私 立 東 海 大 學

資 訊 工 程 與 科 學 所

碩 士 論 文

指導教授：朱正忠 博士

利用多階層軟體架構幫助現有系統導入網路服務標準

Using Multiple tier Software Architecture to Migrate Systems to Web Services

研 究 生：王靜慧

中 華 民 國 九 十 一 年 六 月

摘　要

網路興起帶動電子商務風潮，企業趨向 e 化，資料傳遞注重移動性(m 化)，這個潮流增長人們對網路的需求，過往的靜態資料已無法滿足使用者需求，網路服務(Web Services)因此崛起。網路服務有別於網頁服務，企業資源透過網路釋放，這項服務可以是一個系統(辦公室系統)、環境(高速網路環境)，區域性服務經由網路拓展成跨國際服務，應用軟體透過網路釋放資源，使用者利用網路使用被釋放的資源。

同時新興平台不斷增加，可攜式硬體日漸普及，無線電話(cell phone)、個人數位助理(Personal Digital Assistant)、手提電腦(Notebook)使用率年年上升，所以現有系統導入新硬體平台日趨重要。

有效的企業電子化可以增加公司組織的產業競爭力，但是數位化的資訊或應用軟體若無有效的管理整合機制會導致資訊混亂，資料檔案不一致等後遺症。完整的事件處理流程，加上有系統的應用處理程式，可以確保資料的正確性，同時可以降低時間、人力成本。

也就是說，公司企業不只需要電子化，更要對企業內部數位化後的應用程式處理程序及電子資料進行管理。就組織內部而言，各部門間應用軟體的整合，可以完成組織內部數位化工作流程，達到部門間的資訊交流，保證系統間的資料一至性。就組織對外溝通而言，企業與企業之間，可以透過系統整合達成溝通模式，也就是說公司內部的應用程式與商業夥伴直接進行溝通，形成網路交易鏈，免去過往的紙上作業。

面對組織內部應用軟體整合的迫切性，企業間交易電子化的必要性，與新硬體平台導入的急切性，加上網路服務標準(Web Services)是當今企業組織進行

系統整合最好的方法，但卻苦不見一套完整的設計方法，有效的解決系統整合問題。所以本篇論文利用 Multiple-tier 軟體架構加上設計模型(Design Patterns)結合 XML 的觀念提出一個架構，這個架構幫助程式開發者整合舊有系統(EAI)，導入網路服務(Web Services)環境。

關鍵詞：*網路服務、系統整合、設計樣版、延伸式記語言、Multiple-tier 系統架構*

Abstract

The rise of Internet promotes tide of e-commerce. Enterprise trends toward electronic enterprise. The exchange of data is emphasized the mobility. This trend extends the network requirement that people expect. Because users don't satisfy static data format, Web Services rises sharply. In Web Services, services are different from web pages and released by enterprise resources. The Web Services can be a system (office organization system) or environment (high speed network environment). Regional services can be extended to transnational services. Applications release resources and users use released resources in way of network.

Today new platforms and popularization of portable hardware increase. Utility rates of cell phone, Person Digital Assistant, and laptop raise by years. Therefore, it's important that legacy systems support new hardware platforms.

Efficient e-enterprise can increase the competitiveness of industry. However, digital information or applications without efficient cooperative management may cause the data confuse or inconsistence in data file. Complete event handle flow with systematic handle applications can ensure the data correctness and decrease time and personnel cost.

Furthermore, enterprise needs not only digitalize, but also management the application handing processes and digital data. Inside the organization, integrating applications in each apartment can complete the digital workflow inside the organization. It could attain the data consistence between systems. Outside the organization, communicational model can be procured by system integration. Therefore the inner applications inside the enterprise can communicate directly with business partners. This mechanism shapes the business chain and avoids the paper work in the past.

Faced with the immediacy of integrating inner applications, the necessary of the electronic business between enterprises, and urgency of supporting new hardware

platforms, system designer suffer form a complete design plan. In addition, Web Services is the best standard for integrating enterprise systems. Therefore, this paper provides a framework that combines multiple-tier software architecture, Design Patterns, and XML. This framework helps system designers to integrate legacy systems to support the Web Services architecture.

Keyword: Web Services, EAI, Design Patterns, XML, Multiple-tier Software Archiecture

章 節 目 錄

圖 表 目 錄

第一章 序論

1.1 前言

龐大的企業組織、企業夥伴，現今的通訊模式多半透過電話傳真或是書面的方式傳遞資訊，這些有形或無形的資訊需要高額的人力成本，同時容易因為人為疏失延誤交易時程。這樣的情況觸發電子商務的興起，以電子化的方式處理企業間資訊的傳遞，這樣的模式由過往的紙上作業晉升為應用程式對應用程式的運作模式，不但可以增加交易時效，同時可以降低人力成本。

有效的企業電子化可以增加公司組織的產業競爭力，但是數位化的資訊或應用軟體若無有效的管理整合機制會導致資訊混亂，資料檔案不一致等後遺症。完整的事件處理流程加上有系統的應用處理程式，可以確保資料的正確性，同時可以降低時間、人力成本。

再加上，硬體平台日新月異，輕薄型硬體風行，傳統系統通常不支援這些硬

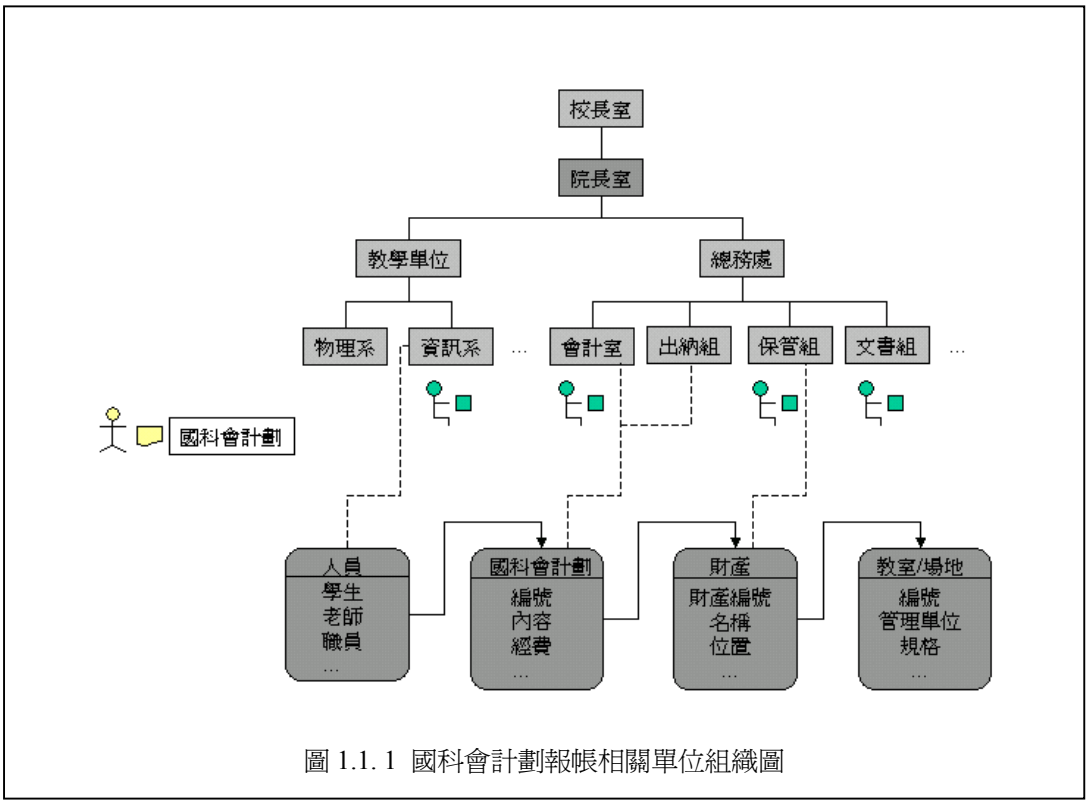


圖 1.1. 1 國科會計劃報帳相關單位組織圖

體設備，導入新興平台刻不容緩。現有系統的封閉性及軟體特性，常常增加程式開發者擴充困難度，而且面對多重的硬體平台及應用程式，有效的導入方式是現今資訊產業極力想解決的問題。

舉一個校園內經常發生的問題，校內公文如國科會計劃的報帳流程，，計劃內容包括計劃編號、本文、核定經費...等，計劃報帳的過程關連到學校的會計室、出納組、保管組、文書組等單位，這些單位之間通常透過有形文件的傳遞來完成溝通，所以對於一個報帳者而言，它必須帶著公文依據公文處理流程走訪各處室。基本上，這樣的公文處理流程都是在處理國科會計劃相關的事情，它應該是一份完整的資料，記在所有國科會計劃相關資訊，包括財產編號、經費運用情況、期中報告、期末報告...等，而不是分散在各地如圖 1.1.1 所示，這些資訊需要透過人為處理才能達成一致。

圖 1.1.2 描述了校園各單位內部的應用系統整合機制，校園系統管理的資訊莫過於學生資訊、老師資訊、課程資訊、有形的財產、教室、場地、文件...等等，這些資料目前都由不同的處室來管理，但是有些資訊是相關性相當高的，而且這些資料是不可被分割的，如國科會計劃，計劃編號、計劃購置的財產、帳目、文件等這些代表的是一份國科會計劃，不能被不同的應用系統如帳目系統、財產編列系統、發文系統所分割，相關的資訊分別被記載在不同的系統，這樣的情況造

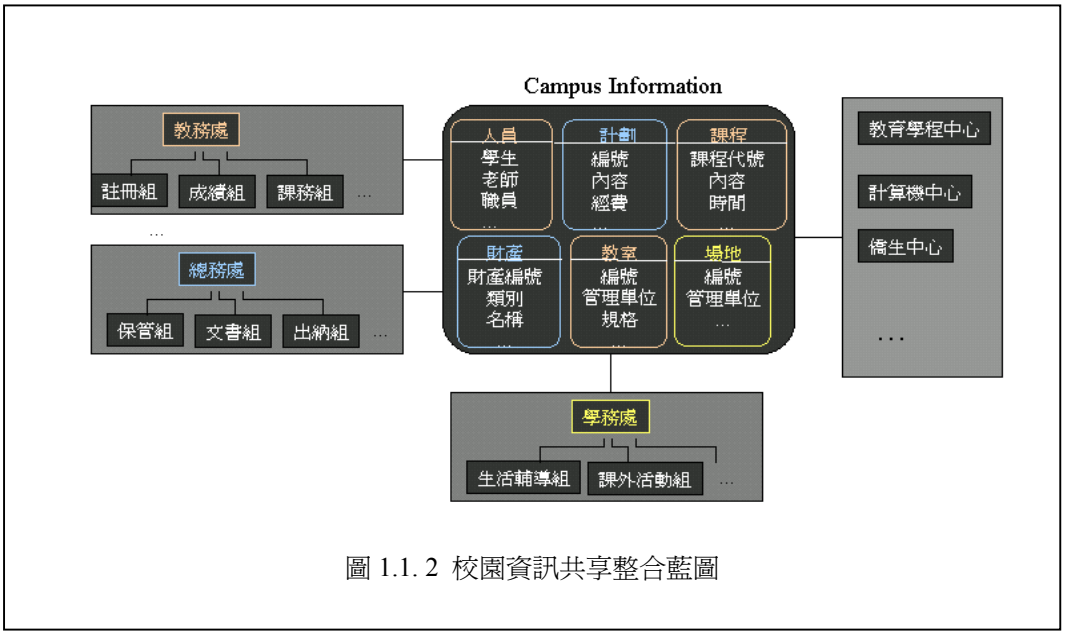


圖 1.1. 2 校園資訊共享整合藍圖

成使用者極大的困擾，所以這些相關聯的系統必須進行整合，適當的釋放系統內部相關資訊，如此一來系統與系統間的溝通不在依靠人力輸入，系統可以輕易取得相關資訊，同時可以維護資料的一致性。

1.2 研究動機

系統整合是企業組織勢在必行的趨勢，過去有許多研究利用分散的方式進行系統整合，其中包括 Java Message Service(JMS)配合目錄式服務(Directory Service)，Java Remote Method Invocation(RMI)、CORBA 等等，但是這些技術都無法完全滿足跨平台、跨語言、運用一般的通訊協定(http)的問題，因此網路服務(Web Services)崛起，網路服務之所以戰勝 CORBA 是因為網路服務基礎於最普遍的通訊協定 http，網路服務之所以位居 JMS、RMI、目錄服務之上，是因為網路服務運用最一般性的通訊協定及跨平台的訊息格式，所以可以套用在任何語言發展的應用軟體之上，以這樣的優勢未來系統整合必定朝向網路服務標準。

由上述可知，企業必定朝這個方向邁進，以分散式的環境為基礎，企業內部進行適度的系統整合，分享內部資源。各家大廠有見於這塊大餅紛紛定義網路服務階段性標準如 WSDL、UDDI，同時提供一些幫助開發的套件如 SUN 的 Web Service Pack，但是這些都只是局部的標準與套件，有鑒於當今資訊產業尚未有一套完整的導入網路服務程序，連結所有的標準和套件，有效的幫助傳統系統導入網路服務機制，同時提供平台擴充機制，以調整軟體擴充系統功能，導入元件模組，建製高彈性合作機制。

希望依據上述這些方向，幫助現有系統導入架構，有效的進行系統整合，近而提昇企業組織生產效率，降低人力時間成本，增加企業競爭力。

1.3 研究目的

本研究之主要目的為利用多階層軟體架構幫助現有系統導入網路服務標準，透過這樣的方式達到系統整合及資源分享的目的。

詳細的項目包括下列四項：

1. 提供分散式系統整合機制：網路服務是一種分散式系統整合標準，以分散式的環境為基礎，資源可以被置放在不同的位址上，系統間利用網路服務標準文件，擷取有效資源的相關資訊，包括真實位址及訊息傳遞格式等，透過這樣的方式系統達成溝通。

2. 動態的建立元件合作機制：一個網路服務可能由多個來自不同系統的元件組成，面對這些元件，本研究提出一套動態的元件合作的模組，透過這個模組元件可以動態的建立元件合作程序，降低系統設計的複雜度。

3. 軟體元件包裝機制：網路服務中的元件是來自於一至多個應用系統，因為它們的來源不同所以包裝的方式也不同，本研究對於元件包裝提出的幾種客制化的方式，幫助建立具合作機制的網路服務元件。

4. 平台獨立：由於本研究的研究重心環繞著網路服務這個標準，所以便繼承網路服務所提供的特性，那就是 XML 文件。XML 文件以文字模式存在，任何應用程式可以輕易的讀寫它，所以具有跨平台得效果。

1.4 章節安排

本論文之架構如下：第二章為本論文的背景知識，第三章定義階層功能，分別為各階層的目的進行定義。明定各個階層的目的之後，各個階層為了達到這些目的可能面臨一些問題，第四章透過設計樣板勾畫階層設計藍圖，所以第四章說明設計樣板與階層功能的映射關係。第五章介紹多階層架構實作設計，具體指出各個階層的設計架構及程式設計方法。第六章為結論，第七章為本研究之未來工

作，說明本研究之未來發展方向，第八章列舉所有相關的參考文獻。

第二章 背景知識

2.1 多階層軟體架構

多階層軟體架構(N-tier Software Architecture)[9]以多個階層劃分軟體設計架構，階層間透過溝通介面進行存取，階層只與相鄰的兩個階層有關係，程式設計師可以任意置換一個階層，達到隨插即用(plug and play)的特性，這個特性增加軟體設計彈性與再使用性。

