南　華　大　學

資訊管理學系碩士論文

以 XML 的技術，為舊有的資訊系統注入新活力

The New Energy to Legacy System with XML

研 究 生：蘇全福

指導教授：王昌斌 博士

中　華　民　國　九　十　二　年　六　月

南　華　大　學

碩 士 學 位 論 文

資訊管理學系

以 XML 的技術，為舊有的資訊系統注入新活力

The New Energy to Legacy System with XML

研 究 生：[redacted]

經考試合格特此證明

口試委員：[redacted]

[redacted]

[redacted]

指導教授：[redacted]

所　　長：[redacted]

口試日期：中華民國 九十二　年　六　月 十七 日

以 XML 的技術，為舊有的資訊系統注入新活力

學生：蘇全福　　指導教授：王昌斌 博士

南華大學資訊管理學系碩士班

摘要

舊有的資訊系統更新與整合，是每個導入資訊系統的組織，所必須面臨到的一個問題，隨著現代資訊科技的不斷推陳出新，當舊系統無法因應需求時，組織常面臨重新建置與更新舊資訊系統的兩難，本文的提出即是針對舊資訊系統的更新，提供一個改良式的架構，使的組織能持續保有舊資訊系統存在的價值，並且引進新的技術（如 Web Services），結合新一代行動通訊，為舊有的資訊系統賦予新生命。

本文的研究架構是改良舊有的三層式架構，在原有的三層式架構中加入「管道層」，並將原有的中介軟體（Middleware）輸出的資料，轉換為可延伸式標記語言（Extensible Markup Language，XML），以 XML 為資料交換的標準格式，並結合「管道層」形成一個新的架構，為舊有的系統注入新活力。

關鍵詞：可延伸式標記語言,舊資訊系統,網路服務,管道層,中介軟體

The New Energy to Legacy System with XML

Student：Su Chuan Fu　　Advisors：Dr. Wang Chin Bin

Department of Information Management
The M.B.A. Program
Nan-Hua University

ABSTRACT

Every organization that would like to lead into a new information system will have to face a problem: how to integrate and update the legacy system. As progressing of the new information technology, it is hard for a organization to make a decision whether to update or rebuild the information system when the legacy system might fail to meet new requirements. Our study is to provide a improved architecture for the legacy system so that we can not only let the legacy system to keep on working but also be endowed with new abilities by new technology, such as web services, and be integrated by the new generation of mobile communication, than give the legacy system a brand new life.

The architecture of this study is to improve the former three-tier architecture by adding a pipeline tier, the information generated by the Middleware are translated into extensible markup language, namely XML. We use the XML as our standard format for data exchange to form a new architecture with the combination of our pipeline tier. We hope this new structure will enhance the power of former system.

Keywords：XML, Legacy System, Web Services, Pipeline Tier, MiddleWare.

誌　謝

本論文撰寫與完成，非常感謝王昌斌教授費許多心力的指導批示，提供許多寶貴的修正意見，使得本論文能順利如期完成，並感謝各位口試委員的建議與指正。其次感謝南華大學資訊室，系統發展組組長曾清義先生，提供許多實作的寶貴意見，及多年的提攜與幫助，使得本人能如期完成學業，在此深深致謝。

論文的撰寫過程中，更感謝內人淑娟的體諒，並幫忙做論文排版，使得論文的進度能如預期完成。此外在論文撰寫之初，感謝藍建文大哥、孫嘉明老師給予的建議與論文的方向思考。謝謝大家了。

目錄

圖目錄

表目錄

第一章、緒論

第一節、研究背景

隨著資訊科技的不斷推陳出新，組織內部原有的資訊系統，在面臨新技術的便利性及系統老舊的情形下，生命週期日漸縮短，且隨著組織的成長，內部的資訊系統逐漸老舊、無法滿足內部使用者及外部客戶的需求，因此在降低成本與逐步更新不影響舊有的系統運作下，是資訊系統更新的一種思考方向。

隨著組織的成長，新舊資訊系統不斷地被導入組織中，但由於導入的時間不同，各資訊系統之間的差異，因此有企業資源規劃（Enterprise Resources Planning，ERP）與企業應用程式整合（Enterprise Application Integrated，EAI）的出現與相關探討。由於 ERP 與 EAI 所探討的議題廣泛，當企業組織在面臨既有的資訊系統更新與重置之間的抉擇時，所能提供的解決方案就顯得較廣泛，與無法找到最佳切入點。因此如何在整合上能較接近實際的系統轉換與更新，並為舊有的系統注入新的活力，導入新的資訊科技如 Web Service 的應用與企業 M 化，更是現在的企業組織所思考的方向。

可延伸式標記語言（Extensible Markup Language，XML）在 1998 年正式成為 W3C（WorldWide Web Consortium）的一項標準，由於 XML 的結構嚴謹且具驗證功能，也具有可擴充性及結構化的方式描述資料的內容，容易了解文件內容所表達的意義，因此逐漸成為資訊交換格式的主流。並且因為 XML 的出現所引起的各項資訊技術發展，如 Web Service 的相關技術 SOAP、WSDL 及 UDDI 等，使得組織內舊系統更新與整合

的技術，出現新的解決方向。

再加上資訊科技的演進，由過去的大型主機系統，到 1980s 的主從式架構系統，Middle Tier 開始盛行的 1990s[10]，資訊科技的技術不推陳出新，讓組織對於舊有的資訊系統的更新與整合，不斷的調整以符合組織因應時代的需求。

第二節、研究動機

許多較大規模的組織的資訊系統，不論是企業界或是學術界，都面臨到舊有資訊系統的更新問題，而多數組織將系統更新策略以漸進式的更新為主，其原因是組織在成長的過程中，資訊系統也隨著組織發展，資料也愈來愈多，而這些資料或資訊通常是組織的重心，此外，組織的商業邏輯（Business Logic）流程通常也隱藏在這些舊有的系統上。

因此如何為這些舊有的資訊系統注入新活力，將它與現有的科技或工具整合，透過整合與創新發展，即是本研究的動機所在。Rob Morris & Pete Isaksson 在其研究中便提出四個整合後的效益[22]：

一、不需要再維護舊有的系統。

二、可以容易的存取到舊系統的資料。

三、降低維護舊系統的費用。

四、快速的開發。

綜合以上四點，不論組織大小，當在面臨資訊系統的轉型與更新時，都會考慮到舊有資料（例如資料庫 DBMS）的繼續存在與應用，因為系統在做更新轉換時，通常舊系統都會繼續存在。系統透過更新與整合的過程，首先除引進新技術的考量外，其次便是在於整合後所引進的新技術，能降低成本費用。資訊系統整合後的延展性，通常是新技術或創新

所帶來的效益，例如企業由 e 化邁向行動化（M 化)。

第三節、研究目的

組織內部的舊有資訊系統，都會有其生命週期及因時代變遷所衍生出的整合與更新需求，因此如何在更新系統時達到最大的效益，是所有組織所期望的。

資訊系統的建置不同於一般的成本，因為資訊系統經常與組織的運作流程互相結合，這也使許多組織在更新與整合資訊系統，在考量成本時都會將組織流程、人力、影響的成本都包含進去，因為資訊系統建置或是轉移失敗，對於組織的影響相當大，嚴重甚至會危及組織生存。

如同 IDC 在 2002 年九月所發表的一篇報告，針對 Windows 作業系統與 Linux 作業系統兩者做比較[13]，以五年為期將整體的資訊相關的總成本（Total Cost of Ownership，TCO），做各項的評比。因此在針對舊有的資訊系統的更新與整合時，除了考慮 TCO 成本、舊有的資訊系統的投資報酬率（Return On Investment，ROI）之外，再來就是新舊系統轉移所造成的影響。

為了組織中的舊系統能成功與順利的轉換或更新，本文以三層式架構為基礎，加上本文所提出的「管道層」改良式架構，配合以 XML 為資料交換，以漸進式的方式更新舊有的資訊系統。

因此本研究的目的有下列幾點：

一、改進傳統的三層式架構，利用 XML 為資訊交換格式，建構出新的架構。

二、在更新或改進系統時，結合 Web Services 技術，讓資訊系統更具延展性。

三、因應 XML 的特性，延伸出跨語言（Cross-Language）的快速開發環境。

第四節、研究方法及架構

資訊系統科技的演進，由過去的大型主機系統（Mainframe），到主從式架構系統（Client-Server）及分散式架構系統（Three-tier Distributed），由於各資訊系統的導入與建置的時間不同，因此所選擇的解決方案也不同。但是隨著 XML 在 1998 年正式成為 W3C 的一項標準後，各種以 XML 為資訊交換格式的整合探討不斷被提出，本文所提出的架構，也以 XML 為資訊交換格式。

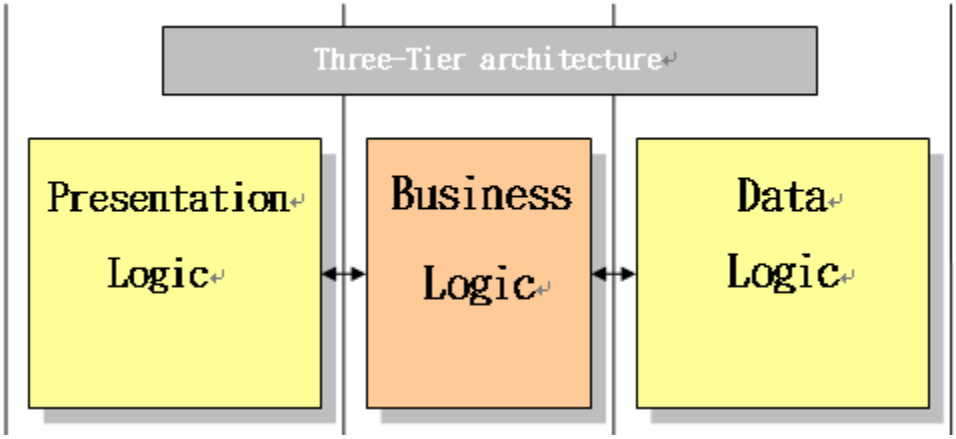

圖 1. 三層式架構圖[24]

資料來源：IEEE IT Pro 2000 年 5、6 月—Distributed Architechture Technologies

在 Robert Peacock 的論述中指出，資訊系統管理者（IT Managers）面臨舊有系統轉換的重大抉擇時，通常會考量到許多因素：例如作業系統（Operating System）、語言（Language）、網路通訊協定（Net-working Protocals）及應用程式（Application）等等層面。而針對不同的系統架構，所提出的解決方案也不同，本文將以三層式架構的資訊系統為基礎「參見圖 1」，針對三層式架構作改進，再加一層「管道層」，並加入 XML 的

轉換機制，此外並利用 XML 的特性，再延伸出新架構的資訊系統，整合其它的新技術與開發工具。

根據 Temara Petroff 的研究指出，舊有的資訊系統更新可分為下列三種策略[31]：

一、引用 package 來取代原有的系統。

二、合併原有的系統並且再發展。

三、重新建置一個新的資訊系統。

因此本文將採用第二種策略，配合本文所提出的架構，將資料轉成 XML 資料型態作為資料交換，並採取漸進的方式，為原有的資訊系統注入新活力，且在不影響原有的系統運作下，確保更新的過程中可以增加成功率。

第五節、研究限制

由於舊有的資訊系統更新與整合，所探討的議題廣泛，因此本文的研究主要著重在整合與更新上，以期能較接近實際的系統轉換上，並將研究的重點與實例探討放在分散式架構的舊有系統上，至於大型主機系統與主從式架構系統，由於機器設備與實作環境的限制，因此在本文中只作理論上的探討，而沒有實作及其結果。

本文的實例探討與所研究的範例，以 Windows 作業平台為主要的平台。

第二章、文獻探討與相關研究

本章的內容主要在探討，舊有的資訊系統更新與整合，及如何運用 XML 的相關技術與舊有的系統整合的探討研究。本章共分為四小節，第一小節探討舊資訊系統的問題與更新考量；第二小節探討舊有的資訊系統與 Middleware 的關係；第三小節探討如何將 XML 的相關技術（例如 XML、SOAP 及 Web Services）導入舊有的資訊系統；第四小節探討如何將舊有的資訊系統，導入新的技術成為一個以服務為導向的架構。最後對相關文獻整理，並彙整出本文所提出的研究架構。

第一節、舊有的資訊系統

本小節主要介紹，為何要繼續投資與更新舊有的資訊系統？在舊有之資訊系統的更新與整合時，為何不以 EAI 或 Web Services 直接作為解決的方案？及舊有的資訊系統，在更新時所要面對的問題有哪些等等。

壹、為何要更新舊有的系統？

舊有的資訊系統大致可分為三類：

一、大型主機系統（Mainframe）

二、主從式架構系統（Client-Server）

三、分散式架構系統（Three-tier Distributed）

對舊有資訊系統的兩種處理方式：

一、不再符合商業利益需求，重新買一套新的資訊系統或是重新開發。

二、還算符合商業利益需求，用現代化的方式再重新為系統注入新

活力[7]。

這個觀念簡單且容易明瞭，因為所有的組織在面臨到系統的更新時，最適合用此規則來處理，先決定是否保留還是重新建置，如此才能繼續往下處理其他相關的問題與解決方案。資料是任何企業的重心，而許多策略性的資訊，通常是隱藏在舊有的系統中，因為這些舊有系統的處理流程與商業邏輯，已經與企業組織的流程緊密的結合在一起[22]，這也是許多企業組織，在面臨舊有資訊系統的更新時，多數都會選擇保留舊有系統，並再引入新技術的整合與更新的解決方案。

企業組織在現代的生存獲利競爭壓力較過去來的激烈，網際網路的興起與發達並帶動企業 e 化，因此那些擁有大型主機系統（Mainframe）或主從式架構系統的企業組織，如何將資料或資訊放在網路上，以增加企業的競爭力，更是企業組織所關注的重點，因此才不斷的激起組織改進與整合舊有的資訊系統。

許多採用分散式架構系統的組織，常面臨到一個問題，那就是企業組織想將資料或資訊與其他公司作交流時（例如 B2B），但受限於企業本身的資訊系統。目前最常見的分散式系統架構科技為：CORBA（Common Object Request Broker Architecture）與 DCOM（Distributed Component Object Module）[2]，而這兩種架構各有其平台與環境的限制，這也是舊系統更新的原因之一。

因此針對擁有舊資訊系統的組織，在因應現代化的競爭，下列三種新科技可以提供組織因應[10]：

一、可延伸式標記語言（Extensible Markup Language，XML）

二、Web Services

三、Wireless Technology

針對此三種新科技，在後面章節會有詳細的介紹。但當面臨新技術的引進與考量時，我們所要著重的不在於新技術的種種特性，而是當新的技術引入組織時，能為組織帶來何種效益？此項新技術的考量與分析，是否真能為組織創造價值，而不是一昧的追求流行新科技。

而關於此分析論點 Tony M. Brown 在其論述中指出：「如同在 1990 年代，曾經有一個年輕的 IBM 市場經理指出，大型主機軟體的收益會以每年 7%逐年遞減，而當千禧年所引發的恐慌，可能使收益減半。但事實證明什麼也沒發生，而且某一個大型主機的軟體還大幅成長。」[32]

企業組織應該關注的是新技術有沒有替組織解決問題，新技術的出現如何與組織內部舊有的系統整合，因為過去 10 年以來，以新技術的解決方案完全取代舊有的系統，在企業組織中並不常見，且所佔有的比率也非極高，因此如何與組織內部原有的系統並存與整合，是許多組織所常見的策略之一，這也是本文所要探討的議題。

貳、舊有系統的更新策略

談到舊有的系統整合, 所直接聯想到的名詞不外乎「EAI」、「Web Services」，EAI 的分類大致上可分為兩種：[17]

一、Intra-EAI：企業組織內部的應用程式整合。

二、B2B-EAI：企業間的應用程式整合。

EAI 所探討的議題廣泛且困難，加上供應商可能不只一家或昂貴的解決方案，都讓許多組織卻步不前。而 Web Services 是另一個新興的科技，對於舊有的資訊系統來說，是另一個延伸的解決方案

而非取代方案，Web Services 就其字面上的意義來解釋，就是網路上的服務，其概念著重在「服務」上，所以當舊有系統更新與整合時，可以將此觀念導入作為舊有系統的更新策略之一。

一、EAI 所面臨的問題

Doron Sherman 在其研究中指出，當組織在面臨系統更新與整合挑戰時，有三種選擇「參見圖 2」：[8]

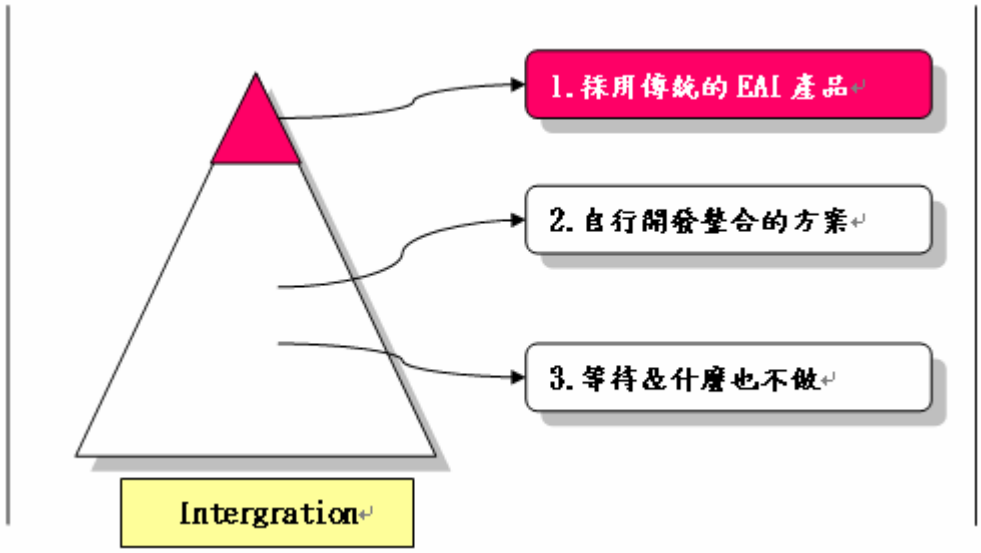

圖 2. 組織面臨系統更新與整合挑戰時之選擇圖[8]

資料來源：eAI Journal 2002 年 11 月—Web Serice Orchestration

(一)、使用傳統的 EAI 整合方案：(Use a traditional product)

使用此解決方案的組織，在軟體金額的部分大約要花費 100 萬美元，在諮詢業務與顧問費的金額大約 300 500 萬美元，而 EAI 的專案執行與週期耗時約 1 3 年。圖 2 中即可看出屬於最頂端的部分，通常這種方案只有大企業組織較有可能採用，因為太耗費時間並且所需的成本代價相當高，最重要的是並不保證一定會成功。

（二）、建立自己的整合架構：(Build your own integration fabric)

此解決方案是大部分組織最常用的解決方案，在圖 2 **中即可看出所佔有的比率。以此方式來整合風險性較高、較缺少適應性、且不容易管理與監控。**

（三）、什麼都不做並且等待著：(Do nothing and wait)

因此在本文所採用的策略，屬於第二種方式，這也是本研究提出的主要動機，因為 EAI **的議題大且廣泛，能採用的組織屬少數的大型組織團體，而當系統面臨更新與整合時，卻是所有組織都會面臨到的問題，不同的是時間發生的早晚問題，而本文便是針對此作為研究的探討方向。**

而針對 EAI **導入組織所面臨的風險，根據** Boris Lublinsky & Michael Farrell **兩人的研究，指出** EAI **導入組織十大失敗的原因，其內容如下：**[3]

（一）、EAI **的導入遠比所預想的更複雜、更昂貴。**

（二）、企業組織本身不願溝通與協調。

（三）、EAI **的導入沒有標準的模式與方法。**

（四）、EAI **在最初導入時，通常沒有考慮到橫跨系統的權限整合。**

（五）、EAI **導入與實施的差異性，遠遠大於應用程式的導入與實施。**

（六）、應用程式間的負責溝通互動，所引發的資料模組不協調。

（七）、為了整合不同系統間的溝通，所導致的全面效能不彰。

（八）、交易（Transactions**）與資料完整的議題。**

（九）、後端系統間的管理與整合。

（十）、效能矩陣定義、負荷測試、研究相關等工具的缺乏。

綜合以上十點，我們可以瞭解 EAI **導入所面臨的問題，及** EAI **所涉及的高成本支出、整合複雜度，都不是一般組織所能輕易嘗試的一種整合方案，此外在之前所提到的組織面臨系統整合與更新「參見圖** 2**」，就不難推論出為何大多數的組織都不願將時間與金錢投入** EAI **的解決方案。**

二、Web Services 的採用考量

企業組織再導入 Web Services **實有下列幾種策略：[**18**]**

（一）、資料的提供者(Data Providers)**。**

（二）、延伸架構(Subscription-based notification)**。**

（三）、複雜的的商業功能(Complex Business functionality)**。**

（四）、企業間的整合(B2B Collaboration)**。**

（五）、企業應用程式整合(EAI)**。**

Web Services **能帶來的不止整合，在舊有系統更新時的第一個目的，就是保留組織舊有的資料，其第二個目的就是運用新資訊科技所帶來的便利，第三個目的是整合舊有的** MiddleWare**。**

而這三個目的符合上述的前三點，因此我們可以將上述的前三點，作為舊有系統的更新策略考量，至於第四、五點屬於 EAI **的延伸議題，範圍太大且較不容易達成，可以當成組織資訊整合長期的發展策略考量。**

參、舊有系統更新的考量

在舊系統的更新議題上，許多文獻所探討的都著重在新科技的引

進的討論上（例如 Web Services），或是物件基礎科技(Component-based technologies)的議題，[26][15][17][27][28]，但對於舊資訊系統的更新與整合上，我們所要考慮到的部分，不只在於新科技的運用與整合上，此外還必須注意到，舊有的系統仍然必須正常地在組織內部運作，而且有關於內部原有的作業流程與運作，在舊系統的更新整合過程中，還必須注意到組織內部仍必須依賴舊有的資訊系統，這也是本文提出的架構，選擇以漸進式更新的策略最主要的原因。

此外不論網際網路如何發達, 安全仍然是所有組織必須考量到的最主要因素，這是在探討有關舊系統的整合與更新，所必須面臨到的議題，因此許多舊有的應用程式（Application），仍然會在舊資訊系統中持續的被保留，這是現代組織即使在因應新技術的引進時，一直所保留的新舊系統並行，或不同段的整合應用方案，因為「安全」與「穩定」，仍然是組織對資訊系統的兩個大要求。所以當許多銀行業在邁入二十一世紀的今天，仍然不願放棄其舊有的資訊系統，因此不難想像其決策的來源依據。

第二節、MiddleWare & 舊有的資訊系統

本小節的重點，在於探討 Middleware 與舊有的資訊系統的關係，Middleware 的定義為何？Middleware 在舊有的資訊系統，為何佔有重要性的原因？以及 Middleware 所面臨的問題與如何解決等議題。

壹、MiddleWare 的定義

根據 Andrew T. Campbell 對於 MiddleWare 所下的定義，「是一個仲介軟體，可以容易的存取到遠端的資料庫（Remote DataBase

Access)，並且也肩負著系統的交易處理（System Transactions)，也是一種管理負責性的異質分算式架構，主要在於提供開發者，一個簡單的分散式架構開發環境。」[2]，我們可用「圖 3：MiddleWare 架構圖」來解釋 MiddleWare 於資訊系統架構上所扮演的角色。

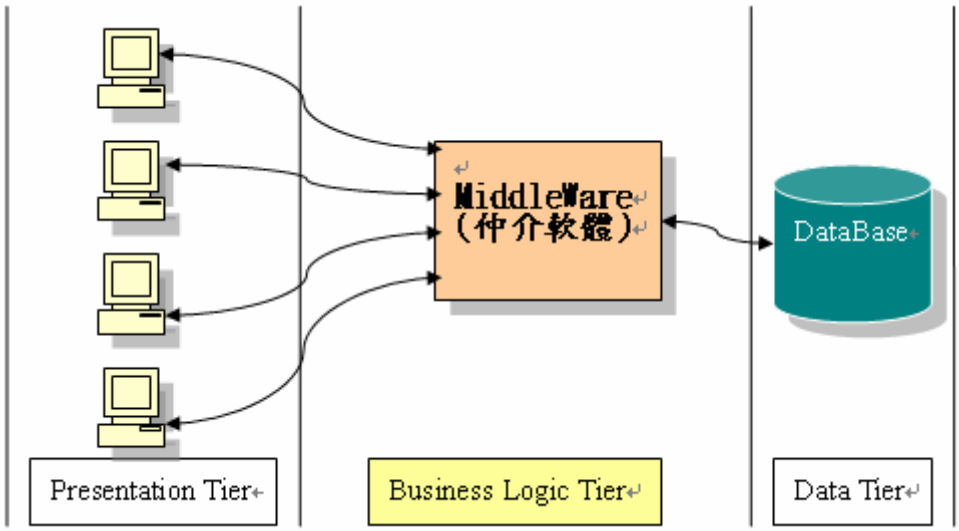

圖 3. MiddleWare 架構圖

MiddleWare 通常以不同的方式存在於資訊系統中，有下列幾種：[28]

一、IBM 的 Customer Information Control System （CICS)。

二、IBM's MQ Series。

三、the Common Object Request Broker Architecture （CORBA)。

四、Microsoft's Component Object Model （COM)。

五、Java 2 Enterprise Edition （J2EE)。

六、Web Services（the latest rage)。

由於 MiddleWare 呈現的方式不同，加上組織內部的舊資訊系統可能採用各種技術，因此如何為這些 MiddleWare 注入新活力，也是所面臨到的另一問題。

貳、MiddleWare 在舊有系統所扮演的角色

拜物件基礎的科技（Component-based technologies）之賜，MiddleWare 是建置可重複使用的商業邏輯層，最常被引用的一種解決方案。Frank P. Coyle 在其研究中提出，網際網路的革命再加上前幾年 Y2K 問題，使得許多公司已經大幅度的將其舊有的系統，以應用物件基礎科技，例如 CORBA、EJB（Enterprise Java Beans）及 DCOM，來改善其舊有的資訊系統，使其能在 Web 上提供更多的服務。[10]

自分散式架構的觀念問世，企業自舊有的大型主機系統，不斷的朝新科技方向前進，在企業成本與客戶資訊服務的考量下，許多大型企業也不斷的建置其新一代的資訊系統，而企業為了達到降低成本與增加效益，紛紛在 MiddleWare 的投資上不斷的增加金錢、人力與時間。

因此 MiddleWare 在資訊系統所佔有的比重愈來愈大，根據 IDC 的研究報告指出，MiddleWare 所佔有的市場在 2003 年將會達到 11.6 億美元，由此我們更可瞭解到，MiddleWare 在未來的發展趨勢是非常重要的。[35]

參、MiddleWare 所面臨的問題

分散式架構的資訊系統，時至今日已經有許多的組織採用，而在建置商業邏輯層（Business Logic Layer）時，最常應用 OMG（Object Management Group）的 CORBA 或是 Microsoft 的 DCOM 來當成解決方案。在建置 MiddleWare 的同時，組織流程大部分被納入 MiddleWare 中，經過 MiddleWare 的處理，最後以各種資料的型態到前端。根據 Robert Peacock 的研究中其架構「參見圖 4」，前端與後

端的部分都存在著商業邏輯（Business Logic）的部分，這也是舊有的分散式系統所面臨到的問題，這問題在前面章節有提出，再加上 OMG 的 CORBA 或是 Microsoft 的 DCOM 這兩個環境架構，各有其技術性的限制，導致分散式架構的資訊系統在整合上出現困難。

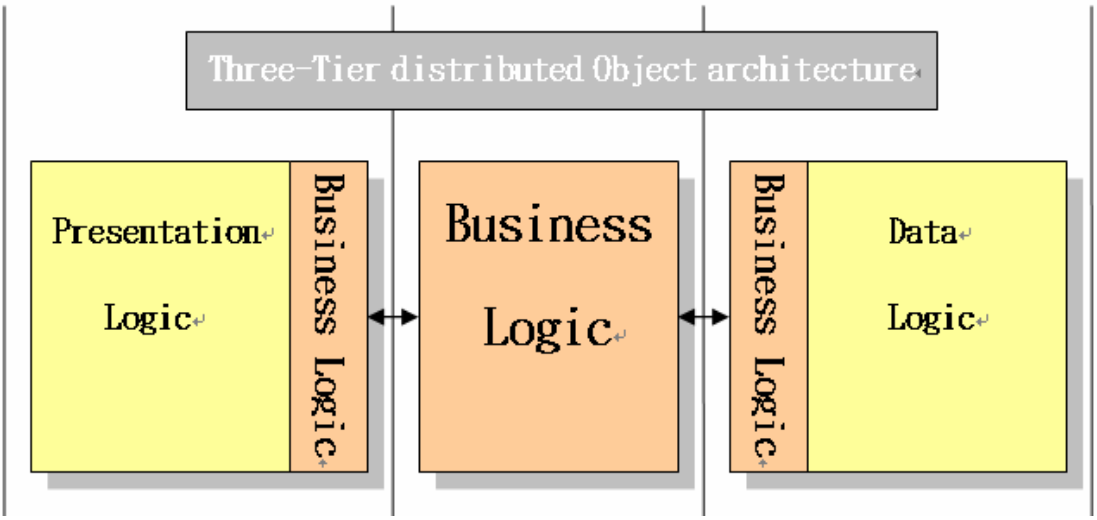

圖 4. 三層分散式架構圖[24]

資料來源：IEEE IT Pro 2000 年 5、6 月—Distributed Architechture Technologies

本文將針對 MiddleWare 所面臨的問題，提出新的架構，以 XML 為資料交換格式，並且可以與 Web Services 整合在一起，配合延伸出 Wireless Technology，幫助組織在作舊系統更新的同時，將觸角伸到行動化（企業 M 化）的議題。

