私 立 東 海 大 學
資 訊 工 程 研 究 所
碩 士 論 文

指導教授：朱正忠 教授

支援模型驅動架構之模型追蹤工具
Model Traceability Tool for Model Driven Architecture

研究生：林柏宇

中華民國 一百 年 六月

摘要

模型驅動開發的應用越來越廣泛，主要的概念是獲取所有系統設計的重要資訊，利用正規或是半正規的模型描述系統不同的抽象層級，將複雜的問題簡化，並且透過自動化程式碼產生的技術，產生出實際的程式碼，有效的加速系統開發，降低開發的成本，其中物件管理組織所定義的模型驅動架構是目前最接近此一概念之架構，統一塑模語言更成為其主要的核心語言，它可以描述系統不同面向的靜態或是動態觀點，這些觀點彼此相依並且有所交集，但是統一塑模語言並未定義模型間的關聯，且缺少檢驗模型之間的關聯性或是一致性的機制，當不一致的模型資訊出現時，更會影響模型驅動架構下程式碼產生的結果。

在本篇論文中，我們分析統一塑模語言的儲存格式，並根據彼此之間的關聯性，在可延伸標記語言為基礎的整合模型中，建立其水平及垂直的一致性，且透過相似度的量測，比對不同模型，找出潛在的關聯來建議系統開發人員，以達到軟體模型的追溯性，最後我們在 Eclipse 的環境下，建構此一外掛工具，協助系統開發人員進行塑模。

關鍵詞: 模型驅動、統一塑模語言、可延伸標記語言、*一致性*、*追溯性*

Abstract

The application of model-driven developments has grown wider recently. The main concept is to fetch all of the significant properties referring to system design, and use formal or semi-formal model to describe system in different levels of abstraction. It through code generation technique to generate source code, effective speed system development and reduce development costs. The model-driven architecture defined by object management group is the closet structure to this concept, and unified modeling language is the core modeling language of it, which can describe static or dynamic views of different facilities in a system. But lacks of consistency and relationship checking mechanism between models will affect the result generated under a model-driven architecture when inconsistent modeling information appears.

In this paper, we analyze the format of unified modeling language, according to the relationship between the model, using a XML-based unified model to establish vertical and horizontal consistency. Through the similarity measurement, it recommend system developers with potential relationship by comparing different models, to achieve software traceability. At last, build an assistant tool in Eclipse environment to help developer with software modeling.

Keywords: Model-Driven, UML, XML, Consistency, Traceability

章節目錄

圖目錄

表目錄

第一章 導論

1.1. 前言

軟體系統在我們日常生活中已扮演不可或缺的角色，隨著重要性的提升，許多學者、專家嘗試發展建構更容易、更快速、更低成本的軟體系統，過去幾年許多提高軟體開發及維護效率的方法相繼提出，包含軟體再利用(Reuse)[1]、可擴展標記語言(Extensible Markup Language)、設計樣板(Design Pattern)[2]及以元件為基礎的軟體工程(Component-Based Software Engineering)及服務導向架構(Service-Oriented Architecture)等。而近年來興起的模型驅動開發方式(Model-Driven Development, MDD)[3]應用越來越廣泛，其主要概念是獲取所有系統設計的重要資訊，利用正規或是半正規的模型描述系統不同的抽象層級，將複雜的問題簡化，並且透過自動化程式碼產生的技術，產生出實際的程式碼，有效的加速系統開發，降低開發的成本[4]，這些模型也能讓開發人員了解整體系統或是與專案相關的人員進行溝通，有利於確認與驗證。

物件管理組織 (Object Management Group, OMG)在 2001 年定義了(Model-Driven Architecture, MDA)[7]的方法，主要是用來實現模型驅動開發的框架，將開發過程分為三個階段，分別是獲取系統需求的計算獨立模型(Computation Independent Model，CIM)，只擷取系統功能但不考慮執行環境的平台獨立模型(Platform-Independent Model, PIM)以及將 PIM 轉換到明確的執行平台的特定平台模型(Platform-Specific Model, PSM)，並且統一塑模語言(Unified Modeling Language, UML)也逐漸成為 PIM 中最主要的塑模語言，且透過 Profile 的機制，使得這些圖形可以進行擴充，套用到適當的領域。

1.2.研究動機

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，可追溯性在軟體開發領域中指的是從需求階段一直到實作階段的追溯能力，也被稱作需求可追溯性(Requirement Traceability)，Gotel 等人給與這樣的定義[5]:「The ability to describe and follow the life of a requirement in both a forward and backward direction.」，與此同時，模型驅動開發越來越盛行，它強調著以模型作為系統開發的依據和模型與模型之間的轉換應用，因此可追溯性更應廣泛的包含所有的軟體產出物(Artifacts)，在這樣的趨勢，Aizenbud 等人給予更寬廣的定義[6]：「We regard traceability as any relationship that exists between artifacts involved in the software engineering life cycle.」

這些開發過程中的產出物，有可能是系統的需求文件、設計模型、程式碼或是測試案例，不同的產出物彼此之間有著強烈的相互依賴關係，必須要適當的去管理而且要隨著各種改變進行適當的調整與維護。追溯性可以幫助系統開發人員從需求階段細化(refine)到低階層的設計元件、建置可執行的系統和有效率的測試，進行一步的協助系統分析，了解隱含的改變以及確保沒有多餘的程式碼存在。更有利於往後系統的維護，減少人工校閱的時間，降低系統開發的成本。

1.3. 研究目的

有鑑於前一小節的動機，本論文提供一種方法能夠在模型驅動的環境下，建立軟體產出物的可追蹤性，尤其 UML 已經成為模型驅動架構下的核心語言，它描述系統的不同觀點與抽象層級，這些觀點彼此相依並且有所交集，必須有效的整合，達到跨階段、觀點間相關聯資訊的串接與再用，協助軟體的開發以及設計，並且我們在 Eclipse 的環境下，建置一個外掛工具，以達到此一目的。

1.4. 章節安排

論文的章節安排分述如下：

第二章　背景知識與相關研究，此章節說明與本論文相關的研究。

第三章　此章說明本論文使用之方法。

第四章　系統的設計與實作，本章說明系統的架構以及在 Eclipse 環境下如何被建構。

第五章　案例研究，根據所建構之系統，套用到案例，並展示相關功能。

第六章　結論及未來方向，此章將本論文之研究成果做總結，並提出未來研究方向。

第二章 背景知識與相關研究

2.1. 軟體生命週期

如圖 一所示，軟體開發的幾項基本活動包含需求分析、系統設計、實作、測試以及維護[8]，這樣的流程是彼此相依且可能具有重複性的，在每個階段都會有不同的產出物產生，隨著開發的流程，這些產出物會不斷的被細化，變成更詳細的系統資訊，保持這些產出物的一致性，能夠協助系統的開發，更能確保系統的維護性。

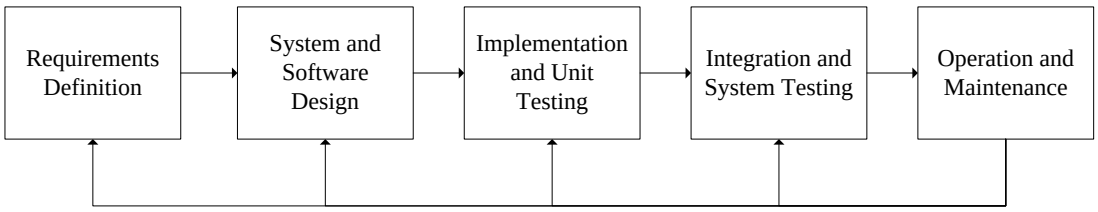

圖 一 軟體生命週期

2.2.模型驅動架構(Model Driven Architecture, MDA)

MDA 是 2001 年 OMG(Object Management Group)所採用作為軟體發展的一種標準框架(framework)，MDA 可以支援 OMG 所建立的各種模型標準：UML(Unified Modeling Language)、MOF(Meta-Object Facility)、XML(Extensible Meta Language)、XMI(XML Meta-Data Interchange)、CWN(Common Warehouse Meta-Model)及 COBRA(Common Object Request Broker Architecture)等[9]。

MDA 主要是一種軟體開發架構，透過以下核心模型進行系統發展[10]：

1. 平臺獨立模型(Platform Independent Model, PIM)：PIM 是一種抽象的模型，單純描述企業規則和功能，使用完整定義的語法和語義讓電腦可以自動解譯，

由於只是一種抽象的描述檔，所以在不涉及開發技術與系統運作平臺下可以將問題描述的更為明確，此階段主要是描述如何解決企業運作的問題，最常見的方法是以統一塑模語言（Unified Modeling Language, UML）來完成此階段的塑模。

2. 特定平臺模型特定平臺模型（Platform Specific Model, PSM）：PSM 是將上一階段 PIM 產生的檔轉換成使用者指定平臺；相對於 PIM，此階段是相依於特定平臺技術，如有關於 SQL Server 的 PIM 就轉換為相對應技術，像是"Table"、"Primary Key"等相關 pattern，都在此階段進行轉換。
3. 轉換程式模型轉換程式模型（Generating the Application）：產生程式碼（Generating the Application），因為上一階段 PSM 已經轉換為使用者指定的技術平臺，所以此階段只需根據 PSM 的相關檔案轉換為程式碼即可。

2.3.統一塑模語言(Unified Modeling Language, UML)

統一塑模語言(UML)[11]是一種 Modeling Language，結合了 G. Booch，J. Rambaugh，與 I. Jacobson 三人的物件導向方法論所提出的一種物件導向語言，並被 OMG 納入為標準。UML 結合了 Booch 的物件導向方法、Rambaugh 的物件塑模技術與 Jacobson 的物件導向軟體工程，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。應用在其他領域如：Real-Time System、Embedded System 以及 Workflow。

雖然 UML 被應用在許多領域，同時也被視為是塑模的重要工具，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[12]。然而不只 UML，包含所有現有的模型建置技術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，目前 OMG 亦發現這個問題，訂定了 Diagram Interchange 標準[13]，不過僅限於 UML 相關的圖形轉換，對於整合其他標準或非圖形塑模方式，尚未支援。

2.4. 文字資料之相似度量測 (Similarity Measurements of Text Data)

在資訊檢索中，相似度的量測被用來分析和彙整大量文字資料，應用在建立索引、進行搜尋、分群或分類等，都會用到相似度的量測方法，在向量空間模型中，常見的相似度量測包括 Simple Matching Coefficient、Dice Coefficient、Jaccard Coefficient、Overlap Coefficient 和 Cosine[14]，表 一顯示了這些方法的定義。

表 一 文字相似度的量測方法

相似度量測	定義
Simple Matching Coefficient	$\lvert X \cap Y \rvert$
Jaccard Coefficient	$\frac{X \cap Y}{X \cup Y}$
Dice Coefficient	$\frac{2\lvert X \cap Y \rvert}{\lvert X \rvert + \lvert Y \rvert}$
Overlap Coefficient	$\frac{\lvert X \cap Y \rvert}{\min(\lvert X \rvert, \lvert Y \rvert)}$
Cosine	$\frac{\lvert X \cap Y \rvert}{\sqrt{(\lvert X \rvert \times \lvert Y \rvert)}}$

在相似度的測量方法中，Simple Matching Coefficient 是最簡單易懂的，採用交集的原理，比對兩個文字集合之間的關聯性，Dice Coefficient 則是可以判斷出詞彙在兩集合中的出現次數，藉此可以找出類似、相關和同義等關係。Jaccard

Coefficient 最大優點即為其標準化的考量。Overlap Coefficient 和 Cosine 則相似於 Dice Coefficient，差異則為考量不同的權重作為分母。

2.5.XML 與 XMI

XML(eXtensible Markup Language)是由 W3C(World Wide Web Consortium)所制定的標準[14]，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。

XML 本身就是一種 Meta Language，可利用 XML 定義各種 Model 的文件語言，在進行各種 Model 轉換時可以利用 XML 延伸出的 XMI 來幫助檔的交換及存取。XMI 是 XML Metadata Interchange 的縮寫，在 XMI 裡 X 代表 XML 跟 extensible 雙重意義。XMI 在 1999 年 3 月 23 日正式被 W3C 所建議為正式標準。

XMI[16]是一種提供以 XML 來交換 Object-Oriented Models 和資料的標準，XMI 同時是 OMG(Object Management Group)所認可的一種新的標準，讓使用者可以根據 XMI 的標準標籤(Tag)將 UML 文件轉換為 XML 及 DTD 文件。XMI 讓 XML 使用起來更加的容易，因為它應用了 UML 圖形化的特性來產生 XML 所需的文件及 DTD 檔。而且 XMI 讓 DTD 檔、軟體設計及 XML 檔資料可以一致。

2.6.基於 XML 之整合模型(XML-Based Unified Model)

XUM 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、設計與實作階段。另一部分是是模型間的整合關連。XUM 實體上由三個部分組成，元件型態、關連型態與整合連結型態三部份，其中整合連結型態定義三種連結來達到不同觀點的整合。

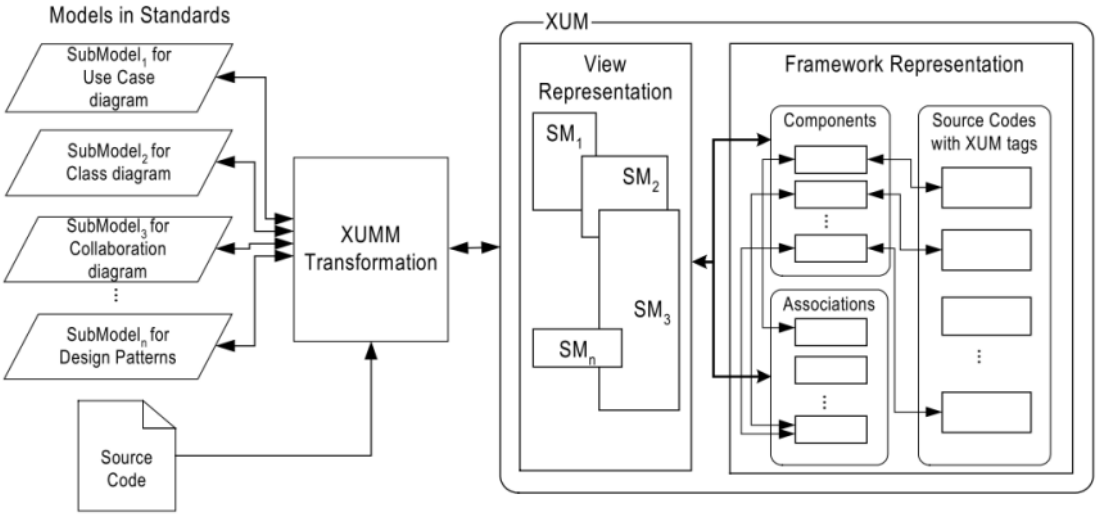

圖 二 XUM 與子模型之關係

2.7.原生 XML 資料庫(Native XML Database)

