私立東海大學資訊工程與科學系研究所

碩士論文

指導教授：朱正忠 教授

應用 XUM 整合軟體開發流程及變更管理

Applying XUM Approach to Integrate Software Development Workflow and Management

研究生：戴宇駿

中 華 民 國 九十八 年 七 月 二 日

摘要

軟體開發勢必產生及使用不同開發階段的文件，考慮如何有效整理文件資料並透過開發流程文件資訊擷取進行開發，進而維持開發流程文件的可利用性(Availability)與一致性(Consistency)[9]將會影響系統開發成敗與否。系統開發過程模型都有各自的元素、元件定義與關聯性(Association)，各自成為一獨立的體系，彼此歧異度很大，如何在階段間的文件中以正確的方式取得開發資訊已成為現今相當重要的議題。

為了有效管理與利用系統開發流程中各階段開發文件，因此我們建立一個統一的整合模型(Unified Model)來整合原有的各種文件，這個模型有一套統一的邏輯描述、有一套擷取關鍵資訊的機制、並與各原有的開發文件良好相容，並在概念、結構以及表現形式等觀點加以探討。

關鍵詞：開發流程、統一模型、關聯性、一致性、可利用性

Abstract

We must deal with the document with data and models of heterogeneity in the software development. Considering how to effectually manage the document and develop system through completely software development process document information catching to array data and carry on materials to maintain models' availability and consistency. On the System development process, they have their own elements, component definitions and association. Each of them is independent and has difference relationship. How to catch the correct information by the right way through development phase document is now become an important issue.

In order to manage and capitalize developing document effectively in every stage of system development, we need to set up an unified model to combine various kinds of original models, which has a set of unified logic to describe, and a mechanism to catch the pivotal information. Every original inclusive model and probe into such views as the concept, structure and technique of expression.

Keywords : Development Process 、Unified Model 、Association 、Consistency 、Availability

目錄

圖目錄

表目錄

一、序論

1.1 前言

隨著軟體產業(Software Industry)重要性的提昇，軟體工程近幾年受到更多的重視，過去幾年許多產業亦願意投入經費及資源來改善其軟體品質的相關活動，例如導入能力成熟度整合模型(Capability Maturity Model Integration, CMMI)來提昇團隊之服務品質及團隊的成熟度。

軟體工程之相關議題，從軟體生命週期的各個階段衍生出來，包括需求分析、系統設計，到軟體開發、驗證、測試，直到最後的維護[4]。過去幾年，軟體再利用(Reuse)、可擴展標記語言(Extensible Markup Language)[2][23]、設計樣版(Design Pattern)[1]及以元件為基礎的軟體工程(Component-Based Software Engineering)[3][5][12]及物件導向開發(Object-Oriented Development)[1][7][12]等相關技術相繼提出，目的都是希望提高軟體系統開發及維護的效率。

1.2 研究動機

有鑑於軟體在本質上具有高度的彈性，當不同公司或不同團隊採用各別的方法、Models及Data Schema描述方式，便會造成進行軟體開發(Development)、整合(Integration)、再利用(Reuse)時的極大困難[1][13]，於是探討如何提供一個標準的機制與架構方法及整合模型成為目前重要的議題。同時，也將伴隨增進軟體開發時程掌握性、開發風險與開發成本的降

低等等所帶來之附加價值。

軟體的生命週期一般而言包含了需求、設計、實作、測試及維護五個階段，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。然而當我們於階段間使用不同的標準，可能會引發階段間整合溝通的不相容(Incompatibility)[9][19]。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，因此我們提出整合各階段模型，以及模型間溝通的方法。

1.3 研究目的

軟體工業公認有缺陷的開發流程是造成成本浪費及進度落後的主要原因，在開發專案的生命週期裡，最重要的一個部份就是要整合系統各階段之不同的標準，因此本團隊先前提出一個Unified and Integration Model，以XML為基礎整合軟體標準模型，稱之為XML-Based Unified Model Metamodel[11]來描述。將相關的資訊表示成元件(Components)、關聯(Associations)以及統一鏈結(Unification Links)，藉此有效地將不同的標準整合在一起，透過一致的表現方式以及統一鏈結(Unification Links)的串接，有效達到標準之間相關聯文件間資訊的串接與追蹤，其範疇涵蓋軟體元件(Software Components)的辨識(Identification)、取得(Extraction)、表示方法(Representation)、擷取(Retrieval)、整合(Integration)、元件設計(Component Design)及元件開發(Component Implementation)[10]。

1.4 章節安排

本研究內容分為五章，各章節內容的架構如下：

第一章、序論

介紹本研究的緣起、研究動機以及研究目的，讓讀者快速瞭解背景，並作為以下章節說明的基礎。

第二章、背景知識與相關研究

概述本研究根據的理論基礎與研究內容的關聯性及研究中使用的相關技術基本介紹與研究中的研究發展技術使用。

第三章、研究方法

對於本研究中訂定出的軟體開發流程方法，以及流程開發階段文件描述方式作詳細地說明。

第四章、案例研究

將簡單以案例研究說明本研究方法實際導入軟體需求之開發流程產生的過程以及成果。

第五章、結論及未來工作

對本論文的研究方法及案例研究內容作一個總結，並概述未來的研究方向。

二、背景知識及相關研究

2.1 eXtensible Markup Language (XML)

XML[22]是由World Wide Web Consortium(W3C)所制定的標準，是SGML (Standard Generalized Makeup Language)的一部分，一種延伸式的標記語言，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，使用者可定義各種XML Schema或者是根據W3C提供的DTD建構物件實體，完成一份標準的XML Document。[15][16][17]

XML 1.0版規格在1998年2月正式推出，其相關技術在近幾年不斷地制定，目前已逐漸有不少實際運用的例子，且XML Schema經由廣泛的使用已成為正式規格，微軟繼推出MSXML Parser 4.0版之後在2009年初亦發布將於2009~2010年研究發展將推出支援XLink的瀏覽器。

XML寫法上類似HTML，但在功能上可以補足HTML標籤的不足，而擁有更高的擴充性。不過XML不是用來編排內容，而是用來描述資料，他並沒有如同HTML一般的預設標籤，而是使用者自己定義描述資料所需的標籤。[14]

XML相關技術已經逐漸擴充發展，大致如下圖 一[26]所示：

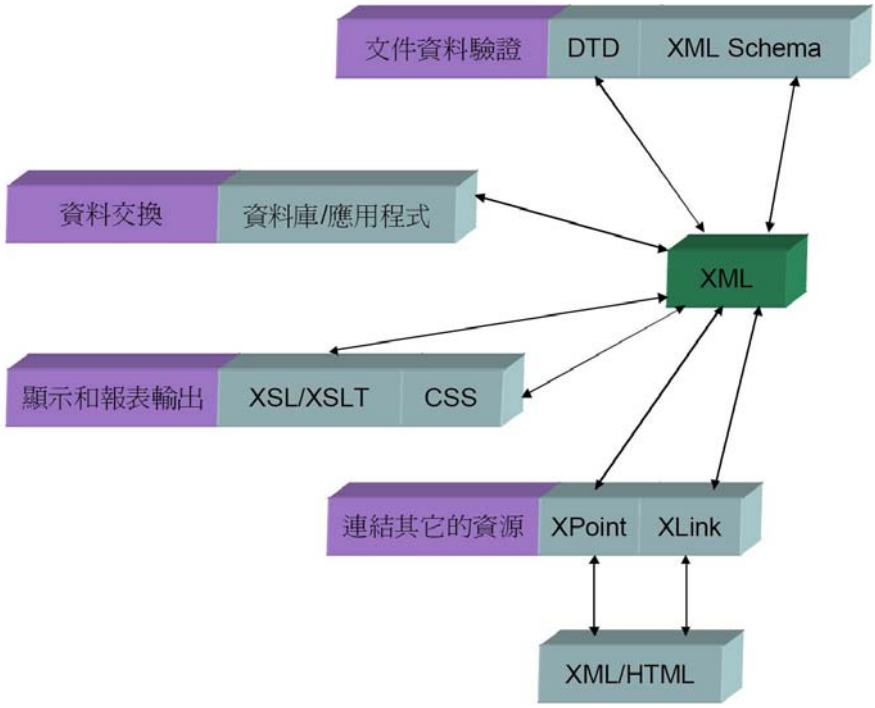

圖 一、XML 相關技術概圖

2.1.1 XML相關技術

(1). Document Type Definition (DTD)

DTD就是SGML的語法檢查，DTD可以幫助XML Parser來解釋XML文件的內容，用來驗證文件的結構及標籤內容是否符合使用者定義的規則[25]。

(2). XML Schema Definition Language (XSD)[16][17]

XML Schema是由軟體廠商，如：Microsoft、Inso、ArborText和DataChannel提出的檢查方式。XML Schema定義語法的方式是使用XML TAG，因此它的架構其實就是一份XML文件，所以在使用上和撰寫一份XML文件相同，不需要像DTD一樣學習另一種語法。

(3). XML連結[14]

有鑒於HTML使用超連結快速連接其他網頁或網路上的資源，XML提出了XPoint和Xlink，提供類似超連結的技術支援，然而目前所訂定的標準只限於連結HTML以及XML文件。

2.2 Unified Modeling Language (UML)

系統開發過程必須產生及參考不同階段的資料與模型，開發者需要考慮如何有效整理資料並進行資料擷取的工作，如此一來才能維持系統開發成果的可利用性(Availability)及與客戶需求的一致性(Consistency)。為了有效管理與利用這些模型，於是建立一個統一的整合模型(Unified Model)來整合原有的各種模型，這個模型有一套統一的邏輯描述、有一套擷取關鍵資訊的機制、並與各原有模型良好相容。對於整合模型的建立，需要考慮的因素有以下幾個重點[3]：

(1). 概念上：具有高度抽象的、可共用的概念，可有效、全面、完整地適應與涵蓋現有的資料模型以及資料範圍；不針對某個特別的應用而設計。

(2). 結構上：應是穩定的、靈活的、可擴展的；存放最詳盡的資料，保證足夠的靈活性，適應複雜的實際用途，在模型發生變化或者新增來源時能易於擴展；核心結構在很長時間內應保持穩定性，便於面對可能不斷產生、不斷變化且無法預先定義的軟體應用環境中產生

的問題。

(3). 表現形式：應有統一的規範，易懂；包括各類命名規範，規則定義，度量方式等。使用統一的邏輯語言進行模型設計，易於使用人員的理解和使用；也有利於人員的溝通。

物件導向的分析方法提出已有一段時間，隨後才由主要三位物件導向大師Booch、Rambaugh以及Jacobson以物件導向方法論所提出的一種物件導向語言(OO Language)[6][12]，並被OMG納入為標準，主要有兩個優點：

(1). UML是撰寫軟體藍圖的標準語言。

(2). UML可以用視覺化的方式訂定、建構軟體為主的系統的產出。[3][5]

UML結合了Booch的OO Method與Rambaugh的OMT與Jacobson的OOSE，發展至今(OMG UML v2.0)已經包含了13種Diagrams分別是：Class、Component、Composite Structure、Deployment、Object、Package、Activity、State Machine、Use Case、Communication、Interaction Overview、Sequence、UML Timing以上各類型的Diagrams，改進了UML 1.X版符號集合有時難以適用於較大的應用程式，加上模型邏輯流程過於複雜容易造成圖形不容易閱讀。

目前UML已成了一種通用的物件導向語言(Object-Oriented Language)，廣為被應用在各類系統的描述及設計上，成為一種溝通設計理念的語言，

除了主導廠商Rational的推廣及擴展之外，其他為數眾多的廠商及研究單位也利用UML 做為表達設計或研究成果的語言。

在軟體工程中，UML已被應用在相關的研究中，譬如：Software Architecture、Framework、Patterns、Software Process以及Workflow。目前支援UML的廠商與團體紛紛推出許多的工具，除了最著名的IBM Rational之ROSE (Rational Object Software Engineering)以外，TogetherSoft與VisualUML也推出了UML的相關工具，Microsoft Visio 2007也支援UML的Diagram繪製模組[3][4]。

2.3 XML-Based Unified Model (XUM)

XUM[11]為以XML做為基礎的軟體標準整合模型，將相關的方法表示成元件(Components)、關聯(Associations)以及整合鏈結(Integration Links)。當系統開發時，將Use Case Diagram轉換為XML文件之後，再行轉換為XML-Based Unified Metamodel (XUMM)如下圖 四，將XML以Model的處理方式模型化儲存，分為View以及Framework，並以透過連結，將Use Case以及XML文件中的TAG原有的關係建立起來。透過XUM方式，將不同的標準整合在一起，讓文件達到一致性，並透過圖 二、圖 三清楚呈現開發過程中的元素以及關係表，有效達到跨標準間關聯資訊的串接與再利用。

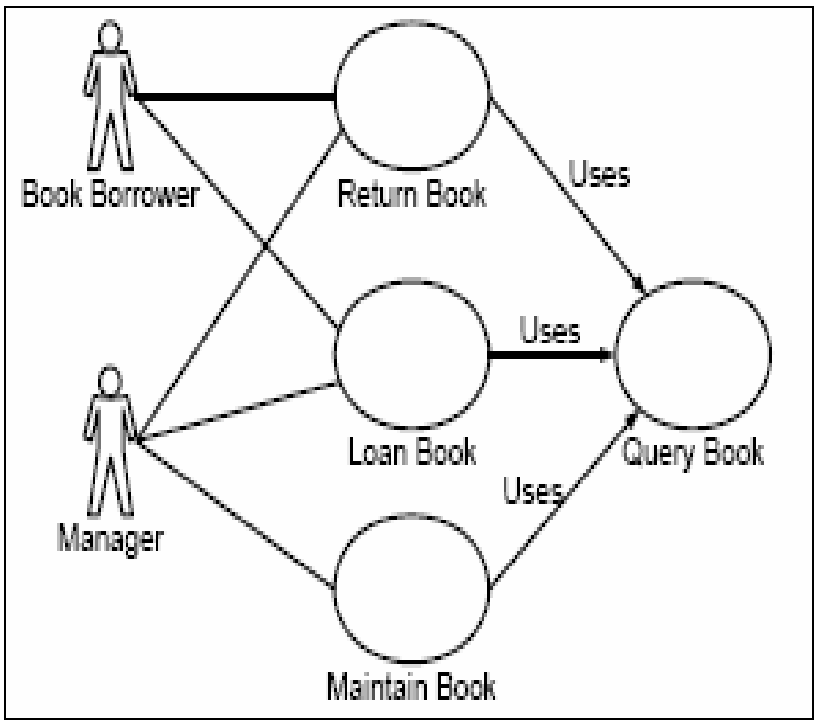

圖 二、The Use Case Diagram of The Library Sub-System

```
<Requirement>
    <UseCase_Daigram>
        <Actor name="Book Borrower"/>
            <Actor name="Manager"/>
                <Usecase name="Loan Book">
                    <Abstraction_link xlink:label="A_Loan_Book" xlink:title=
                    "Use Case of Loan_Book" xlink:from="A_Loan_Book" xlink:to="?
                    "/>
                    <Abstraction_link xlink:from="A_Loan_Book" xlink:to=" ?"/>
                    ……
                </Usecase>
                <Usecase name="Return Book">
                ……
                ……
    </UseCase_Daigram>
</Requirement>
```