N-Tier Architecture 是近年來軟體開發技術中最熱門的主題，除了 Microsoft 的開發架構從 DDE、OLE、COM、DCOM 到未來的 COM+，開發方式推陳出新，讓人目不暇給，而在 Delphi 的發展上也由 3.0 開始支援 N-Tier 開發架構，Delphi4.0 更加入了可直接開發 MTS(Microsoft Transaction Server)及 CORBA 的功能。

2.2 網路服務

網路服務(Web Service)利用 XML 透過 HTTP 通訊協定將網路推向下一個紀元，B2B、B2C 的商業模式，企業利用延伸式標記語言(XML)模式描述自己提供的服務，使用者利用這些資訊找尋自己需要的服務，異質系統平台也利用延伸式標記語言(XML)文件作為資訊傳輸媒介，網路服務(Web Service)利用這樣的理念使物件導向、非物件導向的系統均能進行溝通。

網路服務(Web Service)運用了多項延伸式標記語言相關標準如 UDDI[10]、WSDL[11]，Universal Description, Discovery, and Integration (UDDI)是一組網路註冊中心的名字，這些註冊中心儲存商品資訊及服務相關技術的溝通介面(API)。通過註冊的任意一個公共的 UDDI 入口站(Operator Site)，集合所有商品資訊，使用者可以查詢商品或代表公司實體的 Web 服務的相關信息。獲得這些服務信息有效的提供了一種讓他人找到企業直接交易的管道。這樣有利於小型或個人企業

在這個電子商務的時代大幅度地提高自身的公開性。

一個商業實體可以註冊的信息包括若干種數據。這些數據能夠為那些意圖發現服務的用戶提供訊息，包括(1)商業實體的名稱，商業實體的標誌和聯絡方式；(2)基本分類如企業代碼與產品分類，以及資料交換方式；(3)服務網址與每種類型的服務入口相關的位址(URL)或 Email 等註冊信息；(4)一些註冊的指引信息，這些指引分別引導用戶如何走訪特定軟體的路徑或技術接口的訊息交換格式。這些引用在 UDDI 的相關文件被稱為 tModel。

為了讓兩個或更多的軟體得以互相兼容，也就是說，為達到我們期望的互相交換數據的目標，它們必須共享一些數據資料和通用規範，所有的 UDDI 入口站都支持註冊信息模組就是基於這樣的理念。過去，為了架構一個具有兼容能力的軟體，兩家公司唯一可以執行的方式，就是使用同樣的規範達成溝通。這種規範信息，也就是傳統上稱作的元素結構，在 UDDI 中稱其為 tModel。

WSDL(Web Service Description Language)[11]是一種 XML Schema，由 IBM、Microsoft 兩家公司定義出，提供制定網路服務定義的描述語法(Web Service definitions)，此 WSDL schema 包括 Types, Message, Operation, Port Type, Binding, Port, Service 等七部份，Types 定義 WSDL 文件中所使用的資料型態，Message 描述傳遞的訊息，Operation 定義一些處理這訊息的方法(method)、商業程序(business process)或運作模式(operation)，Port Type 定義一系列的連結模式(binding operations)提供連結到符合的供應商。Port Type 中的 binding operations 如何真正達成目的呢？是由 Binding 來進行定義，Binding 分別定義傳出的資料型態及訊協定，Port 結合 Binding 與網路位址建立連節，Service 收集了一系列相關的服務定義，將這些資訊紀錄在檔案中。

服務提供者利用註冊標準 UDDI 進行註冊、公告，利用服務描述標準 WSDL 說明使用者程式如何能夠連接到網路服務(Web service)，描述遠端資訊存取格

式、商業交易流程(business process)、供應鏈(supply chain)、網路位址及傳輸方式等資訊。

2.3 設計樣板

設計樣板是由一些專家在經過成功經驗的累積，針對許多經常發生的設計問題歸納出各類可重複使用的解決方式。

Gamma et. al.[12]在 1994 年提出設計樣板(Design Patterns)，設計樣版的應用常被定位在解讀性高，快速建置軟體架構上，在 Gamma et. al.列舉了三大類設計樣板，這些設計樣板都可以應用在資料結構、事件處理、訊息傳遞等常見的設計，加強了系統在設計時的可讀性及在使用性。設計樣版便成為幫助在特殊應用領域中了解軟體元件中舊有的架構與設計概念格式基礎。關於設計樣版對應(patterns mapping)方面，在 Moriconi，Qian 與 Riemenschneider[13]中提出了一種利用一些事前定義好的，且經過證明為正確的 refinement pattern，以 interpretation mapping 的方式，將較高階抽象的 software Architecture 轉換成正確、實際低階的 Software Architecture，快速的建構出可再使用度高的系統。

2.4 延伸式標記語言

目前網站上最流行的 HTML 是 SGML 的一個子集，專對資料展現進行描述，包括字型、顏色、大小、格式等等。但是由於其固定標籤(tag)的設計，只提供了在網路作業中資訊展現的功能，而無法提供資料再使用、資料交換等功能，限制了應用系統方面的發展。基於此需求，W3C (World Web Consortium)於 1998 年初公佈了 XML[14]（eXtensible Markup Language）的標準，提供了一個可自訂標示名稱功能的標記語言(Markup Language)，使用者可以很容易的運用 XML 在網路上自行製作所需要的文件內容及格式，並廣泛的發展不同層次的應用。XML

不但具有標記語言的特性外，也可以視為是中介語言(meta-language)，因為它可以讓建置者自行定義語法規則，亦即透過定義的延伸，指定標籤的語法，而不是端視標籤本身的定義。

此外，因為具有 meta-language 的特性，XML 也被常用於描述各種特殊領用資訊，如應用在數學公式上、化學方程式上以及其他的專業上的標準格式。在美國一個宣傳和促進在網際網路上面做電子交易的 CommerceNet 組織，就召集了一些業界的成員來制訂電子交易專用的 DTD，以支援 XML 的 EDI 文件格式。

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

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

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

Sun 提供了一些專門解決 XML 與 Web Service 問題的套件 Web Service pack，其中包括 JAXB[15]、JAXM[16]、JAXR[17]、JAX-RPC[18]、等 API，以

下對這些套件作一些描述。

JAXB

The JavaTM API for XML Binding (JAXB)，JAXB 提供一種快速且方便的 two-way mapping 方法，他對 XML document 和 Java objects 進行 mapping，使用者給定一個 DTD，JAXB compiler 會產生一系列的 Java source，JAXB compiler 如何 compile 這個 DTD 是基於一個 schema，在這個 schema 當中描述那一個參數是類別名稱，那些是變數名稱，哪些是 method，程式開發者便可以實作這些類別產生適當的物件。

JAXM

The JavaTM API for XML Messaging (JAXM)提供一種標準的方法在 Java 的平台上透過適當的通訊定傳送 XML documents。這項標準基於 SOAP 1.1 和 SOAP Attachments specifications。

JAXR

The JavaTM API for XML Registries (JAXR) 提供一種標準的方式透過網路存取 standard business registries，對程式開發者而言，JAXR 提供一種 Java-based 標準存取方法 business registries，在這所指的 business registries 可能是 ebXML 或是 Industry consortium-led specifications 如 UDDI 等。我們可以利用 JAXR 幫助我們做兩件事，註冊和搜尋，企業可以利用 registries 對子自己的企業在網路上進行註冊，如各個企業都以標準的方式進行註冊，如此一來 JAXR 參照各個 service registries information 幫助企業找到適當的供應商或是中盤商等等合作夥伴，這些動作不在沿用傳統電話或是傳真的方式，完全透過網路。

JAX-RPC

Java 已經以兩種 API 能夠進行 remote procedure calls 的動作，一是 Java IDL 它基於 CORB A architecture，另一個是 Remote Method Invocation(RMI)，The JavaTM API for XML-based RPC (JAX-RPC)運用 XML 的方式來進行 remote procedure calls，這是與前兩者不同的地方。

第三章 多階層系統架構—階層定義

本論文針對系統整合導入 Web Services 設計了五階層架構(如圖 3.1 所示)，幫助程式設計師迅速整合現有系統導入網路服務環境，降低人力成本，增加電子商務商機。這五層分別為使用者層(Client tier)、網路服務層(Web Services Tier)、合作層(Collaboration Tier)、包裝層(Wrapper Tier)、應用層(Application Tier)，此階層架構由標準的三層架構延伸而來，因為五層架構中的網路服務層(Web Services Tier)、合作層(Collaboration Tier)、包裝層(Wrapper Tier)可視為一個連接器(connector)接合使用者端和與網路服務應用程式。這三個階層囊括網路服務建置的所有動作，以階層的方式劃分使用者應用程式和應用層系統。

Client Tier 提供使用者端應用程式，這些應用程式可以幫助各種不同的硬體平台處理取得服務前的前制動作，包括擷取網路服務位址、溝通格式等，使用者可以利用不同平台上的瀏覽器連結這些應用程式，因此而取得適當的網路服務。

Web Services Tier 紀錄網路服務的註冊資料，這些資訊來自於網路服務提供者，他們利用公認的註冊標準開放自己的資源給特定的使用者或是所有使用者，這些註冊資料包含網路服務的相關資訊，包括服務資訊、商品資訊、使用位址等，提供使用者搜尋。系統可以透過這個階層所提供的系統服務資訊找到關連的系統進行動態整合連結，企業間的應用系統透過這樣的方式進行交易，公家機關間的應用程式透過這樣的模式達到資料交換，無形中建製了分散的整合性系統環境。

Collaboration Tier 一個網路服務包含了許多的元件，這些元件原本是各自讀地的個體，為達到合作的目的，本論文設計一套動態建立元件合作模式的方法，這個機制有效的解決複雜的元件溝通問題，同時程式設計師可以很輕易的擴充事件處理元件合作模式及置換元件合作模式。

Wrapper Tier 負責包裝軟體元件，這些元件可能來自不同的應用系統，可能是建構在不同的應用程式之上，可能是依循不同的設計架構。在這樣多變的情況之下，本研究針對元件包裝、元件內部事件處理及客制化元件功能三點進行設計。

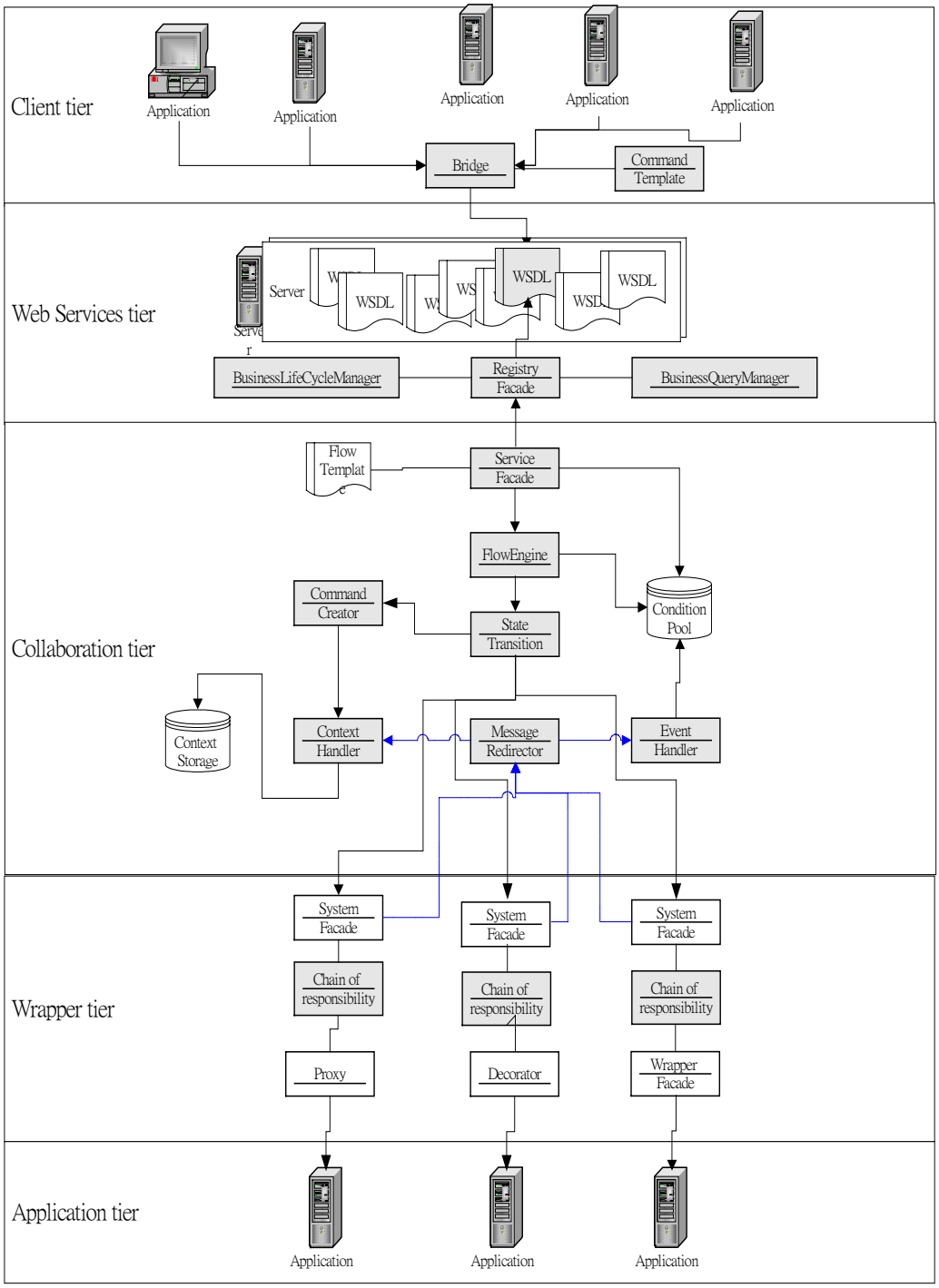


圖 3. 1 多階層架構圖

Application tier 存在不同平台(UNIX、Windows、LINIX)、不同架構(Distribution、Parallel、Central)、不同語言(C、Java、C++、Ada…) 的應用程式，這些應用程式與使用者使用者之間的關係可能是 B2B 或 B2C，這一層的主要結

構建立一個富有彈性、容易管理、且具有分享的機制。

這一章節從多階層式架構為出發點，分別說明各個階層的目的、階層中存在的角色及階層間的關係。透過這樣的說明可以了解階層獨立的必要性。以下分別為各階層的設計架構進行說明。

3.1 使用者層(Client Tier)

企業導向ｅ化，電子化應用軟體充斥，企業組織、企業夥伴間如何建立良好的應用軟體合作模式，是現今資訊產業所關切的。企業組織透過應用軟體直接進行溝通增加事件處理效率，免去繁雜的交通及電信費。

所以在本研究所定義的使用者層存在許多現有的應用軟體，這些應用軟體需要有技巧建立合作關係。以校園內部資訊系統為例，學生資訊管理系統、課程管理系統、成績管理系統，這些系統分別管理與學生相關的訊息，學生的個人資料、成績、修課情況，就目前的情況而言，學生資訊分別被不同的系統管理，所以學生資訊系統無法查閱學生修課情況，課程管理系統沒有學生完整的個人資料，基本上學生的所有資料應該是一份唯一的資訊，不應該重複出現在不同的系統當中，如此一來很容易造成資料不一至，維護不易等問題。