原生 XML 資料庫[17]是以 XML 文件最為基本的儲存單位，有別於關聯式資料庫是以表格最為基本的儲存單位，原生式 XML 資料庫不會改變 XML 文件的結構，適合用來儲存以及維持 XML 的完整性。XML 資料庫對於 XML 文件的存取，一般來說都會表示成階層式的資料儲存模型或是樹狀的文件結構，不需像關聯式資料庫或是其他 XML 文件操作方式，還需要額外的轉換步驟，並且類似關聯式資料庫 SQL 的查詢語法，原生 XML 式資料庫可以使用 XQuery 的語法進行文件的檢索。

第三章 研究方法

UML 已經廣泛的使用於軟體系統的設計當中，每一種模型代表系統的不同觀點與面向，並隨著模型驅動架構的出現，提供一種從 UML 模型轉換到原始碼的標準。自動化的產生原始碼避免了手動產生可能發生的錯誤與不一致，這也使得設計模型間的一致性更為重要，當不一致的模型出現，所產生出的原始碼也會是錯誤的。尤其隨著系統設計的細化與演化，模型會不斷的改變與修正，確保這之間的一致性與完整性就顯得勢在必行，更進一步的能加速系統的開發與降低系統維護的成本。

3.1.一致性的分類

過去的研究指出 UML 模型的一致性可以分類為下面幾種[18][19]:

垂直一致性(Vertical Consistency)：又稱交互模型(Inter-Model)一致性，指在不同的抽象層級中，不同的模型應該保持一致性。在軟體開發的程序中，當規格被轉換到更細部的規格時，一致性應要能被確保，隨著軟體程序這樣的動作會不斷的重複。

水平一致性(Horizontal Consistency)：又稱內部模型(Intra-Model)一致性，意指在相同抽象層級中不同的模型間應該保持一致性，例如類別圖、循序圖以及狀態圖彼此之間應該保持一致。

演化一致性(Evolution Consistency)：驗證不同版本的相同模型應保持一致性。

語意的一致性(Semantic Consistency)：在 UML 的 Metamodel 中，語意的意義應該被驗證。

語法的一致性(Syntactic Consistency)：在 UML 的 Metamodel 中，UML 模型的規格應該被驗證。

在本論文中，我們利用 XUM 的理論，達到 UML 垂直與水平的一致性，整

合使用案例圖(Use Case Diagram)、類別圖(Class Diagram) 、狀態圖(State Machine)、循序圖(Sequence Diagram)等模型，這四種圖型最被常用來描述系統的不同觀點，因此優先被探討。

3.2.整合模型 XUM

UML 可以描述系統不同面向的靜態與動態觀點，但本身並未定義彼此之間的關聯，在過往的研究當中探討了不同的 UML 模型間的關聯性[20][21][22][23[24][25][26]，可以歸納如圖 三，系統的描述由使用案例開始，表明了系統的功能面向和使用者與其之關係，再更進一步的細化設計則利用類別圖顯示系統的靜態觀點，當某個類別圖的邏輯過於複雜時，狀態的轉移可能分散在不同的使用案例中，則由狀態圖針對事件以及狀態做統一的描述，而循序圖分別可以描述使用案例以及類別圖的動態觀點，以進行驗證，最後這些模型會產生明確的程式碼去實作。

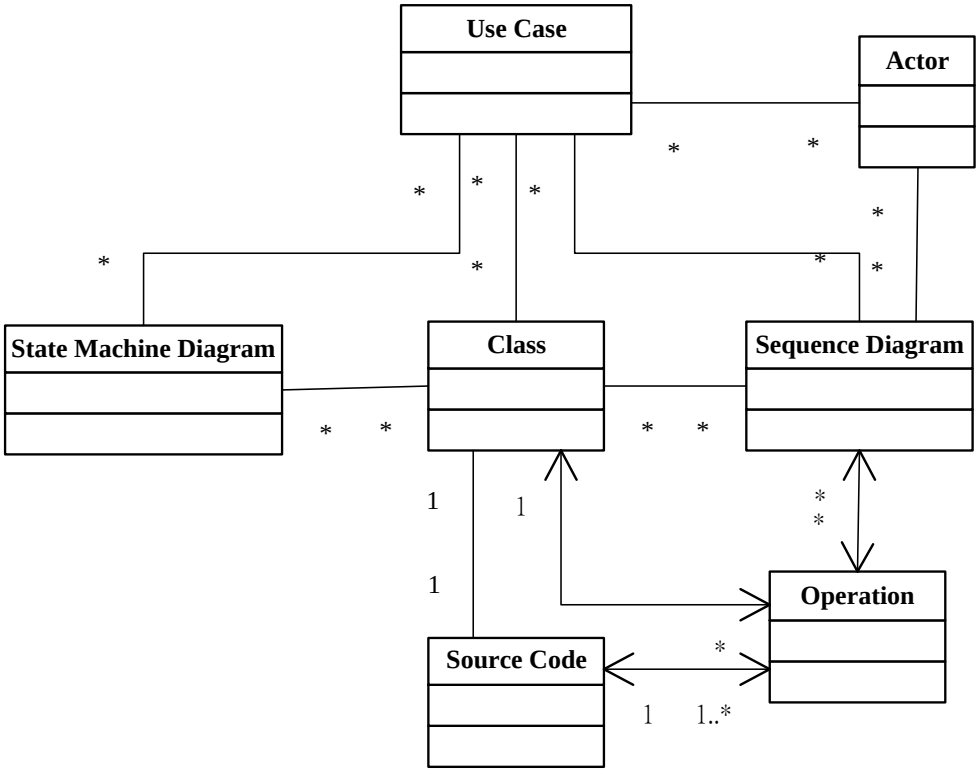

圖 三 UML 之間的關聯

為了建立模型之間的追蹤性，我們運用 XML 的標準建立整合模型，記錄必要的模型資訊，在模型驅動的架構下，目前的 UML 模型資訊都以 XMI 的標準進行儲存，以圖形交換(Diagram Interchange)的格式記錄圖形的名稱、位置、關係、大小等圖形化的資訊，所以要建立完整的整合模型，必須從這兩個標準的檔案中取得相關的資訊。圖 四顯示類別圖在 XMI 格式下的表示方式，以 packagedElement 的標籤記錄不同的 UML 模型，包含 xmi:type 用來識別此 UML 的類型，另外具有唯一的識別 ID 和此類別的名稱。而類別所在的圖形名稱則是從圖形交換的格式中獲取，如圖 五所示，在樹狀結構的最上層包含類別所在的圖形名稱，而樹狀結構內，可以找出符合相同 ID 之屬性值。

```
<packagedElement xmi:type="uml:Class" xmi:id="_VlIh8PcDEd-tF4FYV-UJQA" name="EncodedDataBufferManager">
  <ownedAttribute xmi:id="_BpYIsPcEEd-tF4FYV-UJQA" name="buffer1" type="_vaLNUPcDEd-tF4FYV-UJQA" isUnique="false"/>
  <ownedAttribute xmi:id="_DSie4PcEEd-tF4FYV-UJQA" name="buffer2" type="_vaLNUPcDEd-tF4FYV-UJQA" isUnique="false"/>
  <ownedOperation xmi:id="_8zND8PcDEd-tF4FYV-UJQA" name="EDTI"/>
  <ownedOperation xmi:id="_88-D8PcDEd-tF4FYV-UJQA" name="RTS"/>
  <ownedOperation xmi:id="_9CYvcPcDEd-tF4FYV-UJQA" name="SFS"/>
  <ownedOperation xmi:id="_9E0_cPcDEd-tF4FYV-UJQA" name="SVF"/>
  <ownedOperation xmi:id="_AQ89MPcEEd-tF4FYV-UJQA" name="getfilename"/>
</packagedElement>
```

圖 四 XMI 中類別的描述

```
<di2:Diagram isVisible="true" fontFamily="Arial" lineStyle="solid" fontColor="255:255:255" foregroundColor=
"255:255:255" backgroundColor="255:255:255" borderColor="255:255:255" position="0:0" name="VSS Class Diagram">
                                        ⋮
    <semanticModel xsi:type="di2:Uml1SemanticModelBridge" presentation="TextStereotype">
      <element xsi:type="uml:Class" href="VSS Class Diagram.uml#_VlIh8PcDEd-tF4FYV-UJQA"/>
    </semanticModel>
                                        ⋮
</di2:Diagram>
```

圖 五 圖形交換格式的描述

當我們從上述的兩種標準獲取相關的資訊時，即可將需要的資訊寫入至整合模型之中，如圖 六是 XUM 整合模型中類別圖的定義，分別從 XMI 的檔案中取得類別的 ID、名稱以及來源的檔案，並寫入指定的屬性，從圖形交換格式取得此類別所屬的圖形名稱，另外定義三種不同的鏈結記錄與其他 UML 的關聯性。

```
<xsd:complexType name="classElementType">
    <xsd:sequence>
        <xsd:element ref="abstraction_link_id" maxOccurs="unbounded"
            minOccurs="0">
        </xsd:element>
        <xsd:element ref="integration_link_id" maxOccurs="unbounded"
            minOccurs="0">
        </xsd:element>
        <xsd:element name="sourcecode_link_id" type="xsd:string" maxOccurs="1"
            minOccurs="0">
        </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="from_file" type="xsd:string"></xsd:attribute>
    <xsd:attribute name="from_diagram" type="xsd:string"></xsd:attribute>
    <xsd:attribute name="name" type="xsd:string"></xsd:attribute>
    <xsd:attribute ref="id"></xsd:attribute>
</xsd:complexType>
```

圖 六 類別圖的定義

在每一種模型的表示下，定義了整合關係(Unification Link)，整合關係是用以整合各階段相關產出物的機制。其中包含了抽象鏈結(Abstraction Link)、整合鏈結(Integration Link)與程式碼鏈結(Source Code Link)，如圖 七定義了抽象鏈結，包含了幾個屬性，分別是此鏈結的 ID、來源的圖形名稱以及類型、指向的圖形名稱以及類型、此鏈結的建立是否由使用者所決定和此鏈結建立的時間。

```
<xsd:complexType name="abstraction_linkType">
    <xsd:attribute name="date" type="xsd:string"></xsd:attribute>
    <xsd:attribute name="userdefined" type="xsd:string"></xsd:attribute>
    <xsd:attribute name="toType" type="xsd:string"></xsd:attribute>
    <xsd:attribute name="to" type="xsd:string"></xsd:attribute>
    <xsd:attribute name="fromType" type="xsd:string"></xsd:attribute>
    <xsd:attribute name="from" type="xsd:string"></xsd:attribute>
    <xsd:attribute ref="id"></xsd:attribute>
</xsd:complexType>
```

圖 七 抽象鏈結的定義

抽象鏈結是用來串聯較抽象的軟體需求與較具體的軟體設計，軟體的生命週期中，在需求分析的階段，會利用使用案例圖獲取系統的需求，描述不同的系統功能面向或是目標，明確的指出系統該做的事情，而不是如何去做，並從較高的

層級來觀看整個系統，這些被識別出的功能會在設計的階段，使用多個類別圖描述系統構成的物件以及其間的關係，再細化成循序圖與狀態圖描述動態的結構和事件，如圖 八顯示使用案例圖與類別圖之間的關聯，兩種圖形分別位於不同的系統抽象層級，其中使用案例的描述，可能會被定義成類別的名稱或是透過特定類別內的方法去實現，透過抽象鏈結，將這些位於不同系統抽象層級的資訊進行整合，以達到垂直一致性。

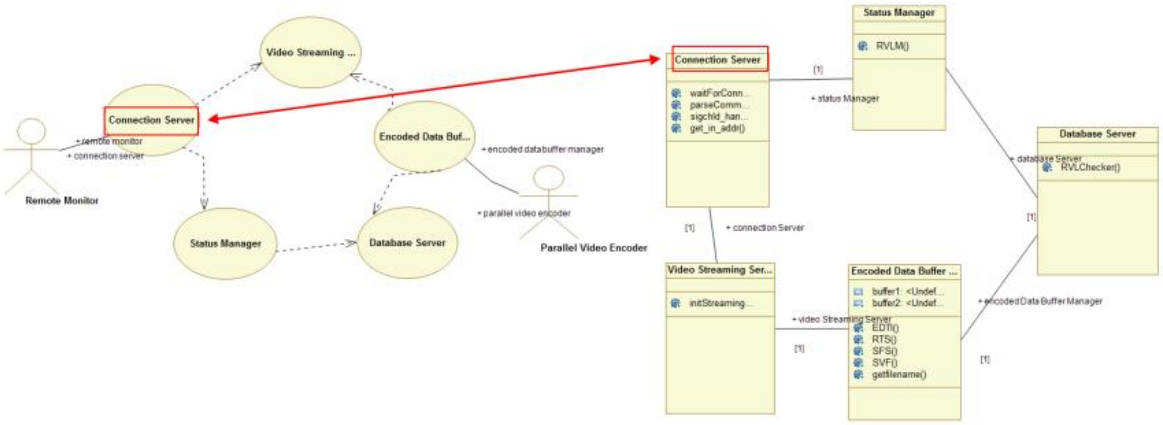

圖 八 抽象鏈結的示意

整合鏈結的目的在於將相同的抽象層級或共用性質的軟體模型做關聯，顯示出物件共享，定義如圖 九所示，同抽象鏈結包含了相同的屬性來記錄鏈結的資料。在設計的階段，類別圖被用來表明靜態的結構，但為了清楚表示使用案例的詳細流程或是物件之間的動態操作關係，會使用循序圖來表示，循序圖主要由生命線以及生命線上的訊息呼叫所構成，循序圖上的生命線或是訊息，應該在類別圖中定義過，彼此間應有多對多的對應關係。而不同於循序圖用來表示多個物件之間的互動行為，狀態圖會被用來顯示單一特定類別的事件、狀態轉移情形，但並不是所有的類別都一定會需要狀態圖的表示，只有某些特定的行為才需要。圖十則顯示這樣的關聯，如 Database Server 類別分別屬於 UML 類別圖中的類別、循序圖中的物件，而 Status Manager 類別另有狀態圖描述事件的狀態，因此彼此

