東海大學資訊科學研究所

碩士論文

指導教授：朱正忠

使用 XML 製作之通用性資料交換模型

Universal Data Interchange Model using XML

研究生：吳明勳

中華民國八十九年六月

中文摘要

在軟體工程的領域中，資料模型間的轉換 (Transform) 與跨模型間的資料交換 (Data Interchange) 行為，一直都是令人相當頭痛的問題。使用者或軟體工程師為了要更有效率的發展系統，他們需要將在系統或輔助工具上的資訊轉移至另一個系統或輔助工具，而有時這些資料的交換甚至是動態的或者是需要互動的。在過去由於發展成本與地域性限制等問題，要提供一個符合各種使用/設計者所要求的工具以及資料庫的特性的共用介面，幾乎是不可能存在的。近年來更因為 Internet 與分散式資料處理系統的方興未艾，更凸顯其問題的重要性。

雖然一直有一些研究在進行著，然而直至現階段仍尚未有任何一種工具或資料模型理論能夠滿足這些需求，例如 CDIF (CASE Data Interchange Format) 等。目前有一些資料交換的標準，其主要是針對目前眾多的 CASE Tools 間的資料轉換而設計，但其所適用系統的格式與標準仍有諸多限制，對現有許多規格不符的軟體系統，助益不大。因此，在這裏所強調的資料模型間的轉換與跨模型間的資料交換行為的重要性，主要目的是為了提高對於訂定 Data Interchange Protocol 之重視，如此將有助於降低重整工程 (Re-Engineering) 的複雜性與相關之成本。

在本論文中，我們提出一個稱為通用性資料交換模型 (Universal Data Interchange Model, UDIM)，其目的在希望能提供一個資料交換的解決方案。UDIM 是以 Meta Data Model 的外掛封裝為基礎，本論文發展一套能夠提供不同資料格式間轉換的環境，希望能藉由此一研究，提供使用者來處理不同系統間資料的重整、萃取、分析、與封裝，進而將其轉入到另一個系統中，以增加資料的再利用性。

近年來成為標準的 XML 提供了處理資料的相關機制，恰巧提供了一個解決方法，本篇論文便是以 XML 為基礎，將之應用於所定義出的通用性的資料交換

模型之中，並實做一套雛形工具，以方便系統維護者進行不同系統資料間的移轉工作；這樣一來，就不必像傳統的做法需要個別開發其轉換程式，因此不但能增進系統開發的效率、資料的再利性、降低軟體系統開發的成本、並可使資料的維護更加容易。此外，在本論文中我們以 Java 語言來開發相關之程式，並以病患醫療資料來作為系統之測試與驗證之資料。

總括來說，本系統之主要的模組與功能包含有：

(1). 檔案轉換規則處理系統 (File Transform Rules System) ：負責處理來源資料庫系統之資料轉換規則，其主要之目的是將使用者所輸入的檔案格式 (File Format) 與資料綱目 (Data Schema) 的內容加以處理，並以 DOM (Document Object Model) 的結構將之傳給資料轉換系統來作為轉換之依據。

(2). 資料轉換處理系統 (Data Transform Process System) ：主要是針對資料庫系統之實體檔案作資料轉換 (Data Transform) 與萃取的處理，並將得到的資料依其原有之結構與關係，重新編寫成 XML 之格式，並透過 DOM 來將其轉換為樹狀的資料型態，以利於提供資料之交換與再利用 (Reuse) 的處理。

(3). 資料交換處理系統 (Data Interchange Process System) ：將從資料庫檔案中經萃取而重新整合過的資料，透過 HTML、CSS (Cascading Style Sheet) 等描述語言、介面或程式，進而傳送給其它之目的應用系統，以完成資料交換之工作。

Abstract

In Software Engineering, data and data models defined in one system are usually not transferable or interchangeable to another. In order to develop systems efficiently, sometimes users and software engineers who manipulate the systems need to move information from one tool or system to another, that is, exchanging data between repositories with different formats. This kind of exchanging may even happen dynamically and interactively. In the pass, to solve the problem with providing common interface between all tools and repositories is infeasible due to the limits of development-cost and geographical issues. And this demand is become more and more essential in modern software lifecycle, through the ascendancy of internet activities and the development of distributed data processing systems.

Although some recent researches have been proceeding, there are still little tools and data models which satisfy all the requirements in this domain. For example, the CDIF project, it is designed for data interchange between different CASE tools, but establishes many format and standard constrains to the adapted systems. In other word, it can make no help to old systems which haven’t good specifications on itself. So, we focus on data model interchange and the behaviors among the models, the goal here is to emphasize how to define a data interchange protocols to apply into the cross-model data sharing. If not so, the cost of re-engineering will be extremely expensive.

In this thesis, we propose the UDIM (Universal Data Interchange Model) using XML. UDIM is approach based on Meta Data Model, we plan to develop an environment for data interchange between different formats. In this environment, data activities between models like re-engineering, model retrieving, analysis,

encapsulation, model optimizing and data transformation will proceed. The architecture and modules of UDIM is listed below:

1. File Transform Rules Process: XML Parser, File Transform Rules Generator.
2. Data Transform Process: File Transform Processor, Data Integrator
3. Data Interchange Process: Data Interchanger Processor

Keywords: Data Interchange, Software Re-engineering, Data Reuse, XML (extensible Markup Language), DOM (Document Object Model)

目錄

圖目錄

表目錄

第一章 緒論

在軟體工程的領域中，資料模型間的轉換與跨模型間的資料交換行為，一直都是相當重要的問題，不同系統以及開發工具或環境間擁有的資訊通常不會和另一個系統、工具或環境要求的一樣，而使用者或軟體工程師為了要更有效率的發展系統，他們需要將在系統或輔助工具上的資訊轉移至另一個系統或輔助工具，將資訊在輔助工具與資訊庫之間或是資訊庫與資訊庫之間的轉移，若使用者要使用多種輔助工具的時候，有時這些資料的交換甚至是動態的或是需要互動，例如“round-trip”engineering 等。雖一直有一些研究在進行著，然而直至現階段仍尚未有任何一種工具或資料格式理論能夠滿足這個需求的所有問題；因此，我們在此強調資料模型間的轉換與跨模型間的資料交換行為的重要性，另一重要目的是為了提高國人對於訂定 Data Interchange Protocol 之重視，否則以後重整工程 (Re-Engineering) 的成本就會非常高。

由於不同的輔助工具或是系統，都有其定義資料模型的特定格式，而這些不同的格式卻沒有一個共通的標準或轉換方式，為了再使用 (Reuse) 這些資料，以致於要投入人力、金錢重新去分析，以人工的方式轉換成另一種系統或輔助工具的格式，如此不但延誤軟體開發時程，也增加了開發的成本，這個現象在現今由於 Internet 與分散式資料處理系統的方興未艾，其問題更形凸顯。有感於目前軟體業界對這種不同格式資料間轉換的需求與空白，因此本論文提出一個通用性資料交換模型 (Universal Data Interchange Model, UDIM)，其包含了相關的轉換工具環境及格式，以具可攜性、開放性的機制來提供設計人員在系統開發與使用過程中，一套完整、便利的資料交換環境，俾以增進軟體系統的效率與降低維護成本。

1.1 研究動機與目的

在系統工程師開發一套整合系統時，往往會使用到一些輔助工具或面臨一些

其他系統所提供之資料環境，然而這些輔助工具 (CASE Tools) 或系統環境的資料模型 (Data Model) 表達方式不盡相同，每一個輔助工具或系統環境所擁有的資訊很有可能不會和別的輔助工具或系統環境要求的一樣，而整合系統工程師為了要更有效率的發展系統，他們需要將在這些輔助工具或系統上的資訊轉移至另一個輔助工具或系統，或將資訊在輔助工具與資訊庫之間或是資訊庫與資訊庫之間的轉移，有時這些資料的交換甚至是動態的或是需要互動。

如果要提供一個符合各種使用/設計者所要求的工具以及資料庫的特性的共用介面，這種東西在以往幾乎是不可能存在的，最簡單的原因就是發展的成本。一般而言，系統工程師在開發整合性系統時，最大的問題就是他們需要重複利用他們原本在其它工具上所創造的資料，其原因如下：

- 由於使用者通常不會考慮軟體的整個長程生命週期，從策略性的計劃到維護，包括專案和品質的管理等。所以使用者往往被強迫更改使用工具，當然其原有的資料一定要再利用。
- 即使已經有了不錯的整合工具組，但很有可能並不是其中的每個成員都是該領域最好的，市場上可能有工具比其中某些成員所提供的更好，或是能為使用者解決特定的問題。因此使用者會想去使用這些較好的工具，甚至這些工具已是專案中非計畫中卻不可取代的一部份，此時，若沒有良好的內部合作機制，使用者很可能使用到各種工具，或是根本沒有模型化工具可供使用。
- 當一個組織不只使用一個方法或程序時，就像是在異質化的環境下一樣，單一工具與格式是無法滿足所有需求的，也因此工具與格式的配合及內部的合作機制是必需的。
- 當系統在發展時，使用者必須經常存取所有設計者所用的 CASE 及其相關工具、模型等發展文件，除非使用者所使用的工具和設計者所使用的工具是一模一樣，這將是一個很大的負擔。

- 倘若某個大組織有將所有專案製品記錄保存達幾十年以上，我們可以假設今天的工具到那時候已不能再使用，那我們要如何尋找解決升級的替代方案呢?
- 在大型的網路專案或分散式系統中，可能分佈在很多的區域，要所有的參與者都選擇同一個工具組是不太可能的，所以要如何讓他們能夠互相溝通呢?

綜合以上問題，當不同系統或工具間浮現關於資料共享或資料交換的需求時，若沒有特別的軟體介面用來轉移資訊，則除了每每費時費力重新建立模型輸入資料外，似乎也沒有更好的方法了。但這是個很費功的解決方案，因為重新建立與輸入可能會帶來錯誤；更由於操作者需要精通兩方面的工具且懂得如何轉換資料，這不適合同步工程和團體分工合作，所以會需要較昂貴或稀有的資源。

而軟體工程中的資料交換，可使用不同工具間的機制來達到資料交換的目的，使彼此的資料能共同分享與重覆使用，即使未來的環境有所變遷，只要使用的電腦輔助軟體工具有此資料交換格式的支援，便可將原有的資料順利移植到新的電腦環境或平台。但是直至現階段仍尚未有任何一種工具或資料格式能夠滿足這個需求的所有問題：例如 CDIF 等。而這些標準大都只是針對目前眾多的 Case Tools 間的資料轉換而設計，其間對所適用系統的格式與標準仍有諸多的限制，對現有許多規格不符的系統，助益不大，然而在目前資料電子化的趨勢下，不同格式的資料交換需求大增，所以，提供一套跨模型的資料交換的規則便是當務之急。

目前的平台多樣化，通訊協定定義不同，各資料庫管理系統 (DBMS) 間不同的介面，資料便在這麼多不同因素的應用系統之中傳遞。當資訊作業想要達到減少人為操作因素、人為錯誤及時間延誤；進而達到自動化 (Automation) 作業，以上種種現象，如能透過統一介面，配合現有資訊架構及整合作業流程，達到資

料交換 (Data Exchange) 的完整規劃為一不可或缺的成功因素。為達到上述之要求，本論文研究以 XML (eXtensible Markup Language) 為核心、並以 Java 程式語言為開發語言，提出一套資料交換系統的方案，將各種不同的資料模組的結構與 Meta Data Model 的對應、互換關係，以 XML 為表示方式，透過 DOM 技術來予以處理，進而成為一套資料系統整合的管理機制。

此外，伴隨著電子商務時代的來臨，各種商業模式在網際網路上試驗、成型，而其根本課題之一即在如何透過網際網路，使企業間的系統能使用同一種語言或程式以達到資料交換的需求。目前採用 HTML 進行資料交換存在著下列限制：延伸性不足、結構性不足、驗證性不足、設計彈性不足等等問題。而一個好的資料轉換標準需要能包含所有的資料格式，並且有一種好的原始模型來做基礎，這個模型必須能描述其資料項目，幫忙定義出資料轉換的準則。本研究論文以『Java』語言建構一個資料自動交換的雛型系統，並將交換的內容以可延伸性標示語言 XML 來表示，發揮 XML 應用的特性。

本研究以 Meta Data Model 為基礎，發展出一套能夠提供各種不同資料格式間轉換的轉換環境，希望能藉由這一個環境，完整提供使用者從不同系統間資料的重整、萃取、分析、封裝、資訊表示最佳化，進而轉入到另一個系統中。

本系統特性包含：

（1）資料內容的結構化：

可彈性訂定擷取的資料交換項目，在不影響原有 DATA 的情況下與原有資料庫整合並自動將內容以 XML 資料的格式傳送，以提昇資料的互通性。

（2）資料交換的驗證：

產生文件的文件定義檔可用來檢驗交換的資料是否有內容缺少或順序

錯亂的問題。

在現今資訊變化快速的時代，電子資料交換已成為不可或缺的一部分；而大多數的電子資料交換都在分散的處理作業之間運作，故如何快速、安全且更合乎成本效益為我們研究的努力方向。建構資料交換成為一個標準，充分利用有限的資源，將資料廣泛分散，到達每一個應用角落，為未來結構的再造工程。資料交換雖是一個簡單的觀念，卻能獲得非常有效的成果，這就是本論文研究的原動力。

1.2 論文架構

本篇論文除了緒論外共分成五章。第二章為相關背景知識與研究，此部分將介紹有關資料交換的相關研究、ODBC（Open Database Connectivity）、EDI（Electronic Document Interchange）、CDIF（CASE Data Interchange Format）、HL7（Health Level 7）以及 XML 等相關背景與相關工作。

第三章為 UDIM 系統設計。本章首先提出 UDIM 系統的設計目標，接著我們敘述系統架構、各子系統模組的關係與運作方式，以及整個系統所能完成的工作。

第四章為 UDIM 系統實作。本章描述系統的軟、硬體發展環境、資料結構、DTD（Document Type Definition）的設計理念、以及應用程式的實作過程和技巧。

第五章為系統驗證與應用。首先提出驗證 UDIM 系統的方式，接著說明系統執行與其結果。

第六章為結論與未來工作。本章總結本研究，並探討 UDIM 系統可改善之處與未來能擴充之方向。

第二章 背景知識與相關研究

本章旨在說明背景知識與相關的研究。本章將敘述有關資料交換處理之背景知識與回顧以往之相關研究和目前存在的一些標準，以期能對資料交換模組的設計與應用有進一步的瞭解。

本章共分成六節。其中，2.1 節敘述在 Data Interchange 的需求與研究。2.2 節闡述電子資料交換（EDI）的發展、使用及相關的限制。2.3 節說明目前資料庫系統中用於資料交換的開放資料庫連結（ODBC）之架構。2.4 節簡介目前常用於 CASE Tool 間的資料交換標準 CDIF，並說明其運作方式與設計。2.5 節探討普遍用於醫療資訊系統的資料交換標準 HL7。最後，我們將介紹由 W3C 所建議使用的 XML 之起源與其相關之應用。

2.1 資料交換

系統間的資料交換 (Data Interchange) 的標準與應用在軟體工程的發展中一直都是個相當重要的課題，在[9]中，作者曾清楚的提出，現今 World Wide Web 會如些流行，主要是因為 WWW 制定出了一種標準，我們稱之為 HTML，同時也指出，一個好的資料轉換標準，這個標準需要能包含所有的資料格式，並且有一種好的原始模型來做基礎，這個模型必需能描述其資料項目，幫忙定義出一些項目內聚力高的操作標準，這是需要時間來制定的。

在[10]中，作者說明了在現今電腦快速發展的時代，製造出一個沒有限制的應用程式介面是非常重要的，但更重要的是，定義出一個共通的資料轉換模組，如些一來對於應用程式的整合計劃，會有很好的幫助。