行動通訊時代的來臨，促使可攜式硬體設備漸普及，這樣的風潮帶動多型態硬體設備時代。這些使用者透過不同的瀏覽器連結使用者層上的應用程式，利用使用者層上的應用程式擷到適當的網路服務擷取資料。

歸納上述這些問題，以下針對使用者層存在的目的列舉具體目標。

使用者層之目的如下：

1. 提供使用者層應用程式與網路服務相關系統的獨立性：本論文以多階層軟體架構為論文主軸，所以使用者層繼承了多階層軟體架構的特性，就是階層間只透過單一介面進行資訊交換，使用者層應用程式不須考慮網路服務系統的運作機制，只要實作階層溝通介面就可以達成溝通。

2. 提供系統整合擴充性：使用者層存在了許多應用系統，這些應用系統需要建立合作機制，本論文獨立出使用者層就是為了讓系統整合能夠達到橫向擴充的機制，程式設計師很容易界能導入其它應用系統進行整合。

使用者層存在的角色如下：

1. 組織應用軟體：在這所指的組織包括公司、企業、學校、公家機關、..等等，就組織對組織的例子而言有相當多的情況，如企業內部系統間的整合，企業夥伴間的系統合作，公家機關內部的系統整合，國家政府機關間的系統整合，所以在使用者層存在的應用軟體小可以很小，大可以很大。

3.2 網路服務層(Web Services Tier)

網路服務簡而言之就是透過 HTTP 通訊協定夾帶 SOAP 封包傳遞訊息資料，SOAP 封包包裝了 XML 文件，XML 文件中描述資料訊息，所以透過這樣的方式可以達到資料交換的目的。

要達到系統整合，服務提供者必須利用一種以 XML 為基礎的公開規格撰寫服務項目，文件內部描述服務索取方式及網路位址等相關資訊，最後將這些資訊公佈在網路上。網路上有一些特殊的站台專門提供網路服務提供者公開它們的服務資訊，這些站台便成為資訊收集中心。資訊中心有規劃的儲存所有網路服務相關資訊，提供服務索取者搜尋，利用網路服務站上的資訊找到網路服務實體。透過這樣的方式應用系統可以在分散式的環境裡進行合作溝通。

所以在我們的設計當中獨立 Web Services Tier，在這個階層當中集合所有與網路服務相關的註冊資訊，基礎於開放的網路伺服器環境，企業透過開放的資訊找到適當的服務對象，服務提供者利用標準的註冊文件描述這些服務。

目前有一些標準規範描述註冊格式，各個軟體系統必須依循公定的註冊標準(published registry standards)如 ebXML[20] 、 WSDL[11] 、 UDDI[10] 、 RosettaNet[21]、Repository Standard 等等標準，來描述自己所提供的服務，如此一來其他的系統或使用者依據這些標準找到適合的網路服務(web service)。

歸納上述這些現狀，以下針對網路服務層存在的目的列舉具體目標。

網路服務層之目的如下：

1. 應用系統間的溝通橋樑：應用軟體透過正規的網路服務描述，說明應用軟體所提供的服務及相關資訊，並將這些文件公告在網路服務註冊中心，其它應用系統透過這些公告的資訊建立系統合作機制。這樣的模式，建立在企業間，足以增加企業商機，運作在企業組織內部，可以加快事件處理成效。

2. 分散式系統整合環境：本論文所提出的設計架構，並非集中管理應用程式間的相關資訊，也不是直接建立應用系統間的合作關係，而是保持系統原有的分散式環境，利用 XML 文件的描述動態的鏈結應用系統。

3. 資源釋放媒介：資源涵蓋的範圍很廣，可以包括有形的物體或是無形的環境，在這所指的資源包括資料、網路環境、應用程式、...等，凡是可被利用的都叫資源。透過網路服務可以很方便的釋放有利資源，如網路環境。高速網路環境並非人人皆有，但是許多研究學門如生物科技、基因演算等研究常常需要高速電腦輔助運算，網路服務層可以接收類似的網路服務描述，透過這樣的模式釋放有效資源。

網路服務層存在的角色如下：

1. 網路服務伺服器：整個網路世界裡存在許多被公認的網路服務伺服器，這些伺服器扮演著網路服務註冊站的角色，企業、組織、公司等團體都可以對網路服務註冊站存取網路服務相關資訊。

2. 網路服務描述文件：網路伺服器裡存在許多網路服務描述文件，這些文件依據一些標準的網路服務描述撰寫而成，這些文件由服務提供者公告，內容描述了服務的內容、通訊方式及位址等相關資訊。

3.3 合作層(Collaboration Tier)

一個網路服務由一至多個元件組合而成，這些元件來自不同的應用系統，所以合作層建構在包裝層和應用程式層之上，因為合作層負責鏈結整合來自不同應用程式的元件，建立元件合作關係。

一個網路服務包裝了多個不同的元件，對於使用者應用程式的事件存取，需要提供一個管理機制，也就是提供一個網路服務對外溝通窗口，透過這個溝通介面網路服務可以對外傳送訊息，透過這個介面網路服務可以接收外部應用程式的存取。

通常一個網路服務面對多種不同的使用者應用程式，如何建構網路服務內部元件合作機制才能滿足各種不同應用程式的要求？本論文在合作層提出一套動態建立元件合作機制，動態的組合元件合作模式，希望使網路服務內部元件運作更為彈性。

歸納上述這些問題，以下針對合作層存在的目的列舉具體目標。同時針對這一層存在的角色進行進一步的說明。

合作層之目的如下：

1. 動態建立元件合作機制：網路服務面對不同的外部存取，需要一種彈性的機制連結所有異質的元件，所以利用 XML 文件描述元件合作流程以及元件間訊息傳遞，這樣的方式不但可以達到跨平台、跨應用程式的效果，同時可以很迅速的建立修改元件合作關係。

2. 建立網路服務對外溝通窗口：一個網路服務結合多個應用元件，如果使用者的命令都直接與網路服務內部元件個別溝通，這樣的設計將會增加實作的困難度，所以透過單一的窗口接收所有網路服務事件可以化簡繁複的溝通模式。

合作層存在的角色如下：

1. 網路服務對外溝通窗口：提供網路服務單一存取窗口，這個窗口接收所有的外部事件，這樣的模式讓使用者應用程式不會直接存取網路服務系統程式。同時我們也可以藉由這個窗口控管使用者端應用程式事件。

2. 網路服務元件合作模組：網路服務外部事件觸發會引起網路服務內部元件一

連串的反應，與外部事件處理相關的元件工作流程是由一連串的狀態改變與事件驅動所組成，這些狀態的改變與網路服務內部事件的產生關係著不同的元件，所以我們必須利用一種模組勾畫出元件合作的流程，狀態改變的條件。在這個階段我們利用 XML 的特性解決異質平台元件溝通不易的困難，利用 XML 描述事件處理流程，利用 XML 描述以上所述種種，組成本研究所提出的網路服務元件合作模組。

3.4 包裝層(Wrapper Tier)

在本研究論文當中，網路服務基本上是一個層層相包的多層結構，一個網路服務底下包含了多個軟體元件，一個軟體元件分別來自不同的應用系統，系統底下又有多支應用程式。就本論文的架構而言，合作層提供網路服務對外的窗口，底下包含多個合作元件，這些合作元件分別具有不同的底層架構，所以在系統進行整合的同時必須考量舊有系統的屬性，為了使這些元件個別的特性不影響系統合作機制。

所以本設計架構提供一個封裝層，專門負責包裝異質系統，以系統元件化的方式為傳統系統提供系統溝通機制。否則，一個系統面對多重合作系統，複雜的存取會導致系統設計維護困難，所以良好的存取介面控管可以提昇系統維護性。

歸納上述這些問題，以下針對包裝層存在的目的列舉具體目標。同時針對這一層存在的角色進行進一步的說明。

包裝層之目的如下：

1. 建立元件對外溝通窗口：網路服務內部元件來自多個不同的應用系統，這些應用系統可能具有不同的特性，如執行於不同的平台、使用不同的程式語言、遵循不同的資料格式等等，這些環境因素造成系統合作不易。所以，以軟體元件化的方式，包裝這些應用程式，建立元件間的合作關係，如此一來才能達到互動。本論文在包裝層階段，對應用程式進行包裝，對外提供元件溝通介面，管理元件對內對外的訊息傳遞，網路服務透過元件拼裝達到整合，增

加系統整合的彈性。

2. 管理元件外部事件存取：一個軟體元件來自一至多個不同的應用系統，而且一個軟體元件可以接收多種不同的事件處發，如此多樣的情況增加管理事件處理者的困難度，如何管理這些外部存取事件，適當的分配有利的事件處理者？本研究提出一種事件分派機制，有效的分配事件給適當的處理者，降低了元件設計的複雜度。

3. 客制化元件功能：進行系統整合作適度的軟體重整在所難免，在包裝層希望能為舊有系統不足的地方進行客制化工作，也就是添加系統功能。其中包括導入代理人的機制加入物件導向應用程式功能，為軟體元件添增訊息重導機制，導入非物件導向式軟體功能等。

包裝層存在的角色如下：

1. 元件對外溝通窗口：一個元件一定有一個對外窗口，透過這個窗口接收外部資訊，也經由這個窗口傳遞資訊，這個溝通介面底下包裝所有元件內部相關應用程式。

3.5 應用層(Application Tier)

應用層存在許多現存的應用軟體，這些軟體有被整合的需求，但是這些軟體可能使用不同的語言、架構、技術開發而成，同時他們可能運作在不同的平台，使用不同的網路結構，單就這些系統差異性，我們希望在不要修改系統的前提下，企業與企業或個人與企業間的系統能夠達成溝通的效果，所以本研究將這類型的應用軟體獨立出來，置放在應用層當中，希望能夠提供網路服務的擴充性。

歸納上述這些問題，以下針對應用層存在的目的列舉具體目標。同時針對這一層存在的角色進行進一步的說明。

應用層之目的如下：

1. 提供網路服務擴充機制：獨立應用層，劃分了網路服務與應用程式之間的關係，可以幫助現有系統快速導入網路服務，因此而達到網路服務擴充性。

應用層存在的角色如下：

1. 現有應用系統：這些應用系統包括物件導向式應用系統、非物件導向式應用系統、分散式應用系統等不同性質的系統。它們彼此之間具有整合的需要，所以可以被置放在本研究所提出的多階層架構中的應用層。

第四章 設計樣板與階層功能的映射關係

第三章明確定義多階層架構中各階層的目的，針對這些設計目的，連帶產生許多設計問題，針對這些設計問題本章節以設計樣版為基礎，勾畫系統設計藍圖。透過樣板架構分別解決各階層問題，利用樣板合作[24]達成多階層式軟體架構設計。同時本章節並描述與各階層設計相關的現有技術。以下分別為各階層的設計架構進行說明。

4.1 使用者層(Client Tier)

使用者層存在許多待整合的應用軟體，這些應用軟體彼此之間可能具有相當的異質性，如何建立這些異質軟體的溝通模式？本研究提出一套跨平台、跨語言的解決方案，幫助現有系統進行系統整合。

為提供系統整合的擴充性，本論文建議利用 Gamma et. al.[12]在 1994 年提出設計樣板(Design Patterns)中的 Command Pattern 架構，建立跨平台的資料訊息。

表 4.1.1 列出使用者層面臨的問題及簡述問題解決方法。依據「如何化簡系統整合機制?」這樣的問題，使用者層提出兩項解決模組，分別是訊息格式模組、

表 4.1.1 使用者層解決的問題及解決方法

問題	解決機制
如何化簡系統整合機制?	套用 Bridge Pattern 架構建立一對多的應用軟體整合擴充機制，利用 XML 文件作為訊息傳遞的媒介，解決異質平台的溝通問題。

溝通模組，以下針對這些模組進行說明。

在這個階段，本論文對使用者層的應用程式提出一種較好的設計架構，原因是，不同的硬體平台需要不同的資訊展現方式，對於應用程式的設計，本論文建議使用 MVC 架構，此架構將應用程式中的 Model、View 和 Controller 劃分開來，

如圖 4.1.1.1 所示，Model 描述商業邏輯(business logic)，View 負責描述所有的展現邏輯(display logic)，Controller 負責接收控管從 View 產生的指令(Command)，在這個架構當中我們多為它外加一個管理介面(Manage Interface)，因為一個指令(Command)可能牽涉多個 Model，管理介面(Manage Interface)負責對指令進行分類指派，將請求資訊(request Information)傳遞給適當的 Model，所以管理介面(manage interface)擔任事件管理者的角色。藉由對資訊呈現、控制管理和商業邏輯的劃分，設計者可以很彈性的置換資訊表現的方式。

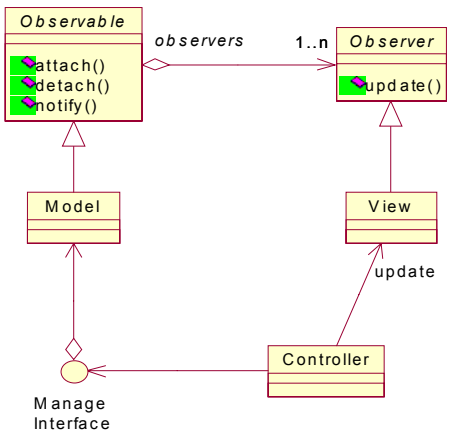


圖 4.1.1. 1 MVC 架構圖

單以 MVC 架構建製同步性高的應用程式略顯不足，所以附加在 MVC 架構之上的，為達到 View 與 Model 之間的關聯性，本論文建議使用 Observer Pattern[12] 架構，Observer Pattern 定義出一對多的相依關係，Observable 中包含了 attach、detach、notify 等方法，Observable 透過 attach 和 detach 控管 Observer，應用程式中的 Model 繼承 Observable，View 繼承 Observer，當 Controller 改變了相關的 Model 之後，Model 會通知與這個 Model 相關的 View，以這樣的機制便可以達到同步的效果，如圖 4.1.1.1 所示。

以上說明開發應用程式的一個技巧，是就開發的角度談論系統架構設計，但在使用者層裡存在的是等待被整合的應用軟體，這些應用軟體需要被統一的就是訊息溝通方式，本論文依據這個問題提供一套溝通模組，產生跨平台的訊息物

件，建立應用軟體合作模式。

4.1.1 溝通模組(Communication Module)

系統溝通包括本地端指令存取和遠端呼叫，異質平台溝通存取在所難免，以物件的方式進行溝通可以降低平台環境的相依性，所以將指令(command)封裝成物件可以達到系統獨立性(system independent)，本論文使用 Command Pattern[12]，利用圖 4.1.1.2 中的 Wrapper 對訊息進行封裝，UnWrapper 幫助訊息解封裝，將溝通資訊包裝成物件，降低資訊對系統平台的相依性。