Steve Vinoski 在其研究中指出，MiddleWare 的相關費用，許多小公司並不能負擔其高成本的費用，研究中指出假設負責管理系統的員工，也有相關的 MiddleWare 的經驗與技能，老闆也不會考慮這種昂貴的解決方案，因為通常老闆認為 MiddleWare 太複雜，並認為風險太高，超過員工所能處理與掌控。

而通常在兩種不一樣的作業平臺（Unix 與 Windows），如果要相互溝通，通常是將資料轉成純文字的檔案方式（Text Files），透過

FTP 或其他方式達到此目的。而這種純文字檔案的解決方案，應用在 Unix 平臺上現實世界所常見到的，而且範例多的如同一座小山一樣。[29]

因應此問題的解決方式，可以使用可延伸式標記語言（Extensible Markup Language，XML）與 Web Services 的技術來解決，針對這些新科技，在後面章節會有詳細的介紹。

MiddleWare 的發展，隨著新的科技技術不斷的發展，不論是在硬體或是在軟體的進步與革命，將對 MiddleWare 產生衝擊與問題，這是擁有 MiddleWare 資訊系統的組織勢必面臨到的問題，因為不同的 MiddleWare 之間必須互動，而如何整合不同的 MiddleWare 屬於 EAI 的議題，此處不再作深入的探討。

但由於 Web Services 近年來的蓬勃發展，Web Services 經常被引用作為 MiddleWare 之間的整合，或是有關 EAI 議題的探討，但由於 MiddleWare 與 Web Services 之間的整合是本論文所要研究的目的之一，因此提出這兩者之間的互動。

第三節、舊資訊系統的新生命力

本小節的重點在於 XML 相關技術的導入與整合。探討 SOAP、WSDL 及 UDDI 三者之間的關係，Web Services 的定義，為何要將 Web Services 導入舊有的資訊系統，以及 Web Services 與 MiddleWare 之間的關係與整合，透過文獻的探討將其相關的資料彙整，並提出本文研究架構的整合與考量。

壹、XML

可延伸式標記語言（Extensible Markup Language，XML）在 1998

年正式成為 W3C（WorldWide Web Consortium）的一項標準以來，有許多的領域將 XML 用到其相關的議題上，在資訊相關的領域，XML 也可作為資料交換的標準，之前還常拿來與 EDI 作討論，這也是因為 XML 的其種種特性。

在本文中 XML 是核心觀念之一，也是其他相關新技術的核心觀念，例如 SOAP、WSDL 及 UDDI 等，在本文的後面章節會做詳細的描述。XML 可以解決不同的程式語言、資料庫、作業系統平台所引發的資料格式問題，Sanjay Pathak 在其文章中指出，在分散式架構的環境下，使用 XML 來存取遠端的服務是一個睿智的方法[26]，這與 Web Services 基本精神是相同的，但本文的研究將把此觀念，在延伸到 MiddleWare 與 Web Services 兩者的整合上作探討。

一、XML 的定義

可延伸式標記語言（Extensible Markup Language，XML），由 W3C（WorldWide Web Consortium）所提出的，並且在 1998 年二月中旬宣佈成為正式的標準規範。XML 開發者會告訴你 XML 本身並不是一個程式語言，而是一個用來定義其他語言的系統，W3C 正在制定許多與 XML 相關的建議規範，XML 是「一種用來表達資料結構的通用語法」，結構化的意義是在資料中使用標籤來標示內容的語意以及用法。

XML 文字格式檔案是用來表現結構化資料的一種方法，由於語法嚴謹因此在資料定義上的特性強，再加上 XML 是由許多規則所組成的集合，可以用來針對資料的文字格式作設計，而且依此格式定義出來的檔案，可以非常容易的被電腦產生及讀取，不會因含糊不清與模稜兩可，而產生最常見的問題，

就是依存於某個特定的平台，因此歸納出 XML 具有下列幾個特色：

（一）、XML 適合人或機器直接解讀。

（二）、XML 可以適用於任何作業系統平台。

（三）、XML 文件之分析、處理非常容易自動化，可以將人力參與程度降至最低。

（四）、XML 本身的設計就已經可以支援多國語言。

以資料的角度來看，XML 的文件是由實體所組成，在 XML 文件中包括了 Text 以及 Binary 兩種型態，Text 是以字元所組成，可以分為標記（Markup）和字元資料（Character data），XML 文件中的 Binary 實體則有自己本身的表示方法，以字元來說則必須考量是以 Unicode 或其他標準來定義。另外 XML 還事先定義了明確的類別，例如字母（Letter）、空白（White Space）或數字等，以及事先定義的樣本。

一份 XML 文件的組成通常可分為三類：XML、DTD 及 XSL。XML 文件是主體，是用來存放文件內容資料的格式，文件型別定義（Document Type Definition, DTD）是用來定義 XML 文件中資料的邏輯結構，而 XSL 是定義 XML 文件的顯示，一個 XML 檔案可以用許多 XSL 檔來描述，而這也是其最大的特性，因為 XML 將資料的結構與內容和它們的呈現方式完全分離開來，因此同一份 XML 原始文件僅需撰寫一次，便可以使用不同的 XSL 來顯示。

二、XML 的文件結構與格式

XML 文件有兩種的型式，分別是 valid 與 well-formed。

這兩種型式為文件提供良好的結構，以及使資料成為自我描述的資料，因為一份合於語法的 XML 文件必須遵從一些 XML 語法的通用規則，這套規則比舊有的 HTML 或 SGML 都來得更嚴謹。

（一）、valid 的 XML 文件

一個有效（valid）的 XML 文件，必須要嚴謹地遵從在 DTD 文件中規定的所有規則， XML 的文字資料一定要有起始標籤相對應的結尾標籤，例如＜FOROMID＞＜／FORMID＞，XML 文件的產生大多由工具，或是系統程式轉換而來的，而這兩者會同時檢查文件的有效性以及是否合於語法。

（二）、well-formed 的 XML 文件

一個合於語法（well-formed）的 XML 文件，所指的是裡面的標籤用法，符合某一套 DTD 所規定的規則，但卻沒有比有效性的 XML 文件來的那麼嚴格。而檢查 XML 文件是否有效性，最常用來使用的工具就是瀏覽器，因為只需要確定該文件是合於語法的，瀏覽器就可以正確讀取了。

三、XML 的標示

XML 的標示可以分為以下幾種：「詳見表 1」

表 1. XML 的標示說明

XML 的標示	說明
標籤	開始標記之後，一定要有一個結束標記。例如：＜Student Name＞＜／Student Name＞。
文件類別宣告	主要是用來描述 XML 的版本, 而文件類別宣告必須出現在 XML 文件中第一個開始標記之前，例如：＜？XML version= "1.0"？＞

表 1. XML 的標示說明（續）

	＜Tag Name＞Hello world！＜／Tag Name＞
字元資料	除了 XML 的標示外，其他符號都算字元資料，也就是 XML 文件的本身。
字元資料段落	必然是以＜！[CDATA] 做為開始，以]]＞ 做為結束，若在這之間仍有標記會被 XML 忽略，另外在此段落中不需要有巢狀結構，而在此段落中可以用來放置程式碼、圖片等，不希望被 XML 處理的資料。
註解	必然是以＜！--作為開始，以-->作為結束，例如：＜！-- Markup Data --＞
執行指令	必然是以＜？作為開始，以？＞作為結束，但系統保留字除外。
空白字元控制	如果需要控制文字的空白，可以使用 XML-Space 屬性來處理。

四、XML Schema

一份 XML schema 也是一份 XML 文件，透過 XML Schema 或是 Document Type Definition（DTD） 可以來定義資料的結構與格式，這也是 XML 的最大效用所在，不論任何舊系統的資料透過 XML Schema 的定義後，其資料就可用 XML 的格式型態呈現，因此不論在新舊資訊系統的發展、更新與整合，以此方法來轉換舊有的資料型態，將有助於資料格式問題的解決。

而一份 XML Schema 文件可包含下列八種不同型態的元素、文法定義的語法與結構「詳見表 2」。

表 2. XML Schema 文件中八種不同型態的元素

元素	說明
＜Schema＞	是 XML Schema 文件的根元素。
＜ElementType＞	定義可能使用在 XML Schema 文件中所出現的元素型態資料。
＜AttributeType＞	定義可能使用在 XML schema 文件中的屬性型態

表 2. XML Schema 文件中八種不同型態的元素（續）

＜Attribute＞	在＜ElementType＞內出現的元素, 可以用來定義該＜ElementType＞可使用屬性。
＜Element＞	在＜ElementType＞內出現的元素, 可以用來定義該＜ElementType＞所使用的元素。
＜Group＞	在＜ElementType＞內出現的元素, 可以用來定義該＜ElementType＞如何群聚使用元素。
＜Datatype＞	在＜ElementType＞或＜AttributeType＞內出現的元素，定義該＜ElementType＞或＜AttributeType＞的資料型態。
＜Description＞	提供＜Schema＞，＜ElementType＞或＜AttributeType＞ 文件。

五、XML 的相關技術

XML 1.0 定義了「標籤」 和「屬性」 代表了什麼意義的規格書，但有關 XML 相關技術的配合，有許多不同的模組針對各種不同的工作，而其相關的技術如下：[41]

（一）、Xlink 是用來描述一種在 XML 文件中加入超鍊結的標準方法。

（二）、XPointer 與 XFragments 是用來指向 XML 文件中某部份的語法，XPointer 類似 URL，但它所指向的不是網際網路上的文件，而是在 XML 文件中的一部份資料。

（三）、CSS 文件樣式語言, 在 XML 文件中的使用與在 HTML 上一樣合用。而 XSL 是用來表達文件樣式的進階語言。它使用 XSLT（是一種超出 XSL 所支援不到的轉換語言）為基礎，用來加入或刪除 XML 文件中的標籤和屬性。

（四）、DOM 是一組標準的函式呼叫，與 SAX 的目的相同，都是用來解析 XML 文件資料， 兩者都可供程式語言去解析 XML 文件。

（五）、XML 命名空間（Namespaces）是一種用來描述 XML 文件，如何讓每一個單獨的標籤和屬性，能夠關聯到一個 URL 上面的規格書，而這個 URL 的使用目的，是由讀取它的應用程式來決定的。

（六）、XML Schemas 1 與 2，可以幫助開發者精確地定義，他們以 XML 為基礎所制定的檔案格式。

貳、SOAP、WSDL 與 UDDI

一、SOAP 的發展背景

SOAP（Simple Object Access Protocal）是在 1998 年由 Dave Winer 所提出的概念延伸而來，最早概念是 XML-RPC，因為 Dave Winer 與其他人覺得，當時所存在的 RPC（Remote Procedure Call），例如 DCOM（Distributed Component Object Module）與 IIOP（Internet Interoperable Orb Protocal），並不適合網際網路的使用，因為這些通訊協定需要複雜的環境支援，DCOM 與 IIOP 都綁在本身的物件模組上，再加上有許多組織的防火牆對這些通訊協定都有一定的限制。[33]

因此更阻礙了網際網路程式設計的困難度，因此對此限制而發展出 SOAP，因為 SOAP 並不會綁在任何特定的物件模組上。

二、SOAP 的特色

SOAP 最大的特色就是解決資訊流通問題，在不同作業系

統、開發工具等因素下，每個組織所選擇的資訊系統因應方案也不同，因此會延伸出在相關技術上的瓶頸，造成資訊流通困難的問題。SOAP 在 Web Services 所扮演的角色是傳送使用端的請求（Request）到接收端，利用 SOAP 通訊協定將訊息傳遞於兩者之間。

SOAP 通訊協定所使用的技術，是標準的 HTTP（Hyper Text Transfer Protocal）呼叫與可延伸式標記語言（Extensible Markup Language，XML）的資料格式，因此可以解決不同平台的資訊交換問題。例如使用 Microsoft Visual Basic 所撰寫的應用程式，可以使用 SOAP 的通訊協定，呼叫在 UNIX 系統平台上的 CORBA 架構所撰寫的方法（Method）[33]。

SOAP 在之前的來源介紹已說明過，由於 SOAP 是源自於 XML-RPC，所以 RPC（Remote Process Call）是分散式架構的一個重要觀念，因此當有關 SOAP 被引入舊有的資訊系統探討時，經常被引入主從式架構資訊系統（Client-Server System），這類舊有資訊系統的更新與整合問題上探討。

因此 SOAP 可說是一種傳送 XML Message 的通訊協定，因此由另一個觀點解釋 SOAP，可以將 SOAP 看成是網際網路上的一種訊息導向的中介軟體（Messaging - Oriented Middleware，MOM），所以當 SOAP 被建議當成解決 B2B 之間的解決方案時，也不需要感到意外。

SOAP 也能帶領我們往大量的分散式架構的議題方向前進，因為分散式應用程式整合（Distributed Application Integration，DAI）是繼企業應用程式整合（Enterprise Application

Integration，EAI）後的下一個邏輯步驟。[11]。

Timothy M.Chester 在其研究中指出，使用 SOAP 與 XML 來解決組織內部因各平台間差異所造成的資料流通問題[34]，且成功的整合新舊資訊系統的軟硬體環境，可當成此方面的參考依據。[40]

三、WSDL 的定義

WSDL 在 Web Services 所扮演的角色是服務的描述，WSDL(Web Services Defination Language)是一種以 XML 語法來作為描述的語言，主要是用來描述與定義 Web Services，WSDL 很類似 CORBA 所使用的 IDL(interface Definition Language)，WSDL 是一種 Web Services 的介面定義語言，其中的定義包含資料類別的定義(Data type definitions)、輸入輸出訊息的格式(input/output message formats)、網路位址通訊協定等。[12]

WSDL 為了保持其中立性，但它在內部描述了 SOAP 的支援，因而與 SOAP 建立了不可分割的關係，所以將介紹 WSDL 的部分放在 SOAP 之後。

WSDL 文檔可以分為兩部分：由上半部的抽象定義與下半部的具體描述組成。[39]

（一）、抽象定義包含：Types、Messages 與 PortTypes「參見圖 5」

圖 5. WSDL 抽象定義圖

（二）、具體描述包含：Bindings 與 Services「參見圖 6」

圖 6. WSDL 具體描述圖

四、UDDI 的定義

UDDI 在 Web Services 所扮演的角色是登錄註冊，將 Web Services 的資訊整合在一起，以便於其他需求者可以作搜尋、查閱，找到所需要的 Web Services，UDDI 的觀念很像 Windows 作業系統平台的登錄器(Register)。UDDI 的目的在於，如何將 Web Services 與其相關資料資訊，在網際網路上進行註冊的標

準規範。這些資訊包含了提供某項服務的組織，及如何存取該項服務的技術相關資訊等。

UDDI 中包含三個主要的註冊實體：[14]

（一）、公司的聯絡資訊：一般都將此資訊放在「白頁」(White Pages)中。

（二）、公司的標準分類資訊（例如：公司產業的代碼、圖形索引等）：一般都將此資訊放在「黃頁」(Yellow Pages)中。

（三）、有關「服務」的技術性資訊（例如：e-Business 的描述、服務的位址）：一般都將此資訊放在「綠頁」(Green Pages)中。

而當 UDDI 中所註冊的服務數量，已經累積到一定程度的時候，動態搜尋服務功能便會付諸實現，而現在的資訊大廠，例如微軟、 IBM 都已經相繼投入 UDDI 的建置，建立了各自的公用 UDDI 目錄服務，然而在這些 UDDI 服務受到各界廣泛接受之前，各企業組織也都會在企業內部使用自己的 UDDI，私有 UDDI 資料庫能夠讓各組織較為緊密地控制本身系統，以及應用程式資料的存取方式。

參、Web Services

本小節探討 Web Services 的用意在於，將現今最熱門的 Web Services 科技作一整體的分析，而非直接採用 Web Service 當成解決方案，並找出 Web Service 的優缺點與延伸性為何？如何將 Web Service 當成一個連接舊有系統的管道接口，將原有的組織內部「資訊寶藏」，透過此「管道」運送到外界。

一、何謂 Web Services？

Web Services 是一項新的網路科技的革命，Web Services 的出現帶來的，是新一波服務導向(Service-oriented)的網路新科技，Web Services 將讓原本不同的平台，所存在的處處藩籬限制將漸漸消除, 讓原本散落在四方的資訊可以透過 Web Services 的技術逐漸的整合在一起，達到資訊流通的目的。為何 Web Services 可以如此輕易的將資訊整合，而不需面臨太多的技術性的問題？

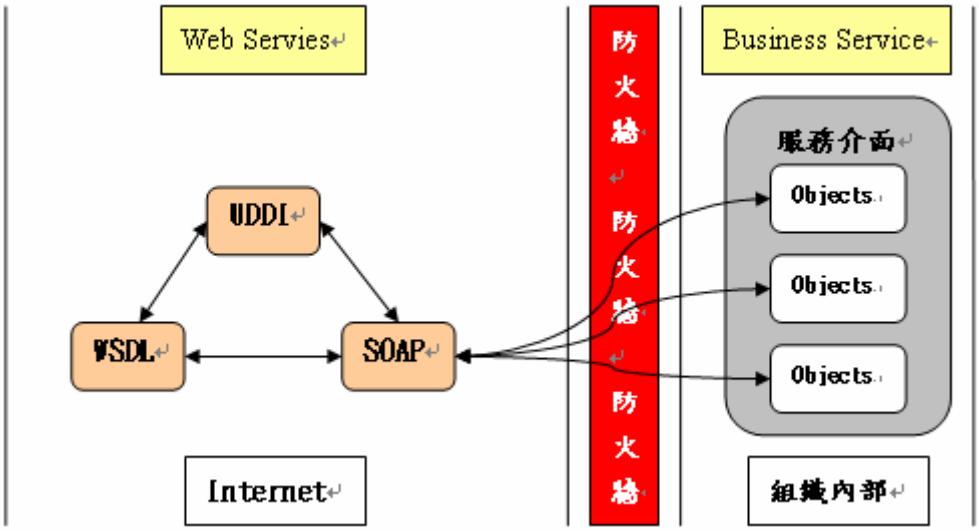

圖 7. 企業之間的溝通管道 -- Web Services[18]

資料來源：eAI Journal 2002 年 1 月—Developing a WebServices Strategy

主要原因也是拜網路興起與發達之賜，網際網路的蓬勃發展帶動許多企業與組織，將許多的網路基礎建設不斷往上提升與加強，而 Web Services 所使用的 SOAP(Simple Object Access Protocal)是建構於 HTTP 通訊協定之上，因此能輕易的穿透各企業組織間的防火牆(Firewall)，使的企業與企業之間的溝通比以往更容易達成「參見圖 7」，這也是 Web Services 近年來快速發展的最主要原因。

Web Services 的構成要素有三項 SOAP、WSDL 與 UDDI[12]，前面的章節已經介紹過，綜合以上資訊，我們可得知 XML 是這些技術的最核心觀念「參見圖 8」，而這三種新技術的核心觀念在於如何去應用，而非作相關技術的深入探討，這也是本文研究的另一個目的之一，因為這三種科技的出現講求的是架構觀念的創新，將過去已有的架構或技術（例如 SOAP 技術便是使用已存在的 HTTP 通訊協定，在 HTTP 上再延伸而出觀念性技術），加以運用與整合才能達到系統更新、整合的最大效益。因此如何善用 Web Services 的技術，便成為資訊系統所擁有的組織，所要思考的一個議題與方向。

XML 在 Web Services 的傳輸協定 SOAP 中扮演極為重要的角色，Mike Rosen & Jim Boak 在其研究中提出兩個原因說明 XML 的重要性：[18]

（一）、XML 提供訊息(Message)描述的完整性與擴充性，這大為改進從前在網際網路上傳送與接收的鬆散結構缺點。

（二）、XML 可以重新呈現 ASCII(American Standard Code for Information Interchange)碼，並且在使用 HTTP 通訊協定傳輸時，可以輕易的穿越組織間的防火牆。

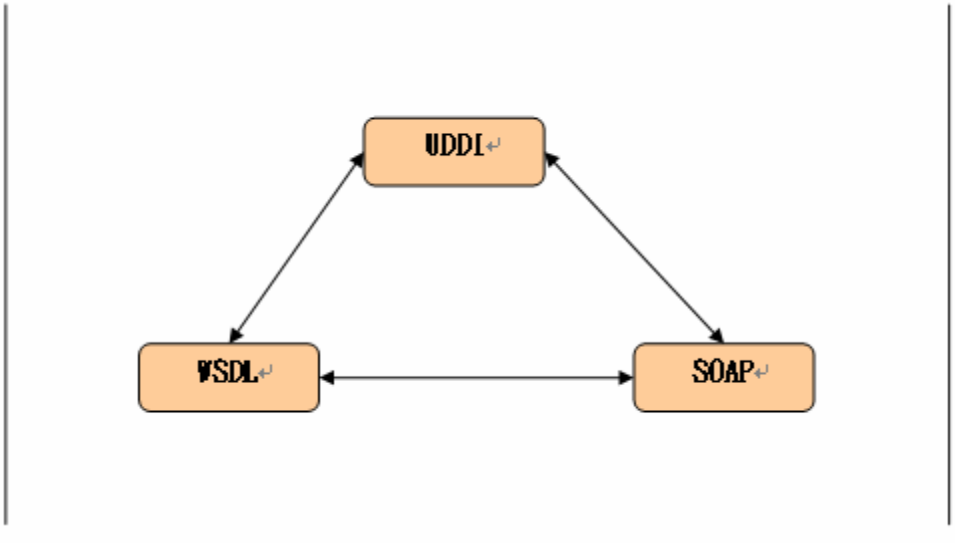

圖 8. Web Service 與 UDDI、WSDL、SOAP 關係圖

二、服務導向的 Web Services

由資料導向 (Data-Oriented) 邁向以服務為基礎的整合 (Service-based integrated)是一個清楚的趨勢。資訊系統的架構、交易(Transactions)、與分散式物件的技術方法，已經被運用的相當成熟，但是 Web Services 的出現卻對這一切產生衝擊，因為由 Microsoft 的.NET 策略我們可以很清楚的明白此趨勢。[5]

Web Services 其重要的概念是「服務」，因此 Web Services 並不是萬靈丹，組織在作更新與整合時，應該清楚的為 Web Services 作出明確的定位，對於此觀點 Don Berman 在其研究中指出，Web Services 的應用應著重在下列幾種資訊：[6]

（一）、金融的財務報表：(Financial calculators) 。

（二）、即時性的金融資訊，例如股市行情：(Timely financial informations) 。

（三）、複雜的報表資料：(Complex calculators) 。

（四）、旅遊與飛機時刻資訊：(Travel and flight informations) 。

（五）、複雜的轉換，例如匯率、溫度等：(Complex conversions)。

（六）、複雜的檢核，例如地址、電話等：(Complex validations)。

（七）、應用程式的通訊問題，例如 E-Mail、傳真等：(Communication from applications)。

三、Web Services 的應用

根據 Bob Dunn 研究，Web Services 是一種新的分散式系統的新典範，主要可以將其所提供的資訊，應用在下列四個方向[4]：

（一）、ERP 的應用程式，在其他的領域或與其他公司的互動。

（二）、企業間的資訊交流。

（三）、客戶的服務入口。

（四）、無線裝置（如 PDA、SmartPhone 等）的連接提供者。

由以上四點我們可以清楚的瞭解到，Web Services 所著重的乃是整合與延伸的概念，因此如何將 Web Services 的觀念，應用與整合到組織內部之舊有資訊系統，並往行動通訊方面作延伸，正是本文所要探討的研究目的之一，因為在上述第一點中所提到的，如果再深入去作探討，就是舊有的系統整合，即使是分散式架構的資訊系統，也有其一定的系統封閉性（例如作業系統平台所造成），如在 Windows 作業系統上的 API 或 COM 都是典型的實例，而 Web Services 的出現可以將這些舊有的限制克服，並將這些舊有的物件或資訊封裝，透過 Web Services 在網路上提供服務。

四、如何善用 Web Services

Steve Vinoski 在其研究中指出，如何善用 Web Services 可從下列四個重點著手：[28]

（一）、普遍存在的基礎建設(Ubiquitous infrastructure)：

Web Services 所應用到的技術，大多屬於舊有的 Web 相關技術（例如 HTTP 通訊協定），因此如何在這些舊有的基礎建設上加強與管理，都是加強 Web Services 效能的的立足點。

（二）、由已被驗證的方面著手(Proven approach)：

Web Services 的運用，可由已經被組織所運用的 MiddleWare 分散式架構資訊系統，加以整合並且相互合作產生最大的效益。此論點與本文研究的目的不謀而合，雖然有差異，但是其觀念大致相似，都是將組織已存在的 MiddleWare 與 Web Services 技術整合並加以運用。

（三）、可延伸式標記語言(Extensible Markup Language，XML)：

XML 為 Web Services 主要的核心技術觀念，如何以 XML 的特性應用在舊有的系統更新與整合上，是本文所研究的目的之一。透過 XML 資料的轉換特性，對於 MiddleWare 所受到的開發環境限制將不存在，例如特殊的 IDLs(Interface Definition Language)限制。此論點與本文研究的第三個研究目的相同，不同的是本文所提出的架構，不需透過 Web Services 就能在 Windows 平台達到此目的。

（四）、商業標準(Business Standards)**：**

在應用 Web Services **時更應注意到企業的組織活動，因為在未有資訊系統前，有的企業組織就已經存在已久，因此如何使用過去已經存在的商業標準，更是** Web Services **能被成功的運用的一大要素。例如電子資料交換**(Electronic Data Interchange，EDI)，**就可說是** XML **的先驅，**EDI **不論現在其效益為何？它仍舊是一項已經存在的商業標準。**

五、 Web Services 所面臨的挑戰

Web Services **被提出至今，即使有那麼多的優點，但其本身仍然有些問題或限制尚待克服，**Bob Dunn **在其研究中指出四點** Web Services **所面臨的挑戰**[4]**。**

（一）、Web Services **的安全標準仍然在還在發展階段：**

雖然 XML **本身的簽章與資料加密是安全的，但是目前的** Web Services **的數位授權與認證，在傳輸上要達到成功是相當困難的。而為了因應此問題，各大廠在發展** Web Services **必定有其各自的策略，例如** Microsoft **在** Web Services **上有關這部分，最近才發行了** Global XML Web Services Architechture(GXA)**來解決此問題。**

（二）、XML **的文件本身能加密，並且安全的傳送到一個或多個接收端：**

針對此點的達成是透過原來已經存在的技術 Secure Sockets Layer(SSL)，**但在某些情況就會出現問題，例如信用卡號的認證，因為由不同的接收端所傳送**

過來，但在進行驗證時，卻要傳送一份相同的結果給不同的接收端，就會產生問題。還有其他有特殊需求的問題，例如個別交易(Transaction)的認證問題。

（三）、有關 Web Services 服務的效能品質，尚有待改善：

這是典型的的魚與熊掌不可兼得的問題，Web Services 的出現在於簡單與整合，由於採用的技術是 SOAP 通訊協定，再加上本身呼叫內部服務的流程處理與相關的問題，自然會犧牲部分的效能，但有關這方面問題的解決，在前面章節 Steve Vinoski 在其研究中指出的第一點，即是此問題的一個解決方法。

（四）、平行交易的問題與管理：

針對此問題與上一個問題相似，是個兩難的問題。由於某些 Web Services 相當的複雜，還呼叫到其他的 Web Services，而且在有關錯誤的處理與訊息的回傳上也尚待克服。這是本文所提出的研究目的之一，作舊系統的更新時，如果採用 Web Services 為全盤的解決方案，或是採取全面大耀進式的更新，勢必會面臨到此問題，因此本文才會以漸進式的方式，將 Web Services 作為延伸的策略，以舊有的 Middleware 透過本文所提出的架構來解決此問題。

Steve Vinoski 也在其研究中指出，許多 Web Services 的 MiddleWare 是被設計成為 Web Services，而這些 Web Services 本身就已經包含原有的商業邏輯(Business Logic)，但如此一來 Web Services 不就可以完全取代分散式架構中的

MiddleWare，不過 Web Services 依然面臨下列三個問題：[27]。

（一）、物件生命週期(Object Life Cycle)：

（二）、物件參照(Object References)：

（三）、錯誤處理(Fault handling)：

Steve Vinoski 所提出的這三個問題，正是本文的研究架構提出用意，因為除了包含原有的商業邏輯之外，如何在保留舊系統的前提下，仍可以接續並引進新技術更是本文所探討的，但並非直接使用 Web Services 作為解決方案，因為在替舊系統注入新生命力，代表著除了 Web Services 之外，在內部的應用程式(Applications)發展上依然其必要性，而且這佔系統一個很大的比率，這也是為何在網際網路出現，且蓬勃發展的今日，有許多舊有的系統持續在發展中。

六、Web Services 與 MiddleWare 的連接

根據 Sam Wong 對 Web Services 的定義指出：「許多採用 Web Services 的企業，都將 Web Services 的焦點擺在工具與科技上，而非 Web Services 能幫他們解決什麼問題」。[30]由此我們便可以知道 Web Services 並非如過去一般的資訊解決方案，也不應該試圖把 Web Services 當成解決方案，應該將它當成是一種整合科技與工具的應用，將它與已存在的舊資訊系統整合產生新的效益。

在探討 Web Services 與 MiddleWare 的整合時, 必須先為兩者作角色定位，Web Services 有許多的特性，但在本文的研究中 Web Services 的角色，在於作為另一個連接舊有的系統出

口, 把舊有的資訊系統透過 Web Service 的特性, 將 Web Services 當成一個延伸的架構, 作為舊有的資訊系統的另一個服務出口。

而原有 MiddleWare 的定位，是繼續扮演著舊有系統的運作重心, 透過 XML 新科技的整合, 將原有的 MiddleWare 架構，在不影響舊有的資訊系統運作下，以漸進式的方式修改，將種種環境的限制，透過以 XML 為資訊的交換格式，讓舊有的 MiddleWare 連接到 Web Services 達到整合目的。