圖 三、The XUM Specification of The Use Case Diagram

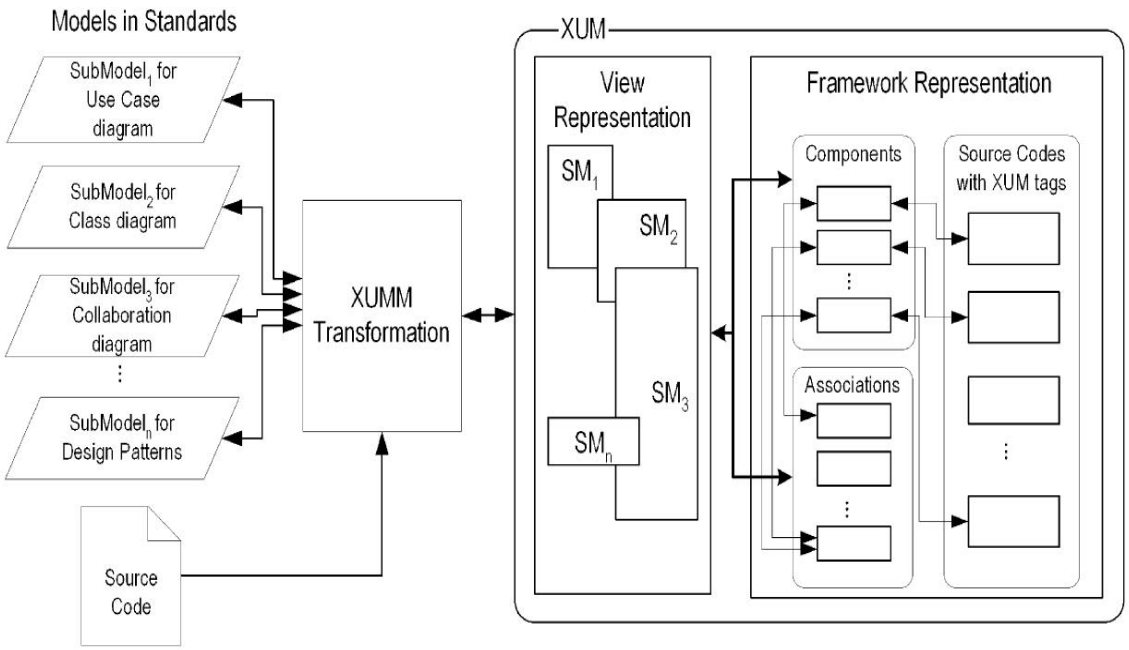

圖 四、The Unification And Integration of Models Into XUM

2.3.1 Unified Model Description

為了建立一個XML-Based Unified Model，需要定義一個以模型為基礎之XML系統文件描述模型，稱之為XML-Based Unified Model Metamodel (XUMM)[11]。它描述出Unified and Integration Model的輪廓(Schema)，同時讓我們可以合併(Consolidate)、協同(Coordinate)在軟體週期各階段中，以不同軟體標準建置的模型資訊。

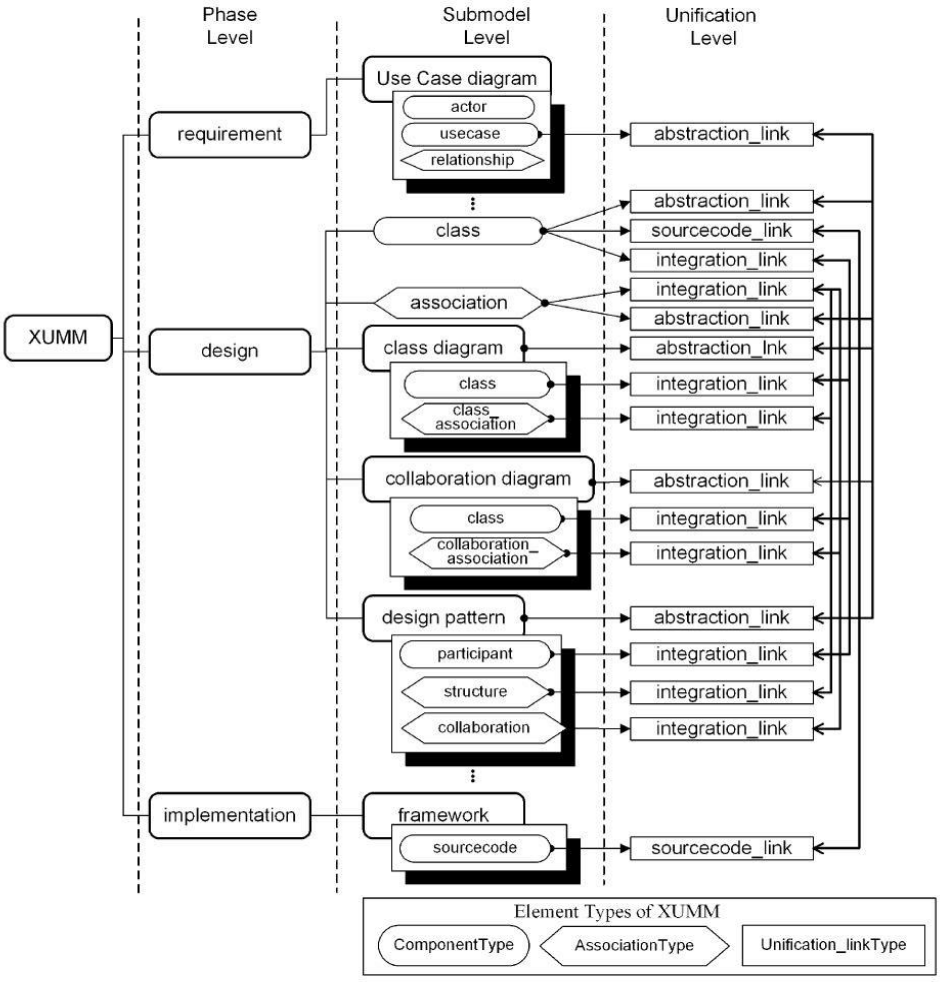

圖 五、The Relationship of Views in XUM

XUMM必須包括資料結構的定義。在此包含物件(Component)、關係(Relation)、與統一關係(Unification Relation)。將軟體開發各階段轉換為XML文件，如上圖 五依照描述模型的結構部份模型化處理儲存XML文件，將各階段的開發文件依照模型化後的類別圖轉換成XML，並將其中關係建立關聯性鏈結，可以合併整合開發週期產出的各式標準開發文件，轉換結果截取出部份內容如下圖 六，在Use Case轉成XML文件時，其中仍保留著原始Use Case當中的物件、作用關係等原始元素，因此可以將Use Case Diagram進行文件轉換，並保留原有文件特性。

```
<xs:element name="usecase_relationship_id" type="xs:string">
<xs:element name="usecase_from" type="xs:string">
<xs:element name="usecase_to type="xs:string">
<xs:element name="usecase_relationship_type" 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>
```

圖 六、使用案例的描述

統一鏈結(Unification Links)的建立是用以整合各階段相關文件的機制。當關聯性鏈結建立的可行性成立，便可藉由XML文件中的關係建立，達到透過XML文件整合系統開發中Requirement、Design以及Implementation等階段，並可提供系統開發階段關聯性追蹤的目的。其中關聯性鏈結包含了抽象鏈結(Abstraction Link)，整合鏈結(Integration Link)與程式碼鏈結(Source Code Link)。如下圖 七以抽象鏈結的建立為例，可以以XML文件來表達抽象鏈結所包含的元素，其中包含了鏈結名稱、鏈結的對象。

```
<xs:element name="abstraction_link_id" type="xs:string">
<xs:element name="abstraction_link_from_id" type="xs:string">
<xs:element name="abstraction_link_to_id" type="xs:string">
<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>
```

圖 七、抽象鏈結的定義

XUM將文件描述模型的結構中需求文件進行分析，在使用者案例(Use Case)中，利用抽象鏈結來描述抽象的軟體需求與軟體設計。在UML中描述需求的使用者案例將應用至設計階段中多個類別圖(Class Diagram)或循序圖(Sequence Diagram)如下圖 八、表 一。依照先前XML文件化的方式，分別將Use Case Component、Component Relation，以及Component Name等需求相關模型資訊[22]轉為XML格式，作為設計階段可用之參考文件。再透過建立Abstraction及Integration Link，將Requirement以及Design兩階段作整合。

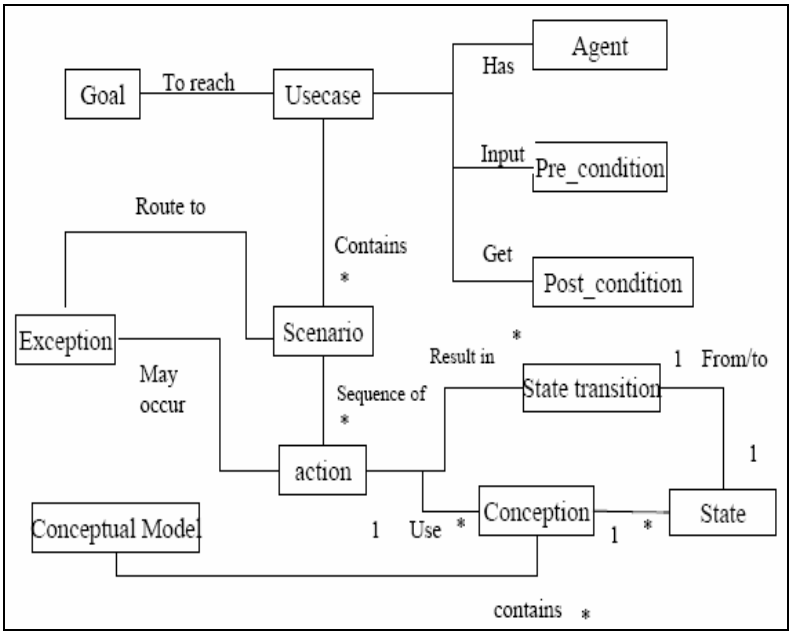

圖 八、需求相關的模型資訊

表 一、需求相關的模型資訊中各元素的描述

Use Case	使用案例，通常描述使用者對於軟體系統的需求。
Agent	一個 Use Case 包含多個 Agent，Agent 代表著 Actor 或是一個子系統。
Goal	一個 Use Case 往往存在一個目標。
Pre-Condition	一個 Use Case 執行前必須滿足的前置條件。
Post-Condition	一個 Use Case 執行後可以得到的結果。
Scenario	一個 Use Case 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(Scenario)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)描述現實需求中多個物件的靜態關係。
Exception	一個 Action 執行時可能會有例外(Exception)的發生。

2.3.2 Component & Association Sharing

上圖表描述完關於XUMM在階段間文件及XUM元素間的轉換對照，後續談到需求的模型資訊與設計階段的類別圖、循序圖、設計樣版以及XUMM的對應關係如下圖。以下圖 九來說明物件共享，在Class D分別屬於UML類別圖中的類別、循序圖中的物件及設計樣版中的角色。因此三者相互間已結合出整合鏈結的關係，在圖 十中，可以看到在Class A與D的關係共享分別使用類別圖中的依賴關係、循序圖中的訊息連結與設計樣版中角色間的結構與通訊關係[21]。

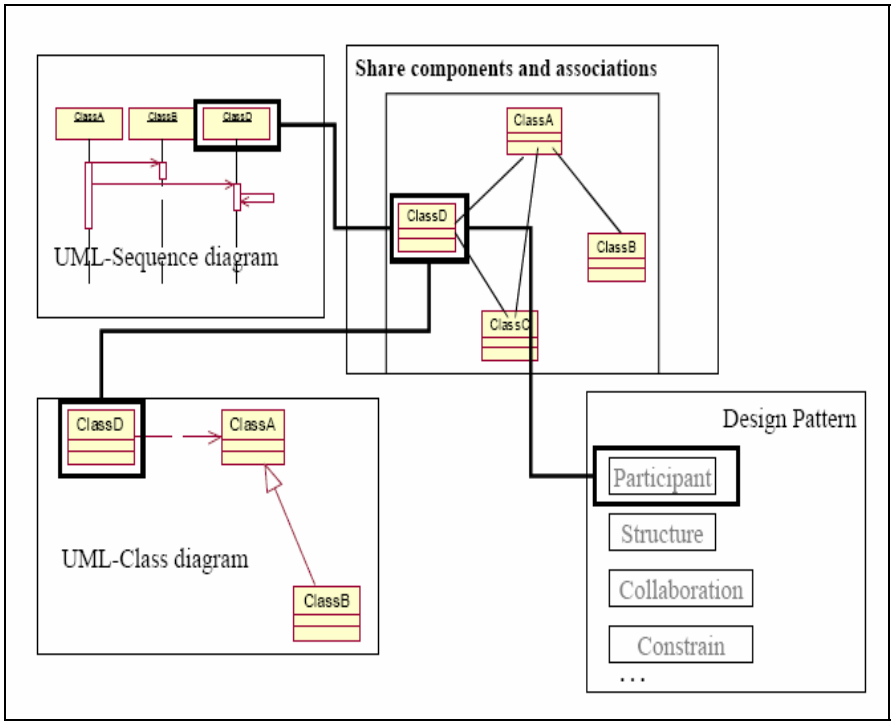

圖 九、整合鏈結中的物件共享

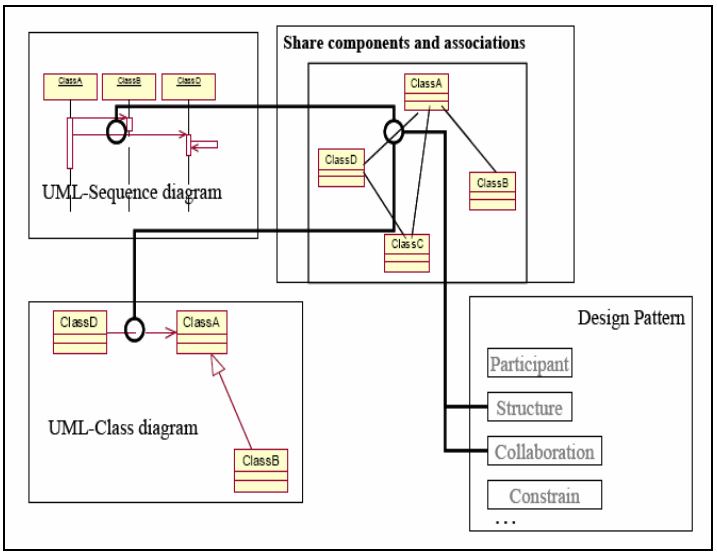

圖 十、整合鏈結中的關係共享

最後對於程式碼鏈結，透過串聯設計的模型資訊中，物件與實作的程式碼，如圖 十一所示。顯示由程式碼擷取出的資訊，從當中可以將Implementation Source Code與先前XML格式的Design Document作連結，建立Source Code Link。可以串聯起物件及關係與Source Code的撰寫彼此之間的轉換結果，透過此過程將Implementation以及Design兩個階段之間作整合。