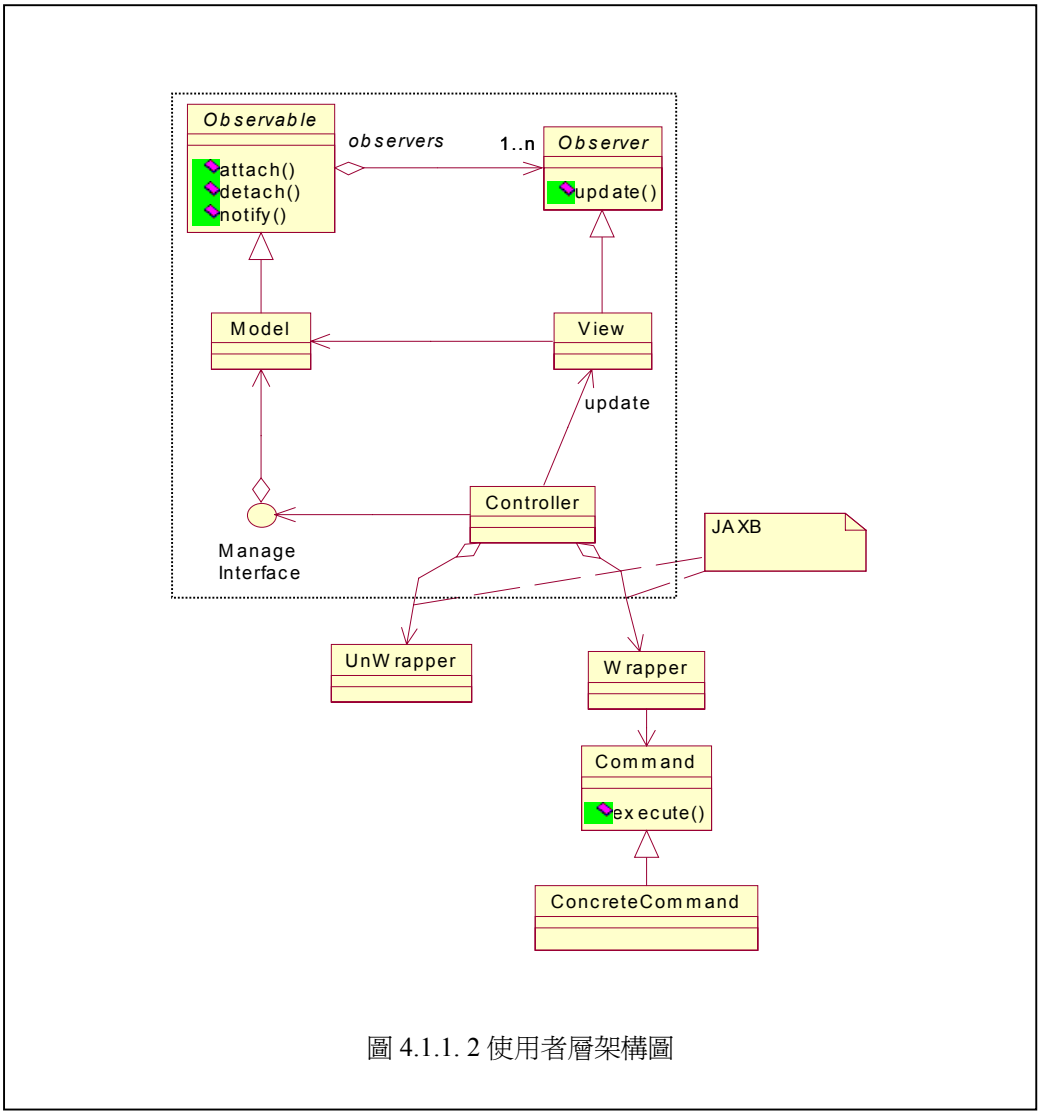


圖 4.1.1. 2 使用者層架構圖

在實作方面，本地端應用程式與遠端應用程式，可以利用傳遞 XML 檔案的方式進行溝通，在使用者層上的應用程式，考量硬體平台的獨立性，所以利用傳遞 XML 物件的方式傳遞資訊，在系統實作方面建議利用 SUN 提出的網路套件

The Java API for XML Messaging (JAXM) 提供一種從 Java 平台透過 HTTP 通訊協定傳送 XML 文件的標準方法，這套 API 以 SOAP1.1 為基礎，並且可以運作在較高階的訊息通訊協定(messaging protocols)如 ebXML。Java 物件與 XML file 之間的轉換動作可以藉由 SUN 提出的 API JAXB(Java Architecture for XML Binding)幫我們完成。

4.1.2 訊息格式模組 (Command Format Module)

應用程式間訊息的溝通，需要先取得對方應用程式的接收格式，如此一來才能達到溝通的目的。舉例來說，校園資訊系統的整合，課務系統與教學系統間的互動，課務系統必須取得取得教學系統應下的參數格式，這樣才能順利取得資訊。

在網路服務的整合環境當中，網路服務利用網路服務描述語言(WSDL)描述服務通訊格式，使用者層中的應用軟體只要取得通訊格式就可以與網路服務進行溝通。

所以在使用者層，本研究設計了訊息格式擷取模組，擷取 WSDL 文件中描述的網路服務溝通方式，透過這些模組使用者應用程式可以建立合作關係。詳細的設計情形在第五章有詳細的介紹。

4.2 網路服務層(Web Service Tier)

網路服務層提供一個用來儲存網路服務資訊(WSDL 文件)的共享空間。WSDL 文件如圖 4.2.1 所示，圖 4.2.1 表示的 WSDL 文件描述一個課程資訊網路服務，這個網路服務可以接收選課的請求，選課需要傳遞學號、密碼、課程代號等參數，系統會回覆選課結果及成敗原因，同時 WSDL 文件描述使用者與網路服務溝通的方式 URL、port 等等資訊。

```
<?xml version="1.0"?>
<definitions name="ClassesService"

targetNamespace="http://www.thu.edu.tw/ClassesWervice.wsdl"
xmlns:tns="http://www.thu.edu.tw/ClassesWervice.wsdl"
xmlns:xsd1="http://www.thu.edu.tw/ClassesWervice.xsd"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns="http://schemas.xmlsoap.org/wsdl/">
<types>
<schema targetNamespace="http://www.thu.edu.tw/ClassesWervice.xs
    xmlns="http://www.w3.org/2000/10/XMLSchema">
     <element name="ChoosingClassesRequest">
       <complexType>
         <all>
          <element name="parameter">
             <element name="studentId" type="string"/>
            <element name="passwd" type="string"/>
          <element name="classesId" type="string"/>
          </element>
         </all>
       </complexType>
     </element>

     <element name="ResponseInfo">
       <complexType>
          <all>
           <element name="parameter">
           <element name="state" type="String"/>
<element name="info" type="String"/>
             </element>
          </all>
       </complexType>
      </element>

   </schema>
 </types>

  <message name="ChoosingClasses">
 <part name="body" element="xsd1:ChoosingClassesRequest"/>
  </message>

  <message name="ChoosingClassesOutput">
    <part name="body" element="xsd1:ResponseInfo"/>
  </message>

……..
```

圖 4.2. 1網路服務定義(ClassesService.wsdl)範例

因為網路服務層存在了許多這類型的文件，以便提供使用者查閱，建立系統溝通模式。所以本文章的網路服務層提供一個網路服務資訊管理機制，其架構圖如圖 4.2.2 所示。這個機制可以幫助程式開發者管理網路服務公告資訊，包括新增功能，刪除功能等。同時提供使用者請求服務機制，使用者應用程式可以透過分類搜尋或是單一服務搜尋找到適當的服務。

在實作方面，本論文使用 SUN 提出的 API JAXR(Java Architecture for XML Registry)幫我們完成網路服務註冊、新增資料、刪除資料等行為，同時也提供使用者應用程式 query 的窗口，讓使用者可以很方便的找到適當的網路服務。

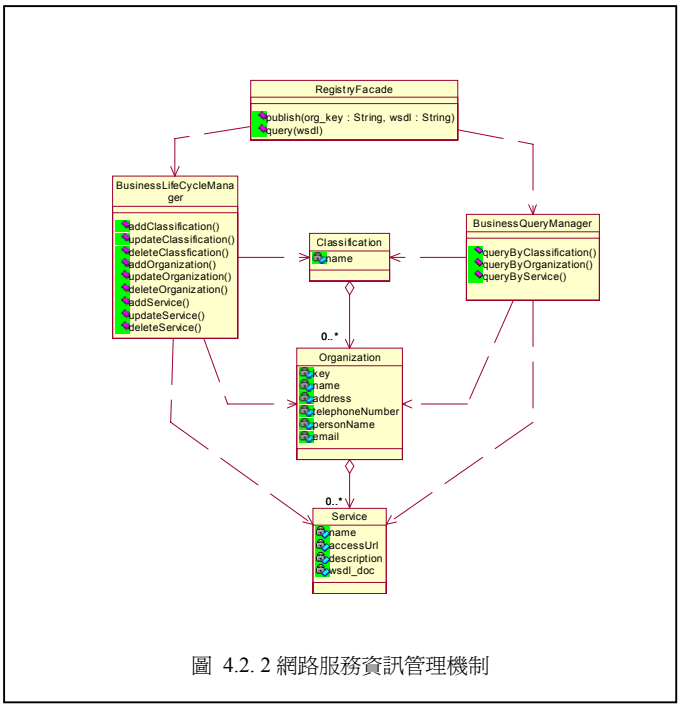


圖 4.2. 2 網路服務資訊管理機制

4.3 合作層(Collaboration Tier)

本論文的多階層軟體架構提出三種不同複雜度的合作機制，這些合作模式是為的建立網路服務內部元件合作關係而建製的，分別為單向溝通模式，雙向溝通合作模式和進階雙向程序型模式。以下針對這些合作模式作進一步詳細的介紹。

4.3.1 單向溝通模式

這種設計模式使用 Façade Pattern[12]架構，提供單一的對外介面(unified interface)，是一個架構於網路服務(Web Service)之上的介面(high-level interface)，封裝服務中所有的應用元件，避免外界軟體、外界使用者(users)直接對網路服務(Web Services)做存取。

Façade interface 對指令進行分類，所以面對單純的系統合作機制，可以透過介面的指令分類，將指令傳遞給適當的服務提供元件，這樣的架構常被利用於網

路服務中的應用系統少有合作機制的例子之上，使用者與網路服務間存在較單純的溝通模式時，如圖 4.3.1.1 所示。

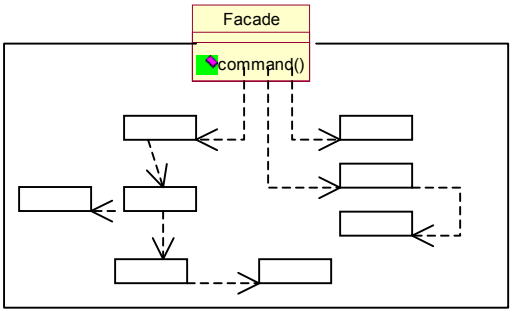


圖 4.3.1.1 Façade interface 架構

4.3.2 雙向溝通合作模式

有別上一階段的設計，此設計架構使用了 Façade Pattern[12]、Chain of Responsibility Pattern[12]、Mediator Pattern[12]、Observer Pattern[12]，提供解決較複雜的事件處理，同時建立系統與系統間的合作機制，使用者與網路服務間達成雙向的溝通模式。

就網路服務接收使用者指令的角度來看這個階段的設計架構，Façade Pattern[12]提供單一的對外介面(unified interface)，架構於網路服務(Web Service)之上的介面(high-level interface)，封裝服務中所有的應用元件，使用者透過 Façade interface 傳遞事件。Chain of Responsibility Pattern[12]提供一個網路服務能夠接收的事件鏈結，事件鏈中的每一個節點代表一種事件處理代理人，每一個事件處理代理人會連結到支援處理事件的元件。圖 4.3.2.1 表示 Façade Interface 與 Event

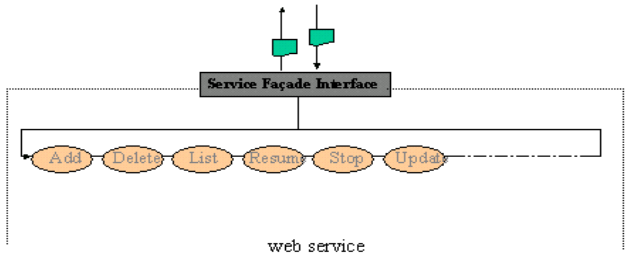


圖 4.3.2. 1 façade Interface 與 Event Chain 之間的關係概念圖

Chain 之間的關係。

由 Façade interface 接收進來的事件會進入事件鏈當中，透過對事件鏈中各節點的走訪，連結適當的服務提供元件，圖 4.3.2.2 描述的 System Façade interface、Service Façade interface 和 Event Chain 之間的架構關係類別圖。

有別於單向溝通模式使用 Chain of Responsibility Pattern 的目的在於解決網路服務複雜的事件觸發，利用 Event Chain 串聯所有事件，以責任劃分的方式，每一個節點負責處理系統服務中的一個項目，只要增加 Event Chain 中的事件節點就可以達成網路服務擴充性。

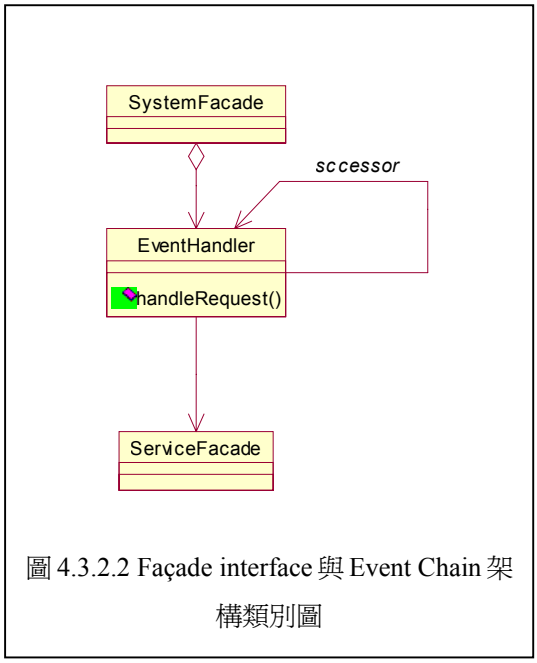


圖 4.3.2.2 Façade interface 與 Event Chain 架構類別圖

就網路服務內部元件回覆使用者應用程式的角度看這個階段的系統設計，如圖 4.3.2.3 所示，雙向溝通合作模式提供一個共享空間(Event Pool)，接收網路服務內部元件產生的事件，網路服務中所有的內部元件都可以存取這個共享空間，內部元件透過這個共享空間進行資料交換，這樣的機制簡化了系統與系統間複雜的存取關係。

如何得知共享空間內是否有事件產生？本模組套用 Observer Pattern 架構，Observer 負責監控網路服務內部共享空間，監控事件產生情況，當事件產生時 Observer 便會通知 Mediator，Mediator 在這個模組當中是扮演一個控制單元的角

色，負責指派示件處理元件，這樣的設計來自於 Mediator Pattern 架構。

再者，系統設計的同時，必須擷取系統與系統間具唯一性的資訊(context)，如會員資訊，這些資訊必須獨立於系統之外，如此一來才能達到資料同步，本設計將這些資訊交由 Mediator 統一管理，由 Mediator 判斷決定唯一性資料(context)的存取。

圖 4.3.2.3 描述 EventPool 繼承 Observable ，Mediator 繼承 Observer，當 EventPool 中有事件產生時，Mediator 便會收到 Observer 的通知，Mediator 掌管所有元件，Mediator 集結很多的 Colleague，每一個系統都繼承 Colleague，所以 Observer 通知 Mediator 有事件發生，Mediator 便可以管理支配事件給適當的系統。