根據 Katherine Hammer 在其研究中指出，Web Services 在企業內部整合時，在三點問題上無法滿足功能性的需求，其內容如下：[14]

（一）、許多 MiddleWare 的產品，使用不同的模組來做溝通。

（二）、對 Web Services 錯誤的認知，認為可以取代原有的商業處理邏輯與介面。

（三）、針對企業整合的維護，Web Services 並沒有提出一套改變管理的需求。

Pual Holland 在其研究中也提到：「當企業組織引進 Web Services，並將舊有資訊系統中有價值的部分，以 Web Services 的技術重新建置, 不但耗費成本且相當耗時」。因此 Pual Holland 提出三個引進 Web Services 的成功步驟, 其三個步驟如下：[21]

步驟一、知道組織本身所擁有的舊系統是什麼。

步驟二、找出舊系統中最有價值的部分。

步驟三、開始建置 Web Services

針對 Katherine Hammer 所提出的上述第一、二個問題，即是本文第三章所介紹的架構之目的，而有關 Pual Holland 所提

的第一、二步驟，是導入 Web Services 時的執行方向依據，因為透過 XML 技術與本文所提出的架構，將 Web Services 與原有的 MiddleWare 連接作為延伸的架構，來因應這些問題與限制。

第四節、服務導向的整合架構

本小節的重點在於服務導向架構(Service-Oriented Architecture，SOA)的概念。介紹 SOA 架構的觀念及為何要往此方向探討的原因。

壹、SOA 的定義

服務導向的架構(Service-Oriented Architecture，SOA)，是下一個應用程式的發展方向，SOA 讓許多異質性的環境與應用程式，讓已存在的應用程式與架構之間找到一個平衡點。[19]過去的 SOA 是由物件所建構而成的，SOA 的核心是將商業處理流程用物件(Objects)透過程式將它表現出來，所以簡單地說舊有的資訊系統是一種狹義的 SOA，因為舊有資訊系統是服務「內部的客戶」。

服務導向的架構過去所呈現，最常見的方式是 Web 化，在西元 2000 年前後，許多公司將其資訊系統的更新與延伸，以 Web 化的方式作為解決方案，大部分所採用的 Web 化發展，均受限於本身的平台語開發環境（例如 Windows 平台的 ASP 或 Unix 平台的 PHP），而當時供應商所提出整合架構，都相當昂貴且系統整合也不容易。Web Services 出現後大為減輕組織在此方面的負擔。

SOA 的觀念對於舊有的資訊系統，在更新整合時屬於延伸性的架構，例如 Web Services 就是 SOA 架構的一種，因為 Web Services 透過網際網路的便利性，來分享共同的商業邏輯流程(Common

Business Logic)或方法(Methods)。本文所提出的第二、三個研究目的，其觀念就是 SOA 的架構觀念，因應時代變遷所作的系統更新與整合，是為了讓新的資訊系統，成為一個可延伸與服務內、外部客戶的架構，而不是單純只為了因應新時代的來臨。

貳、SOA 的效益

根據 Michael S.Pallos 的研究提到，SOA 發展出幾個重大的效益：[19]

一、平衡最初的舊系統投資(Leverage initial investment)：

這與本文的研究動機是相同的，組織過去所投資的系統軟、硬體，如果能再利用等於賦予其新的價值，這也替組織降低成本並增加競爭力。

二、基礎建設的便利性(Infrastructure Commoditization)：

這與後來發展的 Web Services 的觀念是相同的，所強調的在於讓所有的應用程式都能相互溝通。

三、快速的接近市場(Faster time-to-market)：

即是與本文前面所提到的 MiddleWare 改善觀念是相通的，以再利用的觀念為出發點，來縮短過去的組織流程，更快速的提供服務來接近市場。

四、減少支出(Reduce Cost)：

開發新系統的成本，大部分比更新舊系統來的花費大。

五、減低風險(Risk mitigation)：

這與上一點相互呼應，開發新系統的風險遠大於更新舊系統。

六、持續改善商業流程的循環(Continuous improvement cycle for business process)：

七、中心流程處理(Process-centric processing)：

SOA **架構的觀念即是服務，與本文延伸部分觀念相同，都是希望透過所提出的架構能達到上述幾點的效益，而本文架構的提出更期望達到，以較低的成本且較平緩的改進方式，在原有的資訊系統上改進。**

第三章、以 XML 技術為資料交換的管道層架構

綜合第二章的文獻探討可以瞭解到，舊有資訊系統的更新與整合所面臨的三大要素：風險、時間、成本，其中以風險為最大考量，這也是許多銀行還使用幾十年前的舊有的資訊系統的原因，再者新的技術或觀念如 EAI、Web Services，不是成本太高就是時間太長，因此本文以漸進式的方式為策略，改良舊有的三層式架構的觀念，以 XML 為資料交換的格式，配合「管道層」的改良式架構，為舊有的資訊系統注入新活力（新的技術如 Web Services 或行動通訊的應用）。

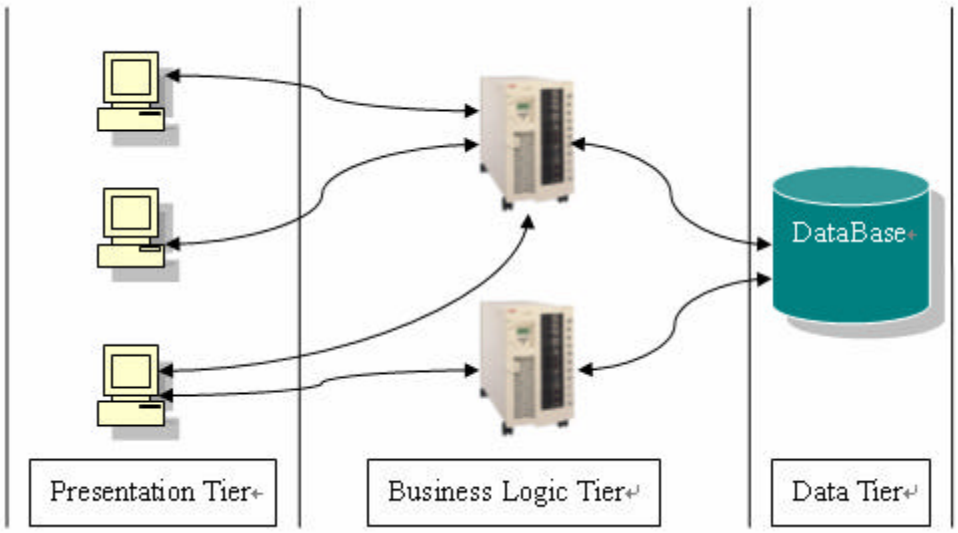

圖 9. 分散式系統架構概念圖

本研究架構的提出在於引進新的科技，來解決現有分散式架構資訊系統的限制，以期望達到「更新」與「整合」的目的。分散式系統架構的概念「參見圖 9」，在系統的延伸發展都受到環境的限制，例如 Microsoft 的 DCOM(Distributed Component Object Module)技術只能應用在其 Windows 的平台，而且 DCOM 內部能包含有許多 Windows 平台的安全限

制，因此工具與作業平台所引發的問題，讓許多在 Widnows 系統下發展的資訊系統受到相當限制，系統只能朝向「更新」的方向漸進，而在系統的「整合」發展上卻困難重重。

因此在本章節中以下列三點為探討的重點，首先探討分散式架構資訊系統的限制，瞭解以元件基礎發展所造成的限制後，再以 XML 為資料交換格式的概念，將元件封裝輸出的資料統一轉換成 XML，最後加上本文所提出的「管道層」架構，達到舊有系統更新與整合的延伸性。

第一節、分散式架構資訊系統的限制

本小節的重點在於說明以元件基礎發展的限制，造成不同開發工具所發展的系統整合上的困難，及因為物件導向的開發技術觀念，所造成的應用程式間的溝通與整合限制。

壹、元件基礎發展的觀念

分散式架構資訊系統的限制，在於物件導向(Object-Oriented)的蓬勃發展所造成，物件導向觀念的應用在資訊系統上，因為不同的平台與不同的工具而造成差異性的加大。企業組織大多採用此方式的原因，來自於元件基礎的發展(Component-Based Development)是一種容易建置系統的方法，而且容易維護不需要相當艱難的技術與程序，其原因是元件基礎的發展本身，具有良好的定義性、高度的重複使用性等優點，因此使用此方式建置系統，可以節省大量時間與成本。

因為在相同環境下所發展的元件，由於使用相同平台、相通的通訊協定與相同的技術背景，更加速了此種型態的發展。而對於此論點的實際呈現，即是 Microsoft 所發展的分散式元物件模組

(Distributed Component Object Model，DCOM)與 OMG(Object Management Group)的共同物件仲介需求架構(Common Object Request Broker Architecture，CORBA)，兩個組織擁有各自的作業系統、開發工具與不同的標準，因此在兩個平台之間的資訊溝通與轉換是最大的限制。

而造成這種限制也來自物件導向的觀念，由於物件本身具有封裝性，因此對於物件之間的溝通，也因為物件本身的封裝性緣故，因此造成不同平台之間的物件溝通障礙，這是採用物件導向設計的系統與平台，所必須面臨的問題與限制。

貳、應用程式間的溝通與整合限制

在此小節中本文以 Windows 作業系統的平台為例，舉出分散式架構資訊系統，在開發時所面臨到的問題與限制，其說明如下：

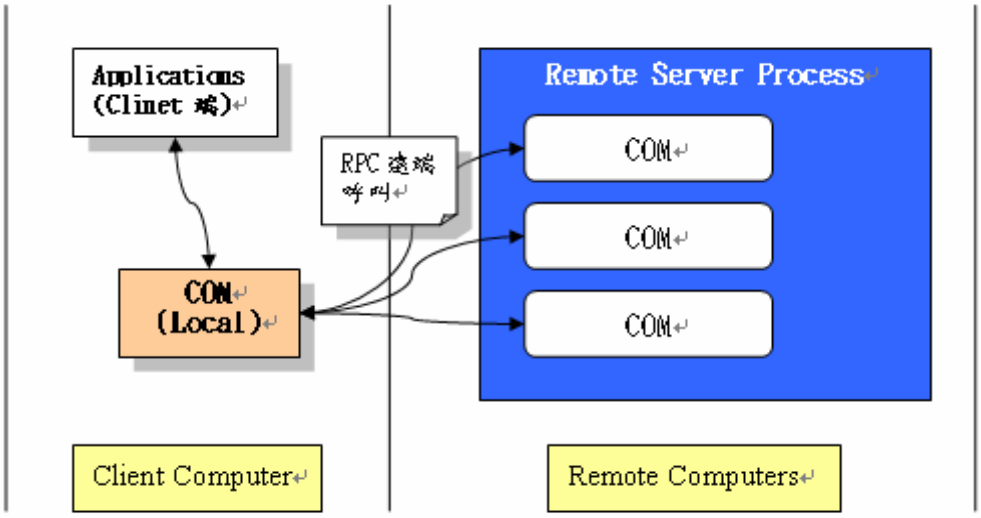

圖 10. DCOM 架構圖

一、元物件模組(Component Object Model，COM)

在相同的機器設備下，Microsoft 的元物件模組(Component Object Model，COM)可以達到最大的效益，因為在相同的作業平台、相同的一部機器、相同的通訊，因此可以發揮出最大的

效益，但真實的世界卻經常超出如此的預設，因此 Microsoft 發展出分散式元物件模組(Distributed Component Object Model，DCOM)來因應此問題「參見圖 10」。

DCOM 是 COM 的延伸架構，最主要的目的是應用於網路上，作為不同機器之間的溝通與資料或訊息傳遞，例如分散式架構的資訊系統如果應用 DCOM 的技術，意指在用戶端(Client)就可以與其他機器設備上的 COM 做互動，而這即是遠端呼叫的概念(Remote Process Call，RPC)，在 Windows 作業系統平台上是一種經常被應用在分散式架構資訊系統發展上的技術。

DCOM 所帶來的便利性，即是由元件的觀念所延伸而來的，在重複使用元件與相同的開發環境下，使應用程式間(Application to Application，A2A)的互動與溝通，可以非常容易的整合在一起，但也因為此便利性限制了系統與系統之間的溝通與互動，甚至在相同作業系統的平台上，因為不同的開發工具即形成了整合上的困難，因為不同的開發工具間的 API(Application Program Interface)，所形成的差異會形成整合上的問題，關於此點會在後面的章節作詳細的介紹。

二、DCOM 的發展限制

Microsoft 雖然以 DCOM 來改善不同機器間的分散式架構，但仍然有許多限制，最明顯的限制即在於通訊協定(Protocal)，DCOM 的最大缺點在於將許多安全上的驗證與作業系統關連。而採用 DCOM 來建置分散式架構資訊系統，其優點在於分散式架構資訊系統，可以提供一個安全性框架，來明確的區分不同的 Client 端用戶，以便系統或應用程式可以知道那

個 Client 端將對某元件進行呼叫與互動。

DCOM 使用 Windows 作業平台所提供的擴展性安全框架，而安全性框架的中心部分是一個用戶目錄，是用來儲存與確認用戶憑證（例如用戶名、密碼、公鑰等必要資訊），DCOM 在使用安全檢查認證的方法時，Client 端必需通過此安全認證，而在採用 DCOM 分散式架構的資訊系統建置，每個 Windows 作業平台的機器上，都儲存和管理許多用戶名稱和密碼，而為了防止用戶進行未授權的互動，因此兩個不同的機器或網域，都會限制 DCOM 的相互溝通。因此即使發展內部的資訊系統，也常在不同的網域(Domain)之間造成問題，因為 DCOM 本身的限制，必須將兩個不同的網域整合，才能便於 DCOM 架構的資訊系統發展，如此造成了一個相當大的技術限制。

三、DCOM 安全限制的改進－Socket 的觀念

為了使 Client 端連上 Server 端提供的服務，使得訊息或資料能夠在網路上相互的溝通，因此有 Socket 的觀念產生，而最常被應用到的網路通訊協定是 TCP/IP，Socket 是一個通訊端點，所描述的是一台 Server 能讓 Client 端使用者，透過 Port 連線進來使用 Service 的概念。

如 Client 端程式要傳遞訊號給 Server 端，則在 Server 端必須開放一個 Port Number（連接埠號），並且告知 Client 端系統服務的位置（包括對方的主機的 IP 位址、所使用的 Port 號碼，如 192.168.1.1:xxxxxxxx），如此 Client 端系統就能夠連結 Server 端的服務，這即是 Socket 的概念，也因為有如此沒有安全認證限制的特性，因此可以解決 DCOM 的部分限制問題，至於

Socket 的安全如何控管，這觀念與防火牆(Firewall)的觀念類似，防火牆的罩門在於 Port Number，Socket 也是如此，至於如何保護 Port Number 不被知道或防止被掃瞄(Scan)，這部分與本論文所要探討的議題無關，因此不作深入的探討。

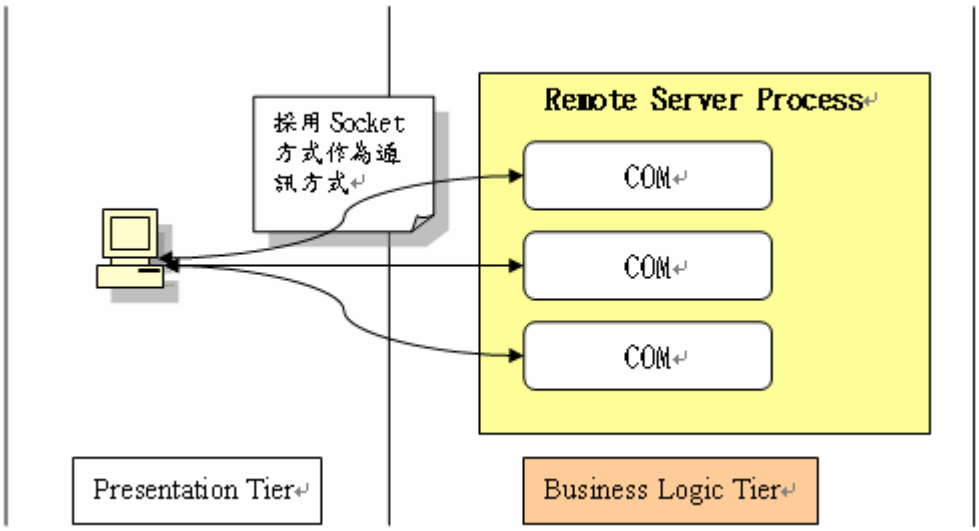

圖 11. 以 Socket 方式作為連接的架構

為了改善 DCOM 所形成的問題，因此有些軟體廠商在此方面發展出不同的解決方法，而 Socket 的採用是往改善通訊協定(Protocal)方面著手，捨棄 DCOM 的架構，改採用標準的 TCP/IP 通訊協定，在 Windows 平台上即是以 WinSock 的方式作為通訊協定，作為不同機器設備之間的互動溝通，而其應用程式間的基礎「參見圖 11」，仍然以元物件模組(Component Object Model，COM)為基本的發展要素，因為用 COM 的標準方法，將可以在相同的作業平台上，以不同的開發工具來發展系統，例如 Windows 作業系統平台上，在 Microsoft .NET FrameWork 架構推出之前經常使用的一種方式，就是以 Microsoft 的開發工具 Visual Basic 開發應用程式，而使用另一種開發工具 Microsoft Visual C++ 來開發某一些特殊的

MiddleWare 的工作。

第二節、以 XML 為資料交換格式的概念

本小節主要詳述以 XML 為資料交換格式所帶來的優點，及過去以元件基礎觀念所發展的系統所形成的資料格式問題，透過以 XML 為資料交換格式，節決資料格式轉換的問題。

壹、解決資料交換與介面呼叫問題

舊有的分散式架構資訊系統，不論是 Windows 作業系統平台或是 Unix 作業系統平台，都會面臨因為元件基礎發展出來的系統底層技術限制，這也是因為物件導向的觀念所延伸而來，例如 Windows 作業系統平台上的 COM 架構，即是一個實際的例子。

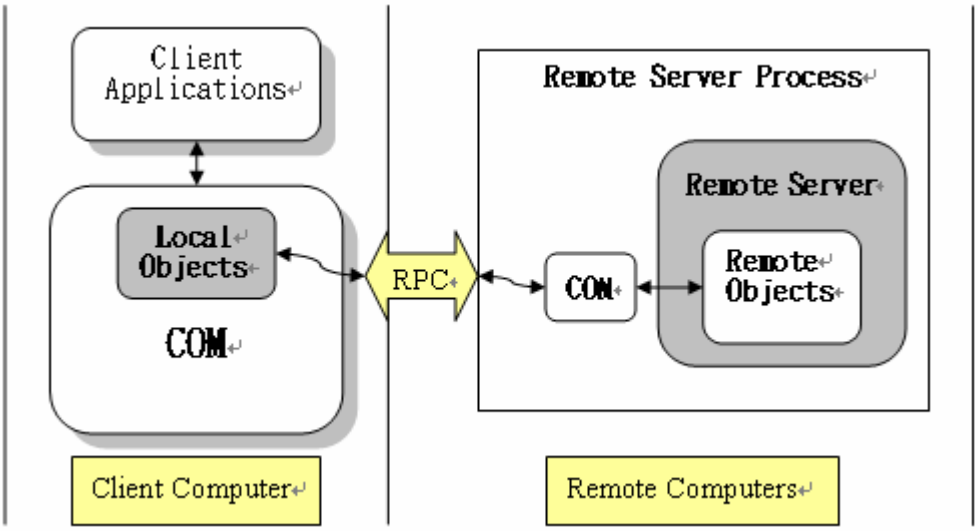

圖 12. 遠端 COM 之間的溝通圖

COM 的基本精神是源自於物件導向程式設計的觀念而來，因此 COM 即具有物件導向程式設計的特性，如某些服務性的 COM 即具有物件的方法(Methods)與屬性(Properties)，因此當 Client 端要連上這些服務性的 COM 時，必須建立起與這些服務性 COM 的關連「參

見圖 12」，而在分散式架構上建立起不同機器間的關聯，即是遠端呼叫處理(RPC)的觀念，而也因為如此的架構與觀念，造成了資訊系統的開發與整合限制，因為必須先瞭解到 COM 本身的 IDL 定義，所以即使在相同的作業系統平台上，資訊系統開發時限制於上述的 COM 架構，這也是許多不同的系統，因開發工具的不同，在 Windows 作業系統平台上所面臨到的一個問題。

在分散式架構資訊系統中，當 Server 端的服務元件，接收到 Client 端的呼叫時，所傳回的資料是經過元件所封裝的資訊，而這些經過封裝的資訊，即是資訊系統的發展限制來源之一，例如 Windows 作業平台上的資訊系統，以 Microsoft Visual Basic 所開發的系統，當 Client 端呼叫到 Server 端的員工基本資料服務時，Server 端往 Client 端傳送的資料，將經過 COM 元件封裝所傳出的資料，必須依照 COM 元件特殊的應用程式介面(API)規格，才能對資料作解析「參見圖 13」，因此常受限於作業系統平台與開發工具。

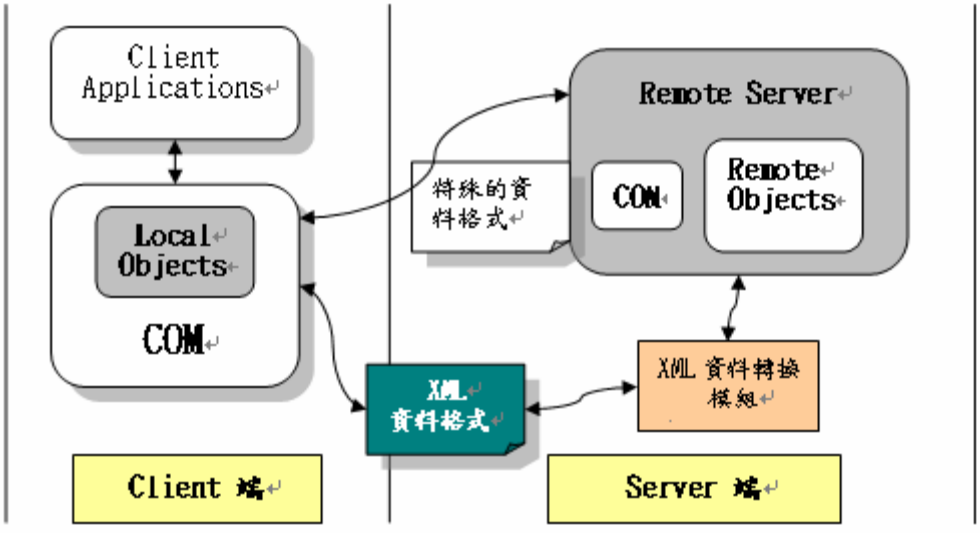

圖 13. 由 COM 所封裝的特殊格式資料

綜合上面的敘述可以得知，解決舊有的分散式架構資訊系統的限制，可由下列兩點問題著手：

一、特殊資料格式封裝問題：

針對特殊資料格式，並不需要由傳統的應用程式介面(API)著手，相反的我們還要保留這種已經存在的模式，在原有的 MiddleWare 上將資料作轉換的動作，即是將特殊資料格式轉換成標準的 XML 資料格式，如此就能克服資料格式所衍生的限制，這也是本文提出的目的之一，以漸進式方式來改進舊有的資訊系統。

二、MiddleWare 的內部元件服務的溝通問題：

針對此問題的解決，利用本文所提出的管道層(Pipeline tier)，連接舊有系統的 MiddleWare，加上 XML 的資料格式轉換，即可解決此問題。

貳、「資料」與「格式」分離

不論主機型系統、主從式架構系統或分散式架構系統，都會面臨到資料、格式與商業邏輯的結合，最後使用 Applications 或 Web 程式，來作最後的呈現資訊目的。

而隨著網際網路的發達，各企業組織對於在網際網路上，資訊的流通與服務需求與變動愈來愈多，而傳統的網際網路程式設計其最大的限制，在於格式與資料混合的程式設計撰寫方式「參見圖 14」，這網際網路程式撰寫方式稱為義大利麵式的程式碼(spaghetti Code) [23]，由於格式與資料混合在一起，因此每當因應客戶或組織內部的需求時，則必須修改這些雜亂的程式碼。

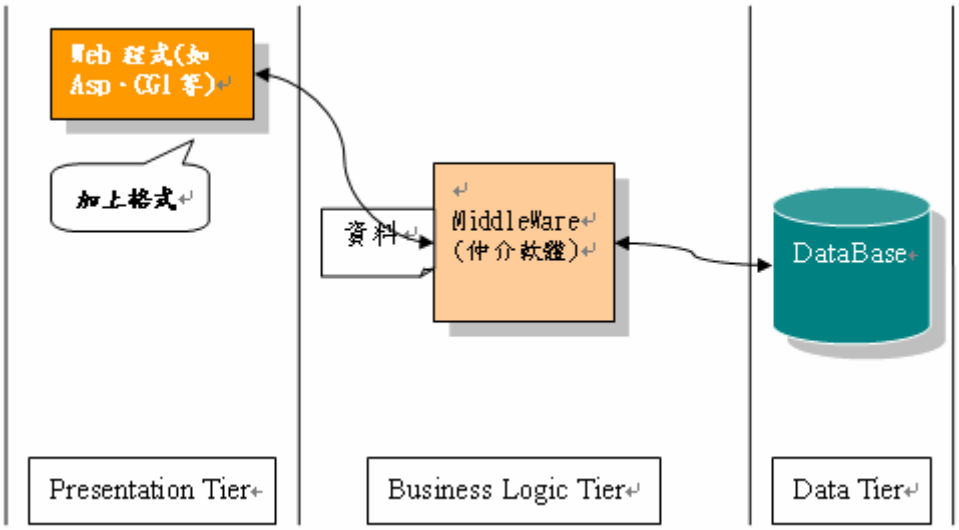

圖 14. 傳統網際網路程式撰寫方式及架構

以 XML 為資料轉換格式，配合本文所提出的分散式系統改良架構，則可以解決此問題，因為這將改變傳統的網際網路程式撰寫方式，徹底的將資料與格式分離，資料的部分由商業邏輯層以 XML 格式送往 Client 端，而格式的部分也是透過 XML 相關技術 XSL，對商業邏輯層所傳送出 XML 的資料作描述，以此種方式即可改變傳統的作法「參見圖 15」。

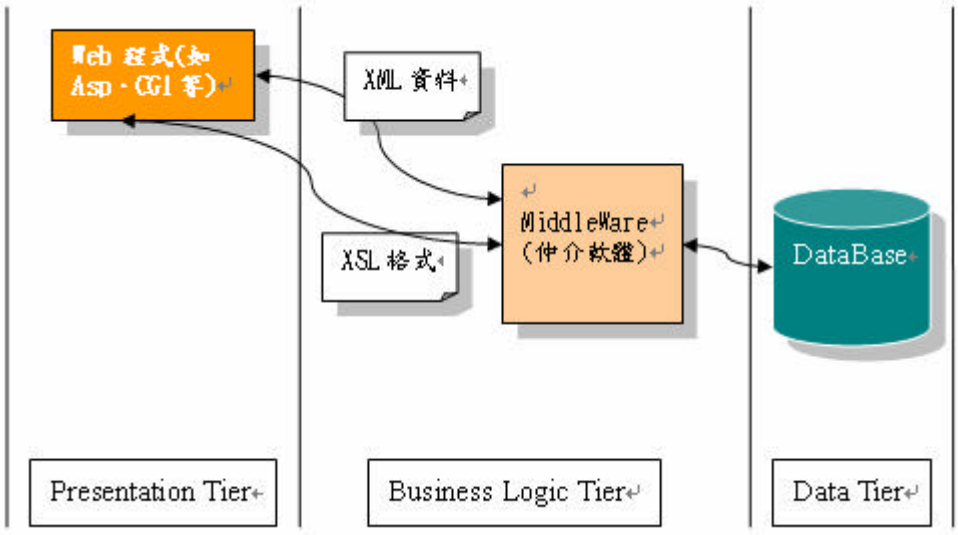

圖 15. 「資料」與「格式」分離方式及架構

當呈現的格式或資訊變動時，只需要針對 XSL 格式描述檔案修

改即可（至於如何修改 XSL 格式檔則可參考相關書籍），不需要如傳統的網際網路程式撰寫方式，針對程式逐一作修改，如此將大量節省系統開發與維護的時間，並且因應組織外部客戶或組織內部使用者的需求時，快速呈現所需要的資訊給相關的需求者。[25]

第三節、XML 資料轉換模組概念

本小節介紹如何將資料轉換成標準的 XML 格式資料，透過本文所提出的轉換模組，以標準 XML 格式資料經由 MiddleWare 的與後端資料庫的存取，達到舊系統「更新」與「整合」的目的。