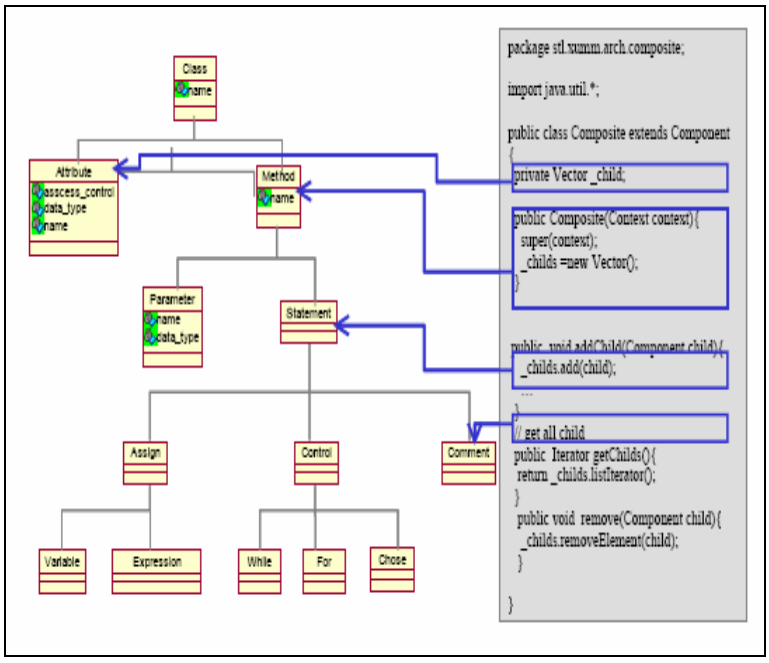

圖 十一、從程式碼中擷取出的資訊

2.4 Software Development Life Cycle (軟體開發生命週期)

軟體模型乃隨著軟體的週期而不斷改變，同時隨著每階段而有不斷的演進**[24]**。不同的軟體工具在不同的階段往往利用不同的軟體標準建立出不同格式的軟體模型，各別的軟體標準都有各別的貢獻，但沒有任何一種標準可以協助於軟體開發裡的各個階段，因此開發者必需在各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，不同的階段間會產生隔閡[4][19]如下圖 十二[11]，於是便要透過轉換，才能確保彼此的協調。

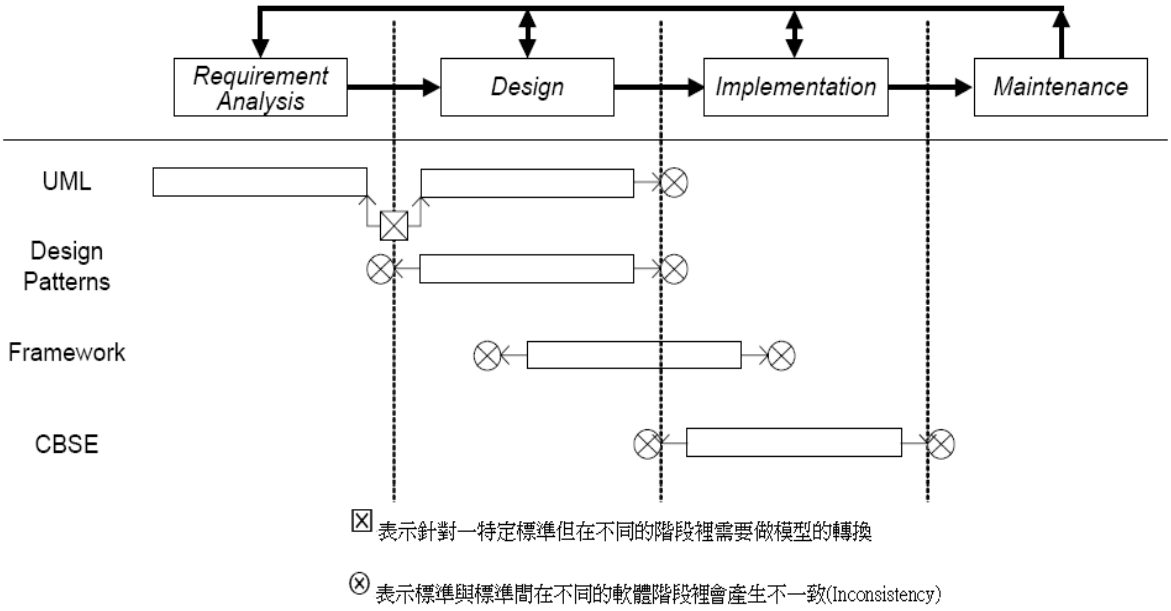

圖 十二、各種標準在不同階段間彼此的關係

在軟體開發週期不同階段間的文件以往必須透過人工表達階段之間銜接的轉換方式，必須增加開發成本維持轉換文件內容的一致性及正確性，

然而卻仍存在著漣漪效應(Ripple Effect)[19]持續影響著開發效率，而這個問題卻存在於跨階段之間，所造成的阻礙卻不得不重視。明確地定義出如何在軟體開發生命週期階段間模型、文件、程式碼中元件的關聯亦成為影響軟體開發難易度及成本的關鍵因素。

2.5 Software Requirement Specification (SRS) (需求規格書)

軟體需求規格書(Software Requirement Specification)[18]乃在描述軟體產品、專案之原始使用者、規格需求，在軟體專案開發的流程中扮演一個最高指導方針的重要角色。軟體需求規格書可以用在評估軟體產品、專案是否達成當初釐定之使用者需求，另一方面當產品交付給客戶後，也可以作為往後該軟體測試驗收時的依據。

軟體需求規格書其應用範圍涵蓋任何資訊系統的軟體，文件中包含前言介紹、系統概述、分項需求、軟體需求規格表、附錄五個大綱項目。適用於全程軟體發展工作，或全程中之部分發展工作。

根據 IEEE Std-830 訂定出軟體需求規格書共有 10 個觀點，共分為 Organized by Mode Version 1&2、User Class、Object、Feature、Stimulus、Response、Functional Hierarchy、Showing Multiple Organizations 以及 Use Case Version，目的是配合軟體開發種類的多樣化提供不同觀點的需求規格書。

2.6 Modeling Traceability & Integration Management

當XUMM模型關係鏈結建立完整之後，便會得到整合各項標準於一個基礎於XML的整合模型。在章節2.7描述了軟體開發模型如何整合以達成彼此的對映關係及交換性，最後經過整合的軟體開發模型XML-Based Unified Model (XUM)。利用此理論方法以實作一個統一模型環境(XUM Software Environment)以達成軟體開發及維護中的可追蹤性。在系統開發的銜接方面，透過關係鏈結的建立將階段性規劃、分析、實作出的Model及Source Code彼此的關係做連結，將Model及Source Code經過轉換成Schema之統一格式，以利於模型在系統開發流程階段間進行資訊的互相交換。

模型追蹤性與整合性管理(Modeling Traceability & Integration Management)中，透過XUM (XML-Based Unified Model)架構來產生所需之XML格式文件，進而利用達到整合性與可追蹤性。主要的架構如圖 十三，針對資料進行經過XUMM Transformation的統一整合格式的步驟，進而利用兩個方法：整合鏈結(Integration Link)、漣漪效應的追蹤(Ripple Effect Tracer)管理並使用XML文件。XUM中所提到的Foundation Modules、Traceability Support Modules分別說明如下：

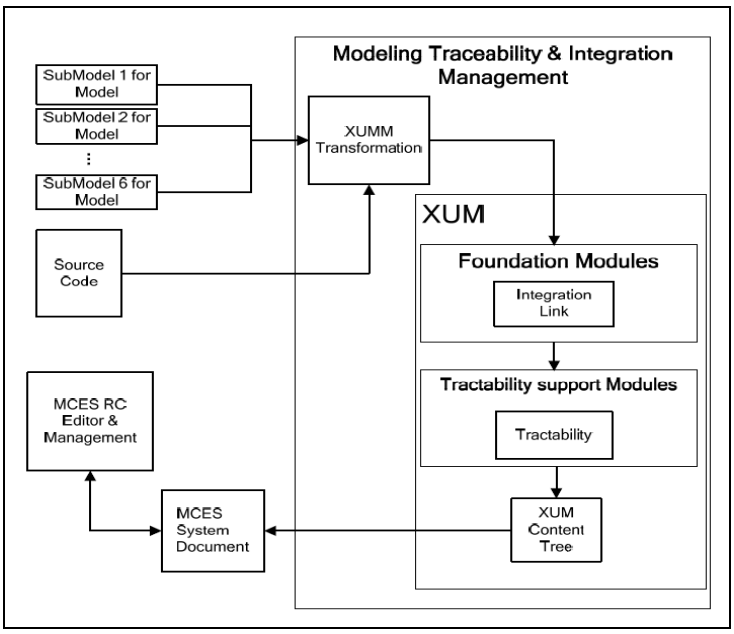

圖 十三、模型追蹤性與整合性管理架構

2.6.1 Foundation Modules

整合鏈結(Integration Link)的目的在於將系統開發中各模組做關聯，每當完成新的模組版本，並將其模組整合存入資料庫時，建立整合鏈結(Integration Link)，並提供模組給其他模組使用，並且由於整合鏈結將系統中的模組串聯起來，當其中來源模組修改時，透過整合鏈結機制，更新資料庫中模組版本，並將系統中重複使用的模組也會一同修改，保持所有模組資訊一致性。

2.6.2 Traceability Support Modules

此部分主要考量了系統開發過程必要的可追蹤性，如漣漪效應的追蹤(Ripple Effect Tracer)[20]。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。建立關聯鏈結的功能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與可追蹤性。

軟體模型間的反映機制(RH)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著改變，以達到軟體維護的過程中各個軟體模型達到同步的機制，也可達到軟體模型間的可追蹤性。如圖 十四，四個點Model A、Model B、Model C、Model D彼此具有關聯。當Model A改變時，透過RH Model B將會被通知，而當Model B改變時，Model C與Model D也將透過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。

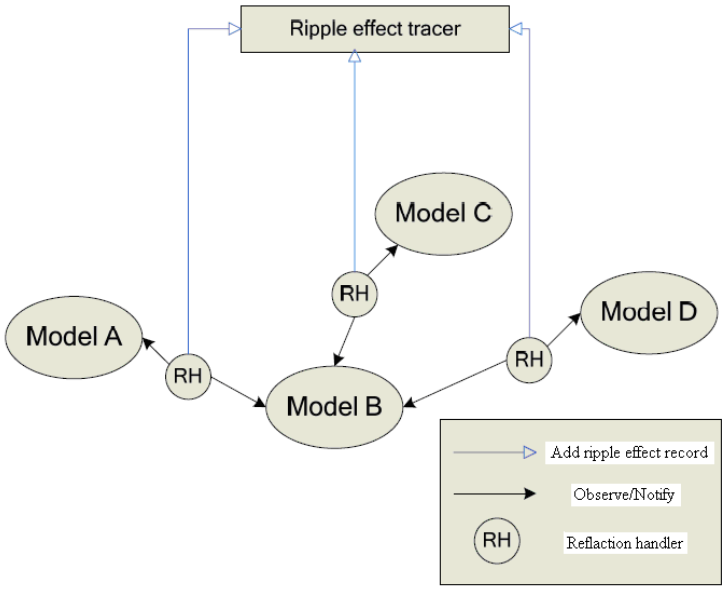

圖 十四、漣漪效應追蹤

2.7 Object-Oriented Development Process (物件導向開發流程)

系統設計方法主要可分為兩大類：

(1). 傳統的結構化方法

(2). 物件導向開發

傳統的結構化方式主要強調如何應用一些概念、策略與工具來提升系統分析與設計的效率，一般利用系統環境圖(Context Diagram)、流程圖(Data Flow Diagram)以及實體關係圖(ER Model)來表示。

而物件導向開發較類似於人類思考模式來分析，以元件化軟體開發方式進行，相對上較容易修改及再利用(Reuse)。在物件導向開發過程主要構成元素就是物件(Object)和類別(Class)，物件可視為問題領域事物的抽象化，可以反應保留系統資訊的能力以及系統互動的能力[7]，類別可視為針對一組共通物件的描述。

物件導向開發流程當中，主要可以分為下列三個階段：[8]

(1). Object-Oriented Analysis, OOA (物件導向分析)：

OOA主要是在需求階段，從需求文件的字彙裡，透過物件以及類別來對問題領域進行塑模。

(2). Object-Oriented Design, OOD (物件導向設計)：

OOD的目的是為了讓開發者能夠了解系統的架構。在OOD的過程當

中會導致物件分解，而OOD主要是使用邏輯以及實體標記法來表示系統靜態(Static)以及動態(Dynamic)的觀點(View)，並將系統間的物件及物件彼此間的關係描述出來。

(3). Object-Oriented Programming, OOP (物件導向程式設計)：

OOP是指在軟體開發過程中，將OOA、OOD的思想進行表達和實現。

2.8 Design Pattern (設計模式、設計樣版)

Design Pattern以Gamma、Helm、Johnson以及Vlissides四人影響最大，他們為設計樣版作了定義[1]：一個樣版(Pattern)是由三個部份組成的規則，分別是一個背景環境(Context)間的關聯、一個問題以及一個解決方案如下表二所示：

表 二、Design Pattern 四個基本特質

項目	描述
Pattern Name	定義一個設計樣版，讓我們可以在高度抽象化階段作設計。
Problem	描述何時可以應用設計樣版。
Solution	以一般化方式描述樣版以應用於不同的環境之下。
Consequence	應用設計樣版的結果，觀察使用中的相互作用，提供評估替代方案與制定決策的要素。

設計樣版可以提供開發者根據前輩的經驗累積重復使用方法來解決面臨的問題，並從中累積良好的開發經驗，提升個人開發能力，此外藉由設計樣版的詞彙提供共通的語言，亦有利於開發團隊的溝通，根據以往的經驗採取最有效的開發方式開發系統。

三、研究方法

本研究中，我們將系統開發過程中所需各階段的需求以及設計文件轉換為以XML格式來表示，透過XUM的概念來將各階段的文件做整合，並建立關係鏈結以達到系統開發的一致性及可追蹤性。

我們將以個案方式進行說明，透過本研究方法當軟體開發流程時，所需進行的相關工作，以及如何以XML文件描述軟體開發流程文件，並說明如何應用XML文件特性達成系統開發流程階段中不同的文件之間關聯鏈結的建立與關聯建立的可行性，以達到提升系統開發流程文件一致性及可追蹤性的目的。

3.1 SRS Document Template

在多個SRS Document Template當中，為了隨後將開發文件XML化表示並透過XUM概念達成需求關係的建立、追蹤此一目的，且由於XUM主要研究針對以功能需求為主的系統，本研究選用IEEE-830 Standard當中For Object & For Use Case兩個Template作為需求規格樣版參考依據，根據兩種需求樣版的特性，制定出針對客戶功能性以及使用案例的需求為擷取重點的需求樣版。