有整合鏈結的關係，透過整合鏈結可以達到相同抽象層級的水平一致性。

```
<xsd:complexType name="integration_linkType">
    <xsd:attribute name="date" type="xsd:string"></xsd:attribute>
    <xsd:attribute name="userdefined" type="xsd:string"></xsd:attribute>
    <xsd:attribute name="toType" type="xsd:string"></xsd:attribute>
    <xsd:attribute name="to" type="xsd:string"></xsd:attribute>
    <xsd:attribute name="fromType" type="xsd:string"></xsd:attribute>
    <xsd:attribute name="from" type="xsd:string"></xsd:attribute>
    <xsd:attribute ref="id"></xsd:attribute>
</xsd:complexType>
```

圖 九 整合鏈結的定義

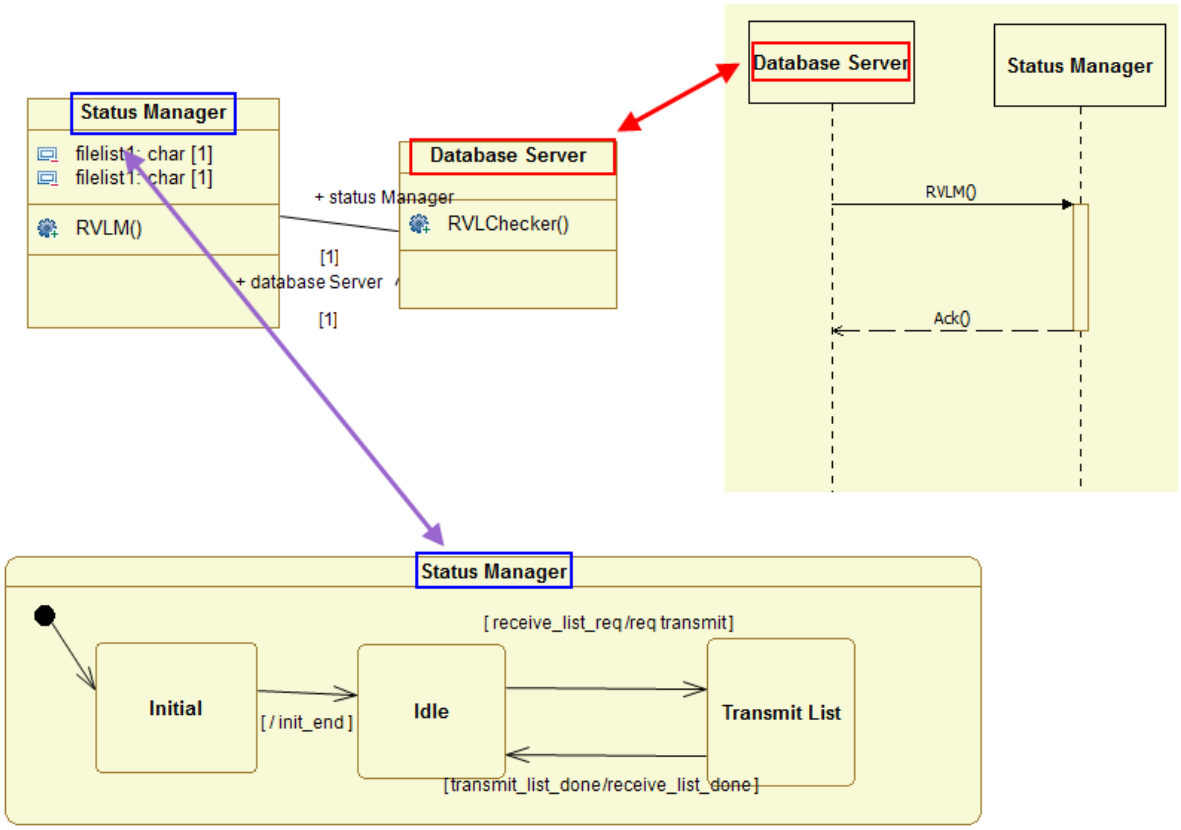

圖 十 整合鏈結的示意

程式碼鏈結則是用來串聯設計模型資訊中的類別與實作的程式碼，如圖 十一是其定義，主要包含此鏈結的 ID、來源的類別圖到指向的程式碼、此鏈結的

建立是否由使用者所決定和此鏈結建立的時間，圖 十二顯示由程式碼擷取出的資訊，類別與程式碼之間，從名稱、變數到方法，彼此之間應該有著一對一的關係，任何一邊都不可以只有單獨的定義出現。

```
<xsd:complexType name="sourcecode_linkType">
    <xsd:attribute name="date" type="xsd:string" />
    <xsd:attribute name="userdefined" type="xsd:string"></xsd:attribute>
    <xsd:attribute name="to" type="xsd:string" />
    <xsd:attribute name="from" type="xsd:string" />
    <xsd:attribute name="id" type="xsd:string" />
</xsd:complexType>
```

圖 十一 程式碼鏈結的定義

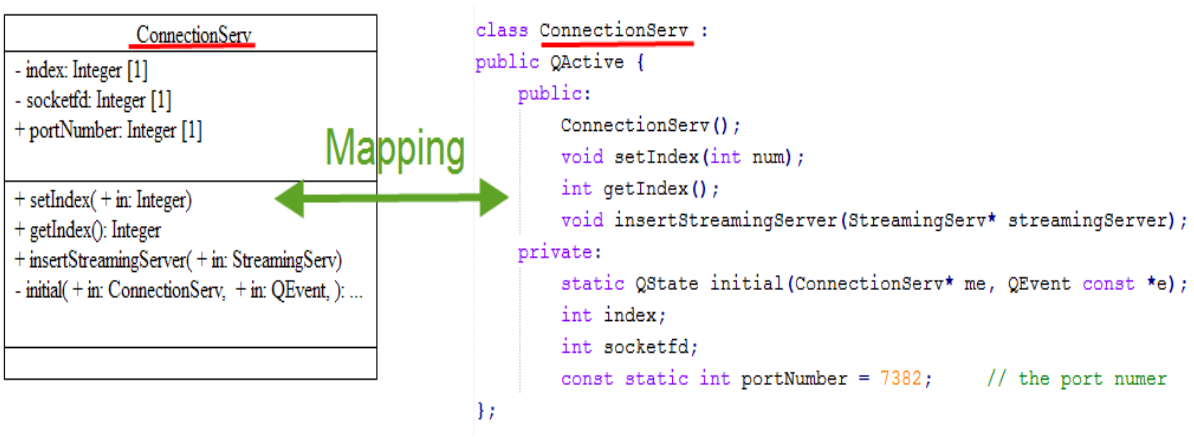

圖 十二 從程式碼中擷取出的資訊

3.3. 模型的相似度比對

在資訊檢索中，常見有幾種方法來量測文字的相似度，包含 Jaccard Coefficient、Bleu Coefficient、Dice Coefficient 等方法。在本論文中利用文字相似度比對的方法來進行軟體產出物的比對，找出可能不一致的部分，發現潛在的錯誤，更能進一步建議使用者建立可能的關聯性，Dice Coefficient 在過往的研究

顯示其成效較不受出現次數多寡的影響，且計算簡單[27]，因此在本篇論文中被使用，定義式(1)所示：

$$s = \frac{2n_t}{n_x + n_y} \quad \text{(式 1)}$$

n_x 代表字串 X 的雙元字串(Bigram)數目，n_y 代表字串 Y 的雙元字串數目，n_t 則是分別出現在 X 與 Y 字串的雙元字串數目，計算的結果會出現介於 0 到 1 之間的值，分別代表完全不同與完全相似。以類別圖與原始碼的類別名稱比對為例，表 二顯示幾個不同計算區間的結果，套用 Dice Coefficient 的計算，除了有量化的數值依據，更可以省去對於文字比對所需要做的前置處理，例如字串的切割等，能夠找出潛在的相同或是相似名稱之軟體產出物，另一方面當使用者寫入錯誤之名稱時，也有利於校正，在本篇論文的系統中，我們預設比對顯示的結果是大於 0.5 小於 1 之間，以避免使用者得到過多不必要之資訊。

表 二 類別圖與原始碼類別名稱的文字相似度比對

類別圖	原始碼類別名稱	相似度
ConnectionServ	ConnectionServer	0.92
	Connection	0.81
	CnnServ	0.52
	Server	0.33
	Quantization	0.25
	StreamingServ	0.24
	Camera	0.11
	HuffStatistics	0.07

第四章 系統的設計與實作

本章依據前一章所提之方法，實作一個基於 Eclipse 環境下之外掛程式，並將簡介其系統架構與相關功能。

4.1. 系統架構

Eclipse 除了核心之外，其餘的部份皆由不同的外掛去組合，形成一套完整的整合開發環境，如圖 十三所示 Platform Runtime 為其核心部分，實作 OSGi R4 規格，並提供環境讓這些不同的外掛去載入、整合與執行。Workspace 負責管理使用者的資源，這些資源組成不同的專案，對應到實際的檔案系統上。Workbench 主要是 Eclipse 的圖形介面，由 JFace 與 SWT 兩個 GUI 元件所構成。PDE 提供工具讓使用者能去開發自己的外掛程式[28]。在此我們利用 Papyrus UML 塑模工具讓使用者進行 UML 的塑模，接著經由我們的系統進行模型的分析。

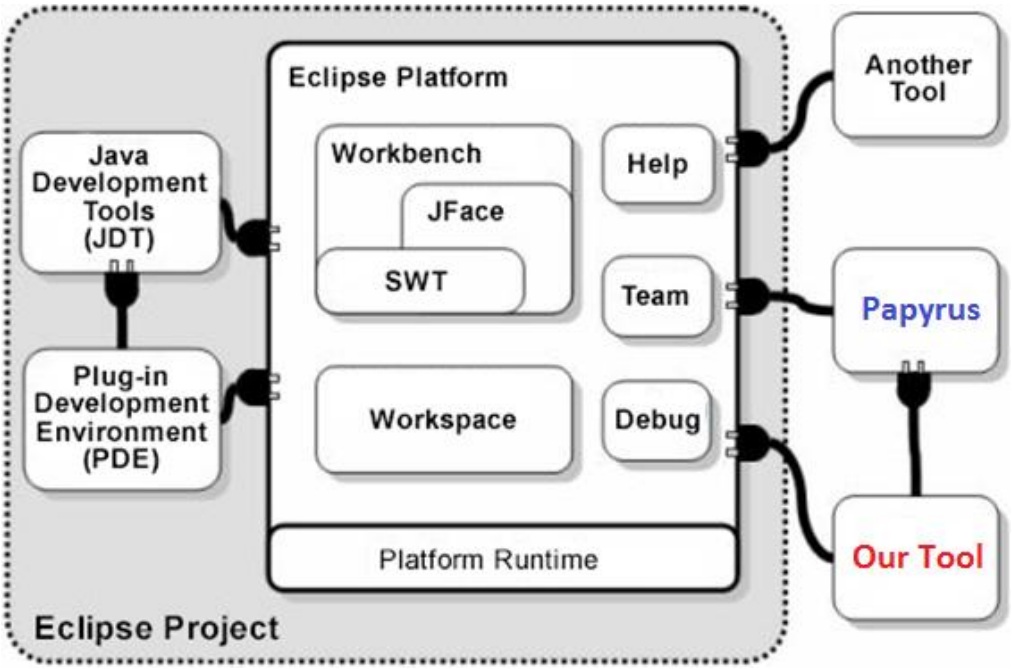

圖 十三 Eclipse 架構

本論文之系統架構如圖 十四所示。

主要可以分成五個部分，簡述如下：

1. 使用者介面部分：提供使用者視覺化的介面，觀看模型間的關聯性，事件的紀錄等。
2. XUM 處理模組：針對 XUM 檔案不同之情況，進行寫入寫出之動作。
3. 追蹤整合管理模組：比對不同模型間之關聯，找出相同或相似之字串。
4. 事件記錄模組：記錄操作過程中產生之事件。
5. XML 資料庫：儲存所有的模型資料、XUM 檔案、事件記錄等。

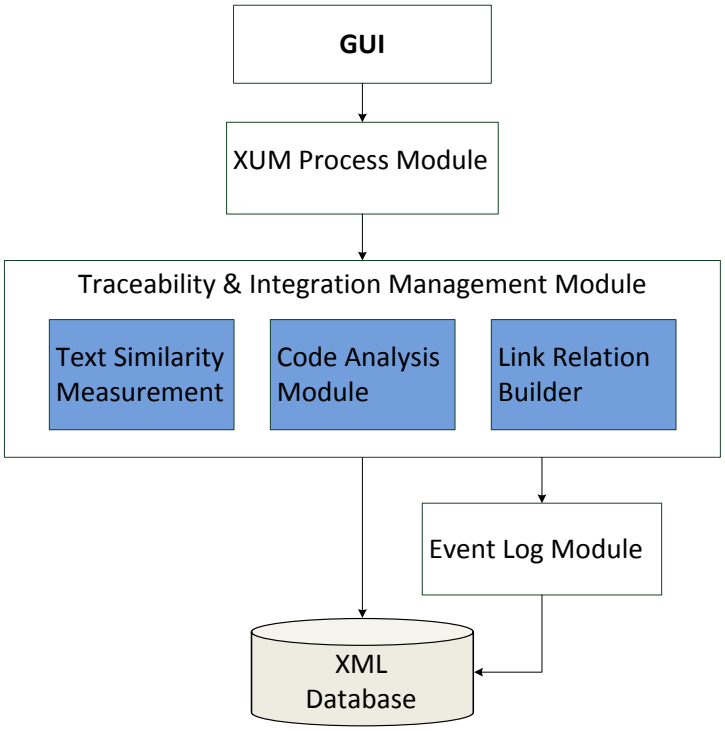

圖 十四 系統架構

4.2. 使用者介面

我們利用 JDT、PDE 所提供之元件開發我們的外掛程式，如圖 十五，整個 Workbench 中包含 Toolbar Menu、Editor、View 三種不同類型的使用者介面