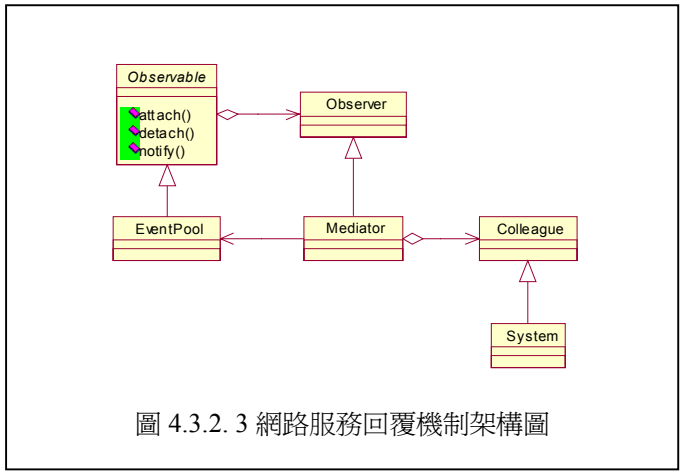


圖 4.3.2. 3 網路服務回覆機制架構圖

4.3.3 進階雙向程序型模式

雖然一個軟體元件可能提供多項服務，但是有些服務需要透過多個系統間的協同合作，形成一個工作流程以提供一個整合性的服務，例如一個電子商務的服務將包含商品選購、線上付款、貨品包裝及配送等服務，如圖 4.3.3.1 所示。雖然個別的服務都可以由個別的軟體元件提供，但是要完成一個購物流程則必須整合各個軟體元件，形成一個工作流程。

以購物流程的例子而言，一樣是線上購物卻可能因為不同的情況而有不同的工作流程，例如使用者在購物完成後可以選擇不同的付款方式如信用卡付款，或 ATM 的付款方式，如圖 4.3.3.2 所示。為解決此問題的方法即提供動態的工作流

程建立,彈性的定義工作流程對應的軟體元件,動態的建立它們彼此之間的關係。

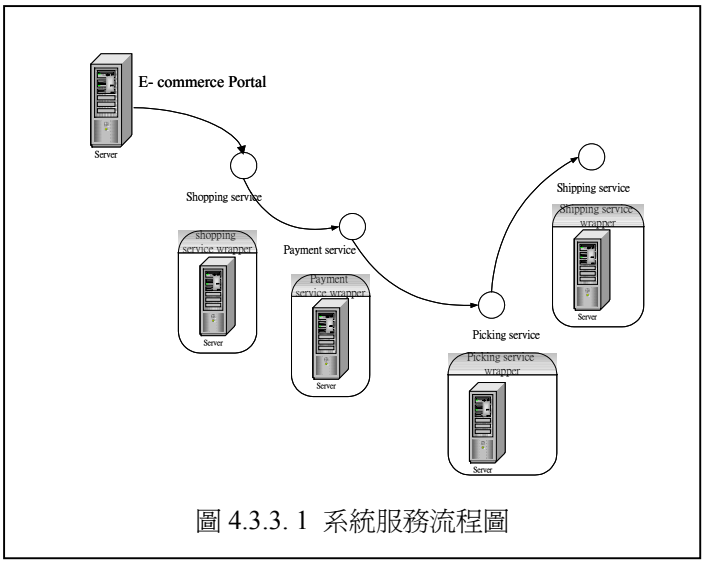


圖 4.3.3. 1 系統服務流程圖

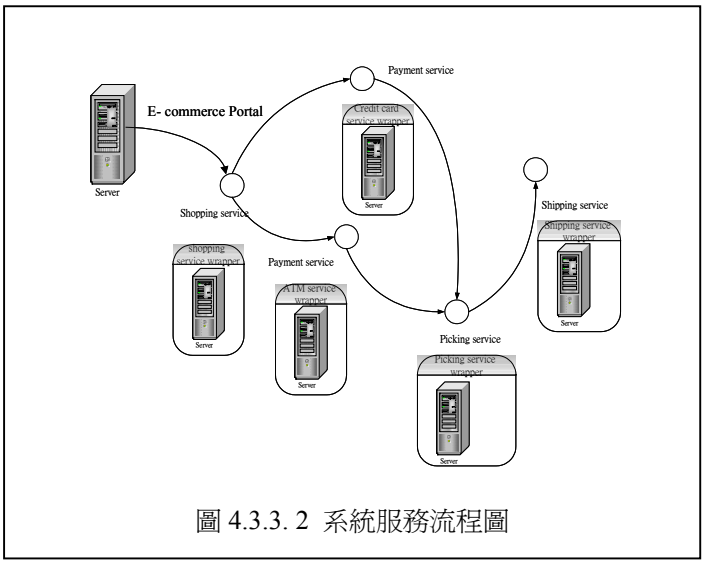


圖 4.3.3. 2 系統服務流程圖

為了建立動態流程的機制，本論文提出了 Service Flow[22]之設計樣板。該設計樣板之結構如圖 4.3.3.3 所示。

在圖 4.3.3.3 中，Service Flow 中各個服務的執行時機 將根據個別的 EventId，一個 EventId 將對應至一個 Condition，由多個 Condition 集合而成的物件即 ConditionSet，而一個 ConditionSet 則對應至一個 StateTransition 物件，該物件封裝了對應的服務及其呼叫的方法(method)。換言之，當服務被執行後將產生一個新的 EventId，其對應的 Condition 將產生新的 ConditionSet，此時其對應的 StateTransition 也將被取出並執行。隨著 StateTransition 的取出與呼叫執行，各個

服務也將逐一執行，而形成一個工作流程。圖 4.3.3.4 為 StateTransition 取出並執行的過程。

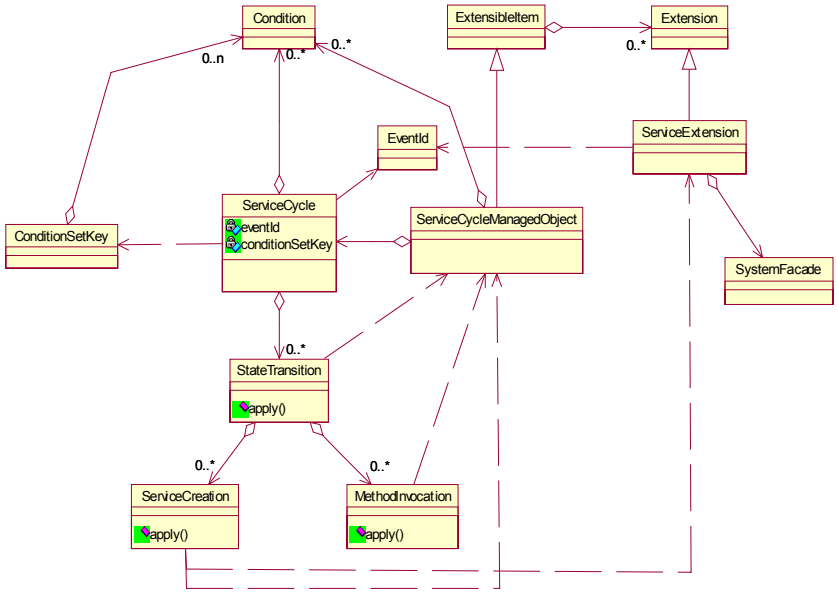


圖 4.3.3. 3 Service Flow 設計樣板結構

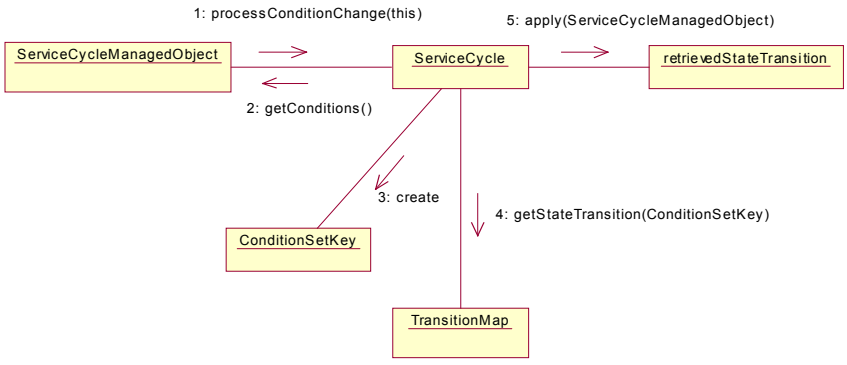


圖 4.3.3. 4 合作流程圖

本論文的合作模式提出這三種不同複雜度的架構設計，程式設計師可以針對整合系統的溝通模式及運作機制判定系統合作機制，以下列出系統合作機制判定要點：

1. 溝通模式：判定系統與使用者之間的溝通方式是以單向或雙向。

2. 合作關係：判定事件驅動為程序性合作關係或非程序性合作關係，如工作流程(workflow)事件，這一類型的事件具有一連串相關程序，這樣的程序往往需要多個元件共同合作完成；行事曆排成事件便不具程序關連性，這類型的事件只與相關應用系統產生互動。所以程式設計師可以依據這些特點選擇適當的元件合作模式。

3. 判定元件間共同資訊：判定舊有系統間是否存在具唯一性且有重複性的資訊。

面對這麼多種模式，現今已經存在了許多技術幫助程式設計師進行系統合作機制，包括 JMS(Java Message Service)[25]、JNDI(Java Naming Directory and Directory Interface)[26]。

4.4 包裝層(Wrapper Tier)

合作層負責建立軟體元件間的合作關係，這些軟體元件是透過包裝層包裝過後的成品，在軟體包裝的過程當中會面臨相當多的問題，包括客制化元件功能、元件內部事件處理以及元件對外溝通模式等。在這個階段本研究針對這些問題，提供相關的架構設計。表 4.4.1 列出為達到包裝層的目的所面的的一些問題，並對這些問題的解決方式提出簡單的概略性描述。

表 4.4.1 包裝層預解決的問題及解決方法

問題	解決機制
如何建立軟體元件?	利用 Façade Pattern 架構建立包裝軟體元件，使軟體元件透過單一窗口接收外部事件，件導向式應用程式，及非物件導向式應用程式單一對外溝通窗口。
如何管理元件內部事件處理?	套用 Chain of Responsibility Pattern 架構，利用責任分配的方式，鏈結所有事件處理代理人，每一個事件代理人連結到真實軟體元件。
如何客制化元件功能?	套用 Proxy Pattern、Decorator Pattern、Message Redirect Pattern、Wrapper Façade Pattern 等架構，客制化軟體元件。

合作層階段站在整合軟體元件的基礎上，為元件建立合作機制，所以在這個階段，包裝層對軟體元件化提出的幾種方法，以解決表 4.4.1 所列出的問題。以下將對各不同的問題之解決方法與設計做更詳細的介紹。

這個階段利用了 Gamma et. al.在 1994 年提出設計樣板(Design Patterns)中的 Façade Pattern[12]、Chain of Responsibility Pattern[12]、Proxy Pattern[12]、Decorator Pattern[12]、Message Redirect Pattern[12]架構和 Douglas Schmidt 在 1999 年提出的設計樣版 Wrapper Façade Pattern[23]架構。透過這些架構勾畫包裝層對元件包裝、元件內部事件處理機制和客制化元件等功能的藍圖，圖 4.4.1.1 說明上述樣板在包裝層所扮演的角色。以下分別對元件包裝、元件內部事件處理機制和客制化元件功能等細項進行進一步的說明。

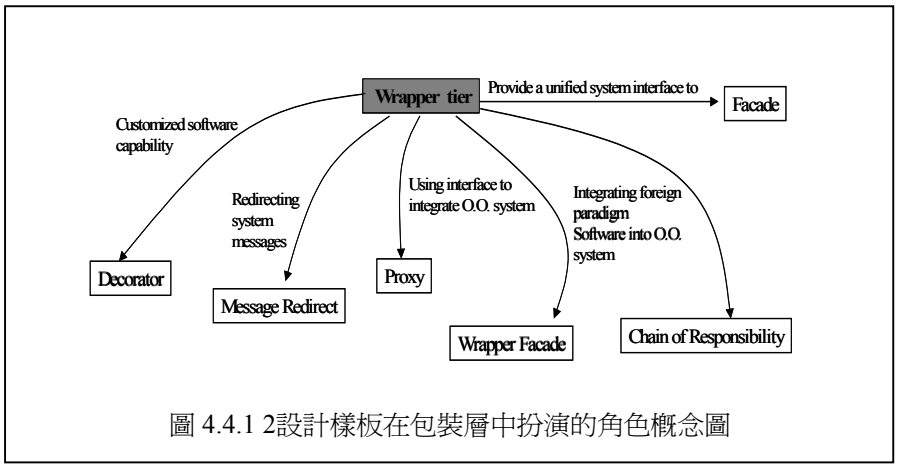


圖 4.4.1 2設計樣板在包裝層中扮演的角色概念圖

4.4.1 元件包裝

一個軟體元件必定具備對外溝通介面，這是構成一個元件最基本的東西。所以本研究架構在包裝層提供元件包裝機制，利用 Façade Pattern 架構包裝現有軟體系統，為軟體元件提供 façade interface，軟體元件透過這個溝通界面管理元件外部存取事件，軟體元件透過 façade interface 對外發送訊息事件。

這樣的模式可以防止外部使用者，任意存取元件中的應用程式，同時 façade interface 可以有效的管理所有元件對內或對外的事件。

4.4.2 元件內部事件處理機制

一個元件能夠接收多種事件的處發，一個元件可以透過不同的方式組成，所以不同的事件觸發可能關連不同的真實系統，良好的事件管理機制便顯得相當重要。

本研究利用責任分配的方式規劃事件處理代理人，每一個事件代理人代表一個事件處理，每一個事件代理人連結到真實的事件處理系統，把所有的事件處理代理人連結起來便形成事件鏈(Event Chain)，這個觀念是來自於 Chain of Responsibility Pattern 架構。

如圖 4.4.3.1 所示，軟體元件的 Façade interface 接收元件外部事件，這些外部事件會流入事件鏈當中，當外部事件對應到相同的事件處理代理人時，事件處理代理人便將事件資訊交由真實的處理者進行處理。透過這樣的方式化簡元件內部事件處理的複雜度。

4.4.3 客制化元件功能

上述的軟體元件包裝和元件內部事件處理機制，勾畫了軟體元件的藍圖，一個軟體元件的內容需要在元件客制化階段建製。所以本研究套用 Gamma et. al. 在 1994 年提出設計樣板(Design Patterns)中的 Proxy Pattern、Decorator Pattern、Message Redirect Pattern 和 Douglas Schmidt 在 1999 年提出的設計樣版 Wrapper Façade Pattern[23]架構。圖 4.4.3.1 說明包裝層中樣板設計間的合作關係概念圖。以下分別就各個 Pattern 的功能作進一步的說明。

Proxy Pattern 此設計架構為實際系統提供外部存取介面，此介面扮演系統代理人的角色，外部事件可以透過此介面與系統實體進行溝通。將這樣的系統代理人放入軟體元件當中，幫助系統導入其它物件導向式系統功能。

Decorator Pattern 幫助軟體元件添增新功能，有見企業軟體進行系統整合時，現有系統進行客制化在所難免；也就是說程式設計師需要對軟體元件適度的添增功能，在這個階段的設計我們利用 Decorator Pattern[6]彈性的附加元件功能。

Message Redirect 在系統整合時軟體元件往往需要取得其他元件的服務、通知其他的元件進行另一個服務或驅動合作層的流程運作。但由於元件本身在設計

時並未考慮對外發送訊息，因在本論文提出了一個事件重導機制各個軟體元件透過 MessageRedirector 對外發送訊息。

WrapperFacade Pattern 對程序式系統進行包裝，引用非物件導向的系統中的函式(function)和資料(data)，軟體元件導入程序式應用程式，對於處理這樣的事件現今 sun 提出 Java Native Interface (JNI)API 幫助 Java 程式開發者有效的導入非物件導向式應用程式。

