# 以 XML 為資料交換的管道層架構模組，為舊有的資訊系統注入新活力

# An Approach based On XML's Data Exchange — The New Energy to Legacy System

王昌斌
Chin-Bin Wang
南華大學 電子商務學系
Department of Electronic Commerce Management

蘇全福
Chuan-Fu Su
南華大學 資訊室
Computer Center

楊惠媚
Yuei-Mei Yang
大同技術學院 企業管理學系
Department of Business Administration

Nan-Hua University
Tatung Institute of Commerce Technology

## 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,網路服務,管道層,中介軟體

## ABSTRACT

**Every organization 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 an organization to make a decision whether to update or rebuild the information system once the legacy system might fail to meet new requirements. Our study is to provide an improved approach for the legacy system,by our approach 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.The approach have been excecuse on Nanhua University' s System.**


**Keywords：XML, Web Services, Pipeline Tier, MiddleWare.**

## 壹、緒論

### 一、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。

## 二、研究動機

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)。

## 三、研究目的

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。

**四 、研究方法及架構**

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 ，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「管道層」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。

## 貳、文獻探討

### 一 、舊有的資訊系統

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。

**(一)為何要更新舊有的系統？**

舊有的資訊系統大致可分為三類：1. 大型主機系統（Mainframe）2.主從式架構系統（Client-Server） 3.分散式架構系統（Three-tier Distributed）

對舊有資訊系統的兩種處理方式：1. 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。2.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]本文主要針對第二點做研究，因為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。

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]：1.XML。2.Web Services。3.Wireless Technology

對於新技術的考量與分析，著重於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方法之一。

**(二)、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。

1.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：[8] (1)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。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。(2)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，且佔有的比率最高，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。(3)、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。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[3]，因此我們可以瞭解 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，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。

2. 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] ：(1)資料的提供者(Data Providers)。(2)延伸架構(Subscription-based notification) 。 (3) 複 雜 的 的 商 業 功 能 (Complex Business functionality)。(4)企業間的整合(B2B Collaboration)。(5)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。

**(三)、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方法最主要的原因。

## 二、MiddleWare & 舊有的資訊系統

**(一)、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]

MiddleWare 通常以不同的方式存在於資訊系統中，有下列幾種：[28] 1.IBM 的 Customer Information Control System（CICS）。2.IBM's MQ Series。3.CORBA。4.Microsoft's Component Object Model（COM）。5.Java 2 Enterprise Edition（J2EE）。6.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。

**(二)、MiddleWare 在舊系統所扮演的角色**

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]

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]

**(三)、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 的研究所論述的 MiddleWare，前端與後端的部分都存在著商業邏輯（Business Logic）的部分[24]，這也是舊有的分散式系統所面臨到的問題，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。

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]

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。

## 三、舊資訊系統的新生命力

**(一)、XML**

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。

**(二)、SOAP、WSDL 與 UDDI**

1. 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 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].

2. SOAP 的特色

SOAP 通訊協定所使用的技術，是標準的 HTTP 呼叫與 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 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（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]

3. 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 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]。

4. 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。

UDDI 中包含三個主要的註冊實體[14]：(1)公司的聯絡資訊：一般都將此資訊放在「白頁」(White Pages)中。(2)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。(3)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。

**(三)、Web Services**

1.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。

Web Services 的構成要素有三項 SOAP、WSDL 與 UDDI[12]，綜合以上資訊，我們可得知 XML 是這些技術的最核心觀念，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，這也是本文研究的目的。

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]：(1) 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。(2)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。

2.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」，但 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 的應用應著重在某些重要的資訊[6]。

根據 Michael S.Pallos 的研究提到，服務導向架構，發展出幾個重大的效益[19]，其中以第一點，平衡最初的舊系統投資(Leverage initial investment)：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。

3.Web Services 的應用與挑戰

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，應用在四個方向[4]：其中第一項的 ERP 的應用程式，在其他的領域或與其他公司的互動，正是本文所要探討的研究目的之一。

Steve Vinoski 在其研究中指出，如何善用 Web Services 其中的第四點[28]，商業標準(Business Standards)：更是本文的研究重點。由此可知 Web Services 應用延伸之廣泛。

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 Web Services 所面臨的挑戰[4]，其中以安全問題為最主要。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，已發行了 Web Services Enhancements (WSE 2.0)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 依然面臨著問題[27]。