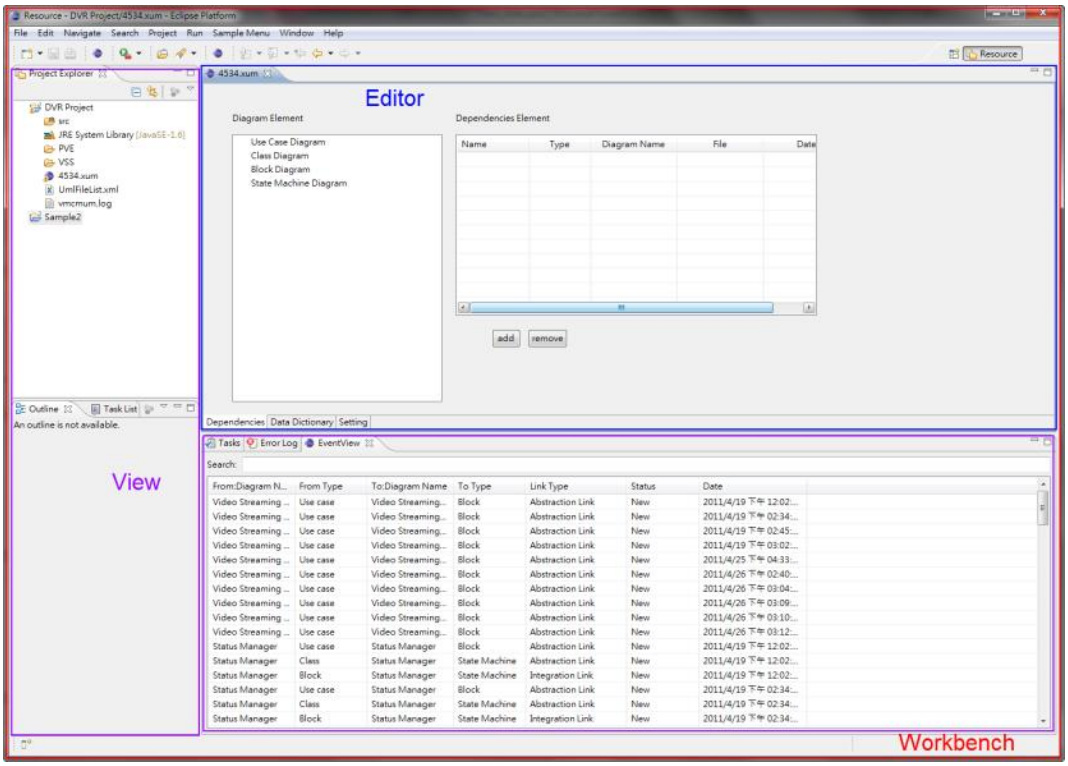

圖 十五 Eclipse Workbench

4.2.1. 編輯器(Editor)

編輯器的目的主要在於顯示、維護檔案的內容，當使用者在 Papyrus 的專案下新增一個.xum 檔案，雙擊此檔案，會根據附檔名開出對應的編輯器，在 Eclipse 的環境下，我們透過 org.eclipse.ui.editors 的延伸點，定義自己的編輯器，如圖 十六所示，並且繼承 org.eclipse.ui.part.MultiPageEditorPart 這個類別進行實作。

```
<extension
      point="org.eclipse.ui.editors">
   <editor
         class="com.vmc.mum.editors.DependenciesEditor"
         default="true"
         extensions="xum"
         icon="icons/sample.gif"
         id="com.vmc.mum.dependencies.editor"
         name="Dependencies Editor">
   </editor>
</extension>
```

圖 十六 org.eclipse.ui.editors 的延伸點

4.2.2. Log View

在 View 的部份，則是定義 org.eclipse.ui.views 延伸點，如圖 十七所示，主要目的用來顯示有階層示的資訊以及目前所開啟的編輯器之資訊，需要繼承 org.eclipse.ui.part.ViewPart 這個類別進行實作。

```
<extension
      point="org.eclipse.ui.views">
   <category
         id="com.vmc.mum.category.view"
         name="Model Depedencies">
   </category>
   <view
         category="com.vmc.mum.category.view"
         class="com.vmc.mum.view.EventView"
         icon="icons/sample.gif"
         id="com.vmc.mum.EventView"
         name="EventView"
         restorable="true">
   </view>
</extension>
```

圖 十七 org.eclipse.ui.views 的延伸點

無論是編輯器，還是事件記錄的觀點，都是採用 Model-View-Controller (MVC) 的設計樣板[29]，MVC 將系統的資料與顯示、使用者互動分離，簡化系統的複雜度，也賦予系統各個部分應有的功能，對於系統的後續維護與擴充提供一個良好的架構。

- **Model:** 封裝與程式的邏輯相關的資料以及對資料的處理方法。
- **View:** 負責資訊的展示和提供基本的使用者互動，在 Eclipse 環境下實作 ILabelProvier 介面，能針對樹裡面的內容呈現不同的圖片或是文字。
- **Controller:** 扮演 Model 與 View 的協調者，透過實作 ITreeContentProvider 介面，將 Domain Model 的物件轉換到 UI 介面上。

4.3.XUM 的建立與追蹤管理

為了要記錄所有的軟體產出物之關連性，我們利用前一章所述之概念，定義模型之內容以及彼此的連結型態，將這些資訊儲存成 XML 的形式，在 JAVA 對於 XML 檔案的存取，最常見的方法就是利用 SAX(Simple API for XML)與 DOM(Document Object Model)，SAX 的方式不將整份 XML 文件讀入記憶體中，所以存取快速，而 DOM 的方式會將資料存入記憶體中，以樹狀結構去表示，但兩者的操作皆較複雜，尤其面對規模較大之檔案形式，需要撰寫而外的程式，且較不易管理與維護[30]。另外一種存取的方式，則是使用 XML 資料綁定(XML Data Binding)，資料綁定的方式可以將 XML 的文件以物件的方式表示在記憶體中，本系統中利用 JAXB(Java Architecture for XML Binding)的技術來建立與維護 XML 檔案，如圖 十八所示，JAXB 是透過定義 XML Schema，並經由 Binding Compiler 來產生操作 XML 對應的類別，與 DOM 相同，JAXB 會完整的將整份 XML 文件載入記憶體中，形成樹狀的結構，這些被產生出的類別定義許多 Set 與 Get 的方法，能直接對 XML 中的元素或是屬性的值進行存取，不需像 DOM 還要為個別 XML 元素撰寫額外的程式。

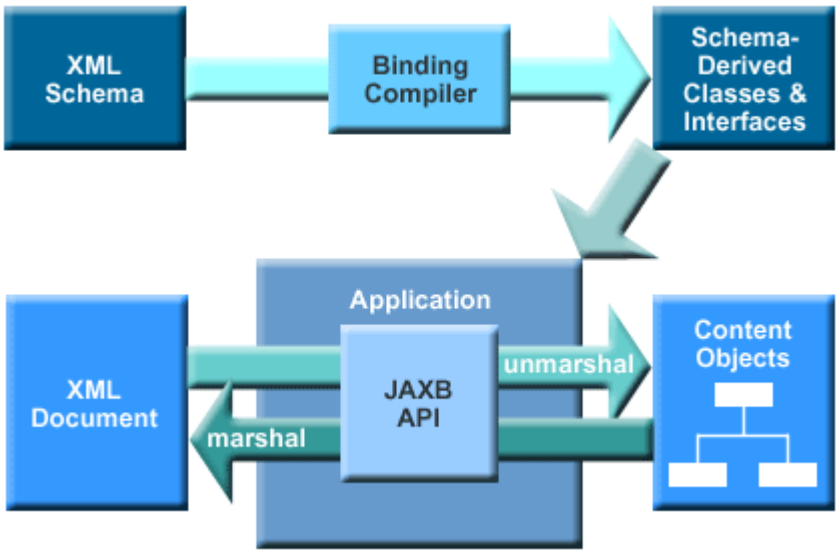

圖 十八 JAXB 示意圖[31]

4.3.1. XUM Content Tree 的存取

JAXB 將 XML 文件轉換成樹狀結構物件透過 Unmarshall 的方法，反向的操作則是利用 Marshall 的方式，在樹狀結構建立之後，為了確保它的唯一性，只有一個物件會被實體化，並節省記憶體的使用以及反覆實體化的資源浪費，在這邊採用了 Singleton 樣板，限制物件實體的數量，類別圖的表示如圖 十九所示，這樣的設計針對物件提供一個全域的存取點，不同的 UML 圖形 Handler 共同存取同一個 XUM 的模型，避免不斷的 Marshall 和 Ummarshall。

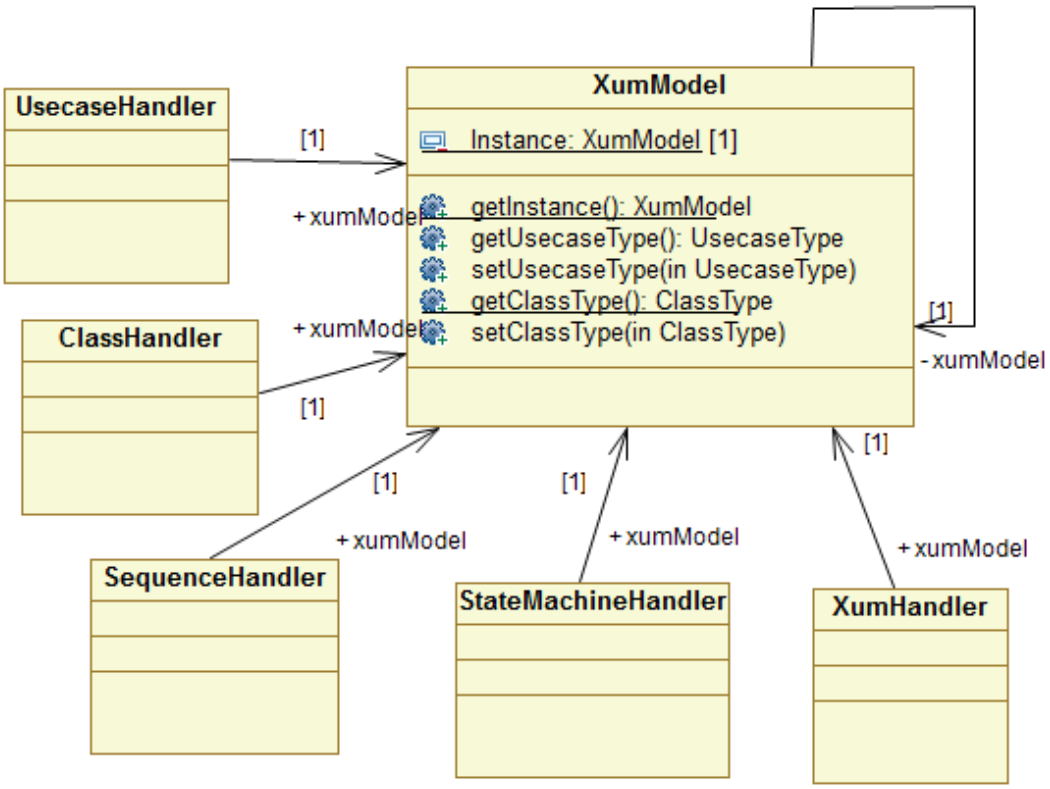

圖 十九 Singleton Pattern 的使用

Singleton 樣板的程式碼如下所述，為避免遭到外部類別的隨意存取，所以建構子被宣告為私有，而透過靜態公開的方法在有需要使用到時才被實體化，也就是所謂的 Lazy Initialization。

```
public class XumModel {
    private static XumModel instance=null;
    private XumModel(){
        //..
    }
    public static XumModel getInstance() {
        if(instance==null){
            instance =new XumModel();
        }
        return instance;
    }
}
```

4.3.2. 模型變更的通知

因為 UML 模型間彼此存在著關聯，當其中一種模型改變時，勢必會反映到其他有關聯的模型，彼此有著一對多的依存關係，而 Observer 樣板在這邊被用來實踐模型變更的事件通知。當被觀察者的資料或狀態的變動，因為這個物件的改動會影響到其中物件，就會自動通知其他和互有關連的部分，也就是觀察者，並通知使用者是否其他有關聯的資料也要一起更動。在 Java 的語言中，被觀察者透過繼承 java.util.Observable 進行實作，而被觀察者則實作 java.util.Observer 的介面，圖 二十的類別圖顯示其間的關係。

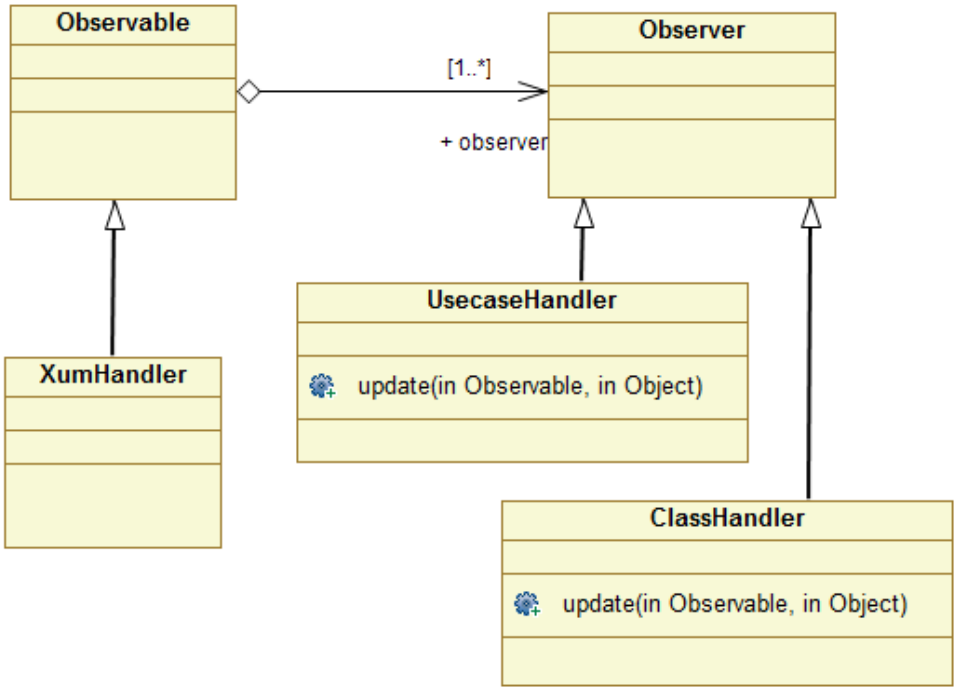

圖 二十 Observer Pattern

4.4. 事件記錄

當系統開發人員對各種不同的模型進行新增或修改時，都會影響到彼此模型關係鏈結的建立，這樣的操作過程應該要能妥善且完整的紀錄，提供一套詳細的歷程，讓系統開發人員能追溯過往的行為，所以本篇論文使用 Java 所提供的 Logging 應用程式介面，進行這些動作的紀錄與管理，能夠幫助系統開發人員了解系統開發過程中所發生的情況，有利於後續的追蹤與維護。