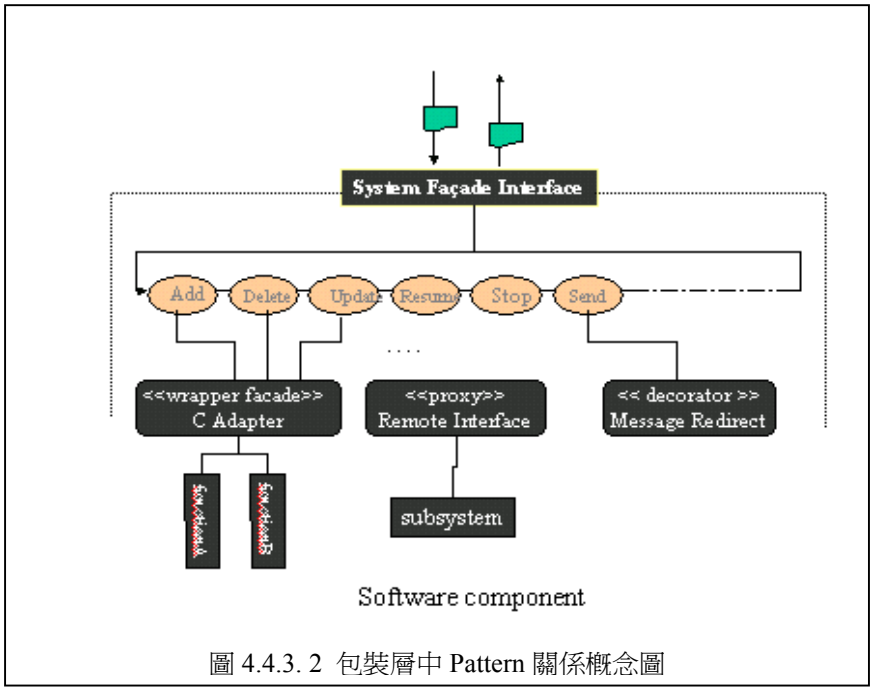


圖 4.4.3. 2 包裝層中 Pattern 關係概念圖

4.5 應用層(Application Tier)

這一層存在許多現存的應用軟體，這些軟體可能使用不同的語言、架構、技術開發而成，同時他們可能運作在不同的平台，使用不同的網路結構，單就這些系統差異性，我們希望在不要修改系統的前提下，企業與企業或個人與企業間的系統能夠達成溝通的效果，所以有獨立這一層的必要，因為我們不希望其他的修改影響原有的系統，希望達成 plug and play 效果，元件化現有系統，我們只要為這個元件提供一個與外界的溝通介面，如此一來此元件可以很容易的與其他系統進行溝通。

在這個章節的最後部份，總結上述各階層所建議使用的設計樣版，以圖形關

係表示設計樣版之間的關係，如圖 4.5 所示。

由 Client Tier 觀之，使用者應用程式架構於 MVC 架構之上，利用 Observer 建立 Model 與 View 之間的監控關係，Command 包裝訊息物件，交由 Controller 對外發送訊息。在 Collaboration Tier，本架構利用 Façade 提供網路服務對外溝通窗口，依據需求選取 Chain of Responsibility 提供系統擴充機制，或是 Service Flow 處理流程式事件，Chain of Responsibility 必須配合 Mediator 加上 Observer，Observer 監控事件發生情況，再將事件資訊交給 Mediator，Mediator 負責支配事件，交給適當的事件處理單元。在 Wrapper Tier，負責封裝軟體元件，所以利用 Façade 提供元件對外溝通介面，套用 Chain of Responsibility Pattern 管理元件內部事件，Wrapper Façade 幫助元件整合程序式應用程式，以物件導向式系統觀之，Proxy 提供系統代理元件，讓使用者間接存取真實系統，Decorator 提供客制化現有系統機制，Message Redirect 提供元件訊息重導機制。表 4.5 列出本架構建議使用的各個樣版，並且對樣版使用目的作簡要的說明。

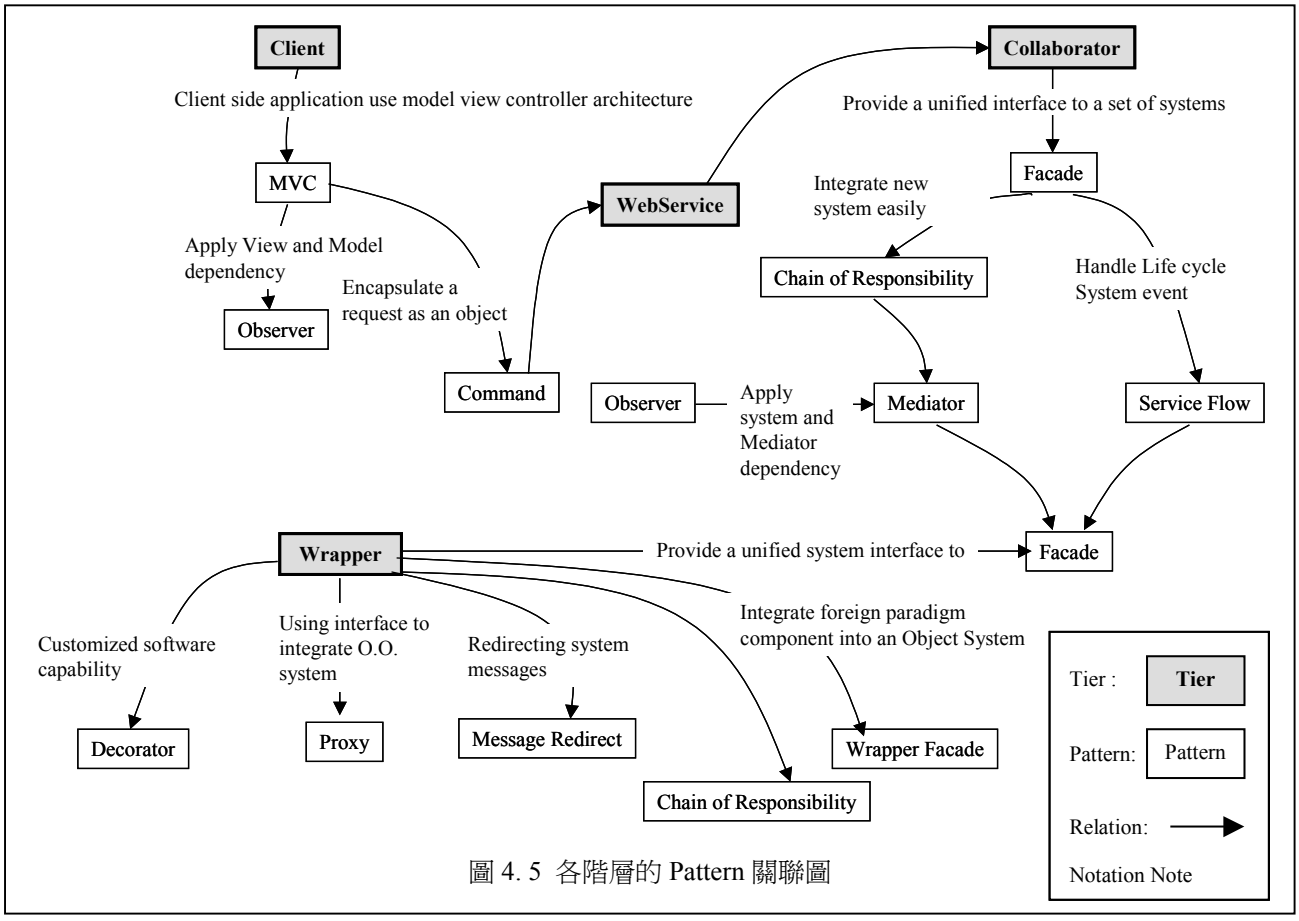


圖 4. 5 各階層的 Pattern 關聯圖

設計樣版名稱	設計問題	解決方法
Client		
MVC	不同介面大小的硬體設備需要不同的資訊呈現方式	套用 MVC Pattern 的觀念將應用程式的 Model、View、Controller 分開，增加設計的彈性。
Observer	應用程式設計的 Model 和 View 之間的關連性建立，當 Model 被改變 View 如何得知？	將 Model 和 View 套用 Oberver Pattern 架構，建立 Model 和 View 的關連性，提供即時機制
Command	異質系統間的訊息格式問題	套用 Command Pattern 架構將訊息包裝成 XML 物件，利用 XML 的跨平台特性，解決系統命令與系統的相依性。
Collaboration		
Facade	使用者應用程式與網路服務應用程式之間的溝通複雜，如何減低複雜度？	提供網路服務應用程式單一對外窗口，所有的訊息命令都透過這個窗口發送與接收。
Chain of Responsibility	網路服務系統接收的事件相當多元，如何有效的管理這些事件？	套用 Chain of Respnsobility Pattern，利用責任分配的方式，每一個節點代表一個處理事件，每個節點對應到事件處理者，鏈結所有事件節點。事件產生後便流入事件鏈當中，找尋適當節點並對應到適當的事件處理者。
Mediator	網路服務外部事件存取複雜，所以網路服務需要適當的事件處理指派網路服務元件內部合作機制。	網路服務內部元件間的合作，是透過事件指派者進行分配，這個事件指派者繼承 Mediator Pattern 架構觀念，建立元件間的合作機制。
Observer	網路服務內部元件會將元件對外的訊息暫存在網路服務中的共享空間，所以必須要有一個監控者隨時監控這個共享空間的狀態。	網路服務內部監控者繼承了 Observer Pattern 的設計架構，所以可以監控網路服務內部系統彼此溝通情況。
Service Flow	網路服務中的某些事件需要透過網路服務內部元件的合作才能達成目的，然而這樣的服務通常是屬於流程式服務，所以需要一種流程式事件處理機制	利用 Service Flow Pattern 得架構設計流程式事件處理機制。
Wrapper		
Facade	網路服務內部的軟體元件間有複雜的合作關係，如何減低複雜度？	提供軟體元件單一對外窗口，所有的訊息命令都透過這個窗口發送與接收。
Message Redirect	網路服務中的軟體元件通常不具備對外發送訊息的功能，所以需要有一種機制幫助軟體元件對外發送訊息，這樣一來才能達到元件合作的機制。	利用 Message Redirect Pattern 幫助程式設計師將原本運作於軟體元件內部的訊息，重新導正方向，使元件具有對外溝通的功能。
Proxy	網路服務中分散式系統或是保護性系統的溝通機制	利用遠端存取的方式建立分散式系統對外溝通介面，或是利用代理人的方式管理系統外部對系統的存取。
Decorator	當舊有系統功能無法滿足需求，往往需要添增新功能	套用 Decorator Pattern 架構，為軟體元件建立客制化功能。
Wrapper Facade	物件導向式系統與非物件導向式系統的整合	套用 Wrapper Façade Pattern 的架構整合非物件導向式系統。

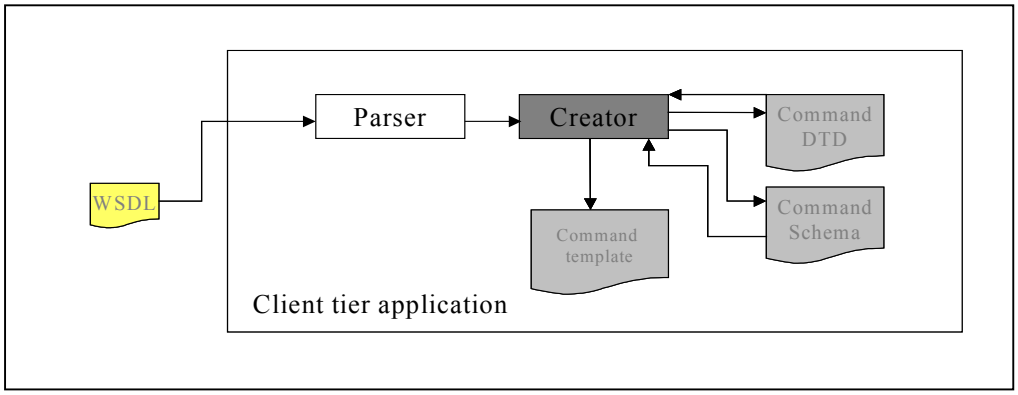
表 4. 5 多階層樣板列表

第五章 多階層架構實作設計

第五章介紹本論文提出的多階層架構之實作設計，具體指出各階層的設計架構及程式設計方法。以下分別為各階層進行說明。

5.1 使用者層(Client Tier)

下圖描述使用者應用程式擷取網路服務溝通模式流程概念模組，使用者應用程式中包含有 Parser、Creator 等元件，Parser 對於使用者應用程式所擷取到的網路服務規格(WSDL file)進行分析，再透過 Creator 建立網路服務命令模版(Command Template)，此命令模版式經由 Command DTD 借助 SUN 所提供的 JAXB 將 XML DTD 編譯成 Java 類別，使用者應用程式實作這些類別便可以產生網路服務可以接收的命令物件。



應用程式的初始化會連結網路服務註冊中心，擷取網路服務溝通模式，這些資料來自網路服務 WSDL 文件中的訊息傳遞格式，如圖 5.1.1 所示，圖中的虛線方框表示訊息溝通模式，圖 5.1.1 是描述一個選課的網路服務，這個服務可以接收兩種訊息(ChoosingClasses、ChoosingClassesOutput)，選課(ChoosingClasses)時必須傳入學號、密碼和課程代號等資訊，利用這些資訊可以建構溝通訊息模板(Command Template)如圖 5.1.2 所示，應用程式套用這些模板產生網路服務能夠接收的 SOAP 封包如圖 5.1.3 所示。訊息溝通的程序可分為兩種一為接收、二為傳送，應用程式透過溝通模組中的對外溝通介面(Facade)接收外部訊息，這些 XML 文件透過溝通事件模組中的解包裝元件(UnWrapper)將訊息轉換成系統參

數。反之，應用程式利用溝通事件模組中的包裝元件(Wrapper)將系統資訊包裝成 XML 文件，交由溝通模組中的對外溝通介面(Facade)傳送給網路服務，如圖 5.1.4 所示。

```
<?xml version="1.0"?>
<definitions name="ClassesService"

targetNamespace="http://www.thu.edu.tw/ClassesWervice.wsdl"
xmlns:tns="http://www.thu.edu.tw/ClassesWervice.wsdl"
xmlns:xsd1="http://www.thu.edu.tw/ClassesWervice.xsd"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns="http://schemas.xmlsoap.org/wsdl/">
<types>
<schema targetNamespace="http://www.thu.edu.tw/ClassesWervice.xs
    xmlns="http://www.w3.org/2000/10/XMLSchema">
      <element name="ChoosingClassesRequest">
        <complexType>
          <all>
           <element name="parameter">
              <element name="studentId" type="string"/>
             <element name="passwd" type="string"/>
           <element name="classesId" type="string"/>
           </element>
          </all>
        </complexType>
      </element>

      <element name="ResponseInfo">
        <complexType>
          <all>
           <element name="parameter">
           <element name="state" type="String"/>
<element name="info" type="String"/>
             </element>
          </all>
        </complexType>
       </element>

    </schema>
 </types>

   <message name="ChoosingClasses">
  <part name="body" element="xsd1:ChoosingClassesRequest"/>
   </message>

   <message name="ChoosingClassesOutput">
     <part name="body" element="xsd1:ResponseInfo"/>
   </message>

……..
```