在[11]中也提到有關資料轉換的問題，作者說明了在現今盛行的個人電腦上，應用在 Windows 系統上的 OLE 和 Opendoc 等技術，雖然成功的達到不同應

用程式之間的轉換，但是它的使用只能限制在 Windows 這個作業環境之中，因此我們需要對資料轉換給予更大的定義空間，使其能適用於各種不同的作業環境之下。

對於描述一個子系統與另一個大系統之間的連接關係，先前已有不少人在研究，在 1975 年，DeRemer 和 Kron [12]就開始探討有關程式模組相連接之間的表示方法，並且首先創造出模組內部連接語言 (Module Interconnection Language, MIL) 的觀念，來做為不同模組之間的表示方法，更有不少人改進模組內部連接語言[13]的特性，更深入的探討不同的模組之間的關係與更好的表示方法，所以，模組內部連接語言已經被拿來做為對特定語言的發展標準工具，像是可以用來支援軟體建構[14]，版本控制[15]，系統群組[16]，動態組態設定[17]等等一些功能。

在[18]提到了一種有關新聞資料格式交換的方法，The IPTC Category Code Working Party 設計出了一個新的階級架構來描述一個新聞題材的所有內容，這種超包含的新聞檔案格式允許我們任意的插入相關資料的資訊，可以表示所有的多媒體，大家可利用這種共通格式存取這些新聞資料。

2.2 電子資料交換

電子資料交換 （Electronic Document Interchange, EDI） 標準起源於 1960 年，企業為了能滿足迅速回應顧客之需要及採購流程之合理化，致使各種文書工作急速增加、交貨日期之確認處理激增、交易次數頻繁、以及文件錯誤增加等問題。因此，電腦化作業之推廣與電腦連線作業成為必然之趨勢，但顧客所採用的資訊系統不盡相同，反而造成因資料格式轉換帶來的成本大增，而此時 EDI 標準因應而生。1980 年美國完成了 ANSI X12 之國家標準，到了 1990 年國際 ISO 標準組織更將 UNEDIFACT 納入 ISO 9735 正式成為國際標準，並將 EDI 做了以下的定義：從一台電腦的應用系統，運用協定的標準與資料格式，經電子化的傳遞方式，將資料傳送到另一台電腦的應用系統，讓電腦能夠自動『瞭解』、『處理』和『回應』。

所謂的 EDI，是一種整合電腦、網路及相關軟體所形成的一種新的資訊技術應用；一般而言，係指企業與企業之間的業務往來資料，以標準化的格式，在電腦與電腦之間，運用電子的型態傳送資料。 更進一步的說，EDI 所處理的是企業與企業之間正式商業交易資料，例如，採購單、對帳單及付款通知單等。基本上這些資料都是由電腦中之應用系統所產生，而非人工製作輸入。當 EDI 資料傳到其交易對象的電腦內時，是直接傳入適當的應用系統中加以處理而非印出報表，並經由業務相關人員閱讀研判後才處理。由此可知，EDI 並不只是「電腦與電腦之間」的資料傳輸，而是將資料以電腦可直接處理的格式，在「電腦應用系統之間」加以傳輸。

EDI 使得企業間往來的文件、傳票改成標準化的表格與規約，透過電信網路以格式化電子型態作電腦與電腦間情報與訊息的傳遞，換言之，就是把業者的交易訊息格式統一，並加以數位化。而它也是促成企業間進行“電子商務”的重要機制之一，將企業與企業間的資訊系統利用網路整合，讓這些交易行為電子化，進而提升商業行為的效益。以往的電子訂單系統（EOS）型態的資料傳遞後往往不能被對方系統接受，另需人工重新鍵入，恐有被登錄錯誤之虞，易造成人力浪費與正確情報接收時間的延遲，因此各企業為使彼此間的訊息交換效率化，紛紛採用 EDI 為資料交換的標準，EDI 它有下列優點：

1. 速度：縮短訊息傳送時間，符合即時傳送的觀念。
2. 環保：節省紙張、郵政成本。
3. 經濟：減少應用軟體開發的時間與成本。
4. 準確：降低資料被登錄的錯誤。
5. 相容：資訊系統間能互惠互利。

EDI 訊息多經由通訊線路傳輸，所以 EDI 系統的組成要素有：EDI 標準（包

含資料轉換標準、硬體的銜接標準）、軟體（EDI 軟體、通訊軟體、公司內部資訊系統能把要傳輸的資料轉成 EDI 軟體需的普通文字檔）以及硬體（通訊網路、個人電腦、數據機）等相互配合。以下分別來加以說明：

1. EDI 標準：

所謂 EDI 標準，就是各種單據或文件在不同的電腦系統中以其製定的的格式來傳輸，此種格式即為標準。因為 EDI 的特性就是資料在兩個電腦之間傳送，而且能自行處理，無需人為的介入，所以往來的資料必須是讓交易雙方的電腦都能看得懂；也就是說其語法(syntax)和格式(format)必須是一致的，要做到這一點，收送雙方當然就要有共同的標準。

由於各行業在進行商業行為，例如訂貨、驗收及開發票等作業，所應用到的資訊內容都不盡相同，所以生產業要先訂定出該產業內各種資訊往來的標準。而訂定產業的標準並非是要將該產業中各廠商間往來所使用的單據內容與格式加以統一，而是以國際的標準為基礎，將各廠商所使用的表單內容彙整，並取其聯集，使各廠商除了能保有原來的單據內容及格式外，也能符合國際標準，以便未來能與其它產業、國外廠商共同運用 EDI 來作業，因為大家使用的標準都是一樣的。當今國際上的標準有兩大系統，分別為美國的 ANSI X12 及聯合國所訂之 UN/EDIFACT，而 ANSI X12 標準正朝向 UN/EDIFACT 加以整合。

2. 軟體方面：EDI 的運作需具有下列三種軟體，分別是：

①轉換軟體：這是從應用系統資料庫中擷取所需要傳送的資料，並依事先設定的次序排列，以供翻譯軟體將其資料轉成 EDI 的標準格式。因為每家公司的資料庫結構與資料儲存的內容均不相同，所以轉換軟體需要針對各公司所採用的系統而特別開發。

②翻譯軟體：翻譯軟體是將轉換軟體所擷取出來將要傳送的資料，翻譯成

EDI 標準的格式，或是將接收到標準 EDI 格式資料，轉換成要送入應用系統前的檔案格式(flat file)。

③通信軟體：其目的是把翻譯完成的 EDI 標準格式資料，利用此軟體所記錄客戶的電話號碼並自動執行撥接，將資料透過通訊網路傳至加值網路公司之 EDI 資料交換中心。

3. EDI 網路：

應用 EDI 與交易廠商傳送資料，如果交易對象不多時，可以使用點對點(point-to-point)的傳輸方式，直接與交易對方的電腦連線來收送資料，而不需要透過網路公司之通訊網路。但交易對象眾多時，用點對點的方式，就非常不方便，也不經濟，而且又必須面對交易對象可能使用不同廠牌的電腦設備，在通訊技術上也必須逐一加以克服，使得整個作業環境會變得很複雜又不易掌控。

透過加值網路公司的網路系統，使用間接傳遞 (Indircet Exchange) 的方式，只要把送出的 EDI 資料利用電腦送到網路公司，網路公司就會依收作廠商，將其資料存放在各廠商專用的電子郵箱 (mail-box) 內等待取用。所以，使用者只要和網路公司連線就能把 EDI 資料送給各個不同的交易廠商，同時也可以從己自己的電子信箱中收取各方所送來的 EDI 資料，而無須逐一和各交易廠商連線，能節省不少的時間與成本。

雖然 EDI 提供了一套完善的資料交換的解決方案，但其標的只能用於企業間的商業行為的資料與訊息交換，同時也必須遵循其固定的標準格式，這使得其應用的領域變得較為陝隘，同時系統也缺少了發展的彈性；同時，使用者必須相當熟悉公司內部的資訊系統，否則會為了要使用 EDI 而會花費相當高的人力與成本。

2.3 開放性資料庫連結

Microsoft 公司為視窗環境下的應用程式，制定了一種開放式應用程式服務界面讓視窗下的軟體都可以自由使用這些 API，提供一種標準化，使得研發應用程式更為容易。符合這種制式規格的 API，便是 WOSA (Windows OpenServices API) 的一員，而 ODBC 正是 WOSA 的第一個成員。ODBC 是 Open DataBase Connectivity 的簡寫，中譯為「開放式資料庫連結」，事實上它與 lnformix、CodeBase、Paradox Engine、Jet Engine…這些資料庫函數類似，亦即能讓我們透過它所提供的函數來存取資料庫裡的資料。

其實它的觀念與 SQL、多媒體使用的 MCI (Multimedia Control Interface) 命令的概念很相像，使我們不必每接觸一套不同的資料庫就得學習一次新的資料庫函數，就以同樣的新增一筆資料而言，每套資料庫所使用的語法就往往不盡相同，甚至也有單機或連線、循序與隨機的區分。如果同一套系統裡，需要同時使用三種不同的資料庫、同時混用三種不同的函數，恐怕在設計與使用上會相當複雜，增加了許多系統開發的因難性。

而 ODBC 能夠處理許多形態的資料，無論是 DB2、Oracle、Informix…等，但特別要注意的是，ODBC 需要有各種不同的驅動程式，例如我們透過 ODBC 使用 IBMDB2 資料庫的時候，便一定要有 DB2 的 ODBC 驅動程式，ODBC 才會曉得 DB2 的資料結構與格式。而這些驅動程式必須從相關經銷商處來取得。

2.4 電腦輔助軟體工程資料交換格式

對使用者或是廠商來說，CASE (Computer Aided Software Engineering) 工具的資料格式是一個相當棘手的問題，因為每種 CASE 工具的資料格式都不盡相同，而為了讓各個不同的 CASE 工具間的資料能做互轉的動作，於是就有了用以解決資料交換的 CDIF (CASE Data Interchange Format) 標準產生。此標準是為了解決企業間使用不同的 CASE 工具而引發的資料交換或移轉問題，其支援了 Data

Flow Modeling、Data Modeling、State Event Modeling、Object Oriented Modeling、Computer Aided Control System design 及 Project Management 等模型，所以只需介面轉換原有的資料到 CDIF 的格式，就可以在各個模型之間做移轉，而不需要大費周章為轉換不同的模型而實作不同的轉換程式與介面，如此將節省了不少人力和系統開發成本。

CDIF 的出現，有助於 CASE Tool 間資料交換的工程，它是屬於一種資料格式的規格，因為電腦輔助軟體工程工具及供應廠商很多，彼此所支援的工作平台也不盡相同，例如 UNIX 平台上的工具與 PC 平台上的應用程式，基本上其所使用的資料是不能互通的，若要能互通，便須有一共通的資料交換格式，才可達到各工具間資料的彼此交換。

而 CDIF 的訂定，可使不同的 CASE Tool 所使用的資料達到交換的目的，讓彼此的資料共同分享與重覆使用，即使未來的作業環境有所變遷，只要使用的 CASE Tool 具有支援此資料交換格式的能力，便可將原有的資料順利移植到新的電腦環境或平台。

以下針對 CDIF 的產品現況及趨勢分析加以評估說明：

(1) 規範共識程度

由於電腦輔助軟體工程工具廠商於 1991 年制定此規範，已有數年的時間，目前各工具廠商已建立相當的共識，但此規範尚未成為國際標準，但將要在不同的電腦輔助軟體工程工具廠商中被採用，故規範共識程度的評估等級屬於中等。

(2) 產品可獲得性

目前支援 CDIF 的工具或廠商很多，產品也容易取得，惟近年來新

的物件導向方法論需補充之，且是不同平台的電腦輔助軟體工程工具均有支援此規範，故產品可獲得性的評估等級屬於高等的。

(3) 規範穩定性

CDIF 對於電腦輔助軟體工程工具的資料基本交換規則、文法、編碼等均有詳細規範，並已有多家廠商支援之，預期近年內不至於有太大的改變，故規範穩定性的評估等級也是高等的。

(4) 規格完整性

CDIF 對於電腦輔助軟體工程工具的基本交換規則、文法、編碼等均有詳細規範，規格極為完整，惟近年來新的方法論如物件導向需補充之，故規格完整性的評估等級是屬於中等的。

(5) 技術成熟度

目前支援 CDIF 的工具或廠商很多，相關製作的技術與文字格式的交換技術，並非新的技術，已能被充份了解及應用，故此規範的技術成熟度的評估等級屬於高等的。

(6) 業界使用狀況

目前大部份的電腦輔助軟體工程工具廠商所開發出來的產品大都支援 CDIF 的交換格式，此規範在業界使用狀況的評估等級是高的。

(7) 使用自由度

CDIF 對於電腦輔助軟體工程工具的基本交換規則、文法、編碼等均是以文字格式（Text Format）表示，非常獨立且無限制地被電腦輔助軟體工程工具廠商使用，此規範使用自由度評估等級屬於是高等的。

(8) 符合性驗證

目前尚無專責的機構提供測試、認証以及測試的標準。

CDIF 的主要功能與用途可以用下圖來加以說明，簡單地定義，CDIF 是用來描述兩個電腦輔助軟體工程工具間資料交換所需的資料結構或儲存格式，而使這兩個工具間彼此資料交換，達到資料共享及重覆使用的目的。

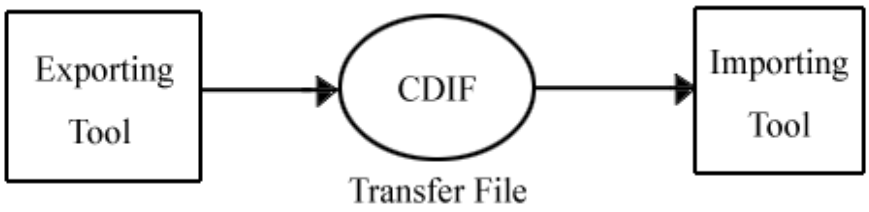

圖 1 CDIF 運作概念圖

在 CDIF 系列中有三個標準分別是：

- EIA/IS-81 CDIF - Framework for Modeling and Extensibility
- EIA/IS-82 CDIF - Transform Format Definition
- EIA/IS-83 CDIF - Standardized CASE Interchange Meta-model

是由美國 Electronics Industries Association Engineering Department, 1991 年七月制定至今。

而 CDIF 對於交換資料檔的格式與規範是在 EIA/IS-81 CDIF-Framework for Modeling and Extensibility 中規範，其中有關語意表示（Semantic & Presentation）內容是用 Meta-Model 來定義，此定義被規範於 EIA/IS-83 CDIF - Standardized CASE Interchange Meta-model 中，另外語句（Syntactic）結構內容，亦即實際交換檔的格式是由 EIA/IS-82 CDIF - Transform Format Definition 中來規範。