4.4.1. 鏈結事件觀點

記錄的觀點包含兩種，第一種是記錄鏈結建立或異動的過程，由來源的 UML 模型指向目的地的 UML 模型，如表 三顯示此觀點所包含的欄位。使用者可以透過不同的欄位組合與排序，獲取不同的訊息，例如，搜尋 Name 與 Date，使用者可以知道某一個 UML 項目過往的異動歷程；搜尋 From：Name 與 To：Name，使用者可以知道 UML 項目的關聯性，而 Name 加上 Link Type 的組合，使用者則可以知道此 UML 項目位於相同或是不同的抽象層級。

表 三 鏈結事件觀點欄位

欄位名稱	描述
From: Name	來源 UML 項目的名稱。
From: Type	來源 UML 項目的類型，例如 Use Case、Class 等。
From: Diagram Name	來源 UML 項目的圖形名稱。
To: Name	目的地 UML 項目的名稱。
To: Type	目的地 UML 項目的類型，例如 Use Case、Class 等。
To: Diagram Name	目的地 UML 項目的圖形名稱。
Link Type	鏈結的型態，例如抽象鏈結、整合鏈結等。
Status	此鏈結異動的狀態，例如新增、刪除等
Date	此鏈結異動的日期。

4.4.2. 維護記錄觀點

第二種記錄觀點主要是用來記錄 UML 項目名稱變更的過程，每一個 UML 項目都具有唯一的識別 ID，經由 ID 的核對，可以檢查出此項目名稱等相關資訊是否經過使用者異動，這些被使用者變動的資訊，會被記錄下來，如表 四顯示其所包含的欄位

表 四 維護記錄的欄位

欄位名稱	描述
Description	表明使用者異動的內容，例如：Change use case name to "Video Streaming"。
Date	此操作發生的日期。

4.5. 資料庫設計-實作 Facade 樣板

資料庫的使用對於資料的存取與儲存能夠提供較好的管理方式，本系統使用 XML 原生資料庫來儲存文件，有別於關聯資料庫是以表格的方式儲存，原生資料庫以一個檔案為基本單位，使用指定的 ID 當作識別檔案的依據，並使用 XQuery 語法查詢指定檔案的內容，在此本篇論文的系統中，整合了開放原始碼的 Sedna XML Database 作為的資料庫，並實作 Facade 樣板，提供單一介面對資料庫進行存取，簡化資料庫的使用、隱藏所依賴的程式庫、降低對程式庫的耦合、有利於分工合作，如圖 二十一所示，在 Sedna 本身提供的 API 中，Sedna Connection 主要是負責資料庫的連線功能，而 Sedna Statement 則是負責文件的異動與查詢，透過 SednaDB 的統一介面，可以讓操作更為簡單。

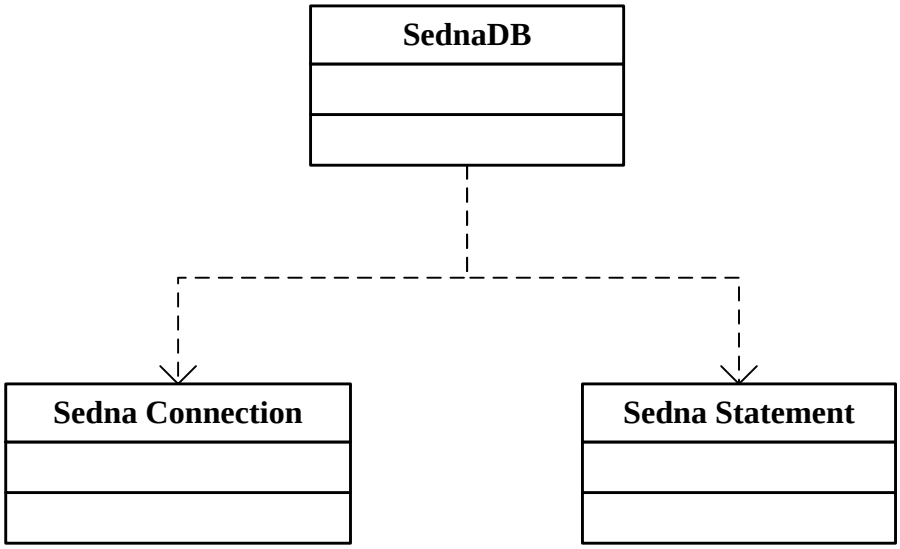

圖 二十一 Facade 樣板

另外，為了讀取 UML 模型的資訊，系統的運作流程會將.uml 的檔案以及.di2 的檔案，儲存到資料庫中，兩者分別是 XMI 與 Diagram Interchange 的標準，再利用 XQuery 的語法獲取所需要的資訊，XQuery 類似 SQL 都屬於宣告式語言，並由路徑運算式、FLWOR(For, Let, Where, Order by, and Return)運算式、條件運算式和 XQuery 函數所組成，以下為語法的範例：

```
declare namespace xmi='http://schema.omg.org/spec/XMI/2.1';
declare namespace uml='http://www.eclipse.org/uml2/3.0.0/UML';
declare namespace xsi='http://www.w3.org/2001/XMLSchema-instance';
declare namespace ecore='http://www.eclipse.org/emf/2002/Ecore';
declare namespace di2='http://www.papyrusuml.org';

for $d in doc('PVE Class Diagram.uml')/uml:Model/packagedElement[@xmi:type='uml:Class']
let $s :="PVE Class Diagram.di2" return (string($d/@xmi:id),string($d/@name),$s)
```

首先針對 XMI 格式中的命名空間進行宣告，接著這段語法分別從.uml 的檔案中取得類別的 ID 以及名稱，另外從.di2 的檔案中取得此類別所屬的圖形名稱。

第五章 案例研究與評估

在本節中，我們將針對 DVR (Digital Video Recording)系統中的塑模作為本篇論文的研究案例。DVR 是一個即時錄影的多媒體系統，可以提供遠端使用者即時觀看或者是收看已錄影的存檔影片，由多部攝影機與資料庫結合建構而成的系統，DVR 系統主要分為三大部份，包括：Remote Monitor Client (RMC)、 Video Streaming Server (VSS)及 Parallel Video Encoding (PVE)。

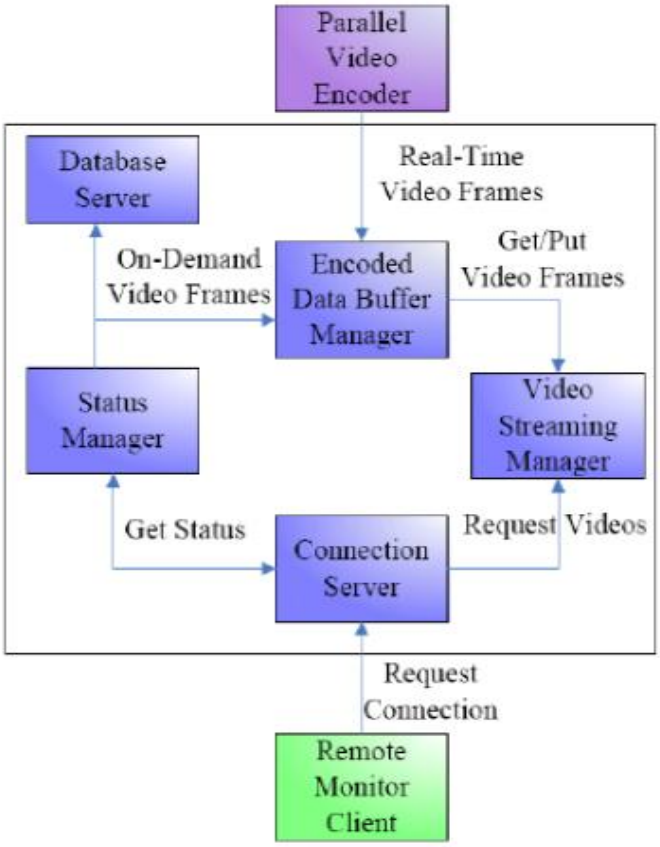

圖 二十二 DVR 系統架構

在本案例中，使用者會透過 Papyrus 對系統進行塑模，產生包含使用案例圖、類別圖、狀態圖、循序圖等，接著透過本論文所提之工具，對模型間進行分析，最後會將這些資訊儲存到資料庫中。

5.1. 系統流程

本系統的流程如圖 二十三活動圖所示,使用者必須先在 Papyrus 的專案下，建立一個副檔名為 XUM 的檔案，接著按下檢查的按鈕，系統會抓取專案下所有.uml 與.di2 的檔案，並儲存至資料庫內，接著將 UML 的模型與 XUM 的整合模型進行比對，將相關的資訊寫入 XUM 檔案中，最後顯示給使用者。

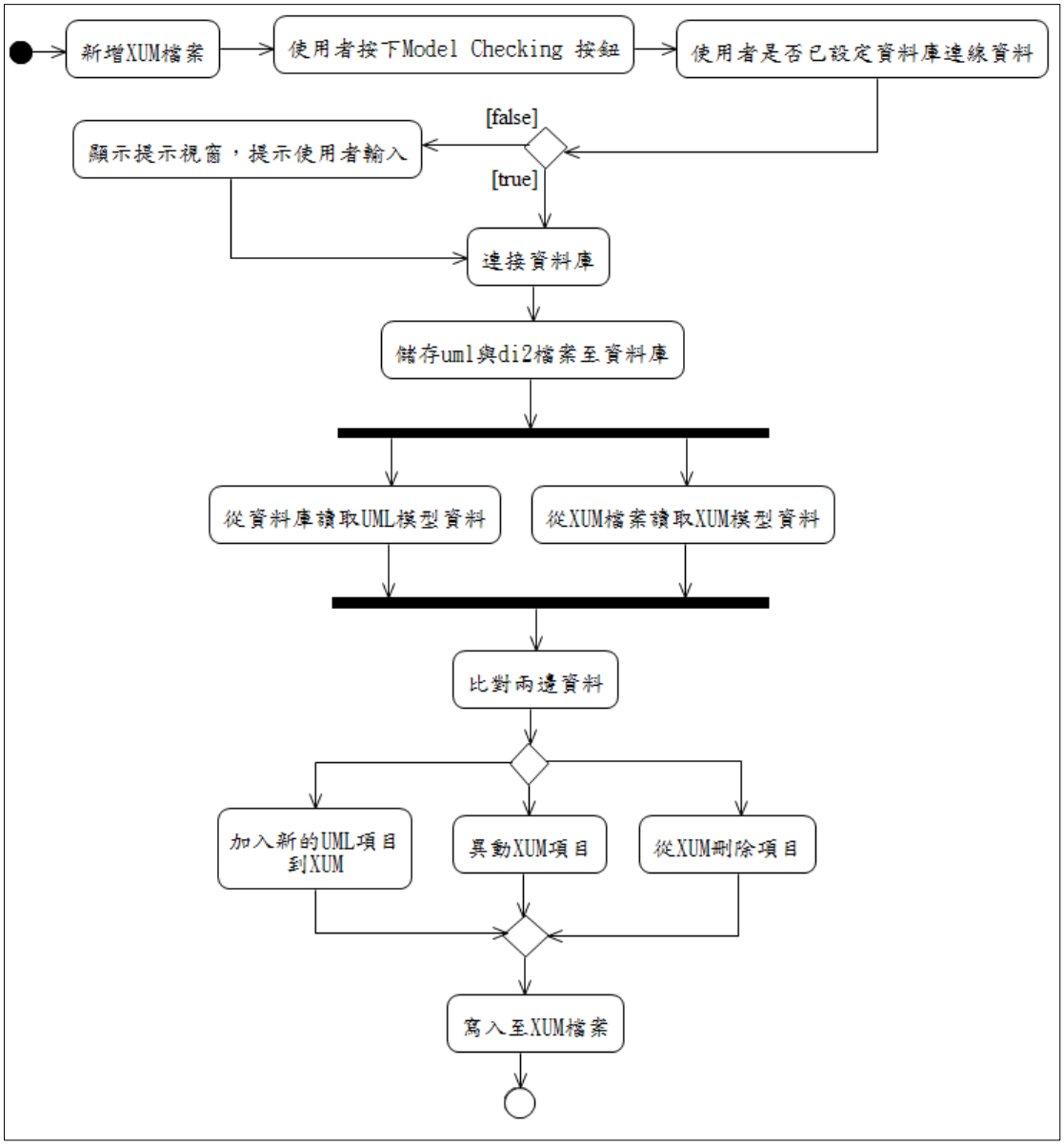

圖 二十三 系統活動圖

5.1.1. 加入新的 UML 項目到 XUM

針對不同的比對結果，會有不同的處理程序，當要加入一個新的 UML 項目到 XUM 的整合模型中，如圖 二十四活動圖所示，系統會先比對是否有相同之 UML 項目，例如新加入一個使用案例，則會比對類別圖、狀態圖、循序圖中，是否有名稱完全一致之項目，如有，則建立此一關聯，並根據其抽象層級，寫入對應的整合鏈結，假使比對後，未發現完全一致的 UML 項目，則會透過相似度的量測，提供可能符合之項目，由使用者自行決定是否要建立此一關聯。

