題進行討論，包含模型的解析(understanding)、自動化(automation)及驗證(verification)。

模型解析的目的在於提供一個機制用以歸納模型間的一致性，例如設計模型(design model)與實作模型(source code)間是否存在不協調的問題[33]。而做法則是建立系統設計與實作間模型與模型的反映機制(reflection)[1,33]。軟體開發人員可以基礎於這樣的概念針對一個軟體系統作逆向工程，以便由程式碼取出較抽象的軟體模型[46]。

至於自動化軟體開發的部分，目前的技術大部分都必須具備詳盡的設計模型才能夠自動產生程式碼，而且該程式碼往往還需要程式人員經過修改才算完成，軟體自動化的目標在於能夠有系統的由使用者需求建立出系統設計模型，接著找

出所需要的軟體元件。正規化方法是達成此目標的一個方式(formal methods)[22]，但是目前為止大都都僅止於部分自動化的階段。另外也有利用使用者情節及不完整的狀態圖自動的建立設計模型所需較完整的狀態圖[27]。

在模型驗證的部分，有相關的研究可以利用自動化的技術，針對已完成軟體模型之規格進行模擬，已驗證該模型內部及與其他模型的關係是否具不協調性[9]。另一種驗證模型間不協調性的方法是由 OMT 方法所提出的 OSM(Object structure model)。此方法包含了物件模型(Object Model)、動態模型(dynamic model)及功能模型(functional model)，並使用 pre-condition 及 post-condition 的定義[13]。

另外也有利用有限狀態機(finite state machine)的方式來作驗證[3]。首先使用者的需求會被表示成 SCR(Software Cost Reduction)，接著被轉換成 CTL(Computationa1 Tree Logic)，最後進行驗證[13]。

軟體模型的改變會隨著軟體的週期包含分析、設計、修改及維護的階段。而不同的軟體工具會在不同的階段往往會利用不同的標準建立出不同格式的軟體模型。正如同以上的章節所描述，個別的軟體標準都有各自的貢獻，但並沒有任何一種軟體標準可以提供協助於軟體開發裡的各個階段。因此軟體開發人員必須在不同的階段裡使用適當的標準。然而這些標準本身並沒有支援與其他標準整合的特性，因此在不同的階段往往就會產生隔閡，如圖 2 顯示出各種標準在不同的階段間彼此的關係。其中⊠表示針對一特定標準但在不同的開發階段裡需要作模型的轉換。而⊗則是表示標準與標準間在不同的軟體階段容易發生不協調的問題。

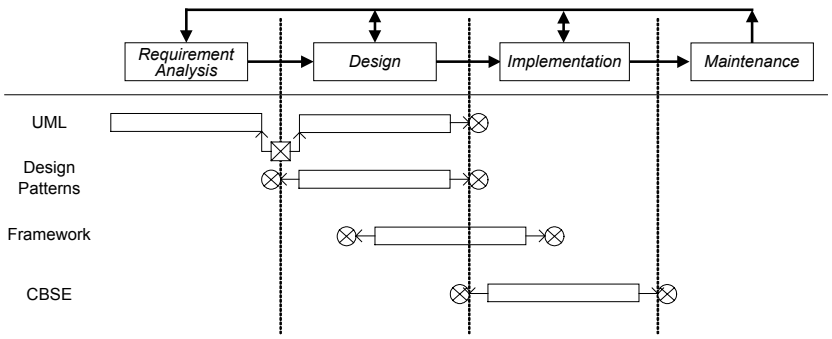

圖 2 各種標準在不同的階段間彼此的關係

新的軟體標準往往是因應於軟體開發的需求而被制定，並且提供某種程度的

貢獻。但不可否認的，單一一種標準僅能從軟體開發的某些角度切入，因此軟體標準間的相容性問題將無法避免。

2.6 eXtensible Markup Modeling Language (XML)

XML[13]是由 W3C 所制定出的一種標準。XML 的好處在於應用中立、具擴充的彈性、可以表現任何複雜的資料、可驗證性及可讀性。XML 包含了以下的部分:

DTD(Document Type Definition):

DTD 是用來定義 XML 的文件規格，包含該 XML 有哪些標籤(tag)、標籤的從屬關係及限制條件。XML 的驗證也是利用 DTD 而得以完成。

XSL(XML Style Language)

XML 用來表示資料，而 XSL 則是用以描述 XML 的外觀風格。XSL 乃基礎於 DSSSL(document style semantics and specification language)與 CSS(Cascading Style Sheet)。

Xpointer 與 Xlink 是用來定義 XML 文件間的鏈結。

在[42]中提出了方法用以將 UML 描述成 XML，稱之為 XML-based UML eXchange Format (UXF)。但是該方法強調於模型資訊的交換，而 XUM 所提出的方法則是強調在各種模型的整合。

2.7 CASE Tools

CASE(Computer Aided Software Engineering) Tool 電腦輔助軟體工程工具，在軟體發展的生命週期中逐漸受到重視 。良好的 CASE Tool 可以幫助軟體在開發、整合、維護的過程提供相當的便利，因此可見其重要性甚至大過硬體環境之良莠。

CASE Tool 的開發，常是針對軟體系統生命週期中不同階段而設計[51]。CASE Tool 可由其功能、角色、支援的環境建構(硬體及軟體)或由來源、價格進

行分類。如:專案管理、程式編輯、雛形製作、分散式環境下物件導向資料庫設計[50]、Date Warehouse 設計[56]、Multimedia 的生命週期呈現[57]等都有相關的 CASE Tool 產品 。

由於購買 CASE Tool 的支出成本高，學習曲線陡峭，因此 CASE Tool 的評估[49] 、選擇適當的 CASE Tool 也成為現今探討的焦點。物件導向技術風行之後，也有 OO (Object Oriented) CASE Tool 的產品[49]問世，並有套用 Design Pattern 來開發、支援 UML 的發展趨勢。

隨著企業需求的日益強烈，且因現今的 CASE Tool，大都只能在局部的軟體開發階段發揮效用[52]，再加上和 PSEE(Process Software Engineering Environments)[53，54]間還是有落差存在[55],因此整合性 CASE Tool (I-CASE Tool)之需求更是日益增高 。

整合性 CASE 環境需要每一個 CASE Tool 完善的軟體資訊、Tools 間標準轉換介面的定義、確立軟體工程師與各 CASE Tool 介面統一的溝通機制，雖說上述問題已有沿生出解決方案，但仍是不夠全面化、自動化。復因軟體開發標準、模型相繼提出，本論文實作出一套軟體工具，期使軟體的開發由需求分析、設計到程式碼的實作，透由此 CASE Tool 達到自動化、可追蹤性及可驗證性的目標。不同以往的，本論文的 CASE Tool 乃是藉由整合軟體發展各階段的標準，將各階段的可在用資訊整合串接起來，進而降低開發成本、加速軟體的開發及軟體的可維護性。

第三章 XUM—基礎於 XML 之整合模型

為了將所有標準整合於一個統一模型，在 XUM(XML-based Unified Model)中利用了 XML 作為模型表現的格式，稱之為。而 XUMM(XML-based Unified Meta-Model)則是用來建立出 XUM 的資料型態。另外，由不同標準所建置的軟體模型都是從軟體開發過程某個角度所見，而在該方法中稱之為” submodel”。經由 XUMM 的轉換，一個 submodel 可以轉換成 XML 的資料格式，在該方法中稱之為 XUM 的”view”。

由圖 3 所示，透過 XUMM，各個 submodel 可以整合於 XUM 而成為整體軟體開發的一個 view。

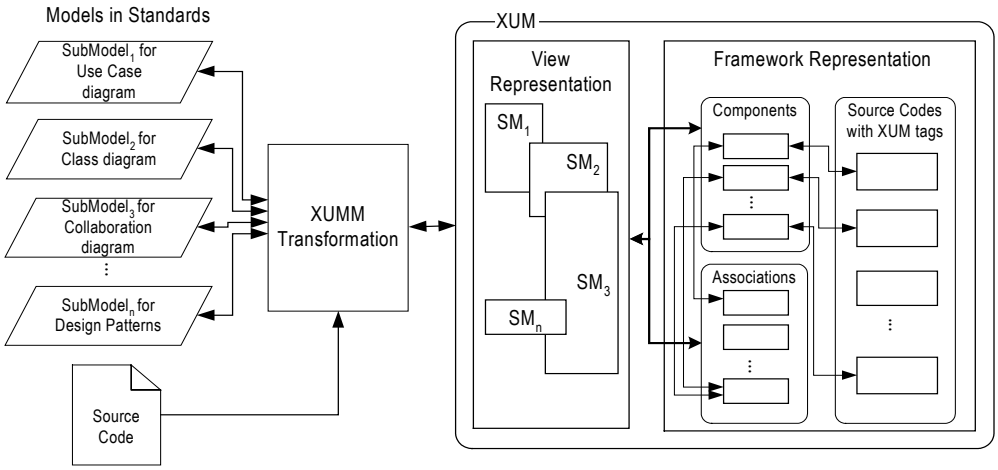

圖 3 XUM 與 submodel 的關係

利用 XUM 將可達成以下目標

- 在軟體開發的過程中，各種標準下的軟體模型都將轉換成 XUM 的 view。
- XUM 整合了不同軟體標準下的軟體模型。
- XUM 提供有系統的模型管理並有利於資訊擷取的機制。
- 在軟體開發及維護中 XUM 提供了自動化的特性。
- XUM 中各個 view 之間可以做同步性的確認(consistency checking)。
- 不同階段下的各種軟體模型可以彼此反映。

以下將明確的討論 XUMM 與 XUM。

3.1 統一模型的資料型態—XUMM

圖 4 顯示出 XUM 裡各個 view 的關係。

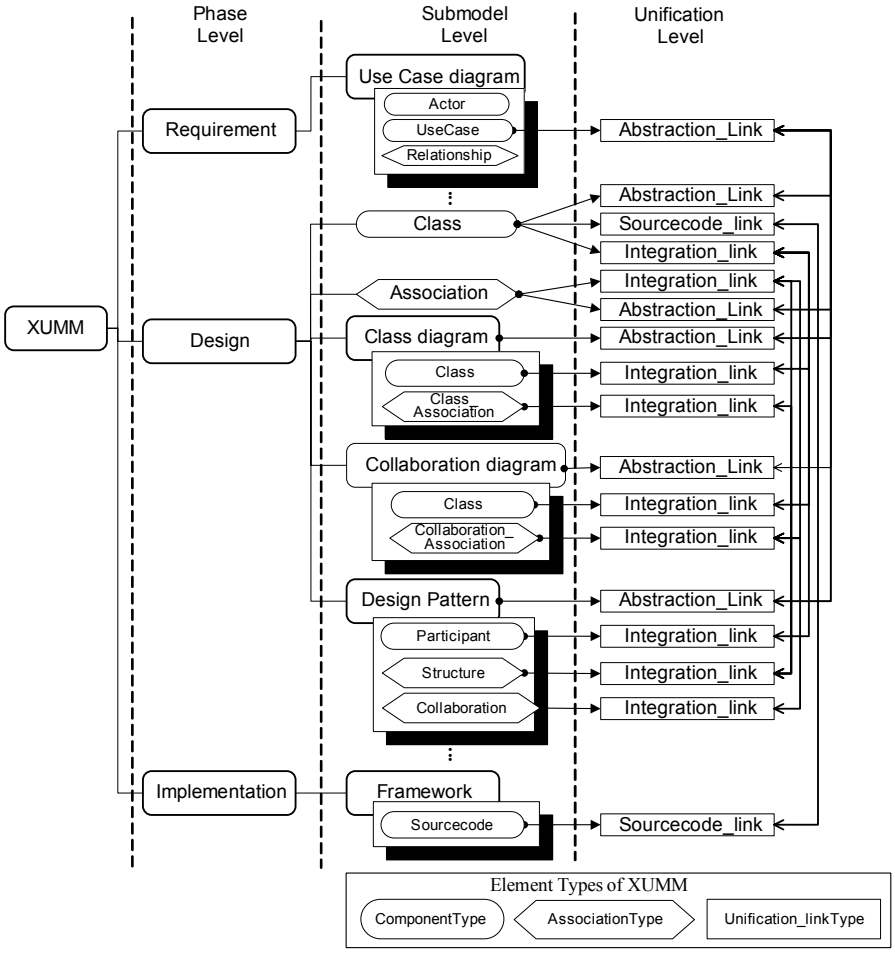

圖 4 XUM 裡各個 view 的關係

XUM 主要包含了兩個部分，其一為軟體開發週期裡所有的軟體模型包含需求(Requirement)階段、設計(Design)階段與實作(Implementation)階段。其二為各個軟體模型的整合關係(Unification Link)。

XUMM 與 XUM 的關係正如同 DTD 與 XML 的關係，XUMM 定義了 XUM 裡該有的資料型態，關於 XUMM 的詳細定義可參考附錄 A 。

XUMM 的定義裡包含三個部份，包含物件(component)、關係(relation)與整合關係(unification relation)。不管是物件、關係或整合關係都存在一個唯一的 ID。在 XUM 中物件代表模型中如 UML 的使用案例(usecase)、類別(class)等等，如圖 5 為需求模型中 Actor 的定義。

```
<!ELEMENT actor (actor_id,actor_name) >
<!ELEMENT actor_id (#PCDATA) >
<!ELEMENT actor_name (#PCDATA) >
```

圖 5 DTD 裡 Actor 的定義

關係(relation)則是代表模型中物件間彼此的關係如使用案例間的使用關係(use)與類別間的繼承關係(inheritance)，圖 6 中定義了使用案例的關係。

```
<!ELEMENT usecase_relationship (usecase_relationship_id,usecase_from,usecase_to,usecase_relationship_type) >
<!ELEMENT usecase_relationship_id (#PCDATA) >
<!ELEMENT usecase_from (#PCDATA) >
<!ELEMENT usecase_to (#PCDATA) >
<!ELEMENT usecase_relationship_type (#PCDATA) >
```

圖 6 關係(relation)的定義

整合關係是用以整合各個模型的機制，其中包含了抽象鏈結(Abstraction link)、整合鏈結(Integration link)與實作連結(Source code link)。例如圖 7 中定義了抽象鏈結。

```
<!ELEMENT abstraction_link (abstraction_link_id,abstraction_link_from_id,abstraction_link_to_id) >
<!ELEMENT abstraction_link_id (#PCDATA) >
<!ELEMENT abstraction_link_from_id (#PCDATA) >
<!ELEMENT abstraction_link_to_id (#PCDATA) >
```

圖 7 抽象鏈結的定義

各種鏈結將分述如下:

- **抽象鏈結(Abstraction link)**

抽象鏈結是用來串聯較抽象的軟體需求模型與較具體的軟體設計模型，簡而言之，抽象鏈結串聯了抽象化程度不統的兩個 view，例如 UML 中描述

需求模型的使用者案例(Use Case)往往會對映至設計模型中多個類別圖(Class diagram)、合作圖(Collaboration diagram)或循序圖(Sequence diagram)。

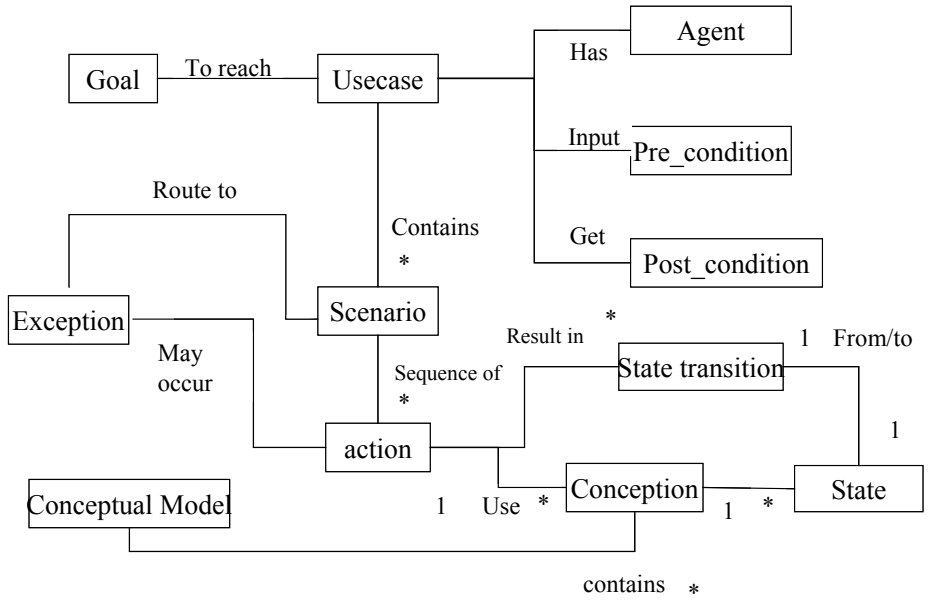

圖 8 需求模型中的資訊

圖 8 為需求模型中的相關資訊，其中各個元素的描述如以下圖表:

Usecase	使用案例，通常描述使用者對軟體系統的需求。
Agent	一個 usecase 包含多個 agent，agent 代表著 actor 或是一個子系統。
Goal	一個 usecase 往往存在一個目標。
Pre_condition	一個 usecase 執行前必須滿足的前置條件。
Post_condition	一個 usecase 執行後可以得到的結果。
Scenario	一個 usecase 根據執行中產生的例外(exception)通常會分出多個情節。
Action	一個情節(senario)由數個 action 串連而成。
Conception	一個 action 執行時會使用到數個 conception，conception 往往代表一個物件。

State transition	一個 action 執行後會使一至多個 Conception 發生資料屬性的改變。
State	一個 conception 往往包含了數個資料屬性，一個 state 會對映到一個資料屬性。
Conceptual Model	Conceptual Model(概念模型)描述現實需求中多的物件的靜態關係。
Exception	一個 action 執行時可能會有例外(exception)的發生

在定義出需求模型裡包含的資訊後，抽象鏈結則是建立這些資訊與設計模型的鏈結關係。至於需求模型的資訊與設計模型中的類別圖、循序圖、合作圖

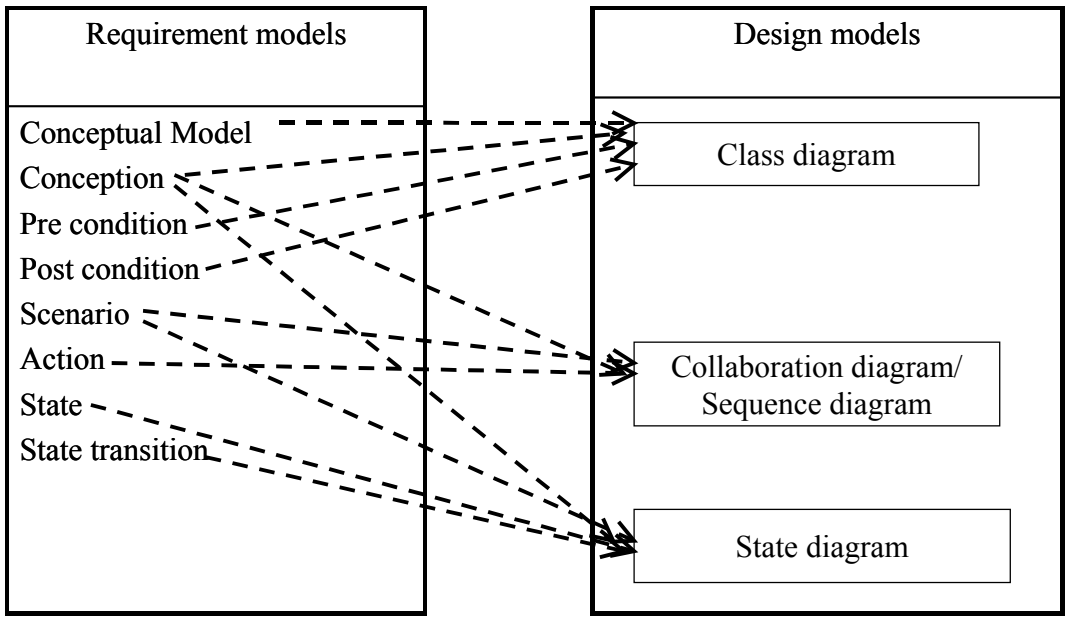

圖 9 需求模型資訊與設計模型對應關係

及狀態圖的對應關係如圖 9 所示。

- **整合鏈結(Integration link)**

在 XUM 裡一個 component 與 association 可能會被使用於多個 view 裡，如此一來這些 view 間存在著整合鏈結的關係，如圖 10 與 11 中分別代表