而對於上述 CDIF 系列標準（EIA/IS-81, EIA/IS-82, EIA/IS-83）的相互關係與架構可參考圖 2。

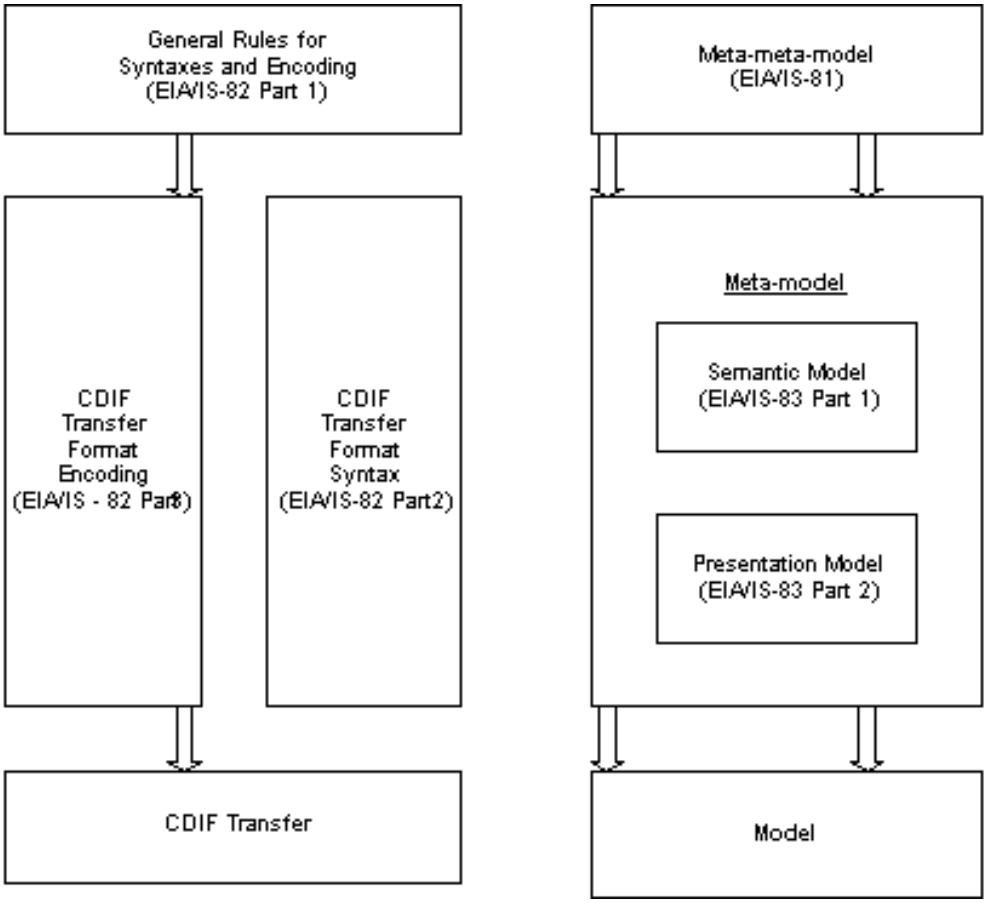

圖 2 CDIF 系列標準

雖然 CDIF 亦提供了一套相當完整的資料交換格式，然而，其卻未對一些過去未遵循其資料轉換格式的系統該如何加以處理，而且其所訂定的規格僅限於 CASE Tool 間，對於一般的資料庫系統完全無法適用。再者，其所定出的規格相當複雜，一般使用者要進行瞭解甚至加以應用，都不是一件簡單的工作。

2.5 健保第七層

健保第七層 (Health Level 7, HL7) ，它參考了 ISO (International Standard Organization) 開放式系統架構 (Open System Interconnection，簡稱 OSI) 的通訊模式，將 HL7 納為最高的一層，也就是第七層次的應用層。其主要目的在於發展各型醫療資訊系統間的電子資料交換標準以及共通的訊息交換格式。

(1) 標準發展背景

在一個醫療組織之中，不同的應用系統之間必須時常交換病歷、臨床檢驗結果以及財物交易等重要的資料。另一方面，隨著醫療組織之間合作關係的日益密切，也使得不同的醫療組織之間必須互相交換病患相關的資料。然而舊有的系統多半是針對各個醫療組織甚至各個部門所設計的專屬系統，這使得各個系統對資訊的表示方式都不相同，也使得不同的應用系統之間的整合相當的困難，因此在面對新的資料交換需求時，往往必須耗費相當大功夫來作資料的轉換。

(2) 標準發展目的

為了解決上述的問題，以加速應用系統之間的整合，「保健第七層」(Health Level 7，簡稱 HL7) 組織於 1987 年在美國成立，開始研擬醫療系統間共通的 HL7 標準，其主要目的即在於發展各型醫療資訊系統 (如醫院資訊系統、檢驗系統、藥局資訊系統、護理站系統、檢驗系統及管理資訊系統) 間，有關臨床、財務及行政資訊之電子資料交換標準，訂定各類型醫療資訊系統間共通的訊息交換格式。

(3) 應用技術探討

HL7 發展主要的之目的為經由對醫療保健應用系統間資料交換標準之定義，減少界面間程式製作及維護之負擔，以簡化應用系統之間整合的複雜度，降低系統開發的成本。

2.6 可延展性標記語言

可延展標記語言 (eXtensible Markup Language, XML)是由 W3C(World wide Web Consortium）所制訂出來的一種描述資料的標準語言，它具有高度的擴充性（Extensibility)，使用者可以自行宣告資料型態定義（Document Type Definition, DTD)，並利用自行訂定的標籤（Tag)〉及屬性（Attribute）來描述各種型態的資料，並能以結構化的方式來完整的紀錄文件本體的結構及內容。

2.6.1 XML 與資料的關係

若說 HTML（Hyper Text Markup Language）是用來顯示資訊的，那麼 XML 即可被視為是用來描述資訊的。XML 是一種標準的語言，具有描述資料能力，並可將其內容予以結構化，讓不同的應用程式了解這些資料的內容，進而加以剖析、抽取與驗證。XML 的特色在於能夠將資料內容與顯示的使用者介面予以分離。

XML 文件被認為具有自我描述的能力。也就是說，每個文件包含一組規則，文件中的資料都必須遵從這些規則。因為任一組規則皆可在另一文件中重複使用，因此，若需要的話，其它的人可以容易地創造出相同類別的文件。

2.6.2 使用 XML 來處理資料

(1) 使用 XML 作為資料轉化格式

許多使用一段時間的系統，我們稱之為 legacy system，包含不同格式的資料。開發人員投注大量工作，利用網際網路來連結這些系統。他們所面臨的挑戰之一即是要能交換原本不相容的系統間的資料。XML 可以解決這些問題。因為 XML 文字格式為標準格式，資料可轉換成 XML 格式然後其他系統以及應用程式易於處理。

(2) 使用 XML 處理 Web 資料

想像有一 HTML 網頁，其中不含任何內容。然而，內容是被儲存於 XML 檔案中。而 HTML 網頁的目的單單只為了格式化與顯示。內容可被修改與轉換成另一種語言，否則若不為作者本人修改的話，極可能需要碰觸到 HTML 程式碼。

(3)為因應不同方式使用的訊息，可使用 XML 創造出一通用的資料儲存庫

若以 XML 格式來編寫文件，則可同時使用於各種不同的環境之中，因為資料的內容與如何被顯示之間並無關。格式、輸出等等是依使用資料的應用程式而定，而不是依附於內容本身。更進一步來說，用於顯示資料的應用程式只需撰寫一次，然後即可顯示許多的文件資料。因此，其不僅僅能有效於 Web 上使用，相對地也可在其他方面的加以應用，對儲存文件資料來說，XML 是一項強力的處理方法。

2.6.3 XML 的特性

XML 是為了能有效地在 Web 中運作而設計的。此目標雖為設計的主因，但 XML 仍能在 Web 以外的環境中運作，包括出版業、資料交換以及各種商業的應用當中。為了能讓 XML 廣泛地在不同的環境中應用，XML 的設計人員知道 XML 必須為簡單、強有力並容易讓不同類型使用者製作文件的語言。XML 在設計時具有下列的特色：

(1) XML 要能在網際網路中直接地使用。

XML 必須能在網際網路中有效的運作，並能考慮到在分散式網際網路環境中執行程式的需要。這並不意味著 XML 必須立即融入目前的 Web 應用程式當中，最重要的應是能於網際網路中完善的運作。

(2) XML 應廣泛支援不同種類的應用程式。

這個目標規定了 XML 須廣泛地使用於任何應用領域。如文件製作

工具〈authoring tools〉、內容顯示引擎〈content display engines〉、翻譯工具甚至資料庫應用。XML 設計人員了解到，若要 XML 快速被接納，有賴於容易取得即使用的 XML 應用程式。

(3)XML 應與 SGML 相容。

在 XML 設計方面這目標相當重要。但是，這也是屬於眾多麻煩的目標中的一項。這想法是說，任何有效的 XML 文件亦為有效的 SGML 文件。建立這樣的目標使得現存的 SGML 使用工具能處理並分析 XML 程式碼。注意到，有效的 XML 相對地也是有效的 SGML 文件，但反之則不一定成立。記得，XML 是由去除 SGML 中不需要的部分而建立的標示語言，所以 SGML 包含許多造成 XML 處理器拆解失敗的符碼。

(4) 處理 XML 文件的程式應很容易撰寫

隱藏在此目標背後的想法是：語言的被採用程度將與工具的可取得性成正比。何謂"簡單"，當然是相對的定義。其實原本的構想為資訊本科系人員能在一至二個星期內，撰寫出基本的 XML 處理器。這樣兩個禮拜的基準目標已不在被採用了，主要是因為 XML 工具數目激增，而且大部分為免費軟體，由此我們可以印證此目標確實是已達到。

(5) XML 中選擇性功能數量應保持在最少數目

此目標來自存在於 SGML 中的問題。SGML 規格書中包含許多功能選項—很多功能選項從未使用過。這些增加了 SGML 處理器額外的負擔並增加文件與處理器之間的相容困難度，例如，一應用程式設計讀取與處理包含特定選項的 SGML 文件，如此應用程式不能正確地了解使用不同功能選項的文件。XML 藉由減少選項的數量至零，以此來避免不相容的潛在問題。這意味著，任何 XML 處理器應能分析任何 XML 文件，不論文件本身包含哪些資料或架構。

(6) XML 文件必須易讀且清楚明瞭

此目標包含了理想的與實際的目的。因為 XML 使用純文字描述資料與資料間的關係，它有易於處理與易於讀取的優點，比起完全以二進位的儲存方式更佳。

(7) XML 文件應可被快速地製作

如前已述過，需要一具有延伸性的語言以因應 Web，如此有了 XML 的初步構想。而這一目標也就自然地被納入了。若是 XML 不能快速而有效地延伸 HTML，其它的組織則可能提出更適當的解決方法。SGML ERB 相信延伸性的解決方法需要從 SGML 設群中尋求。設計人員亦確定正確的解決方案應開放的架構與具擴充性的能力，並且不能只讓單一軟體業者所專有。

(8) XML 的設計應力求嚴謹與簡明

此目標集中焦點於 XML 規格書。這個目標是想藉由規格書中文字嚴謹的處理，使它們儘可能地簡單明瞭。為了要達成這個目標，規格書使用了可延伸的 Backus-Naur 格式 (Extended Backus-Naur Form, EBNF)，此為一用來描述程式語言的標準格式。為了避免單調，在整個使用 EBNF 過程中，盡可能保持規格書嚴謹與簡單明瞭。假若 XML 語言容易了解與使用，那此目標與第四個目標在一起即是代表此語言將更容易讀懂。

(9) XML 文件應易於產生

正如第六項目標所陳述的，XML 文件應讓讀者能容易了解。儘管 XML 文件能以某些像純文字編輯器一樣簡單的編輯器創造出來，但實際上，複雜的文件有時太過笨重，以致於不能在那樣的環境下運作。這

樣有賴於市場來決定這樣的目標是否需要符合，不過很多使用 XML 的工具已經是很容易取得了，這即指出 W3C 目前為止已符合了這目標。

(10) XML 標示符號不可太過於精簡

此目標是要彌補 SGML 文件中的問題。SGML 支援最小化技術（Minimization Technique），簡單說即是：SGML 允許捷徑，已減少作者打字的煩擾。SGML 最有名的最小化方式，也是 HTML 所擁有的一部份，即是具有忽略許多元素之結束標籤。許多這樣的語言，於下一個開啟標籤出現時，即能發出信號告知先前的元素應關閉。雖然這可降低作者的工作量，但卻是造成讀者困惑的根源。在 XML 中，明確性是優先於簡潔性。

目前網站上最流行的 HTML 語言提供了在 Internet 環境下可展示資料交換的功能，但是由於其固定標籤（tag）的設計，只提供了在網路作業中資訊展現的功能，而無法提供說明資訊內容的能力，而限制了資料在 Internet 網站上被再運用及應用系統方面的發展。基於此需求，W3C (World Web Consortium)於 1998 年初公佈了 XML 的標準，提供了一個可自訂標示名稱功能的標示語言(Markup Language)，使用者可以很容易的運用 XML 在網路上自行製作所需要的文件內容及格式，並廣泛的發展不同層次的應用作業。XML 本質上並非標記語言，而是所謂的轉換語言(meta-language)，它可以讓建置者定義本身的標記語言，亦即透過定義的延伸，指定標籤的語法，而不是端視標籤本身的定義[3]。

此外，XML 為文件增加功能，特別是在技術文件方面，可以把那些文件從靜態的資料儲存器，變成它們自己的應用系統。而應用在數學公式上、化學方程式上以及其他的專業上，也都有倡議之中的標準格式。

XML 具有擴展性 (Extensibility)、結構性 （Structure）、描述性

（Description）、確認性 （Validation）等特性，其最初設計的目的便是希望成為全球資訊網上能具有描述性且又是可以交換結構性資料的標準格式。因此，XML 可讓整個產業、學術界、專業機構發展個別的文件格式定義，以標準化的方式呈現文件內的資訊[5]。

XML 的主要優點之一，就是它能為一個文件建立起結構。每一個 XML 文件都包含了邏輯結構和實體結構，邏輯結構在敘述以何種順序包含那一些元素；而實體結構則為文件中的真實資料[6]。XML 的資料架構可讓使用者依不同的方式來檢視資料、按不同的法則排序、或瀏覽時固定顯示些資訊讓文件使用更有效率如點選某個程式軟體的聯結、便只秀出符合該使用者視窗版本的資訊。

此外，XML 也可應用在 Intranet 或 Extranet 像是以資料庫為主的網站、依某些格式顯示大量的資料、XML 有很好的解決方案。而以 Extranet 的觀點來看、企業可透過 XML 瀏覽器將資訊傳達給使用者、整個產業更可依循某特定的 XML 標準、將資訊完整呈現。在存取異質資料庫方面 XML 也優於 HTML，資料可以一致的方式呈現[7]。

2.7 文件物件模組

文件物件模組 (Document Object Model, DOM)，是一種 API (Application Program Interface)之應用，主要可以提供使用者藉由相關之程式語言來呼叫 DOM 的物件；而這些 DOM 的物件可以用來存取、操作、增加及刪除文件的內容，它可以用於 HTML 和 XML 之類的文件。

對 HTML 文件來說，我們可以透過 DOM 來建立動態的網頁內容；對 XML 文件來說，DOM 可以用來描述文件的重要架構，並載入 XML 文件並加以剖析、截取與操作 XML 文件中的資料。DOM 提供網站建置者更緊密的個別網頁控制，因為每個要件都是物件，我們可以分別存取每個物件，使用 DHTML 操作它們，

這相對也解決了動態及互動式網頁的相容性問題。

DOM 是由 W3C (World Wide Web Consortium)，所推薦使用，其不同的 Level 版本特性說明如下：

Level 0：是 DOM 最原始版本，是使用 JavaScript Syntax 來操作 HTML 文件的一份聲明 (Restatement)。

Level 1：是目前已通過 W3C 認定所推薦使用的版本，也是現在最常見到的版本。它本身是與任何平台及語言無相依性的介面 (Interface)，可用以存取並處理整份 XML 文件的內容，但無法處理其相關之 DTD 與 Style Sheet 內容。

Level 2：這是 W3C 目前 Candidate Recommendation 的版本，它除了具有 Level 1 的功能外，亦能加入 interators 到相關的模組中，並可允許來存取 DTD、Style Sheet 以及 Namespaces。

