摘要

由於軟體開發過程與軟體本身的複雜程度日趨提高，與軟體系統相關的各種資訊，如需求規格、分析模型與設計模型等，分散在各種不同的呈現形式中。軟體系統開發者需要能及時地追蹤與擷取出軟體系統相關資訊，以期正確地進行系統開發作業，而一個具可追蹤性的軟體系統可為軟體開發過程帶來許多益處，如提供更有效率的系統相關資訊管理或是提昇軟體系統的可維護性[18]，另外，一個缺乏可追蹤性的軟體系統往往導致軟體工程作業的高成本與系統的低可維護性。

許多軟體標準被提出並建議導入軟體開發，以提高軟體開發效率與系統品質。如 UML 被提出作為軟體模型的標準呈現語言，軟體設計樣板(DESIGN PATTERN)被提出作為系統架構設計的參考等…在 XML-based Unified Model [1]的理論裡提出一個方法用以整合目前廣被接受的軟體模型標準於一個基礎於 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。

在此研究中，我們基礎於此理論方法並以實作一個軟體系統相關文件的追蹤與管理的雛型工具系統，此雛型工具的功能包括讓使用者以樹狀層級結構建立文件的追蹤模型、建立文件分類規則、文件搜尋、與文件內容差異性比對的應用。並結合 XUM 的 UML 塑模工具，能從文件內容追蹤擷取出以此工具編製的 UML 模型，以期能提高文件追蹤管理的自動化程度與可維護性。

關鍵字：追蹤性，軟體可維護性，XML

Abstract

Due to the grow of complexity of software development, and software information is distributed in different representation such as requirement specification, design model and analysis model. Developers need to capture traces [19, 20] between software information in these different representations in order to perform engineering activities correctly. Besides, a traceable software system model can brings benefits to software engineering, such as efficiency software information management and improving software maintainability [18], and a software system lack of traceability usually lead to high engineering operation cost and inefficient maintenance.

Software standards are highly recommended because they promise faster and more efficient ways for software development with proven techniques and standard notations, such as UML is used to express software model and design pattern is used in software architecture design …An XML-based unified model (XUM)[1] is proposed to improve both software development and maintenance through unification and integration of software standards.

In this paper, we implement a prototypical tool based on the XUM, to support establishing the trace model by a tree-from structure, imformation searching and retieving, and content differencing. We integrated this tool with XUM modeling tool, so that users can retrieve UML model from document content edited by our tool in this research, to improve software maintainability and software information trace and retireive.

Key words: Traceability, Software maintainability, XML

章　節　目　錄

中文摘要 1

英文摘要 2

章節目錄 3

圖目錄 4

第一章　序論 7

1.1 **前言** 7

1.2 **研究動機** 7

1.3 **研究方法** 7

1.4 **章節安排** 8

第二章　背景知識及相關研究 9

2.1 **相關研究** 10

2.2 **背景知識** XML-based Unified Model 10

第三章　建立軟體模型追蹤性的方法 12

3.1 **方法概觀** 12

3.2 **定義追蹤模型以作為軟體模型的分類規則** 12

3.3 **註冊軟體模型物件為追蹤模型之分類物件** 14

3.4 **以搜尋與物件內容檢索機制建立追蹤性** 15

第四章　雛形系統實做 16

4.1 **設計與實作目標** 16

4.1.1 **支援使用者定義與管理追蹤模型** 16

4.1.2 **提昇軟體模型物件的註冊作業自動化程度** 16

4.1.3 **軟體模型物件的追蹤** 16

4.2 **雛形系統架構** 17

4.2.1 **統一模型資料儲存** 17

4.2.2 **統一模型內容樹** 20

4.3 **整合模型控制模組** 19

4.3.1 **基礎功能模組** 19

4.3.2 **追蹤模型建製模組** 20

4.3.3 **軟體物件追蹤模組** 21

4.4 **應用功能模組** 23

4.4.1 **標準化文件編輯與內文追蹤資訊擷取** 23

4.4.2 **文件內容之差異性比較** 24

第五章 應用範例 25

5.1 **應用案例-專案文件管理作業** 25

5.2 **建置追蹤模型** 25

5.3 **註冊文件** 26

5.4 **文件追蹤** 28

5.5 **內文檢索與** UML **模型追蹤** 31

5.6 **文件內容之差異性比對** 33

第六章 結論與未來工作 34

參考文獻 36

圖 目 錄

圖 1 XUM 中軟體模型、關連與整合連結間的關係示意 10
圖 2 建立軟體模型追蹤性的概念示意 12
圖 3 一個追蹤模型範例 : XUM 定義的軟體模型類別與關連性與 XUM 呈現 13
圖 4 一個類別關連性的呈現範例 14
圖 5 DTD 中的模型物件定義 15
圖 6 雛形工具之系統架構示意 17
圖 7 內容樹與 XML 檔案間的內容轉換 18
圖 8 XML 資料內容產生 BIND CLASS 的過程 18
圖 9 以 XUM 內容樹所呈現的追蹤模型結構 20
圖 10 一個模型物件的註冊資訊範例 21
圖 11 自動分類規則的 XUM 呈現範例 21
圖 12 Visitor 的走訪機制 22
圖 13 軟體模型物件的瀏覽與紀錄追蹤路徑系統執行範例 23
圖 14 應用案例的追蹤模型 25
圖 15 建置應用案例的追蹤模型 26
圖 16 依據檔名特徵建立中英文文件分類規則 27
圖 17 文件註冊完成後的追蹤模型 27
圖 18 以關鍵字搜尋文件名稱 28
圖 19 點選文件與相關聯文件的瀏覽模式 29
圖 20 系統自動紀錄使用者的文件追蹤過程 30
圖 21 追蹤結果顯示與檔案下載 31
圖 22 利用系統編寫文字文件的系統執行範例 32
圖 23 由文件內容的模型物件名稱建立追蹤聯結並檢視該模型物件 32

圖 24 **新舊版本文件內容的差異比對之系統執行範例** 33

第一章 序論

1.1 介紹

隨著軟體應用的普遍化，軟體服務與應用的需求隨著增加。使用者對於軟體服務的提供速度與系統品質的要求逐漸提高。在目前已有越來越多的軟體系統被開發完成且上線使用，新的軟體服務可能是利用舊的架構設計，甚至是重整舊系統。因此必須對舊系統的設計或需求進行追蹤，以擷取出可所需要的資訊，而這些對舊系統的維護作業往往花費筆當初開發的成本高出許多 [21]。許多不同型式的軟體相關資訊隨著開發過程不斷的被產生，並已不同的型式被呈現，如需求規格、或是以 UML 呈現的系統架構模型等…系統維護者在維護作業時仍需能夠擷取出這些資料做為參考，因此系統開發者為這些資料建立追蹤性與相關管理作業是基本且必要的。

追蹤性，是指兩個以上的軟體開發過程產出物之間可被建立的關連性，特別是產出物間具有原始-繼承者或從屬關係的產出物[18]。追蹤性關連可被用來輔助需求驗證、變異管理、了解開發產出物的演進過程。

1.2 研究動機

許多軟體標準被提出以提高軟體開發效率如 UML(Unified Modeling Language)[5] and XML(eXtensible Markup Modeling Language)[6，7] 提供了軟體系統模型圖形化呈現的與資料格式的標準。但是軟體系統的相關資訊是以各種不同的型式呈現著，使用者需要的資訊可能散佈在不同的文件中，然而，為這些軟體系統相關資訊間的關連建立起追蹤與管理機制仍然是相當缺乏[3,6]。

1.3 研究方法

我們先前的研究 XML-based Unified Model [1]的理論裡提出一個方法用以整合目前廣被接受的軟體模型標準於一個基礎於 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。本論文基礎於此理論方法實做一個雛形工具系統，功能包括讓使用者以樹狀層級結構建立文件的追蹤模型、建立文件分類規則、文件搜尋、與文件內容差異性比對的應用。並結合 XUM 的 UML 塑模工具結合，能從文件內容中追蹤擷取出以此塑模工具編製的 UML 模型，以期能提高文件追蹤管理的自動化程度與可維護性。

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。第三章則是建立軟體模型追蹤性的理論方法。第四章說明了此雛形工具的系統設計。第五章安排了一個使用情節用以說明本雛形工具系統如何落實理論，達成自動化、可追蹤性以提昇軟體資訊追蹤作業效率及軟體可維護性。而結論、未來工作則描述於第六章節。

第二章 背景知識與相關研究

2.1 相關研究

目前已在此研究領域中被提出研究與工具，多數的研究只針對特定開發階段如需求工程階段或分析設計階段的資訊追蹤，而提供跨階段追蹤的工具則強調特定的軟體系統模型進行追蹤，如針對在不同開發流程階段的需求規格。以下簡介兩個與本研究相關的研究:

Traceability of Object-Oriented Requirements (**TOORs**) [11] 是一個做為需求追蹤的方法與工具，它的目的在於系統開發的過程中支援需求追蹤，可讓使用者自訂系統模型間的關連性，並可從需求規格到、設計規格與程式碼中進行追蹤。然而 TOOR 的研究只對需求工程相關的特定資訊做追蹤管理，並未對其他的系統相關資訊支援。本研究的實做工具整合符合 UML 標準的軟體模型，包括需求、設計與實做階段的軟體模型呈現，以利於使用者追蹤更全面的軟體模型資訊。

Requirement Engineering Through Hypertext (**RETH**)[13] 是一個支援需求工程的理論與方法，其理論基礎於物件導向與 Hypertext，以自然語言描述資訊並以正規化的節點(node)與 Hypertext 呈現關連性。RETH 提供了一個追蹤資訊模型提供了重整工程對需求追蹤所需的資訊架構，然而 RETH 的使用者無法對此追蹤模型做修改。本研究以資訊的分類與關連作為追蹤模型的基礎，使用者可彈性的定義與修改，已定義的追蹤模型可被匯出以做其他專案的再利用與易於調整的特性。

2.2 背景知識

2.2.1 XML-based Unified Model

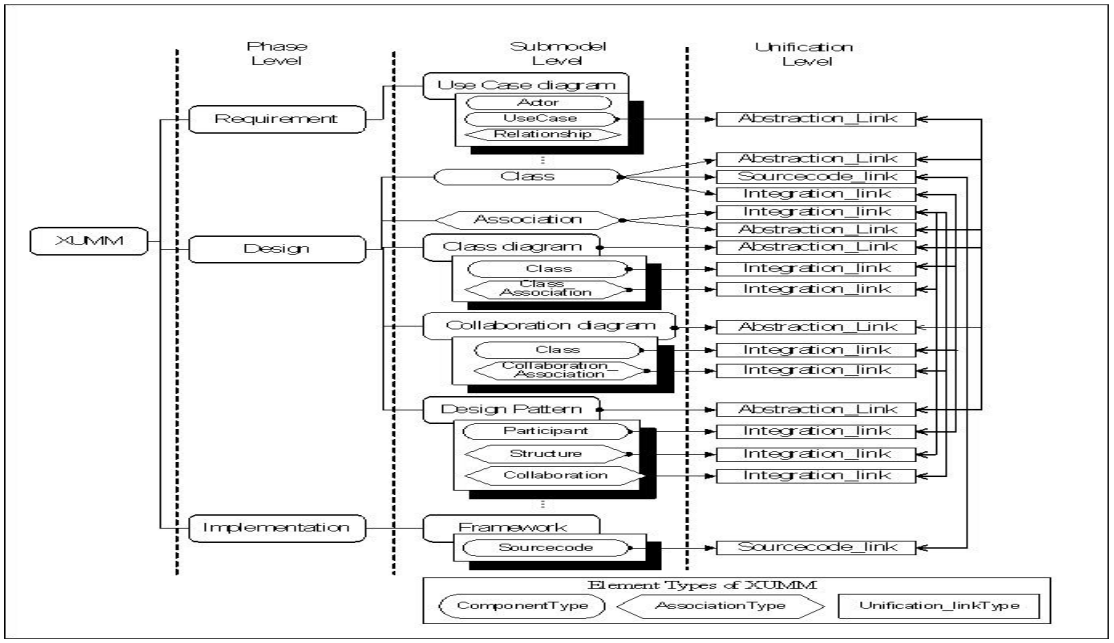

圖 1. XUM **中軟體模型、關連與整合連結間的關係示意**

XUM (XML-based Unified Model) **理論的目的在於整合軟體系統資訊於一個基礎於** XML **呈現的整合模型，而** XUMM(XML-based Unified Meta-Model) **則是用來定義** XUM **中的資料型態。在** XUM **中，我們以**"SUB-MODEL"**呈現以不同標準表示的軟體系統模型, 每個** SUB-MODEL **根據** XUMM **的定義轉換成定應的** XML **格式，此以** XML **型式呈現的系統模型我們稱為一個**"VIEW"**。**XUM **的概念可分為兩部分，一個是來自各個軟體開發階段的軟體模型，包括需求、設計與實做階段。另一部分是是模型間的整合關連。**XUM **中個部分的相互關連如圖一所示。**XUMM **與** XUM **間的關係如同** DTD **與** XML **間的關係。**XUM **實體上由三個部分組成, 軟體模型、關連與整合連結三部份, 例如** UML **中的** USE CASE **與** CLASS **是** XUM **中的元件, 而** USE CASE **中定義著**"USE"**的關係即屬於** XUM **的關連。而整合關連則是** XUM **用來整合所有** XUM **中的模型，它包含以下三種連結**