透過軟體需求規格書即可初步定義出系統物件以及彼此作用關係，如此一來便可以節省後續要將需求規格書以XML形式表達時額外的人工轉換人事成本。

以下表 三為本研究中所使用的SRS Template片段，擷取出針對客戶使用需求以及功能性需求的部份需求規格書架構：

表 三、需求規格書樣版

1	Specific Requirements
1.1	External Interface Requirements
1.1.1	User Interfaces
1.1.2	Hardware Interfaces
1.1.3	Software Interfaces
1.1.4	Communications Interfaces
1.2	External Actor Descriptions
1.2.1	Human Actors
1.2.2	Hardware Actors
1.2.3	Software System Actors
1.3	Functional Requirement/Use Case
1.3.1	Functional Requirement 1 Use Case 1
1.3.2	Functional Requirement 2 Use Case 2
.....	.....
1.3.n	Functional Requirement n Use Case n

3.2 Use Case Modeling

在系統開發初步過程當中，最主要的工作除了完整擷取顧客對於系統的需求之外，則是確保根據需求文件內容所開發出來的系統必須符合顧客的需求，在系統開發之前必須釐清開發者與顧客之間認知上是否有差異，以避免開發成本做無謂的提升。

3.2.1 Use Case Modeling With XUM

本研究中使用的Use Case Modeling為根據Booch所提出的方法[5]，主要以UML作為開發流程描述的方式，以Use Case Modeling作為開發起始點，並利用類別圖及循序圖個別以靜態觀點以及動態觀點描述系統觀點，不僅可以提供需求擷取階段發現顧客需求所呈現出來的系統雛形，亦可以將系統開發流程的需求階段以及設計階段作一貫的流程整合，更可以避免使用不同的描述標準而造成以XML文件格式轉換描述上的不一致，Use Case Modeling承接本研究主要流程為下列幾點：

(1). 在設計之前先定義出系統功能項目。

(2). 引出系統行為者與目的。

(3). 根據行為者與目的訂出Use Case。

(4). 根據Use Case及顧客需求引出使用Use Case Scenarios。

(5). 以Sequence Diagram描述Scenarios中的事件。

(6). 根據Sequence Diagram定義出系統設計之Class Diagram。

(7). 根據初步文件決定使用之設計樣版做為開發依據。

3.2.2 Use Case Scenarios

本研究根據使用 Use Case Diagram 以及 Use Case Scenarios 來描述使用者需求的部份，藉由對使用者案例描述出使用案例腳本，將顧客對於系統的使用案例要求以腳本的模式敘述，以釐清訪談者對於顧客的需求是否有誤解，讓開發者能夠初步地發現系統中所包含的類別、物件及事件流程，下表 四為本研究之 Use Case Scenarios Template：

表 四、Use Case Scenarios Template

使用案例名稱		
行為者	主要行為者	次要行為者
使用案例描述		
功能需求		
前置條件		
後置條件		
主要成功情境基本流程		
例外事件情境基本流程		

3.2.3 Use Case to XML TAG

為了符合XUM以及物件導向開發的精神，我們將開發流程文件以XML形式表示，並透過以文件表示物件以及物件之間的關係，讓系統開發流程可以藉由XML技術達到整合開發流程文件、變更軟體需求追蹤以及物件導向開發概念。

要將開發流程文件轉換成XML格式方式呈現，首先需要定義出XML Document TAG，本研究使用案例圖以及使用案例腳本與XML TAG對照如下表 五、表 六：

表 五、Use Case基本元素與XML元素對照表

Models/Paradigms	Elements	XUM Element Representations
Use Case	使用案例名稱	<UseCaseName>
	使用案例編號	<UseCaseNO>
	行為者	<Actor>
	主要行為者	<MainActor>
	次要行為者	<SecondaryActor>
	使用案例描述	<UseCaseDescription>
	功能需求	<RequirementNO>
	前置條件	<Precodition>
	後置條件	<Postcodition>
	主要成功情境基本流程	<BasicProcess>
	例外事件情境基本流程	<Exception>

表 六、Use Case Diagram 基本元素與 XML 元素對照表

Models/Paradigms	Elements	XUM Element Representations
Use Case Diagram	使用案例圖名稱	<UseCaseName>
	行為者	<Actor>
	主要行為者	<MainActor>
	次要行為者	<SecondaryActor>
	使用案例	<UseCase>
	關係	<Relation>

3.3 Sequence Diagram

我們可以根據 Use Case Diagram 以及 Use Case Scenarios 的內容以 Sequence Diagram 描述系統基本設計概念,並作為將需求以及設計元素進行連結的媒介。透過循序圖引出系統類別並敘述功能之間的互動關係，在角色和物件之間的描述讓我們可以按照系統中事件流以較精確順序來描述隨時間變化的系統行為，並且訂定出系統開發物件所應包含的 Method，以及不同的物件之間的動態關聯和作用關係。

並透過XML格式描述系統事件循序關係，做為整合開發流程的文件之一，循序圖與XML TAG對照如下表 七：

表 七、Sequence Diagram 基本元素與 XML 元素對照表

Models/Paradigms	Elements	XUM Element Representations
Sequence Diagram	事件名稱	<Event>
	操作	<Operation>
	類別/物件	<Class>
	事件循序編號	<SequenceNO>

3.4 Class Diagram

對於客戶訪談結果，依照 Sequence Diagram 以及所需求規格書當中描述的系統內容，我們可以將系統的靜態結構以 Class Diagram 的方式做描述，將系統的靜態關聯做圖形化的方式描述讓程式設計師可以根據 Sequence Diagram 所描述的類別設計出對於此系統開發所適合的物件內容。並且透過反覆的循環可以讓系統設計師釐清先前的設計階段訂定出系統開發架構上的缺失。

以下表 八為根據此系統需求以Class Diagram方式所描述的靜態觀點當中主要包含的幾項 XML 基本元素對照：

表 八、Class Diagram 基本元素與 XML 元素對照表

Models/Paradigms	Elements	XUM Element Representations
Class Diagram	類別名稱	<ClassName>
	類別特徵	<Feature>
	類別關係	<ClassRelation>
	操作	<Operation>
	參數串列	<Parameter>
	可視性	<Visibility>
	屬性	<Attribute>
	屬性型態	<type>

3.5 Development Document Relation Link

承襲XUMM概念，以XML格式描述開發文件將軟體開發流程視為物件的集合，透過建立物件之間的關聯落實物件導向開發的概念，由於XML為了符合XML Well-formedness的判定，在文件當中以TAG輔助描述物件之間的關聯在XML Schema仍然會被視為Component之一，XML Document描述關聯元件結構如下圖 十五、圖 十六兩種：

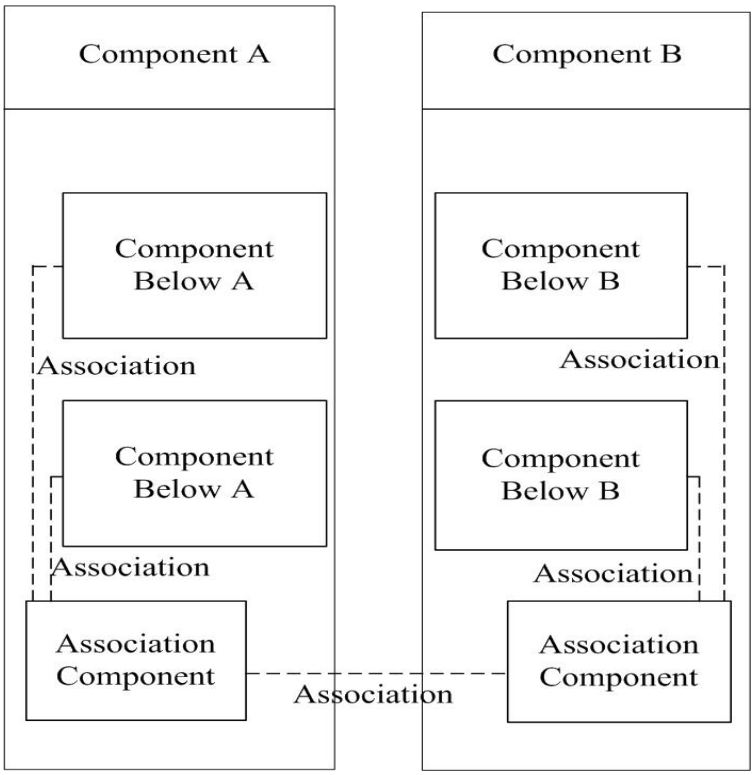

圖 十五、結合式 XML Association Component Model

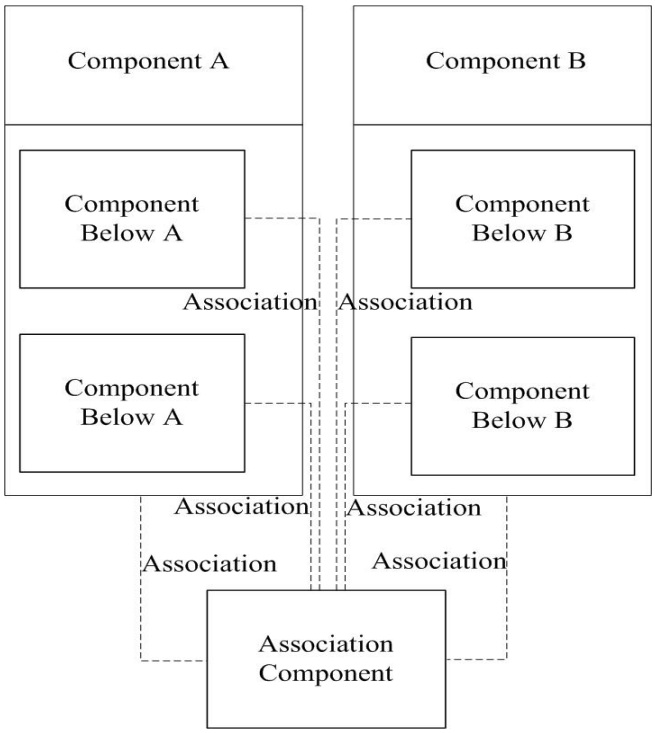

圖 十六、分離式 XML Association Component Model

XUM將XML文件中TAG都視為Component，並透過建立Association Component描述Component之間的關係，然而TAG在XML文件中有階層的關係，以TAG的階層表示文件的結構，而在XML Schema中階層的表示是上一層TAG階層結構中的TAG會被視為包覆在上層的描述當中的元件如此一來為了達到物件導向開發及需求變更管理的目的，勢必必須開發XML Parser將XUM Component中描述關聯的部份從中抽離出來管理，然而XUM描述物件之間關聯的Association Component在XML文件編輯若是採取結合式描述方式，在Component的管理及分類上將會增加複雜性，而若是採取分離式描述方式，在XUM關聯的描述集合將會相對龐大且複雜。

本研究為了整合軟體開發流程文件，範圍橫跨需求階段、設計階段及軟體開發階段文件，以TAG方式描述關聯並無法完全滿足所需，經由物件共

享的概念我們將不同階段內所描述的同一物件以一致的表示方式呈現，並透過XLink建立關聯，將文件內的關聯以及文件之間的關聯取代以XML訂定的XLink標準來表示如下：

```
<xs:element name="Relation">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="UnificationLink" maxOccurs="unbounded"/>
            <xs:element ref="AssociationLink" minOccurs="0"
              maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

我們將Component關聯的建立區分為以下Unification Link以及Association Link兩種。

```
<xs:element name="UnificationLink">
    <xs:complexType>
        <xs:choice>
            <xs:element ref="Abstraction"/>
            <xs:element ref="Integration"/>
            <xs:element ref="SourceCode"/>
        </xs:choice>
    </xs:complexType>
</xs:element>
```

再根據原本的Unification Link延伸出Abstraction、Integration以及Source Code Link，同時加入取代XUM原本文件內的物件彼此關聯的Association Link描述方式並增加開發流程不同階段文件間彼此的關聯的Document Integration Link描述方式，以一致的方式表達XUM對於關聯的描述方式。