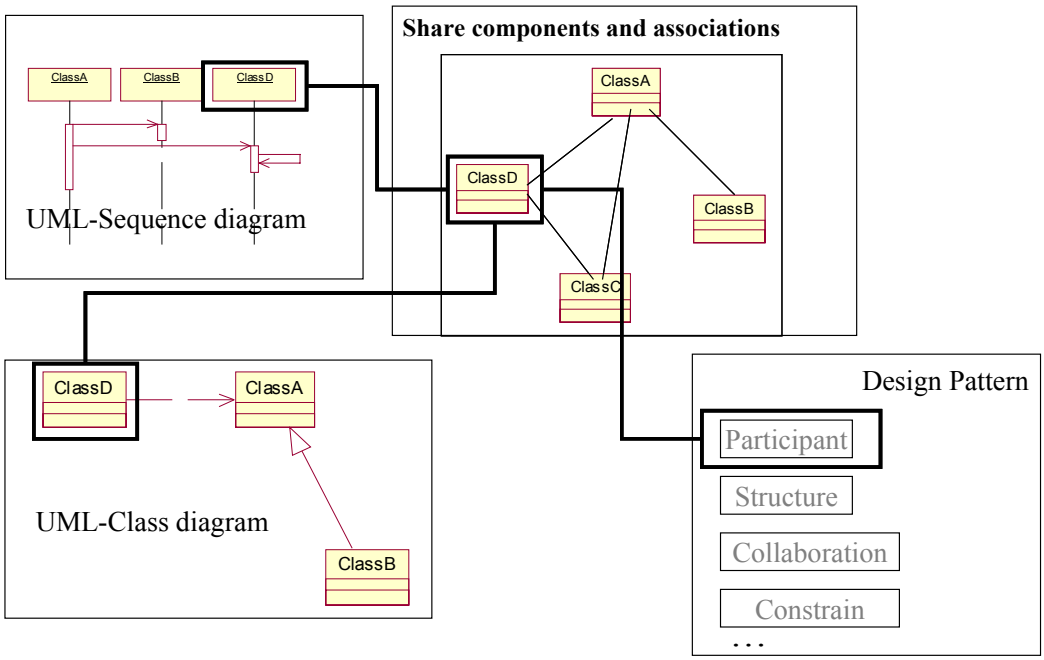

圖 10 整合鏈結中的物件共享

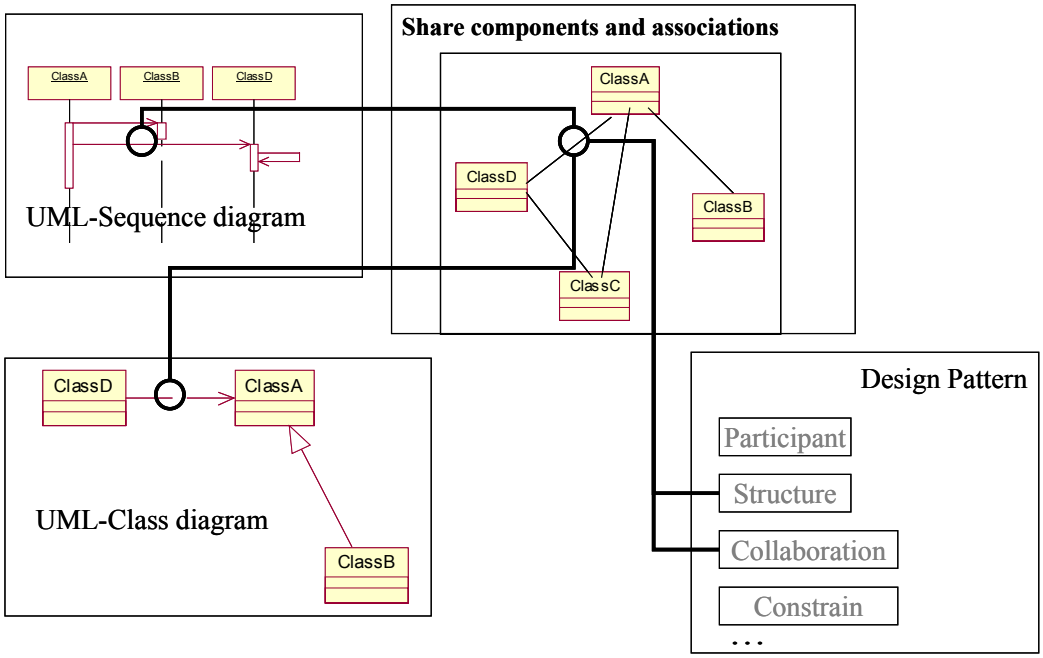

圖 11 整合鏈結中的關係共享

整合鏈結中的物件及關係共享。以圖 10 而言 class d 分別使用於 UML 的

類別圖中的類別、循序圖中的物件及設計樣版中的角色。因此在 XUM 中，這三個 view 具有整合鏈結的關係。而以圖 11 而言 class a 與 b 間的關係分別使用於類別圖中的依賴關係(dependency)、循序圖中的訊息連結(message link)與設計樣版中角色間的結構(structure)與合作(collaboration)關係。

- **程式碼鏈結(Source code link)**

程式碼鏈結則是用來串聯設計模型中的物件與其對應的實作程式碼。如圖 12

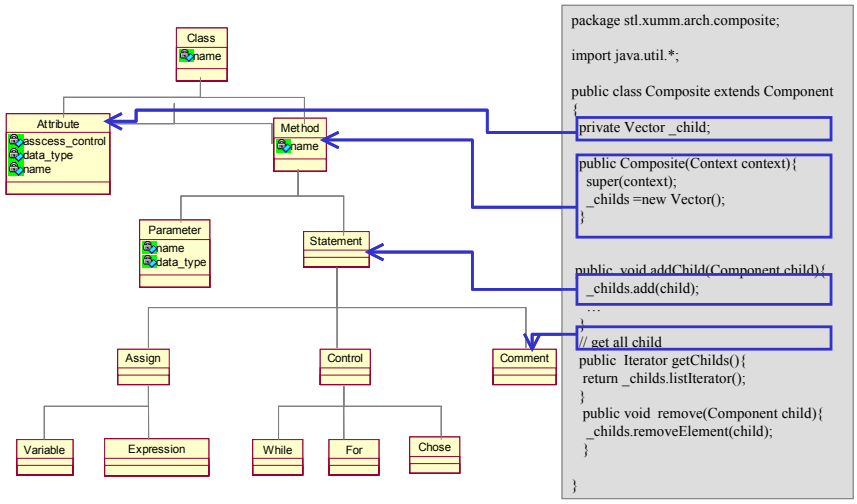

圖 12 由程式碼中擷取出的資訊

顯示出由程式碼擷取出的資訊。而程式碼鏈結的目的則是建立物件及關係與該資訊的對應關係。

3.2 統一模型—XUM

利用 XUMM 的定義可以建立出 XUM 之統一軟體模型，XUM 裡整合了各個軟體階段中利用各個標準建立出的模型。圖 13 顯示了 XUM 裡的結構關係。

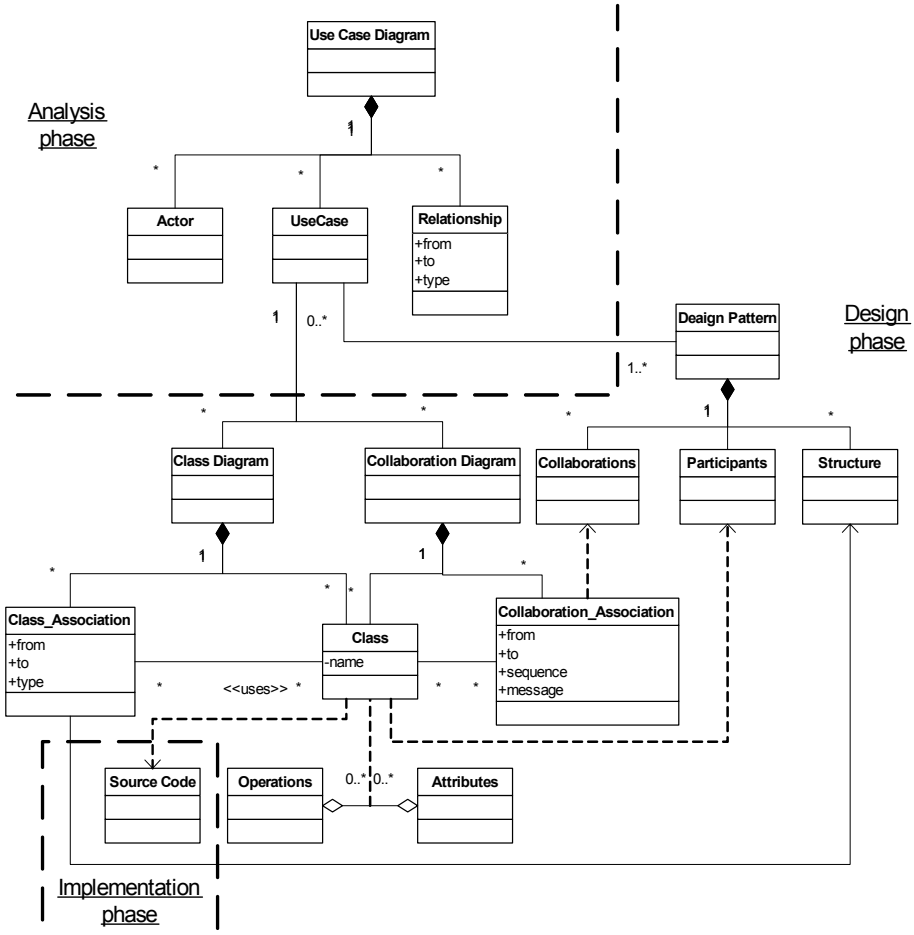

圖 13 XUM 裡的結構關係

由圖 13 可看出各種類型的模型如使用案例圖(Usecase diagram)、類別圖(Class diagram)、合作圖(Collaboration diagram)、設計樣版(Design Pattern)及實做的程式碼等等都包含於 XUM。XUM 裡的整合鏈結提供了各類模型的整合機制，如果模型間存在著抽象關係，XUM 會記錄該模型間的抽象鏈結。例如在本論文使用的範例(詳見於第五章)中，需求模型中的”Add teacher”使用案例(圖 14 中的 A 區塊)與設計模型中”Add teacher”類別圖(圖 14 中的 B 區塊)存在著抽

象鏈結，而抽象鏈結自紀錄於<unidication_link>的標籤之中(圖 14 中的 c 區塊)。

如果軟體模型有使用到相同的資訊時，則 XUM 裡會記錄該模型間存在著

```
<xum>
...
 <requirement>
  <usecase>
   <usecase_id>f1680f:ecfa98b93e:-7fed</usecase_id>
   <usecase_name>add teacher</usecase_name>
  </usecase>                                          A

   ...
 </requirement>
 <design>
  ...
  <class_diagram>
   <cd_name>Add teacher</cd_name>
   <cd_id>523df:ecfabbb50d:-8000</cd_id>
    ...
  </class_diagram>                                   B
  ...
 </design>
 <unification_link>
  ...
  <abstraction_link>
   <abstraction_link_id>9b4748:ed2fd3d826:-8000</abstraction_link_id>
   <abstraction_link_from_id>f1680f:ecfa98b93e:-7fed</abstraction_link_from_id>
   <abstraction_link_to_id>523df:ecfabbb50d:-8000</abstraction_link_to_id>
  </abstraction_link>                                          C
  ...
 </unification_link>
</xum>
```

圖 14 XUM 裡的抽象鏈結

整合鏈結，如範例中同屬於設計模型的類別圖”Add teacher”(圖 15 中的 A 區塊)與”Update teacher” (圖 15 中的 B 區塊)同時使用了”Teacher”類別(圖 15 中的 C 區塊)，則這些模型間存在著整合鏈結，該資訊紀錄於圖 15 中的 D 區塊。

```
<xum>
...
<design>
 ...
  <class_diagram>
   <cd_name>Add teacher</cd_name>
   <cd_id>523df:ecfabbb50d:-8000</cd_id>
   <class_integration_link>
    <cil_id>523df:ecfabbb50d:-7ff0</cil_id>
     ...
   </class_integration_link>
   ...
  </class_diagram>                                          A

   ...
  <class_diagram>
   <cd_name>Update teacher</cd_name>
   <cd_id>523df:ecfabbb50d:-7fff</cd_id>
   <class_integration_link>
    <cil_id>523df:ecfabbb50d:-7fe6</cil_id>
     ...
   </class_integration_link>
    ...
  </class_diagram>                                          B
...
  <class>
   <class_id>523df:ecfabbb50d:-7fef</class_id>
   <class_name>Teacher</class_name>
    ...
  </class>                                                  C
 ...
</design>
<unification_link>
...
  <integration_link>
   <integration_link_id>523df:ecfabbb50d:-7fed</integration_link_id>          D
   <integration_link_from_id>523df:ecfabbb50d:-7ff0</integration_link_from_id>
   <integration_link_to_id>523df:ecfabbb50d:-7fef</integration_link_to_id>
 </integration_link>
...
  <integration_link>
   <integration_link_id>523df:ecfabbb50d:-7fe2</integration_link_id>
   <integration_link_from_id>523df:ecfabbb50d:-7fe6</integration_link_from_id>
   <integration_link_to_id>523df:ecfabbb50d:-7fef</integration_link_to_id>
 </integration_link>
...
</unification_link>
</xum>
```

圖 15 XUM 裡的整合鏈結

至於程式碼鏈結的部分則是串聯設計模型中的類別與對應的程式碼，如圖 16 中設計模型的類別”Teacher” (圖 16 中的 A 區塊)與實作模型中的程式碼”Teacher.java” (圖 16 中的 B 區塊)存在著程式碼鏈結，該資訊紀錄於圖 16

中的 C 區塊。

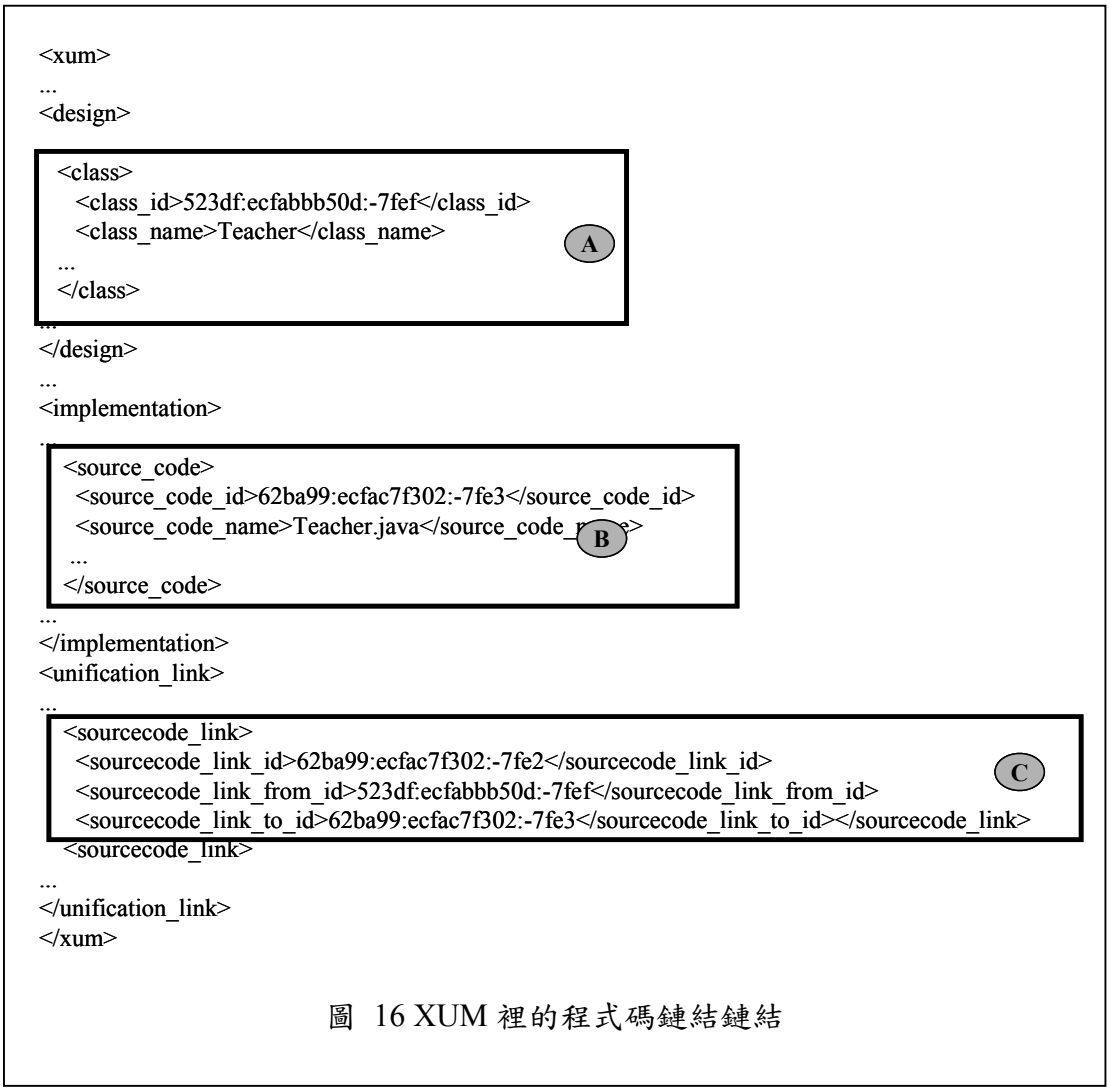

圖 16 XUM 裡的程式碼鏈結鏈結

第四章 工具軟體的設計與實作

4.1 設計與實作的目標

本系統的設計在於完成以下目標:

- **建立軟體模型的建置及管理機制**

本系統的目標之一是以圖形化的操作介面提供軟體模型的建置，系統開發人員可以利用本系統所提供的模型編輯工具建立軟體開發過程中各個階段的軟體模型。產生的各個軟體模型將整合於一個統一模型內，並做到有效率

的管理，如模型的繪製、刪除、修改及儲存成 XML 格式的檔案等。

- **建立模型資訊的擷取機制**

建置於系統內的統一模型包含了各個軟體開發階段的各個模型資訊，本系統利用設計樣版提供的機制，彈性的提供了模型資訊的擷取機制。除了應用在模型資訊的圖形顯示外，也提供了統計資訊的製作等機制。

- **建立追蹤(tractability)的機制以解決模型間的連漪效應**

本系統的實作目的之一是建立出模型反射的機制，包含模型間的抽象鏈結、整合鏈結及程式碼鏈結。在使用者修改軟體模型的過程中，系統將隨時紀錄這些修改的模型，並追蹤其直接或間接關聯的模型，輔助使用者作軟體的維護。

- **建立自動化(automation)以輔助軟體開發**

本系統提供了需求階段至設計階段的自動化及設計階段至實作階段的自動化。在需求階段至設計階段的自動化部分，系統可以將需求階段的資訊適當的轉移至設計階段以輔助系統設計人員建立設計模型。至於設計階段至實作階段的自動化部分，系統則是建立了設計模型與實作程式碼的同步性。

- **建立模型驗證(validation)的機制**

XUM 裡整合了 UML 與設計樣版，由於設計樣版是一種規範性的軟體設計架構，因此當 UML 整合了設計樣版之後，系統設計人員就不能夠在任意的使用 UML 建置設計模型，而必須遵循於設計樣版的規範。因此本系統提供了驗證的機制。在使用者修改設計模型的過程中驗證使用者是否違反了設計樣版的限制。