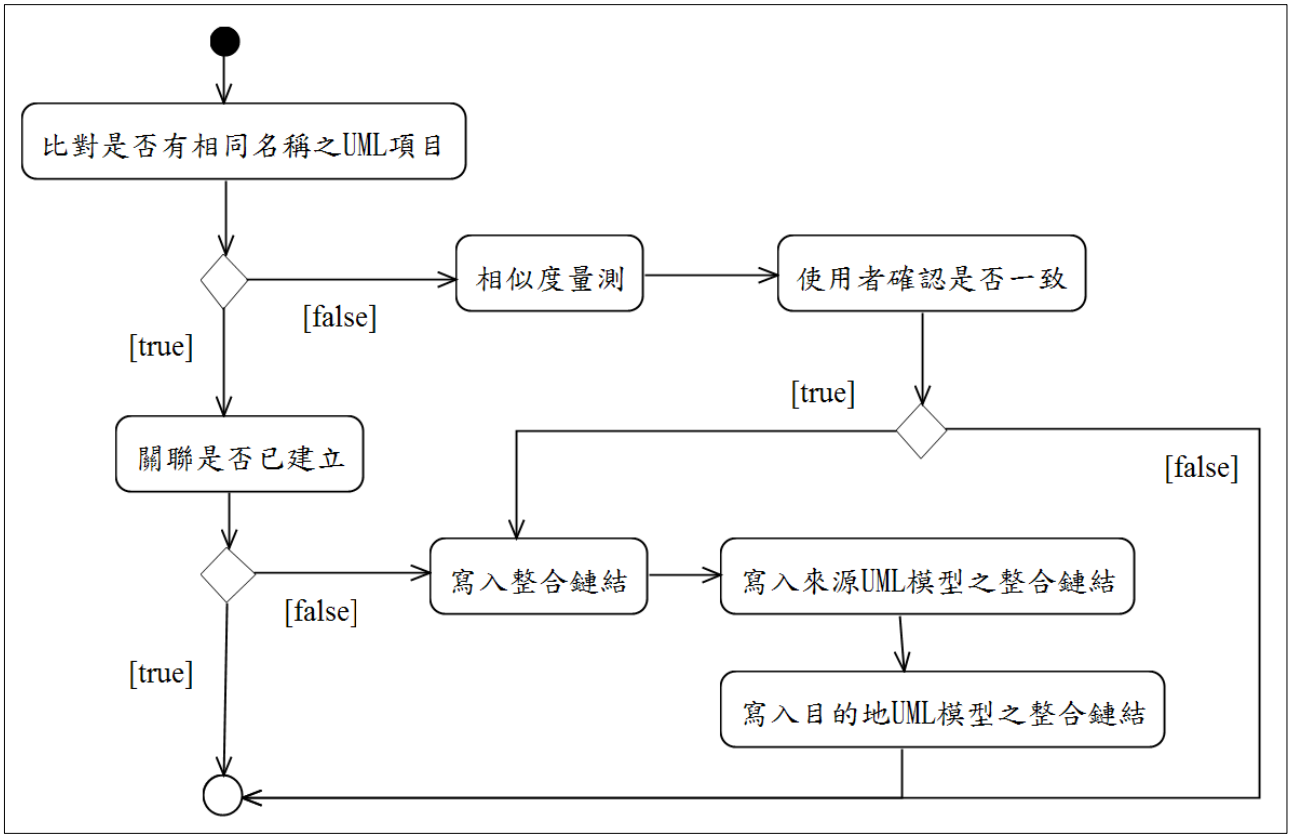

圖 二十四 加入新的 UML 項目到 XUM 之活動圖

5.1.2. 異動 XUM 項目

XMI 的規格以唯一的 ID 去識別不同的 UML 項目，當使用者對已塑模之模型進行修改名稱時，原本建立的關聯勢必也會有所異動，其處理的流程如圖 二十五之活動圖所示，會先檢查已建立之鏈結名稱是否與異動過之 UML 項目一致，如果不一致且此一鏈結是由使用者所建立，則會提示使用者進行確認，假使非使用者所建立，則會刪除不一致的鏈結，再重新檢查所有的 UML 項目。

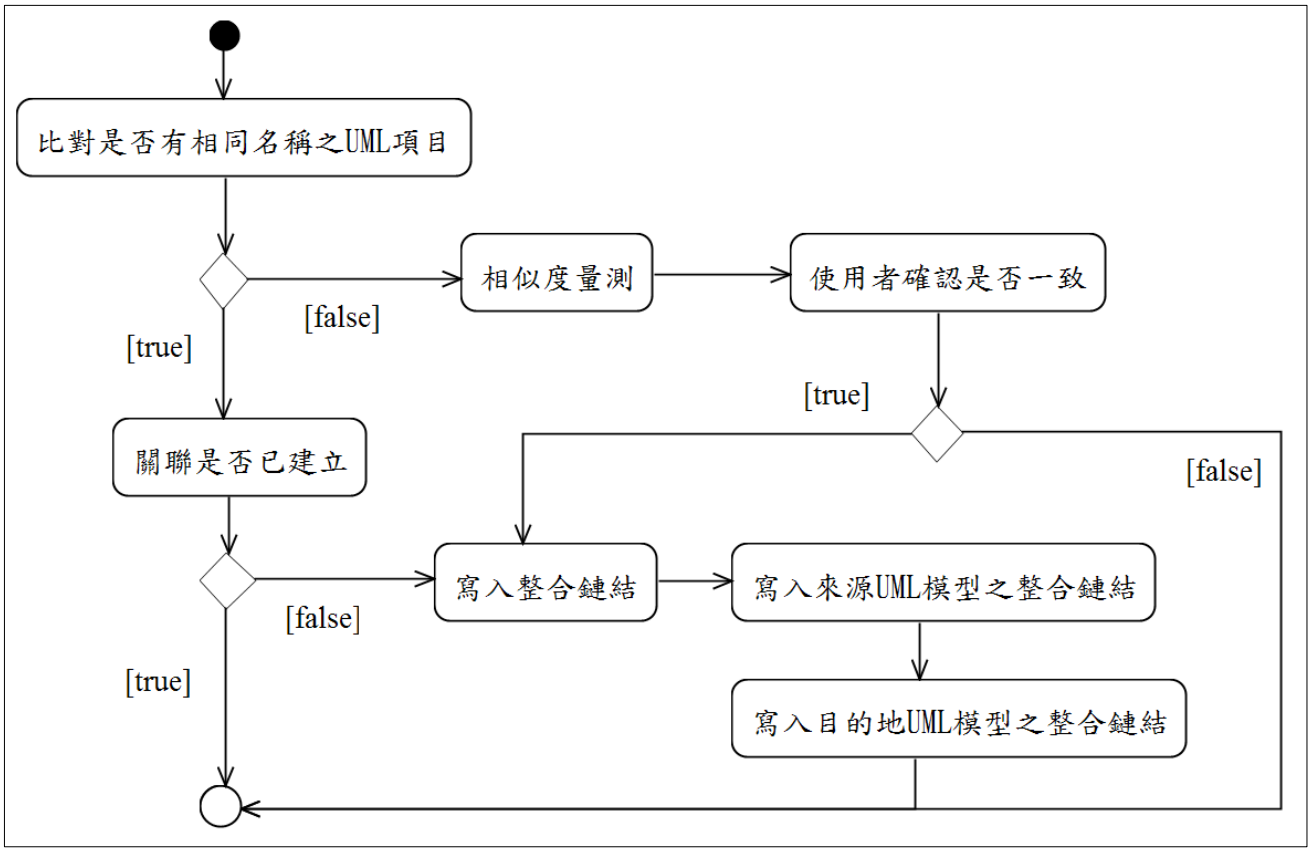

圖 二十五 異動 XUM 項目之活動圖

5.2.建立 XUM 檔案

為了要在 Papyrus 專案下建立 XUM 的檔案，使用者可以透過本系統之新增檔案的 Wizard，加入 XUM 的檔案，如圖 二十六所示。

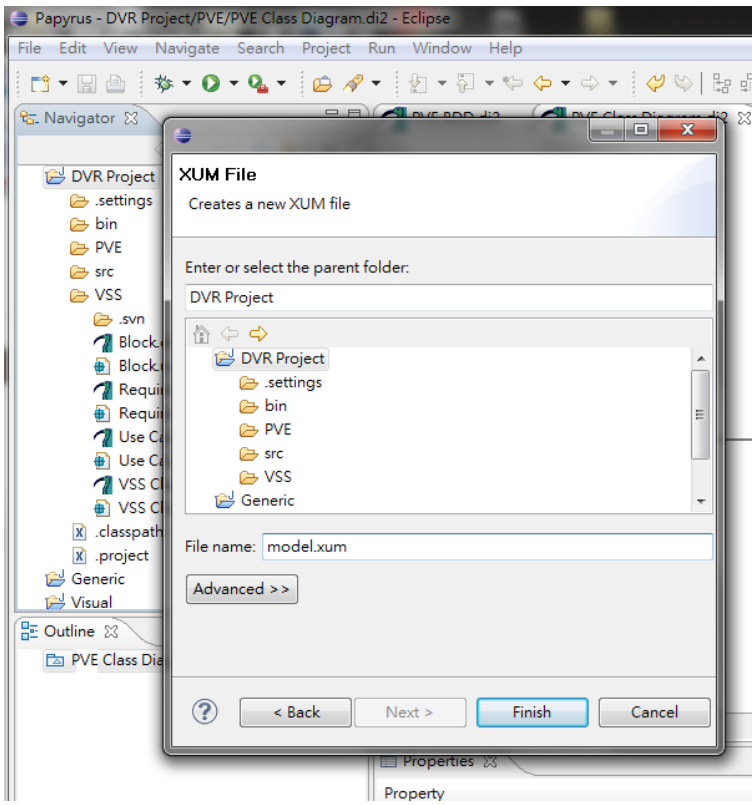

圖 二十六 新增 XUM 檔案 Wizard

5.3. 開啟編輯器

在 XUM 的檔案建立後雙擊此檔案，Eclipse 的環境會根據此副檔名開啟對應的編輯器或是工具列按鈕，如圖 二十七，本系統包含了編輯器、觀點等功能，在首次開啟時，其內容並無任何資料存在，使用者必須藉由點擊 Model Checking 的按鈕，進行分析，系統會抓取在此專案目錄下，所有.uml 與.di2 的檔案並儲存至資料庫內，再進一步寫入部分必要之資訊以及彼此之關聯到 XUM 的檔案中。

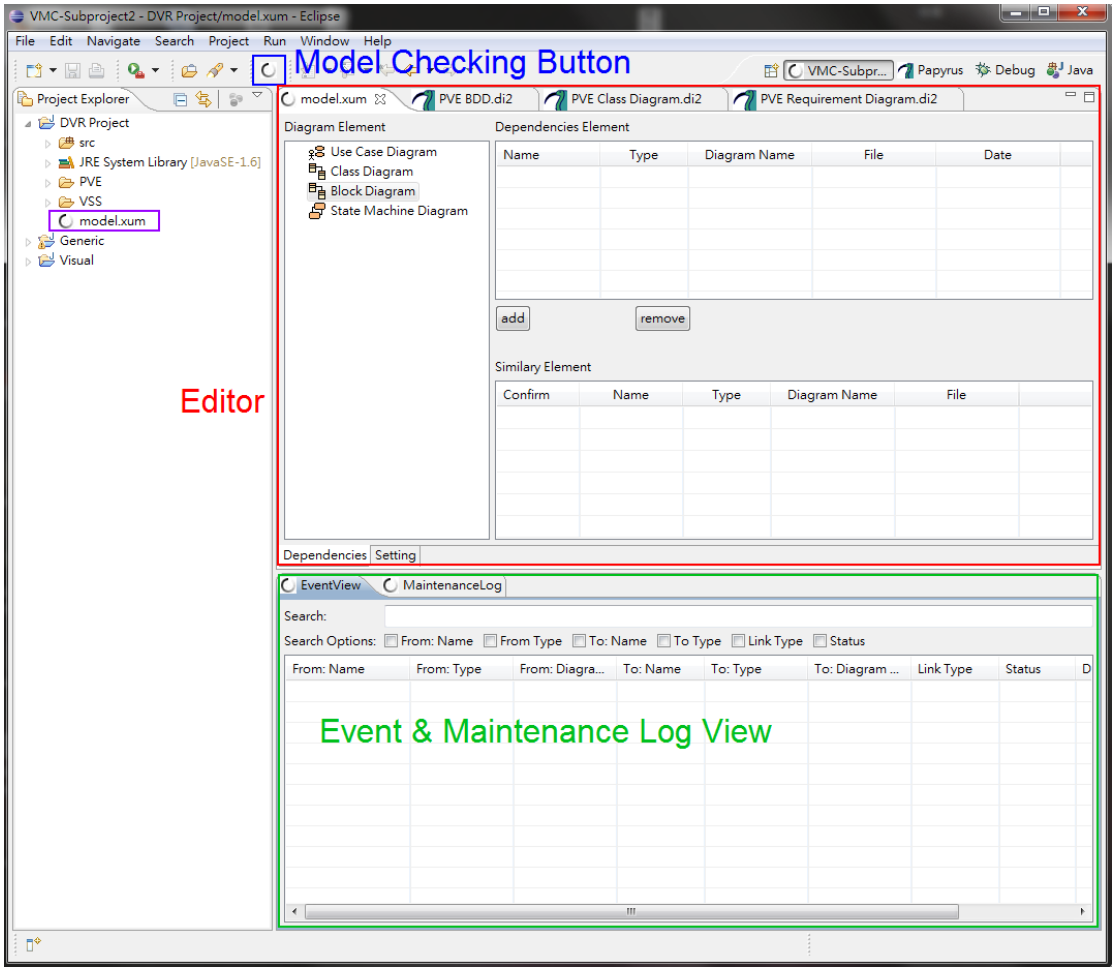

圖 二十七 系統執行畫面

5.4. 模型關聯的呈現

在模型分析完成後，編輯器會自動更新資料，如圖 二十八所示，左邊的樹狀結構顯示所有的 UML 項目，點擊其中任一項目，在右邊的上方表格顯示與此名稱一致之項目，右邊的下方表格，則是利用相似度比對，找出可能潛在結果，使用者可以在右下方的表格，透過勾選 Checkbox 進行確認。另外，考量到使用者可能需要進行手動變更的需求，使用者也可以透過 Add 的按鈕自行手動加入相關連性的項目，透過 Remove 的按鈕將不相關的項目移除，如圖 二十九所示。