```
<xs:complexType>
    <xs:simpleContent>
        <xs:extension>
            <xs:attribute ref="xlink:type" use="required"/>
            <xs:attribute ref="xlink:title" use="required"/>
            <xs:attribute ref="xlink:show" use="required"/>
            <xs:attribute ref="xlink:herf" use="required"/>
            <xs:attribute ref="xlink:actuate" use="required"/>
            </xs:extension>
    </xs:simpleContent>
</xs:complexType>
```

本研究利用XLink Definition當中的xlink:title來將關聯的種類做區分，不同於XUM應用Association Component，關聯的建立式描述於文件當中，所以我們利用xlink:herf來描述關聯目標，而無須指定關聯的起始目標與終點目標，並且以xlink:show來決定開發流程中使用者檢視相關聯物件以及文件的方式。

3.6 Design Pattern

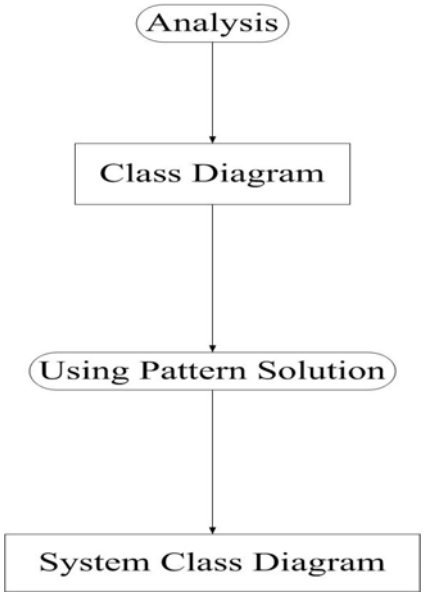

圖 十七、設計樣版使用流程

如上圖 十七所示為本研究當中針對系統的功能需求開發使用Design Pattern之設計階段流程圖。當我們初步類別圖以及循序圖決定出來之後我們可以根據靜態面以及動態面去分析這個系統的特性，並且挑選適當的Design Pattern來修改系統初步的類別圖設計為系統類別圖，作為物件導向開發的參考依據。

在開發流程當中的Design階段文件，根據初步分析決定所使用的Design Pattern將系統類別圖設計結果以XML型式表示作為系統Implementation階段參考文件，並將Design Pattern關聯內容描述於流程文件當中。

3.7 XUM Document

本研究利用XUM概念結合軟體開發流程文件，並透過XLink建立關聯達成需求變更管理及變更追蹤，我們利用XLink標準當中xlink屬性定義actuate=“onLoad”及show=“embed”作為開發流程文件整合鏈結，當使用者檢視XUM整合文件時，透過定義XLink屬性讓使用者可以自由選擇所欲參考的文件，並且提供原始頁面及開新頁面檢視方式減少使用者在參考文件是關聯性的混亂，透過提供文件關聯檢視的方式便可以獨立編輯個別的文件，並在文件當中引述同一個Component的時機以XML標準重複引用Document內文，而無須在開發流程文件當中重複描述同一個Component，透過文件整合鏈結的使用來達到XUM物件共享概念，同時也避免在不同文件當中描述方面的落差、不一致甚至是矛盾，造成開發者在檢視需求及設計階段文件的誤解進而導致開發錯誤及失敗。

四、案例研究

在這個章節中我們以實際的研究案例呈現上一章的研究方法所描述，如何應用XUM概念整合系統開發流程文件，並透過Unification、Association以及Document Integration Link的建立達到開發流程的關聯追蹤，讓開發者可以在開發系統的同時檢視需求及設計階段文件內容，以及開發流程元件之間的關聯，達到開發流程的一致性、可追蹤性。

4.1 SRS Example

此節當中，本研究為根據研究方法中所選用的需求規格書樣版做為需求文件擷取的依據樣版。

由於功能性需求是往後開發流程用以釐清行為以及相關聯的物件彼此之間的作用順序以及關係，所以在功能性需求描述方面必須明瞭、清楚地表達主詞、動詞、受詞以及的所做的相關功能描述方式來建立，擷取出Chap.1.2以及1.3為例，以此Template為依據的SRS為下表 九：

表 九、需求規格書範例

1.2	External Actor Descriptions
1.2.1	客戶(會員)
1.2.2	旅行社資料庫
1.2.3	旅行社系統
1.3	Functional Requirement
1.3.1	系統需求
1.3.1.1	會員登入功能
	會員必須透過帳號及密碼登入才能使用系統提供的所有功能。
1.3.1.2	會員資料修改功能
	會員可以選擇更改個人會員密碼。

1.3.1.3	個人資料查詢功能
	會員登入後可以查詢資料庫內個人資料(姓名，電話，地址，身分證字號，生日)及訂購歷史記錄。
1.3.1.4	個人資料修改功能
	會員登入之後可以針對個人資訊進行修改。
1.3.1.5	查詢航班功能
	客戶透過輸入欲查詢的航班關鍵字(日期，班次，地點)進行資料庫內相關航班資料的查詢。
1.3.1.6	訂購機票功能
	客戶可以根據航班查詢結果從中選擇欲購買的班次。
1.3.1.7	索取訂購機票個人資料
	客戶選擇完欲訂購的班次之後要求填入訂購明細及乘客資料。
1.3.1.8	取消訂購機票功能
	客戶行程有所改變時可以選擇取消訂購，取消訂購後修改資料庫銷售紀錄及個人訂購記錄。
1.3.1.9	取票通知及帳單的生成及列印
	客戶選擇欲訂購機票之後根據客戶填入資料產生取票通知及帳單提供客戶繳費及取票的依據。

4.2 Use Case Diagram

根據客戶的需求規格書，我們以透過使用Use Case Diagram描繪出客戶對於系統功能的基本雛型，訂定系統/應用程式的外部和內部元素以及系統範圍。因為是以圖形化方式呈現，與使用者溝通系統行為需求將更加便利，以視覺化的方式表達系統如何滿足所收集的業務規則，以及特定的用戶需求等資訊。

以下圖 十八、圖 十九、圖 二十為根據需求以Use Case Diagram描繪出基本相關功能以及客戶系統範圍：

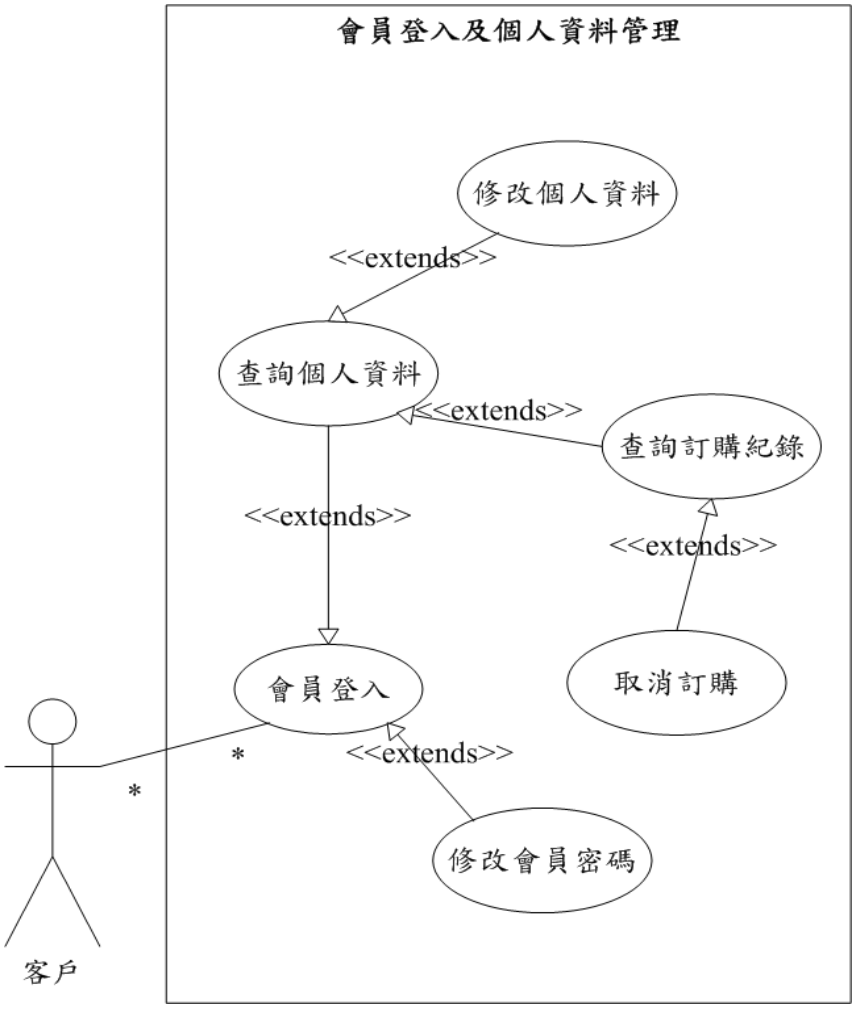

圖 十八、會員及個人資料管理功能使用案例圖

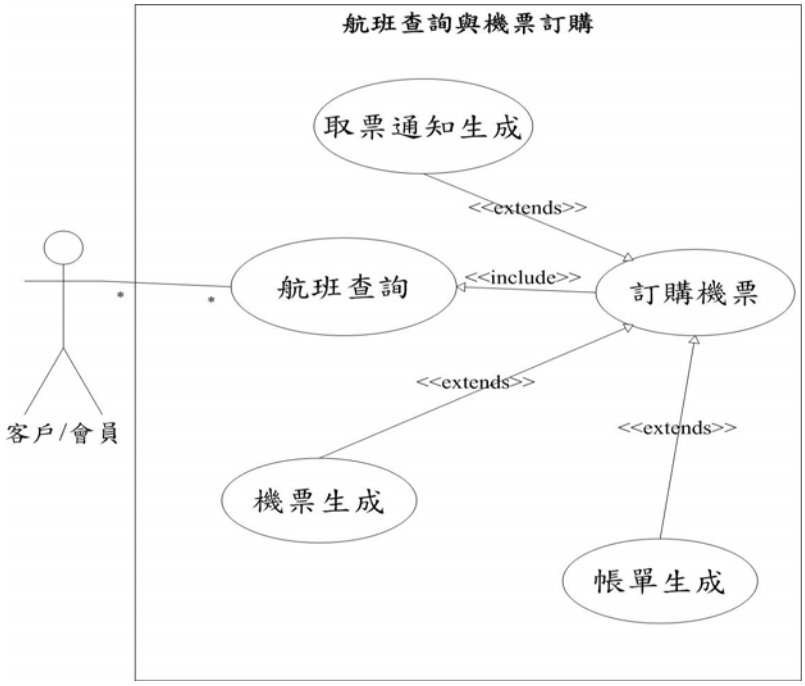

圖 十九、航班查詢及機票訂購取消使用案例圖

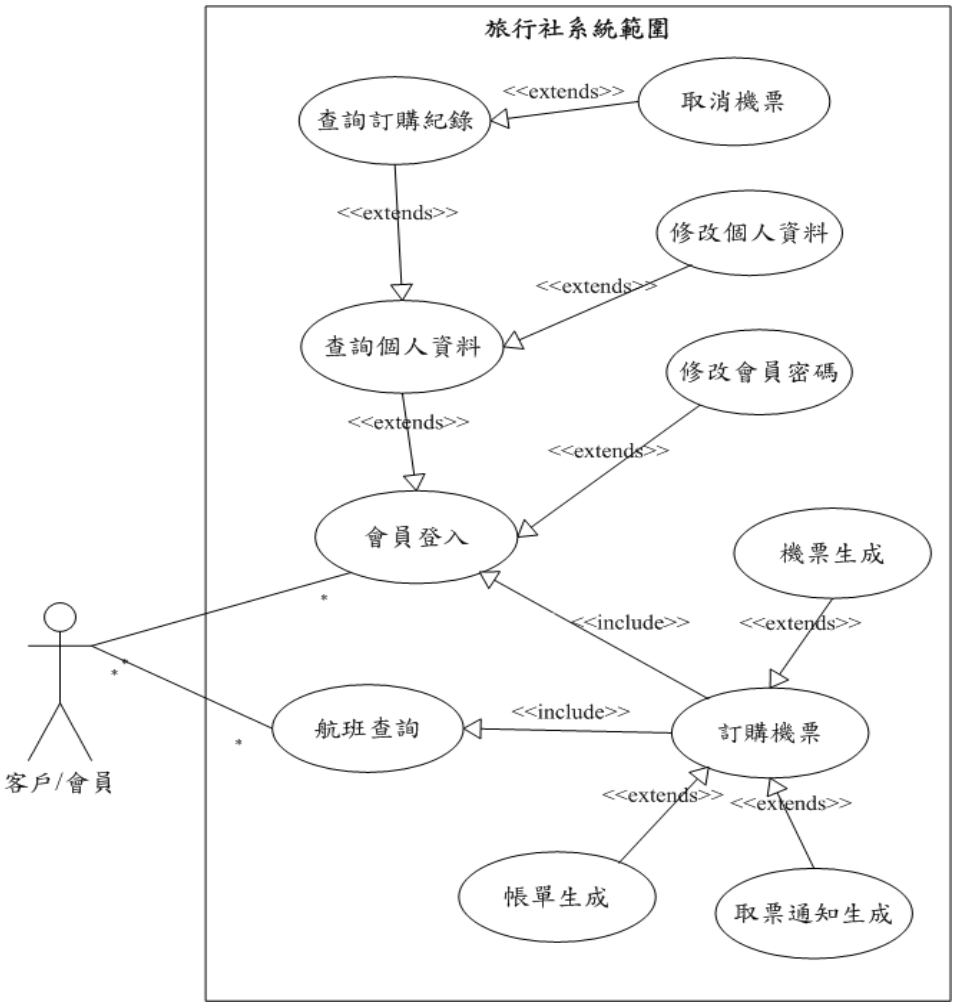

圖 二十、旅行社系統範圍

4.3 Use Case Scenarios

根據對於使用者案例描述出使用案例腳本，將顧客對於系統的使用案例要求以腳本的模式敘述，以釐清訪談者對於顧客的需求是否有誤解，並藉以讓開發者瞭解系統中所包含的操作物件及使用事件流程，協助開發者根據訪談結果訂定出系統架構。

以下表 十～表 十四為本研究研究案例的Use Case Scenarios描述內容：

表 十、登入會員腳本

使用案例編號	U-1	
使用案例名稱	登入會員功能	
行為者	主要行為者	次要行為者
	客戶／會員	無
使用案例描述	會員必須透過帳號及密碼登入才能使用系統提供的所有功能。	
功能需求	1.3.1.1	
前置條件	無	
後置條件	顯示會員功能首頁	
主要成功情境基本流程	(1)客戶點選登入會員。 (2)頁面出現填寫會員帳號及密碼表單。 (3)客戶填寫該帳號密碼之後按確認(E1)。 (4)顯示登入成功。 (5)以會員身份回到首頁進行操作。	
例外事件情境基本流程	(E1.1)顯示密碼錯誤。	

表 十一、會員資料查詢及修改腳本

使用案例編號	U-1.1	
使用案例名稱	會員資料查詢及修改功能	
行為者	主要行為者	次要行為者
	客戶／會員	無
使用案例描述	會員登入後可以查詢個人資料(姓名，電話，地址，身分證字號，生日)及訂購歷史記錄，亦可以選擇更改個人會員密碼以及個人資料。	

功能需求	1.3.1.2 1.3.1.3 1.3.1.4
前置條件	登入
後置條件	會員個人資料
主要成功情境基本流程	(1)客戶可選擇修改會員密碼進入修改密碼頁面。 (2)頁面要求客戶填入原始密碼以及欲更改密碼。 (3)客戶填入新舊密碼後按確認進行修改(E2)。 (4)客戶可點選查詢個人資料向系統要求檢視。 (5)系統查詢客戶個人基本資料以及訂購紀錄。 (6)顯示與客戶相關個人資料及訂購紀錄。 (7)客戶可以新增修改個人基本資料內容。 (8)客戶確認修改內容之後可以按儲存確定修改寫入。
例外事件情境基本流程	(E2.1)原始密碼與會員密碼不符，頁面回傳密碼錯誤訊息。

表 十二、查詢航班資訊腳本

使用案例編號	U-2	
使用案例名稱	查詢航班資訊功能	
行為者	主要行為者	次要行為者
	客戶／會員	無
使用案例描述	客戶根據欲查詢的航班關鍵字(日期，班次，地點)進行相關航班資料的查詢。	
功能需求	1.3.1.5	
前置條件	無	
後置條件	訂購機票	
主要成功情境基本流程	(1)客戶可點選航班查詢向系統要求航班資訊。 (2)系統要求客戶填入查詢關鍵字。 (3)系統根據客戶填入關鍵字於資料庫中尋找。 (4)將客戶查詢的相關航班資料回傳給客戶檢視。	
例外事件情境基本流程	無	

表 十三、訂購機票腳本

使用案例編號	U-2.1	
使用案例名稱	訂購機票功能	
行為者	主要行為者	次要行為者
	客戶／會員	無
使用案例描述	客戶可以根據查詢結果從中選擇欲購買的班次，選擇完	

	欲訂購的班次之後要求填入購買資料及乘客資料，並且根據客戶填入資料產生取票通知及帳單產生提供客戶繳費及取票的依據。
功能需求	1.3.1.6 1.3.1.7 1.3.1.9
前置條件	登入，查詢航班資訊
後置條件	產生訂購紀錄、取票通知以及帳單
主要成功情境基本流程	(1)客戶由搜尋關鍵字列出符合的航班資訊當中選取並點選訂購。 (2)頁面顯示要求客戶填入訂購資訊。 (3)客戶選取欲購買的機票資訊(E3)。 (4)系統計算客戶選取的機票資訊計算票價結果回傳給客戶。 (5)客戶點選確認後顯示要求填入搭乘者個人資料。 (6)客戶按下確定購買後系統將訂購記錄存入資料庫完成訂票(E4)。 (7)系統根據客戶訂購之機票資訊及客戶個人資料產生帳單及取票通知。 (8)系統修改客戶個人訂購記錄。
例外事件情境基本流程	(E3.1)客戶輸入之等級剩餘機位數量不足回到步驟(3)。 (E4.1)輸入資料不完整則回到步驟(6)。

表 十四、取消機票腳本

使用案例編號	U-1.1.1	
使用案例名稱	取消機票功能	
行為者	主要行為者	次要行為者
	客戶／會員	無
使用案例描述	客戶行程有所改變時可以選擇取消訂購，並修改資料庫銷售紀錄及個人訂購記錄。	
功能需求	1.3.1.8	
前置條件	查詢個人資料	
後置條件	客戶訂購紀錄	
主要成功情境基本流程	(1)客戶點選取消訂購，系統輸出可取消之訂購記錄(E6)。 (2)客戶由訂購記錄選取欲取消的訂購資料。 (3)客戶按下確認之後系統修改資料庫中銷售紀錄及客戶訂購記錄。 (4)系統取消客戶訂購之機票。	
例外事件情境基本流程	(E6.1)查詢客戶無可取消之訂購記錄則輸出查無資料。	

4.4 Sequence Diagram

根據 Use Case Scenarios 的內容，我們利用 Sequence Diagram 圖形化的方式描述系統互動流程，將需求及設計元素進行連結。

以下圖 二十一、圖 二十二為本研究案例循序圖：

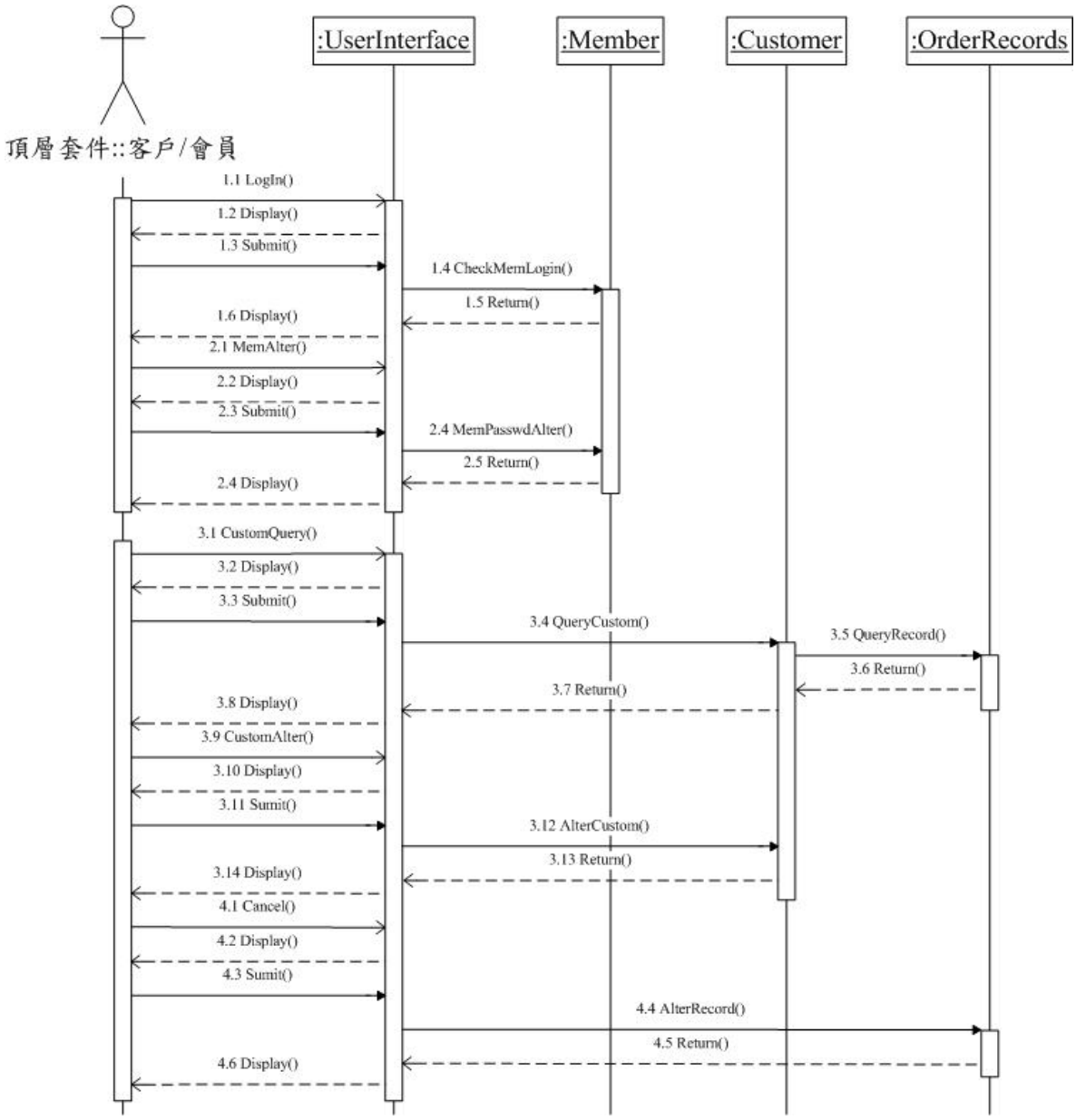

圖 二十一、會員登入修改密碼及個人資料查詢修改循序圖

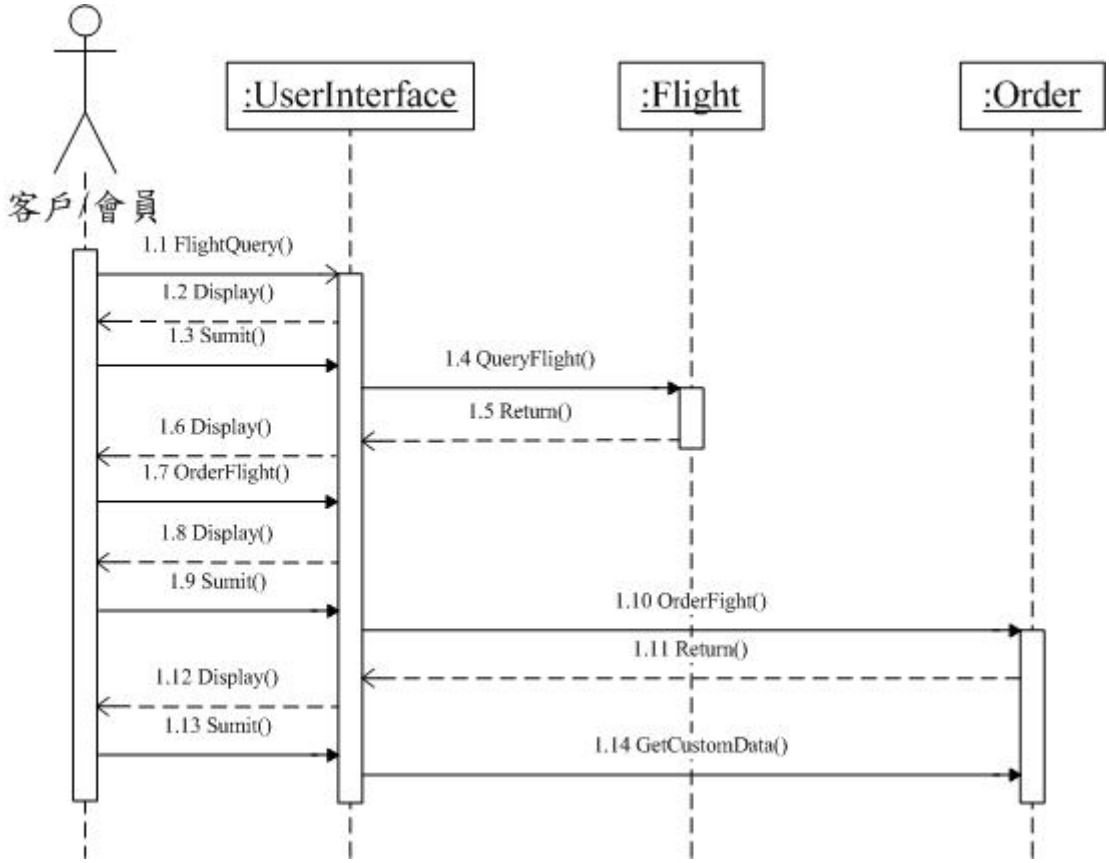

圖 二十二、航班查詢及機票訂購循序圖

4.5 Class Diagram

依照 Sequence Diagram 所呈現的系統互動，我們可以將系統的靜態結構以 Class Diagram 的方式做初步的描述，描述系統的靜態類別關聯讓程式設計師可以根據 Sequence Diagram 所描述的互動關係設計出對於此系統開發所適合的類別/物件。並且透過反覆的循環可以讓系統設計師釐清先前的設計階段訂定出系統設計階段架構上的缺失。

下圖 二十三為此研究案例系統需求以Class Diagram所描述的初步靜態觀點。

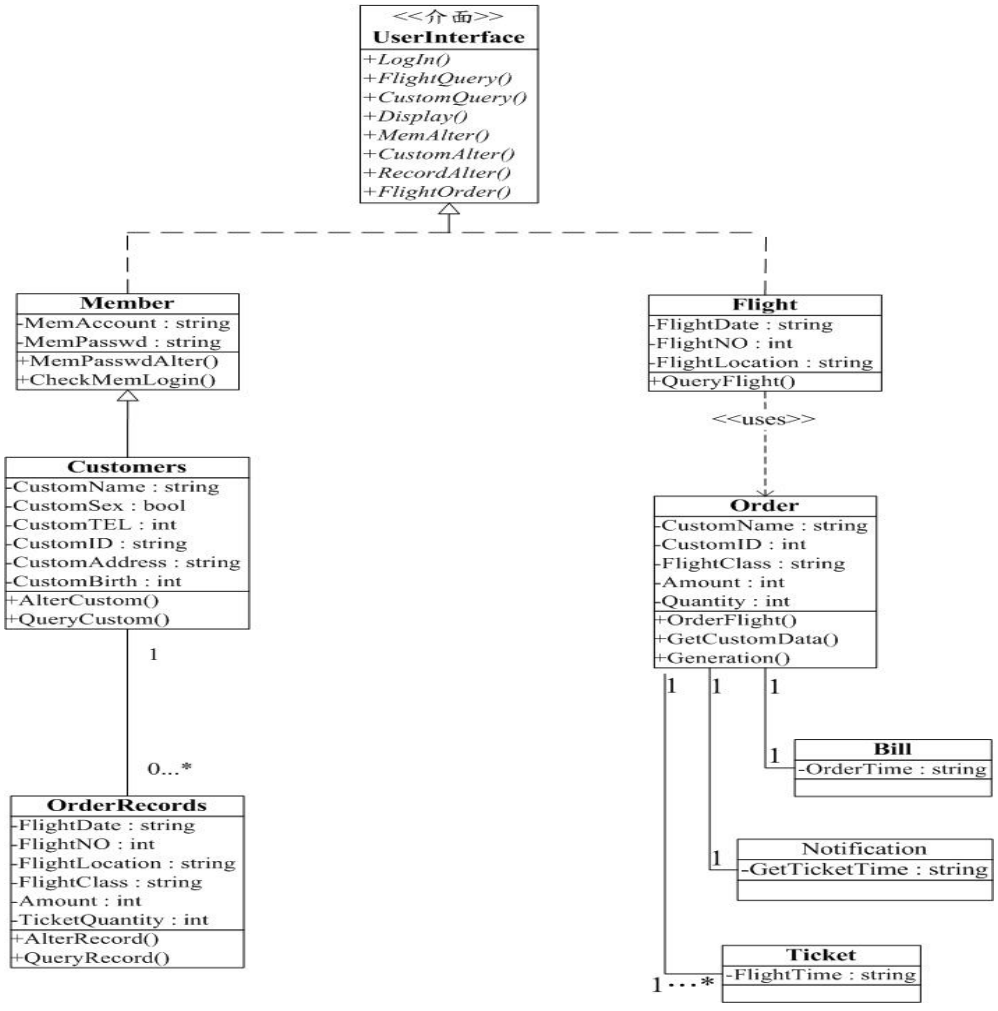

圖 二十三、機票預訂系統類別圖

4.6 Design Pattern

當我們初步類別圖以及循序圖決定出來之後我們可以根據靜態面以及動態面去分析這個系統的特性，並且挑選適當的Design Pattern來使用。

根據分析的循序圖的結果，系統的功能呼叫流程有很大的相似度，系統主要功能是由查詢以及修改去搭配而成，所以根據特性過濾出Prototype及Template Method兩種較為適合，再經過比較兩者的差異，Prototype主要是程式中不變的Method提取出來，再將原本的類別分類，減少原本子類別的數量，而Template Method則是重複的演算法輪廓提取出來讓子類別去繼承，並根據各自的需求去完成完整的程式碼，而本範例系統的特性查詢以及修改又必須分類為航班、個人資料以及訂購紀錄的分別，以較為符合本範例系統使用的考量之下選擇了Template Method作為本次選用的Design Pattern。

以下如圖 二十四Template Method的Class Diagram基本結構：

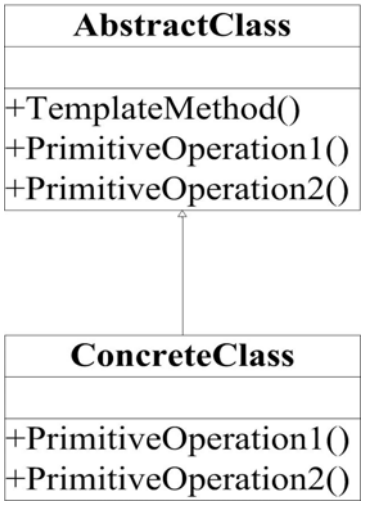

圖 二十四、Basic Class Diagram of Template Method

4.7 System Class Diagram

將我們初步的Class Diagram與我們所選定的Design Pattern做進一步的規劃之後，可以產生如下圖 二十五適合我們將來用於實際開發研究案例系統的System Class Diagram。

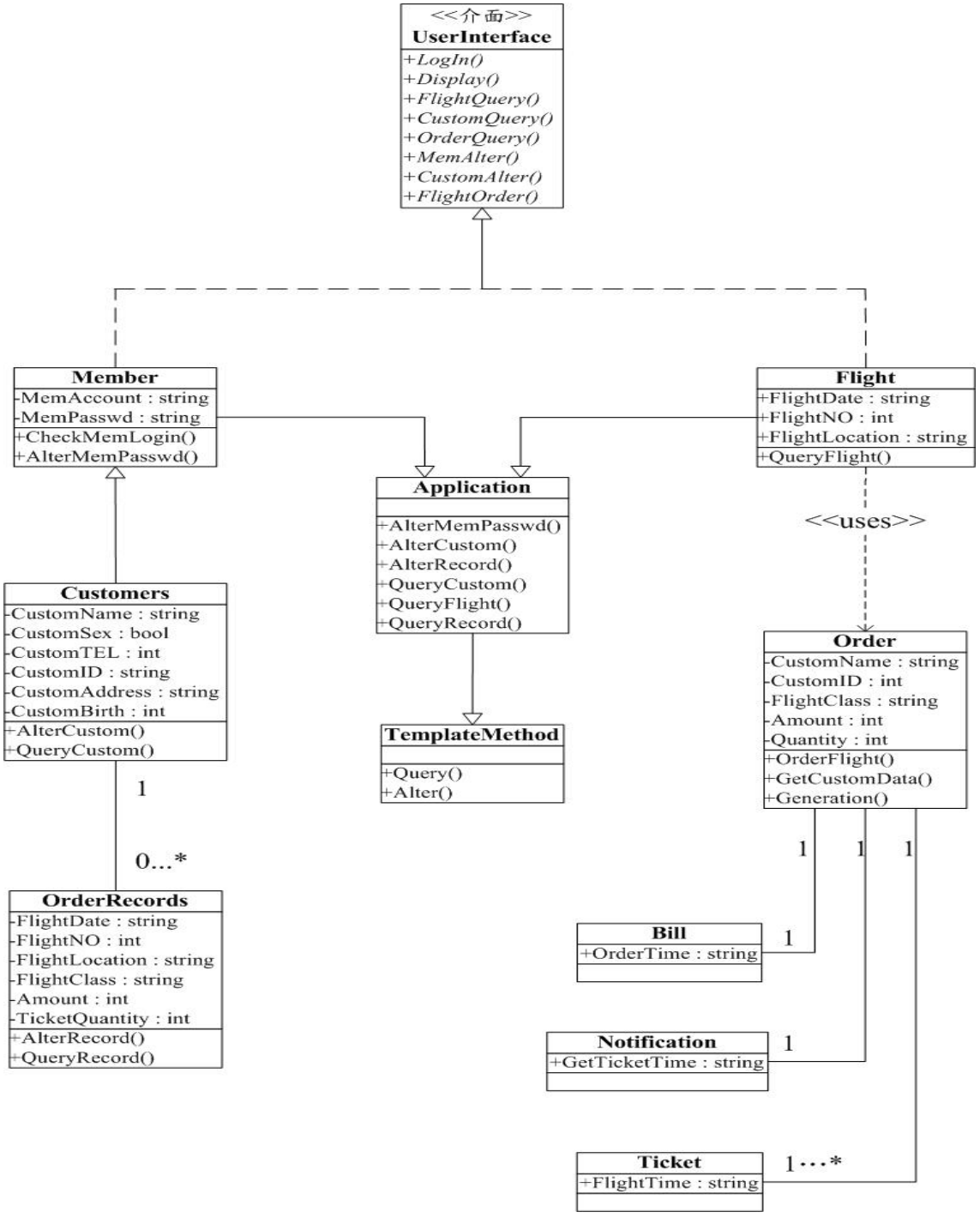

圖 二十五、System Class Diagram

4.8 XUM Schema

根據本研究案例研究之XUM Schema及XUM Structure Model如下表 十五、圖 二十六：

表 十五、XUM Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:import namespace="http://www.w3.org/1999/xlink" schemaLocation="XUM+XLink1.xsd"/>
    <xs:element name="XUM">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="UseCaseDiagram"/>
                <xs:element ref="ClassDiagram"/>
                <xs:element ref="SequenceDiagram"/>
                <xs:element ref="DesignPattern"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="UseCaseDiagram">
        <xs:complexType>
            <xs:choice>
                <xs:sequence>
                    <xs:element ref="UseCaseDiagramName"/>
                    <xs:element ref="UseCase" maxOccurs="unbounded"/>
                    <xs:element ref="FunctionRelation" maxOccurs="unbounded "/>
                </xs:sequence>