1) **抽象連結**

抽象連結用來連結 XUM **中不同抽象程度的軟體模型，如需求階段的** USE CASE **與設計階段的** CLASS DIAGRAM **間的關連。**

2) 整合連結

在 XUM **中，同一個物件可能出現在不同的** VIEW **中，整合連結即是用來連藉這些在不同** VIEW **出現的相同物件。**

3) 程式碼連結

程式碼連藉是 XUM **用來連結模型與相對應的程式碼**

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。**

第三章 建立軟體模型追蹤性的方法

3.1 方法概觀

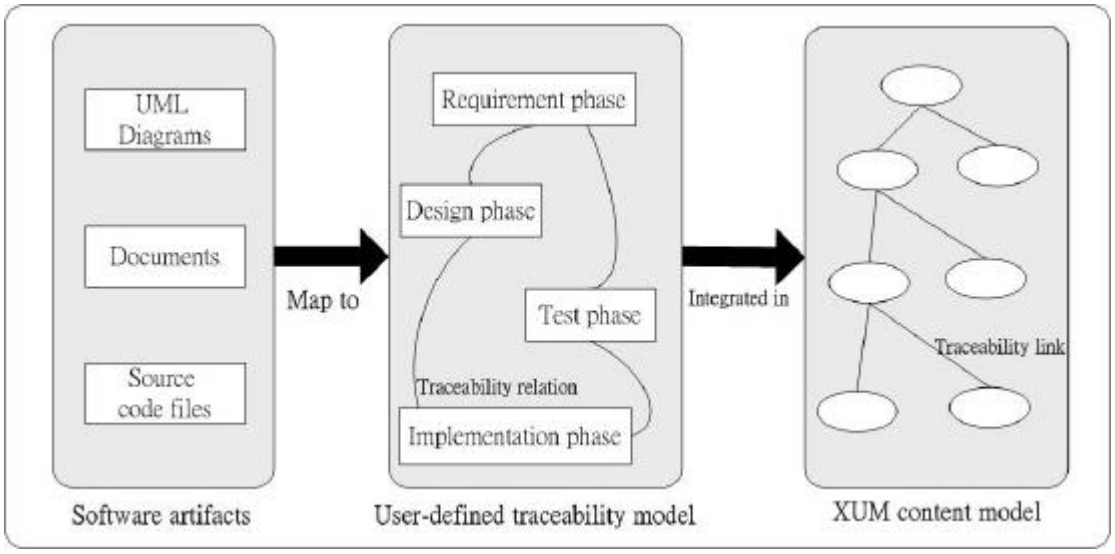

圖 2 建立軟體模型追蹤性的概念示意

圖二呈現本研究實做之工具建立追蹤模型方法概觀。可簡單分為兩個階段。在第一個階段中，使用者在工具中需定義符合專案或是產品需求的追蹤模型，此追蹤模型是由多個分類類別與類別間的關連所組成，使用者可依照需求自定這些類別與關連性。追蹤模型建立完成後，把軟體模型的各種不同呈現，如 UML DIAGRAM，程式碼檔案或是相關文件註冊於分類別之下，類似將檔案移動到資料夾中的動作。第二階段是使用者如何應用工具來追蹤模型中的資訊，在本研究實做的雛形工具中將提供幾個追蹤機制以輔助使用者追蹤資訊。本雛形工具並整合我們先前實做的 UML 塑模工具[1]，提供使用者追蹤與檢視 UML 標準模型的機制，詳細的功能實做與應用說明會在以下的章節呈現。

3.2 定義追蹤模型以作為軟體模型的分類規則

軟體開發的過程中，可能隨者開發系統的性質不同，開發過程中產生的軟體

系統模型也隨著不同，甚至同一個系統的模型或是文件，在不同背景的使用者的使用觀點上(VIEW)，需要不同的呈現方式。因此提供使用者自訂追蹤模型是本研究認為有利於使用者建立符合特定需求之追蹤模型的方式。而一個追蹤模型呈現的是使用者觀點上軟體模型類別之間的相互關係，因此在我們的工具中，即是以提供使用者建立模型類別與類別關連的方式建立追蹤模型。

圖 3 呈現一個追蹤模型的範例與追蹤模型在本雛形工具的內部呈現。此追蹤模型範例是我們先前的研究中所提出的，它包含 Design、Analysis 與 Implementation 三個軟體開發的層面，每個層面定義若干個類別。在本研究中，我們僅以此模型做為範例說明雛形系統如何呈現一個追蹤模型。圖 3 中左側區塊

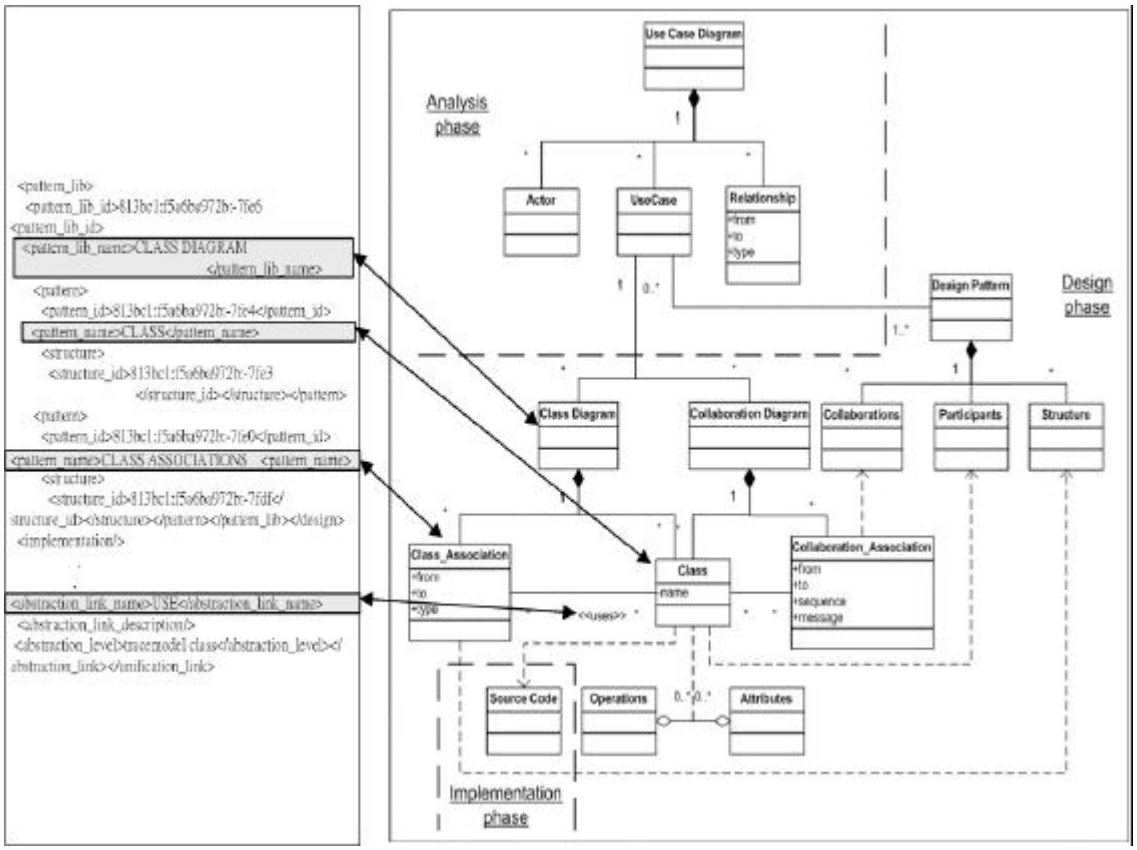

圖 3 一個追蹤模型範例 ： XUM 定義的軟體模型類別與關連性與 XUM 呈現

為軟體模型類別與其相互關係與此模型在 XUM 中的呈現。由圖可見我們將模型中的類別以個別的 XML TAG 呈現並包含屬性資料，而類別間的關連性則以 XUM 的抽象連結呈現。抽象連結的屬性資料包含來源類別、目標類別與關連性名稱，這是因為 XUM 抽象連結適用於連結不同抽象程度的模型類別。

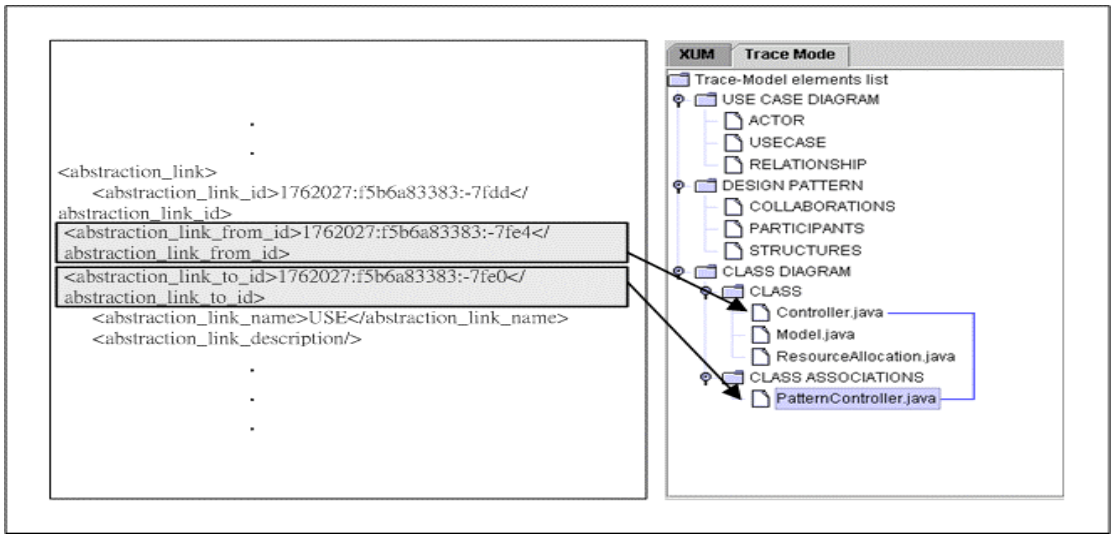

圖 4 一個類別關連性的呈現範例

圖 4 的範例呈現一個抽象連結的 XUM 呈現，XML 標籤包含的資訊有連結的來源類別與目標類別的系統識別碼，右側的系統執行畫面呈現出在兩類別中若有關連性的存在，則雛形工具以一條藍線連接表示。