本系統主要是採用 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

4.2 系統架構

本工具系統的架構如圖 17 所示。

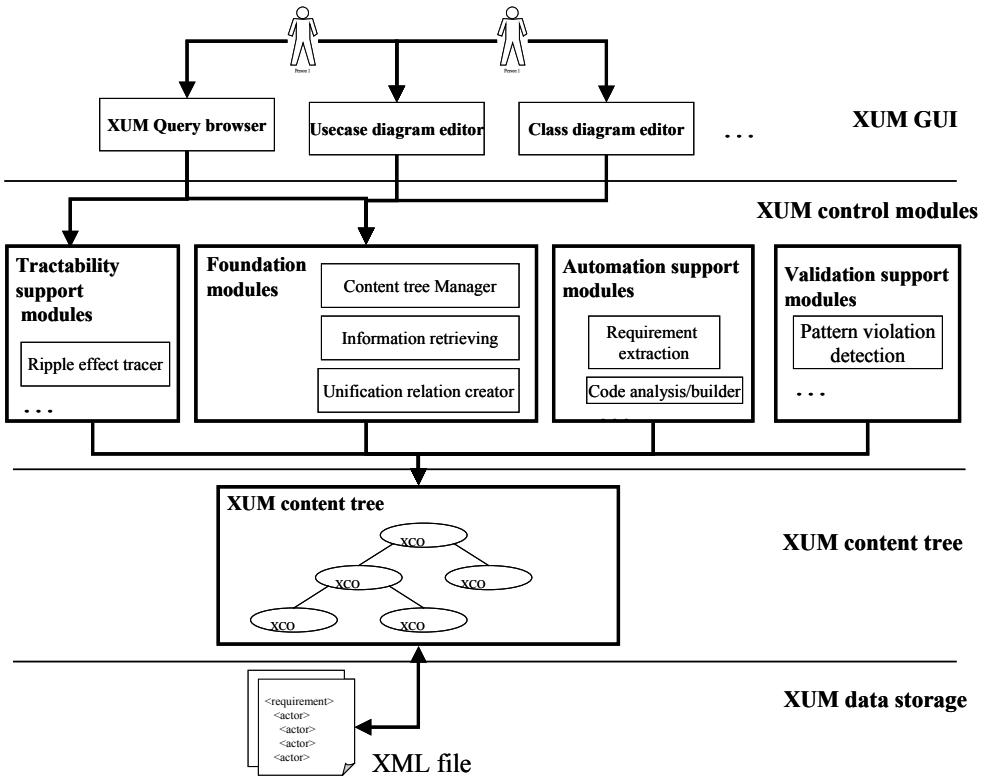

圖 17 系統架構圖

本系統架構主要可分為四個部分，分別為統一模型使用者介面(XUM GUI)、統一模型控制模組(XUM control modules)、統一模型內容樹(XUM content tree)及統一模型資料儲存。

統一模型使用者介面提供了圖形化之介面提供使用者建立各類 UML 標準

的模型如使用案例圖(Use case diagram)、類別圖(Class diagram)等等、也支援了設計樣版的建立及選取機制，統一模型查詢瀏覽工具(XUM query browser)則是提供統一模型的查詢機制。

統一模型控制模組是本系統的核心部分，此部份可分為以下模組:

- 基礎功能模組(Foundation modules)

  基礎功能模組提供了整合模型基本的功能，包含整合模型內容管理(Content tree manager)、統一關聯建立(Unification relation creator)及模型資訊擷取機制(Information retrieving)。

- 可追蹤性支援模組(Tractability support modules)

  可追蹤性支援模組主要提供了模型的可追蹤性，如漣漪效應的追蹤(Ripple effect tracer)。

- 自動化支援模組(Automation support modules)

  自動化支援模組提供了軟體開發的自動化機制包含了支援需求分析至系統設計自動化的需求擷取模組(Requirement extractor)與系統設計至實作自動化的程式碼分析/建立模組(Code analysis/building modules)。

- 驗證支援模組(Validation support modules)

  驗證支援模組主要是提供了模型驗證的機制，如設計樣版與 UML 設計模型間模型驗證的設計樣版驗證模組(Pattern violation detection module)。

統一模型內容樹(XUM content tree)以物件樹狀結構的方式對映至 XML 中的模型資訊，系統的其他模組只要透過管理此內容樹就可以達成間接存取 XML 的機制，以免除複雜的文字解析動作。

統一模型資料儲存是以 XML 的格式將統一模型儲存於檔案中。

在接下來的內容將更詳細的介紹各個模組的設計與實作。

4.3 統一模型內容樹(XUM content tree)

本論文利用 XML 作為統一模型的表現格式,因此 XML 的處理機制相當的重要,而這些處理動作就是文字的解析(Parser)。但由於本系統是以 Java 作為程式開發語言，因此如何將模型中的資訊如使用案例(use case)、類別等表示成物件，而以物件的處理取代繁瑣的文字解析就變得相當的重要。因此本節將討論系統如何將 XML 對映至物件結構的內容樹。一旦建立出內容樹後，系統的其他模組只要透過管理此內容樹就可以達成間接存取 XML 的機制，以免除複雜的文字解析動作。

4.3.1 XUM 與 Java 物件的對應

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。因此本系統利用了 Java Architectural for XML Binder(JAXB)的技術。

利用 JAXB 主要可分為兩階段過程，第一階段是根據 XML 的定義即 DTD 產生一組對應的 Java 類別，稱為 binding class。例如在本系統中使用案例的開始及結束標籤 (<Usecase>..</Usecase>)中包含著使用案例的資訊，而對應的 UsecaseBinder 類別將對應至此標籤所包含的資訊。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 18 為產生 binding class 的過程。圖中 binding schema 是用來定義出哪些元素會對應至何種類別。

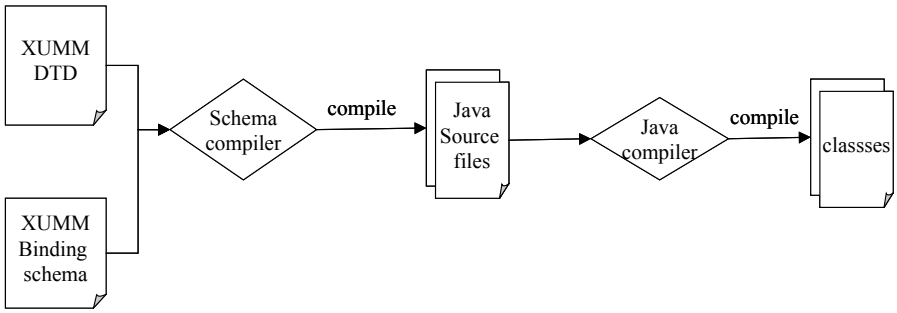

圖 18 產生 binding class 的過程

```
…
<element name="usecase" type="class" class="UsecaseBinder" root="true"/>
<element name="usecase _id" type="value" />
<element name="usecase _name" type="value" />
…
```

圖 19　Binding Schema

如圖 19 所示，usecase 標籤內的資訊將對應至 UsecaseBinder 類別內的成員變數。

在第二階段中可以利用 binding class 產生物件，並以物件的處理取代文字的解析。透過物件的表現方式可將文字格式的 XUM 建立一個由物件組合出的樹狀物件結構(Content Tree)如圖 20 所示。因此對於 XUM 的管理即可直接使用物件的方式進行操作，而無須不斷的針對繁瑣的 XML 文字進行解析動作。

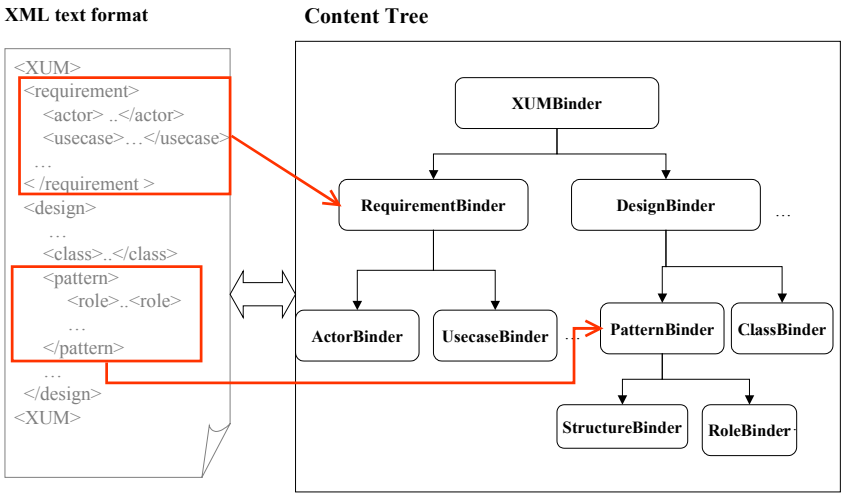

圖 20 由文字格式轉換成物件的樹狀結構

4.4 統一模型資料儲存

系統開發人員利用本系統所建立的所有軟體模型都必須要能夠儲存於檔案中以便於系統再次啟動時所有軟體模型及其鏈結關係能夠再次被編輯。在系統實作上利用 JAXB 所提供的功能可以將 Java Object 的 Content Tree 與 XML

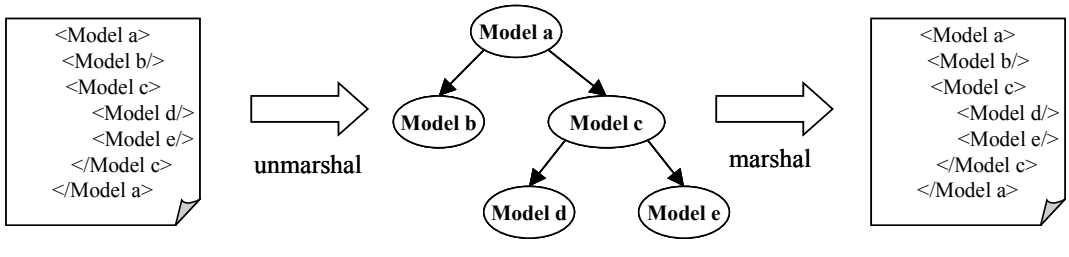

圖 21 軟體模型的永存

之間作轉換的動作，如圖 21 所示。

以下為 XML 檔案與物件樹間轉換的程式碼:

unmarshal:

```
FileInputStream xumFile = new FileInputStream("C:\\java\\xumm\\default\\my_project.xum");
XUMRootBinder xumroot = XUMRootBinder.unmarshal(xumFile);
```

marshal:

```
XumBinder my_project =(XumBinder)xumXCO.getContext();
String         p_path = my_project.getProjectPath();
File xumFile = new File(my_project.getProjectPath());
if(xumFile.exists()){
      int              index    =p_path.lastIndexOf('\\');
      String          dir       =p_path.substring(0,index);
      File             dirfile=new File(dir);
      dirfile.mkdirs();
}
FileOutputStream fos = new FileOutputStream(xumFile);
my_project.validate();
my_project.marshal(fos);
```

4.5 統一模型控制模組

統一模型控制模組提供了統一模型的管理機制，其包含了四個部分，分別描述於以下內容:

4.5.1 基礎功能模組

在 4.3 中討論了如何將文字格式的 XML 表現成物件結構,接下來的問題是如何管理此物件結構，及如何做到彈性的模型資訊處理與擷取及模型間關聯的機制。在以下的內容將對此問題作討論。

4.5.1.1 統一模型內容樹(Content tree)管理

利用 JAXB 的機制所建立出的物件結構雖然可以建立一個樹狀結構，但是在此結構中的每個節點都對應至一個特定類別，一旦要對結構中的某節點作處理，例如新增、刪除節點，往往都必須先判定該節點屬於何種類別。在整個軟體模型包含需求分析、系統分析與實作中有如此多種型態，判斷的動作勢必相當繁瑣。

為了解決上述問題，本系統導入 Composite 設計樣版，並對此樣版做了適當的調整，調整完的設計結構如圖 22 所示。在此設計結構中，個別的 binding class 都會被包裝於一個 XUMCompositeObject 類別，該類別提供了 getContext、setContext 的方法取其封裝的 binding class。而 XUMCompositeObject 彼此間則

存在父、子節點的階層關係。如圖 23 所示。利用此設計結構，模型結構的管理在每個節點擁有統一的基本介面如新增、刪除節點，及下一節所介紹的資訊走訪機制，在圖 23 中 XCO 代表 XUMCompositeObject 產生的物件。

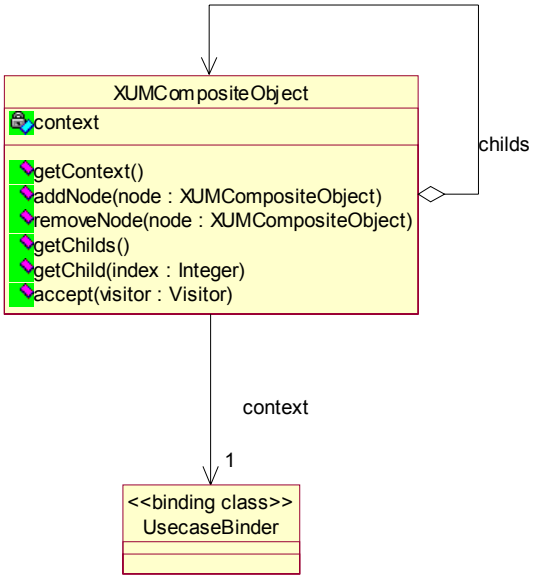

圖 22 導入 Composite 設計結構

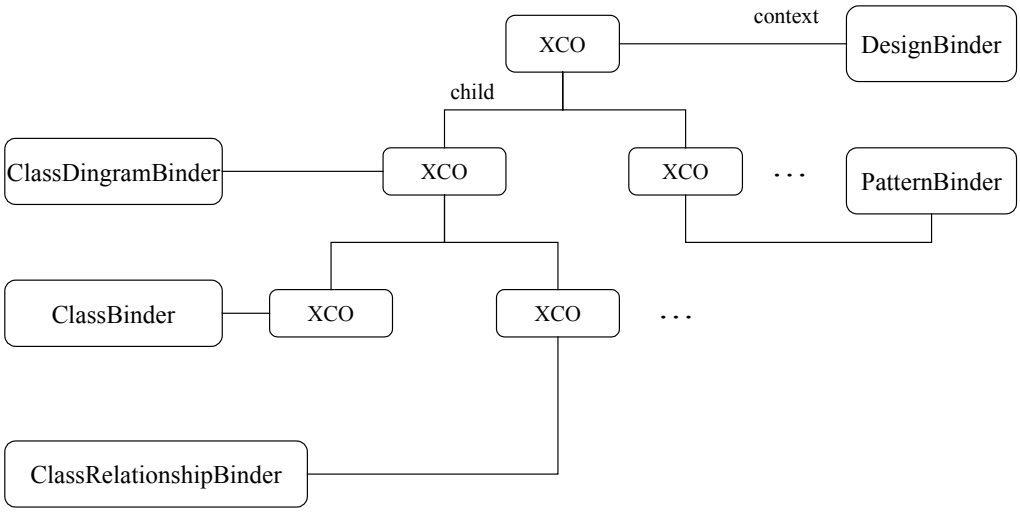

圖 23 利用 XUMCompositeObject 類別包裝成的物件結構

4.5.1.2 模型資訊擷取機制(Information retrieving)

在利用 Composite 設計樣版建立出樹狀結構之後，接下來的問題是如何建立資訊的擷取機制。由於整個 content tree 包含太多類型的資訊，如果因為特別的需求而必須新增節點類別 XUMCompositeObject 的方法(method)，在實作上必然相當的複雜，而面對將來陸續新增的軟體模型，也將使系統不易維護。基於這個問題，本系統使用了 Visitor 設計樣版。其設計架構圖如圖 24 所示。在該架構中，節點類別 XUMCompositeObject 實作了 visitable 的介面，visitable 介面提供了一個 accept 的方法，該方法必須傳入一個 Visitor 物件當參數，Visitor 物件可以在此 content tree 中的各個節點進行深先搜尋的走訪，如圖 25 所示。不一樣的 visitor 可以根據不同的需求及參數在特定節點執行特定的動作。以該圖為例，ClassRetrivingVisitor 及 UsecaseRetrivingVisitor 可以在走訪完整個 content tree 後取得所有的類別及使用案例。

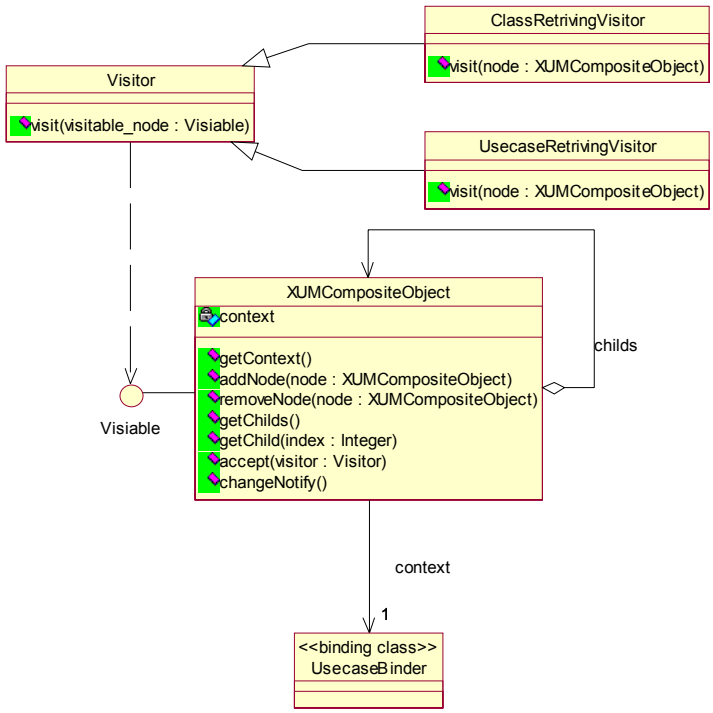

圖 24 導入 Visitor 設計結構

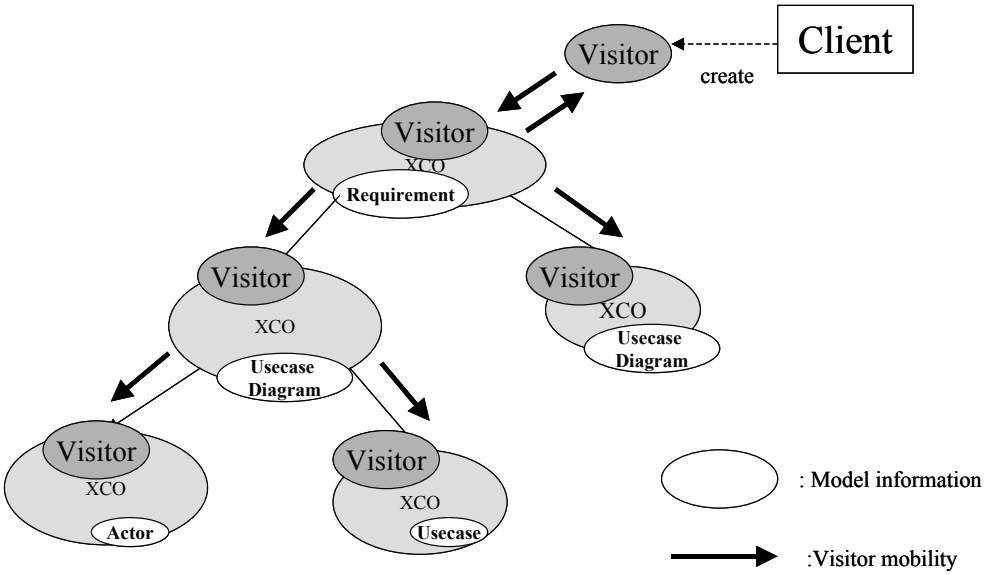

圖 25 Visitor 的走訪機制

4.5.1.3 統一關聯機制(Unification relation)

統一模型的關聯機制是本論文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。

4.5.1.3.1 模型的反映機制

軟體模型間的反映機制即當某一軟體模型經過改變之後，其相關聯的軟體模型也必須跟個改變，以達到軟體維護的過程中各個軟體模型達到同步的機制。為了實作出模型同步的特性，本系統使用了 Observer 設計樣版。圖 26 為其設計架構圖。在其設計架構中 XUMCompositeObject 類別實作了 Observable 介面，而 ReflectionHandler 則實作了 Observer 介面。因此當 XUMCompositeObject 類別內部封裝的模型資訊改變之後，ReflectionHandler 物件會被通知，並呼叫另一個 XUMCompositeObject 物件做改變。換言之，一個關聯的產生是在兩個 XUMCompositeObject 物件中建立一個 ReflectionHandler