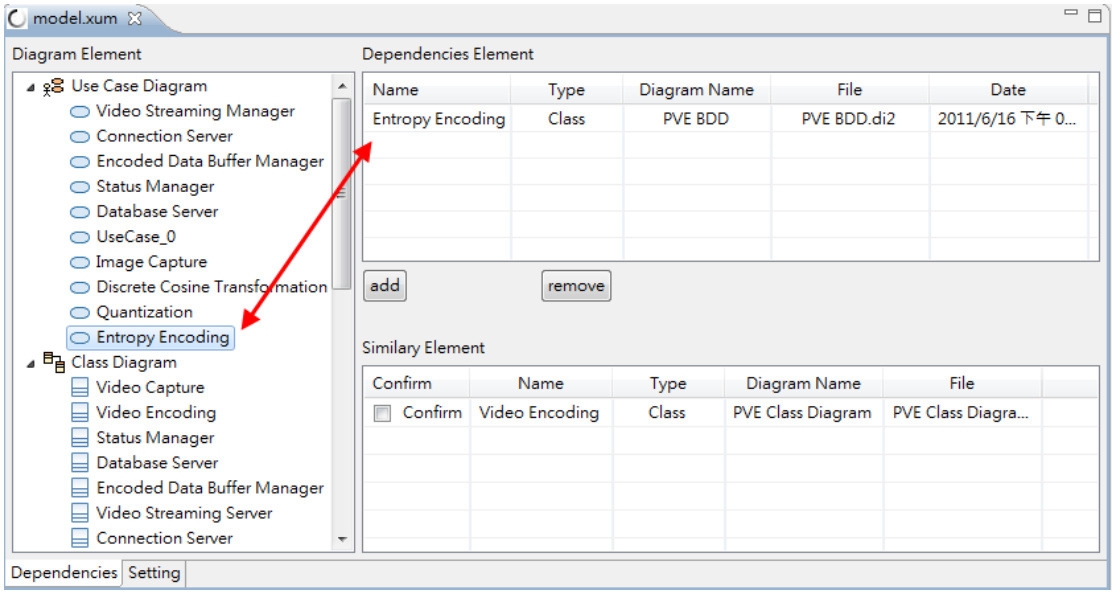

圖 二十八 模型的關聯

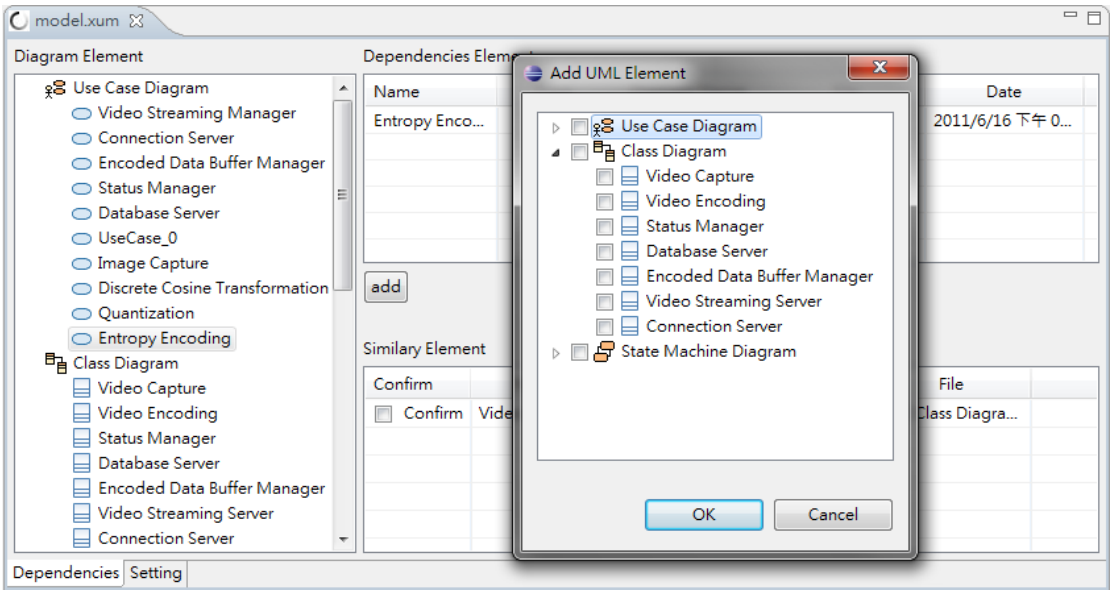

圖 二十九 手動修改關聯

5.5.鏈結記錄的呈現

系統分析模型的過程中，所發生的事件會自動的記錄下來，如圖 三十所示，由 Link Event 的 View 去顯示。使用者可以搜尋或是排序特定的欄位，來獲取不同的訊息，如圖 三十一，使用者經由搜尋 From: Type 的欄位，了解所有 Use case 與其他 UML 模型間的關係。

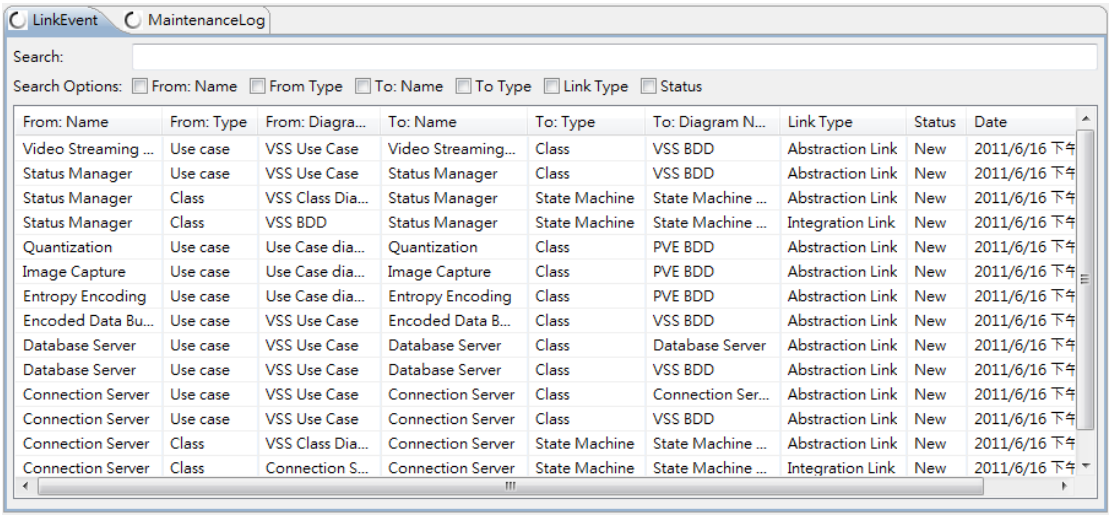

圖 三十 Link Event View

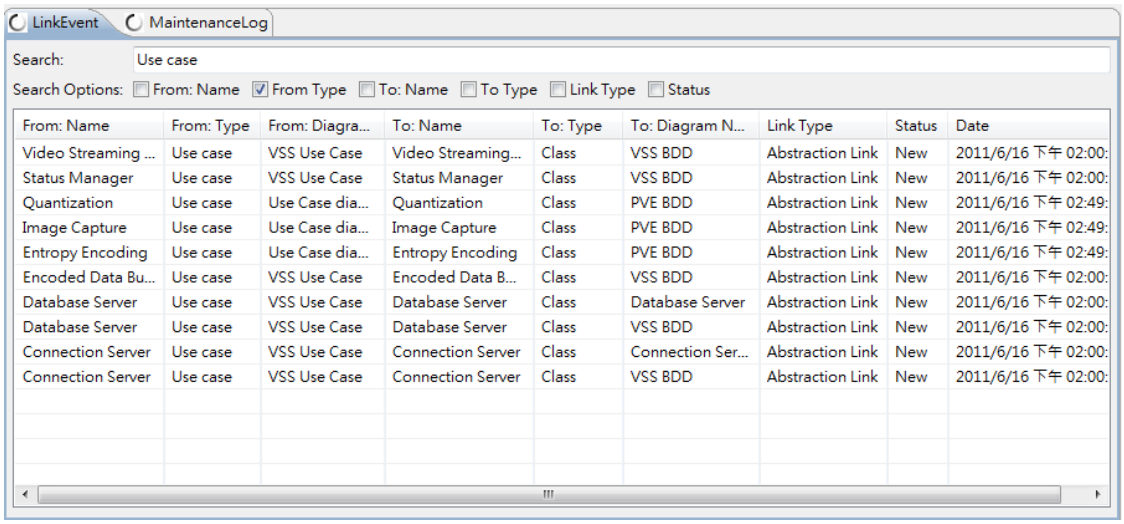

圖 三十一 搜尋特定 Link Event 欄位

圖 三十二則顯示了使用者針對 UML 模型修改的紀錄，這樣的紀錄有助於使用者了解前後版本的差異。

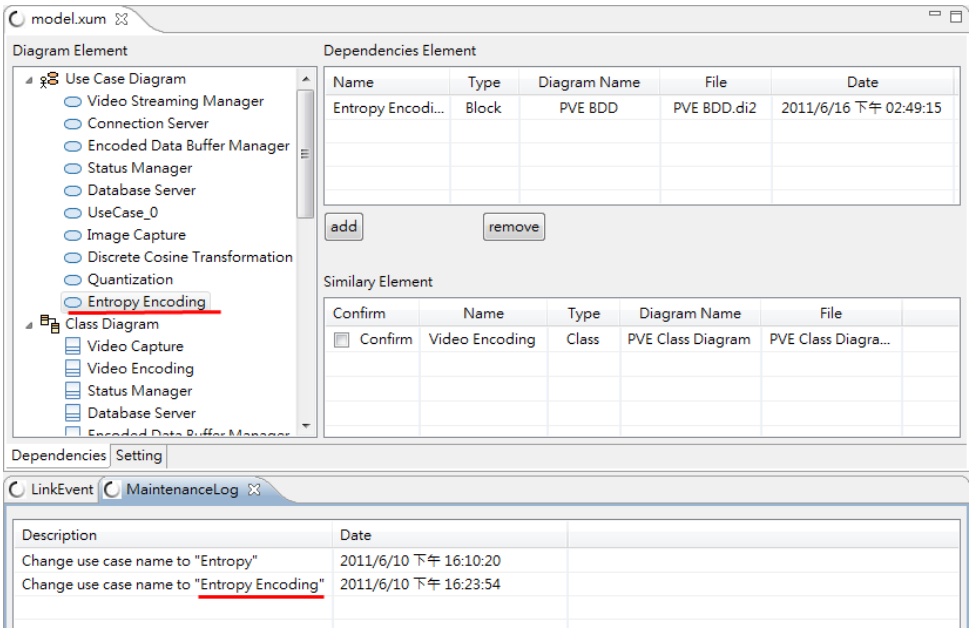

圖 三十二 Maintenance Log View

5.6.評估

在過去 Usman 等人針對不同的 UML 一致性檢查方法進行調查[32]，其中包含正規化、延伸 UML 表示或是無需任何中介表示的方法，並且解決不同的 UML 模型以及達到不同的一致性類型，但大多數的方法並非基於 MDA 的架構下。而 Zhou 等人[33]所建構的 Jasmine 工具也是基於 Eclipse 的環境下，但卻是以獨立的 Java 應用程式來執行，其主要關注於循序圖、活動圖、狀態圖，也分析 XMI 的標準，但缺少最常使用的類別圖。另外 Gongzheng 等人[34]，則是使用正規的語言進行狀態圖與循序圖之間的一致性檢查，主要關注於水平一致性與語意的一致性，提出幾項可以依循之規則，但也缺少最常使用的類別圖。

而本論文所提出之方法在性質上包含靜態與動態的 UML 模型，最大的特點是基於 MDA 的架構下所建置並符合 OMG 相關的標準，在 Eclipse 中的 EMF(Eclipse Modeling Framework)專案主要是用來實作 OMG 所定義的 MDA 架

構以及其相關的規格，如 MOF、XMI 等標準，而本論文所利用的塑模工具 Papyrus，則是基於 EMP 所開發的，我們透過分析這些被實作出的標準，建構出一個整合模型 XUM 來表示模型間的關聯性，以達到模型間的水平與垂直一致性，並建立鏈結能利於系統開發人員進行追蹤，初步以達成半自動化的分析，由於系統塑模過程中會包含部分的自然語言以及不同單字所組成的字串，大多數的比對還是需要人為的決定，所以我們利用相似度的量測，來建議系統開發人員這些潛在的可能性，並建構出外掛工具來輔助塑模的過程。

第六章 結論與未來工作

UML 幾乎已成為軟體塑模的標準語言，尤其 MDA 的興起，更強調模型的建立與轉換，這些模型到後來會經由程式碼生成產生實際的程式碼，因此確保塑模的正確性與一致性就變得更加的重要。

UML 提供系統開發人員針對系統的不同觀點進行塑模，每一種模型分別代表系統的不同面向，但 UML 的定義過於鬆散，並未詳細定義模型之間的關聯性以及一致性，且隨著系統設計不斷的細化以及演進，模型之間的不一致很容易就會產生，進而影響系統的開發過程。因此在本篇論文，我們分析 XMI 以及圖形交換格式，將所需的資訊擷取並儲存到 XUM 的模型，建立垂直與水平的一致性確認機制，針對不同的抽象層級，使用不同的鏈結型態進行串連，並且在分析的過程中，不同事件的發生都會被記錄下來，使得系統開發人員能發現系統開發過程中潛在的錯誤，達到軟體的追溯性，並在 Eclipse 的架構下實作一個輔助工具來達成此一目的。

在未來的工作，能在本論文之架構下，將更多的 UML 模型進行整合，針對不同的系統觀點，建立更完整的軟體追溯性，也可朝向系統文件的關聯建立，包含需求、設計等文件，另一方面，系統測試也佔了軟體生命週期的一大部分，測試階段的整合，更能全面的支援軟體開發的各個階段。

參考文獻

[1] W. B. Frakes and K. Kang, "Software Reuse Research: Status and Future", *IEEE Transactions on Software Engineering*, Vol. 31, No.7, pp. 529-536, 2005.

[2] E. Gamma, R. Helm, R. Johnson and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[3] B. Selic, "The Pragmatics of Model-Driven Development", *IEEE Software*, Vol. 20, No. 5, pp. 19-25, 2003.

[4] A. Mattsson, B. Lundell, B. Lings, and B. Fitzgerald, "Linking Model-Driven Development and Software Architecture: A Case Study", *IEEE Transactions on Software Engineering*, Vol.35, No. 1, pp. 83-93, 2009.

[5] O.C.Z. Gotel and A.C.W. Finkelstein, "An Analysis of the Requirements Traceability Problem", *Proceedings of the First IEEE International Conference on Requirements Engineering*, pp. 94-101, 1994.

[6] N. Aizenbud-Reshef, B.T. Nolan, J. Rubin and Y. Shaham-Gafni, "Model traceability", *IBM System Journal*, Vol. 45, No. 3, pp. 515–526, 2006.

[7] OMG, "MDA Guide", Version 1.0.1, 2004.

[8] I. Sommerville, *Software Engineering*, 9th edition, Addison-Wesley, pp.30-32, 2011.

[9] J. D. Poole, "Model-Driven Architecture: Vision, Standards And Emerging Technologies", *Proceedings of the European Conference on Object-Oriented Programming*, 2001.

[10] R. Soley and the OMG Staff Strategy Group, "Model Driven Architecture", Object Management Group White Paper Draft 3.2 – November 27, 2000.

[11] OMG, "Unified Modeling Language: Superstructure", Version 2.3, 2010.

[12] R.H. Bourdeau and B.H.C. Cheng, "A Formal Semantics for Object Model

Diagrams", *IEEE Transitions on Software Engineering*, Vol.21, No.10, pp.799-821, 1995.

[13] OMG, "UML 2.0 Diagram Interchange Specification", Version 1.0, 2006

[14] V. Rijsbergen and C. Joost, *Information Retrieval*, Butterworths, pp.24-25, 1979.

[15] H. Deitel, P. Deitel, T. Nieto, T. Lin and P. Sadhu, *XML How to Program*, Prentice Hall: Upper Saddle River NJ, pp.1-9, 2001.