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]。且 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]：(1).許多 MiddleWare 的產品，使用不同的模組來做溝通。(2).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。(3).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。

## 參、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。

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。分散式系統架構在延伸發展都受到環境的限制，例如 Microsoft 的 DCOM 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。

### 一、分散式架構資訊系統的限制

#### (一)、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 DCOM 與 OMG 的 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，由於元件本身具有封裝性，因此對於元件之間的溝通，也因為元件本身的封裝性緣故，因此造成不同平台之間的元件溝通障礙，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。

**(二)、應用程式間的溝通與整合限制**

**1.元物件模組(Component Object Model, COM)**

在相同的機器設備下，Microsoft 的 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 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 RPC 遠端呼叫的概念，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。

**2. 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。

**3.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。

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，仍然以，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。

## 二、以 XML 為資料交換格式的概念

### (一)、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。

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「如圖 1」，而在分散式架構上建立起不同機器間的關聯，即是 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。

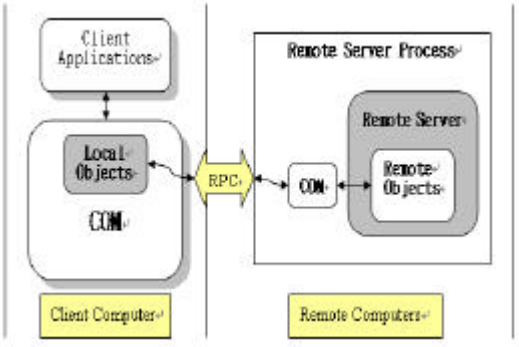



圖 1. 遠端 COM 之間的溝通圖

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「如圖 2」，因此常受限於作業系統平台與開發工具。

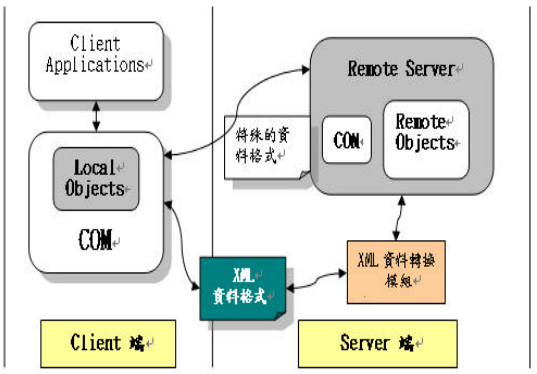



圖 2. 由 COM 所封裝的特殊格式資料

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：

**1. 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。

**2. 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。

**(二)、「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，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。

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。

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，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]

## 三、 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。

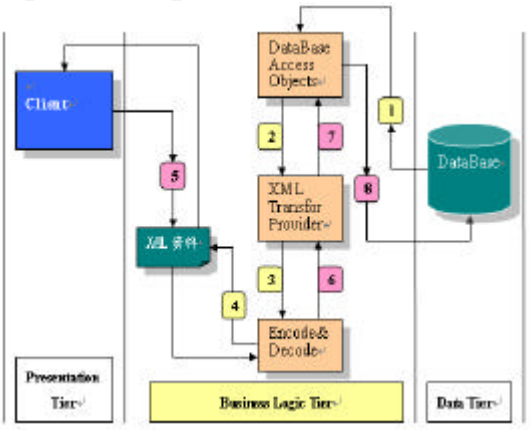



圖 3. XML 的資料轉換模組的運作

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「如圖 3」。

**(一)、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。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，如此才能在不變動到舊有的資訊系統架構與運作。

本文針對舊有的資訊系統注入新活力，由於舊有的資訊系統，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」。

**(二)、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，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「如圖 4」。

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。

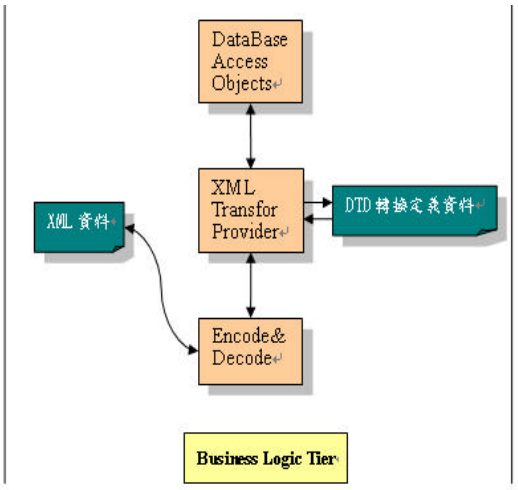



圖 4. XML Transfor Provider 資料轉換 DTD 參照圖

透過 XML Transfor Provider 參照 DTD 轉換檔，將資料庫的資料轉換成標準的 XML 格式資料。至於 Windows 平台常用的 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。