物件，該物件會察覺改變的軟體模型，並通知相關聯的另一個軟體模型，如圖 27 所示。

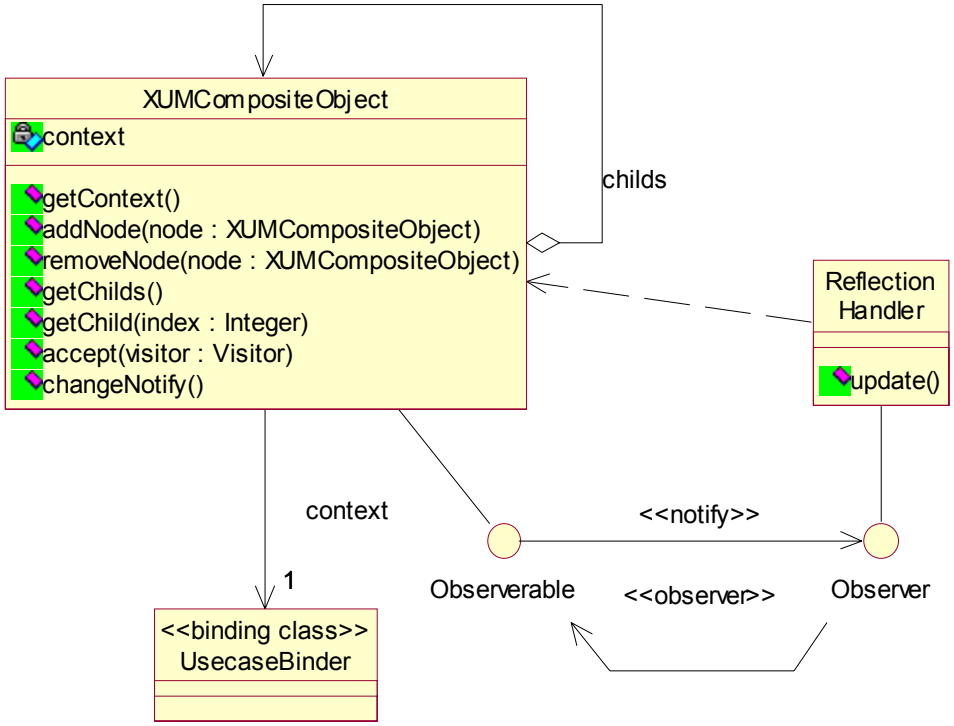

圖 26 模型的反映機制設計架構

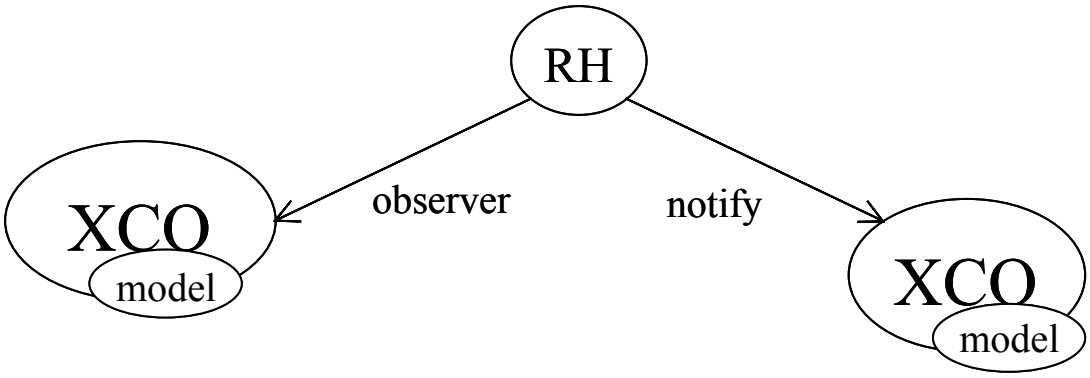

圖 27 模型的反映機制

4.5.1.3.2 抽象鏈結(Abstraction link)

抽象鏈結是用來串聯抽象的需求模型至較具體的設計模型。抽象鏈結的設計架構如圖 28 所示。在該設計架構中 AbstractionLink 類別繼承了 ReflectionHandler 類別並且建立了一個需求模型與設計模型的關聯。

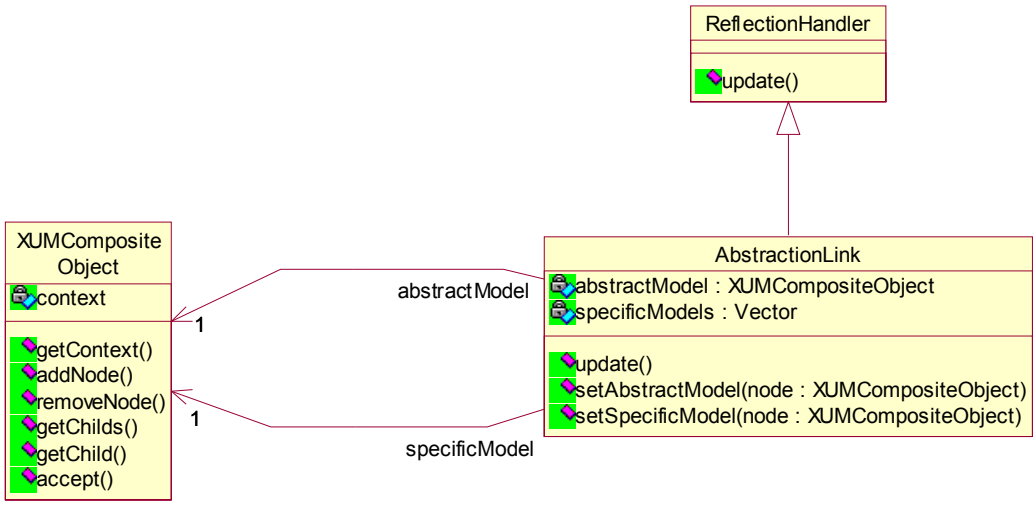

圖 28 抽象鏈結設計架構圖

4.5.1.3.3 整合鏈結(Integration link)

整合鏈結的目的在於將相同的階段，具共用性質的軟體模型做關聯，例如多個類別圖往往共用到多的類別，則這些類別圖則存在整合鏈結 。其設計架構如圖 29 所示。在整合鏈結的設計架構中，IntegrationLink 類別同樣是繼承 ReflectionHandler，在 IntegrationLink 類別中存在兩個變數分別是整合的模型-Integrator Model 與被整合的模型-Integrated Model。

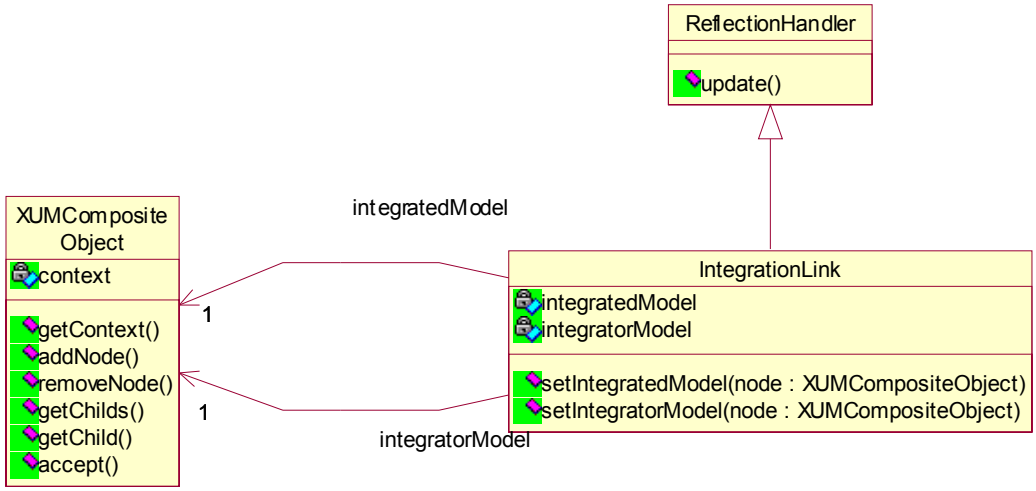

圖 29 整合鏈結設計架構

4.5.1.3.4 程式碼鏈結(Source code link)

程式碼鏈結則是用來串聯設計模型中的類別與其對應的實作程式碼，透過程式碼鏈結，類別的資訊與其對應的程式碼可以達成同步性。程式碼鏈結的設計架構如圖 30 所示。同樣的，SourceCodeLink 類別繼承了 ReflectionHandler 類別，並包含了類別的模型與程式碼的模型。當類別的資訊改變後，其對應的程式碼會被通知且做改變，相反的，當程式碼改變後，其對應的類別資訊也會被通知且做改變。

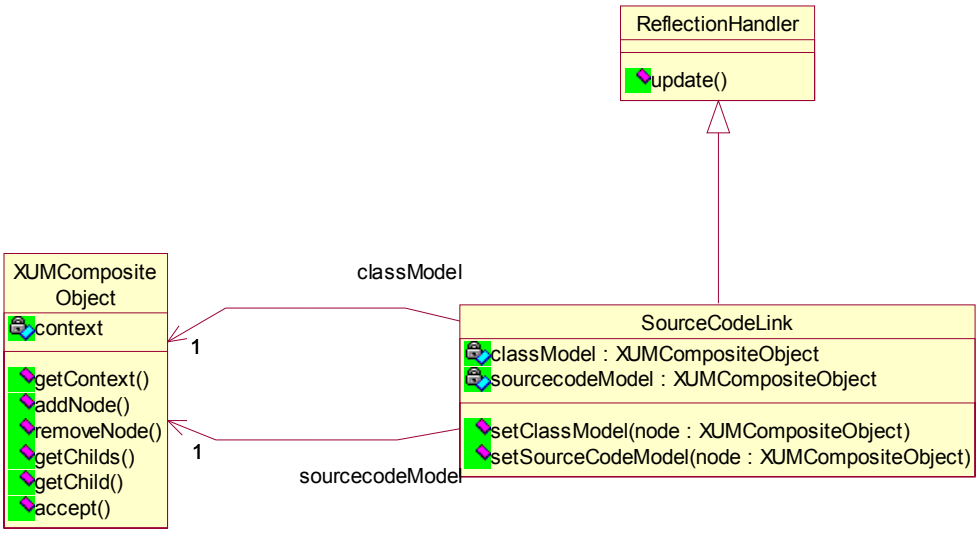

圖 30 程式碼鏈結設計架構

4.5.2 追蹤支援模組

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。本系統的目的之一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。漣漪效應的追蹤機制如圖 31 所示。以該圖為例，四個點 XCO A、B、C、D 彼此具有關聯。當 A 裡的模型改變時，透過 RH B 將會被通知，而當 B 改變時，C 與 D 也將透果 RH 被通知。而當 RH 被啟動反映機制時，RH 會向 Ripple effect tracer 記錄其資訊以建立漣漪效應的紀錄表。

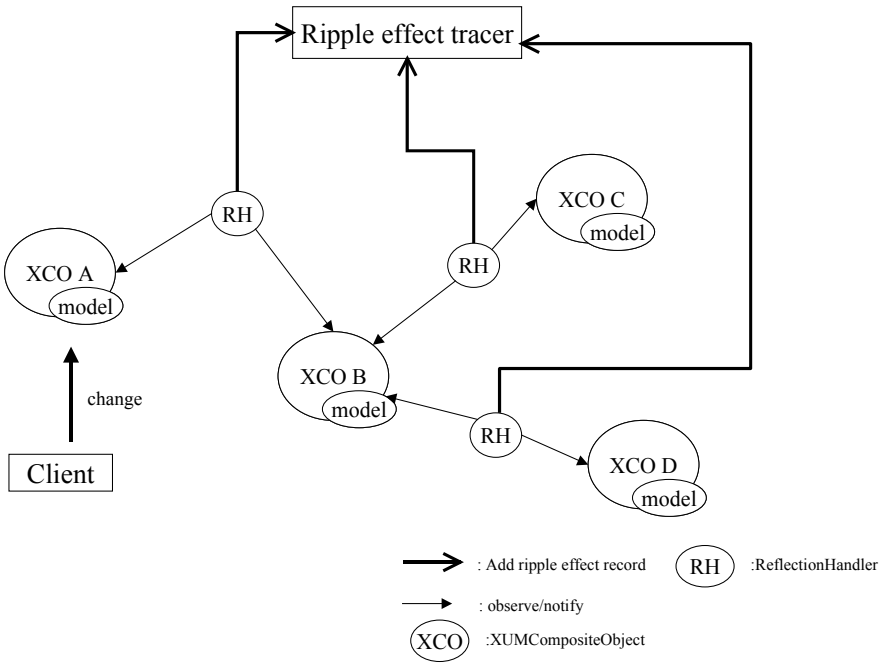

圖 31 漣漪效應追蹤

漣漪效應紀錄表的設計結構如圖 32 所示。在設計架構中，RippleEffectTracer 負責監控各個 ReflectionHandler，並將結果及時的更新 RippleEffectModel。RippleEffectModel 包含了許多個 EffectRecord。一個 EffectRecord 物件代表著一

個模型的變更，它並且包含一個 EffectSource 與多的 EffectTarget。EffectSource 代表模型變更的來源，它記錄了模型的相關資訊包含模型的名稱、所屬軟體開發階段、模型的型態及該模型的改變屬於何種修改動作-EffectStatus。EffectTarget 則是紀錄受影響的其他模型，包含模型的名稱、所屬軟體開發階段、模型的型態及該模型與起源的模型是屬於三種關聯種的何種鏈結。

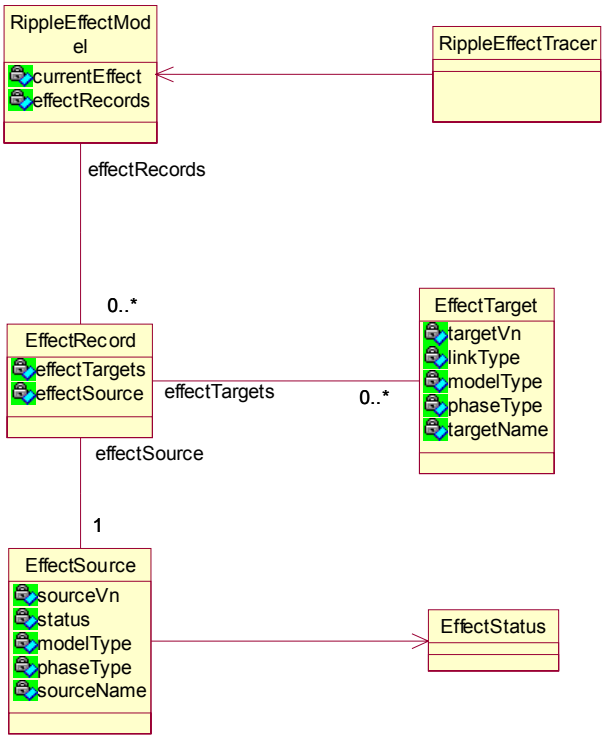

圖 32 漣漪效應紀錄表設計結構

4.5.3 自動化支援模組

自動化支援模組提供了軟體開發的自動化機制包含了需求分析至系統設計自動化的需求擷取模組(Requirement extractor)與系統設計至實作自動化的程式碼分析/建立模組(Code analysis/building)。

4.5.3.1 需求分析至系統設計的自動化

在需求分析至系統設計的自動化部分，由 3.1 節中可知需求分析的資訊及其與設計模型的對映關係。而實際上系統的運作機制如圖 33 所示。

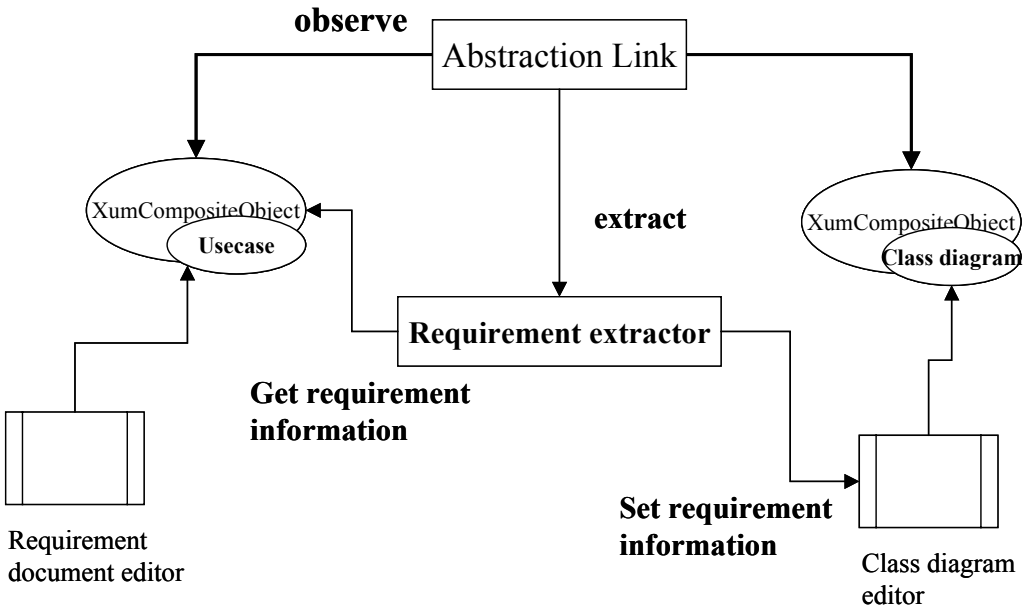

圖 33 需求分析至系統設計的自動化機制

在該機制中，需求分析人員利用需求編輯工具(Requirement document editor)建立 usecase 內的資訊，包含 precondition、scenario、action 等等。在改變了需求模型後，繼承 reflection handler 的 abstraction link 會被告知並且呼叫 requirement extractor 擷取 usecase 內的資訊至對應的設計模型編輯器，如類別圖編輯器(class diagram editor)。而系統設計人員便可以由編輯器中取得相關的需求資訊，以輔助進行系統的設計。

4.5.3.2 系統設計至實作的自動化

在需求分析至系統設計的自動化部分的運作機制如圖 34 所示:

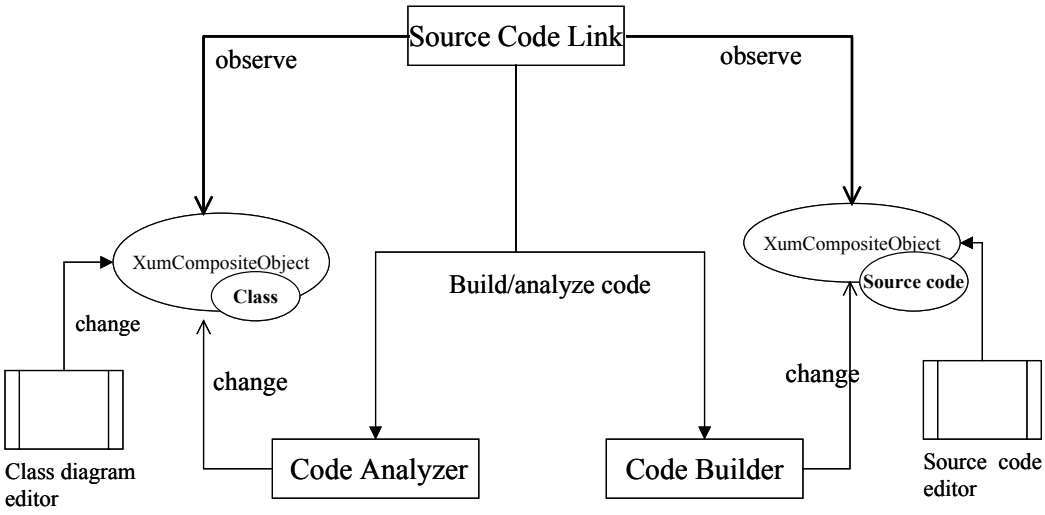

圖 34 系統設計至實作的自動化機制

當系統開發人員透過類別圖編輯介面修改類別內的資訊後，繼承至 reflection handler 的 source code link 會將修改的類別傳至 code builder 更改程式碼，修改完的程式碼將顯示於程式碼編輯器中。相反的當系統開發人員由編輯器中修改了程式碼，source code link 會將修改完的程式碼傳至 code analyzer，code analyzer 在分析程式碼中的資訊後會將資料更新至對映的類別。因此透過 source code link，設計模型與其對映的程式碼可以達成同步的機制。

4.5.4 驗證支援模組

XUM 裡整合了 UML 與設計樣版，由於設計樣版是一種規範性的軟體設計架構，因此當 UML 整合了設計樣版之後，系統設計人員就不能夠任意的使用 UML 建置設計模型，而必須遵循於設計樣版的規範。在驗證支援模組中，系統提供了設計樣版的驗證機制。在使用者修改設計模型的過程中驗證使用者是否違反了設計樣版的限制。

圖 35 是 XUM 中關於設計樣版-Mediator 的資料描述，在 Mediator 中包含

了

….<role>
<role_id>51ef4e:ed76425b08:-7fed</role_id>
<role_name>Mediator</role_name>
</role>
<role>
<role_id>51ef4e:ed76425b08:-7fec</role_id>
<role_name>Colleague</role_name>
</role>
<constrain>
<constrain_id>51ef4e:ed76425b08:-7feb</constrain_id>
<role_from>Colleague</role_from>
<role_to>Colleague</role_to>
<constrain_type>no dependency</constrain_type>
</constrain>