Level 3：此為 W3C 最新提出的需求，它預計的功能是將一份文件的節點搬移到另一份文件之中（非 Level 2 只能做複製），並提供 node ording 的功能等。

以下是利用 DOM 來處理 XML 文件的簡單範例：

如表 1 所列，是一個簡單的 XML 文件，透過 DOM API 中的物件與方法 (Method) 的處理後，將會產生如圖 3 所示之 DOM Document Tree。

表 1 XML 範例文件

```
<?xml version="1.0" ?>
<book-order>
        <customer> Felix </customer>
        <shop> ABC Bookmart </shop>
        <goods>
                <book>
                        <name> Java and XML </name>
                </book>
        </goods>
</book-order>
```

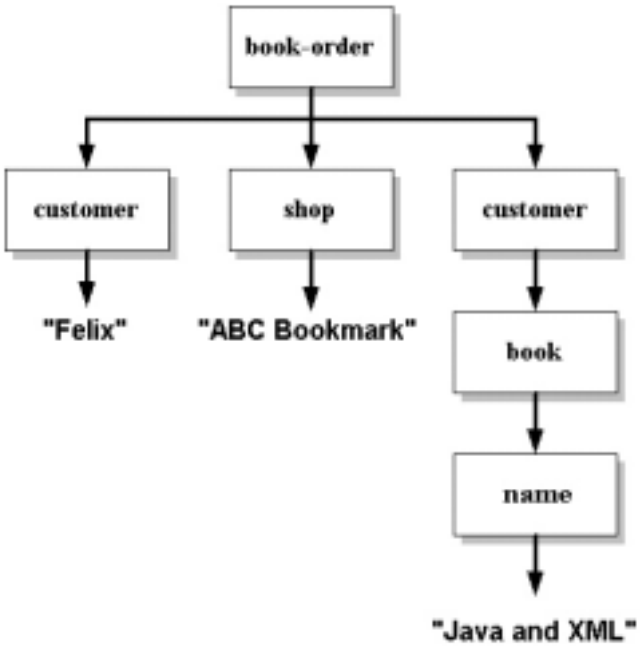

圖 3 DOM Tree 範例

至於有關 DOM 的詳細使用方式與運作方式，將於系統實作章節中做進一步之說明。

第三章 系統設計

本論文使用 XML 技術設計一套通用性的資料交換模型 (Universal Data Interchange Model, UDIM) 及其相關之應用工具組 (Toolset)。我們利用 UDIM 所製定出來的規格文件，透過應用工具組的處理，即能輕易完成不同資料庫系統間或應用程式間的資料交換工作。本章主要說明系統之架構設計與其運作之流程，共分五節來加以敘述。其中，第一節為系統設計目標，主要說明本系統所欲達到的目標與其所欲提供的功能；第二節為系統架構及其運作流程，旨在說明整個系統的架構設計與模組間之運作流程，整個系統主要由：檔案轉換規則處理 (File Transform Rules Process)、資料轉換處理 (Data Transform Process) 與資料交換處理 (Data Interchange Process) 等系統所組成，因此第三節將說明檔案轉換規則處理系統的設計與功能，第四節在說明資料轉換處理系統的設計與運作，而最後一節則說明資料交換系統之架構設計及其功能。整體之系統概念圖如圖 4 所示。

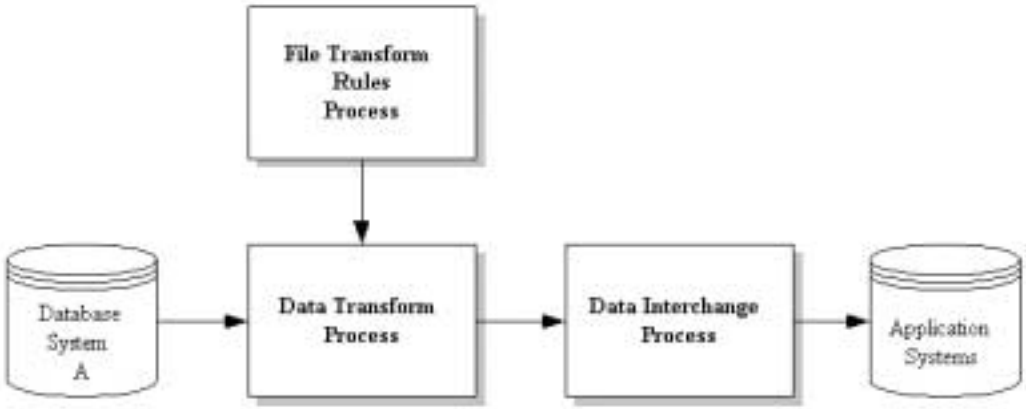

圖 4 系統概念圖

3.1 系統設計目標

理論上來說，如果我們能找到一個標準的資料轉換格式，那麼各個工具或系統只需要一個轉進跟轉出的介面就可以用來跟任何其它符合這個共通標準的工

具或系統來溝通。這可讓使用者對於工具、系統的選擇與環境的設定更有彈性。如此一來，便能改進在機構、專案、部門和整個組織間的資料轉移方式，並且也是整合系統及軟體工程環境的技術核心之一。

對於異質性環境的資料交換問題,較可行的解決方式是對於各種不同的資料型態，共同遵循與使用一個通用性的 Meta Data Model，來做為資料交換的中介途徑；這個 Meta Data Model 的目的是用來描述該資料的型態及其內容，讓目的系統的使用者，能透過此 Meta Data Model 將來源系統的資料輕易地轉換成目的系統內部資料型態;如此一來，我們只要訂定出此一共通的 Meta Data Model 的語法及規則，就可以讓不同的系統間，針對此一 Meta Data Model 而設計出資料格式對應規則的介面，其關係如圖 5 所示。

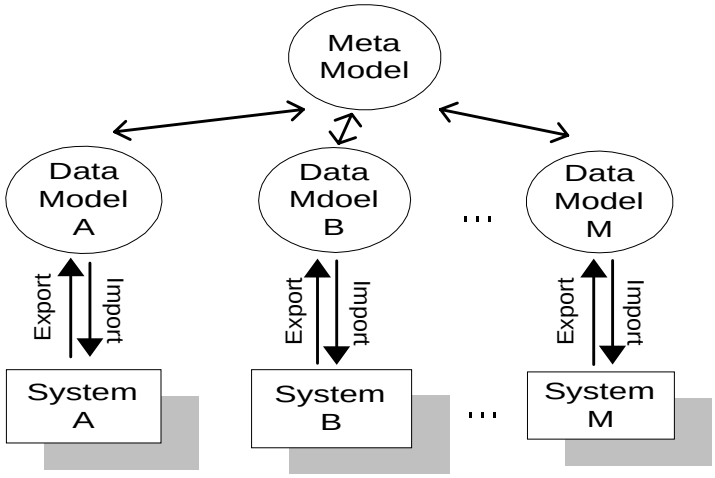

圖 5 System 與 Data Model 的關係

然而，對於這個最關鍵的 Meta Data Model 結構部分，我們綜合數種不同資料庫的特性，歸納出一個大致的基本結構，其主要包含了下列三個部分：

(1). 資料的實體檔案格式描述：

用來記載各種不同系統之特殊的檔案格式，目的在提供系統作檔案格式轉換的參考依據。

(2). 資料的 Schema 內容描述：

除了資料的檔案格式外，我們也必需瞭解資料元素彼此間的關係，讓使用者能更容易掌握各項資料所要表達的意義與資料的結構。

(3) 用以描述 Meta Data 內容的語法：

當我們把來源系統的資料透過檔案轉換處理之後，其所擷取出來的資料（稱為 Selected Data），將利用此描述 Meta Data 的語法予以記錄，以提供其它不同之目的系統來使用這些轉換出來的資料。

為了達成不同系統之間的轉換，我們需要共通的資料模型介質表示方法，用來作為不同系統格式間的資料交換。在兩個不同系統的資料表示方法中，理論上有時可以直接做轉換，但是大部分的情形是各個系統間是無法直接做交換的。這是因為資料本身的基本型態彼此並不相同，如果沒有找出適當的溝通方法，這些資料是不可能做交換的，因此我們定義出一種能封裝多種資料型態的通用性資料交換模型 UDIM，來作為異質系統間的資料交換機制。

而關於各種資料庫結構與 Meta Data Model 的對應規則該如何定義，近年來成為標準的 XML 提供了一個標準化的表示方式，依循 XML 的規則，我們定義了一套適用於描述資料庫格式與資料綱目（Schema）的語法，利用此定義出的語法，我們可以將各種不同資料庫間的資料映對成 Meta Data Model，以 XML 的表示方式來定義出各種規則，並以簡單的文件格式來加以表示。

如此一來，我們便可以先利用原始資料庫與 Meta Data Model 的對應規則，將來源資料庫中的資料轉成符合 Meta Data Model 格式的中間資料，再利用資料

交換機制來運作 Meta Data Model 中的各種資料，進而將欲被轉換的資料元件進一步來讓目的資料庫或系統來使用。

利用中間格式的轉換方式雖然比較麻煩，但是有助於未來的擴充性，一旦有新的資料庫型態出現，只要定義出此種資料庫型態與 Meta Data Model 的對應規則，便可以讓此種新資料庫型態與其他資料庫或系統作資料的互換。

3.2 系統架構

本系統主要由三大部份所組成，分別是檔案轉換規則的處理、資料轉換的處理，以及資料交換處理等系統，整體的架構圖如圖 6 所示：

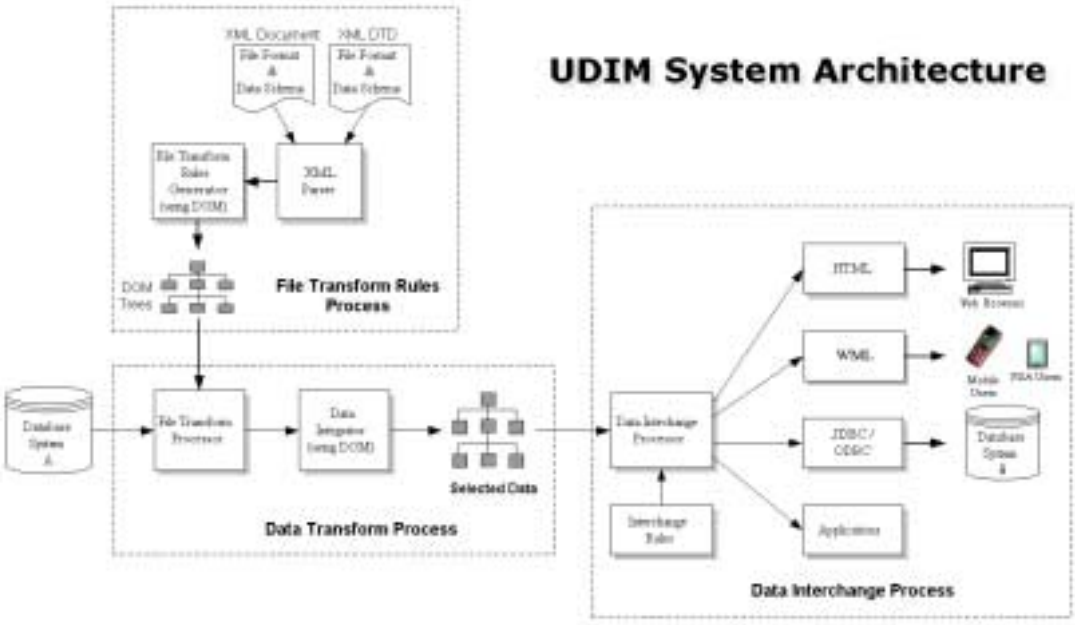

圖 6 系統架構圖

其中檔案轉換規則處理系統主要是用來產生實體檔案的轉換規則，其目的是藉由輸入相關的檔案規格與資料綱目（Data Schema）描述，讓檔案轉換規則產生器（File Transform Rules Generator）能依使用的描述而產生出一份用以擷取來源資料庫中資料的規則文件，而這份文件的描述與製作是完全使用文件物件模型（Document Object Model）來運作。

當檔案轉換規則文件產生出來後，我們將透過此一規格來進行資料庫實體檔案的轉換工作，此部份將由資料轉換處理系統來負責。檔案轉換處理器將資料庫

檔案中的資料予以擷取後，透過使用 DOM 技術的資料整合器（Data Integrator）來產生所欲轉換的資料文件，此資料我們稱之為 Selected Data，其純粹是我們所要用來做為資料交換的內容，而不含任何與資料庫系統有關的格式或特殊檔案結構的資料，而這份資料亦是以 XML 的方式來呈現。

在獲得了 Selected Data 之後，我們就可以透過資料交換處理器（Data Interchange Processor）來進行各種資料交換的工作，將來源資料庫中的資料讓其它異質性資料庫、Web Server 或 CASE Tools 來加以利用。

接下來，將針對此三部分的架構與其運作流程詳加說明。

3.3 檔案轉換規則處理系統設計

正如第二章所提到的，傳統資料庫間的資料交換只能透過 ODBC 或匯出檔案的方式來加以處理，這使得資料交換的工作必需完全依賴特定資料庫系統才能做到，這不但會增加資料交換的複雜性，同時也增加了不少的運作成本，而如此的情況將會降低了資料的可再利用性，致使資料庫系統的管理者必需面對成本與效益間取捨的問題。

而本系統正是為了解決上述的問題所設計出來的，我們由目前常用之系統、CASE 工具以及不同平台間的資料、函數格式加以分析研究，找出其間的差異，據以訂定出彼此間可以通用的資料定義格式，以便整個系統之研究與開發。因此，本系統將透過資料定義的方式來提供一個具可攜性、開放性並兼顧簡便性的資料交換環境。

本系統模組架構圖如圖 7 所示，茲將系統模組中各元件之功能予以說明如下:

3.3.1. 檔案格式與資料網目 DTD

在 XML 技術中，每一份有效 (Valid) 文件的產生，必須依據文件格式定義 (Document Type Definition, DTD) 中的規定來撰寫，因此，我們將先前所定義出來的通用檔案格式描述與資料網目，以標準 DTD 的格式來訂定作為相關 XML 文件依據與驗證的檔案格式描述與資料網目 DTD。

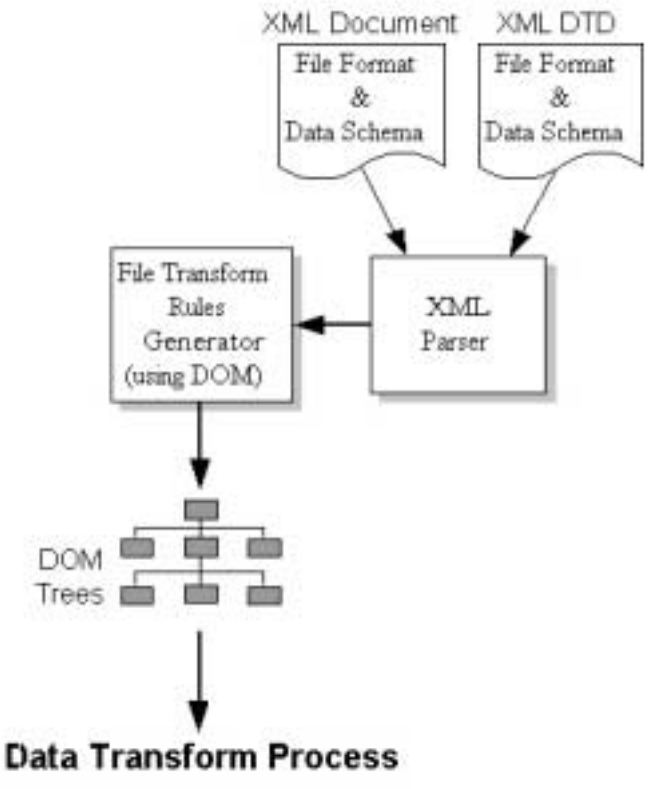

圖 7 檔案轉換規則處理系統

3.3.2 檔案格式與資料網目之 XML 文件

此部分在描述來源資料庫之特定檔案格式以及所欲轉換資料的網目，其根據先前所製訂出的 DTD 來撰寫相關的 XML 文件。

3.3.3 XML 剖析器

此元件負責剖析與驗證輸入進來有關描述檔案格式與綱目的 XML 文件是否與我們所製定的 File Format DTD、Data Schema DTD 的相符，換言之，即是用來辨識所輸入的文件內容是否符合 DTD 中的規定，若 parsing 的結果為正確，則將此結果傳給 File Transform Rules Generator，以進一步製作出資料轉換的規則文件。