            </xs:choice>
        </xs:complexType>
    </xs:element>
    <xs:element name="FunctionRelation">
        <xs:complexType>
            <xs:choice>
                <xs:sequence>
                    <xs:element ref="Function"/>
                    <xs:element ref="Relation"/>
                </xs:sequence>
            </xs:choice>
```

```
        </xs:complexType>
    </xs:element>
    <xs:element name="UseCaseDescription" type="xs:string"/>
    <xs:element name="UseCase">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="UseCaseNO"/>
                <xs:element ref="Relation" minOccurs="0"/>
                <xs:element ref="UseCaseName"/>
                <xs:element ref="Actor"/>
                <xs:element ref="UseCaseDescription"/>
                <xs:element ref="RequirementNO" maxOccurs="unbounded"/>
                <xs:element ref="Precodition" maxOccurs="unbounded"/>
                <xs:element ref="Postcodition" maxOccurs="unbounded"/>
                <xs:element ref="Scenario" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="UnificationLink">
        <xs:complexType>
            <xs:choice>
                <xs:element ref="Abstraction"/>
                <xs:element ref="Intergration"/>
                <xs:element ref="SourceCode"/>
            </xs:choice>
        </xs:complexType>
    </xs:element>
    <xs:element name="SequenceDiagram">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="Event" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="Scenario">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="BasicProcess" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
```

```
<xs:element name="RequirementNO">
    <xs:simpleType>
        <xs:restriction base="xs:string"/>
    </xs:simpleType>
</xs:element>
<xs:element name="ClassOperation">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Operation" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="ClassName">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Name"/>
            <xs:choice>
                <xs:element ref="ClassOperation"/>
                <xs:element ref="ClassAttribute"/>
            </xs:choice>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="ClassDiagram">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Class" maxOccurs="unbounded"/>
            <xs:element ref="Relation" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="ClassAttribute">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Attribute" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="Class">
    <xs:complexType>
        <xs:sequence>
```

```
                    <xs:element ref="Feature"/>
                    <xs:element ref="ClassName"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="BasicProcess">
            <xs:complexType>
                <xs:choice>
                    <xs:element ref="Exception"/>
                    <xs:element ref="Process"/>
                </xs:choice>
            </xs:complexType>
        </xs:element>
        <xs:element name="AttributeTitle">
            <xs:simpleType>
                <xs:restriction base="xs:string"/>
            </xs:simpleType>
        </xs:element>
        <xs:element name="Actor">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="MainActor"/>
                    <xs:element ref="SecondaryActor"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="Attribute">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="AttributeTitle"/>
                    <xs:element ref="Visibility"/>
                    <xs:element ref="type"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="AssociationLink">
            <xs:complexType>
                <xs:choice>
                    <xs:element ref="Association"/>
                    <xs:element ref="Dependency"/>
                    <xs:element ref="Generlization"/>
```

```
                    <xs:element ref="Realization"/>
                </xs:choice>
            </xs:complexType>
        </xs:element>
        <xs:element name="Relation">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="UnificationLink" maxOccurs="unbounded"/>
                    <xs:element ref="AssociationLink" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="Operation">
            <xs:complexType mixed="true">
                <xs:sequence minOccurs="0">
                    <xs:element ref="OperationTitle"/>
                    <xs:element ref="Visibility"/>
                    <xs:element ref="Parameter"/>
                </xs:sequence>
            </xs:complexType>
        <xs:element name="EventSequence">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="Name"/>
                    <xs:element ref="Operation"/>
                    <xs:element ref="SequenceNO"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="Event">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="Relation" maxOccurs="unbounded "/>
                    <xs:element ref="EventTitle"/>
                    <xs:element ref="EventSequence" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="DesignPattern">
            <xs:complexType>
                <xs:sequence>
```

```
                    <xs:element ref="Class" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="Exception">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="Process"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:simpleType name="ST_Realization">
            <xs:restriction base="xs:string"/>
        </xs:simpleType>
        <xs:simpleType name="ST_Intergration">
            <xs:restriction base="xs:string"/>
        </xs:simpleType>
        <xs:simpleType name="ST_Generlization">
            <xs:restriction base="xs:string"/>
        </xs:simpleType>
        <xs:simpleType name="ST_Dependency">
            <xs:restriction base="xs:string"/>
        </xs:simpleType>
        <xs:simpleType name="ST_Association">
            <xs:restriction base="xs:string"/>
        </xs:simpleType>
        <xs:simpleType name="ST_Abstraction">
            <xs:restriction base="xs:string"/>
        </xs:simpleType>
        </xs:element>
        <xs:element name="Name">
            <xs:simpleType>
                <xs:restriction base="xs:string"/>
            </xs:simpleType>
        </xs:element>
        <xs:element name="MainActor">
            <xs:simpleType>
                <xs:restriction base="xs:string"/>
            </xs:simpleType>
        </xs:element>
        <xs:element name="type">
```

```
            <xs:simpleType>
                <xs:restriction base="xs:string"/>
            </xs:simpleType>
        </xs:element>
        <xs:element name="SequenceNO">
            <xs:simpleType>
                <xs:restriction base="xs:string"/>
            </xs:simpleType>
        </xs:element>
        <xs:element name="Visibility">