圖 35 設計樣版中的資料描述

兩個角色，分別是 mediator 與 colleague，所有的 colleague 都由 mediator 所管理，colleague 不能跟 colleague 有直接的關係。因此在圖 35 中定義了< constrain >的標籤，描述了 mediator 具備的結構限制。

圖 36 為設計樣版的驗證機制，系統設計人員利用整合鏈結串聯了類別圖與設計樣版，因此當系統設計人員變更了類別圖的結構關係，integration link 會察覺並擷取設計樣版的限制定義交由 violation dectector 進行驗證。一旦確認了類別結構的修改違反了設計樣版規範之後，integration link 會將此資訊傳至 ripple effect tracer，並建立一筆 ripple effect 的紀錄。

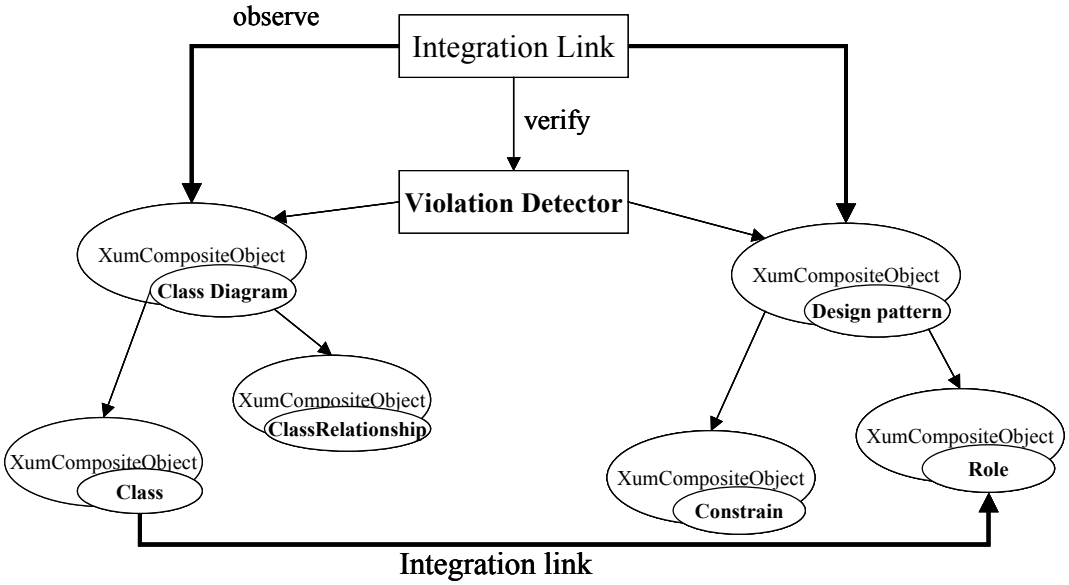

圖 36 設計樣版的驗證機制

4.6 統一模型使用者介面

本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 37 所示。

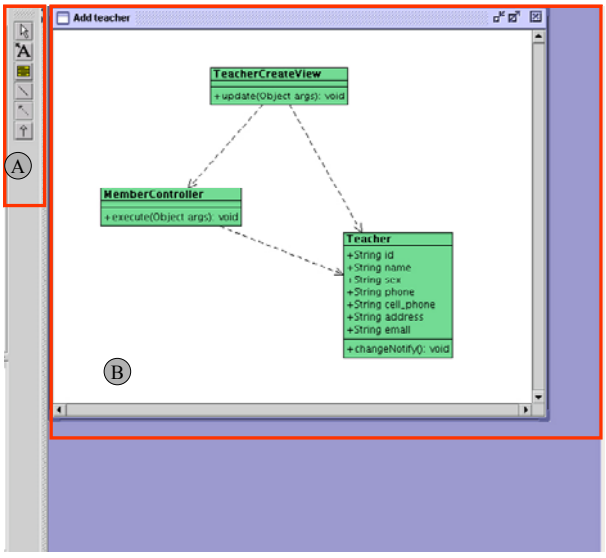

圖 37 圖形化的界面

圖中的 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 38、圖 39 及圖 40 所示。

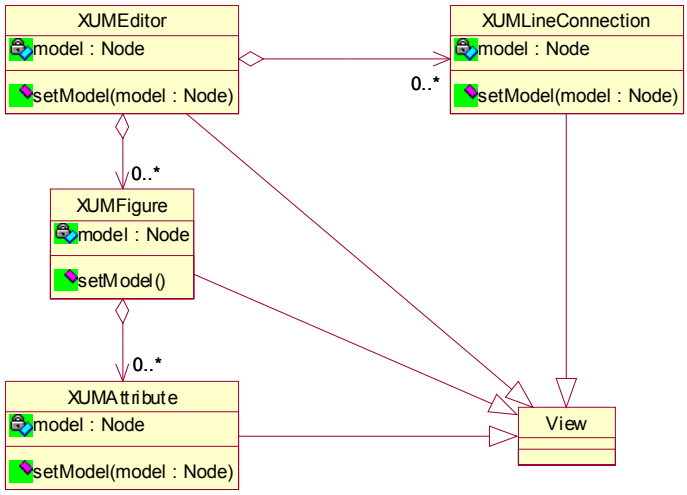

圖 38 圖形介面類別關係

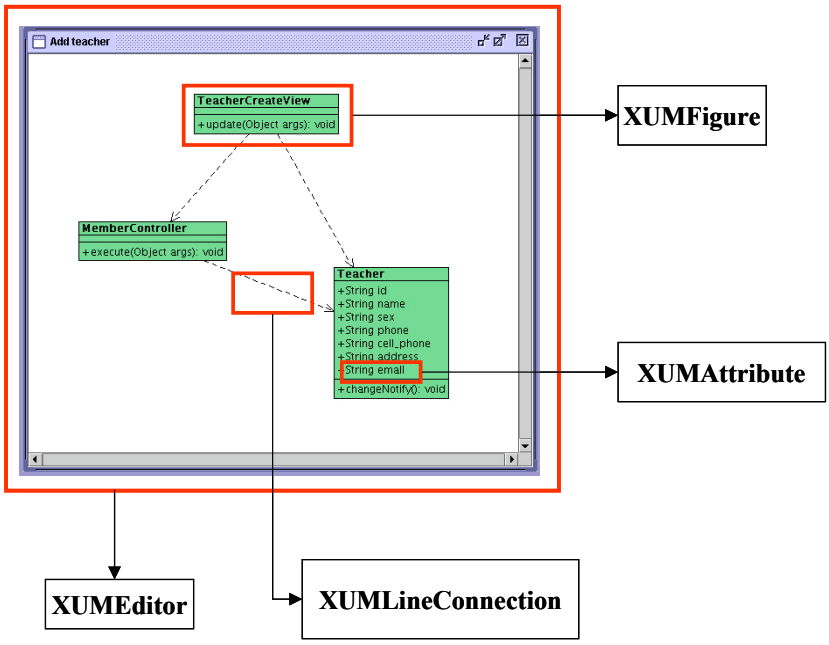

圖 39 圖形介面類別與圖形介面的對應關係

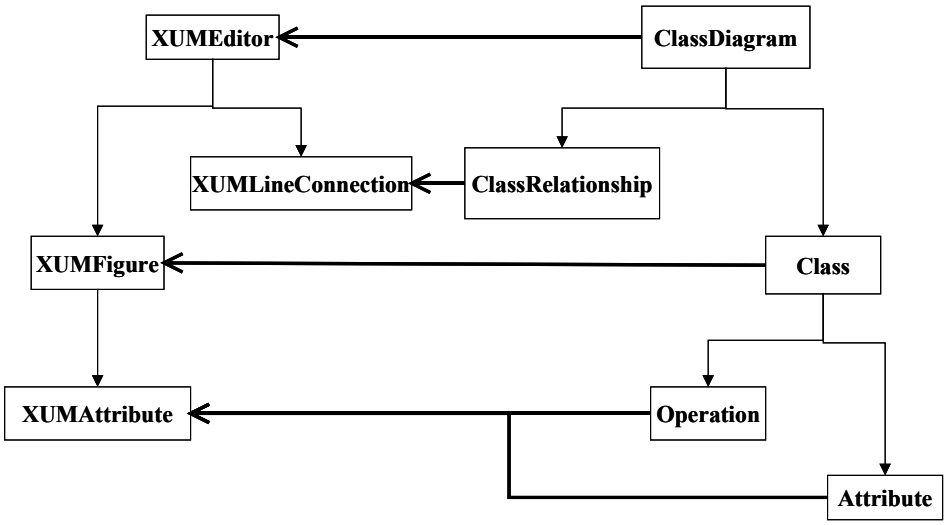

圖 40 圖形介面類別與模型資訊的關係

第五章 系統操作範例

在本章節中，將以一個實際的範例系統-課程管理系統。本章節的開始會先簡介此範例系統，並統計出系統裡包含的軟體模型類型及相關資訊。接著是利用本論文實作出的工具軟體從三個部分對軟體模型作修改。第一部份是更改類別內部的屬性，並顯示出跨模型的 ripple effect。第二部分是由類別間的結構關係去做改變，這個例子將展示跨標準的 ripple effect。第三部分是對程式碼作修改，由程式碼的修改反映到設計部分的軟體模型，接著反映到需求部分的軟體模型而達成跨階段(phase)的 ripple effect。

5.1 範例系統-課程管理系統

此範例系統的目的在於提供課程管理的功能，包含學生及教師的資訊管理、課程資訊管理、學生成績管理及群組通訊系統等等子系統，如圖 41 所示。

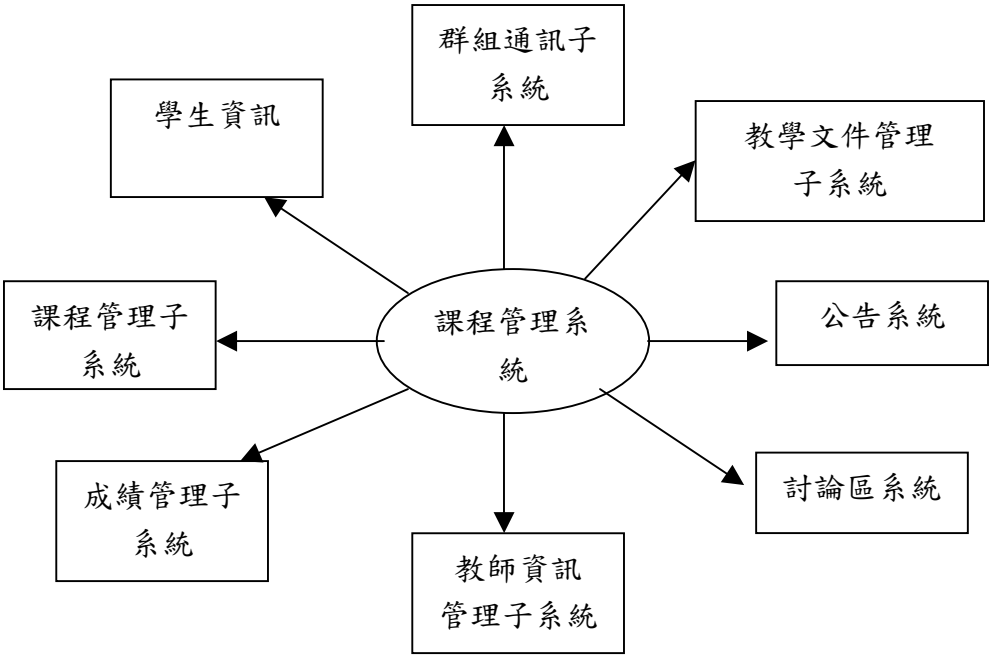

圖 41 課程管理系統架構圖

該系統在經過統計共包含8的使用案例圖、17個類別圖及76個類別等共約5651行程式碼，統計資料如圖42所示:

Phase	Model type	Total	Note
Requirement	**Use case** ● Add teacher ● Update teacher ● Query teacher ● Remove teacher …	28	
	Use case diagram ● Teacher management ● Student management ● Course management ● Grade management ● Group communication ● Document management ● Announcement ● Discussion	8	

Design	**Class** ● Teacher ● Student ● Course ● TeacherCreateFrame ● TeacherEditorFrame ● TeacherQueryFrame ● CourseaManager ● StudentList …	76	
	Class diagram ● Add teacher ● Update teacher ● Query teacher …	17	
Implementation	**Source code(line of code)** ● Teacher.java(70 loc) ● Student.java(197 loc) ● Course.java(181 loc) ● TeacherCreateFrame (267 loc) …	76	Total 5651 loc

圖 42 軟體模型統計資料

5.2 模型整合環境工具軟體

圖 43 為本論文所提供的 XML 為基礎之模型整合環境工具軟體(以下簡稱本工具軟體)執行畫面。其中 A 區塊為該範例軟體各個階段的模型，及模型間的

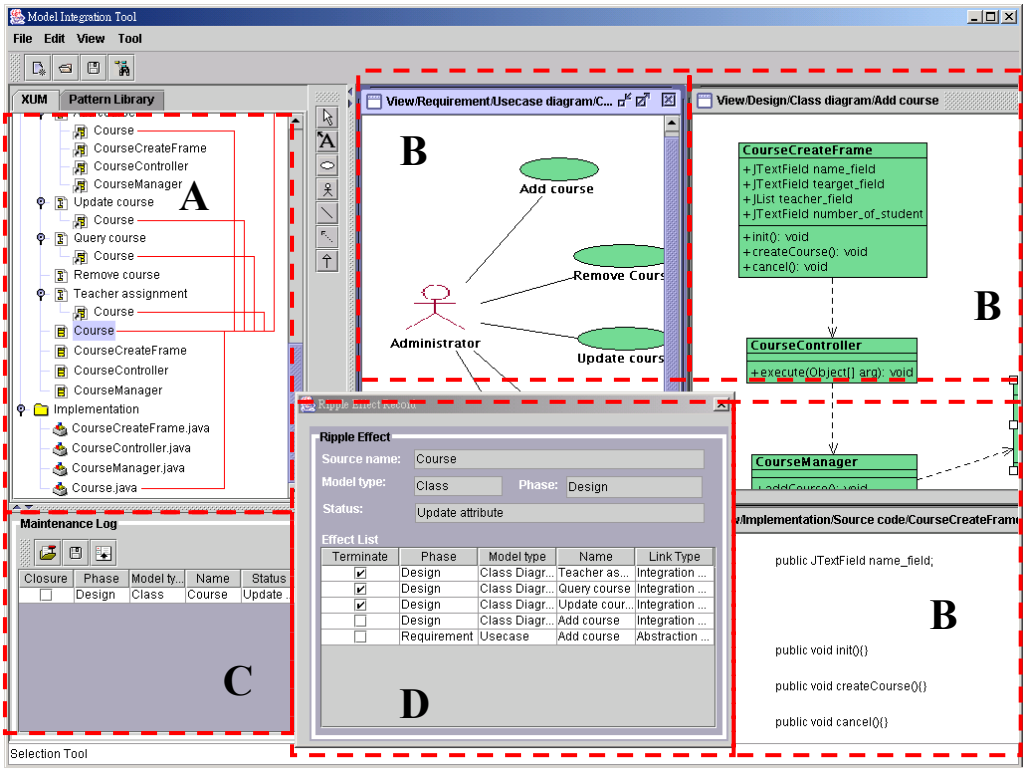

圖 43 工具軟體執行畫面

鏈結關係。B 區塊為個別模型的 View，C 區塊則是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。D 區塊主要是記錄著區塊 C 中個別選項的詳細內容，並記錄著該模型的修改所影響到的其他 View。在接下來的內容則是介紹如何利用本工具軟體提昇軟提可維護性，軟體維護的對象則是 5.1 所介紹的課程管理系統。

5.3 設計模型的修改-修改類別內部的屬性

在此範例系統的課程管理系統中提供了課程的建立、修改、查詢等功能。目前的系統所定義課程的資料包含了課程名稱、對象、修課人數等等，但在之後的系統維護卻發生課程資訊需要加入 teacher qualification 的資訊以紀錄該課程所需的授課資格。為了符合此需求，開發人員決定直接從設計模型中修改課

程類別，為其加入新屬性。

為了快速的找到該類別，開發人員利用本工具軟體提供工具-XUM Query Browser。此工具提供了模型的查詢機制，查詢 Component 的整合鏈結資訊。在範例中設計人員輸入的 Component type 為 Class、Component name 為 Course，並查詢 Course 類別所有的整合鏈結(Unification link)。查詢的結果列出 Course 類別的整合鏈結包含類別圖如 Add Course、Update Course 等等。如圖 44 所示。

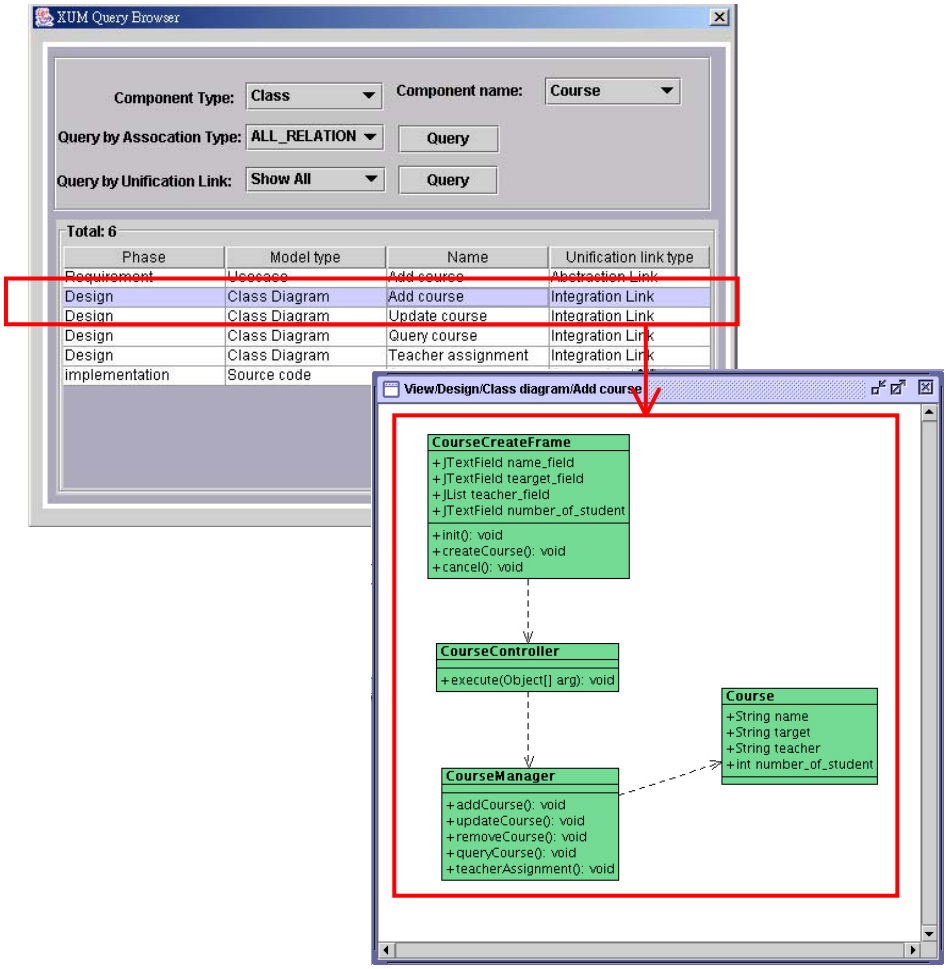

圖 44 利用 XUM Query Browser 查詢機制

接著設計人員在類別圖-Add course 中為 Course 類別新增了一個屬性-teacher_qualification 之後，如圖 45。修改完即產生了一筆 Maintanance log，如圖 46 的 A 視窗。Maintenance log 裡記錄了修改的模型屬於設計模型、模型型