**(三)、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 Encode/Decode，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。

**(四)、延伸應用**

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「如圖 5」。

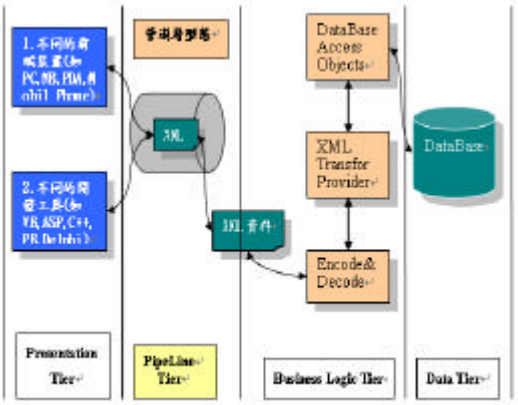



圖 5. XML 的資料轉換模組與「管道層」架構圖

## 四、結合舊有系統的延伸性架構–管道層（Pipeline Tier）

**(一)、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「如圖 6」，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。

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

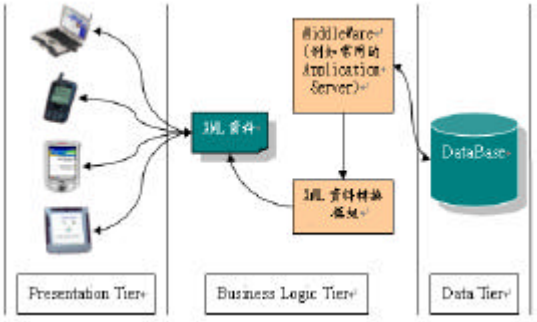



圖 6. 以 XML 為資料交換的架構

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。

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：(1).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。(2).開發工具技術：如 Borland Delphi 7.0 版，可以透過 XML 相關處理 VCL 元件，將資料轉換為 XML 格式。如 Microsoft Visual Studio .NET 中可以使用 ADO 或 ADO.NET 將資料轉換為 XML 格式。如 Sybase PowerBuilder 9 中可以透過 Datawindow，以 Export 的方式將資料轉換為 XML。

**(二)、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「如圖 7」，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。

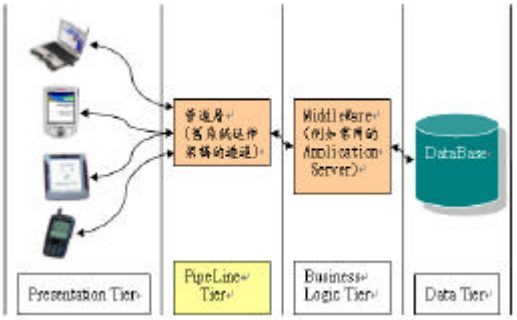



圖 7. 連接舊有分散式系統的管道層

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。

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 的方式存在，或是其他工具所開發出來的型態「如圖 8」，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。

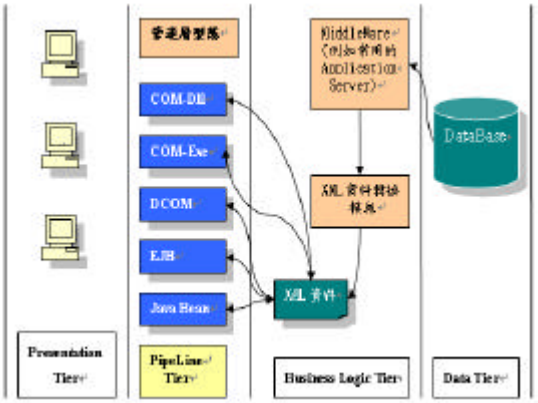



圖 8. 「管道層」的呈現方式

管道層的作用，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「如圖 9」，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。