[16] XML Metadata Interchange (XMI) Version 2.1.1, http://www.omg.org/spec/XMI/2.1.1/PDF/index.htm, Retrieved 2010-7-29.

[17] X. M. Yang and H. Dai, "Native XML Database design and realization based on MDA", *Proceedings of the 2nd IEEE International Conference on Information Management and Engineering*, pp.738-741, 2010.

[18] T. Mens, R. V. D. Straeten and J. Simmonds, "A Framework for Managing Consistency of Evolving UML Models", *Software Evolution with UML and XML*, chapter 1. Idea Group Inc., pp. 1-31, 2005.

[19] R. V. D. Straeten, "Inconsistency Management in Model-Driven Engineering (An Approach using Description Logics) ", PhD thesis, Department of Computer Science, Vrije Universiteit Brussel, Belgium, 2005.

[20] A. Egyed, "Scalable Consistency Checking between Diagrams – The VIEWINTEGRA Approach", *Proceedings of the 16th IEEE International Conference on Automated Software Engineering*, pp. 387-390, 2001.

[21] C. Lu, W.C. Chu, C. Chang, D. Yang, and W. Lian, "Integrating Diverse Paradigms in Evolution and Maintenance by an XML-Based Unified Model", *Journal of Software Maintenance*, Vol.15, No.3, pp.111-114, 2003.

[22] J. Simmonds, R.V.D. Straeten, V. Jonckers, and T. Mens, "Maintaining Consistency between UML Models Using Description Logic", *Proceedings of*

Langages et Modèles à Objets, pp. 231-244, 2004.

[23] U. Bellur and V. Vallieswaran, "On OO Design Consistency in Iterative Development", *Proceedings of the Third International Conference on Information Technology: New Generations*, IEEE, pp. 46-51, 2006.

[24] A. Egyed, "UML/Analyzer: A Tool for the Instant Consistency Checking of UML Models", *Proceedings of the 29th International Conference on Software Engineering*, pp. 793-796, 2007.

[25] B. Berenbach and T. Wolf, "A Unified Requirements Model; Integrating Features, Use Cases, Requirements, Requirements Analysis and Hazard Analysis", *Proceedings of the International Conference on Global Software Engineering*, pp.197-203, 2007.

[26] P. Mader, O. Gotel and I. Philippow, "Getting Back to Basics: Promoting the Use of a Traceability Information Model in Practice", *Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering*, pp. 21-25, 2009.

[27] W. B. Croft, D. Metzler and T. Strohman, *Search Engines Information Retrieval in Practice*, Addison-Wesley, 2009.

[28] E. Clayberg and D. Rubel, *Eclipse Plug-ins*, Addison-Wesley, pp.132, 2009.

[29] T. Reenskaug, "MVC XEROX PARC 1978-79", http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html, Retrieved 2011-01-12.

[30] F. Simeoni, D. Lievens, R.C.H. Connor, and P. Manghi, "Language Bindings to XML", *IEEE Internet Computing*, Vol.7, No.1, pp.19-27, 2003.

[31] E. Ort and B. Mehta, "Java Architecture for XML Binding (JAXB)", http://www.oracle.com/technetwork/articles/javase/index-140168.html, Retrieved 2010-06-14

[32] M. Usman, A. Nadeem, T.H. Kim and E.S. Cho, "A Survey of Consistency Checking Techniques for UML Models", *Proceedings of Advanced Software Engineering and Its Applications*, pp. 57-62, 2008.

[33] Z. Zhou, L. Wang, Z. Cui, X. Chen and J. Zhao, "Jasmine: A Tool for Model-Driven Runtime Verification with UML Behavioral Models", *Proceedings of the 11th IEEE High Assurance Systems Engineering Symposium*, pp. 487-490, 2008.

[34] G. Lu and G. Zhang, "An approach to check the consistency between the UML 2.0 Dynamic Diagrams", *Proceedings of the 5th International Conference on Computer Science & Education*, pp. 1913-1917, 2010.

附錄 A. Plug-in 安裝

1. 安裝於 Eclipse 3.5 版本下

   開啟 Eclipse 所在目錄,將封裝好之外掛 Jar 檔,直接丟入 plugins 資料夾中，重新啟動 Eclipse 即可。

2. 安裝於 Papyrus 1.12.3 版本下

   同 Eclipse 3.5 版本，將 Jar 檔丟入 plugins 資料夾中，另需手動修改 Papyrus 目錄下 configuration\org.eclipse.equinox.simpleconfigurator\bundles.info 此一檔案，手動加入以下文字，重開 Papyrus 即可。

   com.vmc.mum,1.0.0.201105061040,plugins/com.vmc.mum_1.0.0.201105061040.jar,4,false

附錄 B. XUM Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema elementFormDefault="qualified"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="XUM">
     <xsd:complexType>
          <xsd:sequence>
               <xsd:element name="requirement" type="requirementPhase"
                    maxOccurs="1" minOccurs="1">
               </xsd:element>
               <xsd:element name="design" type="designPhase"
                    maxOccurs="1" minOccurs="1">
               </xsd:element>
               <xsd:element name="implementation" type="implementPhase"
maxOccurs="1" minOccurs="1"/>
               <xsd:element name="unification_link"
                    type="unificationlinkType" maxOccurs="1" minOccurs="1">
               </xsd:element>
               <xsd:element name="data_dictionary" type="wordType"
                    maxOccurs="1" minOccurs="1">
               </xsd:element>
               <xsd:element name="database" type="dbType"></xsd:element>
               <xsd:element name="code_path" type="xsd:string"></xsd:element>
          </xsd:sequence>
          <xsd:attribute name="project_name" type="xsd:string"></xsd:attribute>
     </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="xumType">
     <xsd:sequence>
          <xsd:element name="project_name" type="xsd:string"></xsd:element>
          <xsd:element name="model_name" type="xsd:string"></xsd:element>
          <xsd:element name="requirement" type="usecaseType"></xsd:element>
          <xsd:element name="design" type="xsd:string"></xsd:element>
          <xsd:element name="implement" type="xsd:string"></xsd:element>
          <xsd:element name="unification_link" type="unificationlinkType">
```

```
          </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="usecaseType">
      <xsd:sequence>
          <xsd:element name="actor" type="usecaseElementType"
              maxOccurs="unbounded" minOccurs="0">
          </xsd:element>
          <xsd:element name="usecase" type="usecaseElementType"
              maxOccurs="unbounded" minOccurs="0">
          </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="unificationlinkType">
      <xsd:sequence>
          <xsd:element name="abstraction_link"
              type="abstraction_linkType" maxOccurs="unbounded"
              minOccurs="0">
          </xsd:element>
          <xsd:element name="integration_link"
              type="integration_linkType" maxOccurs="unbounded"
              minOccurs="0">
          </xsd:element>
          <xsd:element name="sourcecode_link" type="sourcecode_linkType"
maxOccurs="unbounded" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="abstraction_linkType">
      <xsd:attribute name="date" type="xsd:string"></xsd:attribute>
        <xsd:attribute name="userdefined" type="xsd:string"></xsd:attribute>
        <xsd:attribute name="toType" type="xsd:string"></xsd:attribute>
      <xsd:attribute name="to" type="xsd:string"></xsd:attribute>
      <xsd:attribute name="fromType" type="xsd:string"></xsd:attribute>
      <xsd:attribute name="from" type="xsd:string"></xsd:attribute>
      <xsd:attribute ref="id"></xsd:attribute>
    </xsd:complexType>
    <xsd:attribute name="id" type="xsd:string"></xsd:attribute>
    <xsd:complexType name="integration_linkType">
```

```
     <xsd:attribute name="date" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="userdefined" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="toType" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="to" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="fromType" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="from" type="xsd:string"></xsd:attribute>
     <xsd:attribute ref="id"></xsd:attribute>
    </xsd:complexType>
    <xsd:element name="abstraction_link_id" type="xsd:string"></xsd:element>
    <xsd:element name="integration_link_id" type="xsd:string"></xsd:element>
    <xsd:complexType name="lifelineElementType">
     <xsd:sequence>
          <xsd:element ref="integration_link_id" maxOccurs="unbounded"
               minOccurs="0">
          </xsd:element>
          <xsd:element ref="abstraction_link_id" maxOccurs="unbounded"
               minOccurs="0">
          </xsd:element>
     </xsd:sequence>
     <xsd:attribute name="from_file" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="from_diagram" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="name" type="xsd:string"></xsd:attribute>
     <xsd:attribute ref="id"></xsd:attribute>
    </xsd:complexType>
    <xsd:complexType name="classElementType">
     <xsd:sequence>
          <xsd:element ref="abstraction_link_id" maxOccurs="unbounded"
               minOccurs="0">
          </xsd:element>
          <xsd:element ref="integration_link_id" maxOccurs="unbounded"
               minOccurs="0">
          </xsd:element>
          <xsd:element name="sourcecode_link_id" type="xsd:string"
maxOccurs="1"
               minOccurs="0">
          </xsd:element>

     </xsd:sequence>
```

```
     <xsd:attribute name="from_file" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="from_diagram" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="name" type="xsd:string"></xsd:attribute>
     <xsd:attribute ref="id"></xsd:attribute>
    </xsd:complexType>
    <xsd:complexType name="wordType">
     <xsd:attribute name="id" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="name" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="aliase" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="description" type="xsd:string"></xsd:attribute>
    </xsd:complexType>
    <xsd:complexType name="bddElementType">
     <xsd:sequence>
          <xsd:element ref="abstraction_link_id" maxOccurs="unbounded"
               minOccurs="0">
          </xsd:element>
          <xsd:element ref="integration_link_id" maxOccurs="unbounded"
               minOccurs="0">
          </xsd:element>
          <xsd:element name="sourcecode" type="xsd:string"
               maxOccurs="1" minOccurs="0">
          </xsd:element>
     </xsd:sequence>
     <xsd:attribute name="from_file" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="from_diagram" type="xsd:string"></xsd:attribute>
       <xsd:attribute name="base_class" type="xsd:string"></xsd:attribute>
       <xsd:attribute name="name" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="id" type="xsd:string"></xsd:attribute>
    </xsd:complexType>
    <xsd:complexType name="designPhase">
     <xsd:sequence>
          <xsd:element name="class_diagram" type="classType"
               maxOccurs="1" minOccurs="1">
          </xsd:element>
          <xsd:element name="block_diagram" type="bddType" maxOccurs="1"
minOccurs="1"></xsd:element>
          <xsd:element name="sequence_diagram" type="lifelineType"
               maxOccurs="1" minOccurs="1">
```

```
        </xsd:element>
        <xsd:element name="statemachine_diagram" type="stateType"
maxOccurs="1" minOccurs="1"></xsd:element>
     </xsd:sequence>
    </xsd:complexType>
     <xsd:complexType name="requirementPhase">
     <xsd:sequence>
        <xsd:element name="usecase_diagram"
            type="usecaseType" maxOccurs="1" minOccurs="1">
        </xsd:element>
        <xsd:element name="requirement_diagram" type="requirementType"
maxOccurs="1" minOccurs="1"></xsd:element>
     </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="requirementType">
     <xsd:sequence>
        <xsd:element name="requirement_element" type="requirementElementType"
maxOccurs="unbounded" minOccurs="0"></xsd:element>
     </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="usecaseElementType">
     <xsd:sequence>
        <xsd:element ref="abstraction_link_id" maxOccurs="unbounded"
            minOccurs="0">
        </xsd:element>
        <xsd:element ref="integration_link_id" maxOccurs="unbounded"
            minOccurs="0">
        </xsd:element>
     </xsd:sequence>
     <xsd:attribute name="from_file" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="from_diagram" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="name" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="id" type="xsd:string"></xsd:attribute>
    </xsd:complexType>
    <xsd:complexType name="requirementElementType">
     <xsd:sequence>
        <xsd:element ref="abstraction_link_id"></xsd:element>
        <xsd:element ref="integration_link_id"></xsd:element>
```

```
     </xsd:sequence>
     <xsd:attribute name="base_class" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="from_diagram" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="from_file" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="date" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="name" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="id" type="xsd:string"></xsd:attribute>
    </xsd:complexType>
    <xsd:complexType name="classType">
     <xsd:sequence>
          <xsd:element name="class" type="classElementType"
maxOccurs="unbounded" minOccurs="0"></xsd:element>
     </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="bddType">
     <xsd:sequence>
          <xsd:element name="block" type="bddElementType" maxOccurs="unbounded"
minOccurs="0"></xsd:element>
     </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="lifelineType">
     <xsd:sequence>
          <xsd:element name="lifeline" type="lifelineElementType"
maxOccurs="unbounded" minOccurs="0"></xsd:element>
     </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="stateType">
     <xsd:sequence>
          <xsd:element name="statemachine" type="stateElementType"
maxOccurs="unbounded" minOccurs="0"></xsd:element>
     </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="dbType">
       <xsd:attribute name="password" type="xsd:string"></xsd:attribute>
       <xsd:attribute name="username" type="xsd:string"></xsd:attribute>
       <xsd:attribute name="dbname" type="xsd:string"></xsd:attribute>
       <xsd:attribute name="host" type="xsd:string"></xsd:attribute>
    </xsd:complexType>
```

```
    <xsd:complexType name="stateElementType">
     <xsd:sequence>
          <xsd:element ref="abstraction_link_id" maxOccurs="unbounded"
               minOccurs="0">
          </xsd:element>
          <xsd:element ref="integration_link_id" maxOccurs="unbounded"
               minOccurs="0">
          </xsd:element>
     </xsd:sequence>
     <xsd:attribute name="from_file" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="from_diagram" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="name" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="id" type="xsd:string"></xsd:attribute>
    </xsd:complexType>
    <xsd:complexType name="codeType">
     <xsd:sequence>
          <xsd:element name="sourcecode_link_id" type="xsd:string" />
     </xsd:sequence>
     <xsd:attribute name="name" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="id" type="xsd:string"></xsd:attribute>
    </xsd:complexType>
    <xsd:complexType name="implementPhase">
     <xsd:sequence>
          <xsd:element name="sourcecode" type="codeType" maxOccurs="unbounded"
minOccurs="0"/>
     </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="sourcecode_linkType">
     <xsd:attribute name="date" type="xsd:string" />
     <xsd:attribute name="userdefined" type="xsd:string"></xsd:attribute>
     <xsd:attribute name="to" type="xsd:string" />
     <xsd:attribute name="from" type="xsd:string" />
     <xsd:attribute name="id" type="xsd:string" />
    </xsd:complexType>
</xsd:schema>
```