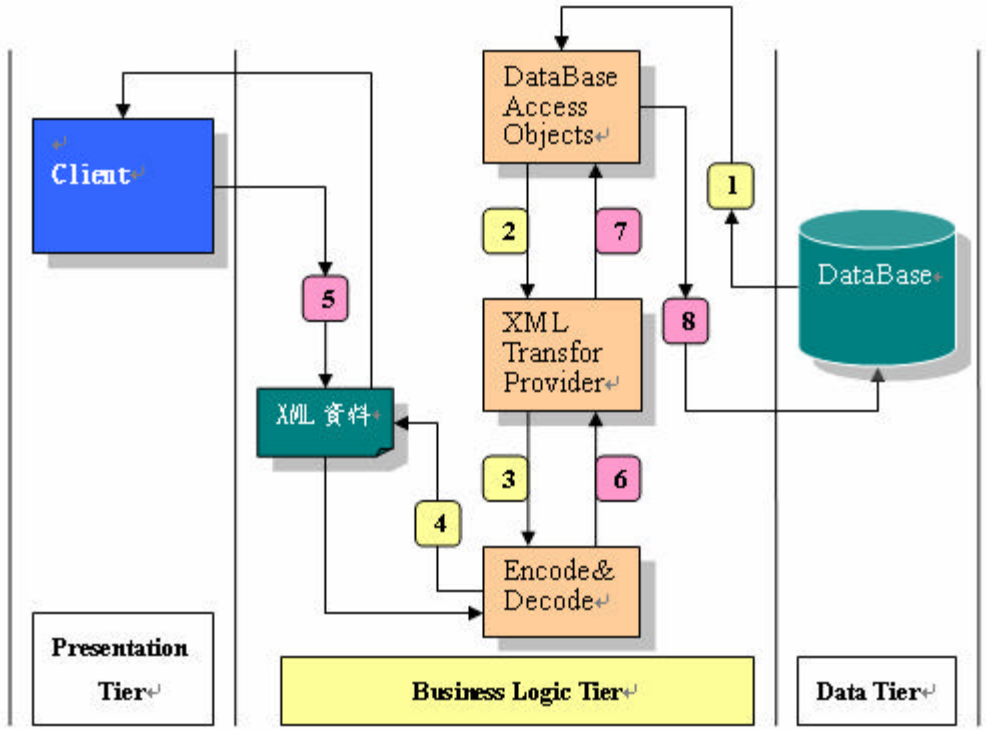

圖 16. XML 的資料轉換模組的運作

而 XML 的資料轉換模組可分為下列三大部分：Database Access Object、XML Transfor Provider、Encode/Decode，透過 XML 的資料轉換模組存取資料庫的資料，如步驟 1、2、3、4 是將資料由資料庫中取出並轉換成 XML 格式的資料，而步驟 5、6、7、8 是各種前端將資料處理後，以 XML

資料格式將資料存回資料庫「參見圖 16」。

壹、Database Access Object（資料庫存取物件）：

Database Access Object 一般存在於 Middleware 中，是資料庫中資料存取的重要觀念之一，例如在本文第四章所介紹的實作系統，其 Database Access Object 所指的就是與資料庫互動存取的 ADO（ActiveX Data Objects）物件，在 Windows 作業平台上是一種常被用來存取資料庫的一種技術。

Database Access Object 在舊有的分散式架構的資訊系統，大多已經存在於舊有資訊系統的 Middleware 中，因此套入本文所提的 XML 轉換模式，正是本文提出的精神所在，如此才能在不異動到舊有的資訊系統架構與運作。

針對 Database Access Object 的呈現可以參考下列函數，即可瞭解大部分舊有的分散式資訊系統，有關 Database Access Object 的實際運作：

```
Function GetPublicNews(Filter-i: String): OleVariant;
   var TmpXML,TmpXMLResult:wideString;
  begin
     with vPublicNews do begin
        Close;
        Open;
        Filtered:=False;
        Filter:=Filter-i;
        Filtered:=True;
      end;//with
   end
End
```

本文一再強調的是針對舊有的資訊系統注入新活力，由上述的函數即可瞭解本文提出的用意，因為上述的函數是存在於舊有的資訊系統中，並沒有因為加入本文所提出的架構，而改變其資料庫架構或是原有的系統架構，這是本文提出 XML 資料轉換模組的目的，

如此才能針對舊有的資訊系統做「更新」與「整合」。

貳、XML Transfor Provider（XML 轉換提供者）

XML Transfor Provider 是針對 Database Access Object 的資料作處理，透過原有的 Database Access Object 再加上 XML Transfor Provider 的轉換機制，將由資料庫中存取出來的資料轉換成 XML 格式，並建立資料庫與 XML 資料格式的 DTD 轉換檔（請參考第四章第一節），如此即可透過 XML Transfor Provider，將資料庫的資料轉換成標準的 XML 資料格式「參見圖 17」。

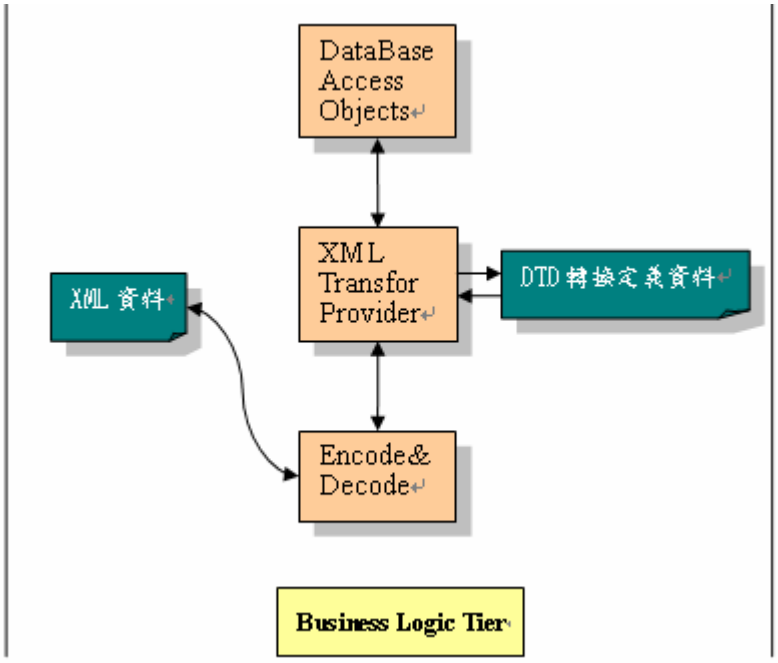

圖 17. XML Transfor Provider 資料轉換 DTD 參照圖

當資料由資料庫透過 Database Access Object 取出時，在經由 XML Transfor Provider 的處理時，XML Transfor Provider 會將事先已經製作好的 DTD 轉換檔（請參考第四章第一節）與資料做轉換的比對，將資料庫的資料透過 XML Transfor Provider 轉換成標準的 XML 格式資料，反過來亦是如此，例如 XML 資料由不同的前端要寫回資料庫，也是透過 XML Transfor Provider 參照 DTD 轉換檔，將資料轉

換成舊有資訊系統 Database Access Object 的格式並寫回資料庫。

以本文所提出的驗證工具 Borland Delphi7，將資料資料庫的資料透過 XML Transfor Provider 的處理，如下列程式碼所示：

```
Function GetPublicNews(Filter-i: String): OleVariant;
    var TmpXML,TmpXMLResult:wideString;
  begin
     with vPublicNews do begin
        Close;
        Open;
        Filtered:=False;
        Filter:=Filter-i;
        Filtered:=True;
      end;//with
      TmpXML:=vOutsideRentXMLTransform.GetDataAsXml('');
   end
End
```

透過 Borland Delphi7 的 VCL 元件 TXMLTransformClient（即是本小節所提的 XML Transfor Provider 實作），參照 DTD 轉換檔將資料庫的資料轉換成標準的 XML 格式資料，此處所舉的範例是 Borland Delphi7 的工具，如果使用其他的工具如 Microsoft Visual Studio.NET，其實作的部分便是以 ADO.NET 將資料轉換成 XML，關於此部分可以自行參考 Dino Esposito 書中的 DiffGram 的轉換方法[9]。

由此可知 XML Transfor Provider 的實際呈現方式，並不是只限定於一種特定的方法，但是針對資料庫的存取讀寫都會透過 DTD 轉換檔的參照，這便是本文提出 XML Transfor Provider 的目的所在。

參、Encode/Decode（編碼／解碼處理）

當資料經由 XML Transfor Provider 轉換後，資料便是符合 XML 的資料格式，但是因應不同國家的語言，因此在 XML 的資料編碼上也有所不同，如本文以繁體中文撰寫，而資料庫的資料也是繁體中文，因此在 XML 的資料轉換便有所不同，如下所示：

<?xml version="1.0" encoding="UTF-8" ?>---((一般的 XML 文件編碼方式))

*<?xml version="1.0" encoding="BIG5"?>---((**繁體中文**))*

*<?xml version="1.0" encoding="Shift_JIS"?>---((**日文**))*

因此如果沒有此程序，所轉換出來的 XML **資料會有編碼的問題。**

透過 Borland Delphi7 **的** VCL **元件** TXMLDocument**（即是本小節所提的** Encode/Decode **實作），將已經轉換好的標準** XML **格式資料，編碼成標準的繁體中文，如此即能將資料編碼成所要的語文格式，如果是不同國家的語言，也是在此步驟將** XML Transfor Provider **所轉換出來的資料編成該國家的語言格式。以** Borland Delphi7 **為實例的轉換原始程式碼如下：**

```
Function GetPublicNews(Filter-i: String): OleVariant;
    var TmpXML,TmpXMLResult:wideString;
  begin
    with vPublicNews do begin
      Close;
      Open;
      Filtered:=False;
      Filter:=Filter-i;
      Filtered:=True;
    end;//with
    TmpXML:=vOutsideRentXMLTransform.GetDataAsXml('');
    XMLDocument.LoadFromXML(TmpXML);
    XMLDocument.Encoding:='BIG5';
    TmpXMLResult:=XMLDocument.XML.Text;
    Result:=TmpXMLResult;
  end
End
```

肆、小結

XML **的資料轉換模組所達到的第一步驟是將「資料標準化」，但有關於達成本文的撰寫目的舊有資訊系統的引展性（如使用** PDA **等裝置）、**Cross-Language **的特性，則必須透過第二個步驟「管道層」的建置，才能達到本文撰寫的目的「參見圖** 18**」。**

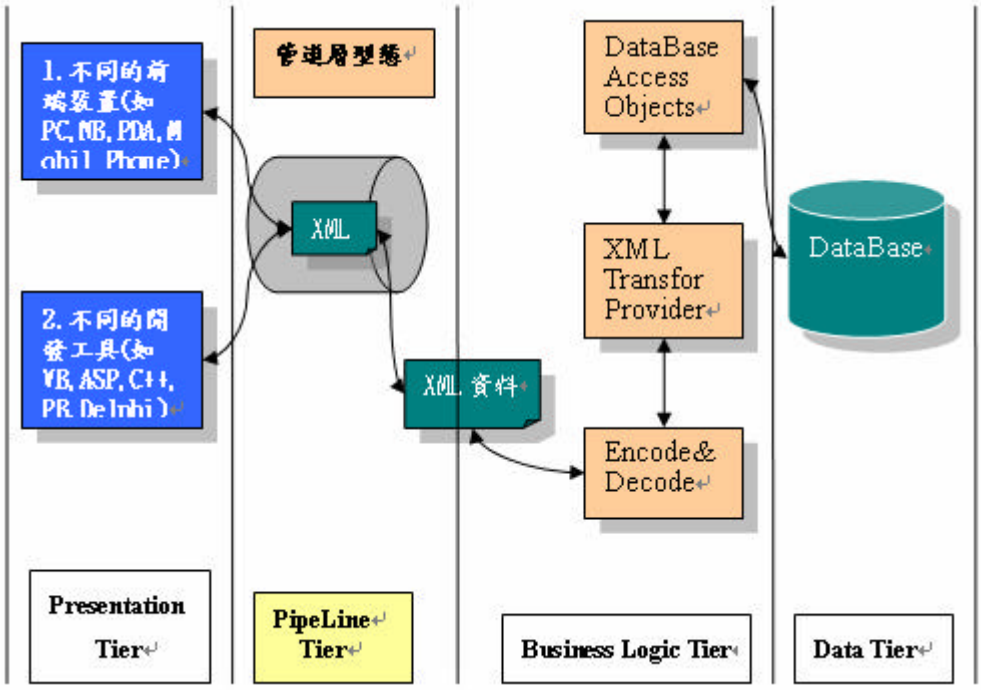

圖 18. XML 的資料轉換模組與「管道層」架構圖

第四節、結合舊有系統的延伸性架構--管道層（Pipeline Tier）

本小節介紹管道層的意義，為何要建立管道層？管道層的改良架構如何為舊有的資訊系統注入新活力，及如何改良舊有系統中的 MiddleWare，配合管道層達到更新與整合的目的。

壹、將舊系統的 MiddleWare 輸出資料轉換成標準的 XML 格式

在前面的章節曾介紹過物件基礎發展的限制，因此將舊有的資訊系統中的 MiddleWare 資料輸出部分，統一將資料轉換成 XML 格式是將舊系統作延伸的第一個步驟「參見圖 19」，如此在相同的平台內的開發工具將更容易溝通，因為自 1998 年 XML 被制訂後，大部分的軟體廠商的開發工具都支援 XML 格式與其相關技術。

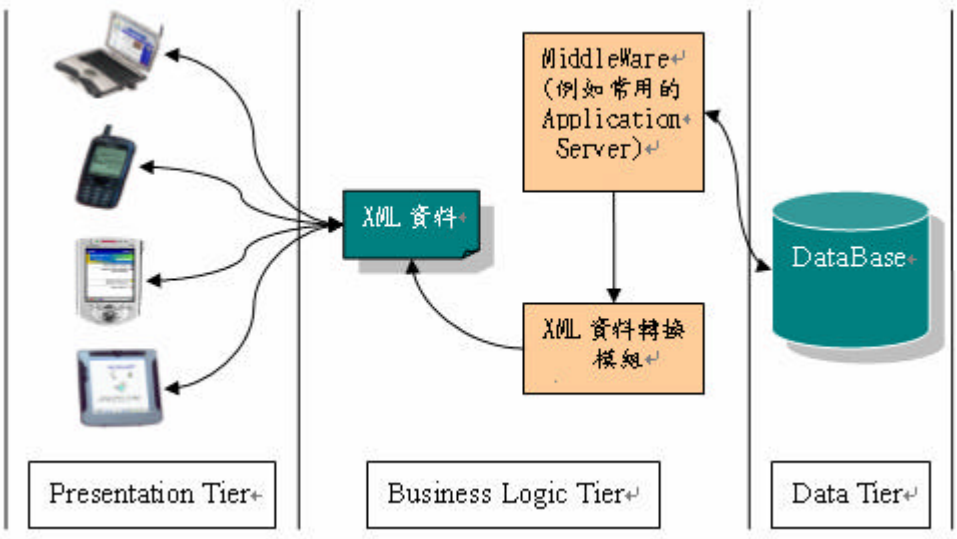

圖 19. 以 XML 為資料交換的架構

如此即能保留舊系統原來的運作方式，這也是本文提出以漸進式改進舊有的資訊系統的動機，透過此方式改良舊有的資訊系統，仍然可以不受影響的正常運作，當初組織所投資在舊系統的時間與成本，其效益依舊繼續存留與組織內部，此部分所帶來的效益，可當成組織管理者對舊系統更新與整合的參考。

而至於 MiddleWare 所傳出的資料如何轉換成 XML 格式，許多軟體開發的廠商已提供相關的工具，例如 Dino Esposito 在其書中，以 Windows 作業平台為例，曾提到 DiffGram 的轉換方法[9]，將資料集(DataSet)透過 ADO.NET 將資料轉換成 XML 標準，而在下一章的驗證中，將以 Window 作業平台為例，使用 BorlandDelphi 開發工具將資料轉換成標準的 XML 格式。

至於如何將舊系統的 MiddleWare 輸出資料轉換成標準的 XML 格式，並非本文的研究重點，因為以現今的技術為例，幾個市場佔有率高的軟體廠商，都有提供轉換的工具或技術，可分為資料庫技術與開發工具技術，其詳述如下：

一、資料庫技術：

例如 Microsoft SQL Server 2000，可以直接以 SQL 結構化查詢語言將結果以 XML 格式輸出。

二、開發工具技術：

（一）、如 Borland Delphi 可以透過 XMLTransformProvider、XMLTransformClient、XMLTransform 等 VCL 元件，將資料轉換為 XML 格式。

（二）、如 Microsoft Visual Studio .NET 中可以使用 ADO 或 ADO.NET 將資料轉換為 XML 格式。

（三）、如 Sybase PowerBuilder 9 中可以透過 Datawindow，以 Export 的方式將資料轉換為 XML。

由上述可知將資料轉換成 XML 格式，已有許多工具與方法可以解決，但本文的研究重點著重在以 XML 的相關技術，為舊有的資訊系統注入新的活力，其重點在於針對舊有系統中的 MiddleWare，先將資料限制與物件封裝的問題克服，再導入本文所提出的管道層，即可讓舊有的系統突破原有的限制。

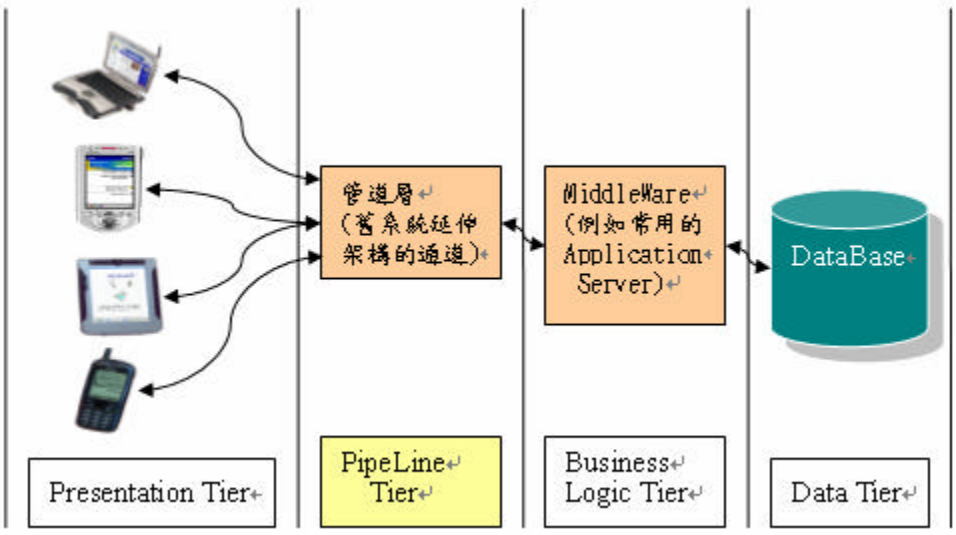

圖 20. 連接舊有分散式系統的管道層

貳、管道層的定義與呈現方式

透過 XML 的相關技術，將資料輸出的格式一致化後，並不能解決所有的前端呈現與連接問題，因此本文提出管道層(Pipeline Tier)來連接舊有的分散式系統「參見圖 20」，配合 XML 的特性，將舊系統的架構作延伸，使得舊有的資訊系統能引入新的技術來作更新與整合。

管道層是一個概念性的層次架構，因應不同的作業系統平台，其呈現的方式也不同，以 Windows 作業系統平台為例，由於 COM 為 Windows 作業系統平台的核心，因此在 Windows 作業系統平台上，管道層的呈現可以使用 COM 的技術來實作，關於此部分在下一章的驗證會有詳細說明。

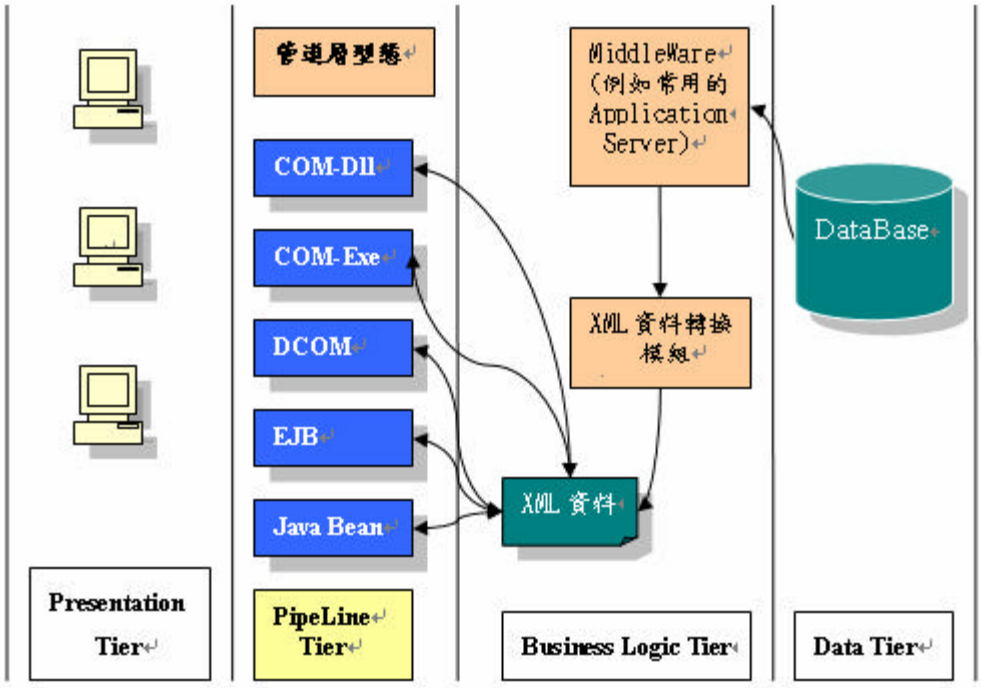

圖 21. 「管道層」的呈現方式

至於其他作業系統平台，例如 Unix 平台上管道層的呈現也可以 EJB(Enterprise Java Bean)的方式存在，或是其他工具所開發出來的型態「參見圖 21」，不限定使用哪種工具與技術來實作管道層，因

為管道層就如同分散式架構的觀念一樣，所提供的是一種概念性的觀念架構。

管道層(Pipeline Tier)的作用，在於連接舊有的分散式系統架構，而其本身並不作任何的動作，只單純作為連接的一層架構「參見圖 22」，透過管道層的建立與 XML 的資料格式轉換，舊有的分散式架構系統，即可透過管道層將系統作延伸，不論是本身發展的平台或是異質的平台，均可透過管道層的連接來達成，此外有關於系統開發的工具，只要支援 XML 格式讀取的相關系統開發工具，即能成為舊系統的開發工具之一，因此對於系統的後續延伸與發展，多了另一種解決的方案。

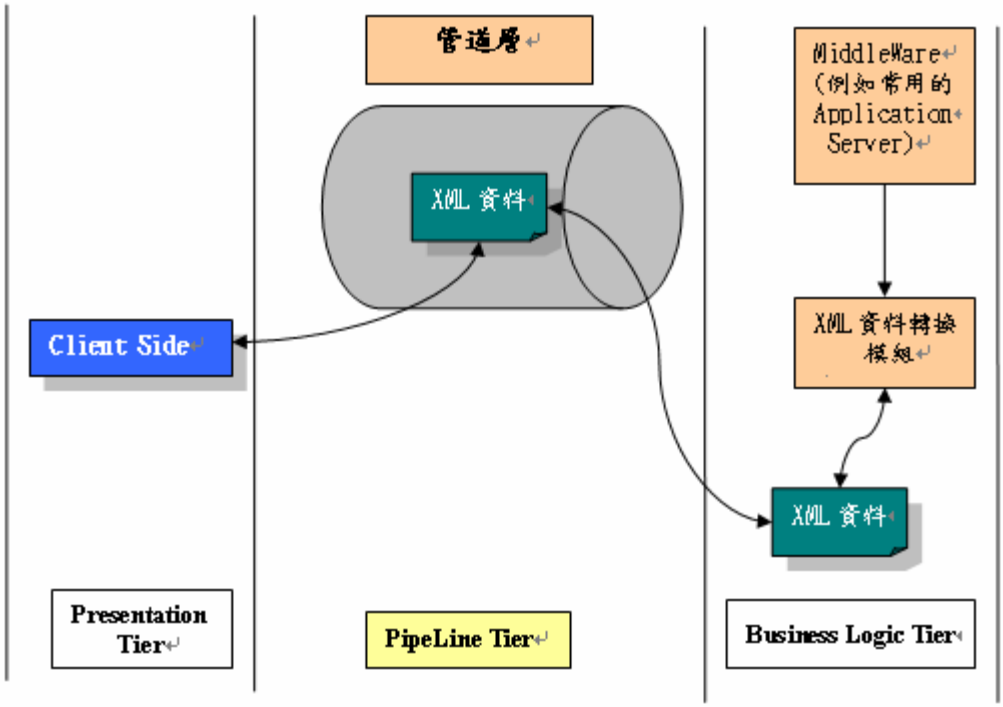

圖 22. 連接 MiddleWare 的「管道層」

管道層的呈現可用不同的方式呈現，例如在 Windows 平台上其呈現的方式，可以使用 in-process 的 COM（Windows 作業平台中的 DLL 檔）呈現，配合 Windows 作業平台上的元件服務，可以達到更

多的系統平台支援與延伸，另一種方式可以使用 out-of-process 的 COM（Windows 作業平台中的 EXE 檔），例如 Borland Delphi 所開發出來的應用程式伺服器(Application Server，App Server)，就是以這種方式呈現，此外更可以利用現有的 App Server 工具來呈現，有關管道層的呈現與實作在下一章中有詳細的敘述。

參、管道層的延伸性

管道層的延伸性來自於與 XML 的相關技術整合，關於此點本文由下列兩點作探討：

一、異質平台間的溝通：

SOAP 通訊協定的制訂是異質平台系統間的溝通管道之一，也是 Web Services 盛行的原因之一，舊有的分散式資訊系統，透過管道層的連接 Web Services，即可達到異質平台間的溝通的目的「參見圖 20」。

但 Web Services 的應用不在於重新建置，而是透過管道層連接舊有分散式資訊系統，將舊有系統的價值以此方式呈現，此乃是本文提出此架構的精神所在，以此方式為舊有的資訊系統注入新的生命力。新一代的行動通訊例如 PDA、MobilePhone 等，皆可以透過 Web Services 連接管道層，而達到企業組織行動化（M 化）的目的「參見圖 23」，有關異質平台間的溝通，在第四章中的實作有詳細的介紹。

於圖 20 中可知道管道層的重點只在於將 XML 資料，由舊有的資訊系統的 MiddleWare 將資料透過 Web Services 的特性，傳送到各種前端裝置。至於如何撰寫各種前端裝置的程式，並非本文所要探討的重點，在此不作深入的探討，因為在市面上

已有 Microsoft 的.NET Framework 架構，而 Microsoft 更以 Visaul Studio.NET 此工具配合.NET Framework 架構，達到各種前端裝置的開發為市場行銷導向，而在有關 Java 的架構上 Borland 公司所推出的 Jbuilder8 所支援的 J2EE(Java 2 Enterprise Edition）也同樣支援 J2EE 上各種前端裝置的開發，關於此部分可自行研究探討。

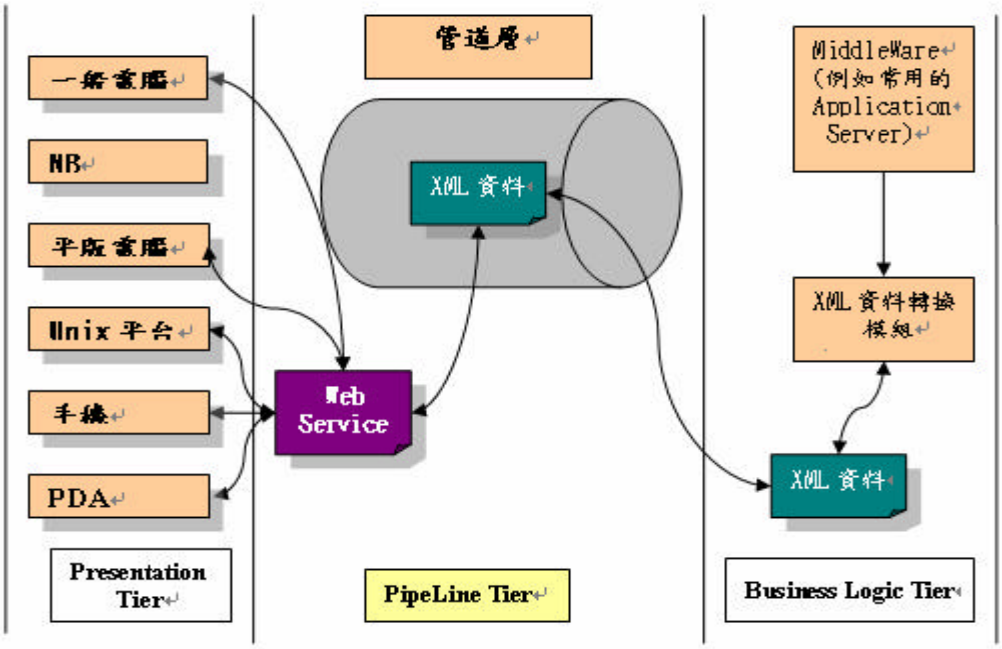

圖 23. 透過「管道層」連接各種裝置

Web Services 的介紹在第二章有詳細的介紹，透過 Web Services 可以輕易的在不同的平台間作溝通，這也是 Web Services 問世的重要貢獻之一，但本文的目的並非研究 Web Services，而是使用 Web Services 的新技術，並且與舊有的資訊系統作整合。例如在 Microsoft Windows Server 2003 的元件服務中，就可以輕易的將元件服務中的 COM 元件，轉換成 Web Services 的服務[37]。

而在上面章節也介紹過管道層的呈現方式，也可以是

Windows 作業平台的元件服務，因此有關異質平台間的溝通，可以透過 Web Services 的相關技術來達成，這也是本文的研究目的所在，透過「以 XML 為資料交換的格式」與「管道層」的改良式架構，達到為舊有的資訊系統注入新的活力。

二、跨語言(Cross-Language)的開發工具：

跨語言的特性來自於 XML 相關技術的發展，但與 Web Services 的意義相同，本文提出此架構的用意，不在於作底層的技術探討，而是強調新架構的整合與發展，本文的架構著重於整合與應用，因為 Microsoft 在 2002 所推出的產品 Visual Studio .NET, 本身即具有跨語言的特性, 因為 Visual Studio .NET 的跨語言的特性來自於.NET Framework 的架構[36]，關於此部分的研究可以參考相關資料，但本文架構的延伸應用不同於 Visual Studio .NET，且其所探討的作業系統平台並不只侷限於 Windows 作業平台。