3.3.4 檔案轉換規則產生器

在獲得了經 Parser 驗證後有關 File Format 與 Data Schema 的 XML 文件資料，檔案轉換規則產生器 (File Transform Rules Generator) 將使用 DOM 技術來把 XML 文件中的資料予以編排整合，進而組成 File Transform 所需的規則文件。

3.4 資料轉換處理系統設計

一般來說，資料庫的資料檔因其特殊的檔案格式與結構，通常只能在該資料庫系統的環境下才能加以運用，這不但會增加資料在維護上的成本，其它異質性系統也必需發展出一套專屬程式來做為資料交換的介面，因而造成資料交換工作變得相當地複雜。

圖 8 Data Transform Process System

此系統之主要目的就是為了能讓資料交換的工作變得更為簡便、更有效率；我們把來源資料庫的資料檔案依據轉檔規則，透過檔案轉換處理器（File Transform Processor）將資料予以萃取出來，並進一步在資料整合器（Data Integrator）中把萃取出來的資料透過 DOM 技術來加以編整，最後將結果資料（Selected Data）以 XML 文件格式來呈現。其結構圖如圖 8 所示，系統中各元件的說明如下：

3.4.1 檔案轉換處理器

檔案轉換處理器（Transform Processor) 的作用在讀取來源資料庫的檔案，並根據檔案轉換規則中對於檔案格式與資料間的關係描述，將來源資料庫資料檔中的各項資料予以萃取出來，再把結果送到資料整合器中予以進一步的處理。

3.4.2 資料整合器

由於 File Transform Processor 所產生出來的資料並沒有一個完整的結構格式，因此無法直接來讓其它資料庫或應用系統所使用；而資料整合器（Date Integrator) 的作用就是將這些由來源資料庫中所萃取出的資料，透過 DOM 的技術來加以重新編整，最後產生以 XML 為格式的 Selected Data 文件。

3.4.3 精萃資料

所謂的精萃資料（Selected Data），就是剔除掉與資料庫系統特性、特殊檔案格式以及其它與資料本質內容無關的描述資料後，所精萃出來的本質資料，而這份資料即是用來作為資料交換與再利用的材料內容。

3.5 資料交換處理系統

在我們萃取出 Selected Data 後，再來就是要將其拿來作資料交換的工作。由

於 Selected Data 是以 XML 格式來記錄資料內容，因此非常適合用來作為廣泛之應用。而此系統即是將 Selected Data 應用於與其它之系統作資料交換的工作，例如可將其傳送給 Web Server 以 HTML 的格式來呈現資訊內容,也可以透過 WML 的格式傳送給 WAP 手機、PDA 等無線設備，或透過 JDBC / ODBC 來連結其它不同的資料庫，當然，也可以傳送給其它應用程式來作更進一步的利用。其結構圖如圖 9 所示。

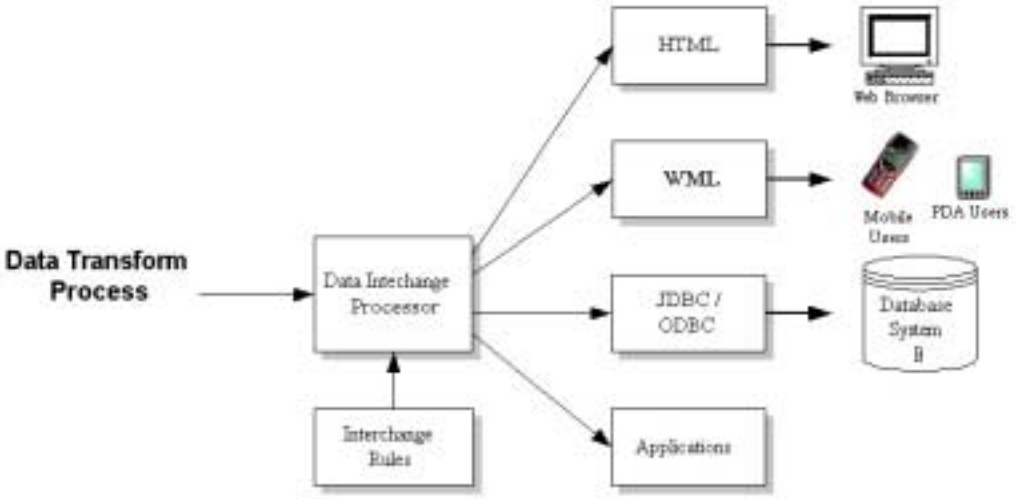

圖 9 Data Interchange Process System

資料交換系統最主要的角色在於提供與其它資料庫、網站伺服器與應用程式間的服務介面處理，資料交換處理器將接收 Selected Data，經由 DOM 技術予以剖析、分割，再將結果依使用者之需求（可於 Interchange Rules 中加以描述），透過此介面傳送給相關之資料庫或應用程式來使用，以達到資料交換的目的。

第四章 系統實作

本章旨在說明 UDIM 系統的相關實作。共分成發展環境、DTD (Document Type Definition) 設計、檔案轉換規則實作、資料轉換系統實作與資料交換系統實作等五小節。其中發展環境說明 UDIM 系統所使用的軟、硬體發展環境；DTD 設計則敘述 File Format 與 Data Schema 的相關設計；檔案轉換規則實作部分說明 XML Parser 與 File Transform Rules Generator 之實作方法；資料轉換系統實作部分則說明 File Transform Processor 與 Data Integrator 之實作方法與技巧；最後之資料交換系統實作部分則說明 Data Interchange Processor 之設計與實作。

4.1 發展環境

UDIM 系統的發展平台與工具如下：

硬體設備：個人電腦（Pentium 200, 96 MB RAM, 4.0 GB HD）

作業系統：Microsoft Windows 98

程式工具：JDK 1.2.2

應用軟體：Microsoft Visual FoxPro、IE 5.0 Browser

4.2 DTD 設計

本節旨在敘述 UDIM 系統中 File Format 與 Data Schema 的 DTD 之設計與其相關之說明。

4.2.1 檔案格式之 DTD 設計

我們根據常見的檔案格式與結構，訂定出一個通用型的 Data Format DTD，使用者只要遵循此 DTD 的規則，就能以 XML 文件描述出該資料庫的資料檔案

格式。以下茲將此 DTD 的內容與結構加以表列說明。

資料庫系統檔案格式 DTD(Data Type Definition)結構說明如表 2 所示：

表 2 資料庫系統檔案格式 DTD 結構說明

Classification	Element	Attribute	Description
File	dbsystem		Root element
	tablebody		表格主體描述
Table	creationdate		建立日期
	creationtime		建立時間
	updatedate		更新日期
	updatetime		更新時間
	tablename		表格名稱
	headerlength		檔案標頭長度大小
	recordno		表格有多少筆資料
	recordlength		一筆資料有多大
	fieldno		欄位數目
	fieldbody		欄位主體的描述
		descriptionstart	欄位主體描述開啟位置
		fieldinfolength	每一欄位資訊長度
	fieldterminator		欄位結束符號
	recordbody		一筆記錄的結構
Field	field.name		欄位名稱
	field.type		欄位型態
	field.displacement		欄位偏移大小
	field.size		欄位大小
Record	record.preceded		每一筆記錄之前置訊息
		deleted	記錄刪除之標記符號
		undeleted	記錄未刪除之標記符號
Misc	loc		位址描述
		sequence	資料存放格式(little-endian or big-endian)
	loc.start		檔案中描述 element 之起啟位址
	loc.len		描述 element 之 byte 數

根據上表的內容，我們設計出其對應之資料庫系統檔案格式 DTD，其內容如表 3 所示：

表 3 資料庫檔案格式 DTD

```
<!--  XDML DTD for Database File Format (dbsystem.dtd)-->
<!ELEMENT loc              (loc.start, loc.len)>
<!ATTLIST loc sequence   (little-endian | big-endian)   #REQUIRED>
<!ELEMENT dbsystem     (tablebody+)>
<!ELEMENT tablebody   (creationdate?,
                       creationtime?,
                       changedate?,
                       changetime?
                       tablename?,
                       headerlength?,
                       recordlength?,
                       recordno?,
                       fieldno?,
                       fieldbody,
                       fieldterminator?
                       recordbody)>
<!ELEMENT fieldbody (field.name+, field.type+, field.displacement+, field.size+,
                       field.index)>
<!ATTLIST   fieldbody     descriptionstart   CDATA   #IMPLIED
                       fieldinfolength   CDATA     #IMPLIED >
<!ELEMENT recordbody (record.preceded)>
<!ELEMENT   record.preceded   EMPTY>
<!ATTLIST     record.preceded   deleted   CDATA   #IMPLIED
                              undeleted CDATA   #IMPLIED >
```

```
<!ELEMENT creationdate    (loc)>
<!ELEMENT creationtime    (loc)>
<!ELEMENT changedate      (loc)>
<!ELEMENT changetime      (loc)>
<!ELEMENT tablename       (loc)>
<!ELEMENT headerlength    (loc)>
<!ELEMENT recordno        (loc)>
<!ELEMENT recordlength    (loc)>
<!ELEMENT fieldno         (loc)>
<!ELEMENT fieldterminator (#PCDATA)>
<!ELEMENT field.name      (loc)>
<!ELEMENT field.type      (loc)>
<!ELEMENT field.displacement (loc)>
<!ELEMENT field.size      (loc)>
<!ELEMENT field.index     (loc)>
<!ELEMENT loc.start       (#PCDATA)>
<!ELEMENT loc.end         (#PCDATA)>
```

使用者根據這份 DTD，即能寫出用以描述檔案格式的 XML 文件，同時也能利用此 DTD 作為 XML Parser 驗證 XML 文件完整性的依據。以下小節將說明資料轉換工作中用來描述資料關係的 Data Schema 之 DTD 設計。

4.2.2 資料網目之 DTD 設計

上一小節所述的是有關資料庫檔案格式的記錄方式，但由於我們無法單從資料庫的儲存檔案中，直接去瞭解各個資料與其它資料間的關係以及整體之關聯性，所以必需加上其 Data Schema 的描述來輔助資料之轉換工作。

在這份 DTD 中，我們將描述有關 table 中各欄位記錄的主要意義及其屬性，再配合上面所設計的對於 File Format 之格式記錄 DTD，就能進行資料檔案之轉換工作。

資料庫 Schema DTD 結構說明如表 4 所示：

表 4 Data Schema DTD 說明

Classification	Element	Attribute	Description
File	File	Filename	檔案名稱
Table	Table.name		表格名稱
	field.no		欄位數目
	field.list		欄位主體的描述
Field	field.name		欄位名稱
	field.attribute		欄位型態
	field.size		欄位大小
	field.relationship		欄位關連描述
	field.index		欄位 index 屬性

根據上表的內容，我們設計出其對應之資料庫 Data Schema DTD，其內容如表 5 所示：

表 5 Data Schema DTD

```
<!-- UDIM DTD for Data schema -- >
<!ENTITY   %   string          “PCDATA”>
<!ENTITY   %   dbtypeattr     “character | date | float | logical | memo | numeric |
                               | double |picture”>
<!ELEMENT   file         (table.no+, table.list+)>
<!ELEMENT   table.list   (table*)>
<!ELEMENT   table        ( table.name, field.no, field.list)>
<!ELEMENT   field.list   (field*)>
<!ELEMENT       field    (field.name,  field.attribute,  field.size,  field.relationship,
                          field.index)>
<!ATTLIST file filename        %string
               creat.date      %string
               creat.time      %string
               update.date     %string
               update.time     %string>
<!ELEMENT field.relationship (relationship.table,relationship.field)
<!ELEMENT table.no             %string>
<!ELEMENT table.name           %string>
<!ELEMENT field.no             %string>
<!ELEMENT field.name           %string>
<!ELEMENT field.attribute      %dbtypeattr >
<!ELEMENT field.size           %string>
<!ELEMENT relationship.table   %string>
<!ELEMENT relationship.field   %string>
<!ELEMENT field.index          %string>
```

4.3 檔案轉換規則處理系統實作

此系統所要處理的工作有二：① 接受使用者輸入的 XML Document，並將其作 Parsing 之處理，其目的在驗證所輸入的 XML Document 是否符合 DTD 之規則；② 若所輸入的 XML Document 是一個正確合法的 (Validating) 文件，則我們必須將用以描述 File Format 與 Data Schema 的 XML Document 中的內容傳給 File Transform Processor 來作進一步的處理，而我們將利用 DOM 技術來擷取 XML Document 的內容。

4.3.1 XML 剖析器

要構成一份 XML 文件必須要遵照一些規格限制，在 XML 的相關技術中，Parser 是最為重要的機制之一，因為它負責判斷所輸入的 XML 文件是否符合良好格式 (Well-Formed)或正確合法 (Validating)的條件；而一個具有 Well-Formed 的 XML 文件，需要滿足下列八個條件：

① XML 文件的第一列必須是「xml」的宣告
② XML 文件中只能有一個根節點
③ 開始的控制標籤與結束的控制標籤缺一不可
④ 空的控制標籤必須有「/」符號
⑤ 所有的控制標籤必須滿足巢狀的排列
⑥ 英文字母的大小寫是有差異的
⑦ 屬性值的設定必須被「”」包圍起來
⑧ 特殊字元的使用需依相關之規定

然而一份真正嚴謹的 XML 文件除了上述 Well-Formed 的條件之外，要稱得上是 Validating 的 XML 文件，還需要再加上 DTD 內容設計與使用上的檢驗，需要符合下列的幾項條件：