圖 5.1. 1 網路服務定義(ClassesService.wsdl)範例

根據使者層的運作流程，為達到系統整合的擴充性，本論文提出訊息格式模組和溝通模組，訊息格式模組幫助使用者應用程式取得網路服務溝通格式，透過這些格式，使用者應用程式可以與網路服務達成溝通，進而與使用者應用程式達成整合。所以透過這兩種模式可以軟體提供整合擴充性。以下分別對這些模組進行設計架構介紹。

```
<!ELEMENT m:ChoosingClasses (parameter)>
<!ELEMENT parameter (name,type,value)* >
<!ELEMENT name (#PCDATA) >
<!ELEMENT type (#PCDATA) >
<!ELEMENT value (#PCDATA) >
<!ELEMENT m: ChoosingClassesOutput (parameter)>
<!ELEMENT parameter (name,type,value)* >
<!ELEMENT name (#PCDATA) >
<!ELEMENT type (#PCDATA) >
<!ELEMENT value (#PCDATA) >
```

圖5.1.2 訊息格式定義(ClassesService.dtd)範例

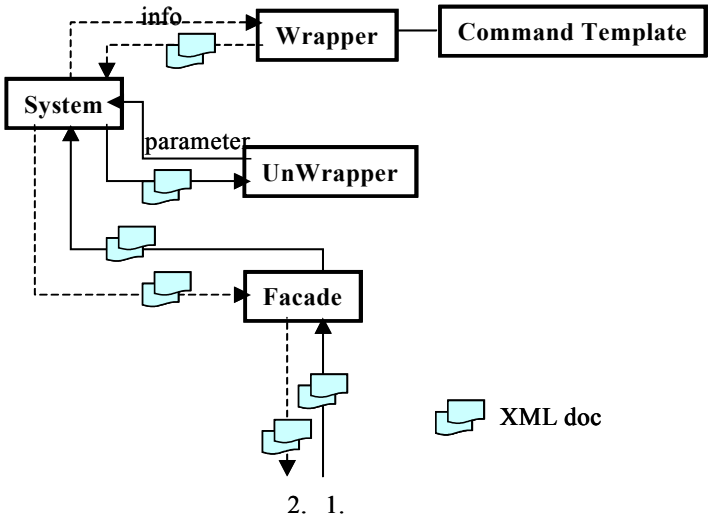


圖 5.1. 4 使用者層應用程式模組運作圖

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/'"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
<m:ChoosingClasses>
                    <parameter>
                     <name> studentId </ name >
                     <type>String</ type >
                     <value> g892804</value>
        <name> passwd </ name >
                     <type>String</ type >
                     <value>1234</value>
         <name> classesId </ name >
                     <type>String</ type >
                     <value> 3006</value>
                    </parameter>
                  </m:ChoosingClasses>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

圖5.1.3 資料訊息範例

5.1.1 訊息格式模組(Command Template Module)

網路服務與使用者應用程式透過 SOAP 封包進行溝通，SOAP 封包包裝了 XML 文件，這些文件描述訊息資訊，使用者應用程式與網路服務建立溝通前必須取得訊息傳送格式，所以使用者應用程溝通前必須取得訊息傳送格式，使用者應用程式在啟動的時候會進行初始化，使用者應用程式要求取得適當的 WSDL 文件如圖 5.1.1.1，這樣的文件描述網路服務與應用程式間的溝通機制。

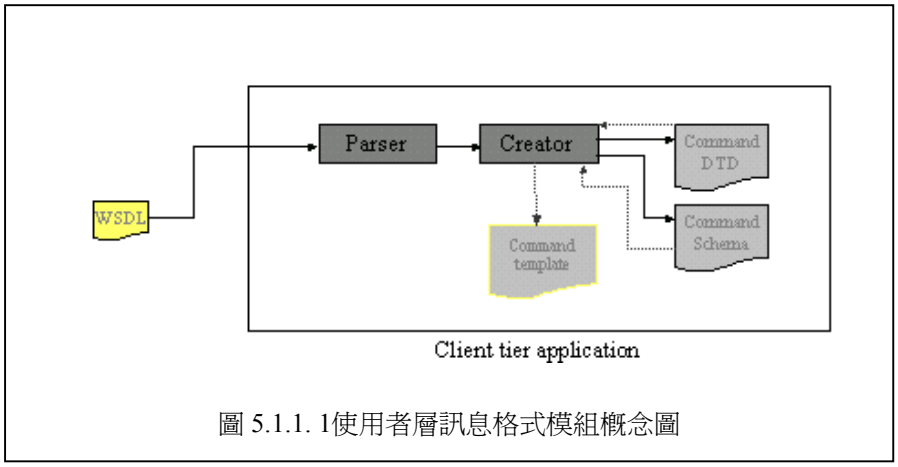


圖 5.1.1. 1使用者層訊息格式模組概念圖

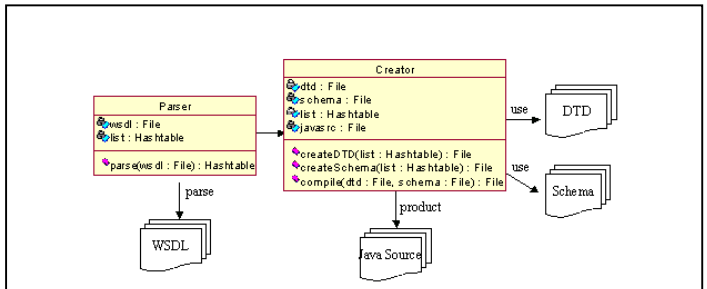


圖 5.1.1. 2 使用者層訊息格式模組架構圖

本論文的擷取網路服務溝通格式模組利用 Parser 讀取 WSDL 文件產生 element 的名稱、型態與值的對應表(hashtable)如圖 5.1.1.2 所示，交由 Creator 建立訊息模板，Creator 產生訊息資料格式(DTD)與資料編碼方式(schema)，Creator 利用上一階段產生的資料格式與資料編碼方式透過 compile 產生訊息類別(class)，這些類別就是訊息命令模板(Command Template)，系統可以透過實作這些模版產生訊息物件，擷取網路服務溝通格式模組如圖 5.1.1.2 所示。

5.1.2 溝通模組(Communication Module)

系統溝通包括本地端指令存取和遠端溝通，所以異質平台溝通存取在所難免，因此以物件的方式進行溝通可以降低平台環境的相依性，所以本文章利用溝通模組傳遞物件資訊，並以 XML 作為資訊交換的基本格式，SOAP 封包作為訊息傳遞媒介。溝通概念模組織概念圖如圖 5.1.2.1 所示，使用者應用程式中的 Controller 擁有多個 Wrapper 和 UnWrapper 分別負責包裝不同的物件，Wrapper 透過訊息格式模組產生的 Command Template 產生適當的 SOAP 物件，UnWrapper 解讀外界傳入的 SOAP 封包，透過這樣的機制達成溝通。

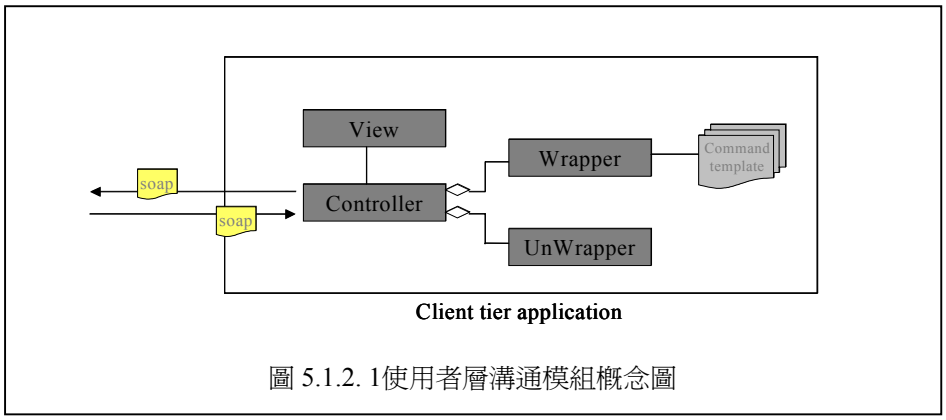


圖 5.1.2. 1使用者層溝通模組概念圖

圖 5.1.2.2 描述溝通訊息模組架構，溝通訊息模組包含 Wrapper 和 UnWrapper 兩個主要元件，Wrapper 提供 getTemplate、wrap、setInfo 三個 method，Wrapper 利用 getTemplate 取得適當的訊息模板，這些訊息模板來自於網路服務溝通格式模組，setInfo 存取模板資訊，最後利用 wrap 包裝成 SOAP 物件對外傳送。UnWrapper 包含 parse、getName、getValue 三個 method，UnWrapper 利用 parse 讀取外部傳進來的 XML 物件，轉換訊息資料型態，透過 getName 和 getValue 擷取訊息資訊，將 XML 文件轉換成應用程式能夠接收的格式。

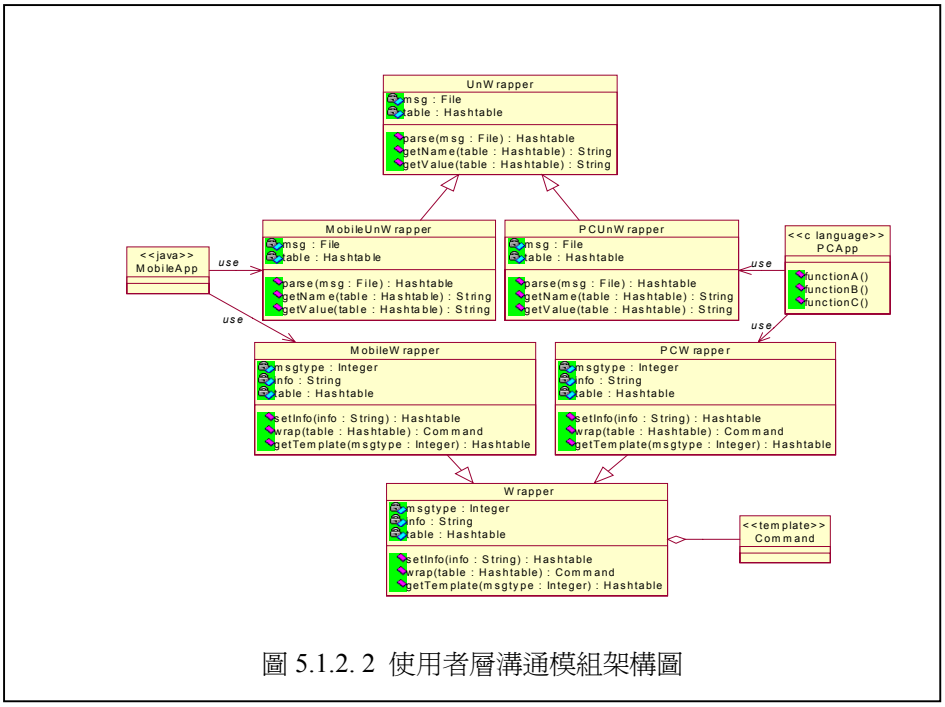


圖 5.1.2. 2 使用者層溝通模組架構圖

5.2 網路服務層(Web Service Tier)

網路服務層提供儲存網路服務資訊的共享空間，本論文提供一個網路服務資訊管理機制，其架構圖如圖 5.2.1 所示。網路服務資訊的資料結構從企業組織分類(Classification)開始到各個不同的企業個體，組織分類可能包括金融、電子、貿易、證券...等不同的類別，在這些組織分類底下包含多個不同的組織(Orgnization)，如台基電、聯電、寶來證券...等，每一個組織裡提供一到多項服務(Service)。

網路服務描述文件，是由服務提供者所提供，讓服務使用者抓取，所以在我們的設計架構中，使用者和服務提供者透過 RegistryFacade 介面存取這些資訊，設計架構中提供 publish 和 query 等 method，網路服務提供者透過 RegistryFacade 介面呼叫 publish method，本設計架構會透過 BusinessLifeCycleManager 管理網路服務相關資訊。當服務提供者公告一個網路服務成功後會得到一個 key 代表這個服務，使用者可以利用這個 key 進行 query。在這個架構當中，使用者應用程式利用 Façade interface 的 query method，query method 會經由 BusinessQueryManager 中的 queryByClassification、queryByOrganization、queryByService 對網路服務資

訊進行不同層面的搜尋。

圖 5.2.2 說明網路服務註冊流程，網路服務透過 RegistryFacade 進行註冊，網路服務資訊管理機制利用 BusinessQueryManager，先增加企業組織於適當的分類項目之下，然後增加企業組織對外提供的企業服務，最後構成一個公開網路服務。

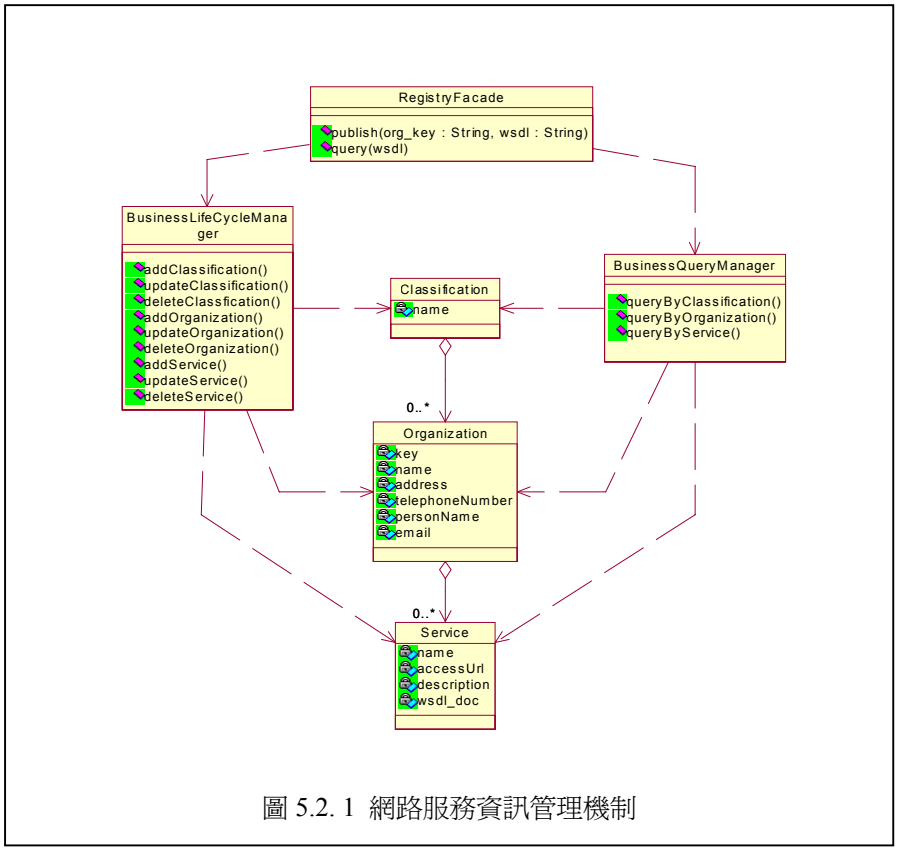


圖 5.2. 1 網路服務資訊管理機制

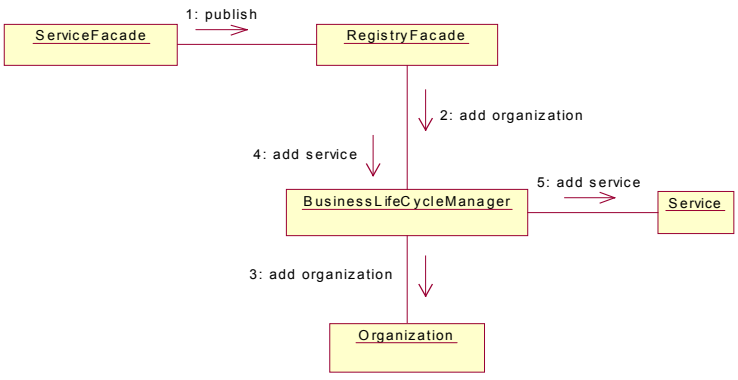


圖 5.2. 2 網路服務註冊流程

5.3 合作層(Collaboration Tier)