態為類別、名稱為 Course、修改的動作是新增屬性。點選此筆記錄可以看

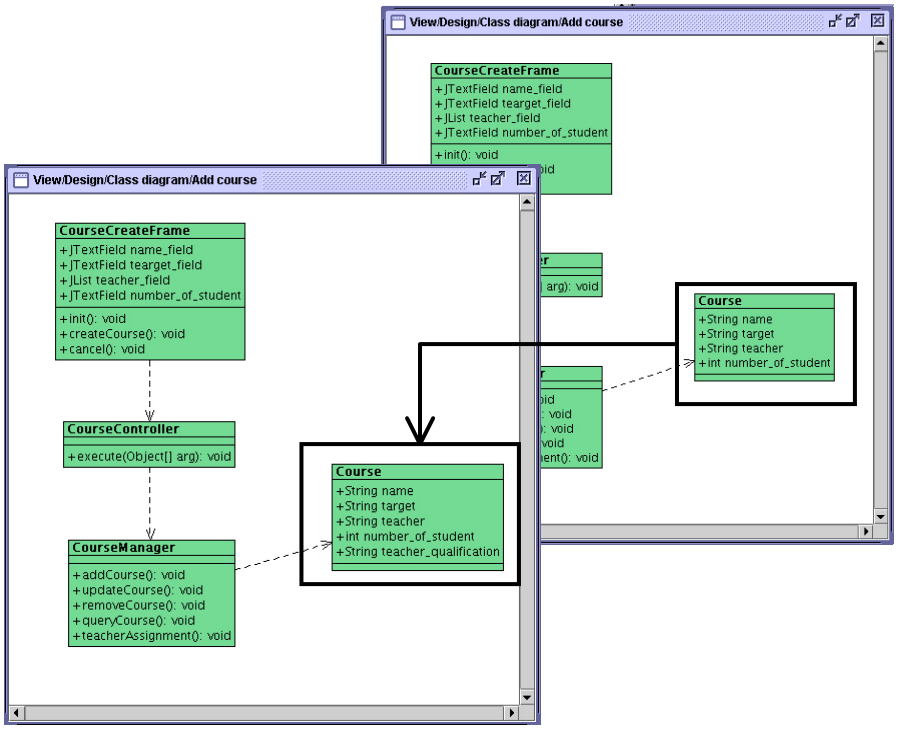

圖 45 新增類別屬性

到更改 Course 類別後產生的 Ripple Effect 資料，如圖 46 中的 B 視窗。其中包含了與 Course 類別存在整合鏈結(Integration link)的各個類別圖。如 Add course、Update course、Course register、Course scheduling 等。另外也包含了與 Course 存在抽象鏈結(Abstraction link)的需求模型及程式碼鏈結(Source code link)的程式碼。

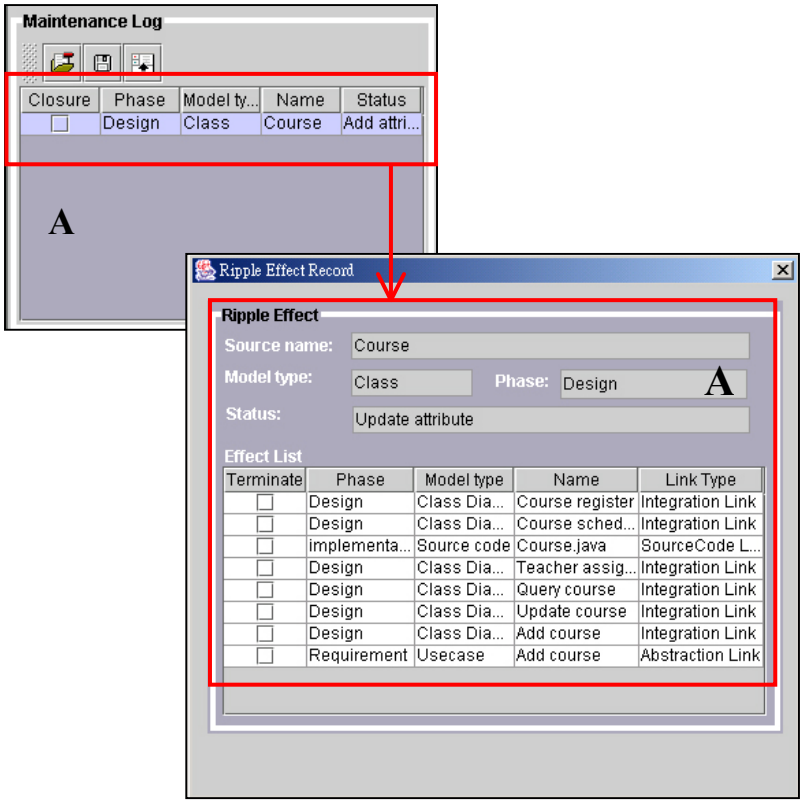

圖 46 漣漪效應資料顯示

使用者接著根據這些資訊逐一檢查並修改受影響的相關模型，在修改的過程中發現 Teacher assignment 的類別圖中，必須為 Teacher 類別額外加一個屬性 -major 以紀錄教師的主修學門，根據 major 以判斷教師適合的課程。如圖 47 所示。圖 48 則是顯示出當 Teacher 類別修改之後，產生第二筆 Maintenance log 及其 Ripple effect 列表。

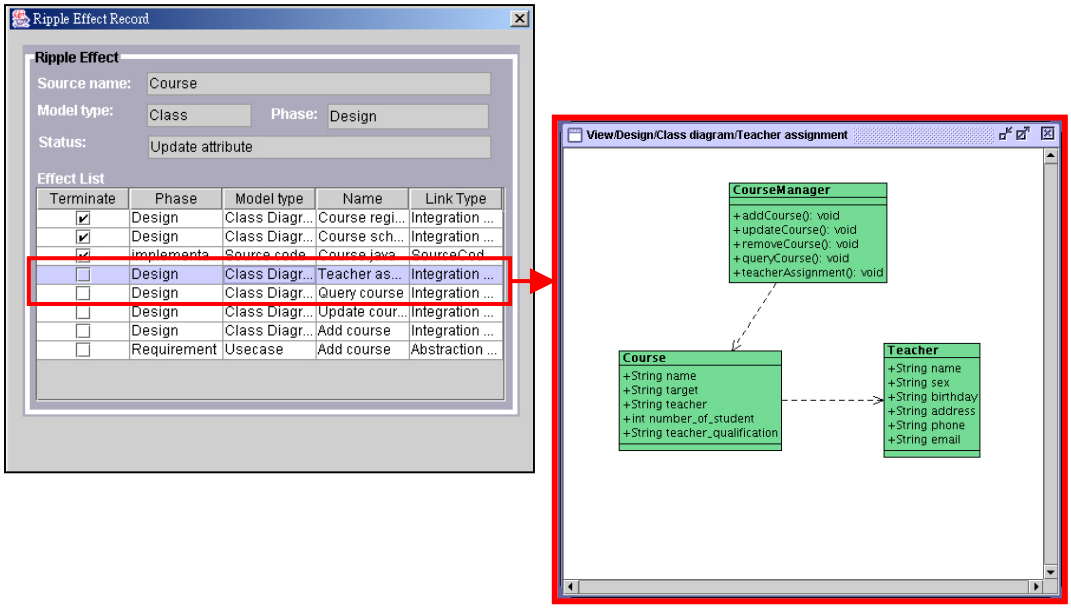

圖 47 檢查及修改各個受影響的軟體模型

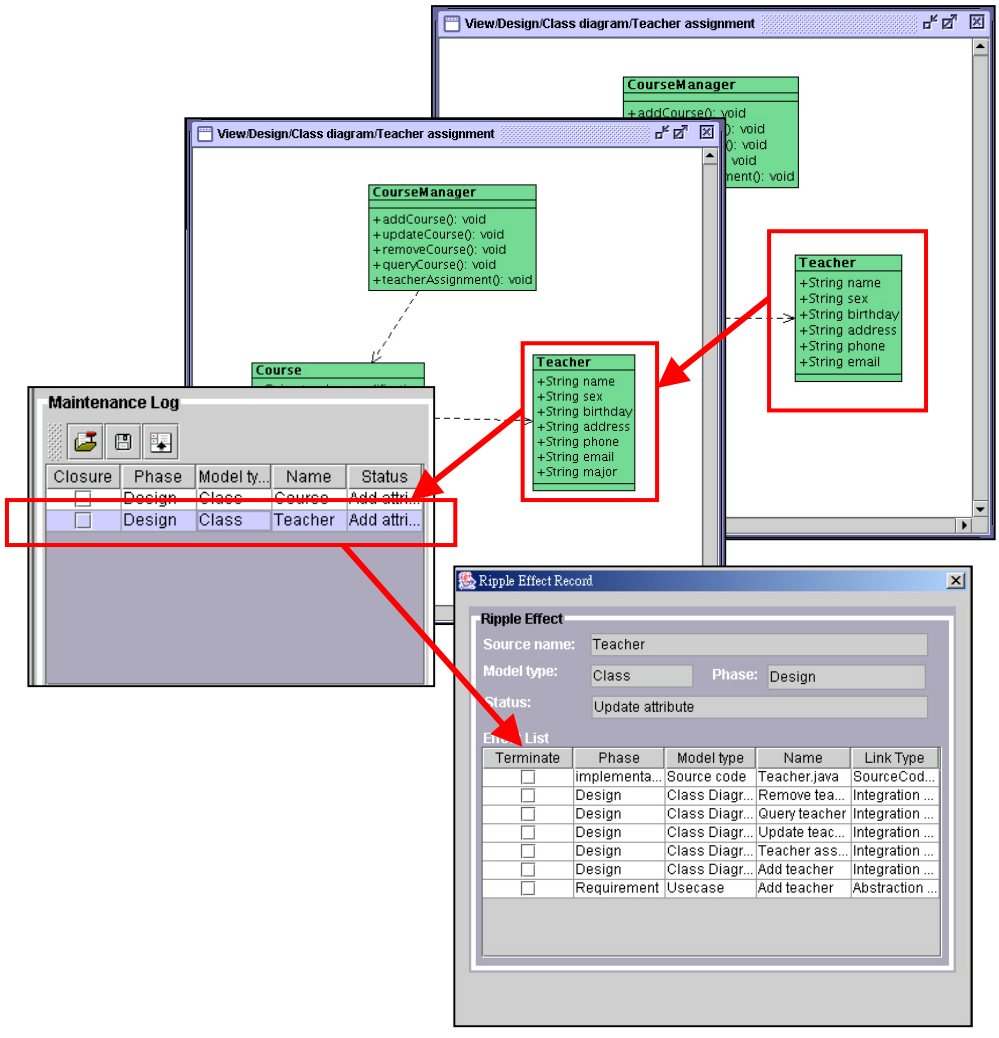

圖 48 第二階段的 ripple effect

在逐一確認及修改完各個模型之後，如圖 49 的 B 所示，各個 ripple effect 都 terminate。而當一個 maintenance log 的 ripple effect 都 terminate 之後，此筆 log 也達成 closure。當每筆 maintenance log 都 closure 之後。一個維護的動作就完成了。如圖 49 的 A 所示。

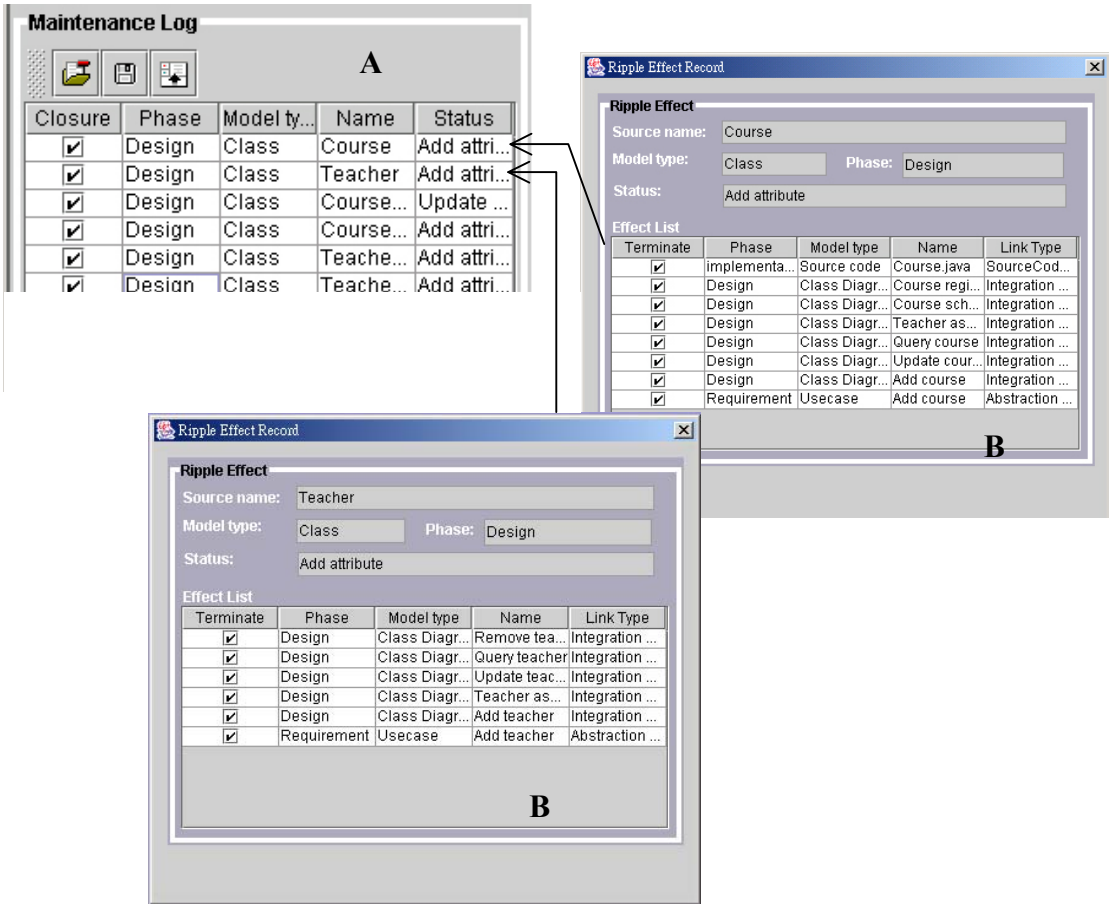

圖 49 終止 Ripple effect

5.4 設計模型的修改-修改類別間的關係

在接下來的內容是討論類別間的結構改變所引發跨標準的 ripple effect。以此範例系統提供的學生選課系統為例。由於原本的系統並無提供選課結果通知的機制。考慮到新的需求，因此軟體維護人員決定修該軟體的設計模型。修改

的軟體模型為 Course register 類別圖，軟體維護人員在該類別圖中加入 Notification 類別，並直接在 Course 與 Notification 類別中建立 dependency 的關係，修改完的結果 maintenance log 中產生了兩筆記錄，如圖 50 所示，其中第一筆 log 是因為新增類別所引起。第二筆 log 是因為建立類別中的結構所引起。因為該類別圖與 Design Pattern 中的 Mediator 具有整合鏈結，因此該類別圖以包含了設計樣版中結構的規範。由於使用者在建立類別間的結構時無意間違反 Mediator 的結構限制，因此產生了 ripple effect，如圖 51 所示。使用者可以查看結構的修改違反了 Mediator 的結構限制，以此範例而言，Mediator 樣版中包含了兩種角色分別是 mediator 與 colleague，其中 colleague 只能與 mediator 有關聯，彼此不能有關聯存在，但是都屬於 colleague 角色的 Course 類別與 Notification 類別卻產生關聯，因此產生了 ripple effect。

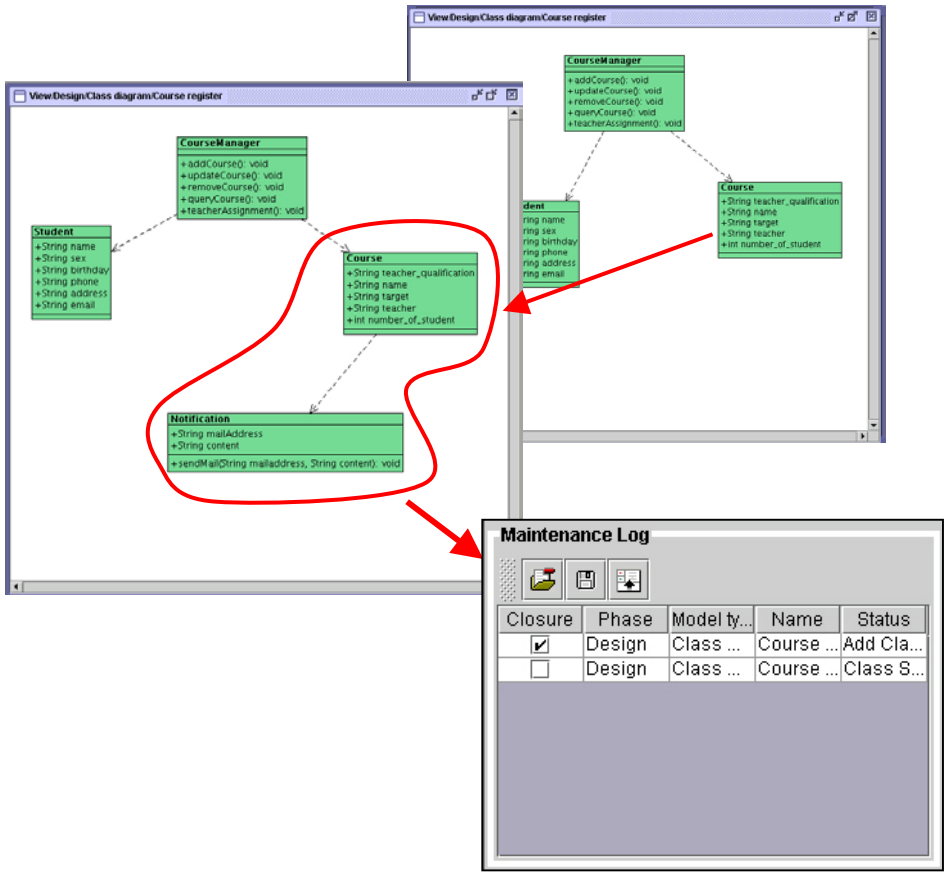

圖 50 修改類別間的關係

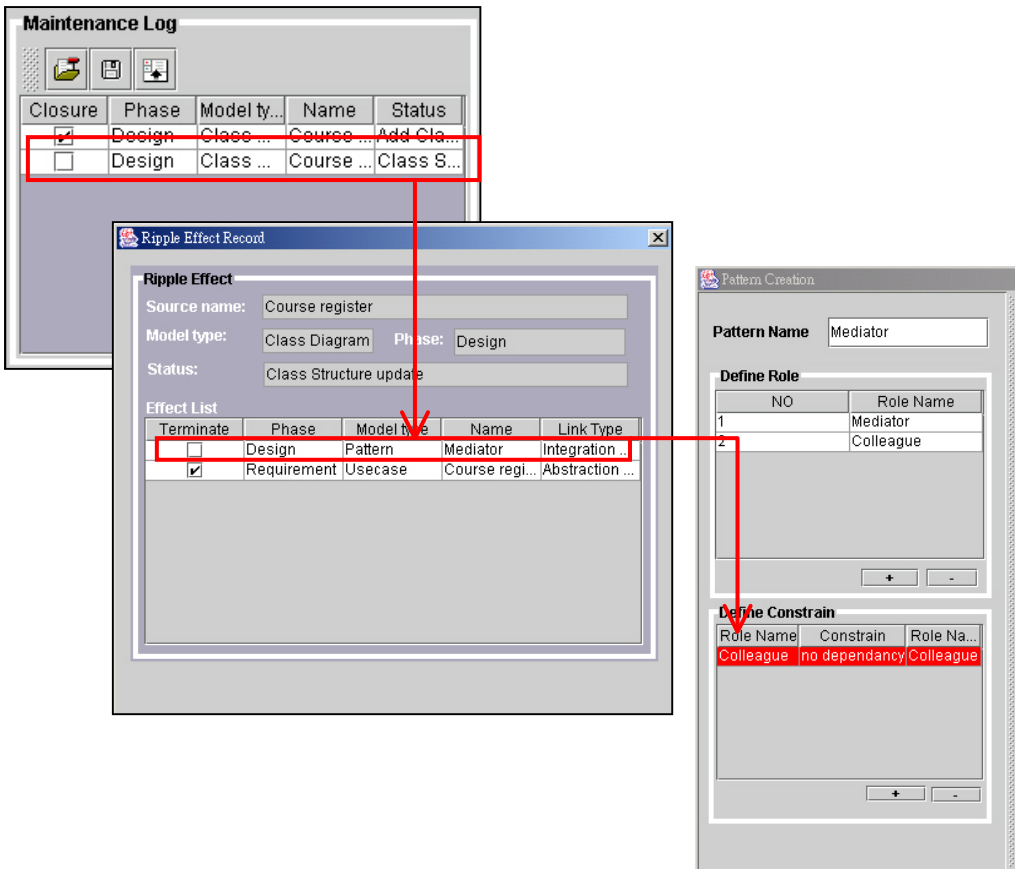

圖 51 類別結構修改所引發的 ripple effect

在察覺此錯誤之後設計人員可以更正原本錯誤的設計，如圖 52 所示。同時確認修改完後 maintenance log 都 closure 為止。

由此可知，本系統工具可以達成跨軟體標準的模型整合。以此例而言，本系統整合了 UML 與 Design Pattern。在 UML 的軟體模型上作修改可以影響到 Design Pattern 的軟體模型。