① 該份 XML 文件具有 Well-Formed 的特性

② 該份 XML 文件有使用 DTD

③ 所使用的 DTD 中的語法是正確的

④ 在該份 XML 文件中有正確地使用相關的 DTD

本子系統所設計的 Parser 將使用 W3C 與 IBM 所提供的 APIs，來處理 XML 文件的輸入、剖析與輸出，而 XML Document 與 Parser 我們也將其視為物件來予以操作，以下是有關 Parser 在讀入一個 XML Document，並將這份文件設為一個物件，之後再將此文件物件傳給 Parser 物件的主要片段程式：

```
import org.w3c.dom.Document;
import com.ibm.xml.parser.Parser;
import java.io.FileInputStream;
public class SimpleParse {
    public static void main(String[] argv) {
        if (argv.length != 1) {
            System.err.println("Require a filename.");
            System.exit(1);
        }
        try {
            FileInputStream is = new FileInputStream(argv[0]);
            Parser parser = new Parser(argv[0]);
            Document doc = parser.readStream(is);
            if (parser.getNumberOfErrors() > 0) {
                System.exit(1);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

4.3.2 檔案轉換規則產生器

當所輸入的 XML 文件經過 Parser 的剖析後，再來就是要處理 XML 文件中內容，而這就是 File Transform Rules Generator 最主要的工作。我們將使用 DOM API 來處理具有結構化性質的 XML 文件中的各項內容。在表 6 中列出 DOM 的主要之 Class Interfaces；圖 10 則顯示 DOM 的 Class/Interface Hierarchy。

表 6 Core Class Interfaces of DOM Level 1

Class Interface Name	Description	Implementation Classes in XML for Java
Node	The primary data type representing a single node in the document tree	Child or Parent
Document	The entire XML Document	TXDocument
Element	An element and any contained nodes	TXElement
Attr	An attribute in an Element Object	TXAttribute
ProcessingInstruction	A processing instruction	TXPI
CDATASection	A CDATA Section	TXCDATASection
Document Fragment	A lightweight document object used to represent multiple subtrees of partial documents	TXDocumentFragment
Entity	An entity, parsed or unparsed, in a DocumentType object	EntityDecl.EntityImpl
EntityReference	An entity reference, as it appears in the document tree	GeneralReference
Document Type	A DTD, which contains a list of entities	DTD
Notation	A notation declared in the DTD	TXNotation.NotationImpl
CharacterData	A parent interface of Text and others, which require operations such as insert a string and delete a string	TXCharacterData
Comment	A comment	TXComment
Text	Text	TXText
DOMException	An exception thrown when no further processing is possible; normal errors are reported by return values	TxDOMException
DOMImplentation	A placeholder of methods that are not dependent on specific DOM implementations	NA
NodeList	An ordered collection of nodes; items in the NodeList are accessible via an integral index	NA
NamedNodeMap	A collection of nodes that can be accessed by name	NA

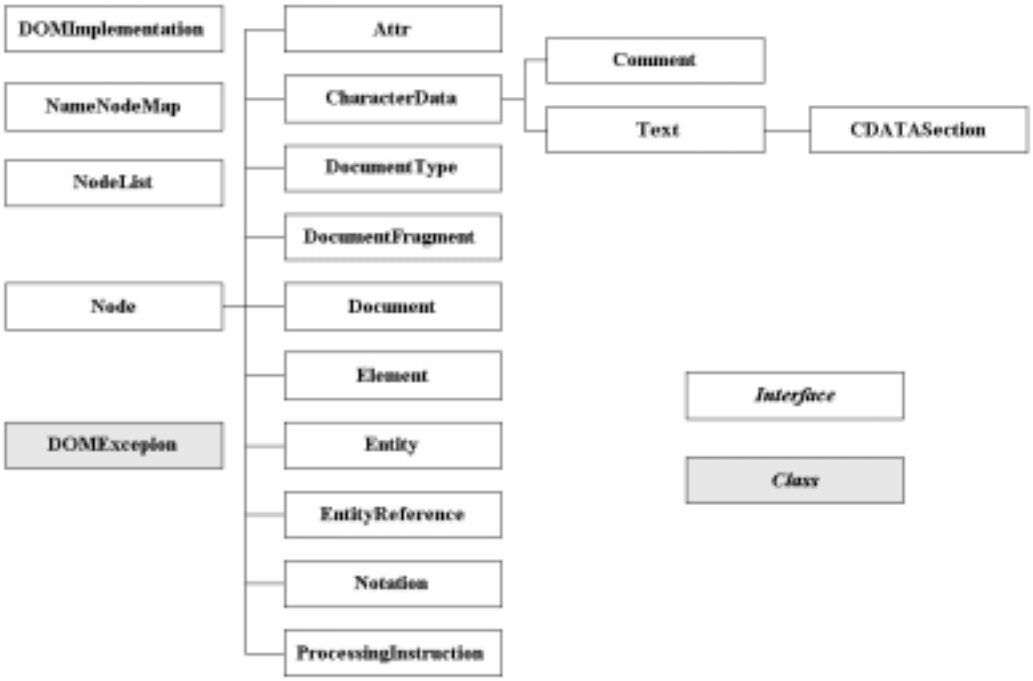

圖 10 Class/Interface Hierarchy of DOM Level 1

每一份文件，在經過 DOM 的處理與分析之後都是一顆"物件樹"，其主要由節點所構成。在 DOM 中，每個出現在文件中的基本元件都以一個物件來代表，而這每一個物件都會被視為一個節點，以支援 Node 這個 Interface。

在實作方面，我們將使用下列程式碼來建來一個 DOM Tree 的結構：

```
JPanel treePanel = new JPanel(new BorderLayout());
m_tree = new DOMTree();
m_tree.setCellRenderer(new XMLTreeCellRenderer());
m_tree.getSelectionModel().setSelectionMode
                    (TreeSelectionModel.SINGLE_TREE_SELECTION);
TreeNode node = (TreeNode)(e.getPath().getLastPathComponent());
                     nodeSelected(node);
```

此外，我們也透過在表 6 中所列的 Implementation Classes 來操作此 Tree 的內容，並將此產生出的 DOM Tree 中的節點內容，傳送給 Data Transform Process 系統來使用。以下列出幾個最常用到之 Node 屬性：

- nodeName：當物件為元素類型時，nodeName 的值就是它的元素名稱；當物件文字類型時，nodeName 的值為#text 這個標誌
- nodeValue：當物件屬於文字類型時，nodeValue 中存放的，即為物件所代表的文字內容
- nodeType：這是一個數值，可用來表示其為元素類型或文字類型
- firstChild：藉著這個屬性，可以得到第一個子節點物件
- lastChild：代表最後一個子節點
- childNodes：這是一個節點陣列，其成員為所有的子節點。利用 item( ) 這個 method，可以用來取得其中某個成員

4.4 資料轉換系統

此系統主要是透過 File Transform Processor 來轉換資料庫的檔案，將該檔案中相關的內容予以萃取出來，並將這些取出的內容傳給 Data Integrator 來進行資料整合的工作，並將結果以 DOM Tree 的方式來呈現，以便讓這些資料作進一步的利用。

4.4.1 檔案轉換處理器

本子系統接收來自 File Transform Rules Generator 所產生出來有關描述 File format 與 Data Schema 的 DOM Tree 內容後，再把所輸入的資料庫檔案中的每一個資料存放位置依 DOM Tree 的記載，將資料庫檔案中的 Data 予以取出；之後再依照 Data Schema 的關係，將這些所取出的資料做初步的排列，以助於之後資料整合的相關工作。

4.4.2 資料整合器

為了要讓從資料庫檔案中所取來資料能做更廣泛的利用，這些資料必須是一份具有中立性質的資料，並與其它的應用系統以及檔案格式無關，所以我們在取出資料庫檔案中的資料後，要將這些資料依其原有的 Schema，重新加以整合。

首先，系統會依照 Data Schema 中資料間關係的描述，自動產生一份有關這些資料關係的 DTD 文件，其設計原則如下：

① 資料庫名稱（或 Table 名稱）設為 Root Element

② Root Element 之後跟隨著此 Table 中的所有 Field 名稱（亦為 Element）

③ 若此 Table 有 Primary Key 的索引，則將其設為 Root Element 中的 Attribute

④ 將具有關聯性的 Field 所關聯到的 Table 接續在後，其設計原則如上

接下來，系統將會把這些資料配合其相互間的關係，依上述的 DTD 做成一份 XML 文件，並且自動產生其 DOM Tree，以利於資料的再利用。

4.5 資料交換系統

此系統負責將由來源資料庫中所取得的資料 (Selected Data) ，進一步作資料交換之應用。系統透過 Data Interchange Processor 來將資料傳送給欲作資料交換之目的系統，同時可以配合 Interchange Rules 來作許多不同系統之資料交換處理工作。

4.5.1 資料交換處理器

由於 DOM Tree 內的資料已具有相當之結構化，因此非常適合讓使用者來作多方面之應用。

我們可以將 DOM Tree 中的內容透過 HTML 或 CSS 在格式上的描述，讓使用者能直接在網頁中看到其所想要的資料，例如透過<b> 或 <i> 和 <font color=> 等標籤設定來改變字體樣式與顏色，或者將資料加上 <TR> 和 <TD> 等標籤，就能將有關的資料以表格之方式在使用者的 Web Browser 中予以呈現.

此外，系統也可以將這些資料轉成 WML (Wireless Markup Language) 格式的文件，而這份文件將能透過 Gateway Server 之傳輸，把文件的內容傳送給 WAP (Wireless Application Protocol) 的使用者。而一份標準的 WML 文件，將具有以上幾個部份：

① 檔頭宣告：<?xml version="1.0"?>

<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"

"http://www.wapforum.org/DTD/wml_1.1.xml">

② <Deck> 標籤：傳送畫面以一個 Deck 為單位，一個 Deck 則可包含數個畫面(Cards)

③ <Card> 標籤：裏面包含了 id 屬性，這是用來處理超連結位址的依據

第五章 系統測試與應用

在本章中，我們將以一個具有三個表格的醫療資料來作為系統的測試與相關之說明。

5.1 測試資料庫檔案格式說明

由於 DBF 檔案為目前較為公開性的檔案格式，因此我們以 DBF 檔做為系統測試的檔案，在表 7 中說明其檔案之資料位址與其相關之意義。

表 7 DBF File structure description 範例

Byte	Contents	Description
1-3	3 bytes	Date of last update; in YYMMDD format
4-7	32-bit number	Number of records in the file
8-9	16-bit number	Number of bytes in the header
10-11	16-bit number	Number of bytes in the record

根據上表，我們將其對應之 XML 文件以表 8 列出：

表 8 用以描述 DBF 檔案格式的 XML 文件

```
<!—File structure description by XML -->
<!-- Example for FoxPro file format (.dbf) description -- >
<?XML VERSION=”1.0” ?>
<!DOCTYPE dbsystem SYSTEM “dbsystem.dtd”>
<dbsystem>
  <tablebody>
    <updatedate>
        <loc sequence=”big-endian”>
            <loc.start> 1</loc.start>
            <loc.len>3</loc.len>
        </loc>
    </updatedate>
    <recordno>
        <loc sequence=”little-endian”>
```