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。

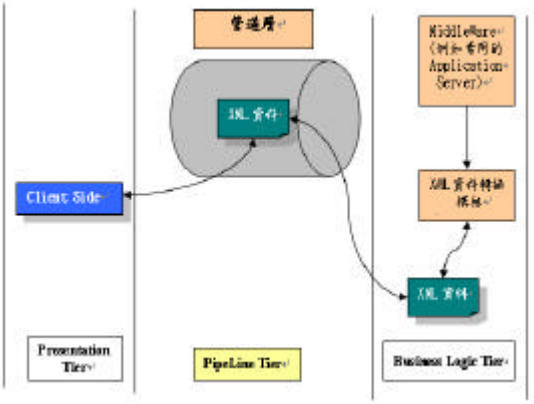



圖 9. 連接 MiddleWare 的「管道層」

**(三)、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：

**1.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「如圖 7」。

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「如圖 10」。

於圖 7 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。

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]。

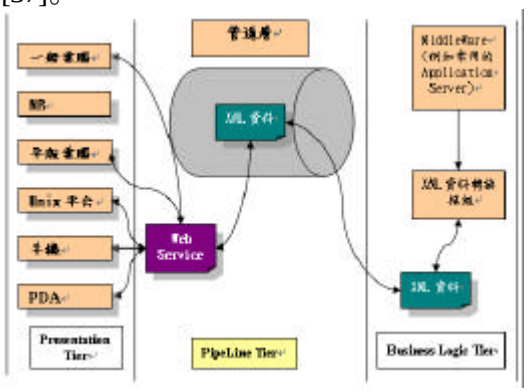



圖 10. 透過「管道層」連接各種裝置

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。

**2. 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，因此提出此架構的用意，不在於作底層的技術探討，而是強調新架構的整合與發展，管道層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，但管道層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「如圖 11」，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。

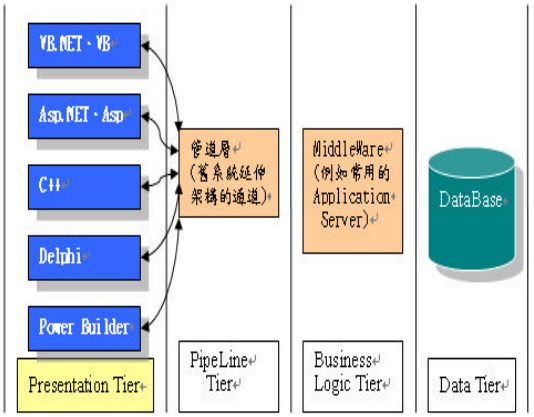



圖 11. 連接管道層的跨語言架構

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：

**(1). 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」「如圖 12」，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。

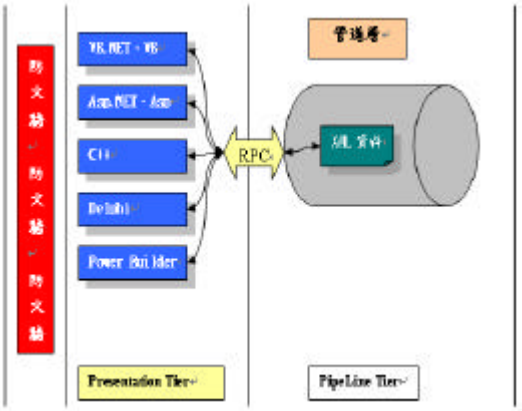



圖 12. 在防火牆內連接「管道層」的跨語言架構

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變慢。

**(2). 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」「如圖 13」，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。