我們用圖 3 與圖 4 的範例說明了我們如何以 XML 呈現一個追蹤模型，包括模型中的類別與關連性。接下來的章節將說明軟體模型物件的註冊資訊在本雛形工具中如何被呈現。

3.3 註冊軟體模型物件為追蹤模型之分類物件

註冊在系統下的軟體模型物件可以是以本系統編寫的 UML 模型、或是需以其他程式開啟的外部檔案。為了使物件的註冊資訊能夠提供有利於使用者追蹤的資訊，註冊資訊中除了包含系統內部程式邏輯所需的資訊外，有需包含能表達對使用者有意義的屬性資訊。基於上述目的我們規劃的註冊資訊如下頁圖 5 的 XML DTD 所示，包括建立日期、版本、建立者、物件狀態、檔案路徑與描述等資訊。在系統內部資訊上，我們定義 "ARTIFACT*"的元素，用來表示物件可包含多個其他物件，以作為系統內部呈現物件群組的目的。

```
<!ELEMENT_artifact
(artifact_id,artifact_name,artifact_path,artifact_version,artifact_description,artifact_status,
register_date,artifact_creator,class_code,xum_internal,attribute*,artifact*) >
<!ELEMENT artifact_id (#PCDATA) >
<!ELEMENT artifact_name (#PCDATA) >
<!ELEMENT artifact_path (#PCDATA) >
<!ELEMENT artifact_version (#PCDATA) >
<!ELEMENT artifact_description (#PCDATA) >
<!ELEMENT artifact_status (#PCDATA) >
<!ELEMENT register_date (#PCDATA) >
<!ELEMENT artifact_creator (#PCDATA) >
<!ELEMENT xum_internal (#PCDATA) >
```

圖 5 DTD 中的模型物件定義

3.4 以搜尋與物件內容檢索機制建立追蹤性

本工具以兩種追蹤功能達到資訊追蹤的目的。當使用者不知道所需物件的分類位置時，可使用搜尋與過濾的方法，工具可以物件名稱或是搜尋物件描述的內容找出所需的物件。而當使用者需要的是不同類別下的多個關連物件時，工具已紀錄"追蹤路徑"的經驗法則達到追蹤效果。工具會先根據類別關連性條列出與某項物件相關的所有物件，使用者點選其中一項物件時，工具將此物件列入追蹤路徑，當使用者找出所有關連物件後，可將此追蹤路徑紀錄於工具中，在以後的使用中也能做為追蹤參考。

關於本雛形工具如何建立追蹤性，我們將在下一個章節中，以系統架構的角度說明，並同時說明雛形工具的實做架構。

第四章 雛形系統實做

4.1 設計與實作目標

我們以實做了一個雛形系統以實現我們的研究，此雛形系統基礎於我們先前的研究成果，XUMM CASE TOOL[1]，以建立追蹤性機制與提昇軟體可維護性為需求加強系統功能。本研究實做的雛形系統的目標說明如下：

4.1.1 支援使用者定義與管理追蹤模型

為了讓不同背景的使用者能建立符合使用觀點的追蹤模型，與滿足不同專案或作業需求的使用者，系統需提供支援可讓使用者彈性自訂追蹤模型，以圖形化的操作介面協助使用者建置模型與基本管理功能如修改及儲存成 XML 格式的檔案等。

4.1.2 提昇軟體模型物件的註冊作業自動化程度

將軟體模型物件註冊於系統中是項頻繁且高度重複性的作業，往往造成使用者不必要的負擔。系統為協助使用者處理繁瑣的模型物件註冊作業，我們以半自動化的方式，讓模型物件能根據使用者設定的分類規則自動分類，以提高註冊作業的自動化程度。

4.1.3 軟體模型物件的追蹤

本系統的實做目的之一即為協助使用者在大量的物件資料中追蹤出所需的資訊，系統提供搜尋過濾與追蹤路徑的模型物件群組化機制，協助使用者在大量的物件中追朔出所需的資訊。本系統並將提供自動由文件內容擷取追蹤資訊與文件內容差異比對的應用功能，以期能帶給使用者實務上的幫助。

4.2 雛形系統架構

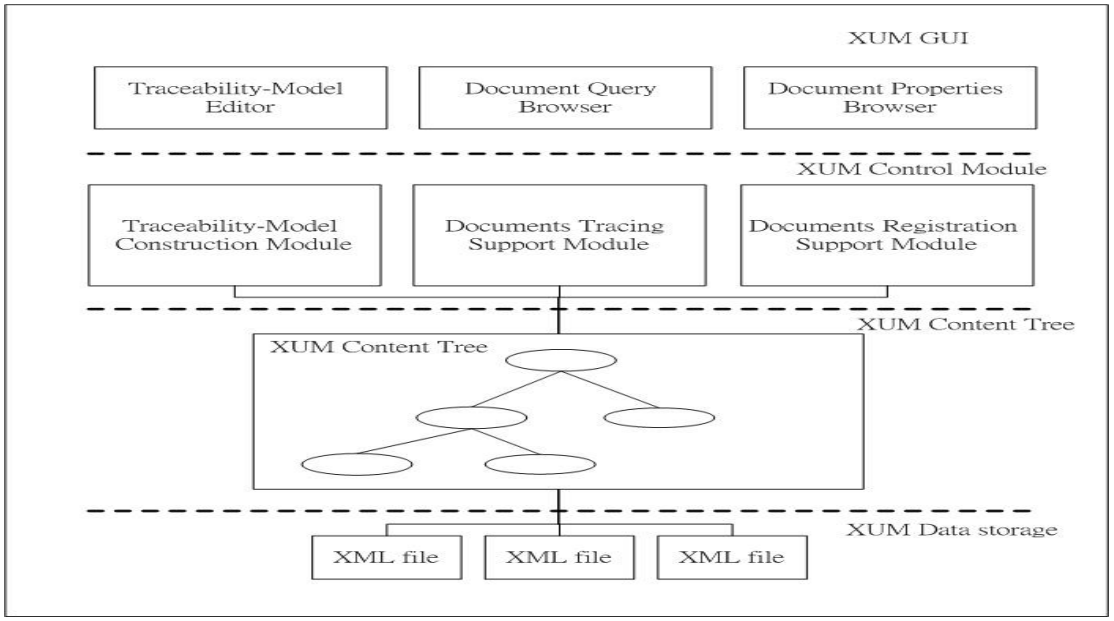

圖 6 雛形工具之系統架構示意

圖 6 呈現本雛形系統的架構，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)、統一型內容樹(XUM content tree)及統一模型資料儲存。以下我們對系統架構各層級以分別的章節進行說明：

4.2.1 統一模型資料儲存

XUM DATA STORAGE 代表本系統儲存實體資料的層級，系統所儲存的資料皆存以 XML 格式檔案，包括 XML DTD 檔、追蹤模型的 XUM 呈現、模型物件的註冊資料與系統 UI 的設定檔。這些資料都將對應到系統的 XUM CONTENT TREE 中。而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 Java Architectural for XML Binder (JAXB)所提供的功能可以將 Java Object 的 Content Tree 與 XML 之間作轉換的動作。圖 7 為 JAXB 轉換 XML 資料與 XUM 內容樹節點資料的示意圖，JAXB 透過 UNMARSHAL 與 MARSHAL

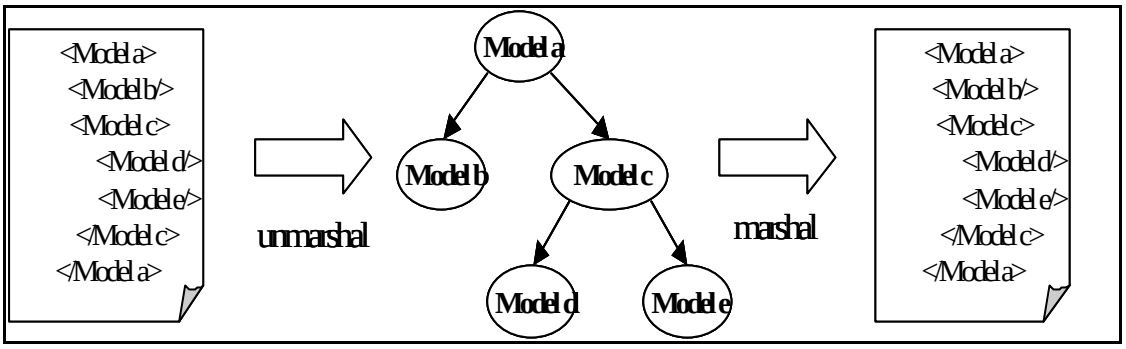

圖 7 內容樹與 XML 檔案間的內容轉換

指令進行檔案與系統資料格式間的轉換動作。圖中的 MODEL 標籤即代表一個 XML 資料元素，JAXB 將其轉換為一個 JAVA CLASS，再以 XUM 內容樹呈現這些 XML 資料元素的關係結構。

4.2.2 統一模型內容樹

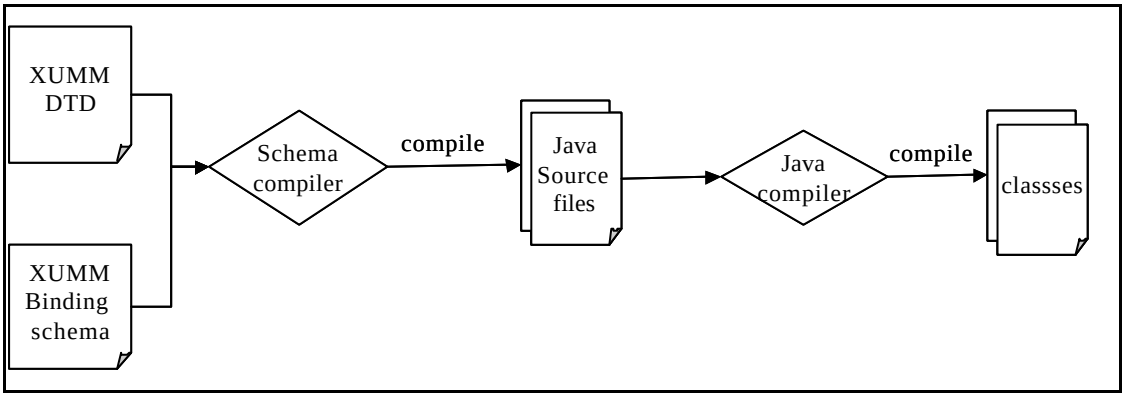

圖 8 XML 資料內容產生 BIND CLASS 的過程

圖 8 說明 XML 資料元素轉換為 XUM 內容樹的 JAVA 物件的轉換機制。本系統以一個樹結構作為主要的資料結構。主要的原因是我們採用 XML 檔案作為系統的統一呈現格式，且系統實做上採用 JAXB 技術。JAXB 技術使系統可直接將 XML DTD 定義的資料元素直接轉換為 JAVA CLASS，如此一來系統可以直接的操作 CLASS 的方式存取 XML 元素，可省去繁雜的 XML 剖析作業。除此之外、追蹤模型中定義的類別與關連，可以對應成一個階層式的分類架構，因此一個樹

狀的資料結構非常適合呈現這樣的階層架構, 且使用者也可直接以管理樹狀結構的方式進行追蹤作業。

4.3 整合模型控制模組

XUM 控制模組即是本系統的核心功能模組，包含基礎控制與功能模組以達到功能需求，以下我們對每個模組詳細說明：

4.3.1 基礎功能模組

本系統基礎於我們先前的研究成果, 利用已建置的基礎功能模組做為系統的底層功能架構。包括內容樹管理模組、潛在關連擷取機制、與統一關聯機制，以下說明這些模組的實做方法。