以 XML 為資料交換格式，所延伸出的跨語言的特性，將更有利於舊有資訊系統的發展，例如：過去導入舊有資訊系統的年代，在因應許多整合與發展所遇到的發展工具限制，如今透過管道層的連接「參見圖 24」，即可解決過去的問題，並且可以利用不同開發工具的特性，來完成舊系統的更新考量，例如：在發展網際網路程式時，即可以採用 ASP.NET、ASP、CGI 等各種工具作開發。

跨語言開發工具的延伸特性來自於 XML 的相關技術，現今多數的開發工具都有支援 XML 的相關技術，如 Microsoft Visual Studio .NET、Borland Delphi7、Sybase PowerBuilder9 等，

以 XML 為資料交換格式，透過 DOM(Document Object Model) [38]、SAX(Simple API for XML) [16]的相關技術來存取 XML 資料，達到開發工具多元化的目的，關於此部分的實作在第四章有詳細的介紹。

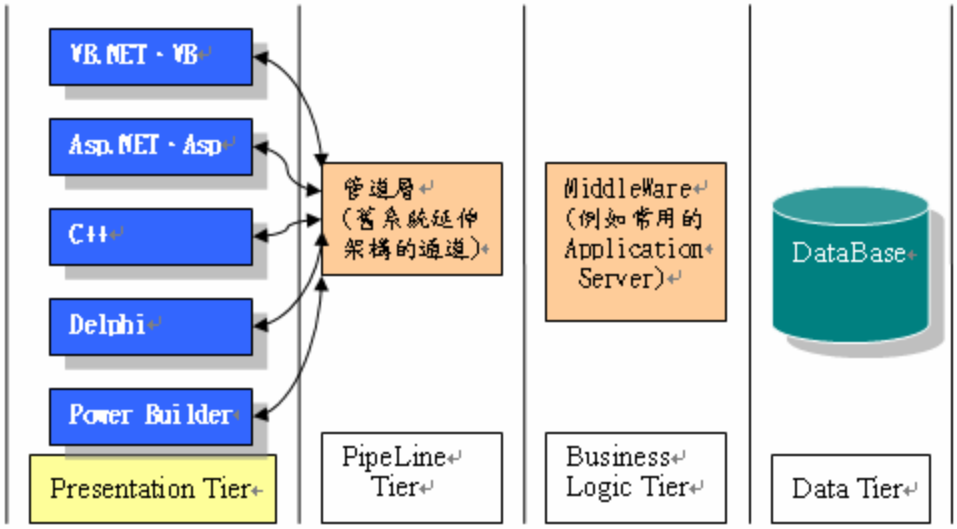

圖 24. 連接管道層的跨語言架構

針對跨語言的開發工具延伸，在此可分為防火牆內部與防火牆外部兩種方式來做探討：

（一）防火牆內部的可以使用 RPC 的方式作為溝通：

防火牆內部的部分可以使用 RPC 的方式作為溝通的方式，因此當「管道層」建立後各種開發工具，就可以透過 RPC 的呼叫方式連接到「管道層」「參見圖 25」，至於為何可以達到跨語言的開發，主要的原因是 XML 的標準為各家軟體開發廠商所認同，且其新一代的開發工具都支援 XML 的格式。

而 RPC 的觀念是分散式架構中重要的觀念之一，雖然各種開發工具使用的方法不同但都能支援，而過去不同的開發工具間無法達到此目的最重要的原因，在於資

料格式的不一致且有關於物件封裝所造成的問題也無法解決，例如 Microsoft ASP 在 Windows 作業平台上，無法直接呼叫某些標準的 COM 即是一個最佳例子，因此本文提出的「管道層」改良架構是解決此問題的方法。

使用 RPC 呼叫只能用在相同的開發平台上，針對異質平台間的溝通可以透過 Web Services 的方式解決，Web Services 雖然可以輕易的達到異質平台間的溝通，但相同的必須付出的代價就是執行效能，有關 Web Services 執行效能問題的解決後面的章節有詳細的介紹。

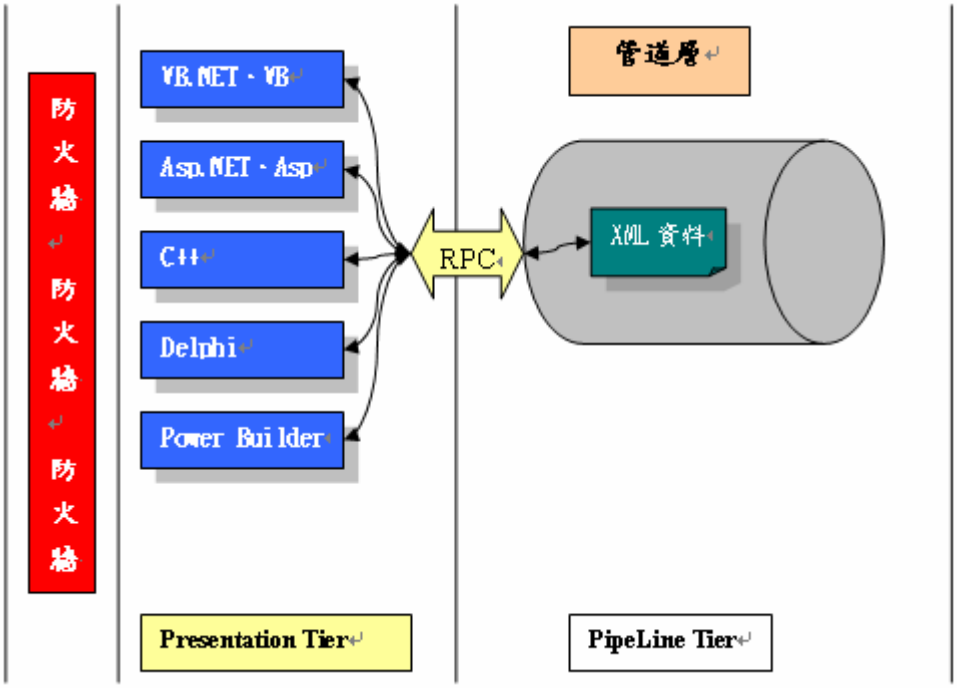

圖 25. 在防火牆內連接「管道層」的跨語言架構

（二）防火牆外部的可以使用 SOAP 的方式作為溝通：

防火牆外部可以使用 SOAP 的通訊協定，因為 SOAP 的通訊協定是建構在 HTTP 之上，因此可以輕易的穿越防火牆連接到「管道層」「參見圖 26」，加上傳輸的資料是 XML 格式，因此可以容易達到跨語言的開發工具整合

目的，且使用 SOAP 的通訊協定加上 XML 資料格式的特性，可以達到異質平台間的溝通，但本文的研究重點在於舊有資訊系統的新技術更新與整合，因此有關 Web Services 的相關技術應用在此不作深入的探討。

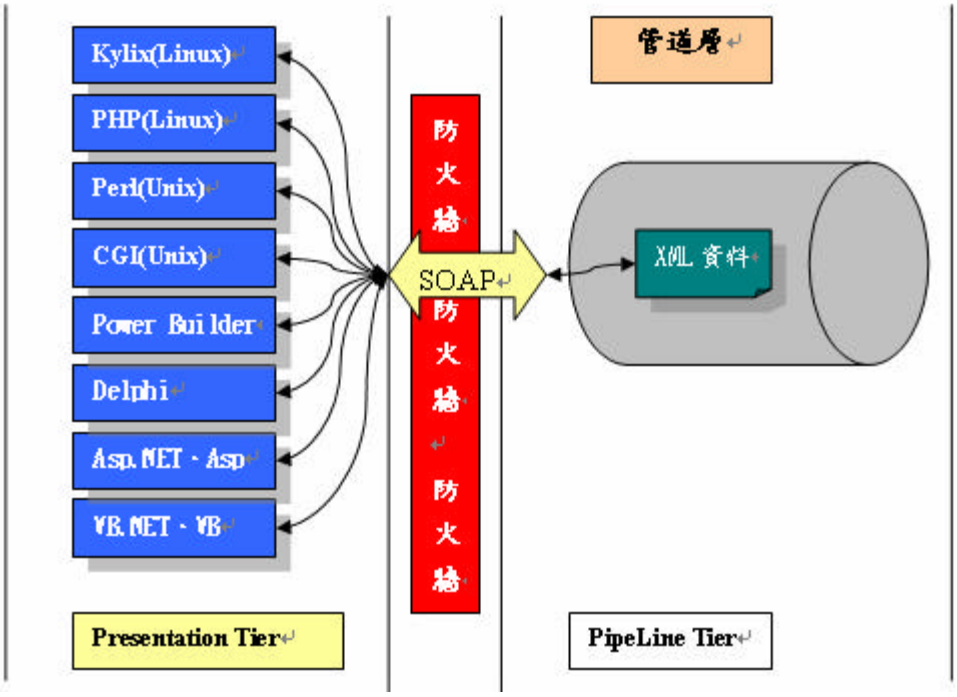

圖 26. 在防火牆外連接「管道層」的跨語言架構

肆、管道層與防火牆

管道層與防火牆的關係，可視為組織內部的第二道防火牆，這觀念與 Web Services 的應用有非常大的關係，因為在組織內部所設立的第一道防火牆，即是傳統的防火牆，一般的資訊系統無法容易的在網路上溝通，與傳統的防火牆有很大關係，因為傳統的防火牆除了一些必要的通訊埠（例如 HTTP 通訊 80 port）之外，大部分的通訊埠都被檔在防火牆外部，如此一來即可保證防火牆內部資訊系統的安全嗎？當然是不一定，因為許多攻擊或資料竊取，都是透過正常管道而來的，例如分散式阻斷攻擊(Distributed Denial of Service attacks，DDoS)，更甚者如資料隱碼攻擊(SQL injection)，這種因為

資訊系統設計的疏失，都是造成危險的原因之一。

因此管道層可在舊資訊系統，引進 Web Services 這類新技術時，當成是組織內部的第二道防火牆「參見圖 27」，任何透過管道層出來的資訊或服務，在設計與系統考量時都可在管道層將不安全的設計避免掉，對系統作更進一層的保護。

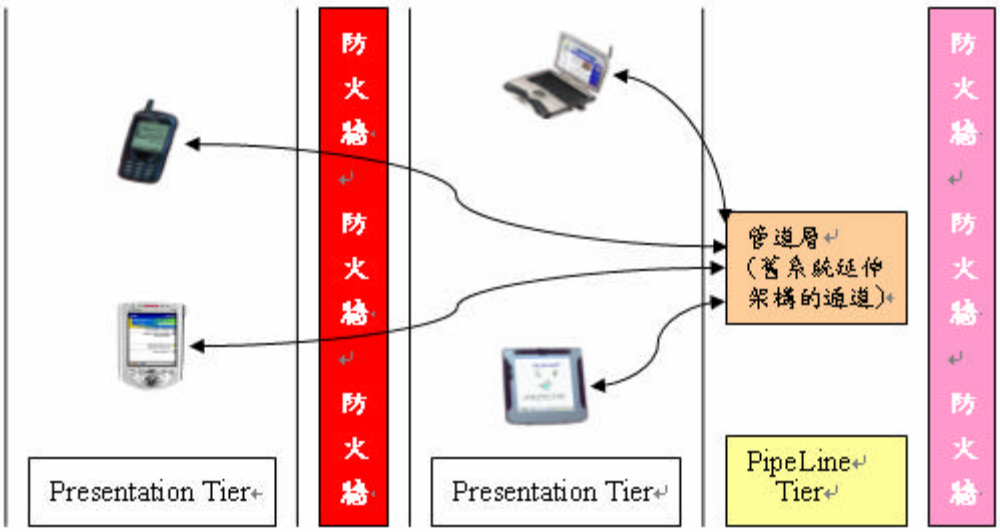

圖 27. 管道層等於第二道防火牆

第五節、延伸性架構的效率探討

本小節介紹管道層的延伸探討，在本章第四節已詳細介紹，但針對不同平台間的溝通、防火牆外的延伸溝通，都必須透過 Web Services 的相關技術達成，而本章節主要探討延伸效率問題，在第二章的文獻探討中已詳細介紹 Web Services 的特性與問題，本小節將配合本文所提出的「管道層」，加上（曾清義，民 90）在其研究中所提出的「中間應用層之改良模組」[1]，形成一個提高 Web Services 改良式架構。

壹、延伸性架構的執行效率瓶頸

Web Services 最大的問題在於速度太慢，這是 Web Services 問世

以來的最大問題，而針對速度的問題不外乎過去的解決方式，例如加強硬體設備的建置（通常所費不貲），或是採用整體架構的分析解決方案，如過去 Windows 作業平台的 DNA（Distributed interNet Applications）架構或是 Oracle 公司最近所推動的 Oracle9i Total Solution，而這些解決方案一樣所費不貲，因為 Web Services 的執行效率瓶頸在於商業邏輯層，不論是重新撰寫、採用舊有的系統架構更新、甚至導入本文所提出的「管道層」改良架構，都會面臨到此問題。

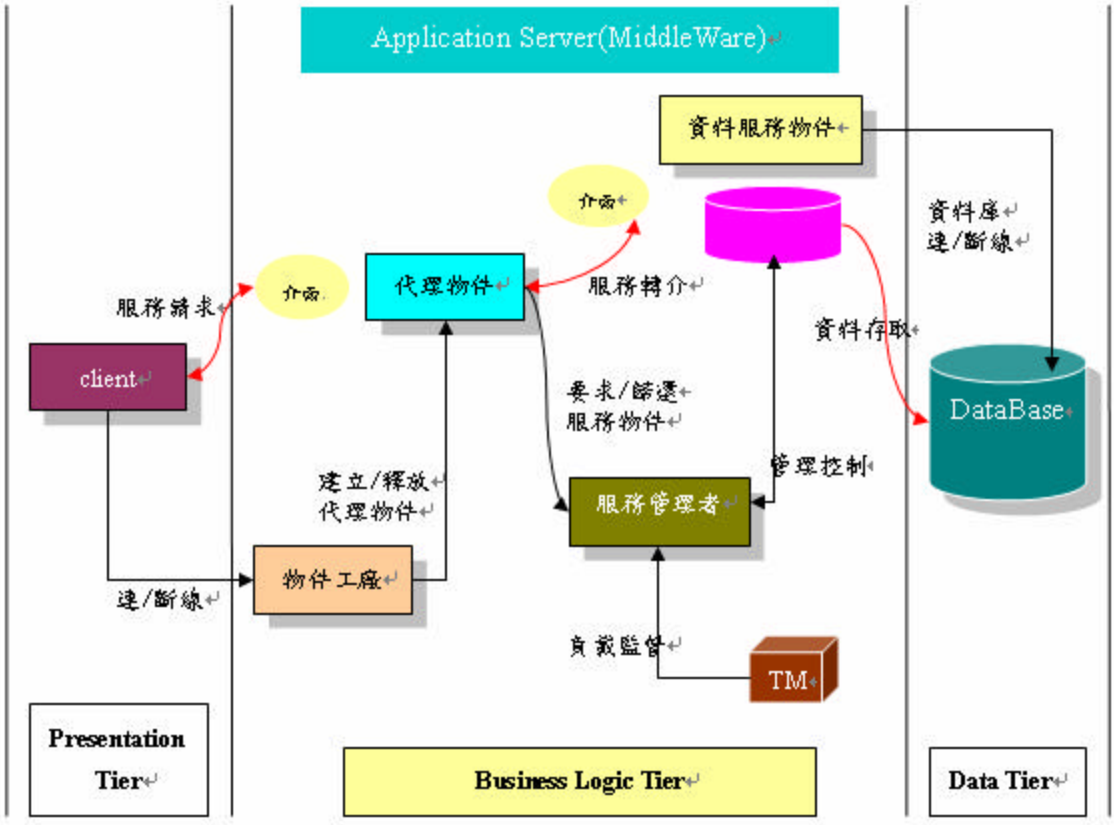

圖 28. 中間應用層之改良模組[1]

資料來源：曾清義，民 90，「中間應用層資料服務物件模型之改進研究」

（曾清義，民 90）在其研究中所提出的「中間應用層之改良模組」[1]，所要解決即是加強商業邏輯層的效率「參見圖 28」，解根據其所提出的方法與實驗分析模組，經實際應用於南華大學校務行

政系統的架構上，確實提高中間層（商業邏輯層）的執行效率且費用極為低廉，以南華大學的選課系統為例，所採用的伺服器主機為一般的 PC（個人電腦）共三台，因此其費用的低廉程度可想而知，至於執行效率可參考其論文研究，而實際執行為南華大學校務行政系統、選課系統，自 89 學年度開始實施至今，都採用其所的改良模式運作於系統中。

貳、延伸性架構的改良

本文所提出的 XML 的資料轉換模組，可以結合在（曾清義，民 90）所提出的「中間應用層之改良模組」中[1]，加上本文所提出的「管道層」所延伸出來的種種特性，都可以在加入「中間應用層之改良模組」後，將原有的執行效率提升，其結合後的架構「參見圖 29」主要是將原有的商業邏輯層再加強。

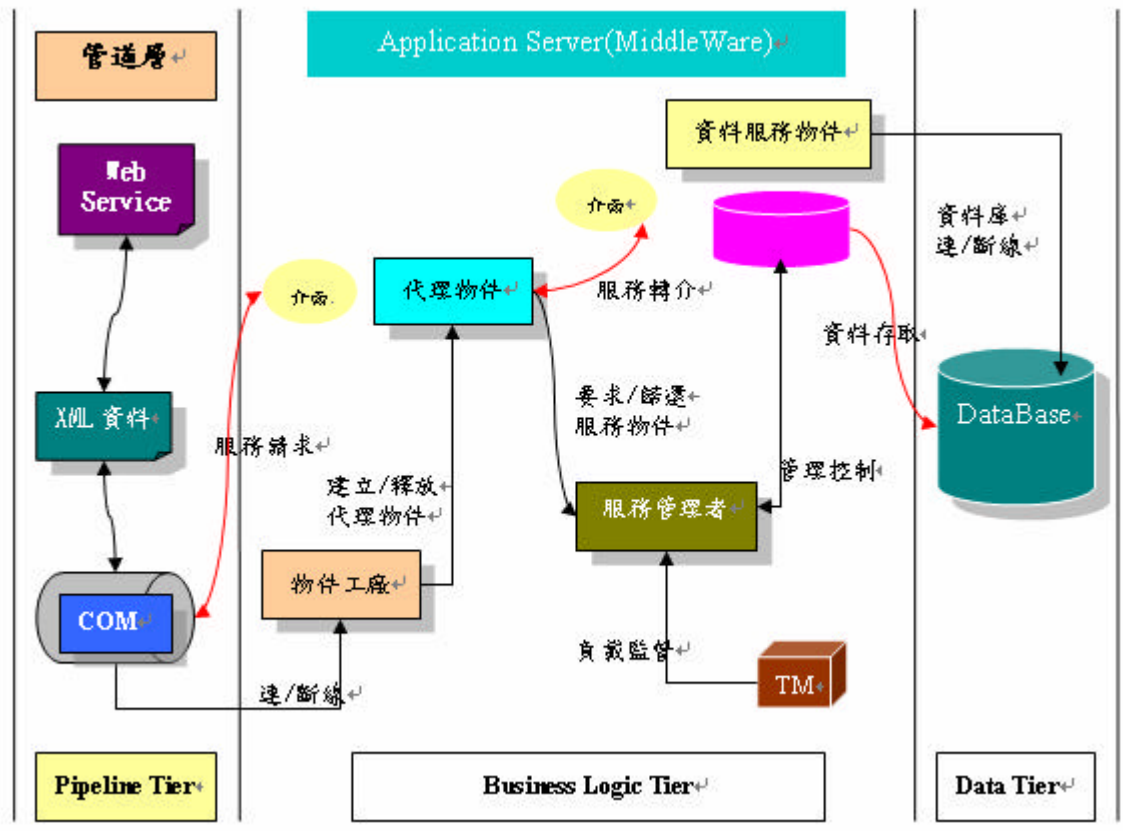

圖 29. 「管道層」結合中間應用層之改良模組的架構

由「圖 29」中可以瞭解到，本文所提出的 XML 的資料轉換模組並非消失，而是結合到商業邏輯層中（實作的原始程式碼，可參考附錄一），而「管道層」則加入到前端與商業邏輯層之間，如此可以將「中間應用層之改良模組」的特性套入本文的研究中，不止可以解決 Web Services 的效率問題，且針對原有的舊有的資訊系統，如能導入「中間應用層之改良模組」的技術，更能提升原有的系統效能。

第四章、驗證與範例實作

在第三章中本文提出一個管道層的改良式架構，透過將舊有的分散式架構資訊系統，將資料轉換成標準的 XML 資料格式。在此章節中將以 Windows 作業系統平台為實作的環境，透過 Microsoft Visaul Studio.NET（2003 Final Beta 版）及 Borland Delphi 7 等開發工具為實作範例的驗證工具，並以下列所提出的三個步驟來轉換舊有的分散式架構資訊系統，使舊系統能導入本文所提的改良式架構中。

第一節、舊系統應用管道層的驗證實作

本小節介紹如何將舊有資訊系統導入本文所提的架構中，透過下列三個步驟：

步驟一、建置轉換 XML 的介面與方法

步驟二、建置管道層

步驟三、透過管道層連接 MiddleWare 的 Web Services

不論是主從式架構或是分散式架構系統，如此就能為舊有的資訊系統注入新活力。而本文的驗證採用南華大學校務行政系統為實際驗證系統，南華大學校務行政系統是以 Borland Delphi 所開發的分散式架構資訊系統，而此章節將逐一介紹上述三個步驟與範例實作。

壹、建置轉換 XML 的介面與方法

針對舊有的資訊系統導入本文的架構，第一個步驟先建置物件導向設計的介面與方法，舊有的資訊系統如果為大型主機系統或主從式架構系統，則必須執行此步驟先建立出 MiddleWare 這一層，如

果是分散式架構系統則不需要，因為不論哪種作業系統平台，只要是採用元件發展方式的物件導向設計，都存在相同的觀念。

當確定舊有資訊系統的型態後，並完成標準的介面與方法的設計後（或是舊有的資訊系統早已經存在），再來就是將資料轉換成標準的 XML 資料型態，而針對資料轉換的處理，依照 XML 的相關技術先建立轉換檔，作為 MiddleWare 資料轉換成標準的 XML 資料型態之用，本文採用 Borland Delphi 中內附的工具(XML Mapping Tool)將轉換檔格式檔案建立出來「參見圖 30」。

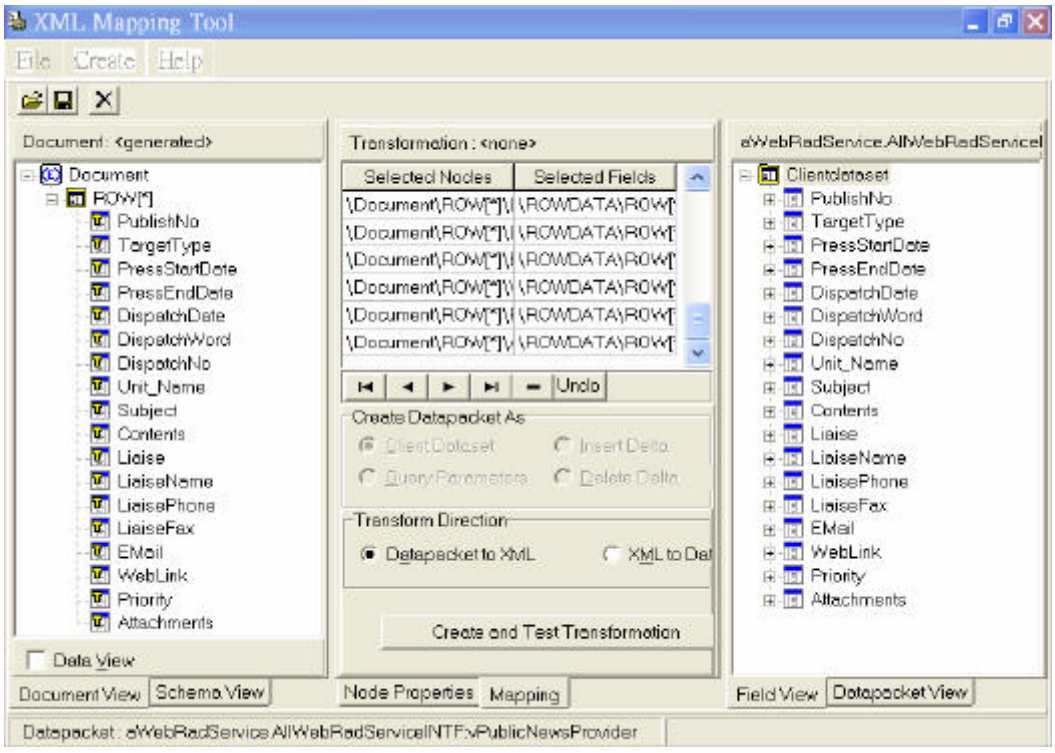

圖 30. Borland Delphi7 XML Mapping Tool

轉換檔是一個 XML 型態的檔案「參見圖 31」，主要作為資料轉換用的對照檔，當舊有系統中的 MiddleWare，要將傳統的資料封裝格式轉換成標準的 XML 格式檔案時，就必須先定義出轉換檔格式檔。

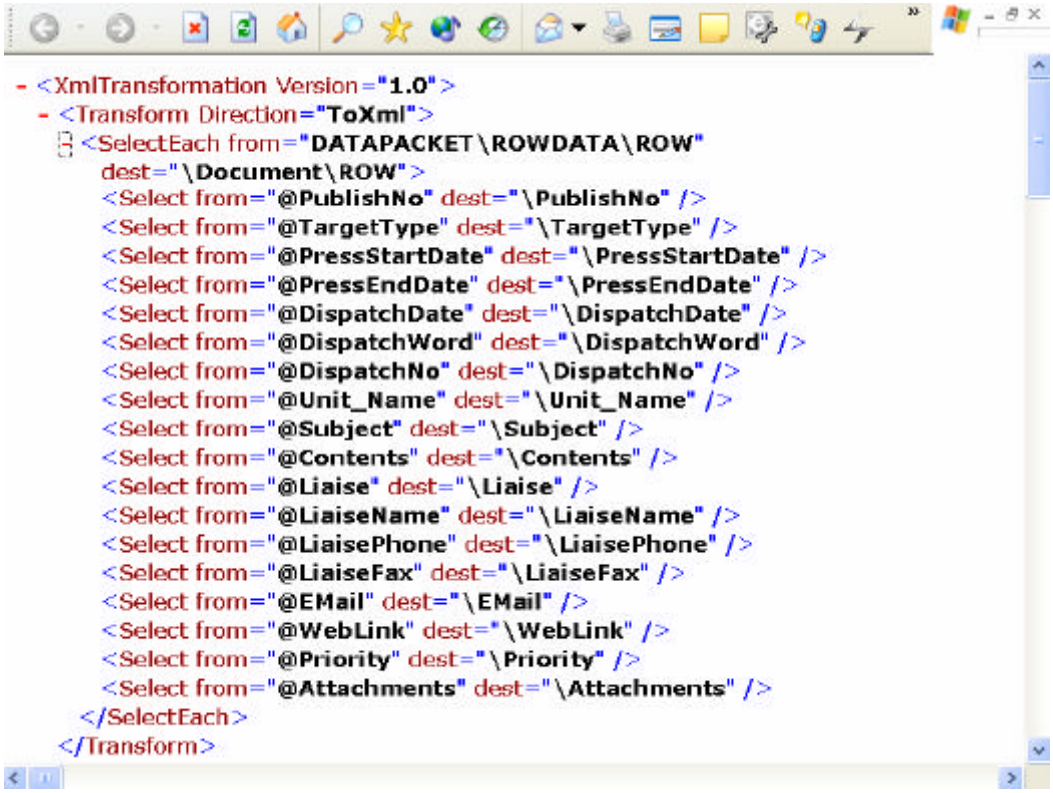

圖 31. 轉換檔的內容

當轉換檔建立後，下一個動作就是沿用舊有系統中介面的方法，將資料透過轉換檔的輸出成標準的 XML 格式「參見圖 32」，這也是本文為何提出此改良架構的用意，因為經過此步驟舊有的分散式架構資訊系統，原先的 MiddleWare 所提供的介面與方法繼續存在著，而原有的應用程式或其他功能可以正常的運作，這也意味著過去所投資的舊有資訊系統，為當初的商業邏輯建置所花費的時間與成本，找到一個再利用的價值。