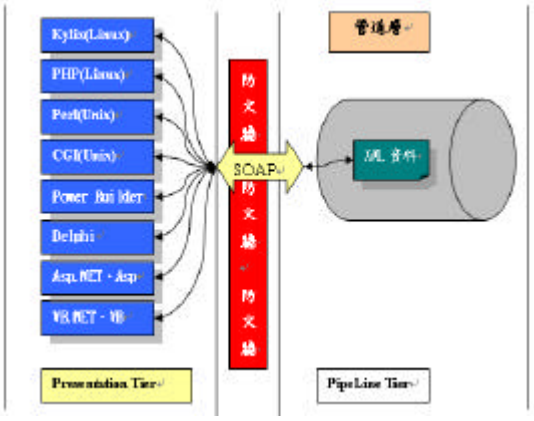



圖 13. 在防火牆外連接「管道層」的跨語言架構

**(四)、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，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。

防火牆的概念在於針對在所設定對規則外，將一切於規則外的連結與連線需求全部阻絕，針對此點「管道層」之所以能成為第二道防火牆，其意義相同。例如在 Windows 作業系統的環境中，「管道層」可以使用 COM 的方式呈現，在於 COM 本身的封裝特性上，如 RPC 的呼叫就已經具備防火牆的功能，如同本文 2.1 中「解決資料交換與介面呼叫問題」所介紹有關 COM 的特性。如此只要注意在實際呈現「管道層」時，其中有關方法（Method）與屬性（Property）的設計時特別注意，不要有類似隱碼攻擊 SQL injection 的設計漏洞即可防範此類攻擊。

再者針對類似分散式阻斷攻擊，「管道層」本身就能阻擋一般針對 80port 的攻擊，因為光是送短封包並無法連上「管道層」，例如最常見的登錄系統安全的帳號檢查，一般的分散式阻斷攻擊，並無法連帶將使用者的帳號密碼連帶送出，即使可以如同網站的壓力測試，也不會使機器停擺，因為關於此點作業系統或工具，將會替我們將此問題排除，以 MS Windows 作業系統

為例，在 IIS 部分有關惡意壓力測試的攻擊，將會替我們排除。

「管道層」如果結合中間層的改良工具，將能更強化效能，如（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]，即是一個最佳例子。所要解決即是加強商業邏輯層的效率，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。

## 肆、驗證與實作

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 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。

### 一、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。

#### (一)、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。

轉換檔是一個 XML 型態的檔案「如圖

14」，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。

```
<XmlTransformation Version="1.0"><Transform Direction="ToXml">
<SelectEach from="DATAPACKET\ROW\DATA\ROW" dest="\Document\ROW">
 <xs:element name="Document" type="DocumentType"/>
 <xs:complexType name="DocumentType">
  <xs:sequence>
   <xs:element name="ROW" type="ROWType"/>
  </xs:sequence>
 </xs:complexType>
 <xs:element name="ROW" type="ROWType"/>
 <xs:complexType name="ROWType">
  <xs:sequence>
   <xs:element name="SeqNo" type="SeqNoType"/>
   <xs:element name="Landlord" type="LandlordType"/>
   <xs:element name="LandlordTel" type="LandlordTelType"/>
   <xs:element name="LandlordAddr" type="LandlordAddrType"/>
   <xs:element name="RentZone" type="RentZoneType"/>
   <xs:element name="RentAddr" type="RentAddrType"/>
   <xs:element name="SquareCnt" type="SquareCntType"/>
```

圖 14. 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「如圖 15」，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。

**(二)、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]：

```
<?xml version="1.0" encoding="utf-8"?>
<string xmlns="http://203.72.2.24/webservice1/"><?xml version="1.0" encoding="BIG5"?>
<Document>
<ROW>
  <PublishNo>746</PublishNo>
  <TargetType>02,學生</TargetType>
  <PressStartDate>20041021</PressStartDate>
  <PressEndDate>20041118</PressEndDate>
  <DispatchDate>20041021</DispatchDate>
  <DispatchWord>南華學內</DispatchWord>
  <DispatchNo>093102101</DispatchNo>
  <Unit_Name>學生事務處</Unit_Name>
  <Subject>失業勞工子女就學補助，每名一萬元！</Subject>
  <Contents>學務處 公告 失業勞工子女就學補助，每名一萬元！一，申請日期：即日起，至9
  <Priority>平常</Priority>
  <Attachments></Attachments>
</ROW>
</Document></string>
```

圖 15. 轉換後的 XML 檔案格式的內容

**1.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）。

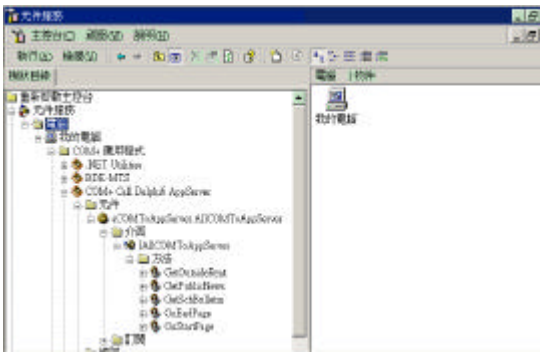
**2.EXE(out-process)**：大多屬獨立功能性的型態，例如 Borland Delphi 以後的版本，多以.EXE 型態建立系統所需要的 MiddleWare。例如在本範例中舊有的資訊系統中，以 Borland Delphi 建立.EXE 型態的 COM 成為舊有系統的 MiddleWare，因此在管道層的建立上，只要遵守著 Windows 作業系統平台上的 COM 架構建立的規範，即可以使用適當的工具實作出管道層。