1) 內容樹管理模組

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，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。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，及下一段所介紹的資訊走訪機制。

2) 資訊擷取機制

接下來我們必須能對以 COMPOSITE 設計樣板建立出的樹狀結構進行資訊的取用。我們以 VISITOR 設計樣板來解決 COMPOSITE 節點包含多樣類型資

訊的問題，系統可利用不同的 VISITOR 以根據不同參數設定的方式針對不同節點執行特定動作，以此機制解決將來新增節點類別的需求。

3) 統一關聯機制

本系統在實做上應用 XUM 理論的抽象鏈結(Abstraction link) 及程式碼鏈結(Source code link)，XUM 抽象連結應用在呈現分類類別間的關連性，而程式碼連結則應用於呈現分類物件間的追蹤連結。

4.3.2 追蹤模型建製模組

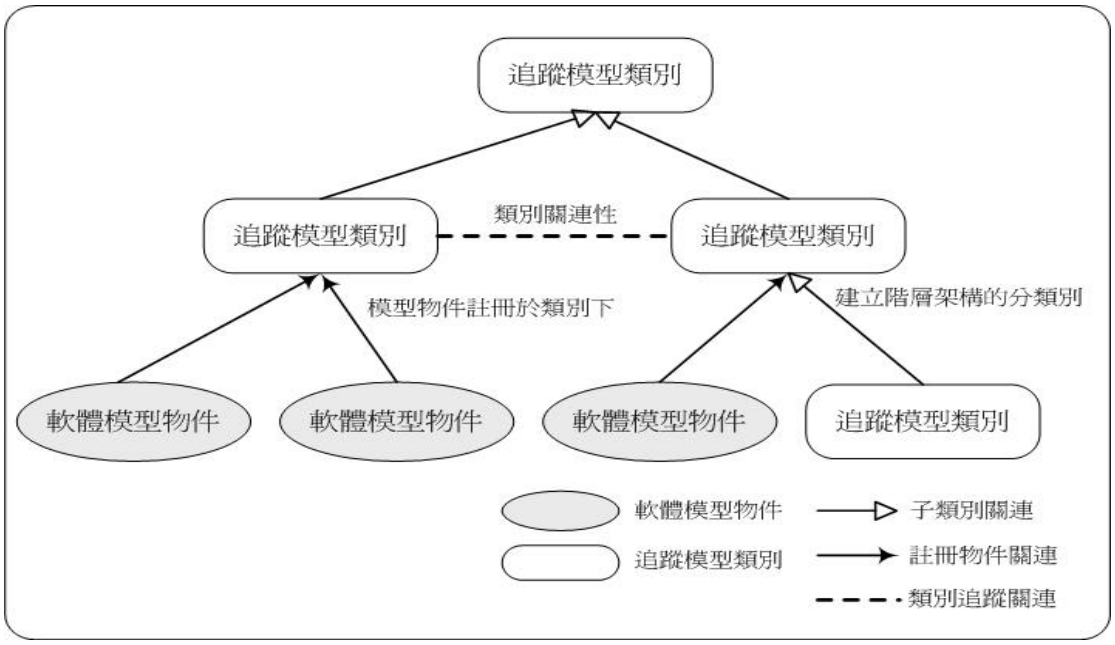

圖 9 以 XUM 內容樹所呈現的追蹤模型結構

本系統提供使用者建立模型類別與關連的方式建立追蹤類別，我們在 XML DTD 中定義類別的資料結構，包括名稱、分類條件、類別關連性與模型物件，其中分類條件包含條件與動作。圖 9 表示追蹤模型在 XUM 內容樹中的呈現結構。使用可以名稱建立一個模型類別，並類別下建立子類別，使用者可建立多階層的分類架構。在建立關連性上，使用者需選擇兩個模型類別分別為來源與目標類別，並給定一個關連性名稱即可建立起兩個類別之間的關連。類別之間的關連代表者模

型物件之間的關連性意義，而註冊在類別下的模型物件會繼承此關連性，當系統提供使用者追蹤參考物件時，使用者即是以此關連性做追蹤的參考與依據。

4.3.3 軟體模型物件註冊模組

```
<pattern>
        <pattern_id>813bc1:f5a6ba972b:-7ffa</pattern_id>
        <pattern_name>USECASE</pattern_name>
        <artifact>
          <artifact_id>14b2f1a:f60d1f52d2:-8000</artifact_id>
          <artifact_name>USECASE UC111</artifact_name>
          <artifact_path>C:\java\xumm\USECASE UC111</artifact_path>
          <artifact_version>none</artifact_version>
          <artifact_description>none</artifact_description>
          <artifact_status>none</artifact_status>
         <register_date>Jun 28 14:35:11 2003</register_date>
          <artifact_creator>none</artifact_creator>
```

圖 10 一個模型物件的註冊資訊範例

圖 10 一個模型物件的註冊資訊範例，由圖可見定義中除了屬性外，並以”<PATTERN>”表示物件所屬的模型類別。

為了減低使用者重複物件註冊作業的負擔，本工具以建立分類規則達到物件自動分類的功能，使用者在類別下註冊多筆的物件資料時，系統根據使用者在類別中設定的分類規則將模型物件分配製特定的子類別中。例如使用者可在父類別中設定分類規則，範例中使用者設定物件名稱中若含’xum’字串則自動註冊到某個子類別下，當使用者選擇依據此分類規則註冊模型物件時，系統及判斷檔案名稱是否包含’xum’字串，如果包含則登記模型物件在指定子類別下。本工具希望利用此機制協助使用者在重複註冊物件作業時可提高作業效率。

```
<pattern_lib>
    <pattern_lib_id>1762027:f5b6a83383:-8000</pattern_lib_id>
   <pattern_lib_name>USE CASE DIAGRAM</pattern_lib_name>
    <sort_contract>
      <sort_contract_id>1aa0e3b:f5b6ebd71d:-8000</sort_contract_id>
    <sort_contract_name>If artifact title match XUM move to ACTOR</sort_contract_name>
      <sort_contract_condition>XUM</sort_contract_condition>
      <sort_contract_action>ACTOR</sort_contract_action></sort_contract>
```

圖 11 自動分類規則的 XUM **呈現範例**

圖 11 **呈現是分類規則的** XUM **呈現，由內容可見我們以**<SORT_CONTRACT>**標籤呈現一個分類規則，內容包含**<SORT_CONTRACT_CONDITION>**標籤紀錄符合分類規則的條件字串**"XUM"**，以**<SORT_CONTRACT_ACTION>**標籤紀錄復合分類規則所執行的動作，即註冊在** ACTIOR **類別。並可由**<PATTERN_LIB>**標籤得知此分類規則屬於**"USE CASE DIAGRAM"**父類別的分類規則。**

4.3.3 軟體物件追蹤模組

系統以搜尋過濾和追蹤路徑兩種功能提供使用者追蹤軟體系統資訊。圖 12 **所示為系統所使用的** visitor **機制架構。系統利用** visitor **機制先深後廣的走訪分類架構的每個節點，我們利用此機制達成資訊擷取與搜尋的功能。在資訊擷取方面，我們利用樹狀結構圖呈現追蹤模型與模型物件，以** visitor **機制走訪每個有關連性的分類別，將分類別下的註冊物件呈現於'關連物件列表'中，使用者可從關連物件列表中點選下一個追蹤物件。當使用者點選下一個追蹤物件時，系統將新點選的物件加入'追蹤路徑物件表'中，以此紀錄使用者追蹤模型資訊的過程，當使用者欲刪除目前點選的物件時，可在追蹤路徑物件表追點選先前加入的物件**

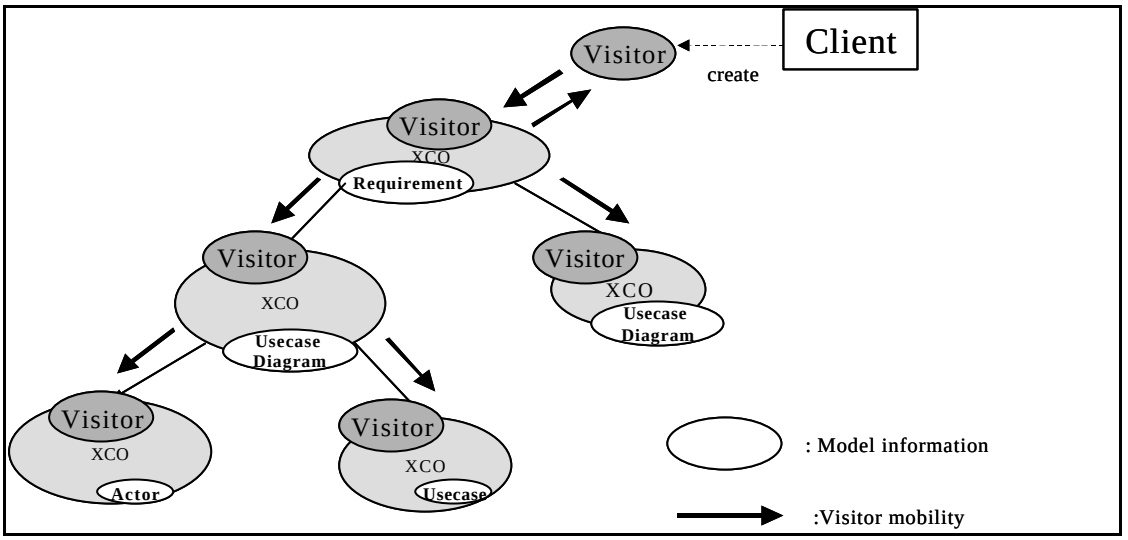

圖 12 Visitor 的走訪機制

系統將會回到該點選物件的瀏覽畫面，追蹤路徑物件表也隨者以該物件為最終的追蹤節點。利用追蹤路徑功能，使用者可將追縱路徑紀錄為特定的模型物件群組，例如一份詳細的需求規格資訊可能包括需求規格、評估分析甚至設計測試的相關文件作為參考，物件群組的設計可以滿足使用者在類似此類的需求。系統紀錄下追蹤路徑後也可在其他使用者進行資訊追蹤時，作為提示參考。如圖 13 所示，系統瀏覽模式之執行畫面，右上方即是追蹤路徑的物件列表，下方的物件列表即是與目前瀏覽物件相關聯的物件列。