```
            <loc.start> 4</loc.start>
            <loc.len>4</loc.len>
        </loc>
</recordno>
<headerlength>
    <loc sequence=”little-endian”>
        <loc.start> 8</loc.start>
        <loc.len>2</loc.len>
    </loc>
</headerlength>
<recordlength>
        <loc sequence=”little-endian”>
            <loc.start> 10</loc.start>
            <loc.len>2</loc.len>
        </loc>
</recordlength>
```

5.2 測試資料說明

在測試檔案中，包含了三個 Table，其分別是：病患資料、病歷資料和藥品資料等。在病患資料中，具有 ID、病歷號、姓名、生日、電話與住址等資料，而其主索引值為病患的 ID 值，並以病歷號和病歷資料表格做關聯；在病歷資料中，包含了病患的病歷號、看診科別、醫師、看診日期、診斷與藥品代號，其主索引值為病歷號，且以藥品代號和藥品資料做關聯；而藥品資料表格中則有藥品代號、藥名、標準劑量、治療途徑、價格與廠商等資料，其主索引值為藥品代號，其整體之關係如圖 11 所示。

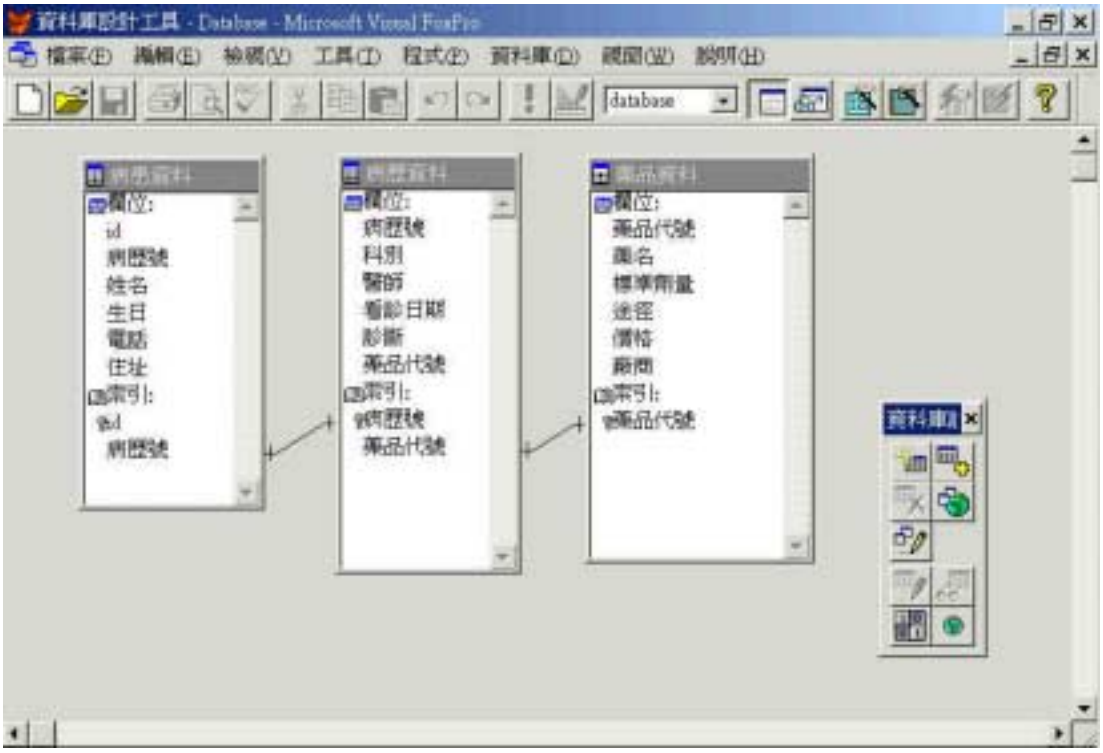

圖 11 資料 Schema 關係圖

而病患資料表格、病歷資料表格與藥品資料表格之詳細內容分別於圖 13、圖 14 和圖 15 所列：

圖 12 病患資料表格內容

圖 13 病歷資料表格內容

圖 14 藥品資料表格內容

5.3 檔案轉換測試

在這一節中，我們將說明檔案轉換工作的處理與其相關之輸出。

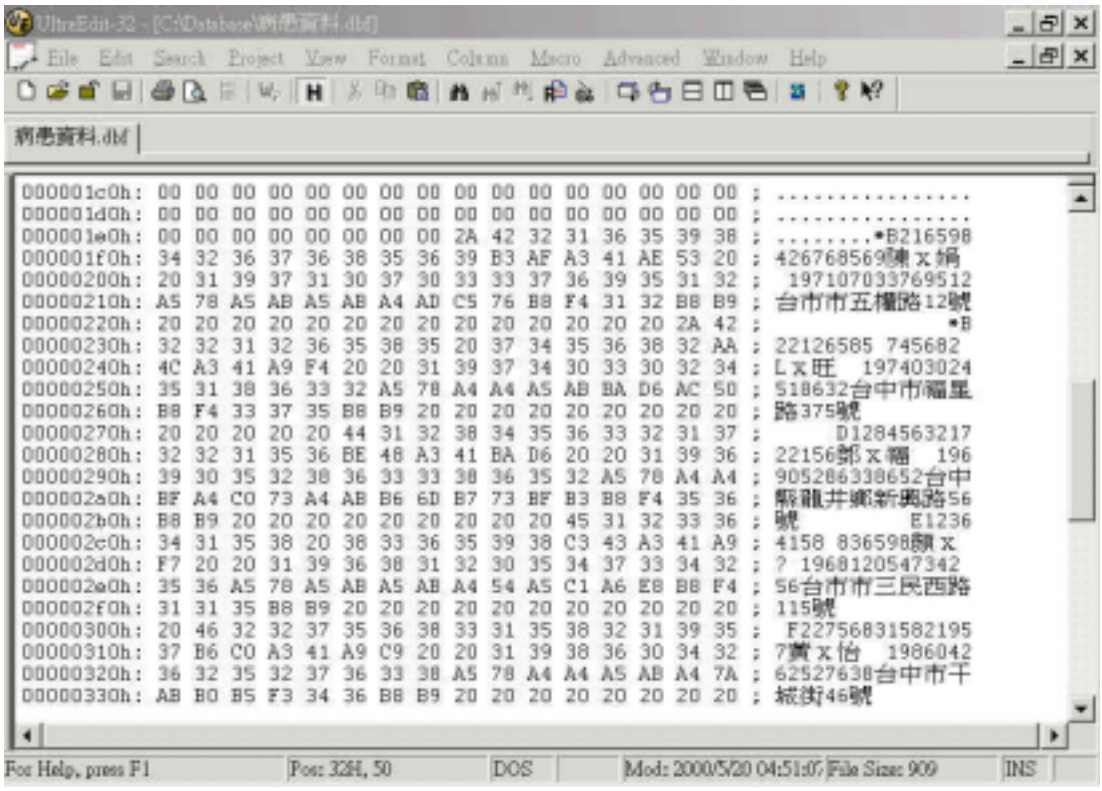

圖 15 測試資料實體檔案

在圖 15 中，所顯示的是病患資料表格的實體檔案內容，我們將病患資料、病歷資料與藥品資料中的資料一一取出後，根據其 Schema 的描述，會產生一份 DTD，其內容如圖 16 所示。

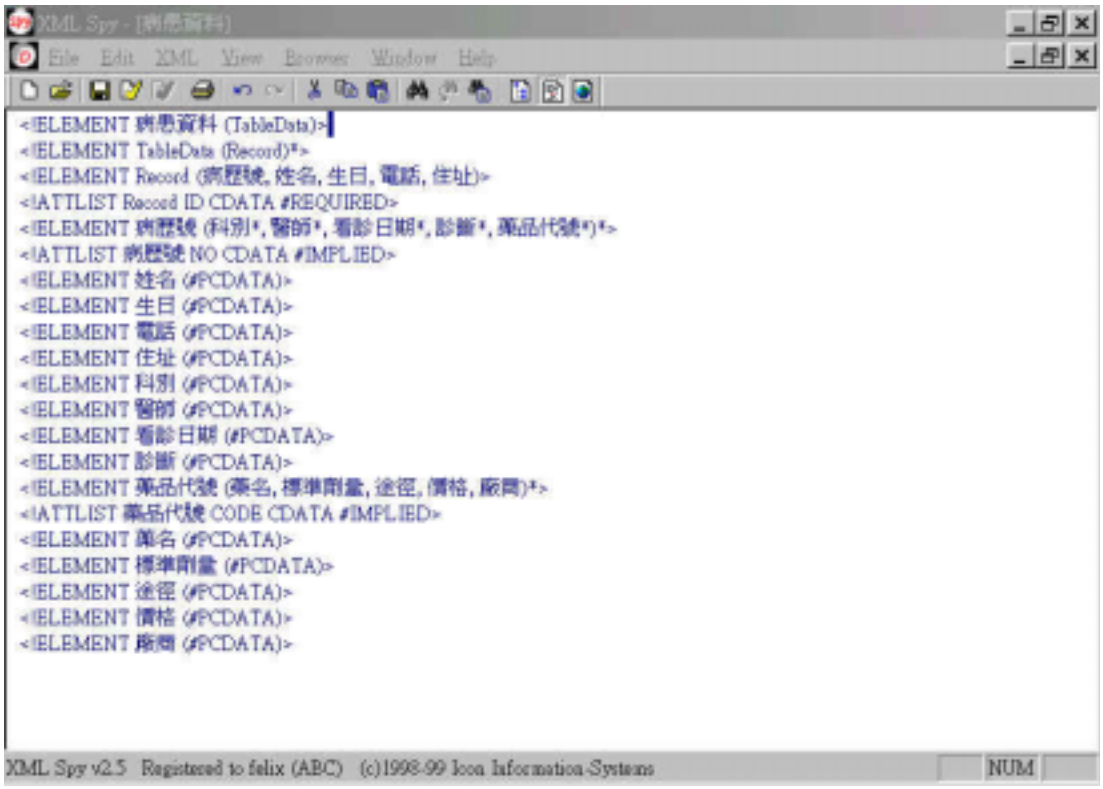

圖 16 病患資料 DTD

有了這份 DTD 之後，系統會依照 DTD 中的關係與規則，將 File Transform Processor 從原始資料庫檔案中所取出來的資料，透過 Data Integrator 製作成一份新的 XML 文件，這份文件在 XML Parser 中的內容如圖 17 所示。

圖 17 經過 Data Integrator 所產生的 XML 文件

而 Data Integrator 另一個功能是將經過轉換後的 XML 文件，透過 DOM Interface 來產生 DOM Tree，如圖 18 與圖 19 所示。

圖 18 DOM Tree view (1)

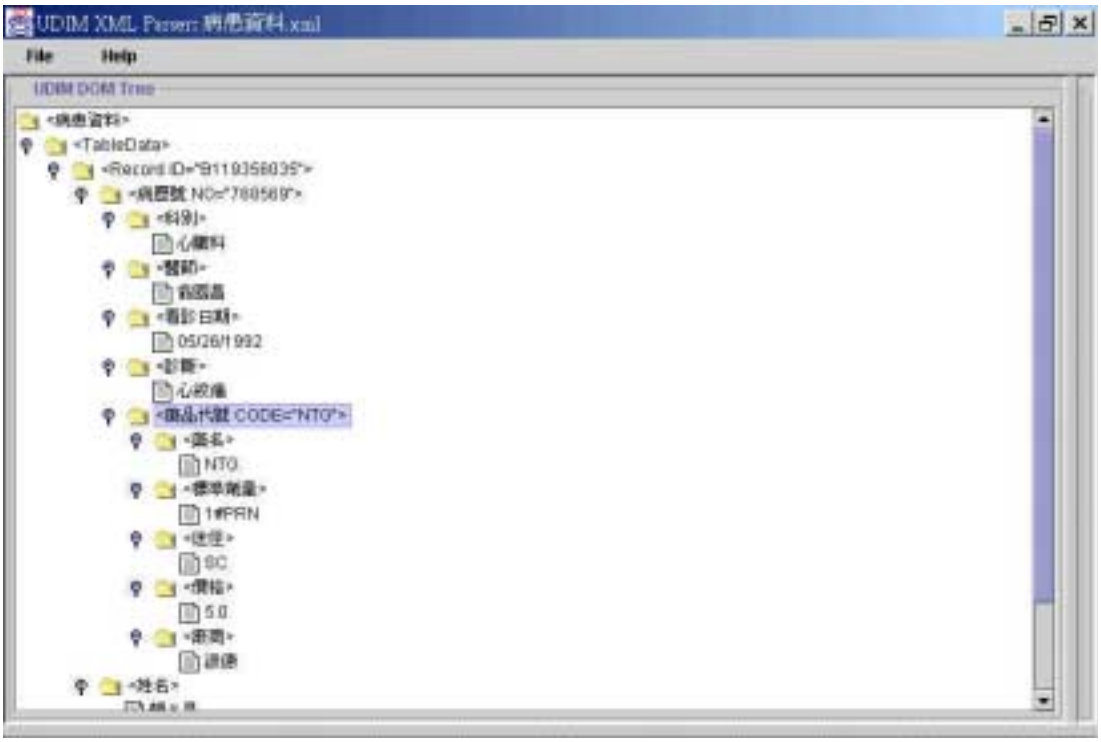

圖 19 DOM Tree view (2)

當系統產生出這份 DOM Tree 結構後，我們就可以將裏面的的內容用來作進一步資料交換的工作。

5.4 資料應用

在本小節中，我們將利用所取出的資料，作資料再利用以及透過 JDBC 來與其它系統作交換的應用。

基本上，在先前所產生出來的 XML 文件，就已經可以讓 IE 5.0 來流覽，其中並能將各個節點做縮放的處理，如圖 20~圖 23 所示。

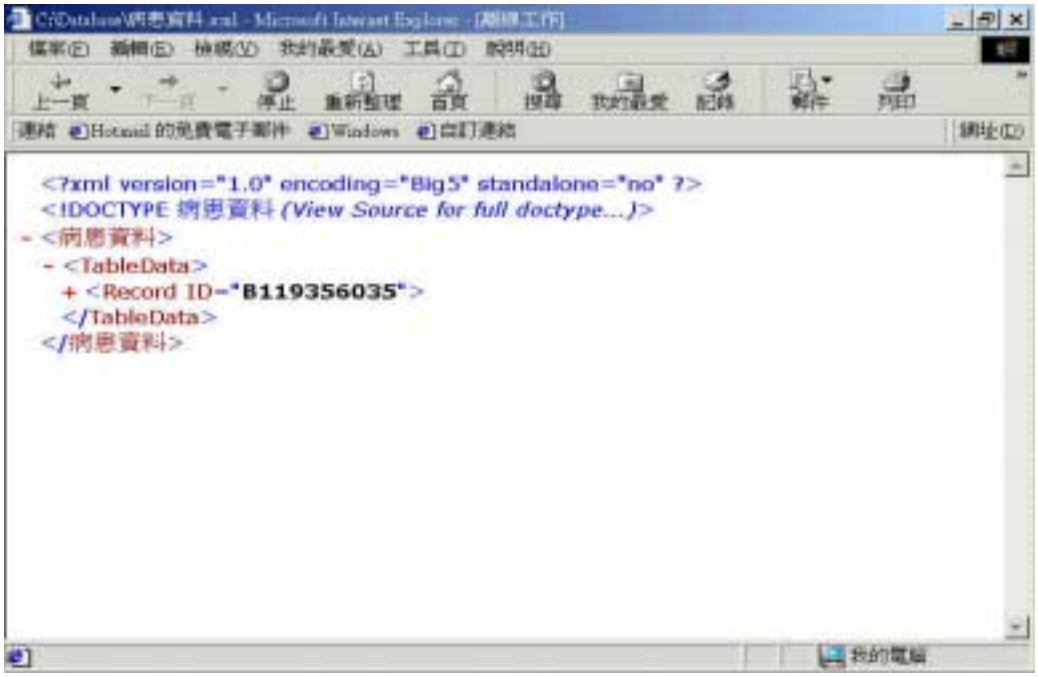

圖 20 在 IE 5.0 中流覽輸出的資料 (1)

圖 21 在 IE 5.0 中流覽輸出的資料 (2)

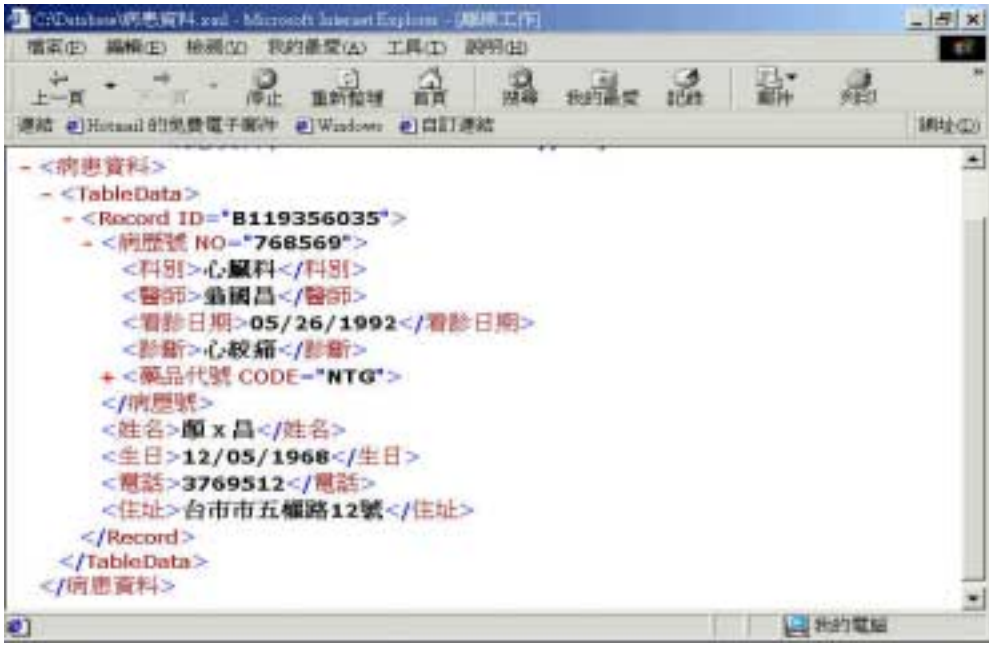

圖 22 在 IE 5.0 中流覽輸出的資料 (3)

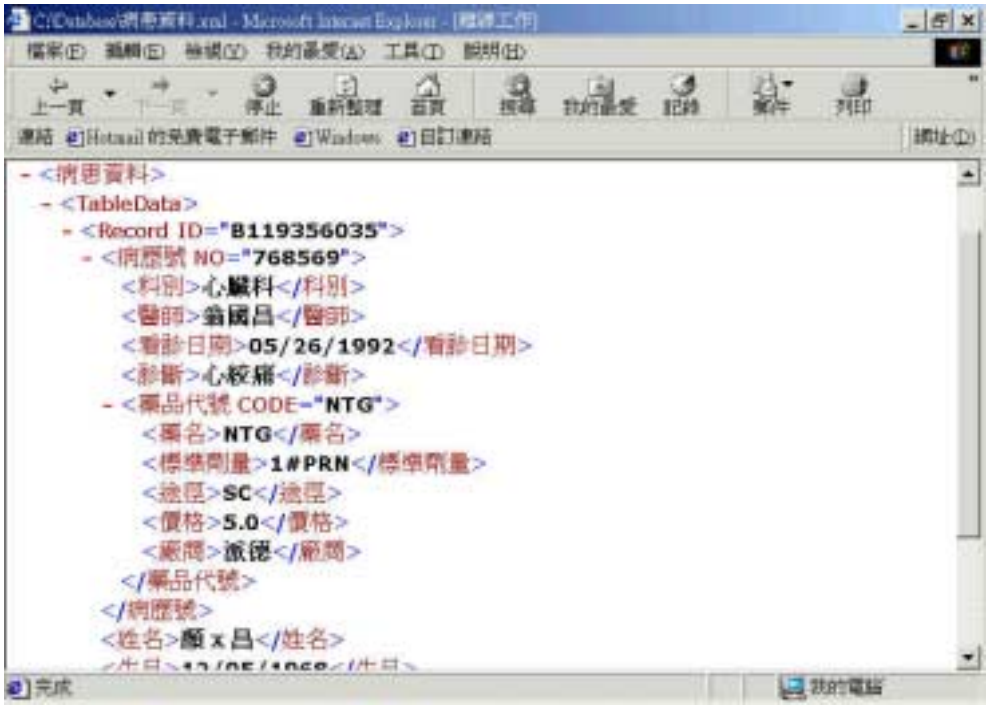

圖 23 在 IE 5.0 中流覽輸出的資料 (4)

除了上述的方式之外，我們也可將其透過 HTML 或 CSS 來將這些資料以不同之方式呈現給使用者，如圖 24 所示。

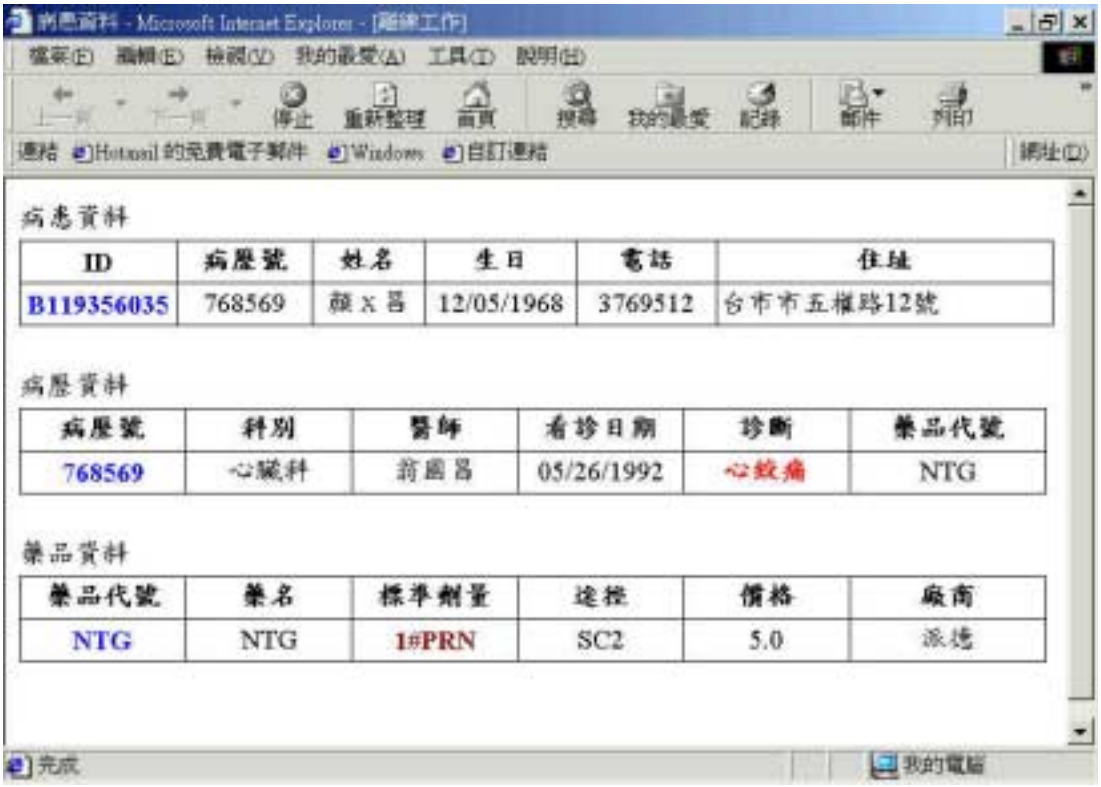

圖 24 透過 HTML 來呈現資料

由於我們所轉換出來之資料是以 XML 之方式呈現，而 WML 繼承 XML 之特性所開發出來的描述語言，因此非常易於將我們的資料之改寫為 WML 之方式來呈現資料。透過 Data Interchange 轉換之功能，我們把原始的資料配合 WML 之格式，而寫成如表 9 中所列之 WML 文件。

表 9 資料轉成 WML 格式之內容