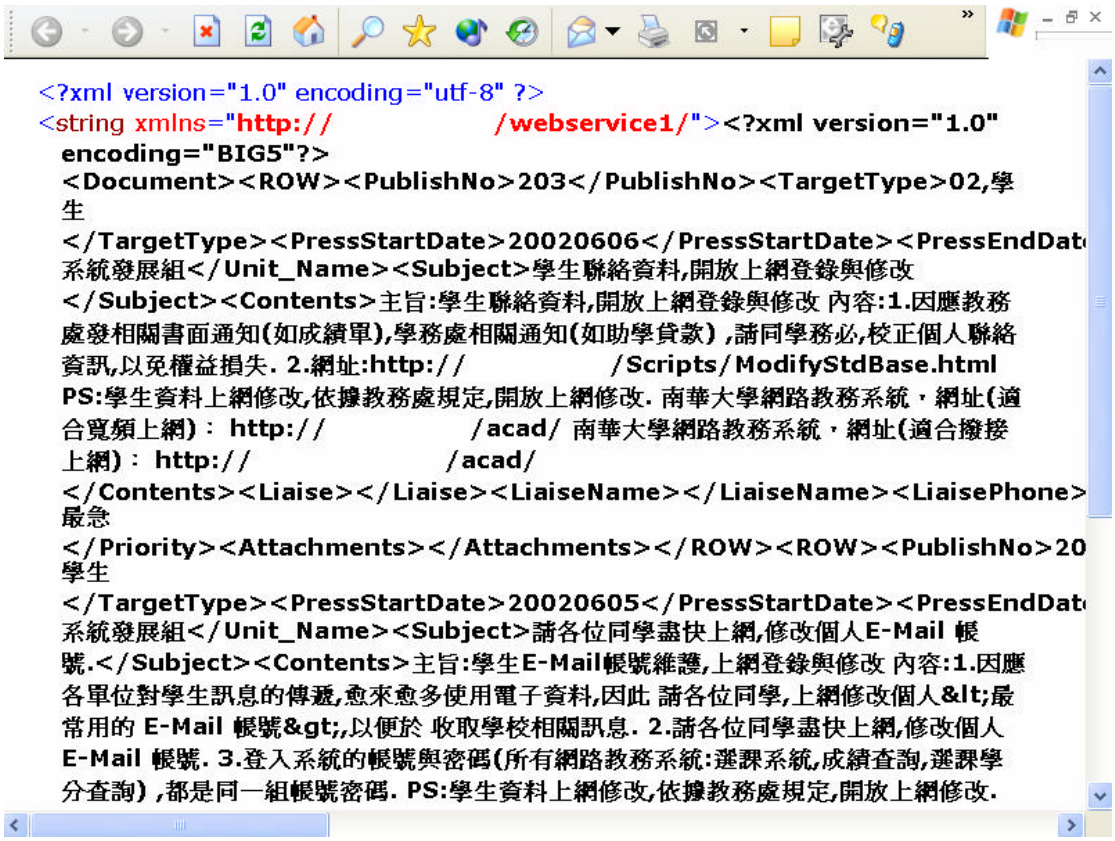

圖 32. 轉換後的 XML 檔案格式的內容

貳、建置管道層

元物件模組 COM 是 Windows 作業系統平台的核心技術，因此針對在 Windows 作業系統平台上，多數的軟體廠商所開發的工具，在於 MiddleWare 的呈現多以 COM 的形式出現，而在 Windows 作業系統平台上，COM 以下列兩種型態出現[20]：

一、DLL(In-process)：大多是以服務的型態出現（如 Windows NT 上的 MTS 或是 Windows 2000 Server 各本版中的元件服務)。

二、EXE(out-process)：大多屬獨立功能性的型態，例如 Borland Delphi 以後的版本，多以.EXE 型態建立系統所需要的 MiddleWare。例如在本範例中舊有的資訊系統中，以 Borland Delphi 建立.EXE 型態的 COM 成為舊有系統的 MiddleWare，因此在管道層的建立

上，只要遵守著 Windows 作業系統平台上的 COM 架構建立的規範，即可以使用適當的工具實作出管道層。

因為管道層的主要工作在於連接舊有資訊系統的 MiddelWare，所以在 Windows 作業系統平台上，我們可以利用 COM 與 COM 之間的溝通方式特性來建立管道層，以下使用 Borland Delphi 的工具實作一個.DLL 型態的 COM，來連接到原有以 Borland Delphi 建立.EXE 型態的 MiddleWare，COM 與 COM 之間的溝通首先必須先瞭解所要呼叫 COM 的 GUID(Globally Unique IDentifer)，然後建立出所要呼叫的介面與方法「參見圖 33」，如此即可完成管道層建置。

```
function TAllCOMToAppServer.GetPublicNews(const Filter_i: WideString): Ole
var TmpMsg:String;
    TmpRtnData:WideString;
    iRoot:IUnknown;
    vObj:IAllWebRadService;
begin
 try
  iRoot:=CreateComObject(ProgIDToClassID('aWebRadService.AllWebRadServiceI
  vObj:=iRoot as IAllWebRadService;
   try
     TmpRtnData:=vObj.GetPublicNews(Filter_i);
   except
     on E: Exception do begin
       TmpMsg:=e.Message;
       TmpRtnData:='<p>DLL 錯誤回傳：呼叫RemoteDataModle(aCOMTOAppServer.e
     end;//on
   end;//if
 finally
   Result:=TmpRtnData;
 end;//try
end;
```

建立出所要呼叫的介面

所要呼叫COM的GUID

圖 33. COM 的呼叫與介面的實作

當管道層以.DLL 型態建立的 COM 出現在 Windows 作業系統平台上，以 Windows 2000 Server 此版本為例，管道層所呈現的型態「參見圖 34 所示」，透過管道層的建置連接到舊有系統的 MiddleWare「參見圖 35 所示」，即可完成本文所提出的改良架構。此外配合 Windows 作業系統平台中的元件服務，能將其功能再加強

延伸，有關於此點因為非本論文的所要探討的議題，因此不作深入的研究。

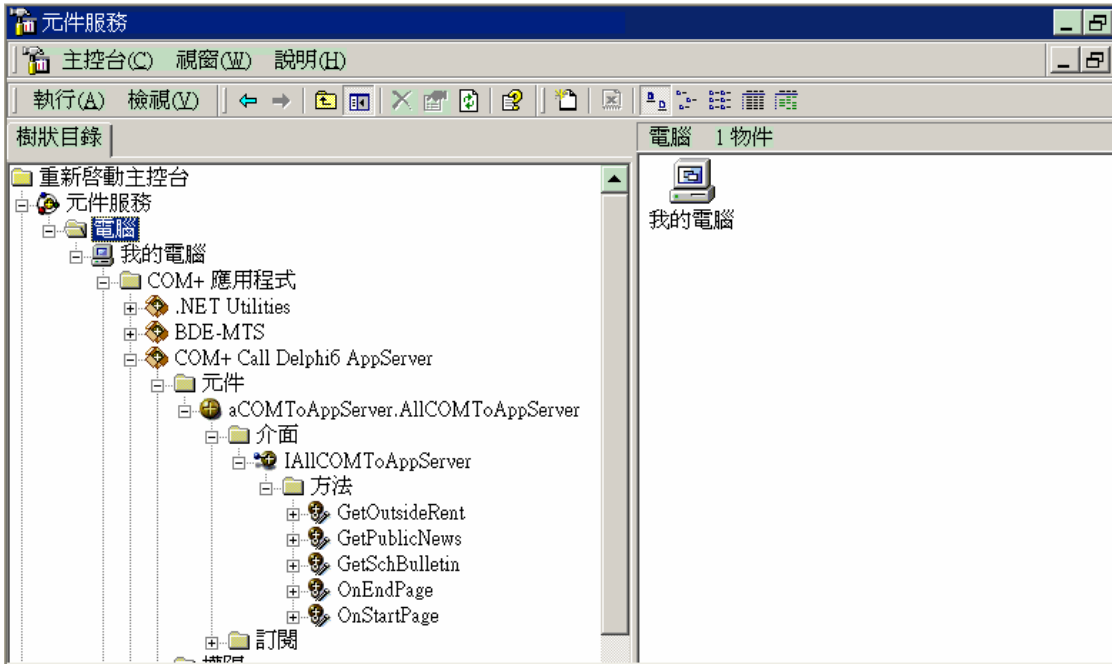

圖 34. 管道層在 Windows 2000 Server 的呈現方式

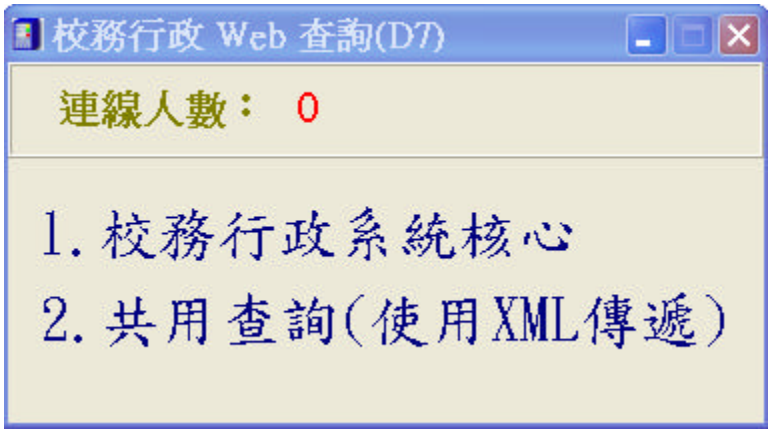

圖 35. 舊有資訊系統的 MiddleWare

以上是在 Windows 作業平台中所建立的管道層，而此種方式只是管道層的呈現方式之一，即使在 Windows 作業平台中，管道層的呈現不一定以此方式出現，只要符合本論文第三章中所提的管道層

觀念，可以使用不同的方式與工具來呈現管道層的架構。

參、透過管道層連接 MiddleWare 的 Web Services

Web Services 新科技的出現，使的過去許多原本難以達成的需求，如今可以很容易的達成，例如行通通訊的各項裝置連接到企業組織的資訊系統。當管道層建立後，Web Services 可以很容易的透過管道層連接到舊有的資訊系統，本文在 Web Services 的呈現，以 Microsoft Visaul Studio .NET（2003 Final Beta 版）來實作透過管道層連接到 MiddleWare 的 Web Services「參見圖 36 所示」，如此透過 Web Services 不只能解決異質平台間的整合問題，且對於過去需要專業程式設計人員才能開發的行動通訊整合，如今都將因為 Web Services 的提出更容易解決此問題。

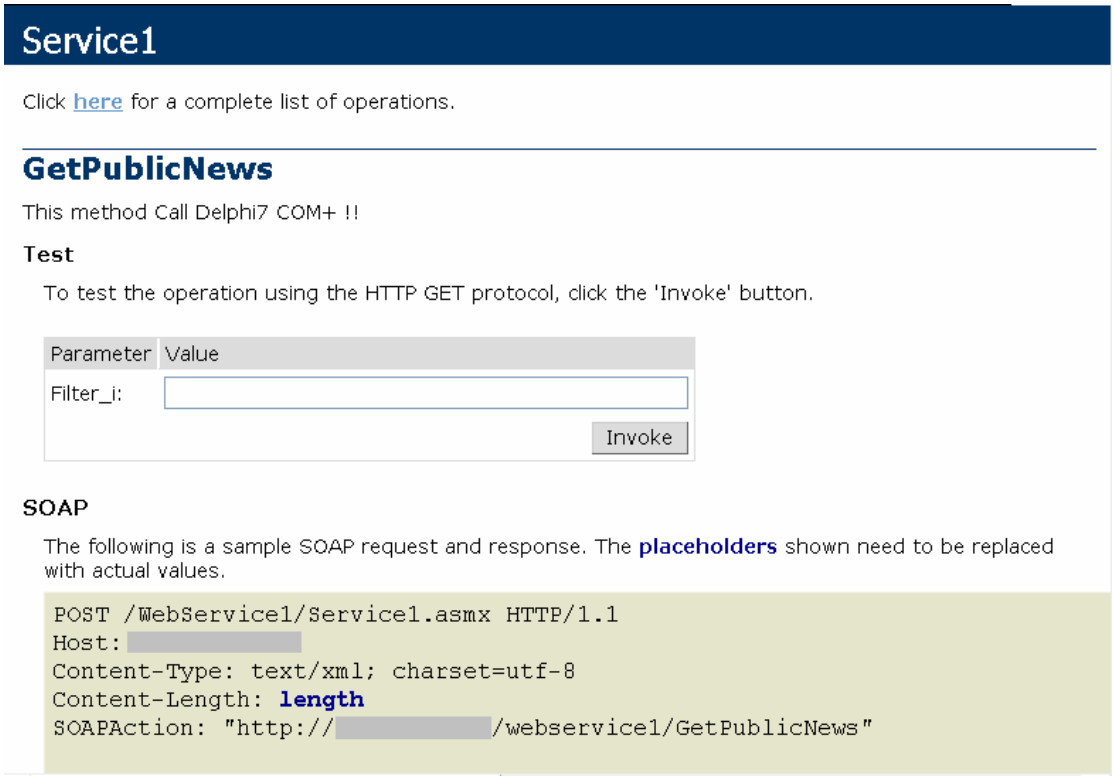

圖 36. 以 MS Viaul Studio.NET 所撰寫的 Web Services

透過管道層連接到 MiddleWare 的 Web Services，除了解決上述的問題之外，加上資料與格式分離的觀念，舊有的資訊系統可以透過管道層的架構，將資料的呈現在各種支援 XML 的平台上（如 PDA、Smartphone 等）。

第二節、「資料」與「格式」分離的實作

透過管道層連接到舊有的資訊系統，將原有的資料轉換成 XML 資料型態，另一個好處就是可以將資料與格式分離，利用此特性可以將舊有的系統，消除因為過去的技術或作業平台的限制，例如過去為了服務客戶所做的部分 e 化，所撰寫的網頁程式（如 Microsoft 的 ASP 程式），當面臨到需求變動時，所需要修改的 ASP 程式碼繁雜且不易維護，如今改變過去在 ASP 的程式撰寫方法，將資料與格式分離的觀念套入，以 XML 與各種不同的 XSL 格式結合，更能快速的因應客戶的變動需求。

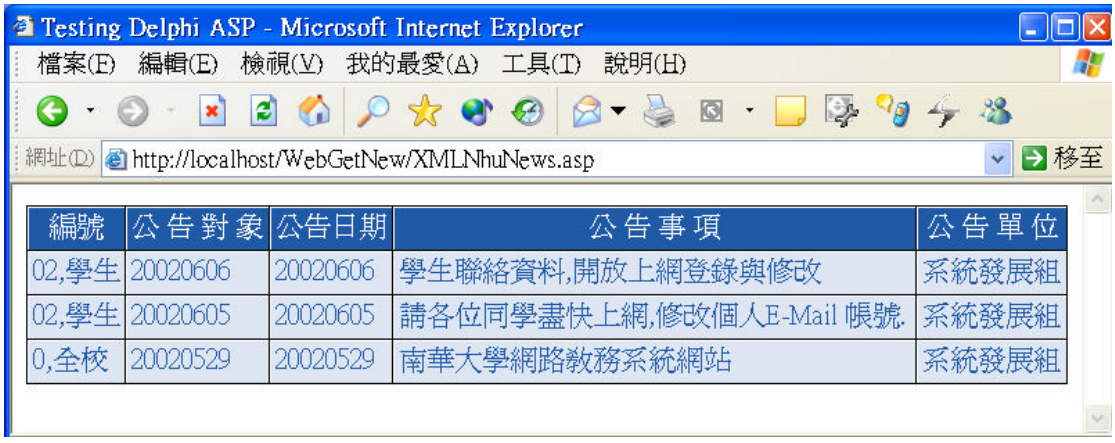

圖 37. 以「資料」與「格式」觀念實作的 MS ASP 程式

針對舊有的 ASP 程式設計的撰寫，套入資料與格式分離的觀念「參見圖 37」，即是以文件物件模組(Document Object Model, DOM) [53]的方式來作為呈現，在舊有的 ASP 程式中建立二個 DOM 來接收資料與格式，其作法如下：

第一個 DOM：透過管道層連接到舊有的資訊系統，將原有所要擷取的舊系統資料放入第一個 DOM 中。

第二個 DOM：將格式檔的資料（如以 XML 結構所寫成的.XSL 格式檔）放入第二個 DOM 中。

資料與格式分離的觀念，不只能應用在過去舊有的 ASP 程式，針對新一代的物件導向工具（例如 Microsoft ASP.NET），更能將此觀念配合新工具、新技術的應用，使的網頁程式的開發更快速、更容易。而對於資料與格式分離的延伸應用，透過 Web Services 可以將此觀念擴展到各式各樣支援 XML 資料格式的平台上。

第三節、Cross-Language 的驗證實作

舊有的資訊系統在面臨更新時，除了種種的考量因素外，對於開發、維護或技術限制與工具瓶頸，都是對於舊有的資訊系統所面臨到的最大內部問題，因此針對開發工具的限制，在管道層建置後，可以透過管道層的連接接收到舊系統所輸出的 XML 型態資料，因此只要開發工具語言支援 XML 資料格式型態，就可以當成開發的工具，或是使用其他技術快速開發各項使用者的需求。

在本小節的驗證中本文採用 Windows 作業平台上，三個國際知名的軟體廠商 Borland、Microsoft 與 Sysbase 所提供的開發工具，來說明 Cross-Language 的實作，並提出了下列六個實作範例，以 Borland Delphi 所開發的南華大學校務行政系統為驗證系統，來說明 Cross-Language 的開發環境驗證：

壹、使用 DOM 技術讀取 XML 資料的範例(ASP、ASP.NET)：

在 Microsoft 尚未推出.NET Framework[36]之前，在撰寫 ASP

網頁程式時，如果要將資料擷取到前端時，就必須透過 DOM 的技術來承接資料「參見圖 38 所示」，因此對於執行效率上的提昇，比過去使用 ASP 內建的 RecordSet Object 來的快速與方便。

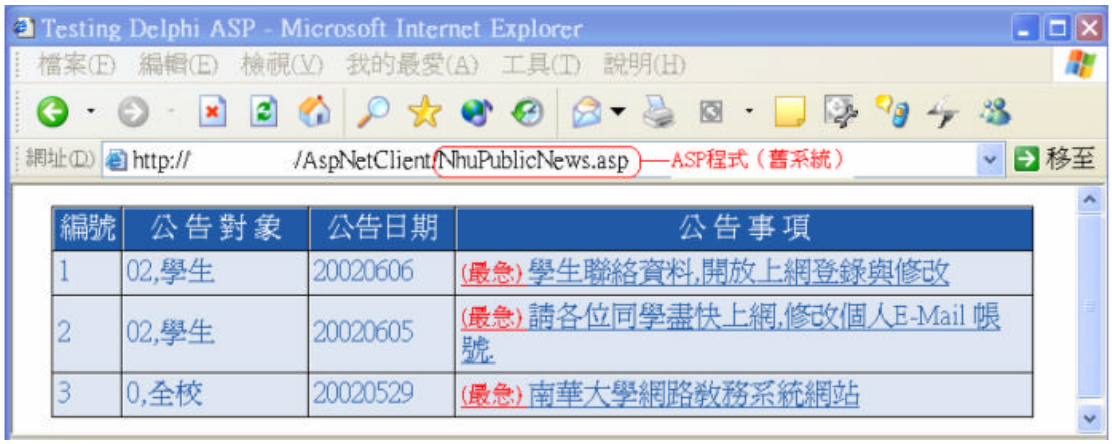

圖 38. 以 DOM 的方式接收 XML 資料的 ASP 程式

而對於新一代的.NET Framework 架構下的開發工具, 將可以更容易解決過去 ASP 開發工具中 RecordSet Object 的限制，在資料處理的部分依然可以使用 DOM 的技術接收 XML 資料「參見圖 39」並作相關處理。還可以使用 ADO.NET 的技術接收 XML 資料，比舊有的 ASP 更具擴展性。

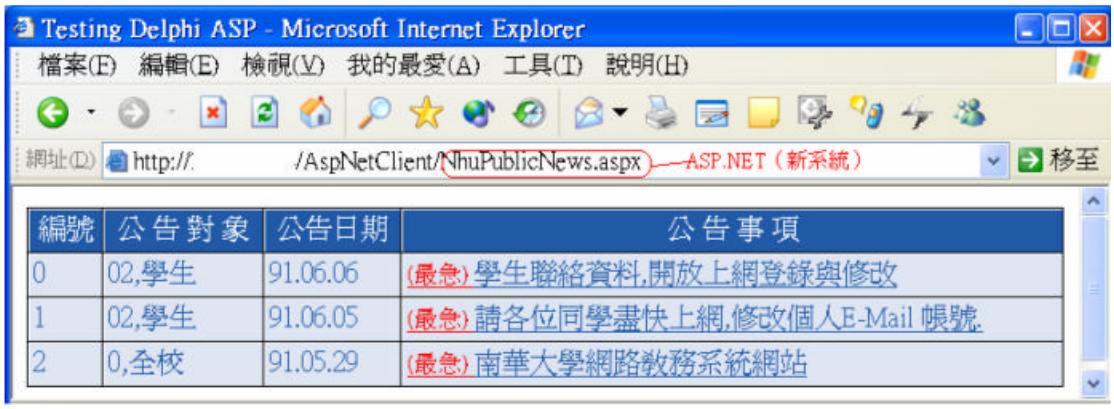

圖 39. 以 DOM 的方式接收 XML 資料的 ASP.NET 程式

貳、使用 ADO.NET 讀取 XML 資料的 VB.NET 範例 (VB.NET、ADO.NET)

在 Microsoft 推出 Visual Studio.NET 之前，有許多組織採用 Visual Basic 6（或以前的版本）當為資訊系統的開發工具，而其中使用 ADO(ActiveX Data Object)當成資料存取物件，是開發系統經常被用到的一種技術，而在.NET Framework 推出後，ADO.NET 的改進比舊有的 ADO 更具彈性與效益，而 ADO.NET 本身也支援 XML 資料格式，因此配合 VB.NET 也可以當成一種開發的技術。

ADO.NET 與舊有的 ADO 物件最大的不同，除了 ADO.NET 因為建置在.NET Framework 上，採用物件導向程式設計的方式外，ADO.NET 可以當成資料的容器，將許多資料以「離線」的方式存放在 ADO.NET 中，而針對舊有的 ADO 物件則無法做到此點，因此在此範例中特別提出作為參考。

在此範例中先將管道層的 COM 建立，透過管道層將 XML 資料輸出，在 VB.NET 中以 ADO.NET 接收 XML 資料，並以 VB.NET 的 DataGrid 將資料呈現在畫面上（參見圖 40 所示）。

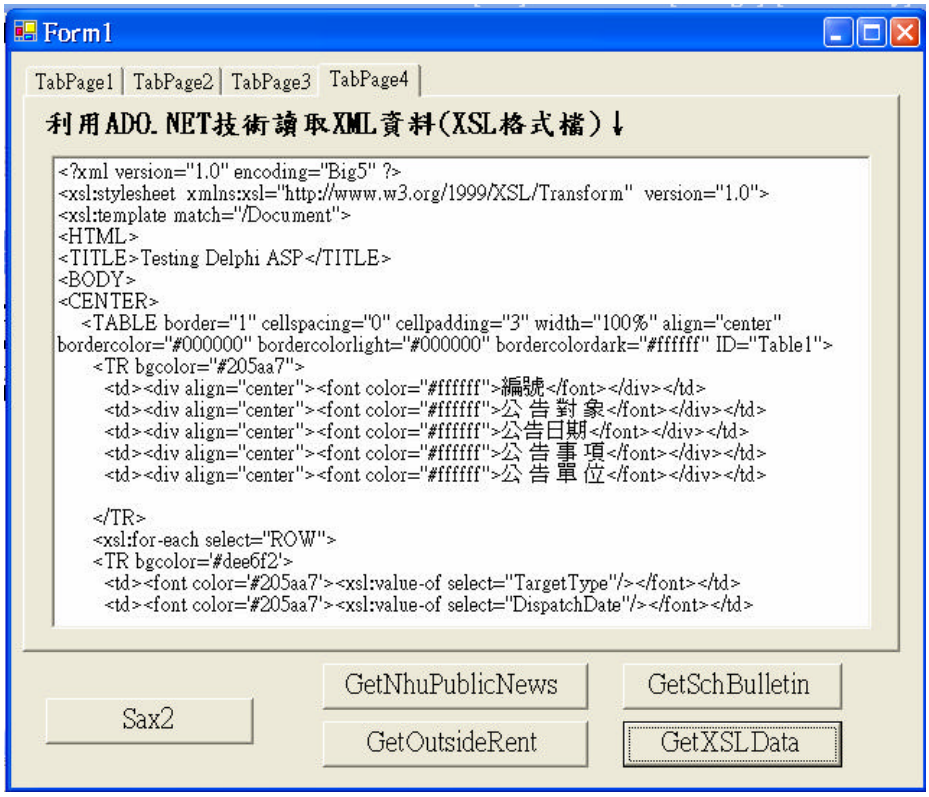

圖 40. 以 ADO.NET 的方式接收 XML 資料的 VB.NET 程式

參、使用 DOM 讀取 XML 資料的 VB.NET 範例。(VB.NET、DOM)

DOM **本身是 W3C 組織中規範的 XML 相關技術，因此用來接收 XML 型態的資料是一種標準的方法, 只要開發工具的軟體廠商支援 XML 的資料讀取，及 DOM 技術的相關規範，即可使用 DOM 來接收 XML 資料「參見圖 41」，但 DOM 的使用有其限制，由於 XML 必須完全的被讀到 DOM 中才能對資料作相關存取，因此針對較大的 XML 檔案資料，會加深軟硬體系統的負載並降低效能，因此在使用 DOM 接收 XML 資料時必須審慎考量。**

雖然 VB.NET 所採用的架構是 Microsoft 新一代.NET Framwork 架構，因此可以使用 ADO.NET 的技術來作為 XML 資料的接收，但如果是舊系統的 VB6 以前的版本, 在有關 XML 資料的接收與處理，此時 DOM 的技術即有很大的效用，這也是本文架構提出的用意，

在系統的驗證上提出此範例驗證的原因。

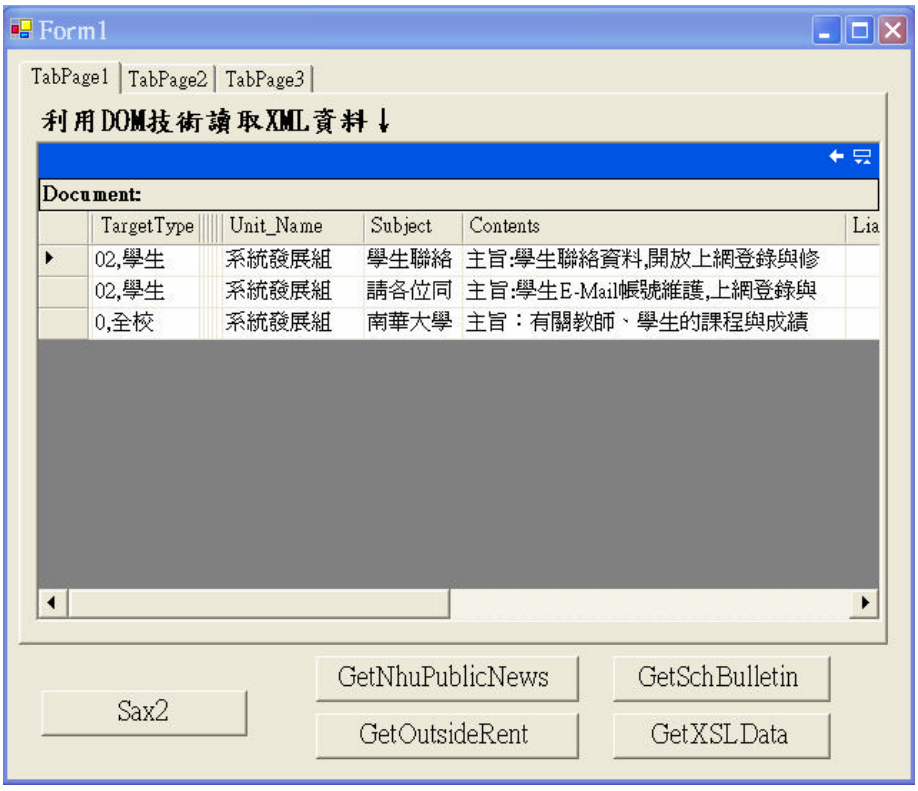

圖 41. 以 DOM 的方式接收 XML 資料的 VB.NET 程式

肆、使用 SAX 讀取 XML 資料的 VB.NET 範例。(VB.NET、SAX)

在上一點中提到 DOM 本身的限制，針對此點可以使用 SAX(Simple API for XML) [16]來解決 DOM 的限制，由於 SAX 本身採用 Event 的特性，因此對於 XML 資料的讀取可以片段的讀入，即使 XML 本身的資料結構有錯誤，也可以讀取到相關的資料，再者不同於 DOM 必須全部的 XML 資料讀完後，才能對 XML 處理的方式讓 SAX 的彈性更大，當 XML 資料檔案太大時，則可以採用 SAX 的方式來讀取 XML 資料。「參見圖 42」即是 VB.NET 採用 SAX 的方式，透過管道層讀取舊系統上 Middleware 輸出的 XML 資料。

SAX 的出現與 DOM 將會有互補的作用，在舊系統套上管道層的架構後，由於透過管道層所接收的資料都是 XML 的資料格式，因

此針對 XML 資料的接收與處理的相關技術瞭解，是本文架構延伸的另一個議題，必須特別注意。

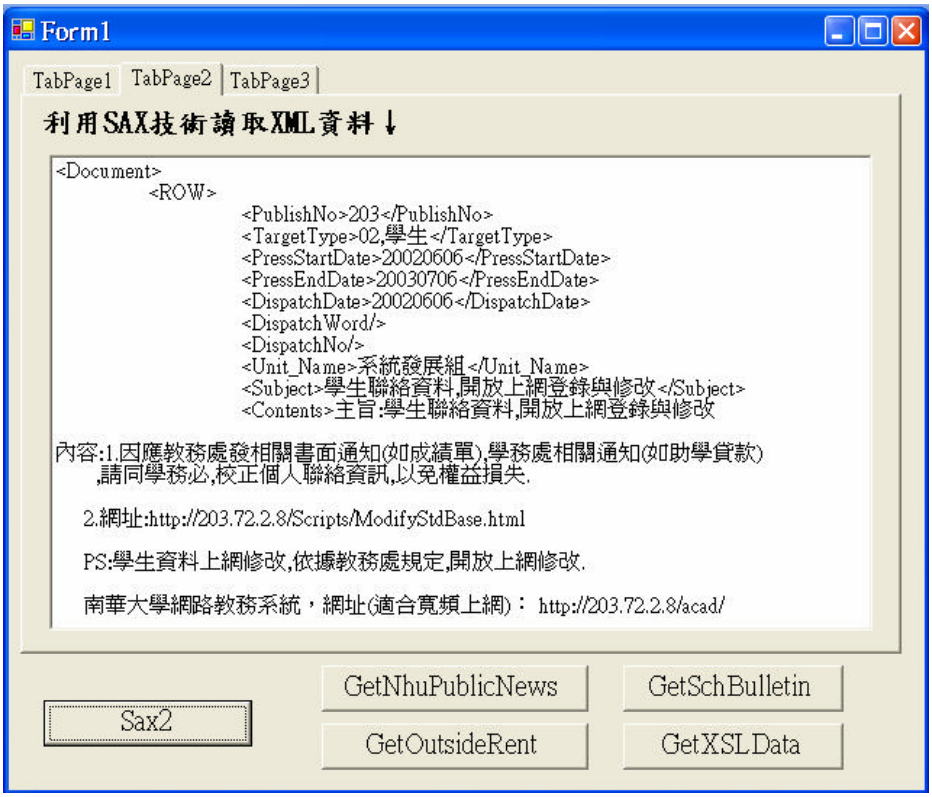

圖 42. 以 SAX 的方式接收 XML 資料的 VB.NET 程式

伍、使用 Borland Delphi7 讀取 XML 資料的範例。(Borland Delphi)