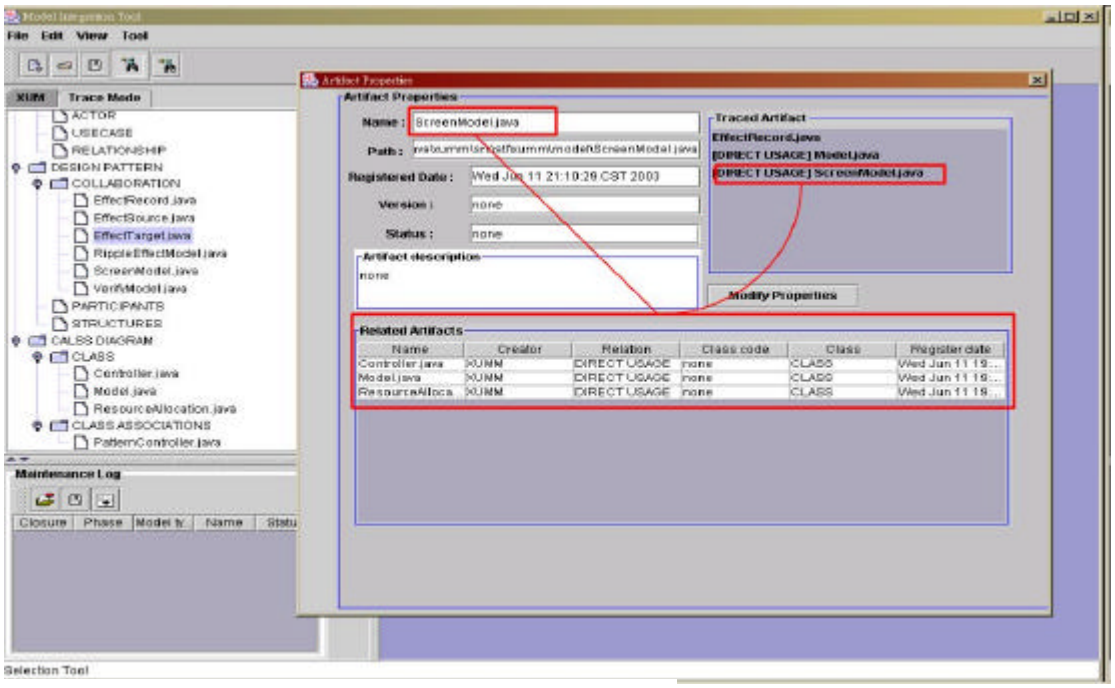

圖 13 軟體模型物件的瀏覽與紀錄追蹤路徑系統執行範例

4.4 應用功能模組

本系統並利用以上章節介紹的系統架構與基礎功能模組實做出以下的應用功能模組。

4.4.1 標準化文件編輯與內文追蹤資訊擷取

本系統提供文件編輯的功能模組提供使用者編輯以經過標準化的文字文件，我們將文件內容分項儲存為 XML 物件，系統可對這些文件的內容執行包括內容檢索與差異性比對的功能，並藉由內文檢索擷取出潛在的追蹤資訊。本系統以"VISITOR"機制比對 XUM 軟體模型物件的名稱與文件內容，如果比對到相同的物件名稱在文件內容中出現，則將該文件列為追蹤參考。例如本系統提供使用者編輯測試報告文件，若使用者在文件內容中寫有某項 USE CASE 的名稱，則使用者利用瀏覽模式檢視該測試報告的登錄資訊時，本系統即會對該測試報告內容盡行內文檢索，若比對出有相同名稱的 USE CASE 名稱，則使用者可直接利用本系統的 USE CASE 編輯器進行檢視。使用者可利用此機制由文件內容追蹤以本系統編製的 UML 模型，本系統利用此功能提供使用者能由文件內容追蹤到 UML 標準模型的機制。

4.4.2 文件內容之差異性比較

本系統並提供一般文字文件的內容差異性比較，使用者可於瀏覽模式選擇一項欲比對的文件，系統將讀取檔案內容，並比對出兩個檔案的差異處，例如新檔案所新增的部分內容，與呈現出新增內容在舊檔案的位置。使用者可利用此功能檢視新舊版本檔案間的內容差別，以追蹤出一份文件的發展過程。

以下的章節中，我們將以一個實際的應用案例並配合系統的執行畫面範例，以說明本系統如何建立實際的追蹤模型，與使用者如何以系統功能實際執行文件追蹤作業。

第五章 應用範例

在本章節中，我們將導入一個實際的應用案例-專案文件管理作業。本章節先簡介此應用案例，接著利用系統中的各項功能規劃出追蹤模型，並將案例中的資訊架構套用於系統的追蹤模型下，最後應用系統的追蹤模組的應用功能。

5.1 應用案例-專案文件管理作業

此案例的目的在分類與管理專案相關的文件資料, 所包含的文件類別包含產品規格、功能規格、測試報告、系統索引與使用手冊。這些不同類別的文件同屬一個專案，而在實際的作業環境中，這些文件已經過簡單的分類，因此第一步我們根據分類的方式設計為追蹤模型並套入工具中。

5.2 建置追蹤模型

圖 14 是我們根據實際狀況所規劃的追蹤模型，產品規格共有三個子類別，完整功能規格、初步規格與產品規格。另外有測試報告類別，我們將它規劃與產

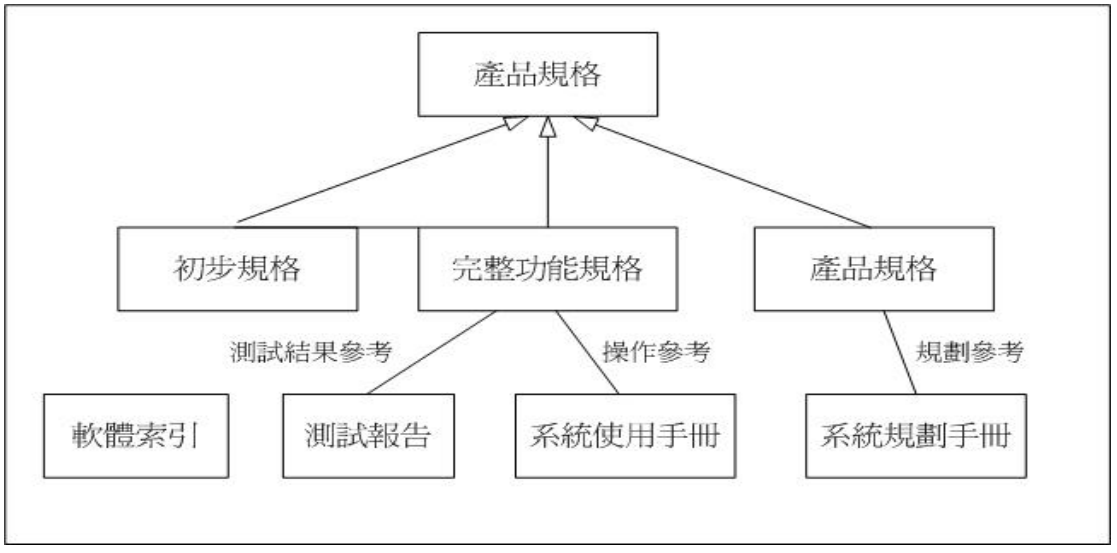

圖 14 應用案例的追蹤模型

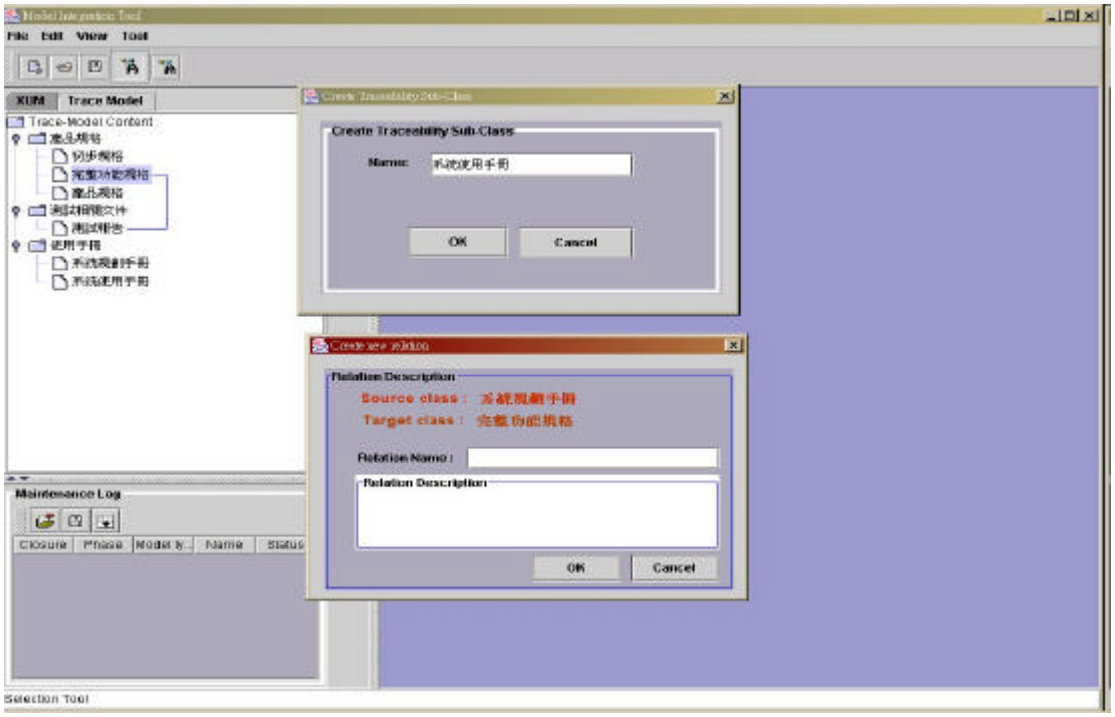

圖 15 建置應用案例的追蹤模型

品規格有"測試結果參考"關連。"使用手冊"類別則與產品規格有"操作參考"關連。根據此追蹤模型我們在工具中規劃的系統執行畫面如圖 15 所示，使用者指定類別名稱後即可建立類別, 而關連性的建立需選擇目標與來源類別後指定關連性。

5.3 註冊文件

在進行文件註冊時我們發現該產品有中英文兩種不同語言版本的產品規格書，且檔案命名時以"EN"與"TW"字串區別，因此我們在產品規格類別中加入依據"EN"字串分類到"產品規格 (英文)"類別的分類規則、與依據"TW"字串分類到"產品規格 (中文)"類別的分類規則。如下頁圖 16 所示。如此一來使用者可在分類類別中選擇"依分類規則新增模型物件"的功能，系統即會根據規則內容自動將物件註冊到符合規則描述的子類別下。如下頁圖 17 的(A)視窗所呈現，使用者可利用檔案瀏覽視窗選擇多筆檔案，由系統自動利用規則分類模型物件，以提昇作業效率。

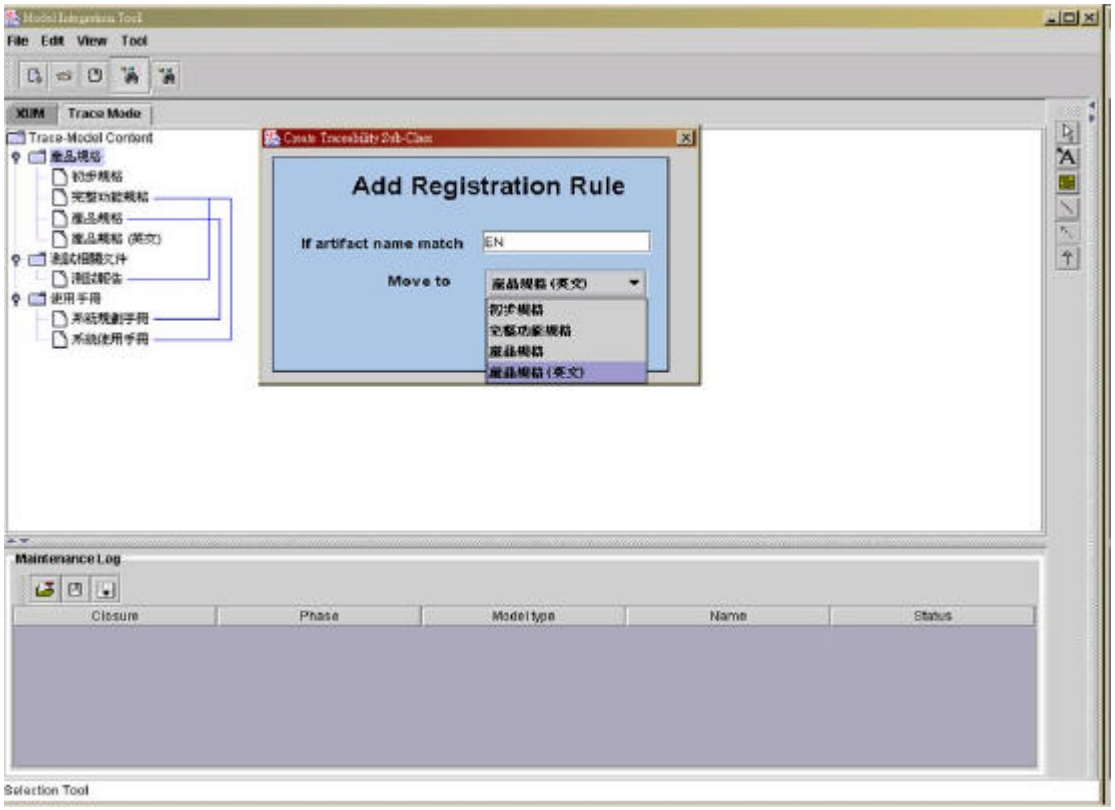

圖 16 依據檔名特徵建立中英文文件分類規則

圖 17(B)呈現的文件註冊的系統執行範例與註冊後的系統內容樹。由圖可見模型類別與模型物件以樹狀結構呈現，提供使用者簡單而直觀的圖形化呈現。接下來我們套用一個使用情節說明本工具如何達成協助使用者追蹤資訊。