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。

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「如圖 16」，透過管道層的建置連接到舊有系統的 MiddleWare，即可完成本文所提出的改良架構。

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。

圖 16. 管道層在 Windows 2000 Server 的呈現

### (三)、透過管道層連接 MiddleWare 的 Web Services

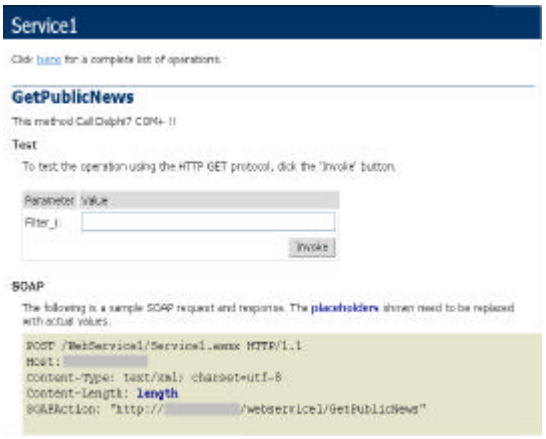



圖 17. 以 Viaul Studio.NET 所撰寫的 Web Services

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 來實作透過管道層連接到 MiddleWare 的 Web Services「如圖 17」，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。

透過管道層連接到 MiddleWare 的 Web Services，除了解決上述的問題之外，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）。

## 二、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，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：

### (一)、使用 Microsoft Visual Studio .NET 讀取 XML 資料的範例

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，就必須透過文件物件模組(Document Object Model, DOM) [38]的技術來承接資料，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。

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「如圖 18」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。

針對舊版 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。

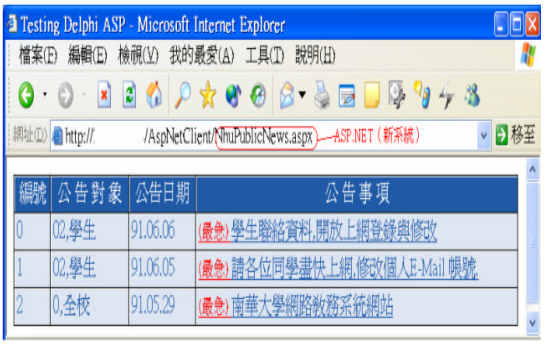



圖 18. 以 DOM 的方式接收 XML 資料的 ASP.NET 程式

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「如圖 19」。

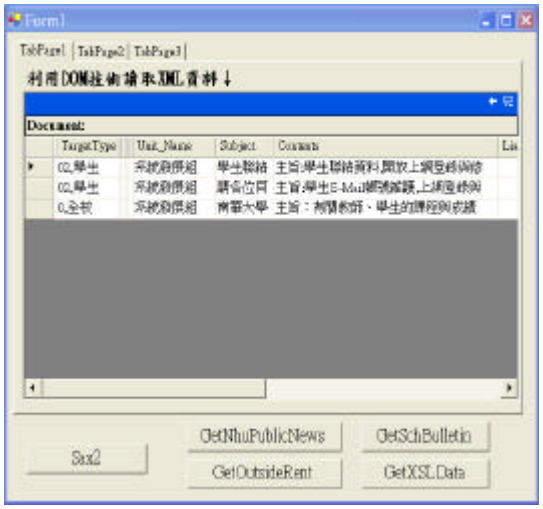



圖 19. 以 DOM 的方式接收 XML 資料的 VB.NET 程式

DOM 本身的限制在於對大量資料存取，造成執行效能不佳，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。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。

**(二)、使用 Borland Delphi7 讀取 XML 資料的範例**

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「如圖 20」。

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 20. 以 Borland Delphi7 程式呈現 XML 資料