            <xs:simpleType>
                <xs:restriction base="xs:string"/>
            </xs:simpleType>
        </xs:element>
        <xs:element name="UseCaseName">
            <xs:simpleType>
                <xs:restriction base="xs:string"/>
            </xs:simpleType>
        </xs:element>
        <xs:element name="UseCaseDiagramName">
            <xs:simpleType>
                <xs:restriction base="xs:string"/>
            </xs:simpleType>
        </xs:element>
        <xs:element name="Process">
            <xs:simpleType>
                <xs:restriction base="xs:string"/>
            </xs:simpleType>
        </xs:element>
        <xs:element name="SecondaryActor">
            <xs:simpleType>
                <xs:restriction base="xs:string"/>
            </xs:simpleType>
        </xs:element>
        <xs:element name="Precodition">
            <xs:simpleType>
                <xs:restriction base="xs:string"/>
            </xs:simpleType>
        </xs:element>
        <xs:element name="Postcodition">
            <xs:simpleType>
```

```
                <xs:restriction base="xs:string"/>
            </xs:simpleType>
        </xs:element>
        <xs:element name="OperationTitle">
            <xs:simpleType>
                <xs:restriction base="xs:string"/>
            </xs:simpleType>
        </xs:element>
        </xs:element>
        <xs:element name="FunctionRelation">
            <xs:simpleType>
                <xs:restriction base="xs:string"/>
            </xs:simpleType>
        </xs:element>
        <xs:element name="FunctionName">
            <xs:simpleType>
                <xs:restriction base="xs:string"/>
            </xs:simpleType>
        </xs:element>
        <xs:element name="Function">
            <xs:simpleType>
                <xs:restriction base="xs:string"/>
            </xs:simpleType>
        </xs:element>
        <xs:element name="Feature">
            <xs:simpleType>
                <xs:restriction base="xs:string"/>
            </xs:simpleType>
        </xs:element>
        <xs:element name="EventTitle">
            <xs:simpleType>
                <xs:restriction base="xs:string"/>
            </xs:simpleType>
        </xs:element>
        <xs:element name="Realization">
            <xs:complexType>
                <xs:simpleContent>
                    <xs:extension base="ST_Realization">
                        <xs:attribute ref="xlink:type" use="required"/>
                        <xs:attribute ref="xlink:title" use="required"/>
                        <xs:attribute ref="xlink:show" use="required"/>
```

```
                    <xs:attribute ref="xlink:herf" use="required"/>
                    <xs:attribute ref="xlink:actuate" use="required"/>
                </xs:extension>
            </xs:simpleContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="Intergration">
        <xs:complexType>
            <xs:simpleContent>
                <xs:extension base="ST_Intergration">
                    <xs:attribute ref="xlink:type" use="required"/>
                    <xs:attribute ref="xlink:title" use="required"/>
                    <xs:attribute ref="xlink:show" use="required"/>
                    <xs:attribute ref="xlink:herf" use="required"/>
                    <xs:attribute ref="xlink:actuate" use="required"/>
                </xs:extension>
            </xs:simpleContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="Generlization">
        <xs:complexType>
            <xs:simpleContent>
                <xs:extension base="ST_Generlization">
                    <xs:attribute ref="xlink:type" use="required"/>
                    <xs:attribute ref="xlink:title" use="required"/>
                    <xs:attribute ref="xlink:show" use="required"/>
                    <xs:attribute ref="xlink:herf" use="required"/>
                    <xs:attribute ref="xlink:actuate" use="required"/>
                </xs:extension>
            </xs:simpleContent>
        </xs:complexType>
    <xs:element name="Dependency">
        <xs:complexType>
            <xs:simpleContent>
                <xs:extension base="ST_Dependency">
                    <xs:attribute ref="xlink:type" use="required"/>
                    <xs:attribute ref="xlink:title" use="required"/>
                    <xs:attribute ref="xlink:show" use="required"/>
                    <xs:attribute ref="xlink:herf" use="required"/>
                    <xs:attribute ref="xlink:actuate" use="required"/>
                </xs:extension>
```

```
            </xs:simpleContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="Association">
        <xs:complexType>
            <xs:simpleContent>
                <xs:extension base="ST_Association">
                    <xs:attribute ref="xlink:type" use="required"/>
                    <xs:attribute ref="xlink:title" use="required"/>
                    <xs:attribute ref="xlink:show" use="required"/>
                    <xs:attribute ref="xlink:herf" use="required"/>
                    <xs:attribute ref="xlink:actuate" use="required"/>
                </xs:extension>
            </xs:simpleContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="Abstraction">
        <xs:complexType>
            <xs:simpleContent>
                <xs:extension base="ST_Abstraction">
                    <xs:attribute ref="xlink:type" use="required"/>
                    <xs:attribute ref="xlink:title" use="required"/>
                    <xs:attribute ref="xlink:show" use="required"/>
                    <xs:attribute ref="xlink:herf" use="required"/>
                    <xs:attribute ref="xlink:actuate" use="required"/>
                </xs:extension>
            </xs:simpleContent>
        </xs:complexType>
    </xs:element>
    <xs:attribute name="type">
        <xs:simpleType>
            <xs:restriction base="xs:string"/>
        </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="title">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:enumeration value="Abstraction"/>
                <xs:enumeration value="Association"/>
                <xs:enumeration value="Dependency"/>
                <xs:enumeration value="Generlization"/>
```

```
                        <xs:enumeration value="Intergration"/>
                        <xs:enumeration value="Realization"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="show">
                <xs:simpleType>
                    <xs:restriction base="xs:string"/>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="herf">
                <xs:simpleType>
                    <xs:restriction base="xs:string"/>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="actuate">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="onRequest"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
        </xs:schema>
```