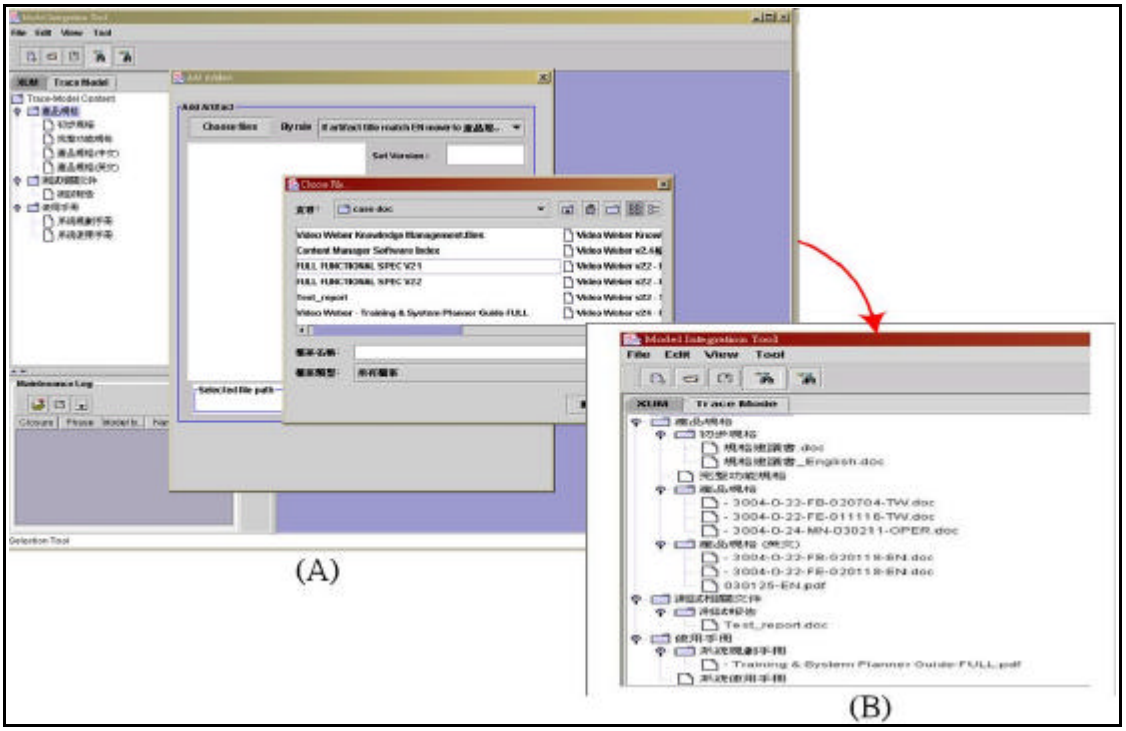

圖 17 文件註冊完成後的追蹤模型

5.4 文件追蹤

當使用者需要準備一份齊全的產品相關資訊時，通常包括產品規格、功能規格、測試報告與使用手冊。為了幫助使用者快速的蒐集完整的資訊，使用者可利用工具提供的搜尋功能。本系統利用關鍵字搜尋的機制，可依文件名稱，描述等屬性進行檢索。如下圖 18 所示，使用者以關鍵字”FB”搜尋文件名稱，結果顯示目前註冊物件中有兩筆符合的資訊，並分別屬於英文版本類別與中文版本類別，並可顯示相關的屬性資訊，如建立日期與狀態等。在此例中即可發現，若使用者不熟悉文件的命名與分類方式，類別關連性的呈現與搜尋的機制有利於使用者節省多餘的查看與詢問時間。

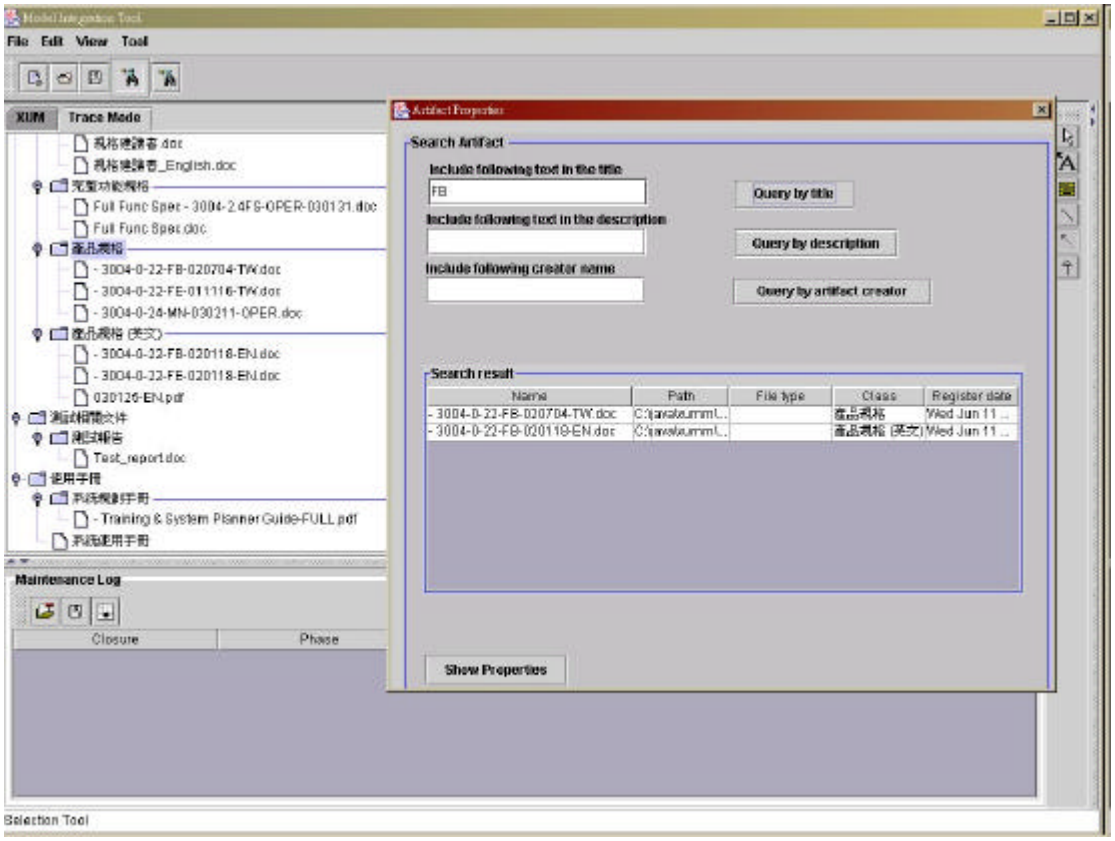

圖 18 以關鍵字搜尋文件名稱

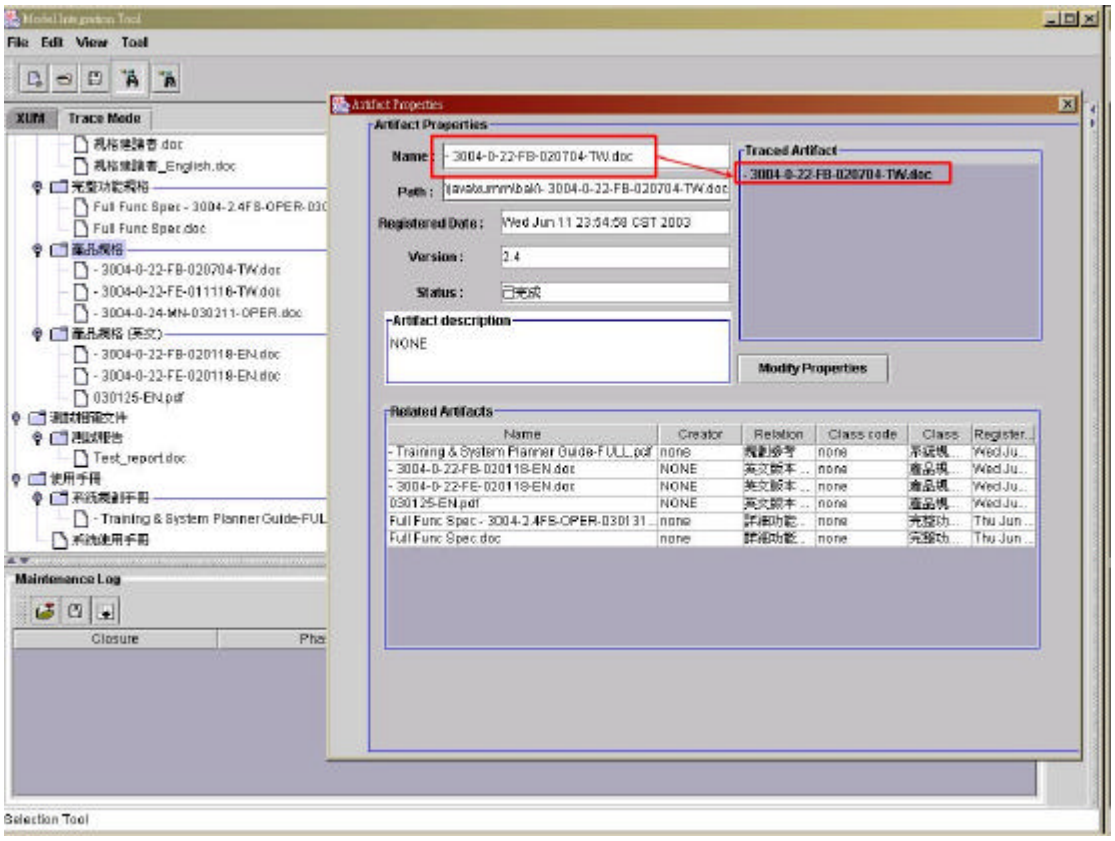

圖 19 點選文件與相關聯文件的瀏覽模式

使用者此刻可點選列表中的文件來得到更進一步的資訊，可得到圖 19 **所示的文件瀏覽模式畫面。瀏覽模式所呈現的包括左上區塊的目前點選文件屬性資料，圖** 19 **中即左側方框所示，目前選擇的是**" - 3004-0-22-FB-020704-TW.doc"**文件。而紅色箭頭所指向的右側紅色方框則是工具提供的追蹤路徑功能，它紀錄使用者瀏覽過的文件。下方列表所呈現的是與目前瀏覽的物件有關連性的其他模型物件。列表中的文件可能包括使用者利用本系統編寫的** UML **模型或是文字文件，也有可能只是註冊在追蹤模型類別下的外部文件。若是本系統可編輯的內部文件，則使用者可在此畫面呼叫編輯程式即時進行編寫。若只是外部文件，則使用者可使用下載功能，系統在提供使用者另存該檔案於其他目錄的視窗介面。我們會在下一個執行範例中呈現上述的下載功能與編寫模式。在上頁圖** 19 **的物件瀏覽模式中，使用者點選下方的關連文件時，即得到圖** 20 **的系統畫面，進入下一個模型物件的瀏覽模式。**