### (三)、使用 Sybase PowerBuilder9 讀取 XML 資料的範例

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 21」，DataWindow 是使用 Sybase PowerBuilder 為開發工具經常使用的物件。

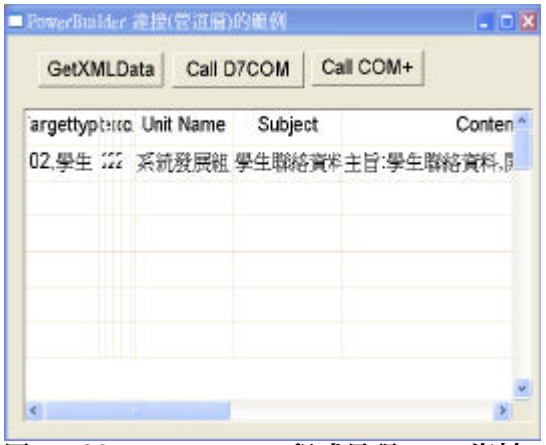



圖 21. 以 PowerBuilder9 程式呈現 XML 資料

由上述三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。

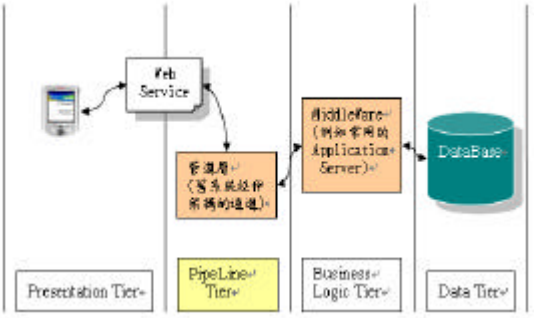



圖 22. 連接管道層的 Web Services

## 三、結合 Web Services 的實作

### (一)、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「如圖 22」，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：

**1. 建置透過管道層連接舊資訊系統 MiddleWare 的 Web Services：**

「如圖 22」。

**2.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，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「如圖 22」，而 Web Services 在本文的範例中是以 ASP.NET 所開發。

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，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「如圖 22」。

但有關於開發工具的限制，以 Microsoft Visual Studio.NET 2003 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「如圖 23」。

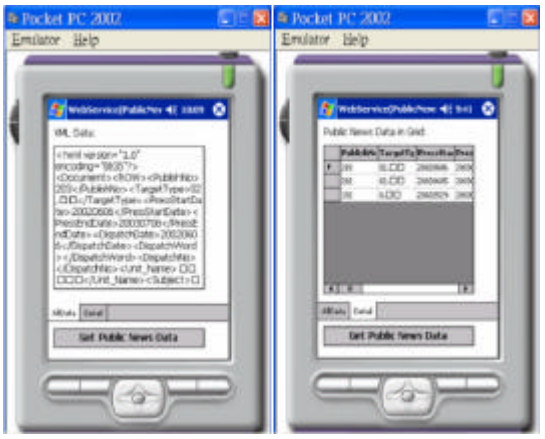



圖 23. 以 PDA 接收 XML 資料的範例

**(二)、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「如圖 24」。

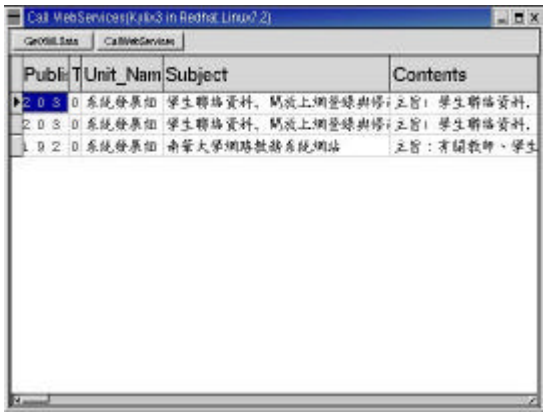



圖 24. 在 Linux平台上接收「管道層」所傳出的 XML 資料範例

## 伍、結論與建議

### 一、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：

**（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：**

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。

**（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：**

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。

**（三）、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。

## 二、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。

## 致謝

本文所提出的架構，已廣泛應用於「南華大學校務行政系統」。特此感謝南華大學，資訊室王昌斌主任、資訊室系統發展組曾清義組長，在系統實作上的相關協助。

## 參考文獻

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。

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.
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/