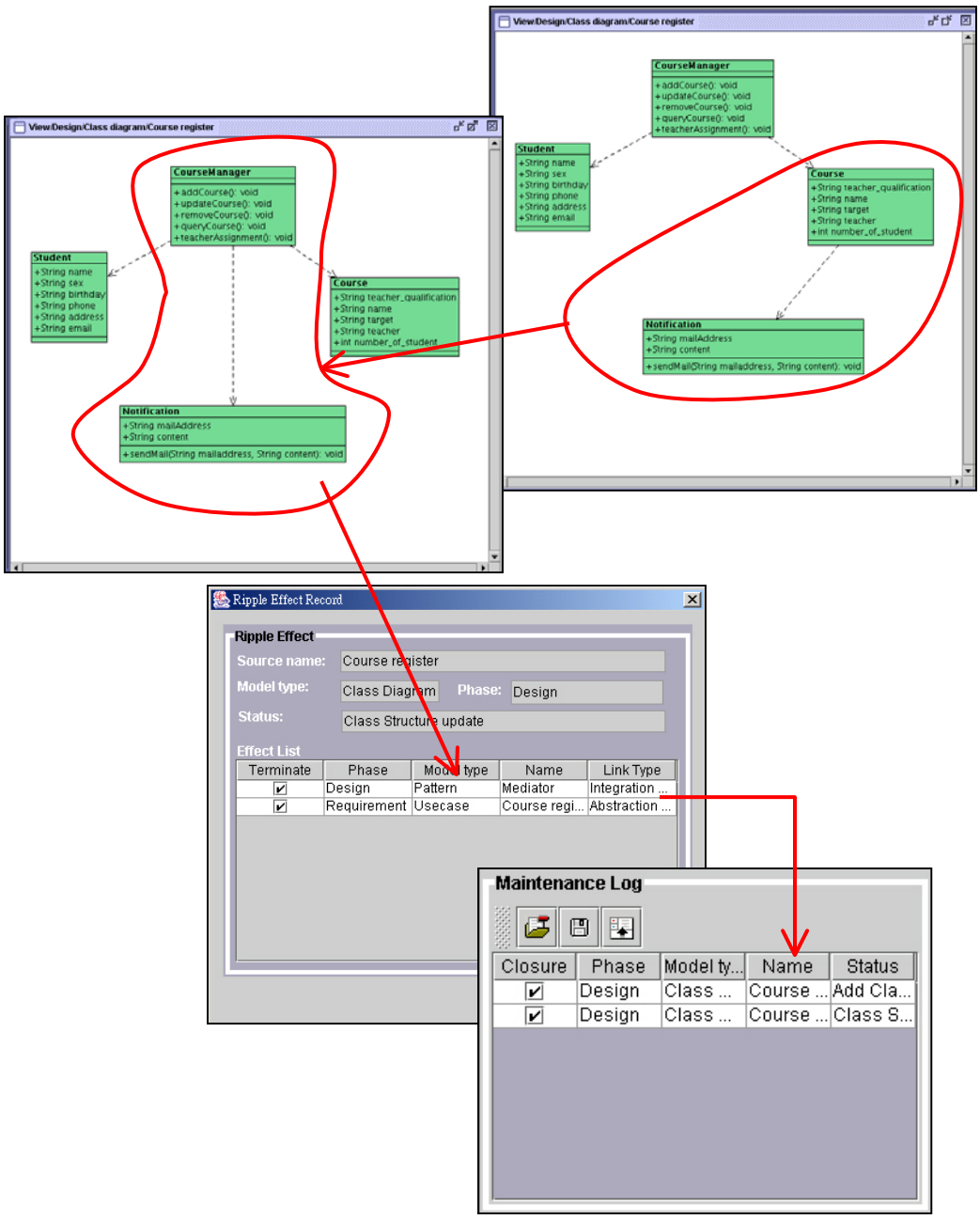

圖 52 更正類別結構並終止 ripple effect

5.5 實作模型的修改-修改程式碼

在介紹完類別內屬性及類別間結構的修改之後，接下來的內容是討論由程式碼直接作修改，並追朔到設計及需求模型，以顯示出跨階段的 ripple effect。本節使用的例子是沿用 5.3 所使用到的例子。在 5.3 中，軟體維護人員直接從類別新增屬性，但是維護人員也可以直接由程式碼修改，加入屬性

teacher qualification 的定義，如圖 53 所示。

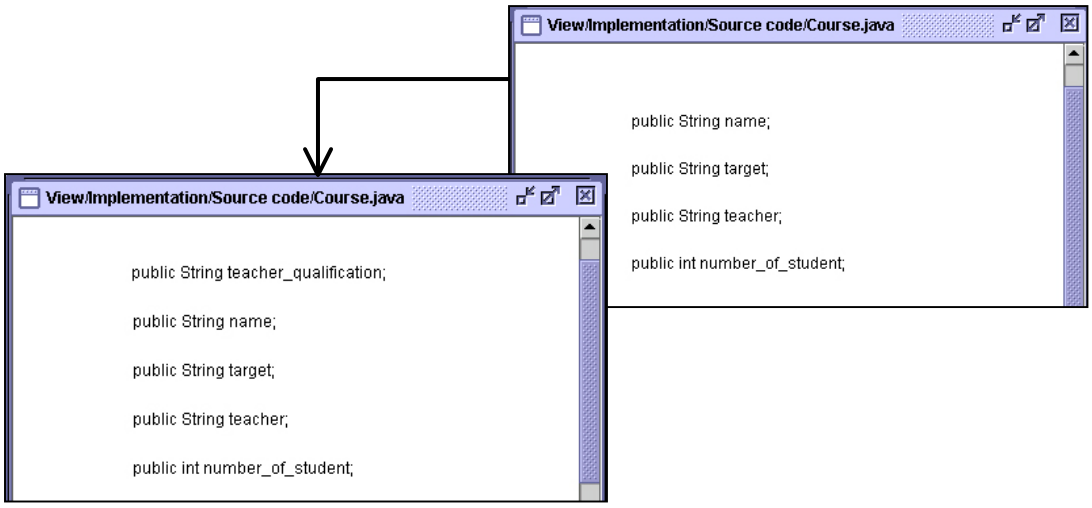

圖 53 維護人員對程式碼直接作修改

修改完成後，maintenance log 上產生了兩筆記錄，第一筆資料記錄了程式碼的修改所影響到的模型，以此範例而言，針對 Course.java 修改之後，設計模型的 Course 類別直接受到影響。由於系統會直接替使用者作程式碼與對應類別的資料同步，因此該 maintenance log 直接呈現 closure 的狀態，如圖 54 所示。

第二筆資料記錄了當 Course 類別新增屬性後所產生的 ripple effect，如圖 54 所示。除了確認相關的設計模型以外，抽象階層的需求模型也必須同步確認。如圖 55 所示雖然程式碼 Course.java 與類別 Course 都存在 teacher_qualification 的屬性，但需求模型的使用案例卻不存在 teacher_qualification 的定義。因此軟體維護人員接著增加需求模型的資料定義如圖 56 所示。

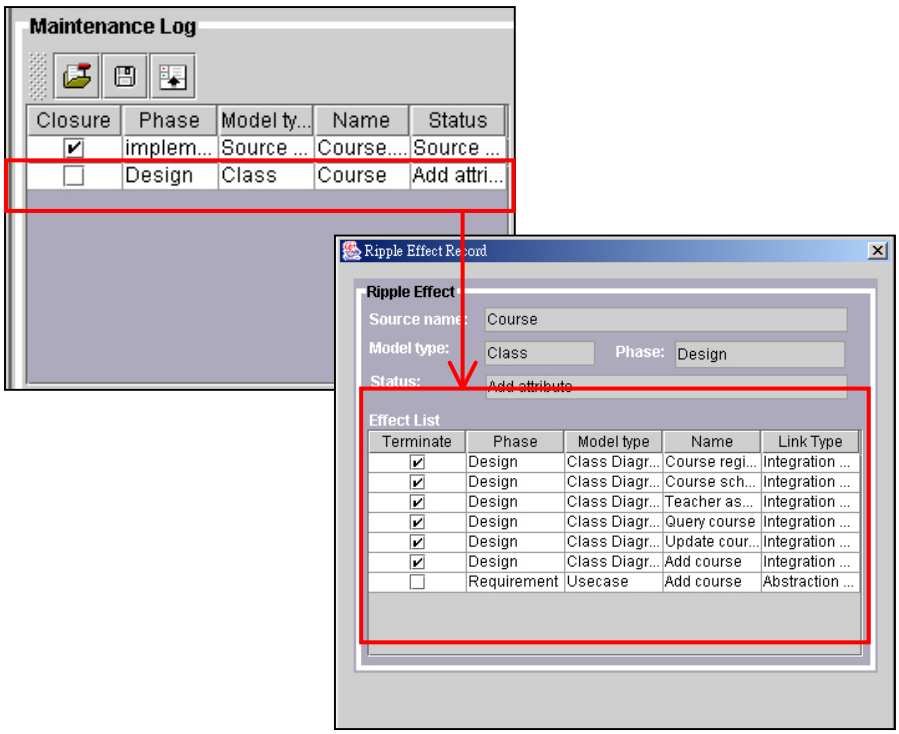

圖 54 修改程式碼產生的 ripple effect

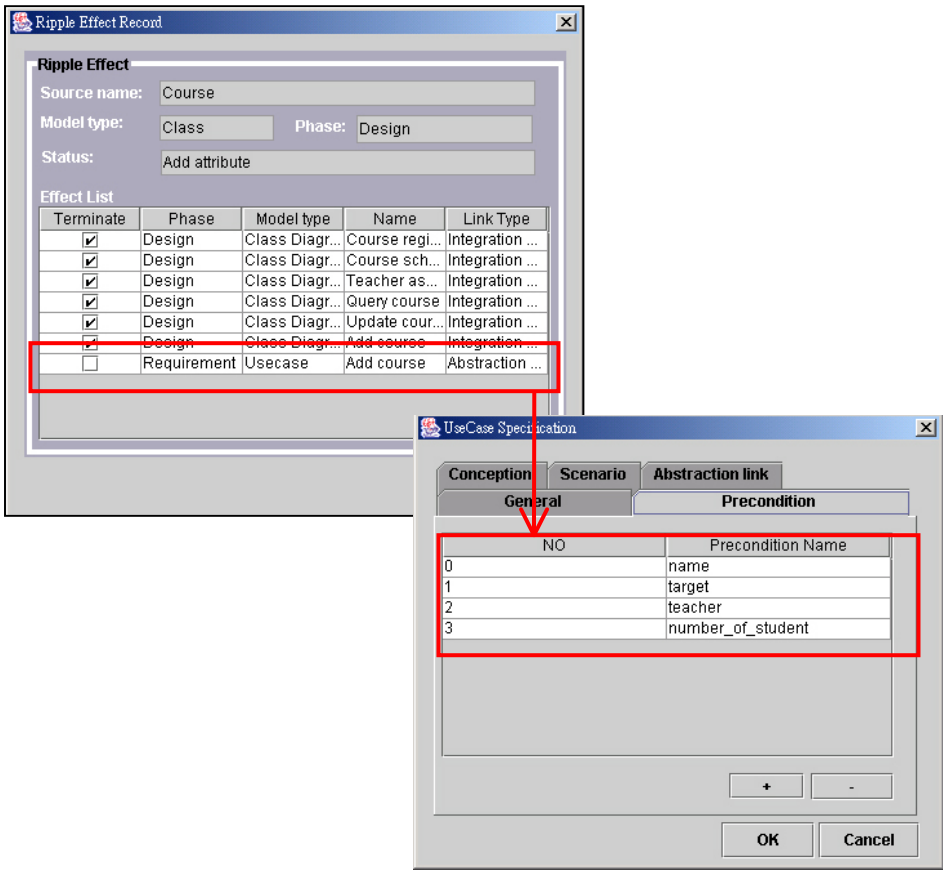

圖 55 由設計模型追朔到需求模型

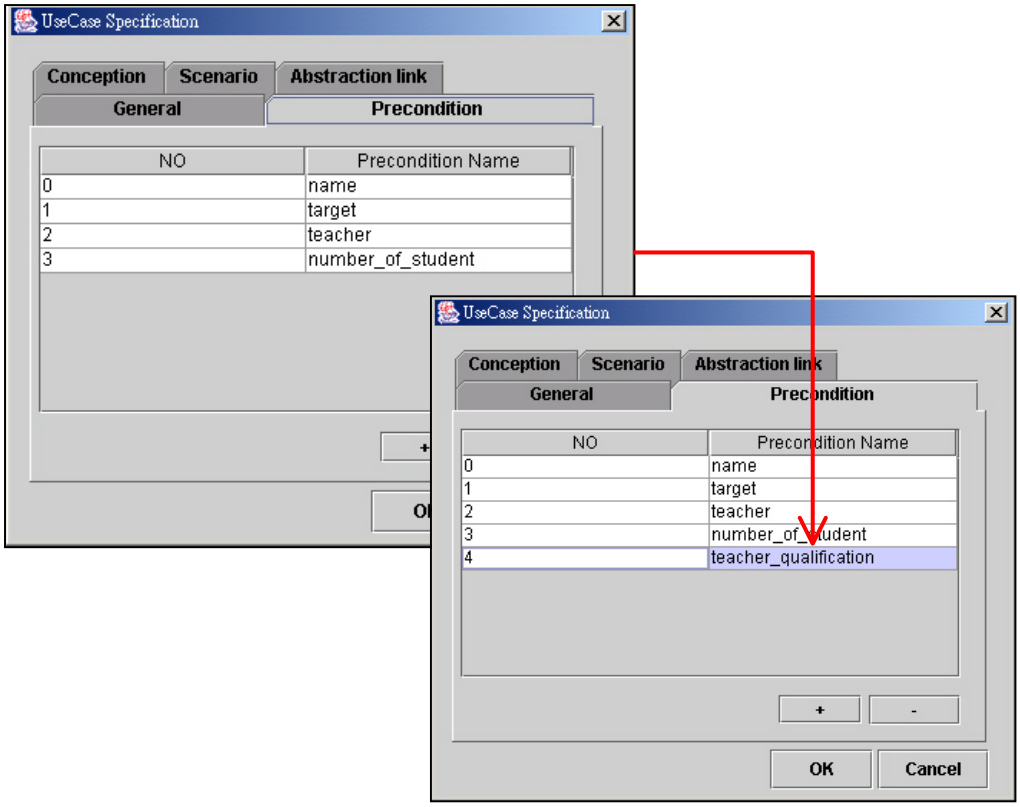

圖 56 增加需求模型的資料定義

由此範例可知，透過本系統的機制可以偵測到跨階段(phase)的 ripple effect，以確認需求分析、軟體設計及實作模型的同步性。

第六章　結論與未來工作

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。

XML-based unified model(XUM)[1]的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。本論文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。

在可驗證性方面，XUM 裡整合了 UML 與設計樣版，由於設計樣版是一種規範性的軟體設計架構，因此當 UML 整合了設計樣版之後，系統設計人員就不能夠在任意的使用 UML 建置設計模型，而必須遵循於設計樣版的規範。因此本系統提供了驗證的機制。在使用者修改設計模型的過程中驗證使用者是否違反了設計樣版的限制。

在未來的發展方面，XUM 的研究及工具軟體的延伸性可以朝以下幾的方向延伸:

- **軟體再使用**

  在軟體開發的過程中導入軟體再使用的特性將可以提昇軟體生產的效率，軟體再使用包含了程式碼的軟體元件再使用與設計架構的再使用。但是能支援各個階段的在使用卻相當少，原因不外乎軟體各個

階段整合上的問題。由於 XUM 以整合各種階段各種標準模型為訴求，因此在軟體再使用的方向將值得深入研究。

- **設計樣版的正規化**

  在本論文中已將幾個慣用的設計樣版導入 XUM，並以工具軟體實作出設計樣版驗證的機制。但是面對更多類型的設計樣版又顯然不足，因此在未來研究方向之一將針對設計樣版做正規化的描述，並導入 XUM 中，而工具軟體也將隨之作功能的擴充。

- **以 XML 為基礎的程式碼分析**

  一般而言，程式碼是一般的文字格式，並無一種結構化的表現方式。但不只是抽象的需求分析模型或設計模型可以利用 XML 表示成結構化的格式，透過程式碼中的註解的描述一樣可以將程式碼表現成結構化的格式並導入 XUM，以建立與需求分析模型或設計模型的串聯。在達成這個目標後，對於程式也就可以用 DOM 的方式做解析的動作。因此以 XML 為基礎的程式碼分析也是研究方向的一部份。

- **軟體系統相關文件的整合與串聯**

  一個軟體系統在發展階段固然有相當多的文件，包含在測試甚至產品包裝時也有相當多的文件，而這些文件將對於此軟體系統而言只是由不同的角度切入，實際上這些文件內的資訊往往是環環相扣的，因此 XUM 的理論發展與工具軟體實作也將會朝向此方向作深入的研究。

參考文獻

1. Chu, W.C**,** Chang C.H., Lu C.W.,Jiau H.C., Chung Y. C., and B. Qiao, "Enhancing software maintainability by unifying standards," To appear in Advances in Software Maintenance Management: Technologies and Solutions, Hershey PA: Idea Group Publishing.
2. Anne, C.L. (1999). XML seen as integral to application integration. *IT Professional*, 2(5), 12-16.
3. Atlee, J.M., & Gannon, J. (1993, January). State-based model checking of event-driven system requirements. *IEEE Transition on Software Engineering*, 19(1), 24-40.
4. Bennett, K. H. (1993). An overview of maintenance and reverse engineering, *The REDO Compendium*, John Wiley & Sons, Inc., Chichester.
5. Booch, G. (1991). *Object-oriented design with applications*. Redwood City, Calif.: Benjamin/Cummings Pub. Co.
6. Booch, G. (1994). *Object-oriented analysis and design with applications* 2nd ed. Redwood City, Calif. : Benjamin/Cummings Pub. Co., 3-25.
7. Bourdeau, R.H., & Cheng, B.H.C. (1995). A formal semantics for object model diagrams. *IEEE Transitions on Software Engineering*, 21(10), 799-821.
8. Chen, D.J., & Chen, T. K. (1994, May). An experimental study of using reusable software design frameworks to achieve software reuse. *Journal of Object-Oriented Programming*, 7(2), 56-67.
9. Cheng, B.H.C., Campbell, L.A., & Wang E.Y. (2000). Enabling automated analysis through the formalization of object-oriented modeling diagrams. in the *Proceedings International Conference on Dependable Systems and Networks 2000 (DSN 2000)*, IEEE, 305-314.
10. Chu, W.C., Lu, C.W, Yang, H., & He, X. (2000, December). A formal approach to component retrieval and integration. *Journal of Software Maintenance*, 12(6), 325-342.
11. Chu, W.C., Lu, C.W., Chang, C.H., & Chung, Y.C. (2001). Pattern based software re-engineering. Handbook of Software Engineering and Knowledge Engineering, Vol. 1, Skokie, IL.: Knowledge Systems Institute.
12. Connolly, D. (2001). *The extensible markup language (XML)*. The World Wide Web Consortium. Retrieved August 21, 2001 from http://www.w3.org/XML
13. Deitel, H., Deitel, P., Nieto, T., Lin, T., & Sadhu, P. (2001). *XML how to program*. Upper Saddle River, NJ : Prentice Hall.
14. Do-Hyoung, K., & Kiwon, C. (1996). A method of checking errors and consistency in the process of object-oriented analysis. in *Proceedings of the 3rd Asia-Pacific Software Engineering Conference (APSEC '96)*, IEEE, 208-216.
15. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Reading, MA.: Addison-Wesley.
16. Gunter, C.A., Gunter, E.L., Jackson, M., & Zave, P. (2000, May/June). *A reference model for requirements and specifications*. IEEE Software, 17(3), 37-43.
17. Hartrum, T.C., & Bailor, P. D. (1994). Teaching formal extensions of informal-based object-oriented analysis methodologies. In the *Proceedings of Computer Science Education*, 389-409.
18. Hayes, F., & Coleman, D. (1991, October). Coherent models for object-oriented analysis. in *Proceedings on ACM OOPSLA'91*, ACM, 171-183.
19. Holland, I.M. (1993). *The design and representation of object-oriented*