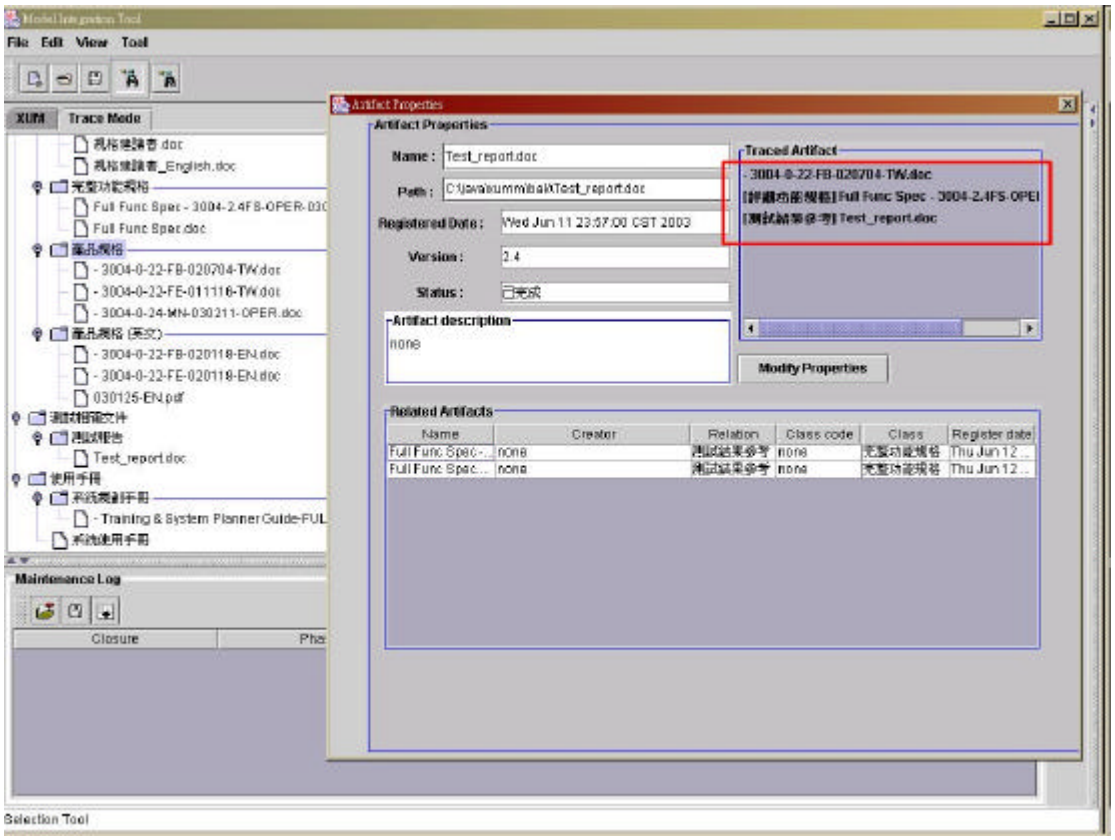

圖 20 系統自動紀錄使用者的文件追蹤過程

紅色方框顯示了使用者陸續追蹤了功能規格與測試報告，因此追蹤路徑列已記錄了這三項文件。追蹤路經的呈現方式除了檔案名稱外也顯示出文件與上一筆文件間的關連性，如此一來有利於使用者能從追蹤路徑列得知整個追蹤過程的邏輯，而不會忘記先前追蹤某份文件的原因。而當使用者認為追蹤路徑中的某份文件不符合需求時，可點選追蹤路徑列中的前一筆資料，追蹤路徑列即回到不符合文件的前一筆追蹤文件，提供使用者可修正追蹤路徑的功能。待使用者追蹤出所有需要的文件後，即可進行文件下載的動作。如下頁圖 21 所示，系統將追蹤路徑中所有的文件列出讓使用者下載到其他目錄。

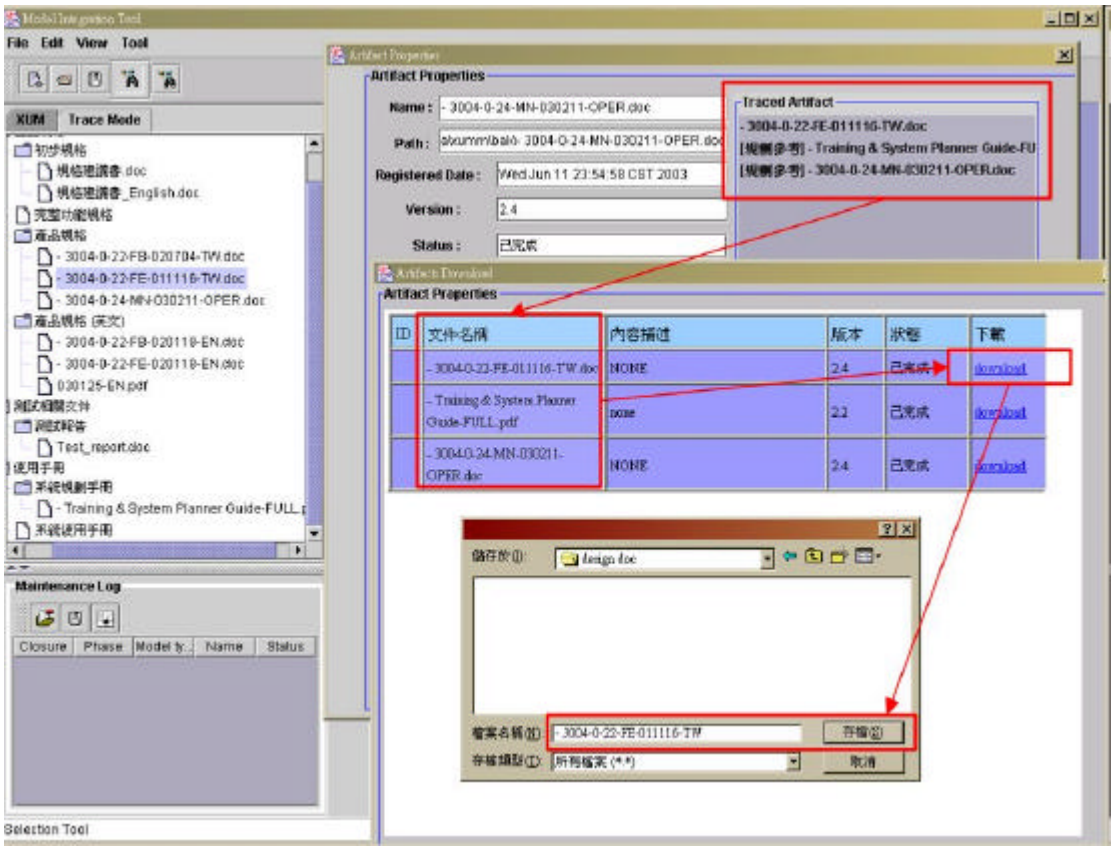

圖 21 追蹤結果顯示與檔案下載

5.5 內文檢索與 UML 模型追蹤

由於許多軟體模型之間的關連性被呈現於文件內容中，因此本系統設計了以模型物件名稱搜尋文字文件內容的機制，以期能在文字文件內容中擷取出與其他文件間潛在的關連性。圖 22 是本系統編寫一份測試報告文件的執行範例，本工具希望能提供使用者標準化樣板文件的編輯功能，若使用者以系統提供的文件樣板編寫文件，則系統能對使用者編寫的文件內容做追蹤資訊的擷取，達到更全面的資訊追蹤。而下頁圖 23 則呈現此測試報告的瀏覽模式。由圖可見在使用者在文件摘要內容中提到”usecase uc121” 的物件名稱，系統即將目前使用者編寫的 UML 模型物件逐一做名稱的檢索。結果搜尋到有相同名稱的 USECASE 名稱，即顯示”usecase uc121”為關連物件，並呈現於關連物件列表中。使用者可點選”usecase uc121” 物件則可得到該 usecase 內容編輯視窗。本系統利用此機制搜尋出文件內容中的追蹤資訊以提供使用者參考。