由於本範例也是由 Borland Delphi 所開發的，因此對於相同開發工具將更容易運用。在 Delphi 工具中一樣支援標準的 XML 資料讀取，為了讓此範例更具說服力，此範例並不使用原有的 Delphi Provider 輸出資料的機制，而是如同 Microsoft 的相關開發工具一樣接收 XML 資料。

在 Borland Delphi 中以 XMLTransformProvider VCL 元件，讀取由舊有的資訊系統所輸出的 XML 資料，透過 Delphi 中的 ClientDataSet VCL 元件將 XML 資料呈現出來「參見圖 43」。

雖然本文採用的驗證系統也是由 Borland Delphi 所開發，但

Delphi 的開發工具依然支援 XML 格式的讀取，及 DOM 相關的技術的處理，因此即使本文採用的驗證系統不是 Delphi 所開發的產品，但只要套上本文所提出的架構，與遵照 XML 的格式轉換型態，依然可以使用此工具為開發工具。

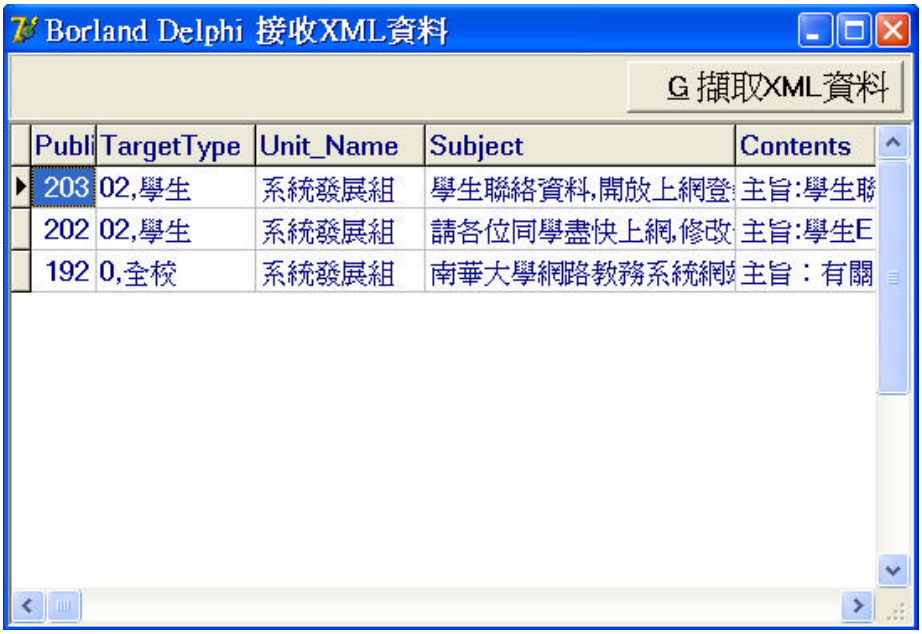

圖 43. 以 Borland Delphi7 程式呈現 XML 資料

陸、使用 Sybase PowerBuilder9 讀取 XML 資料的範例。(Sybase PB)

本範例使用 Sybase PowerBuilder9 程式開發，以 DataWindow 接收管道層傳出的 XML 資料並呈現在 DataWindow 中「參見圖 44」，DataWindow 是使用 Sybase PowerBuilder 為開發工具經常使用的物件。

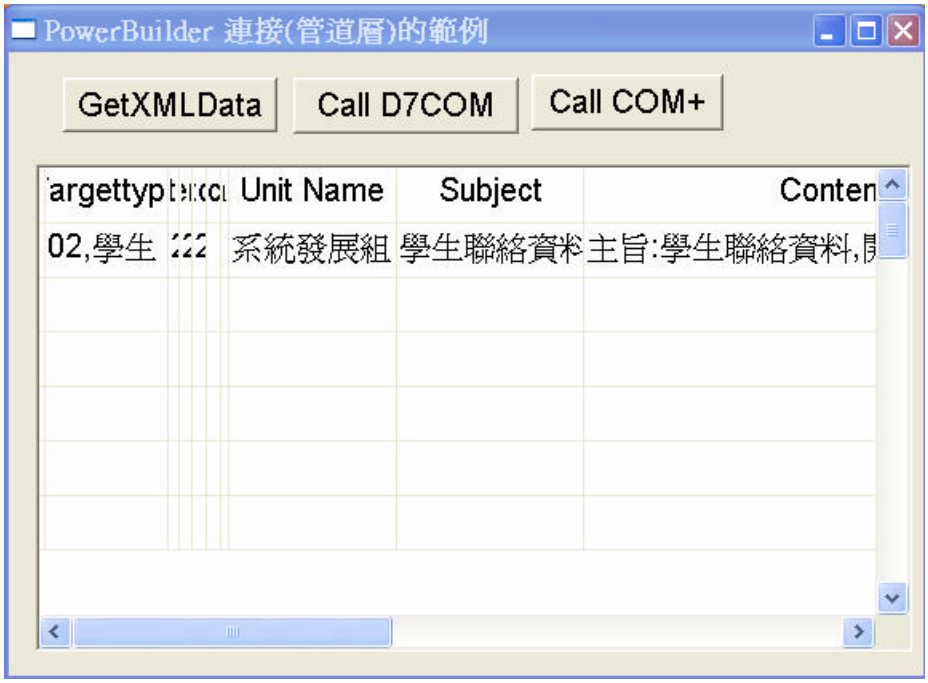

圖 44. 以 Sybase PowerBuilder9 程式呈現 XML 資料

由上述六個範例的實作可以說明在 Windows 作業平台上的三個常見的開發工具，Visual Basic、Delphi 與 PowerBuilder 都能使用 XML 的資料交換格式，並且可以透過管道層的連接，將舊有資訊系統的資料以新的開發工具針對系統「整合」，不再受限於開發工具。

第四節、結合 Web Services 的行動通訊實作

壹、PDA 行動通訊的實作範例

Web Services 的出現不止帶來企業間更容易的溝通與互動，由於 SOAP 通訊協定的出現，讓訊息的傳遞所遇到的限制大為降低，過去開發行動通訊需要高技術門檻，隨著 Web Services 及其他軟體廠商所開發出來的工具，將降低其技術門檻使開發行動通訊變得更容易。透過管道層連接到舊有的資訊系統，可以輕易的將資訊以 Web Services 的方式「參見圖 45」，讓行動通訊輕易的擷取舊系統上所提

供查詢的服務資料，要完成行動通訊的建置除了建置好管道層之外，以 PDA（Personal Digital Assistant，個人數位助理）行動通訊的開發為例，必須完成下列兩個步驟：

一、建置透過管道層連接舊資訊系統 MiddleWare 的 Web Services：

「參見圖 45」。

二、開發行動通訊（如 PDA）的應用程式：

行動通訊的開發目前已有許多工具，例如本範例所採用的 Microsoft Visual Studio.NET（2003 Final Beta 版），透過 Web Services 與 XML 相關技術可以很容易開發行動通訊。

當管道層建置好之後，要完成行動通訊應用程式撰寫之前，必須透過 Web Services 連接到管道層擷取所需要的資料「參見圖 45」，而 Web Services 在本文的範例中是以 ASP.NET 所開發。

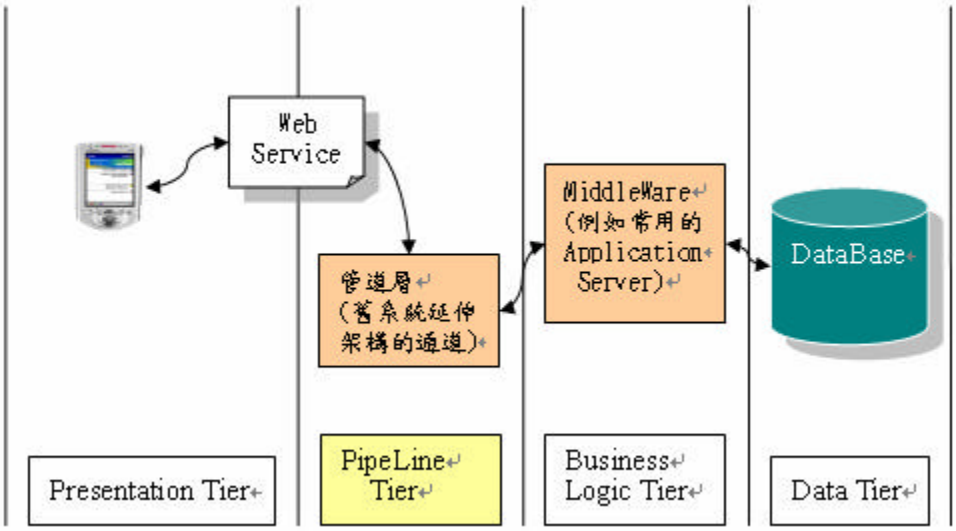

圖 45. 連接管道層的 Web Services

關於行動通訊的應用程式開發，其觀念如同上述的範例一般並無太大的差異點，所不同的是選擇的開發工具會有所限制，例如在本範例中所採用的工具 Microsoft Visual Studio.NET

（2003 Final Beta 版），在開發行動通訊的應用程式，一樣以 ADO.NET 為接收 XML 資料，也是透過管道層連接到舊有的資訊系統，不同的是中間多了一個 Web Services「參見圖 45」。

但有關於開發工具的限制，以 Microsoft Visual Studio.NET(2003 Final Beta 版)為例，由於建置在 Windows .NET Framework 的架構上，因此對於.NET Framework 本身的物件類別都能引用，不同的是行動通訊所使用的是.NET Compact Framework，可以說是一種精簡的.NET Framework，因此在實際的行動通訊應用程式撰寫上，會與一般 Windows 作業平台上所開發的應用程式語法有所差異，但大致上的觀念都相同，所

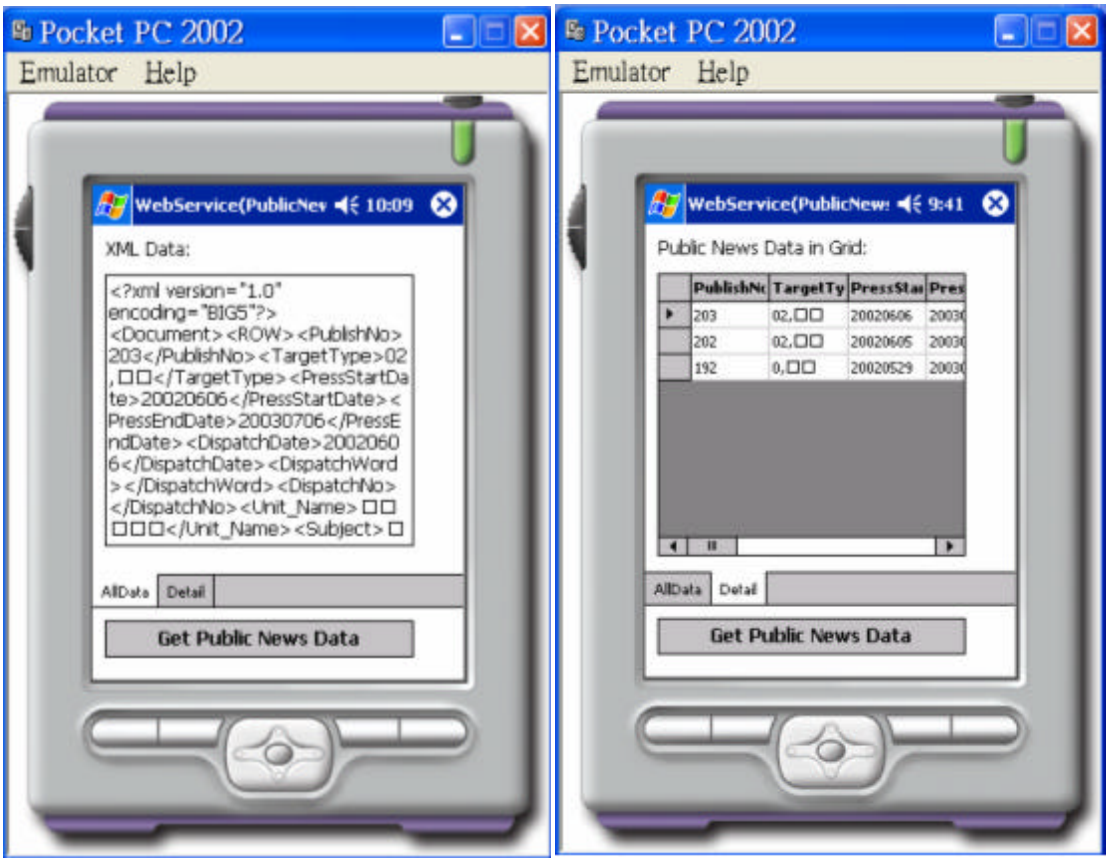

圖 46. 以 PDA 接收 XML 資料的範例

以經由 Web Services 透過管道層，一樣可以將資料呈現在行動通訊上「參見圖 46」。

貳、Web Services 的效率問題與解決方式

Web Services 的通訊採用 SOAP 通訊協定，而 SOAP 的通訊協定是建置在 HTTP 通訊協定之上，因為網際網路的興起與各項基礎建設的建置，提供 Web Services 一個良好的發展條件，因此 Web Services 才可以穿越企業防火牆、整合企業間的資料流，即使企業間的平台都不同，但也因為要達成「通用性」的前提，因此在執行效率上必然會降低，因為組織在建置 Web Services 時，多數建置在 MiddleWare 上（這觀念與 DataBase 的 Pooling 觀念是相同的），因此除了 Web Services 本身的限制外，MiddleWare 是 Web Services 執行效率考量的另一項重點，因此如何建置良好的 MiddleWare，且不需要花費太多成本是組織所必須思考的。

以本範例的驗證系統--南華大學校務行政系統，本身在 MiddleWare 上建置採用改良的模式[1]，因此在 MiddleWare 上有大幅度的效率改良，且此中間應用層之改良模式，已實際應用於南華大學校務行政系統上，不止在效率上改進且能降低硬體成本。

以選課系統為例，Middleware 所用的硬體設備為三台一般的 PC，而不是昂貴的 Server 級主機，因此即使不使用本文所提出的管道層架構，在建置 Web Services 時依然會面臨到 Middleware 的效率問題，因為 Middleware 是分散式架構資訊系統的中很重要的一環，所以改進 Middleware 的效率與降低成本，資訊系統的建置成本與效率必然也會降低成本與提高效率，至於如何改進 Middleware，（曾清義，民 90）在其論文研究中「中間應用層資料服務物件模型之改進研究」[1]有詳細的介紹。

第五節、Linux 平台的實作

本研究除了能達到 Cross-Language、行動通訊等特性外，在橫跨不一樣的作業系統平台上一樣能達到其需求。針對不一樣平台的實作，本範例選擇以 Redhat Linux7.2 為作業系統，以 Borland Kylix3.0 為開發工具，而使用的方法是以 Web Services 的方法作為呈現（本文也可以使用其他方法來實作，但此部分並非本文所要深入探討的，因此不做詳細說明)。

在 Redhat Linux7.2 的平台上所撰寫的應用程式，一樣透過管道層連接到舊有的資訊系統，接收 Middleware 轉出的 XML 資料型態，並以 Borland Kylix3 的工具所撰寫的程式, 將資料顯示在螢幕上「參見圖 47」。

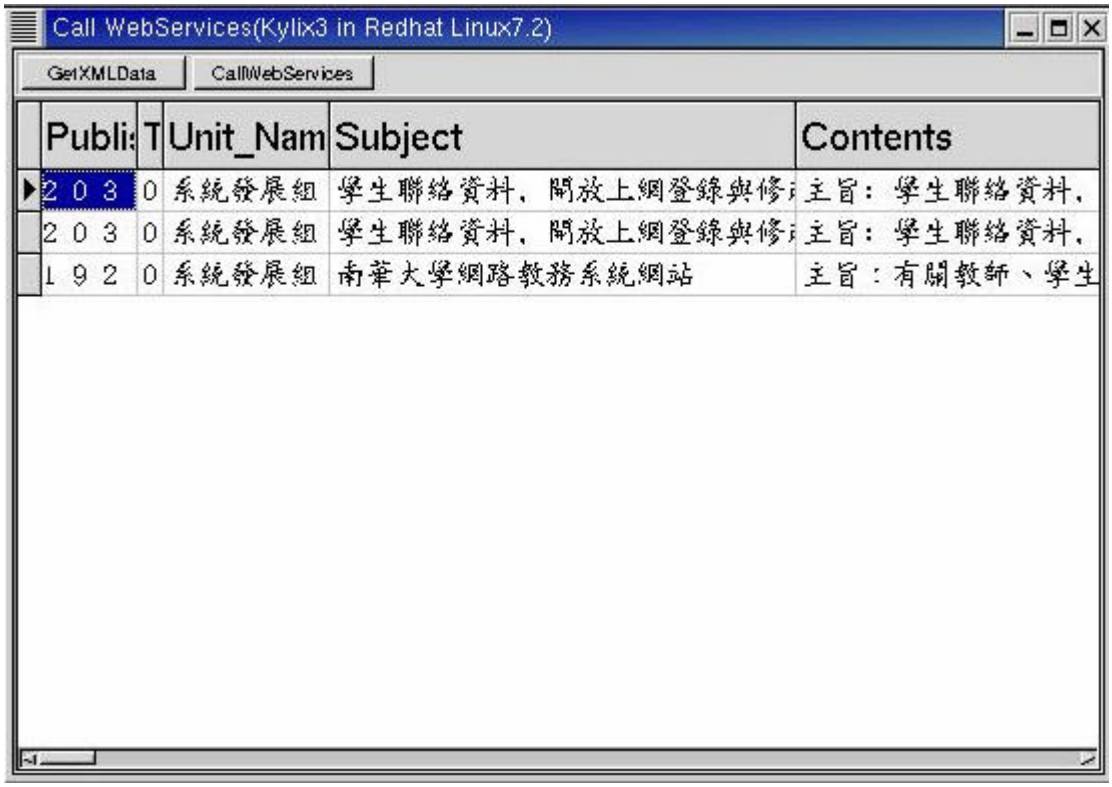

圖 47. 在 Linux 平台上接收「管道層」所傳出的 XML 資料範例

第五章 結論與建議

第一節、結論

資訊科技的進步與不斷的推陳出新，對於組織的影響在於提供更多解決方法的參考，而非只是一昧的追求新的科技與技術。許多組織當面臨舊有的系統更新與整合時，所考量的並非只有新科技或是成本，「穩定」與「成功」的舊系統更新，或是導入新系統更是組織重要的考量，畢竟不論組織是否為營利機構，都有其一定的成立宗旨，而在資訊科技進步的今日，當組織引入資訊系統後，必然會面臨資訊系統的老舊與不符合需求使用，而如何在不需購置新系統的前提下，為舊有的資訊系統注入新活力，正是本文提出的三個研究目的的意義所在。

本文在第三章所提出的「管道層」改量式架構，在第四章的驗證中以 Windows 作業系統平台為實驗平台，以南華大學校務行政系統為驗證的系統，針對本文所提出的三個研究目的及驗證結果如下：

壹、改進傳統的三層式架構，利用 XML 為資訊交換格式，建構出新的架構：

在改進舊有的三層式架構系統，在原有的 Middleware 中將輸出的資料轉換為標準的 XML 資料型態，並提出「管道層」改良式架構觀念，為舊有的資訊系統提供另一個更新與整合的節決方案。

貳、在更新或改進系統時，結合 Web Services 技術，讓資訊系統更具延展性：

在「管道層」建置後結合 Web Services 技術，可為舊有的資訊系統提供一個延伸架構的基礎，在本文第四章的驗證中實際以 PDA 行

動通訊的範例，說明透過 Web Services 技術連接「管道層」擷取舊有資訊系統的資料。

參、因應 XML 的特性，延伸出跨語言(Cross-Language)的快速開發環境：

XML 的標準由 W3C 組織所制訂，其相關的技術近年來已逐漸的被應用，而本文將 XML 當成資料的交換格式，運用各大軟體廠商的開發工具支援 XML 的潮流趨勢，延伸出 Cross-Language 的開發工具特性，並在第四章實際以不同語言的開發工具，與結合 XML 相關技術（如 DOM、SAX）驗證本文所提出的第三個研究目的。

本文所提出的「管道層」改良式架構，其精神在於此架構的概念而非技術上的突破或創新，因此針對本文作第四章所提出的驗證，只是「管道層」改良式架構的一種呈現方式，這也是本文日後延伸作研究的另一個議題。

第二節、未來研究的建議與方向

本文所提出的改良式架構與其相關的研究目的，其重點觀念與實際驗證都以 Windows 作業平台為主，針對在其他作業平台（如 Unix 平台）則較少介紹，因此在日後的研究重點，除了放在「管道層」改良式架構的不同呈現上，在其他作業平台上將本文架構的實際導入，並且作相關的驗證亦是本文未來的研究方向與重點之一。

參考文獻

中文部分

[1]曾清義，「中間應用層資料服務物件模型之改進研究」，南華大學資訊管理研究所碩士論文，90 年 6 月。

英文部分

[2] Anderew T. Campbell, Geoff Coulson, and Michael E. Kounavis, "Managing Complexity Middleware Explained", IEEE IT Professional, pp.21-28, September/October 1999.

[3] Boris Lubinsky, Michael Farrell Jr., "Top 10 Resons Why EAI Fails", eAI Journal, pp.41-42, December 2002.

[4] Bob Dunn, "A Manager's Guide to Web Services", eAI Journal , pp.14-17, January 2003.

[5] David S. Linthicum, "Moving to Service-Based Application Intergration", eAI Journal, pp.9-10, October 2001.

[6] Don Berman, "Web Services：Will Anything Really Change？", eAI Journal, pp.39-40, April 2002.

[7] Don Estes, "Disenfranchising Middleware", eAI Journal, pp.64-70, November/December 2000.

[8] Doron Sherman, "Web Services Orchestration", eAI Journal, pp.42-46, November 2002.

[9] Dino Esposito, Building Web Solution With ASP.NET and ADO.NET, United States of America, Microsoft Press, 2002.

[10] Frank P. Coyle, "Breathing Life into Legacy", IEEE IT Professional, pp.17-24, October 2001.

[11] Greg Grosh, "Coming Clean With SOAP", eAI Journal, pp.8-12, April 2002.

[12] Jaideep Roy, Anupama Ramanujan, "Understanding Web Services", IEEE IT Professional, pp.69-73, September/October 2001.

[13] Jean Bozman, Al Gillen, and Charles Kolodgy etc., "Windows 2000 Versus Linux in Enterprise Computing", IDC, September 2002.

[14] Katherine Hammer, "Web Services and Enterprise integration", eAI Journal, pp.12-15, November 2001.

[15] Leonid Erlikh, "Integrating Legacy Systems Using Web Services", eAI Journal, pp.12-17, August 2002.

[16] Larry Leonard, "Exploring SAX2", Windows Developer,Volume13, Number 3, pp.8-21, March 2002.

[17] Matjaz B. Juric, "EAI and Web Services", eAI Journal, pp.31-35, July 2002.

[18] Mike Rosen, Jim Boak, "Developing a Web Services Strategy", eAI Journal, pp.39-43， January 2002.

[19] Michael S. Pallos, "Service-Oriented Architecture：A Primer", eAI

Journal, pp.32-35, December 2001.

[20] Nasir Darwish, "COPS：cooperative problem solving Using DCOM", Journal of System and Software, Volume 63, Issue2, pp.79-90, August 2002.

[21] Paul Holland, "Building Web Services From Existing Application", eAI Journal, pp.45-47, September 2002.

[22] Rob Morris, Pete Isaksson, "Legacy Within the Enterprise：Imageine the Possibilities", eAI Journal, pp35-38, March 2002.

[23] Richard Stevens, "Writing Better Programs：The Story So Far", The Delphi Magazine, Issue 89, pp.28-32, January 2003.

[24] Robert Peacock, "Distributed Architecture Technologies", IEEE IT Professional, pp58-60, March 2002.

[25] Susan Chapin, Don Faatz, and Sushil Jajodia etc., "Consistent policy enforcement in distribute systems using mobile policies", Data & Knowledge Engineering, Volume 43, Issue3, pp.261-280, December 2002.

[26] Sanjay Pathak, "The Promise of Web Services", eAI Journal, pp.8-12, December 2001.

[27] Steve Vinoski, "Web Services Interaction Models", IEEE Internet Computing, pp.89-91, May/June 2002.

[28] Steve Vinoski, "Where is MiddleWare？", IEEE Internet Computing, pp83-85, March/April 2002.

［29］Steve Vinoski, “MiddleWare ‘Dark Matter’”, IEEE Internet Computing, pp.92-95, September/October 2002.

［30］Sam Wong, “Success With Web Services”, eAI Journal, pp.27-29, February 2002.

［31］Tamara Petroff, “Strategies for Legacy Migration Success”, eAI Journal, pp.32-36, June 2002.

［32］Tony M. Brown, “Web Services – EAI Killer ?”, eAI Journal, pp.48, April 2002.

［33］Tom Jepsen, “SOAP Cleans Up interoperability Problems on the Web”, IEEE IT Professional, pp.52-55, January/February 2001.

［34］Timothy M. Chester, “Cross-Platform Intergration with XML and SOAP”, IEEE IT Professional, pp.26-34, September/October 2001.