```
<?xml version="1.0" ?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
                     "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <card id="init" newcontext="true">
    <do type="options" label="UDIM">
      <go href="#copyright"/>
    </do>
    <p align="center">
      <img src="nokiacp_logo.gif" alt="B119356035"/><br/><br/>
      <b>05/26/1992<br/>
          NTG        <br/>
          1#PRN     <br/>
          SC          <br/>
          price = 5.0
       </b>
    </p>
  </card>

  <card id="copyright">
    <do type="options" label="init">
      <go href="#init"/>
    </do>
    <p align="center"><br/><br/>
      <b>UDIM             <br/>
          Data Interchange<br/>
          by WML
      </b>
    </p>
  </card>
</wml>
```

上面所列之 WML 資料，透過無線傳輸設備，在支援 WAP 功能的手機中，將能在使用者之操作下，以圖 25 之方式顯示。

圖 25 資料透過 WML 處理後顯示於 WAP 手機中

另一個資料交換應用的例子，是我們可以透過 JDBC 來連接其它的資料庫系統。使用者能利用 JDBC API 來將這些資料直接與目的資料庫系統連接，目前 JDBC 可連接的 Database Server 有 Oracle、Sybase、Informix、InterBase、DB2、MS SQL Server、Access 與 Paradox 等常見的資料庫。圖 26 所示即為資料將透過 JDBC 來連接其它之系統。

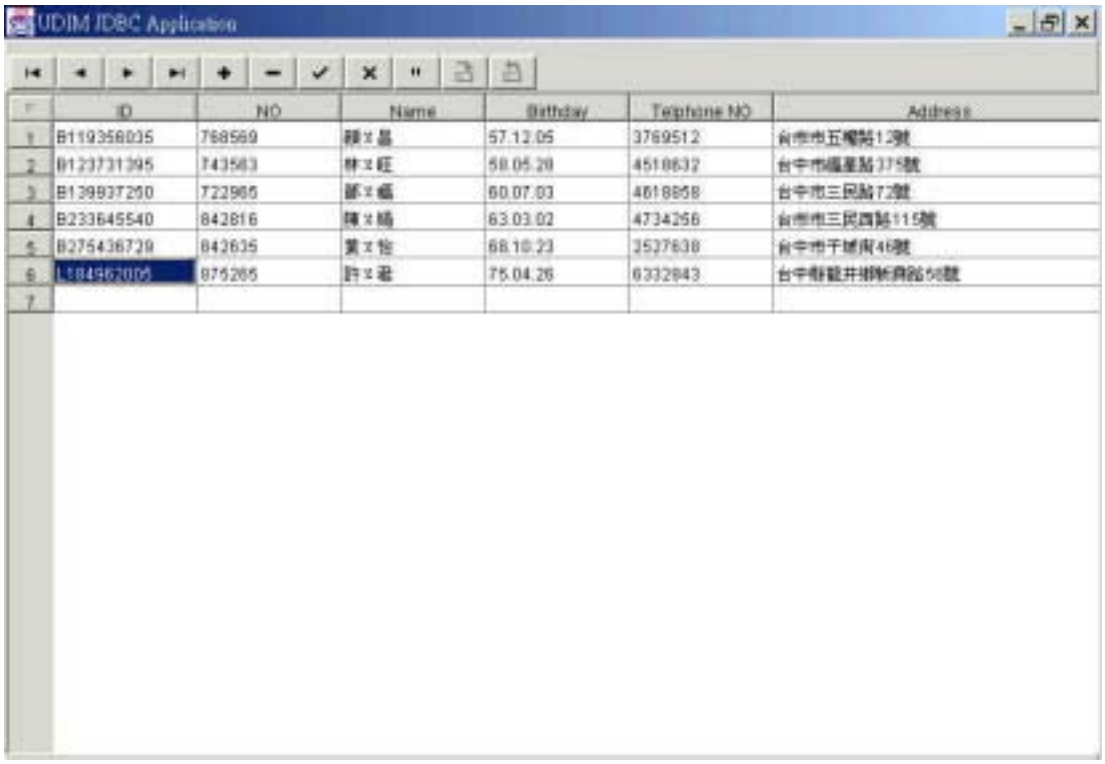

圖 26 利用 JDBC 來傳送資料

第六章　結論與未來工作

6.1 結論

以往在處理不同系統間的資料交換工作時，總是要花費許多的時間與精神在做系統分析與設計的工作，同時也會隨著應用系統的不同而必須重新撰寫程式；在本論文中，我們提出了一個以 XML 技術為基礎來製作之通用性資料交換模型 (Universal Data Interchange Model, UDIM) ，並實做一套雛形工具，以方便系統維護者進行不同系統資料間的移轉，而不必像傳統的做法需要個別開發其轉換程式。UDIM 是以 Meta Data Model 的外掛封裝為基礎，進而發展一套能夠提供不同資料格式間轉換的轉換環境，希望能藉由此一研究，提供使用者來處理不同系統間資料的轉換與交換的工作，進而將其轉入到使用者所需要的系統中，以增加資料的再利用性。

在系統設計上，主要透過 File Transform Rules Process 系統來處理資料庫檔案之轉換規則之取得；Data Transform Process 系統負責對來源資料庫檔案作實際的轉換工作，並將所取出的資料製成 XML Documents 和 DOM Trees；而 Data Interchange Process 系統則將所取得的資料做交換之處理，進而將使用者所需的資料轉入目的系統之中。

而在系統實作上，我們以 XML 與 DOM 之相關技術來完成資料的結構分析與文件內容之存取；最後再將這些處理完成後的資料做了一些應用的範例說明。

6.2 未來工作

隨著 Internet 的盛行與 E-Commerce 的發展需求，使用者對於各式各樣的資料利用將與日俱增；在面對如此多種類新的應用系統之情形，若是能將早期所發展出來的資料庫系統中的資料予以再利用，則能為系統設計者減少許多的負擔，而本系統雖已達成將資料庫系統中的資料予以轉換並加以應用，但其仍有其可擴

充之部份。

在資料來源方面，未來可整合其它非資料庫系統之資料，如傳統之文件或早期 HTML 網頁文件；在資料應用方面，則能加強使用者介面之設計，讓使用者能以簡單之描述或點選方式，來指定其目的系統之需求，讓資料交換的工作更具方便性與應用之彈性。

參考書目

[1] Song C. Choi and Walt Scacchi, " Extracting and Restructuring the Design of Large Systems," IEEE Software, pp.66-71, Jan., 1990.

[2] Jim Q. Ning, Ander Engberts, and W. Kozatizynski, "Automated Support for Legacy Code Understanding," Communications of the ACM, pp. 50-57, May, 1994.

[3] Donald G. Firesmith, "Using Parameterized Classes to Achieve Reusability while Maintaining the Coupling of Application- Specific Objects," Journal of Object-Oriented Program, pp. 41-44, June, 1994.

[4] Bruch H. Barnes and Terry B. Bollinger, "Marking Reuse Cost- Effective," IEEE Software, pp. 13-24, Jan., 1991.

[5] William J. Premerlani and Michael R. Blaha, "An Approach for Reverse Engineering of Relational Databases," Communications of the ACM, pp. 42-49, May, 1994.

[6] J. W. Hooper, R. O. Chester, *Software Reuse: Guidelines and Methods*, Plenum Press, New York. 1991.

[7] William C. Chu and Sukesh Patel, "Software Restructuring by Enforcing Localization and Information Hiding," IEEE International Conference on Software Maintenance, 1992.

[8] William C. Chu and H. Yang, "Component Reuse Through Reverse Engineering and Semantic Interface Analysis," Proceedings of the IEEE COMPSAC'95, Dallas, USA, pp. 290-296, Aug., 1995.

[9] Michael B. Spring, "Reference model for data interchange standards," IEEE Computer Society, Vol. 29, No. 8, pp. 87-88, Aug. 1996

[10] Rodney Bell, David Sharon, "Tools for Engineering New Technologies into Applications", IEEE Computer Society, Vol. 12, No. 2, pp. 11-16, March 1995.

[11] Mary Shaw, Robert DeLine, Daniel V. Klein, Theodore L. Ross, David M.Young, and Gregory Zelesnik, “Abstraction for Software Architecture and Tools to Support”, IEEE Transactions on software engineering, Vol. 21, No. 4, pp. 314-335, April 1995.

[12] R. Prieto-Diaz and J.M. Neighbors, “Module interconnection languages,” J. Syst. Softweare, Vol. 6, No. 4, pp. 307-334, Nov. 1986

[13] W. F. Tichy, “Software development control based on module interconnection,” in Proc. 4th Int. Conf. Software Enginering, Munich, Germany, pp. 29-41, 1979

[14] J. Magee, J. kramer, and Moris Sloman, ” Constructing distributed systems in CONIC,” IEEE Trans. Software Eng., Vol. SE-15, pp. 663-675, 1989

[15] The International Press Telecommunications Council, http://www.xe.net/iptc/index.htm

[16] CDIF - Framework for Modeling and Extensibility, Interim Standard, EIA 1994.

[17] CDIF - Transform Format - General Rules for Syntaxes, Interim Standard, EIA 1994.

[18] CDIF - Transform Format - Transform Format Syntax - SYNTAX.1, Interim Standard, EIA 1994.

[19] CDIF - Transform Format - Transform Format Encoding - ENCODING.1, Interim Standard, EIA 1994.

[20] CDIF - Integrated Meta-model, Foundation Subject Area, Interim Standard, EIA 1994.

[21] CDIF - Integrated Meta-model, Common Subject Area, Interim Standard, EIA 1996. CDIF - Integrated Meta-model, Data Flow Subject Area, Interim Standard, EIA 1996.

[22] http://www.cdif.org.

[23] Howard Haughton, and Kevin Lano, “Objects Revisited,” In Conference on IEEE Software Maintenance, pp. 152-161, Sorrento, Italy, 1991.

[24] R. B. Hull, and C. K. Yap, “The Format Model: A Theory of Data Organization. Journal of the Association for Computing Machinery, pp. 518-537, July 1984.

[25] Ivar Jacobson, “Object-oriented Development in an Industrial Environment,” In OOPSLA Conference, Special Issue of SIGPLAN Notices, pp. 183-191, Orlando, FL, 1987.

[26] Ivar Jacobson, “Is Object technology Software‘s Industrial Platform?” IEEE Software, pp. 24-30, Jan. 1993.

[27] Ivar Jacobson, and Fredrik Lindstrom, “Re-engineering of Old Systems to an Object Oriented Architecture,” In OOPSLA Conference, Special Issue of SIGPLAN Notices, pp. 340-350, Phoenix, AZ, 1991.

[28] G. M. Kuper, and M. Y. Vardi, “The Logical Data Model,” In Principles of Database Systems, pp. 86-96, ACM, 1984.

[29] http://www.w3.org/XML/

[30] http://www.w3.org/DOM

[31] McGrath, Sean., *XML by Example: Building E-Commerce applications*, Prentice Hall, June 1998.

[32] Elliotte Rusty Harold, *XML: Extensible Markup Language*, IDG Books Worldwide, 1998.

[33] Simon St. Laurent, *XML: A Primer*, IDG Books Worldwide, 1998.

[34] Junichi Suzuki, and Yoshikazu Yamamoto; “Managing the software design documents with XML”, Proceedings on the sixteenth annual international conference on Computer documentation, 1998, pp. 127 – 136.

[35] CheckFree Corp et.al. Open Financial Exchange Specification 1.0.2,Open Financial Exchange.

[36] Michael Morrison. *XML Unleashed*, Sams Publishing, 2000.

[37] Hiroshi Maruyama, Kent Tamura, Naohiko Uramoto. *XML and Java Developing Web Applications*. Addison Wesly. 1999.

誌　　謝

關於此篇論文能夠順利完成，首先要感謝的是擔任我研究所生涯的指導教授朱正忠恩師。在這兩年的歲月中，老師引領學生進入了更高深的研究領域，使學生不論在求學及做人處事方面都獲得了不少的增長；在論文研究方面，老師提供給學生許多寶貴的意見與研究的資源，使學生能順利地完成此論文。此外也要特別感謝逢甲大學資工系的何信瑩老師以及成功大學電機系的焦惠津老師，在論文口試過程中所給予之指導與重要建議，使得本論文能更臻完善。

在這兩年的求學過程中，也要感謝呂芳懌所長給資科所一個良好的環境，使得我們在研究的工作上更為順利。同時感謝林若瑩與施春蘭兩位系助理在這兩年來為我們處理許多行政上繁瑣的手續，讓我們能將心力放在我們的課業上。

此外，對於軟體技術實驗室的成員以及班上的同學們，平日相互討論功課，彼此砥礪與鼓勵，也使我獲得了不少重要的科技新知，並學習到許多無法從課本中所得到的寶貴經驗。而實驗室的木松同學、分散式實驗室的大卿同學與系統模擬實驗室的大中同學，兩年來在課業上彼此扶持、鼓勵，在生活上彼此交換心得，成為我在人生的旅途上不可多得的好朋友。另外，也要感謝我的好友惠茹，提供了論文中相關的實驗資料，以及在撰寫論文過程中所給我的鼓勵與信心。這些朋友都是推動我論文完成的動力，感謝你們!

最後，願將本論文及這兩年來的學習成果，獻給我最親愛的家人和朋友，感謝他們一路陪我走過，以及對我的信心與支持。