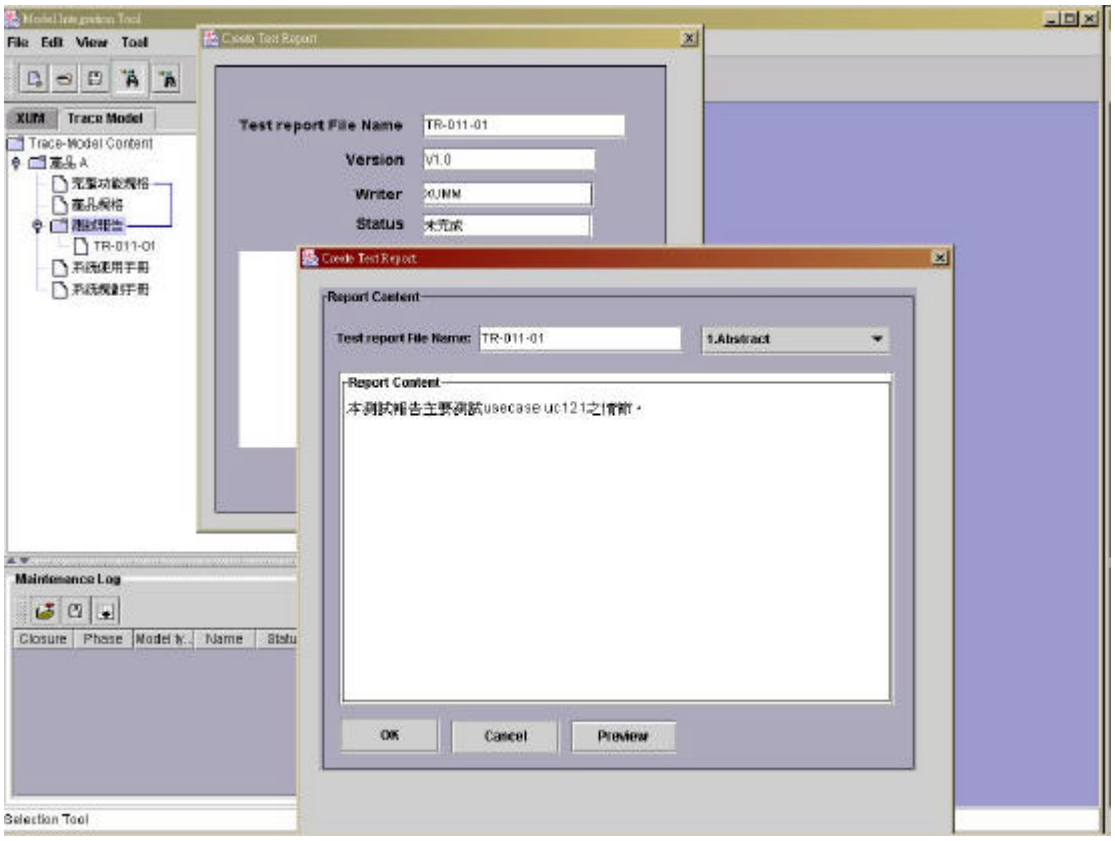

圖 22 利用系統編寫文字文件的系統執行範例

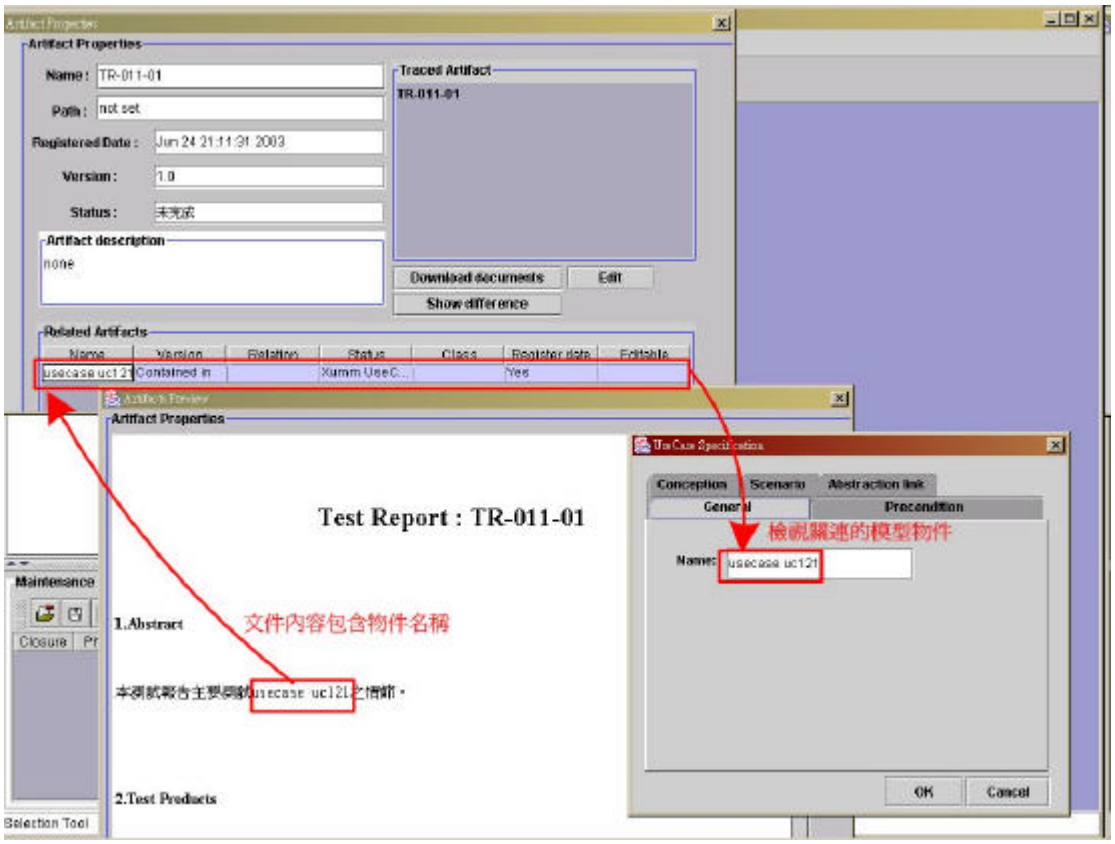

圖 23 由文件內容的模型物件名稱建立追蹤聯結並檢視該模型物件

5.6 文件內容之差異性比對

下頁圖 24 是本系統內容差異性比對的執行範例。在此範例中，我們以本系統不同版本的系統操作介面設定檔作為範例，由執行畫面中系統以紅色標示舊檔案被刪除的部分內容，以綠色的底色呈現經過修改過的部分內容。藉此使用者可快速的比較出舊版本文件的內容差異，有利於節省人工校對的時間耗費以提高作業效率。

由上述的應用案例可見，使用者可使用此雛形工具對專案或產品的相關文件進行分類管理，並透過分類規則提高作業效率，以搜尋功能找出需要的檔案，並利用系統自動的內文檢索擷取出文件內容中包含的 UML 模型圖名稱，並可開啟該 UML 模型圖進行詳細的檢視。使用者在追蹤尋找文件的過程可由系統自動紀錄下，使用者可將此追蹤紀錄設定為文件群組，以作為未來檔案擷取的參考。最後若使用者得到兩份不同版本的檔案，可使用差異性比對的功能擷取出新舊檔案間的內容差異，有利於使用者追蹤出同一份文件的演進歷程。

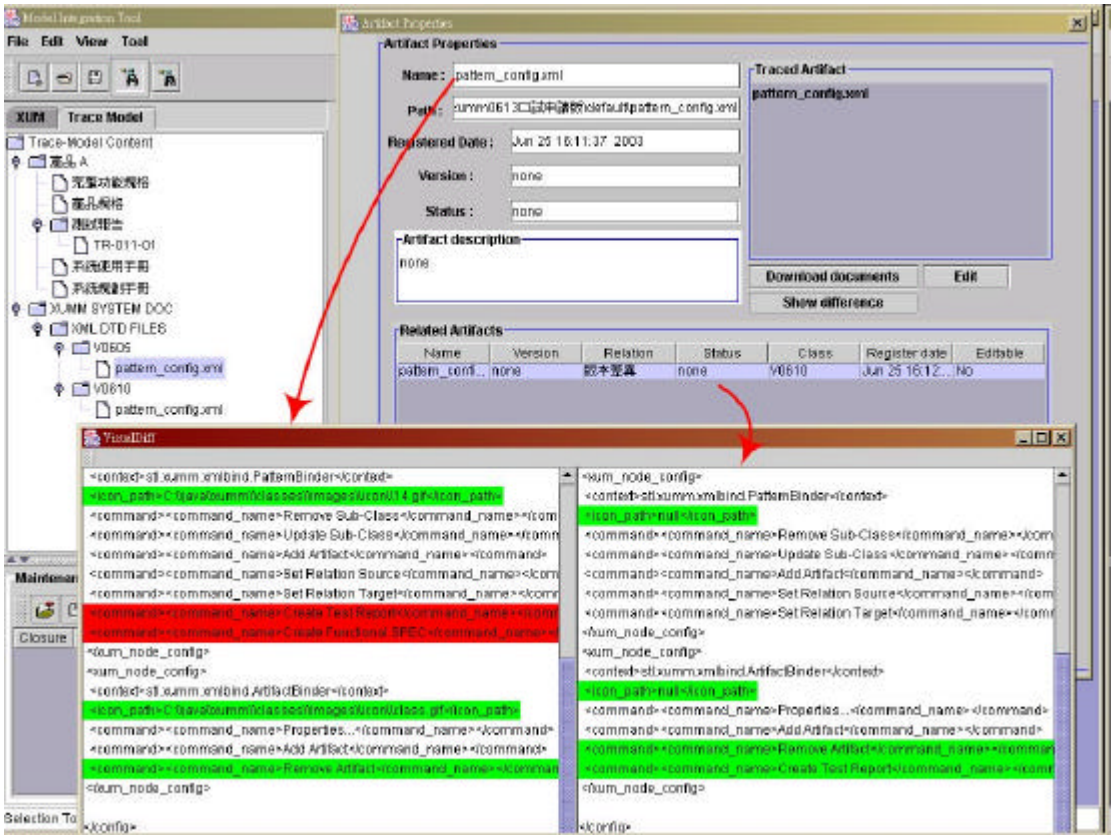

圖 24 新舊版本文件內容的差異比對之系統執行範例

第六章 結論與未來工作

本研究進行的動機在於了解到軟體開發複雜的過程中產生許多的系統相關資訊，而這些資訊的可追蹤性對於軟體維護或是再利用都有顯著的幫助，但是為這些資訊建立追蹤性卻是繁瑣的作業，導致真正可追蹤的系統資訊相當少。

本研究基礎於ＸＵＭ理論實做一個軟體資訊追蹤模型的建製與追蹤雛形系統，並以提高自動化程度，彈性的追蹤模型建立與追蹤機制來提昇軟體系統資訊的可追蹤性與可維護性。

在提高自動化程度方面與彈性建立追蹤模型方面，透過ＸＭＬ結構化的資訊呈現特性，系統可以提供使用者自行建立追蹤模型中的類別與關連，與管理模型的內容，建立好的追蹤模型也可輕易的以ＸＭＬ檔案匯出與匯入，彈性的滿足不同使用者對系統資訊的需求。另外並透過註冊規則的設置，幫助使用者在進行模型物件註冊作業時更有效率，進而提高作業的自動化程度。而樹狀結構的圖形化呈現有利於使用者更容易了解整個資訊架構。

在資訊追蹤時常見的物件群組問題上，我們以追蹤路徑機制，紀錄使用者追蹤文件的過程，以達成物件群組化的需求。另外，藉由ＸＵＭ整合不同標準模型的特性與連結技術，使用者可追蹤到各開發階段軟體模型的詳細資料。在潛在關連擷取上，我們提供由文件內容擷取出潛在的模型物件追蹤關連的功能，提供使用者從文件內容追蹤到 UML 的標準軟體模型或其他文件。另外並新舊文件內容差異性比對的功能，可協助使用者追蹤出新舊檔案的差異，以提高作業效率。

在未來的發展上，ＸＵＭ的研究與工具軟體可朝以下幾個方向發展：

1） 軟體再利用

在軟體開發過程中導入軟體在利用的特性將可以提昇軟體生產效率，包含程式碼與系統架構的再利用。如利用模型追蹤機制的建立，再加上ＸＵＭ整合

各階段標準模型的特性，此方向直得深入研究。

2) 軟體組態管理

軟體組態管理也是提高開發效率的要點之一，更包含了許多軟體開發流程的基本作業，如版本控制、多人協同作業、整合工作流程作業等。另外，軟體組態管理也是 CMMI 的 KPA 之一，值得本研究的工具軟體擴充發展的方向。

3) 軟體模型物件的正規化

目前的系統工具雖可根據分類規則自動分類註冊物件，但仍可透過模型物件的正規化描述，更進一步的提昇自動化程度，是工具軟體可在擴充的方向。

本雛型工具現階段的功能主要應用在提升軟體系統維護作業的效率與建立系統資訊的可追蹤性。藉由上述的未來工作，系統可建立更完整的軟體工程工具功能，以期提昇整體的工程效率。

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] XML: UXF approach. In the *Proceedings on UML'98: Beyond the Notation - International Workshop*, 65-74.

[3] Grundy, J.C., Hosking, J.G., Mugridge, W.B. Supporting inconsistency management for multiple-view software development environments, *IEEE Transactions on Software Engineering, vol. 24, no. 11*, November 1998.

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

[5] 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

[6] Gerald Ebner, Hermann Kaindl, "Tracing All Around in Reengineering" *IEEE SOFTWARE*, pp70-77, May/June 2002.

[7] Anne, C.L. (1999). XML seen as integral to application integration. *IT Professional*, 2(5), 12-16.

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

[9] Peter Haumer, Klaus Pohl, " Requirements Elicitation and Validation with Real World Scenes" *IEEE Trans. Software Eng.*, vol. 24, no. 12, pp. 1036-1054, DECEMBER , 1998.

[10] 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.

[11] F.A.C. Pinhero and J.A. Goguen, "An Object-Oriented Tool for Tracing Requirements," IEEE Software, vol. 13, no. 2, pp. 52–64, Mar. 1996.

[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] H. Kaindl, "A Practical Approach to Combining Requirements Definition and Object-Oriented Analysis,"*Annals Software Eng.*, vol. 3, Sept. 1997, pp. 319–343.

[14] Bourdeau, R.H., & Cheng, B.H.C. (1995). A formal semantics for object model diagrams. *IEEE Transitions on Software Engineering*, 21(10), 799-821

[15] Deitel, H., Deitel, P., Nieto, T., Lin, T., & Sadhu, P. (2001). *XML how to*

program

Bennett, K. H. (1993). An overview of maintenance and reverse engineering, *The REDO* Upper Saddle River, NJ : Prentice Hall.

[16] H. Kaindl, "Difficulties in the Transition from OO Analysis to Design," IEEE Software, vol. 16, no. 5, Sept./Oct. 1999, pp. 94–102.

Compendium, John Wiley & Sons, Inc., Chichester.

[17] Institute of Electrical and Electronics Engineers. *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries.* New York, NY: 1990.

[18] O.C.Z. Gotel and A.C.W. Finkelstein, "An Analysis of the Requirements Traceability Problem," *Proc. First Int' I Conf. Requirement Eng,* pp. 91-101, 1994.

[19] R. Waking and M. Neal, "Why and How of Requirements Tracing" *IEEE Software, vol*. 11, no. 7, pp. 104-106, July 1994

[20] Bennett, K. H. (1993). An overview of maintenance and reverse engineering, *The REDO Compendium*, John Wiley & Sons, Inc., Chichester.