網站部分

［35］http://messageq.ebizq.net/news/news070899.html

［36］http://msdn.microsoft.com/netframework/

［37］http://www.microsoft.com/windowsserver2003/default.mspx

［38］http://www.w3c.org/DOM/

［39］http://www.w3.org/TR/wsdl12/

［40］http://www.w3.org/TR/SOAP/

［41］http://www.w3.org/XML/1999/XML-in-10-points.html

附件一

在附件中將本文所提出的「管道層」與第四章相關範例的程式原始碼，做詳細的解說，共分為下列六大部分：

壹、舊有資訊系統的 MiddleWare 將資料轉換成 XML 格式（Borland Delphi7）：

貳、以 Windows 作業平台上的元物件模組 COM 方式呈現，實作「管道層」原始程式碼（Borland Delphi7）：

參、「資料」與「格式」分離的實作：(Microsoft ASP 程式碼)

肆、Cross-Language 的六個實作驗證原始程式碼：(Microsoft ASP、ASP.NET、Visual Studio.NET, Borland Delphi7, Sysbase PowerBuilder9)

伍、透過「管道層」連接 MiddleWare 的 Web Services，以 PDA 行動通訊的實作範例原始程式碼：(Microsoft、Visual Studio.NET 2003 Beta)

陸、跨平台的範例實作（以 Redhat Linux7.2 為作業平台，Borland Kylix3.0 為開發工具）

壹、舊有資訊系統的 MiddleWare將資料轉換成 XML 格式, 使用 Borland Delphi7 所實作的 Application Server（MiddleWare）原始程式碼：

```
================ uSuAppWebRadService.pas (起始) ================
unit uSuAppWebRadService;

{$WARN SYMBOL_PLATFORM OFF}

interface

uses
  Windows, Messages, SysUtils, Classes, ComServ, ComObj, VCLCom,
DataBkr,
  DBClient, aWebRadService_TLB, StdVcl,uPoolmanager, xmldom,
XMLIntf,
  msxmldom, XMLDoc, Xmlxform, DB, ADODB, Provider,Variants;

type
  TAllWebRadServiceRDM = class(TRemoteDataModule,
IAllWebRadService)
    AdoRadPub: TADOConnection;
    vPublicNewsProvider: TDataSetProvider;
    vPublicNews: TADOQuery;
    vPublicNewsXMLTransform: TXMLTransformClient;
    XMLDocument: TXMLDocument;
    ADOStaffair: TADOConnection;
    vOutsideRentProvider: TDataSetProvider;
    vOutsideRent: TADOQuery;
    vOutsideRentXMLTransform: TXMLTransformClient;
    vSchBulletinProvider: TDataSetProvider;
    vSchBulletin: TADOQuery;
    vSchBulletinXMLTransform: TXMLTransformClient;
    vDataField: TADOQuery;
  private
    { Private declarations }
    function GetFileStream(Filename: string;Dataset: TDataset):
olevariant;
    function FindFile(FileName: String; DataSet: TDataSet):
Boolean;
  protected
    class procedure UpdateRegistry(Register: Boolean; const ClassID,
ProgID: string); override;
    function GetPublicNews(const Filter_i: WideString): OleVariant;
safecall;
    function GetOutsideRent(const Filter_i: WideString):
OleVariant; safecall;
    function GetSchBulletin(const Filter_i: WideString):
```

```
OleVariant; safecall;
    function GetXSLData(const FileName_i: WideString): OleVariant;
safecall;
  public
    { Public declarations }
  end;

  TAllWebRadServiceINTF = class(TIntfObject, IAllWebRadService )
  protected
    function LockRDM: IUnknown;override;  //
    procedure UnlockRDM(Value: IUnknown);override;
    procedure ClientChange( ADD:Boolean );override;
    function GetPublicNews(const Filter_i: WideString): OleVariant;
safecall;
    function GetXSLData(const FileName_i: WideString): OleVariant;
safecall;
  public
    { Public declarations }
//    Destructor Destroy; Override;
//    procedure  Initialize;Override;
  end;


var
  AllWebRadServiceFactory: TComponentFactory;
  AllWebRadServiceManager:TPoolManager;

implementation
uses RadSystemAPI;
{$R *.DFM}

class procedure TAllWebRadServiceRDM.UpdateRegistry(Register:
Boolean; const ClassID, ProgID: string);
begin
  if Register then
  begin
    inherited UpdateRegistry(Register, ClassID, ProgID);
    EnableSocketTransport(ClassID);
    EnableWebTransport(ClassID);
  end else
  begin
    DisableSocketTransport(ClassID);
    DisableWebTransport(ClassID);
    inherited UpdateRegistry(Register, ClassID, ProgID);
  end;
end;

//============================================================
//       以<xml 格式>傳回 vPublicNews 全校公佈欄資料(RDM)
//============================================================
function TAllWebRadServiceRDM.GetPublicNews(const Filter_i:
WideString): OleVariant;
var TmpXML,TmpXMLResult:wideString;
```

```
begin
 try
   with vPublicNews do begin
     Close;
     Open;
     Filtered:=False;
     Filter:=Filter_i;
     Filtered:=True;
   end;//with

   TmpXML:=vPublicNewsXMLTransform.GetDataAsXml('');
   XMLDocument.LoadFromXML(TmpXML);
   XMLDocument.Encoding:='BIG5';
   TmpXMLResult:=XMLDocument.XML.Text;
 finally
   Result:=TmpXMLResult;
   vPublicNews.Filtered:=False;
   vPublicNews.Filter:='';
 end;//try
end;


//============================================================
//             共用 Function <<GetFileStream>>(RDM)
//============================================================
//
function TAllWebRadServiceRDM.GetFileStream(Filename:
string;Dataset: TDataset): olevariant;
var s:TStream;
begin
  Result:=unAssigned;
  with DataSet do begin
    if FindFile(Filename,DataSet) then  begin
      S:=CreateBlobStream(FieldByName('AData'),bmRead);
      try
        StreamToVariant(S,Variant(Result));
      finally
        S.Free;
      end;//try
    end;//if
  end;//with
end;

//============================================================
//             共用 Function <<FindFile>>(RDM)
//============================================================
//
function TAllWebRadServiceRDM.FindFile(FileName: String;DataSet:
TDataSet): Boolean;
begin
  with DataSet do begin
    Filtered:=False;
    if Not Active then Active:=True;
```

```
    Result:=Locate('FileName',Filename,[loCaseInsensitive]);
  end;//with
end;


//============================================================
//    以<String 格式>傳回 XSL 檔案，資料庫中的檔案(RDM)
//============================================================
//
function TAllWebRadServiceRDM.GetXSLData(const FileName_i:
WideString): OleVariant;
var v:variant;
    vStream:TStream;
    tsXSLData:TStringList;
begin
 try
  v:=GetFileStream(FileName_i,vDataField);
  vStream:=TMemoryStream.Create;
  tsXSLData:=TStringList.Create;
  varianttostream(v,vStream);
  vStream.Seek(0,soFromBeginning );
  tsXSLData.LoadFromStream(vStream);
 finally
  Result:=tsXSLData.Text;
  vStream.Free;
  tsXSLData.Free;
 end;//try
end;


{ TAllWebRadServiceINTF }

procedure TAllWebRadServiceINTF.ClientChange(ADD: Boolean);
begin
  AllWebRadServiceManager.ClientChange(ADD);
end;

function TAllWebRadServiceINTF.LockRDM: IUnknown;
begin
  Result:=AllWebRadServiceManager.LockRDM;
end;

procedure TAllWebRadServiceINTF.UnlockRDM(Value: IInterface);
begin
  AllWebRadServiceManager.UnLockRDM(Value);
end;

//============================================================
//        以<xml 格式>傳回 vPublicNews 全校公佈欄資料(INTF)
//============================================================
//
function TAllWebRadServiceINTF.GetPublicNews(const Filter_i:
WideString): OleVariant;
```

```
var
  RDM: IAllWebRadService;
begin
  RDM := LockRDM as IAllWebRadService;
  try
    ReSult:=RDM.GetPublicNews(Filter_i);
  finally
    UnlockRDM(RDM as IUnknown);
  end;//try
end;

//============================================================
//   以<String 格式>傳回 XSL 檔案，資料庫中的檔案(INTF)
//============================================================
//
function TAllWebRadServiceINTF.GetXSLData(const FileName_i:
WideString): OleVariant;
var
  RDM: IAllWebRadService;
begin
  RDM := LockRDM as IAllWebRadService;
  try
    ReSult:=RDM.GetXSLData(FileName_i);
  finally
    UnlockRDM(RDM as IUnknown);
  end;//try
end;


initialization
  TComponentFactory.Create(ComServer,
     TAllWebRadServiceRDM,Class_AllWebRadServiceRDM,
     ciMultiInstance, tmApartment);

end.
```

================ uSuAppWebRadService.pas（結束）================

貳、以 Windows 作業平台上的元物件模組 COM 方式呈現，使用 Borland Delphi7 所實作的「管道層」原始程式碼：

```
================ uSuCOMToAppServer.pas（起始）================
unit uSuCOMToAppServer;

{$WARN SYMBOL_PLATFORM OFF}

interface

uses
```

```
  ComObj, ActiveX, AspTlb, aCOMToAppServer_TLB, StdVcl,
  aWebRadService_TLB,SysUtils;

type
  TAllCOMToAppServer = class(TASPObject, IAllCOMToAppServer)
  protected
    procedure OnEndPage; safecall;
    procedure OnStartPage(const AScriptingContext: IUnknown);
safecall;
    function GetPublicNews(const Filter_i: WideString): OleVariant;
safecall;
    function GetOutsideRent(const Filter_i: WideString): OleVariant;
safecall;
    function GetSchBulletin(const Filter_i: WideString): OleVariant;
safecall;
    function GetXSLData(const FileName_i: WideString): OleVariant;
safecall;
  end;

implementation

uses ComServ;

procedure TAllCOMToAppServer.OnEndPage;
begin
  inherited OnEndPage;
end;

procedure TAllCOMToAppServer.OnStartPage(const AScriptingContext:
IUnknown);
begin
  inherited OnStartPage(AScriptingContext);
end;

//============================================================/
//      以<xml 格式>傳回 vPublicNews 教務--全校公佈欄資料
//============================================================/
/
function TAllCOMToAppServer.GetPublicNews(const Filter_i:
WideString): OleVariant;
var TmpMsg:String;
    TmpRtnData:WideString;
    iRoot:IUnknown;
    vObj:IAllWebRadService;
begin
 try

iRoot:=CreateComObject(ProgIDToClassID('aWebRadService.AllWebRadS
erviceINTF'));
  vObj:=iRoot as IAllWebRadService;
   try
     TmpRtnData:=vObj.GetPublicNews(Filter_i);
   except
     on E: Exception do begin
       TmpMsg:=e.Message;
       TmpRtnData:='<p>DLL 錯誤回傳：呼叫
RemoteDataModle(aCOMTOAppServer.exe)-->ERROR('+TmpMsg+')</p>';
     end;//on
```

```
    end;//if
  finally
    Result:=TmpRtnData;
  end;//try
end;

initialization
  TAutoObjectFactory.Create(ComServer, TAllCOMToAppServer,
Class_AllCOMToAppServer,
    ciMultiInstance, tmApartment);
end.
```

================ uSuCOMToAppServer.pas（結束）================

參、「資料」與「格式」分離的實作

================ XMLNhuNews.asp（起始）================

```
<%
   Dim Obj

   '呼叫<Windows 2000 COM+ Service>中的 COM Object
   Set Obj = CreateObject("aCOMToAppServer.AllCOMToAppServer")

   '呼叫<Borland Delphi6 AppServer>中的 GetOutsideRent Method(),並
   '傳回 XML 形態的 DataSet.
   TmpXMLStr = Obj.GetPublicNews("")
   TmpXSLStr = Obj.GetXSLData("NhuPublicNews.xsl")

   Dim xmlDoc,xslDoc,StrOut

   Set xmlDoc = CreateObject("MSXML2.DOMDocument")
   xmlDoc.async = False
   xmlDoc.validateOnParse=False
   xmlDoc.loadXML(TmpXMLStr)

   Set xslDoc = CreateObject("MSXML2.DOMDocument")
   xslDoc.loadXML(TmpXSLStr)

   StrOut = xmlDoc.transformNode(xslDoc)

   Response.Write(StrOut)
```

```
%>
```

================== XMLNhuNews.asp（結束）================

肆、Cross-Language 的六個實作驗證原始程式碼：

一、使用 DOM 技術讀取 XML 資料的範例(ASP、ASP.NET)：

================== NhuPublicNews.asp（起始）================

```
<HTML>
<BODY>
<TITLE> Testing Delphi ASP </TITLE>
<CENTER>
<table border="1" cellspacing="0" cellpadding="3" width="95%"
align="center" bordercolor="#000000" bordercolorlight="#000000"
bordercolordark="#FFFFFF" ID="Table1">
   <tr bgcolor="#205aa7">
     <td width="7%">
       <div align="center"><font color="#FFFFFF">編號
</font></div>
     </td>
     <td width="19%">
       <div align="center"><font color="#FFFFFF">公 告 對 象
</font></div>
     </td>
     <td width="15%">
       <div align="center"><font color="#FFFFFF">公告日期
</font></div>
     </td>
     <td width="59%" bgcolor="#205aa7">
       <div align="center"><font color="#FFFFFF">公 告 事 項
</font></div>
     </td>
   </tr>
 <%
    Set DelphiASPObj =
    Server.CreateObject("aCOMToAppServer.AllCOMToAppServer")

   '呼叫 aStudentGrade.exe 中的  GetPublicNews<Method>
    TmpID = Request("id")
    If TmpID="" Then
       TmpFilter=""
    Else
       TmpFilter="TargetType like '" & TmpID & "%'"
    End If
    TmpXMLStr=DelphiASPObj.GetPublicNews(TmpFilter)

   '將呼叫<Method>的資料放入 XMLDocument Object
    Set xmlDoc = CreateObject("MSXML2.DOMDocument")
    xmlDoc.async = False
    xmlDoc.validateOnParse=False
    xmlDoc.loadXML(TmpXMLStr)
```

```
  '將 XMLDocument Object 中的資料,以 HTML 的方式展現(1/3)
   Set XMLRoot = xmlDoc.documentElement
   iRecCount = 0
   For Each currNode In XMLRoot.childNodes

  '將 XMLDocument Object 中的資料,以 HTML 的方式展現(2/3)
   iRecCount = iRecCount + 1
   For i = 0 To (currNode.ChildNodes.length - 1)
       TmpNodeName = currNode.ChildNodes(i).NodeName
       If TmpNodeName="PublishNo" Then TmpPublishNo =
            currNode.ChildNodes(i).Text
       If TmpNodeName="TargetType" Then TmpTargetType =
            currNode.ChildNodes(i).Text
       If TmpNodeName="PressStartDate" Then TmpPressStartDate =
            currNode.ChildNodes(i).Text
       If TmpNodeName="Subject" Then TmpSubject =
            currNode.ChildNodes(i).Text
       If TmpNodeName="Priority" Then TmpPriority =
            currNode.ChildNodes(i).Text
     Next
     If TmpPriority="最急" Then TmpSubject = "<font size='2'
            color='#FF0000'>(最急) </font>" & TmpSubject

    '將 XMLDocument Object 中的資料,以 HTML 的方式展現(3/3)
     Response.Write "<TR bgcolor='#dee6f2'>"
     Response.Write "<td width='7%'><font color='#205aa7'>" &
          iRecCount & "</font></td>"
     Response.Write "<td width='19%'><font color='#205aa7'>" &
          TmpTargetType & "</font></td>"
     Response.Write "<td width='15%'><font color='#205aa7'>" &
          TmpPressStartDate & "</font></td>"
     Response.Write "<td width='59%'><a href=ShowNews.asp?id='" &
          TmpPublishNo & "'><font color='#205aa7'>" & TmpSubject
          & "</font></a></td>"
     Response.Write "</TR>"
   Next

   Set XMLRoot = nothing
   Set xmlDoc = nothing
   Set DelphiASPObj = nothing
%>
</CENTER>
</BODY>
</HTML>
```

================= NhuPublicNews.asp（結束）================

================= NhuPublicNews.aspx（起始）================

```
<%@ Import Namespace="System.IO" %>
<%@ Import Namespace="System.Data"%>
<HTML>
  <TITLE>Testing Delphi ASP</TITLE>
  <script language="VB" runat="server">
```

```
  Function GetOutsideRent_Data(Filter_i As String) As String
      Dim Obj
      Dim TmpXMLStr As String

      '呼叫<Windows 2000 COM+ Service>中的 COM Object
      Obj = CreateObject("aCOMToAppServer.AllCOMToAppServer")

      '呼叫<Borland Delphi6 AppServer>中的 GetOutsideRent
      'Method(),並傳回 XML 形態的 DataSet.
      TmpXMLStr = Obj.GetPublicNews(Filter_i)
      Return TmpXMLStr
  End Function

  Function ToCHTDate(ByVal Date_i As String) As String
      Dim TmpDate, TmpYear, TmpMonth, TmpDay As String
      Dim iYear, iMonth, iDay As Integer

      TmpDate = ""
      If Trim(Date_i) = "" Then
          Return TmpDate
      End If
      iYear = CInt(Date_i.Substring(0, 4)) - 1911
      TmpYear = CStr(iYear)
      TmpMonth = Date_i.Substring(4, 2)
      TmpDay = Date_i.Substring(6, 2)
      TmpDate = TmpYear + "." + TmpMonth + "." + TmpDay

      Return TmpDate

  End Function

</script>
<BODY>
 <CENTER>
  <table border="1" cellspacing="0" cellpadding="3"
    width="100%" align="center" bordercolor="#000000"
    bordercolorlight="#000000" bordercolordark="#ffffff"
    ID="Table1">
    <tr bgcolor="#205aa7"><td>
        <div align="center"><font color="#ffffff">編號
        </font></div></td>
    <td><div align="center"><font color="#ffffff">公告對象
        </font></div></td>
    <td><div align="center"><font color="#ffffff">公告日期
        </font></div></td>
    <td><div align="center"><font color="#ffffff">公告事項
        </font></div></td>
    <td><div align="center"><font color="#ffffff">公告單位
        </font></div></td></tr>
<%
'將呼叫<Method>的資料放入 System.Data.Dataview Object
  Dim oDS As New DataSet()
  Dim oTable As New DataTable()
  Dim oDataview
  Dim TmpXMLStr,TmpPara,TmpFilter As String

  TmpPara=Request("ID")
  If Trim(TmpPara) <> "" Then
```

```
      TmpFilter="TargetType like '" & TmpPara & "%'"
   Else
      TmpPara=Request("Unit_Name")
      If Trim(TmpPara) <> "" Then
         TmpFilter = "Unit_Name='" & TmpPara &"'"
      Else
         TmpFilter = ""
      End If
   End If

   TmpXMLStr = GetOutsideRent_Data(TmpFilter)

   If Trim(TmpXMLStr) <> "" Then
      Dim XMLReader As StringReader = New StringReader(TmpXMLStr)
      oDS.ReadXml(XMLReader)

      '將 XML 形態的 DataSet,放到 System.data.DataSet 物件中.
       oTable=oDS.Tables(0)
       oDataview = New System.Data.DataView(oTable)


    '將 Dataview Object 中的資料,以 HTML 的方式展現(2/3)
    Dim i As Integer
    Dim TmpPublishNo,TmpTargetType,TmpPressStartDate,
        TmpSubject,TmpPriority,TmpUnit_Name As String

    For i = 0 To (oDataview.Count - 1)
      TmpPublishNo = oDataview.Item(i).Row.Item("PublishNo")
      TmpTargetType = oDataview.Item(i).Row.Item("TargetType")
      TmpPressStartDate =
            oDataview.Item(i).Row.Item("PressStartDate")
      TmpPressStartDate = ToCHTDate(TmpPressStartDate)
      TmpSubject = oDataview.Item(i).Row.Item("Subject")
      TmpPriority = oDataview.Item(i).Row.Item("Priority")
      TmpUnit_Name = oDataview.Item(i).Row.Item("Unit_Name")

      If TmpPriority="最急" Then TmpSubject = "<font size='2'
          color='#FF0000'>(最急) </font>" & TmpSubject

     '將 System.Data.Dataview 中的資料,以 HTML 的方式展現(3/3)

      Response.Write("<TR bgcolor='#dee6f2'>")
      Response.Write("<td><font color='#205aa7'>" & i &
          "</font></td>")
      Response.Write("<td><font color='#205aa7'>" &
           TmpTargetType & "</font></td>")
      Response.Write("<td><font color='#205aa7'>" &
           TmpPressStartDate & "</font></td>")
      Response.Write("<td><a href=ShowNews.aspx?id='" &
           TmpPublishNo & "'><font color='#205aa7'>" &
           TmpSubject & "</font></a></td>")
      Response.Write("<td><font color='#205aa7'>" &
           TmpUnit_Name & "</font></td>")
      Response.Write("</TR>")
    Next i
   End If
%>
</table>
```

```
</CENTER>
</BODY>
</HTML>
```

========== NhuPublicNews.aspx（結束）==========

二、使用 ADO.NET 讀取 XML 資料的 VB.NET 範例。(VB.NET、ADO.NET)

========== 原始碼（起始）==========

```
 Private Sub GetXSLData_Click(ByVal sender As System.Object,
   ByVal e As System.EventArgs) Handles GetXSLData.Click

        Dim Obj
        Dim TmpXMLStr As String

        '呼叫<Windows 2000 COM+ Service>中的 COM Object
        Obj = CreateObject("aCOMToAppServer.AllCOMToAppServer")

        '呼叫<Borland Delphi7 AppServer>中的 Method(),
        '並傳回 XML 形態的 DataSet.

        TmpXMLStr = Obj.GetXSLData("NhuPublicNews.xsl")
        TextBox7.Text = TmpXMLStr
    End Sub
```

========== 原始碼（結束）==========

三、使用 DOM 讀取 XML 資料的 VB.NET 範例。(VB.NET、DOM)

========== 原始碼（起始）==========

```
Private Sub btnReadXML_Click(ByVal sender As System.Object, ByVal
     e As System.EventArgs) Handles btnReadXML.Click

        Dim filePath As String
        Dim Obj, XMLEle
        Dim TmpXMLStr As String
        Dim DataSetStr As String
        Dim XMLDocData As New System.Xml.XmlDocument()
        Dim oDataSet As New System.Data.DataSet()
        Dim oDataTable As New System.Data.DataTable()

        Obj = CreateObject("aCOMToAppServer.AllCOMToAppServer")
        TmpXMLStr = Obj.GetPublicNews("")

        Dim XMLReader As System.IO.StringReader = New
               System.IO.StringReader(TmpXMLStr)

        oDataSet.ReadXml(XMLReader)
        XMLDocData.LoadXml(TmpXMLStr)
```

```
        '將 xml 檔案中,每個 Row 的資料讀出來
        XMLEle = XMLDocData.DocumentElement
        Dim root = XMLEle.FirstChild
        Dim TmpNodeName, TmpNodeValue

        If root.HasChildNodes Then
            Dim i As Integer
            For i = 0 To root.ChildNodes.Count - 1
                TmpNodeName = root.ChildNodes(i).Name
            Next i
        End If

        dsAuthors.Clear()
        With DataGrid1
            .DataSource = oDataSet
        End With
        oDataTable = oDataSet.Tables(0)
    End Sub
```

========== 原始碼（結束）==========

四、使用 SAX 讀取 XML 資料的 VB.NET 範例。(VB.NET、SAX)

========== 原始碼（起始）==========

```
Private Sub Button3_Click(ByVal sender As System.Object, ByVal e
     As System.EventArgs) Handles Button3.Click
        Dim Obj, SaxObj, TmpStr
        Dim TmpXMLStr As String
        Dim oSaxReader As New MSXML2.SAXXMLReader40()
        Dim oSaxWriter As New MSXML2.MXXMLWriter40()
        Dim oSaxHTMLWriter As New MSXML2.MXHTMLWriter40()
        Dim oVBSAXContentHandler As MSXML2.IVBSAXContentHandler
        Dim oVBSAXErrorHandler As MSXML2.IVBSAXErrorHandler
        Dim oSaxAttribute As New MSXML2.SAXAttributes40Class()
        Dim oSaxLocater As MSXML2.IVBSAXLocator
        Dim oSaxDtdHandler As MSXML2.IVBSAXDTDHandler

        '呼叫<Windows 2000 COM+ Service>中的 COM Object
        Obj = CreateObject("aCOMToAppServer.AllCOMToAppServer")

        '呼叫<Borland Delphi7 AppServer>中的 GetOutsideRent
        'Method(),並傳回 XML 形態的 DataSet.
        TmpXMLStr = Obj.GetPublicNews("")
        Dim XMLReader As System.IO.StringReader = New
            System.IO.StringReader(TmpXMLStr)
        Dim XMLDocData As New MSXML2.DOMDocument40()

        XMLDocData.loadXML(TmpXMLStr)

        '==Show By XMLWriter ==
        oSaxReader.contentHandler = oSaxWriter
        oSaxReader.dtdHandler = oSaxWriter
        oSaxReader.errorHandler = oSaxWriter
```

```
        oSaxWriter.indent = True
        oSaxWriter.standalone = True
        oSaxWriter.output = ""
        oSaxWriter.omitXMLDeclaration = True

        oSaxReader.parse(TmpXMLStr)
        oVBSAXContentHandler = oSaxWriter
        TextBox6.Text = oSaxWriter.output

        oVBSAXContentHandler.startDocument()
        oVBSAXContentHandler.startElement("", "ROW", "Unit_Name",
               oSaxAttribute)
        oVBSAXContentHandler.characters("系統發展組")
        oVBSAXContentHandler.endElement("", "ROW", "Unit_Name")
        oVBSAXContentHandler.endDocument()
        MsgBox("ok")
    End Sub
```

========== 原始碼（結束）==========

五、使用 Borland Delphi7 讀取 XML 資料的範例。(Borland Delphi)

========== uSuGetXMLData.pas（起始）==========

```
unit uSuGetXMLData;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms,Dialogs, xmldom, DB, DBClient, MConnect,
  SConnect, StdCtrls, Grids,  DBGrids, Provider, Xmlxform,
  ExtCtrls;

type
  TSuGetXMLData = class(TForm)
    XMLTransformProvider: TXMLTransformProvider;
    CldsTmpXML: TClientDataSet;
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    Socket1: TSocketConnection;
    Panel1: TPanel;
    BtnGetXMLData: TButton;
    CldsTmpXMLPublishNo: TIntegerField;
    CldsTmpXMLTargetType: TStringField;
    CldsTmpXMLPressStartDate: TDateTimeField;
    CldsTmpXMLPressEndDate: TDateTimeField;
    CldsTmpXMLDispatchDate: TDateTimeField;
    CldsTmpXMLDispatchWord: TStringField;
    CldsTmpXMLDispatchNo: TStringField;
    CldsTmpXMLUnit_Name: TStringField;
    CldsTmpXMLSubject: TStringField;
    CldsTmpXMLContents: TStringField;
    CldsTmpXMLLiaise: TStringField;
    CldsTmpXMLLiaiseName: TStringField;
```

```
      CldsTmpXMLLiaisePhone: TStringField;
      CldsTmpXMLLiaiseFax: TStringField;
      CldsTmpXMLEMail: TStringField;
      CldsTmpXMLWebLink: TStringField;
      CldsTmpXMLPriority: TStringField;
      CldsTmpXMLAttachments: TStringField;
      procedure BtnGetXMLDataClick(Sender: TObject);
    private
      { Private declarations }
    public
      { Public declarations }
    end;

  var
    SuGetXMLData: TSuGetXMLData;

  implementation

  {$R *.dfm}

  procedure TSuGetXMLData.BtnGetXMLDataClick(Sender: TObject);
  var TmpXML:String;
  begin
   try
    CldsTmpXML.Close;
    if not Socket1.Connected then Socket1.Connected:=True;
      TmpXML:=Socket1.AppServer.GetPublicNews('');
      XMLTransformProvider.TransformRead.SourceXml:=TmpXML;
      CldsTmpXML.Open;
   finally
    Socket1.Connected:=False;
   end;//try
  end;

  end.
```

================ uSuGetXMLData.pas（結束）================

六、使用 Sybase PowerBuilder9 讀取 XML 資料的範例。(Sysbase PB)

================ CallXML_DataWindow（起始）================

```
double ldb_retn,a[]
integer li_result
String TmpStr
OLEObject loleObject

loleObject = CREATE OLEObject
li_result = loleObject.connectToNewRemoteObject("Rad224",
            "aCOMToAppServer.AllCOMToAppServer")
if li_result < 0 then
  messageBox("錯誤!", "Connecting to OLE Object
             Failed("+string(li_result)+")..." , stopSign!)
  return
end if
TmpStr = loleObject.GetPublicNews("")
```

```
dw_1.importstring(xml!,TmpStr)
loleObject.disConnectObject()
destroy loleObject ;
```

========== CallXML_DataWindow（起始）==========

伍、透過「管道層」連接 MiddleWare 的 Web Services，以 PDA 行動通訊的實作範例原始程式碼：

========== PDA 原始碼（起始）==========

```
Private Sub BtnShow_Click(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles BtnShow.Click

        Dim Obj, XMLEle
        Dim TmpXMLStr As String
        Dim XMLDocData As New System.Xml.XmlDocument
        Dim oDataSet As New System.Data.DataSet
        Dim oDataTable As New System.Data.DataTable
        Dim oDataview As New System.Data.DataView
        Dim oWebService As New WebReference.Service1
        Dim dsData As New DataSet("NewData")

        TmpXMLStr = oWebService.GetPublicNews("")

        Dim XMLReader2 As System.IO.StringReader = New
                System.IO.StringReader(TmpXMLStr)
        Dim XMLReader As System.Xml.XmlTextReader = New
                System.Xml.XmlTextReader(XMLReader2)

        oDataSet.ReadXml(XMLReader)
        XMLEle = oDataSet.GetXml()

        oDataTable = oDataSet.Tables(0)
        DataGrid1.DataSource = oDataTable
        TextBox1.Text = TmpXMLStr
        MsgBox("Get Data From WebService OK!!")

    End Sub
```

========== PDA 原始碼（結束）==========

陸、跨平台的範例實作（以 Redhat Linux7.2 為作業平台，Borland Kylix3.0 為開發工具）

========== Kylix3.0 原始碼（起始）==========

```
unit uSuCallWebServices;

interface
```

```
uses
  SysUtils, Types, Classes, Variants, QTypes, QGraphics, QControls,
    QForms,
  QDialogs, QStdCtrls, DB, DBClient, SOAPConn, InvokeRegistry, TypInfo,
  WebServExp, WSDLBind, XMLSchema, WSDLIntf, SOAPHTTPTrans, SOAPPasInv,
  SOAPHTTPPasInv, WSDLPub, Rio, SOAPHTTPClient, SOAPHTTPDisp,
    WebBrokerSOAP,
  SOAPDomConv, OPToSOAPDomConv,SOAPMidas, xmldom, QGrids, QDBGrids,
  QExtCtrls, Provider, Xmlxform;

type
  TSuCallWebServices = class(TForm)
    HTTPRIO: THTTPRIO;
    XMLTransformProvider: TXMLTransformProvider;
    CldsTmpXML: TClientDataSet;
    CldsTmpXMLPublishNo: TIntegerField;
    CldsTmpXMLTargetType: TStringField;
    CldsTmpXMLUnit_Name: TStringField;
    CldsTmpXMLSubject: TStringField;
    CldsTmpXMLContents: TStringField;
    CldsTmpXMLPressStartDate: TDateTimeField;
    CldsTmpXMLPressEndDate: TDateTimeField;
    CldsTmpXMLDispatchDate: TDateTimeField;
    CldsTmpXMLDispatchWord: TStringField;
    CldsTmpXMLDispatchNo: TStringField;
    CldsTmpXMLLiaise: TStringField;
    CldsTmpXMLLiaiseName: TStringField;
    CldsTmpXMLLiaisePhone: TStringField;
    CldsTmpXMLLiaiseFax: TStringField;
    CldsTmpXMLEMail: TStringField;
    CldsTmpXMLWebLink: TStringField;
    CldsTmpXMLPriority: TStringField;
    CldsTmpXMLAttachments: TStringField;
    DataSource1: TDataSource;
    Panel1: TPanel;
    DBGrid1: TDBGrid;
    BtnGetXMLData: TButton;
    CallWebServices: TButton;
    procedure CallWebServicesClick(Sender: TObject);
    procedure BtnGetXMLDataClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  SuCallWebServices: TSuCallWebServices;

implementation

uses uSuService1;

{$R *.xfm}

procedure TSuCallWebServices.CallWebServicesClick(Sender: TObject);
var TmpStr:String;
```

```
begin
  TmpStr := (HTTPRIO as Service1Soap).GetPublicNews('');
  showmessage(TmpStr);
end;

procedure TSuCallWebServices.BtnGetXMLDataClick(Sender: TObject);
var TmpStr:String;
begin
  CldsTmpXML.Close;
  TmpStr := (HTTPRIO as Service1Soap).GetPublicNews('');
  XMLTransformProvider.TransformRead.SourceXml:=TmpStr;
  CldsTmpXML.Open;end;

end.
```

========== Kylix3.0 原始碼（結束）==========