components. PhD thesis, Northeastern University. Retrieved March 20, 1996 from http://www.ccs.neu.edu/home/lieber/theses-index.html
20. Holzmann, G.J. (1997, May). The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5), 279-295.
21. IEEE (1983). ANSI, *Standard IEEE Standard Glossary of Software Engineering Terminology*, 729.
22. Jia, X. (1997). A pragmatic approach to formalizing object-oriented modeling and development. in *Proceedings of the COMPSAC '97 - 21st International Computer Software and Applications Conference*, IEEE, 240-245.
23. Jia. X. (1998, August). *ZTC: a type checker for z notation, user's guide,* Version 2.03. Retrieved August 12, 1998 from http://venus.cs.depaul.edu /fm/ztc.html.
24. Jia. X. (1998, July). *A tutorial of ZANS — a Z animation system*. Retrieved Feb. 25, 1998 from http://venus.cs.depaul.edu/fm/zans.html
25. Jia. X., & Skevoulis, S. (1998, August). *VENUS: a code generation tool*, *user guide,* Version 0.1, Retrieved August 25, 1998 from http://venus.cs.depaul.edu/fm/venus.html
26. Johnson, R.E., & Foote, B. (1988, June/July). Designing reusable class. *Journal of Object-Oriented Programming*, 1(2), 22-35.
27. Koskimies, K., Systä, T., & Tuomi, J. (1998, January-February). Automated support for modeling oo software. *IEEE Software*, 15(1), 87- 94.
28. Lano, K., & Malik, N. (1997). Reengineering legacy applications using design patterns. in the *Proceedings of the 8th International Workshop on Software Technology and Engineering Practice*, IEEE, 326-338.
29. Lehman, M. M. (1980). Programs, life cycles, and laws of software evolution, *Proceedings IEEE*, 68(9), 1060-1076.
30. Meyer, B. (1990, September). Tools for the new culture: lessons from the design the eiffel libraries. *Communications of the ACM*, 33(9), 68-88.
31. Moreira, A.M.D., & Clark, R.G. (1996). Adding rigour to object-oriented analysis. *Software Engineering Journal*, 11(5), 270-280.
32. Moser, S., & Nierstrasz, O. (1996). The effect of object-oriented frameworks on developer productivity. *IEEE Computer*, 29(9), 45-51.
33. Murphy, G.C., Notkin, D., & Sullivan, K.J. (2001). Software reflexion models: bridging the gap between design and implementation. *IEEE Transitions on Software Engineering*, 27(4), 364-380.
34. 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
35. Ossher, H., Kaplan, M., Harrison, W., Katz, A., & V. Kruskal. (1995, October). Subject-oriented composition rules. in the *Proceedings of Object-Oriented Programming Systems, Languages and Applications Conference, special issue of SIGPLAN Notices*, ACM, 235-250.
36. Paul, S., & Prakash, A. (1994, June). Framework for source code search using program patterns. *IEEE Transactions on Software Engineering*, 22(6), 463-475.
37. Rine, D. C. (1997). Supporting reuse with object technology. *IEEE Computer*, 30(10), 43-45.
38. Shroff, M., & France, R. B. (1997, August). Towards a formalization of UML class structures in Z. in the *Proceedings Twenty-First Annual International Computer Software and Applications Conference (COMPSAC'97)*, IEEE, 646-651.
39. Smith, R., Szyperski, C., Meyer, B., & Pour, G. (2000). Component-based

development? refining the blueprint. in the *Proceedings of Technology of Object-Oriented Languages and Systems (TOOLS 34'00)*, IEEE, 563-566.
40. Sommerville, I. (1996). *Software Engineering*, 5th edition. Reading, MA.: Addison-Wesley, 4-19.
41. Spivey, J.M. (1992). *The Z notation*, second ed. Upper Saddle River, NJ : Prentice Hall.
42. Suzuki, J., & Yamamoto, Y. (1998, June). Making UML models exchangeable over the internet with XML: UXF approach. In the *Proceedings on UML'98: Beyond the Notation - International Workshop*, 65-74.
43. Szyperski, C., & Pfister, C. (1997). Workshop on component-oriented programming, Summary. In *M. Muhlhauser (Ed.) Special Issues in Object-Oriented Programming –ECOOP96 Workshop Reader*. dpunkt-Verlag, Heidelberg.
44. Wang, E.Y., & Cheng, B.H.C. (1998, June). A rigorous object-oriented design process. in the *Proceedings of International Conference on Software Process*, Naperville, Illinois. Retrieved from http://www.cse.msu.edu/~chengb/pubs.html
45. Wirfs-Brock, R. J., & Johnson, R. E. (1990). Surveying Current Research in Object-Oriented Design. *Communications of the ACM*, 33(9),105-124.
46. Wong, K., Tilley, S.R., MuÈller, H.A., & Storey, M.D. (1995, January). Structural redocumentation: a case study. *IEEE Software*, 12(1), 46-54.
47. Xiao, C. (1994). *Adaptive Software: Automatic Navigation through Partially Specified Data Structures*. PhD thesis, Northeastern University, Retrieved May 7, 2001, from http://www.ccs.neu.edu/home/lorenz/center/ index.html
48. Yau, S.S., & Dong, N. (2000). Integration in component-based software development using design patterns. in the *Proceedings on The 24th Annual International Computer Software and Applications Conference (COMPSAC 2000)*, 369-374.
49. Radmila Juric ,Jasna Kuljis . Building an Evaluation Instrument for 00 CASE Tool Assessment for Unified Modelling Language Support .Proceedings of the Thirty-second Annual Hawaii International Conference on System Science. 1999,pp 1-10
50. Christian Ernst ,Anne Lapujade, Franck Ravat ,Gilles Zurfluh. A CASE Tool To Design Distributed Object Oriented Databases. Proceedings of the Asia Pacific Software Engineering Conference. 1997.
51. Timothy K. Shih, Yih-Jia Tsai, Jason C. Hung, Ding-Rong Jiang. A CASE Tool Supports the Software Life Cycle of Participator Dependent Multimedia Presentations 1998.
52. Taegyun Kim , Nacer Boudjlida. An Experience Report Related to Restructuring OODesigner : A CASE Tool for OMT . Proceedings of the Asia Pacific Software Engineering Conference., 1998 .
53. M. L. Jaccheri and R. Conradi. Techniques for process model evolution in EPOS. IEEE Transactions on Software Engineering,pages 1145-1156,Dec.1993.
54. G.E. Kaiser, P.H. Feiler, and S.S. Popovich. Intelligent Assistant for Software Development and Maintenance. IEEE software ,5(3):40-49,May 1988.
55. Cregut. X , Coulette. B. Filling the gap between CASE tools and PSEEs , Proceedings of the 1997 Workshop on Engineering of Computer-Based Systems (ECBS ‘97), 1997 IEEE.
56. Les Miller, Sree Nila .1998. Proceedings of the 31st Hawaii International Conference on System Sciences (HICSS'98), Data Warehouse Modeler: A CASE Tool for Warehouse Design .

57.Timothy K. Shih, Yih-Jia Tsai, Jason C. Hung, and Ding-Rong Jiang. 1998.Proceedings of the IEEE International Conference on Multimedia Computing and Systems.A CASE Tool Supports the Software Life Cycle of Participator Dependent Multimedia Presentations.

附錄 A

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.2 U (http://www.xmlspy.com) by chchang (hit) -->
<!--W3C Schema generated by XML Spy v4.2 U (http://www.xmlspy.com)-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
   <xs:element name="abstraction_link">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="abstraction_link_id"/>
            <xs:element ref="abstraction_link_from_id"/>
            <xs:element ref="abstraction_link_to_id"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="abstraction_link_from_id" type="xs:string"/>
   <xs:element name="abstraction_link_id" type="xs:string"/>
   <xs:element name="abstraction_link_to_id" type="xs:string"/>
   <xs:element name="actor">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="actor_id"/>
            <xs:element ref="actor_name"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="actor_id" type="xs:string"/>
   <xs:element name="actor_link">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="actor_link_id"/>
            <xs:element ref="actor_x"/>
            <xs:element ref="actor_y"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="actor_link_id" type="xs:string"/>
   <xs:element name="actor_name" type="xs:string"/>
   <xs:element name="actor_x" type="xs:string"/>
   <xs:element name="actor_y" type="xs:string"/>
   <xs:element name="arg_id" type="xs:string"/>
   <xs:element name="arg_name" type="xs:string"/>
   <xs:element name="arg_type" type="xs:string"/>
   <xs:element name="argument">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="arg_id"/>
            <xs:element ref="arg_name"/>
            <xs:element ref="arg_type"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="att_id" type="xs:string"/>
   <xs:element name="att_limit" type="xs:string"/>
   <xs:element name="att_name" type="xs:string"/>
   <xs:element name="att_type" type="xs:string"/>
   <xs:element name="attribute">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="att_id"/>
            <xs:element ref="att_name"/>
            <xs:element ref="att_type"/>
            <xs:element ref="att_limit"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="ca_from" type="xs:string"/>
   <xs:element name="ca_id" type="xs:string"/>
```

```
<xs:element name="ca_to" type="xs:string"/>
<xs:element name="ca_type" type="xs:string"/>
<xs:element name="cd_id" type="xs:string"/>
<xs:element name="cd_name" type="xs:string"/>
<xs:element name="cil_id" type="xs:string"/>
<xs:element name="class_association">
	<xs:complexType>
		<xs:sequence>
			<xs:element ref="ca_id"/>
			<xs:element ref="ca_type"/>
			<xs:element ref="ca_from"/>
			<xs:element ref="ca_to"/>
		</xs:sequence>
	</xs:complexType>
</xs:element>
<xs:element name="class_diagram">
	<xs:complexType>
		<xs:sequence>
			<xs:element ref="cd_name"/>
			<xs:element ref="cd_id"/>
			<xs:element ref="class_integration_link" minOccurs="0" maxOccurs="unbounded"/>
			<xs:element ref="class_association" minOccurs="0" maxOccurs="unbounded"/>
		</xs:sequence>
	</xs:complexType>
</xs:element>
<xs:element name="class_id" type="xs:string"/>
<xs:element name="class_integration_link">
	<xs:complexType>
		<xs:sequence>
			<xs:element ref="cil_id"/>
			<xs:element ref="class_x"/>
			<xs:element ref="class_y"/>
		</xs:sequence>
	</xs:complexType>
</xs:element>
<xs:element name="class_name" type="xs:string"/>
<xs:element name="class_x" type="xs:string"/>
<xs:element name="class_y" type="xs:string"/>
<xs:element name="constrain">
	<xs:complexType>
		<xs:sequence>
			<xs:element ref="constrain_id"/>
			<xs:element ref="role_from"/>
			<xs:element ref="role_to"/>
			<xs:element ref="constrain_type"/>
		</xs:sequence>
	</xs:complexType>
</xs:element>
<xs:element name="constrain_id" type="xs:string"/>
<xs:element name="constrain_type" type="xs:string"/>
<xs:element name="design">
	<xs:complexType>
		<xs:sequence>
			<xs:element ref="package" minOccurs="0" maxOccurs="unbounded"/>
			<xs:element ref="class_diagram" minOccurs="0" maxOccurs="unbounded"/>
			<xs:element ref="xummclass" minOccurs="0" maxOccurs="unbounded"/>
			<xs:element ref="pattern_lib" minOccurs="0" maxOccurs="unbounded"/>
		</xs:sequence>
	</xs:complexType>
</xs:element>
<xs:element name="implementation">
	<xs:complexType>
		<xs:sequence>
			<xs:element ref="package" minOccurs="0" maxOccurs="unbounded"/>
			<xs:element ref="source_code" minOccurs="0" maxOccurs="unbounded"/>
			<xs:element ref="class" minOccurs="0" maxOccurs="unbounded"/>
		</xs:sequence>
	</xs:complexType>
</xs:element>
<xs:element name="integration_link">
	<xs:complexType>
		<xs:sequence>
			<xs:element ref="integration_link_id"/>
			<xs:element ref="integration_link_from_id"/>
			<xs:element ref="integration_link_to_id"/>
```

```
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="integration_link_from_id" type="xs:string"/>
    <xs:element name="integration_link_id" type="xs:string"/>
    <xs:element name="integration_link_to_id" type="xs:string"/>
    <xs:element name="operation">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="operation_id"/>
                <xs:element ref="operation_name"/>
                <xs:element ref="operation_return_type"/>
                <xs:element ref="operation_limit"/>
                <xs:element ref="argument" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="operation_id" type="xs:string"/>
    <xs:element name="operation_limit" type="xs:string"/>
    <xs:element name="operation_name" type="xs:string"/>
    <xs:element name="operation_return_type" type="xs:string"/>
    <xs:element name="package">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="package_id"/>
                <xs:element ref="package_name"/>
                <xs:element ref="package" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="package_id" type="xs:string"/>
    <xs:element name="package_name" type="xs:string"/>
    <xs:element name="pattern">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="pattern_id"/>
                <xs:element ref="pattern_name"/>
                <xs:element ref="role" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="constrain" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="structure" minOccurs="0"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="pattern_id" type="xs:string"/>
    <xs:element name="pattern_lib">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="pattern_lib_id"/>
                <xs:element ref="pattern_lib_name"/>
                <xs:element ref="pattern" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="pattern_lib_id" type="xs:string"/>
    <xs:element name="pattern_lib_name" type="xs:string"/>
    <xs:element name="pattern_name" type="xs:string"/>
    <xs:element name="project_name" type="xs:string"/>
    <xs:element name="project_path" type="xs:string"/>
    <xs:element name="requirement">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="package" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="actor" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="usecase" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="usecase_diagram" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="role">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="role_id"/>
                <xs:element ref="role_name"/>
                <xs:element ref="role_code_link"/>
            </xs:sequence>
```

```
    </xs:complexType>
</xs:element>
<xs:element name="role_code_link" type="xs:string"/>
<xs:element name="role_from" type="xs:string"/>
<xs:element name="role_id" type="xs:string"/>
<xs:element name="role_name" type="xs:string"/>
<xs:element name="role_to" type="xs:string"/>
<xs:element name="source_code">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="source_code_id"/>
            <xs:element ref="source_code_name"/>
            <xs:element ref="source_code_path"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="source_code_id" type="xs:string"/>
<xs:element name="source_code_name" type="xs:string"/>
<xs:element name="source_code_path" type="xs:string"/>
<xs:element name="sourcecode_link">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="sourcecode_link_id"/>
            <xs:element ref="sourcecode_link_from_id"/>
            <xs:element ref="sourcecode_link_to_id"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="sourcecode_link_from_id" type="xs:string"/>
<xs:element name="sourcecode_link_id" type="xs:string"/>
<xs:element name="sourcecode_link_to_id" type="xs:string"/>
<xs:element name="structure">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="structure_id"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="structure_id" type="xs:string"/>
<xs:element name="unification_link">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="integration_link" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="sourcecode_link" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="abstraction_link" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="usecase">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="usecase_id"/>
            <xs:element ref="usecase_name"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="usecase_diagram">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="usecase_diagram_id"/>
            <xs:element ref="usecase_diagram_name"/>
            <xs:element ref="usecase_link" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="actor_link" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="usecase_relationship" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="usecase_diagram_id" type="xs:string"/>
<xs:element name="usecase_diagram_name" type="xs:string"/>
<xs:element name="usecase_from" type="xs:string"/>
<xs:element name="usecase_id" type="xs:string"/>
<xs:element name="usecase_link">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="usecase_link_id"/>
```

```
                    <xs:element ref="usecase_x"/>
                    <xs:element ref="usecase_y"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="usecase_link_id" type="xs:string"/>
        <xs:element name="usecase_name" type="xs:string"/>
        <xs:element name="usecase_relationship">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="usecase_relationship_id"/>
                    <xs:element ref="usecase_from"/>
                    <xs:element ref="usecase_to"/>
                    <xs:element ref="usecase_relationship_type"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="usecase_relationship_id" type="xs:string"/>
        <xs:element name="usecase_relationship_type" type="xs:string"/>
        <xs:element name="usecase_to" type="xs:string"/>
        <xs:element name="usecase_x" type="xs:string"/>
        <xs:element name="usecase_y" type="xs:string"/>
        <xs:element name="xumm">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="project_name" minOccurs="0"/>
                    <xs:element ref="project_path" minOccurs="0"/>
                    <xs:element ref="requirement"/>
                    <xs:element ref="design"/>
                    <xs:element ref="implementation"/>
                    <xs:element ref="unification_link"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="xummclass">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="class_id"/>
                    <xs:element ref="class_name"/>
                    <xs:element ref="attribute" minOccurs="0" maxOccurs="unbounded"/>
                    <xs:element ref="operation" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="class">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="source_id" type="xs:string"/>
                    <xs:element name="attributes" minOccurs="0" maxOccurs="unbounded">
                        <xs:complexType>
                            <xs:attribute name="access_type">
                                <xs:simpleType>
                                    <xs:restriction base="xs:NMTOKEN">
                                        <xs:enumeration value="private"/>
                                        <xs:enumeration value="protect"/>
                                        <xs:enumeration value="public"/>
                                    </xs:restriction>
                                </xs:simpleType>
                            </xs:attribute>
                            <xs:attribute name="data_type" type="xs:string"/>
                            <xs:attribute name="initialize" use="optional"/>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="methods" minOccurs="0" maxOccurs="unbounded">
                        <xs:complexType>
                            <xs:choice>
                                <xs:element name="parameter" minOccurs="0" maxOccurs="unbounded">
                                    <xs:complexType>
                                        <xs:attribute name="name" type="xs:string"/>
                                        <xs:attribute name="data_type" type="xs:string"/>
                                    </xs:complexType>
                                </xs:element>
                                <xs:element ref="statement" minOccurs="0" maxOccurs="unbounded"/>
                            </xs:choice>
                            <xs:attribute name="name" type="xs:string"/>
```

```
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="statement">
            <xs:complexType>
                <xs:choice>
                    <xs:element name="assign" minOccurs="0" maxOccurs="unbounded">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element name="variable"/>
                                <xs:element name="expression">
                                    <xs:complexType/>
                                </xs:element>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="control" minOccurs="0" maxOccurs="unbounded">
                        <xs:complexType>
                            <xs:choice>
                                <xs:element name="chose">
                                    <xs:complexType>
                                        <xs:sequence>
                                            <xs:element name="condition"/>
                                            <xs:element name="then">
                                                <xs:complexType>
                                                    <xs:sequence>
                                                        <xs:element ref="statement"/>
                                                    </xs:sequence>
                                                </xs:complexType>
                                            </xs:element>
                                            <xs:element name="else" minOccurs="0">
                                                <xs:complexType>
                                                    <xs:sequence>
                                                        <xs:element ref="statement"/>
                                                    </xs:sequence>
                                                </xs:complexType>
                                            </xs:element>
                                        </xs:sequence>
                                    </xs:complexType>
                                </xs:element>
                                <xs:element name="case">
                                    <xs:complexType>
                                        <xs:sequence>
                                            <xs:element name="condition"/>
                                            <xs:element name="case" maxOccurs="unbounded">
                                                <xs:complexType>
                                                    <xs:sequence>
                                                        <xs:element name="value"/>
                                                        <xs:element ref="statement"/>
                                                    </xs:sequence>
                                                </xs:complexType>
                                            </xs:element>
                                            <xs:element name="default">
                                                <xs:complexType>
                                                    <xs:sequence>
                                                        <xs:element ref="statement"/>
                                                    </xs:sequence>
                                                </xs:complexType>
                                            </xs:element>
                                        </xs:sequence>
                                    </xs:complexType>
                                </xs:element>
                                <xs:element name="while">
                                    <xs:complexType>
                                        <xs:sequence>
                                            <xs:element name="condition"/>
                                            <xs:element name="do">
                                                <xs:complexType>
                                                    <xs:sequence>
                                                        <xs:element ref="statement"/>
                                                    </xs:sequence>
                                                </xs:complexType>
                                            </xs:element>
```

```
                                        </xs:sequence>
                                    </xs:complexType>
                                </xs:element>
                                <xs:element name="for">
                                    <xs:complexType>
                                        <xs:sequence>
                                            <xs:element name="initialize"/>
                                            <xs:element name="test"/>
                                            <xs:element name="modify"/>
                                            <xs:element ref="statement"/>
                                        </xs:sequence>
                                    </xs:complexType>
                                </xs:element>
                            </xs:choice>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="comment" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
                </xs:choice>
            </xs:complexType>
        </xs:element>
    </xs:schema>
```