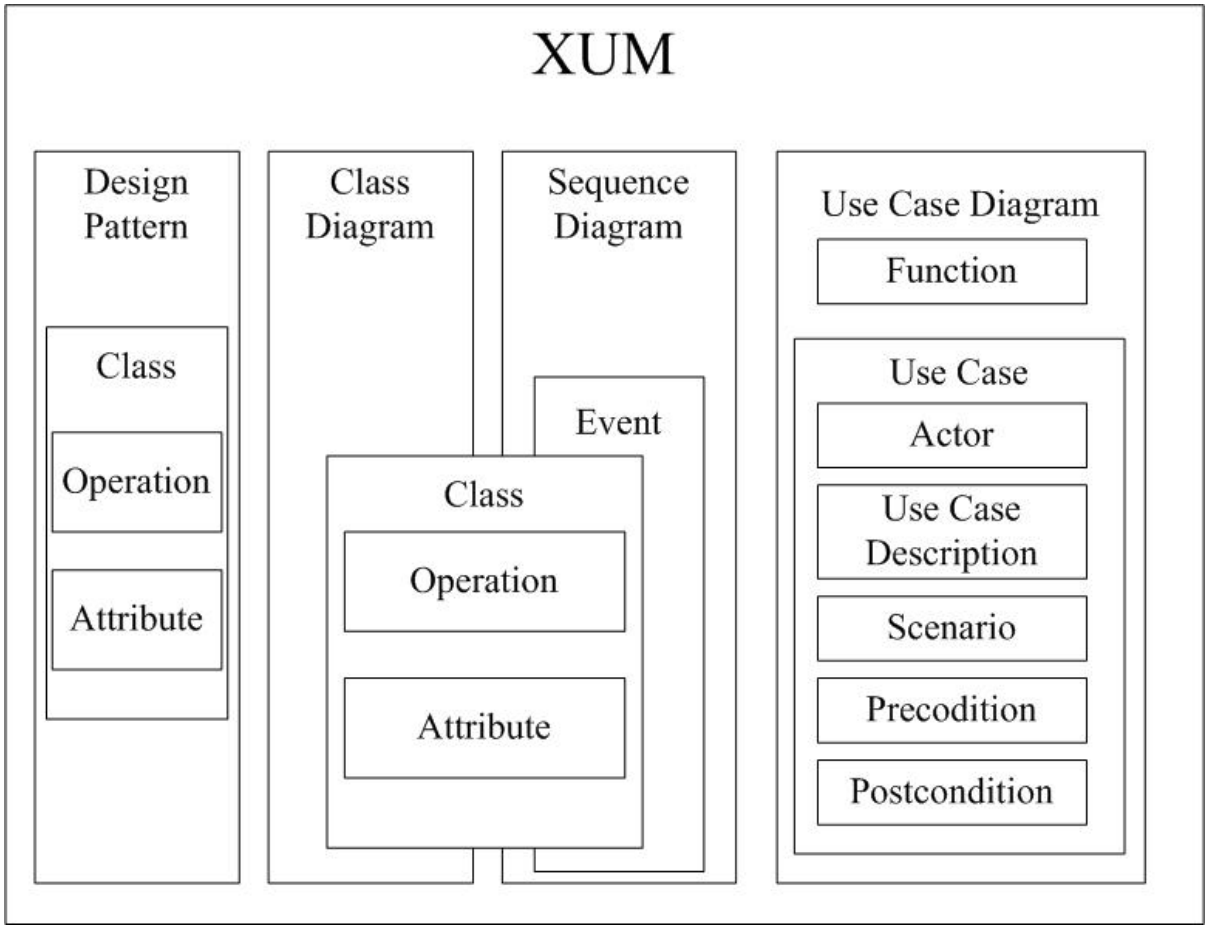

圖 二十六、XUM Schema Structure Model

4.9 XUM Integration Document

在先前階段我們將XUM Component與XML開發流程文件建立關聯，然而在本研究中我們亦將開發流程文件視為XUM Component之一，所以最後亦必須在文件之間建立文件整合鏈結以提供開發流程文件Overview的功能，文件整合鏈結程式碼如下表 十六：

表 十六、XUM Integration Document

```
<?xml version="1.0" encoding="UTF-8"?>
<XUMIntegration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink" xsi:noNamespaceSchemaLocation="I:\ＤＯＷＮＬＯＡＤ\Paper
相關\XMLSCH~1\XUM\XUMINT~1\XUMINT~1.XSD">

<DocumentIntegration xlink:herf="Use Case Diagram.xml" xlink:type="simple" xlink:show="embed"
xlink:actuate="onLoad" xlink:title="Integration">Use Case Diagram</DocumentIntegration>

<DocumentIntegration xlink:herf="Sequence Diagram.xml" xlink:type="simple" xlink:show="embed"
xlink:actuate="onLoad" xlink:title="Integration">Sequence Diagram</DocumentIntegration>

<DocumentIntegration xlink:herf="Class Diagram.xml" xlink:type="simple" xlink:show="embed"
xlink:actuate="onLoad" xlink:title="Integration">Class Diagram</DocumentIntegration>

<DocumentIntegration xlink:herf="Design Pattern.xml" xlink:type="simple" xlink:show="embed"
xlink:actuate="onLoad" xlink:title="Integration">Design Pattern</DocumentIntegration>

</XUMIntegration>
```

根據XUM Integration Document的建立，我們透過研究方法所提出的開發流程文件XML化的表示方式，將文件亦視為元件之一，並且可以將XUM概念當中用來做為文件轉換的XUMM透過檢視文件整合鏈結，將XUMM文

件作為使用者瀏覽系統開發流程中需求、設計、實作階段的Document Component內容，並且透過元件之間的關聯檢視變更過程中相關聯元件或者是欲追蹤的關聯元件，本研究當中XUM Integration Document、Document Component、Component以及Relation Link關聯模型如下圖 二十七所示：

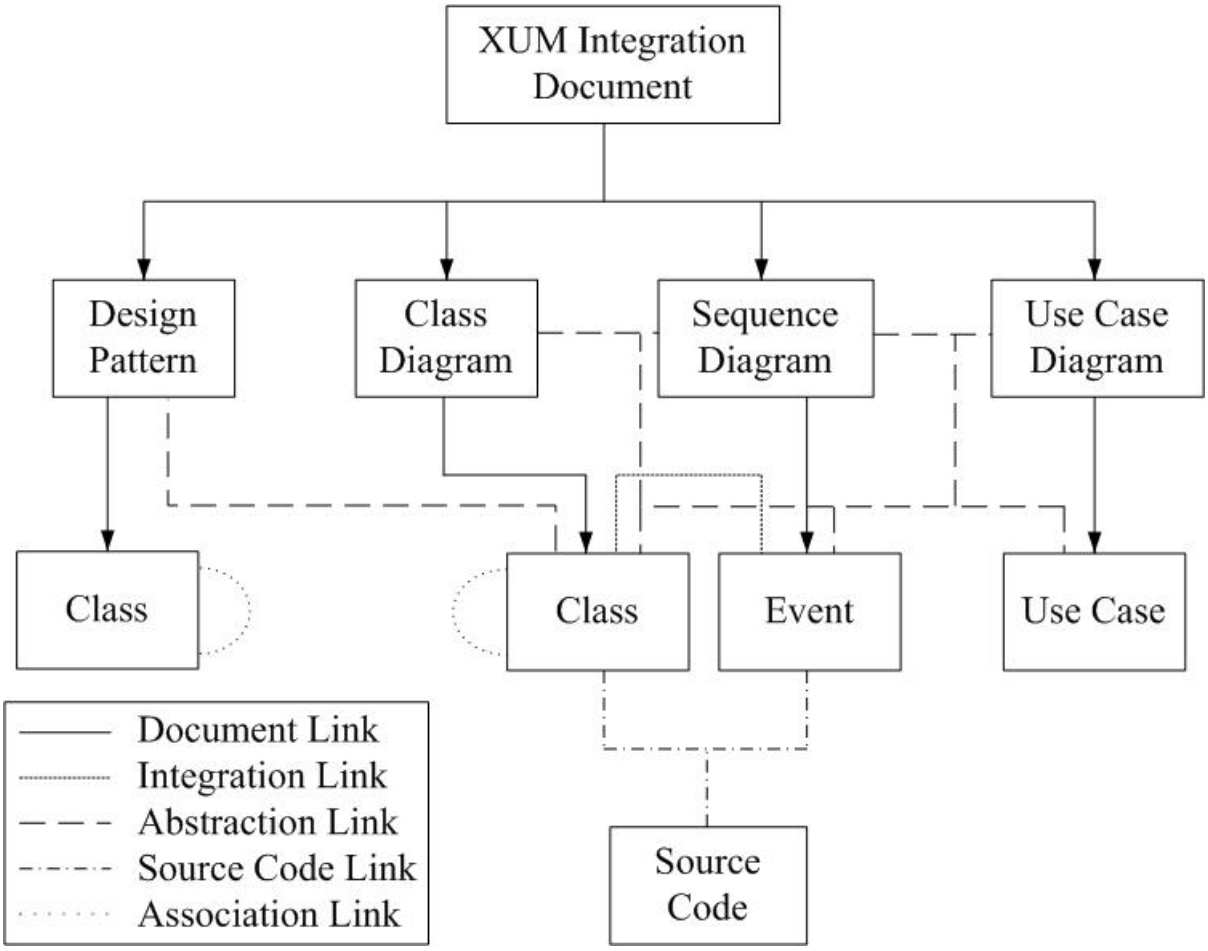

圖 二十七、XUM Integration Model

五、結論

軟體工程不斷提出新的標準以及流程改善，目的不外乎加強軟體工程的嚴謹性以及軟體開發的流暢性。然而不論統一流程的議題如何加強仍然無法完全克服軟體開發流程階段間的鴻溝，每個階段使用不同的標準產生便於使用的文件是必然的，提出整合不同標準、不同階段的文件的方法或許才是目前最有效率的解決方法。

模型化開發文件不但可以幫助管理文件內容，亦可幫助開發過程與開發文件做更有組織的對照，在本篇研究中，我們提出方法，用來輔助在系統開發時，減低開發文件上的不一致性，透過關聯性鏈結的建立可以提高階段間的銜接率，減少階段間的異質性與Miss Match的存在，除了可減少開發中文件不一致性問題，透過鏈結讓開發者能在軟體開發時，進行維護與追蹤。

XML為OMG提出為世界公認的標準文件格式，我們研究中利用XML技術將相關開發中文件進行串聯，最後產出共通使用的XML文件，在未來無論進行延伸開發研究或分析，將可利用本研究產出方向進行討論，亦可整合更多元的文件以及開發平台，提供較佳且完整的軟體工程需求變更追蹤系統以及開發工具。

六、參考文獻

[1]. Gamma E., Helm R., Johnson R. & Vlissides J., *Design Patterns : Elements of Reusable Object-Oriented Software*, Personal Education Taiwan, 2004

[2]. Carlson D., *Modeling XML Application With UML : Practical e-Business Applications*, Personal Education Taiwan, 2001

[3]. Fowler M., *UML Distilled 3rd Edition : A Brief Guide to the Standard Object Modeling Language*, Personal Education Taiwan, 2005

[4]. Sommerville I., *Software Engineering 7th Edition*, Personal Education Taiwan, 2004

[5]. Booch G., Rumbaugh J. & Jacobson I., *The Unified Modeling Language User Guide*, DrMaster, 2001

[6]. Object Management Group, "*OMG Unified Modeling Language Specification. Version 2.1.2*", 2007, from http://www.omg.org/spec/UML/2.1.2/

[7]. Coad P. & Yourdon E., *OOA-Object-Oriented Analysis 2nd Edition*, Prentice Hall, 1990

[8]. Kaindl H., "Difficulties in the Transition from OO Analysis to Design", *IEEE Software*, 1999, Vol. 16, No. 5, pp. 94-102

[9]. Salem A. M., Darter M. O. & Ramanujam B., "A Practical Method for Performing Object Oriented Requirement Analysis", *Carbon Sequestration in Terrestrial Ecosystems Conference Program*, 2004, Vol. 29, No. 2

[10]. Maarek Y. S., Berry D. M. & Kaiser G. E., "An Information Retrieval Approach for Automatically Constructing Software Libraries", *IEEE Transactions on Software Engineering*, 1991, Vol. 17, No. 8, pp. 800-813

[11]. Lu C. W., Chu C. C., Chang C. H., Yang D. L. & Lian W. D., "Integrating Diverse Paradigms in Evolution and Maintenance by an XML-Based Unified Model", *Journal of Software Maintenance : Research and Practice*, 2003, Vol. 15, No. 3, pp. 111-144

[12]. Arlow J. & Neustadt I., *UML 2 and The Unified Software Development Process 2nd Edition : Practical Object-Oriented Analysis and Design*, Addison-Wesley, 2005

[13]. Kruchten P., *The Rational Unified Process : An Introduction 3rd Edition*, Addison-Wesley, 2003

[14]. DeRose S., Maler E. & Orchard D., "*XML Linking Language (XLink) Version 1.0*", 2001, from http://www.w3.org/TR/xlink/

[15]. Holstege M. & Vedamuthu A. S., "*W3C XML Schema Definition Language (XSD) : Component Designators*", 2008, form http://www.w3.org/TR/2008/WD-xmlschema-ref-20081117/

[16]. Gao S. S., Sperberg C. M. & Thompson H. S., "*W3C XML Schema Definition Language (XSD) 1.1 Part 1 : Structures*", 2009, from http://www.w3.org/TR/xmlschema11-1/

[17]. Peterson D., Gao S S., Malhotra A., Sperberg C. M. & Thompson H. S., "*W3C XML Schema Definition Language (XSD) 1.1 Part 2 : Datatypes*", 2009, from http://www.w3.org/TR/2009/CR-xmlschema11-2-20090430/

[18]. Davis A. M. & Buckley F. J., "*IEEE Recommended Practice for Software Requirements Specifications*", The Institute of Electrical and Electronics Engineers, Inc 345 East 47th Street, New York, NY 10017, USA, 1998

[19]. Sherrell L. B. & Chen L. D., "The W Life Cycle Model and Associated Methodology for Corporate Web Site Development",

Communications of the Association for Information Systems, 2001, Vol. 5, Iss. 1, No. 7

[20]. Hassine J., Rilling J., Hewitt J. & Dssouli R., "Change Impact Analysis for Requirement Evolution Using Use Case Maps", *8th International Workshop on Principles of Software Evolution*, 2005, pp. 81-90

[21]. Benjamin A. & Westerberg A., "Anonymous Class in Declarative Mathematical Modeling", *The ASCEND BIBLIOGRAPHY*, 1998

[22]. Regnell B., Andersson M. & Bergstrand J., "A Hierarchical Use Case Model with Graphical Representation", *Proceedings IEEE Symposium and Workshop on Engineering of Computer-Based Systems*, 1996, pp. 270-277

[23]. Bray T., Paoli J., Sperberg C. M., Maler E. & Yergeau F., "*Extensible Markup Language (XML) 1.0 (5th Edition)*", 2008, from http://www.w3.org/TR/2008/REC-xml-20081126/

[24]. Stark G. E. & Oman P. W., "Software Maintenance Management Strategies : Observations from the Field", *Journal of Software Maintenance and Evolution : Research and Practice Software Focus*, 1997, Vol. 9, Iss. 6, pp. 365-378

[25]. Bosak J., Bray T., Connolly D., Maler E., Nicol G., Sperberg C. M., Wood L. & Clark J., "*Guide to the W3C XML Specification ("XMLspec") DTD, Version 2.1*", 2000, from http://www.w3.org/XML/1998/06/xmlspec-report-v21.htm

[26]. Bray T., Paoli J., Sperberg C. M., Maler E. Yergeau F. & Cowan J., "*Extensible Markup Language (XML) 1.1 (2nd Edition)*", 2006, from http://www.w3.org/TR/xml11/