雖然網路服務中的軟體元件提供一至多個功能,但是有些使用者應用程式的要求事件往往需要透過多個元件間的協同合作,形成一個工作流程(workflow)[22]才能提供完成使用者應用程式的要求。再考慮到工作流程的設計,可能隨著需求的增加而新增修改,合作的軟體元件也將跟著新增或置換。同時不同的事件處理需要不同的工作流程,當網路服務中的軟體元件修改,相關的工作流程也必須跟著修改,甚至是重新建置。因此,在本架構中的合作層將提供彈性的工作流程建置機制,開發人員可以很容易的客製出其所需的工作流程,客製出的流程稱為流程模板。

在合作層中根據不同的網路服務外部 SOAP 需求(request)提供了不同的流程模版。因此在以下的內容將先介紹流程模板(flow template)的定義,接著介紹合作流程的整體運作機制包含流程模板的參數設定與流程的運作機制。最後將介紹設計人員將如何彈性的建立合作機制。在介紹的過程中將以一個範例作為說明。

5.3.1 流程模版(Flow Template)的定義

流程模版定義了一個服務的運作流程,稱為模版的原因是因為該流程還必須置入(binding)SOAP 文件裡的參數,才能真正被執行。一個流程(flow)通常包含各個狀態轉換(state transition)、事件(event)、條件(condition),需求的元件溝通介面(SystemFacade)、執行的命令(command)及參數(parameter)等。這些資訊的定義稱

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT flow
(support_service,event*,state_transition*)>
<!ELEMENT support_service (#PCDATA)>
<!ELEMENT event(name,condition*)
<!ELEMENT condition(name)>
<!ELEMENT state_transition(system_facade*,condition
<!ELEMENT system_facade(name,command,event)>
<!ELEMENT command(name,parameter*)>
<!ELEMENT parameter(name,type,value)>
<!ELEMENT name(#PCDATA)>
<!ELEMENT type(#PCDATA)>
<!ELEMENT value(#PCDATA)>
```

圖5.3.1.1資料型態的DTD

為資料型態(meta-mode)，資料型態的 DTD 如圖 5.3.1.1 所示。利用此資料型態可以定義出一個流程模板，該資料型態的詳細定義如 5.3.1.1 中的表格。至於各個型態間的關係則如圖 5.3.1.2 所示。

表 5.3.1.1 資料型態表

型態	描述
流程(flow)	流程資料形態包含了以下所有型態，為單一流程模版的 root 節點。通常一個流程至少支援了一個 SOAP 的 request。
事件(event)	事件是流程運作的基礎，流程的初始必須有一個事件被觸發，而在流程中的各個節點運作時也會產生其他的事件。
條件(condition)	一個事件(event)引發後會使許多條件成立。當一些特定條件成立時流程才會發生狀態轉換(state transition)。
狀態轉換 (state transition)	一個工作流程包含了許多狀態轉換，個別狀態轉換都是流程中的一個節點。一個狀態轉換會執行特定的系統介面(SystemFacade)
系統介面 (SystemFacade)	一個系統介面代理一個提供服務的系統(legacy)。詳細的介紹可參考 3.4
命令 (command)	呼叫一個系統介面執行時必須指定執行的命令。

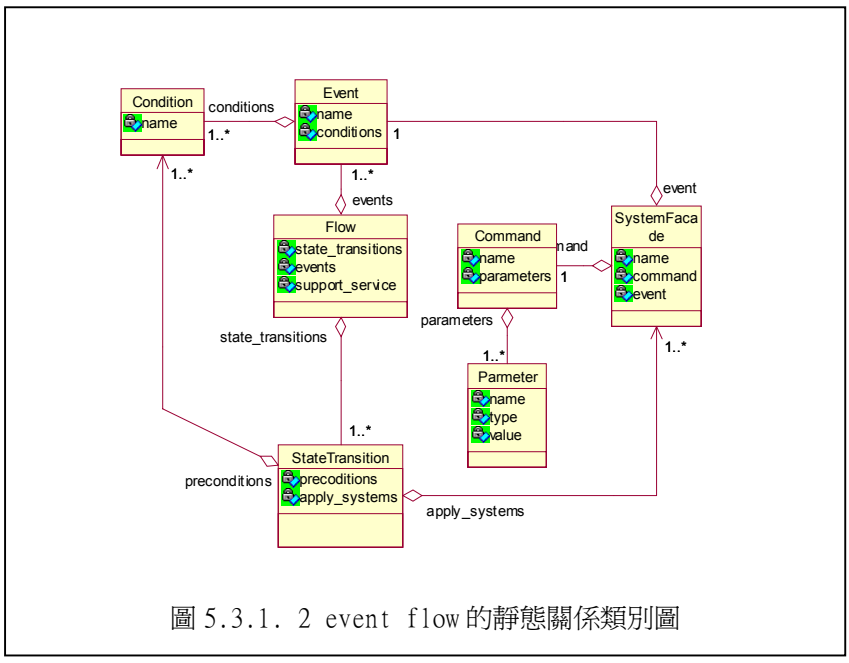


圖 5.3.1. 2 event flow 的靜態關係類別圖

在型態間的關係中，一個流程(flow)包含了許多的事件(event)與狀態轉換(state transition)。一個狀態轉換前必須有特定的一些前置條件(condition)成立，前置條件成立後，一至多個服務系統介面(SystenFacade)會被執行。服務系統執行前必須傳入執行命令(command)及包含於命令中的參數(parameter)。服務系統執行後會引發一個事件(event)，此事件將使其他的條件(condition)成立，而當特定的一些條件成立後，流程會進行下一個狀態轉換(state transition)。

在了解流程模版的定義之後，在接下來的內容則是說明由使用者應用程式傳 soap 文件至合作層後的運作機制。

5.3.2 合作層的運作機制

合作層在接收到 SOAP 的文件之後，接下來的動作可分為兩個階段，第一階段是找出適當的流程模版，並將 SOAP 裡的參數放進 context storage。第二階段則是執行設定完的流程。在本章節中將以學生選課流程為例說明合作層的運作機制。

5.3.2.1 流程模板的參數設定

流程模板的參數設定過程如圖 5.3.2.1 所示。第一個步驟裡使用者應用程式

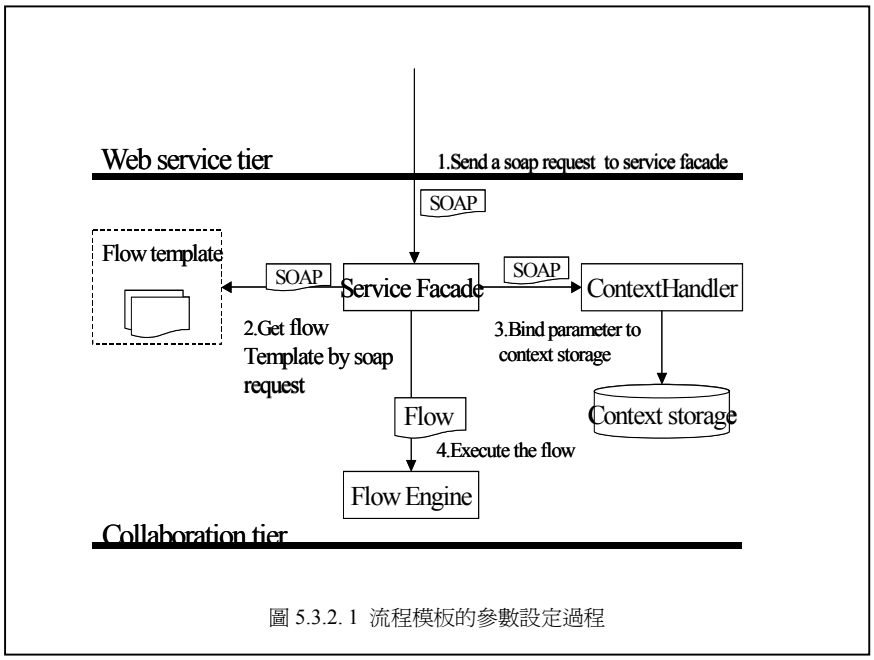


圖 5.3.2. 1 流程模板的參數設定過程

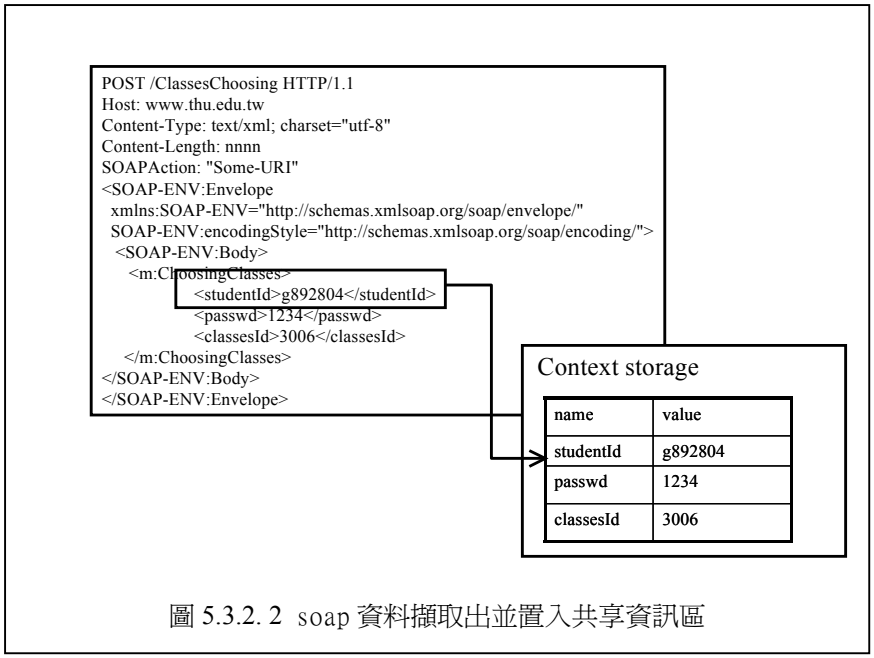


圖 5.3.2. 2 soap 資料擷取出並置入共享資訊區

將 SOAP 文件傳至 ServiceFacade，ServiceFacade 是一個網路服務的窗口，該類別提供一個 acceptService 的方法(Method)，參數為一個 SOAP 文件，該文件封裝著需求(request)的名稱及參數。在第二步驟裡 ServiceFacade 根據 SOAP 裡的資訊找出對應的流程模版。第三步驟裡 ServiceFacade 將流程模版與 SOAP 文件傳至 ContextHandler 類別，該類別會將 SOAP 文件內的資料擷取出並置入共享資訊區(context storage)，共享資訊區裡儲存了各個軟體元件彼此會共用的資訊，此步驟的目的在於當流程開始執行時，服務系統需要 SOAP 裡的參數才能執行命令(command)。以選課流程為例，圖 5.3.2.2 中 SOAP 文件裡的資訊如學號、密碼及課程編號會放置於 context storage。第四步驟則是將 XML 格式的流程模版由 flow engine 執行。

5.3.2.2 流程運作機制

在流程的運作方面如圖 5.3.2.3 所示，由 ServiceFacade 接收 SOAP 外部事件，SOAP 外部事件透過流程模板參數設定進行流程初始化，在流程初始化階段 Context Storage 儲存系統共享資訊，在流程運作機制提供動態定義流程的彈性。

ServiceFacade 接收 SOAP 外部事件驅動，透過工作流程參數設定模組，產生工作流程實體及系統共享資訊，FlowEngine 透過監控 ConditionPool 控管流程

狀態的行徑，當 FlowEngine 發現 ConditionPool 有新的條件產生，同時發現事件條件成立時 FlowEngine 會進行狀態轉換，執行下一個步驟，一個狀態的執行關係到一至多個 SystemFacade，呼叫系統前必須從 Context Storage 取得適當的參數，利用這些參數進行系統指令設定與執行，最後系統會對外發送訊息，透過訊息重導(Message Redirector)機制將系統對外發訊的訊息參數存入共享空間 Context Storage，並產生進入下一狀態的執行條件。

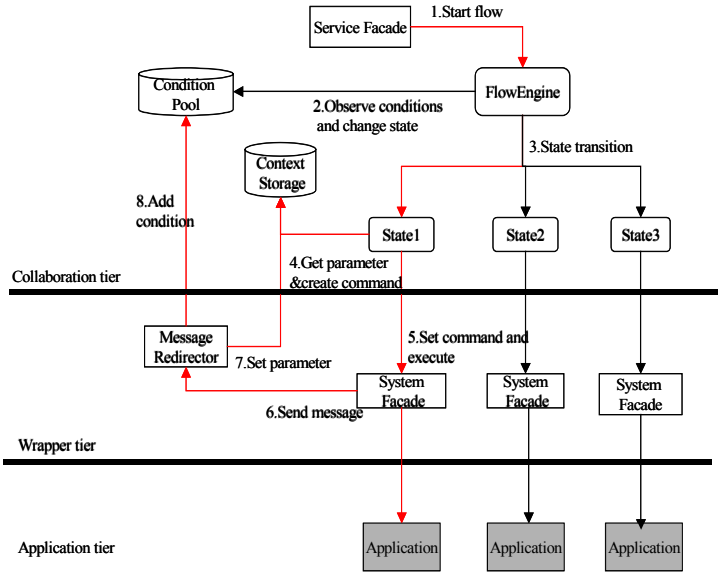


圖 5.3.2. 3 流程運作圖

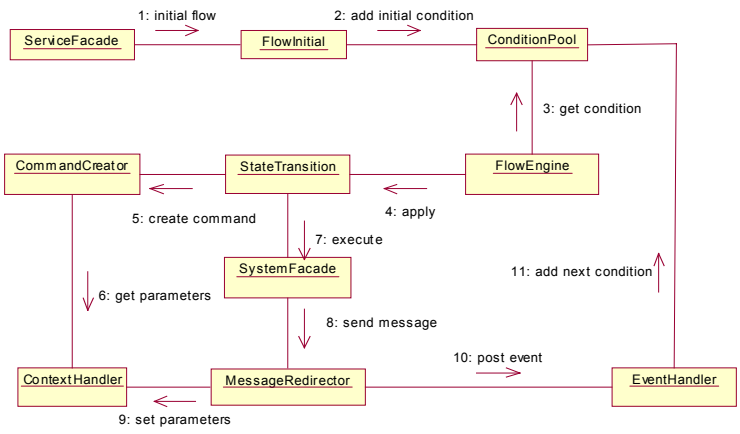


圖 5.3.2. 4 流程運作圖

圖 5.3.2.4 表示流程運作圖，Flow Manager 透過 init 初始化工作流程，並且設定進入流程的進入條件，同時將工作流程定義交給 Flow Engine 開始執行，Flow Engine 透過監控 Condition Pool 呼叫適當的 State Transition，State Transition 透過 apply 呼叫 Command Creator 設定系統命令，Command Creator 呼叫 Context Handler 的 getContext 取得 Context Storage 中的系統共享參數，最後呼叫 System Façade 執行系統命令。系統執行完後產生新的 Event 和 Context，透過 Message Redirector 的 postEvent 和 postContext 將 Context 交給 Context Handler 的 setContext

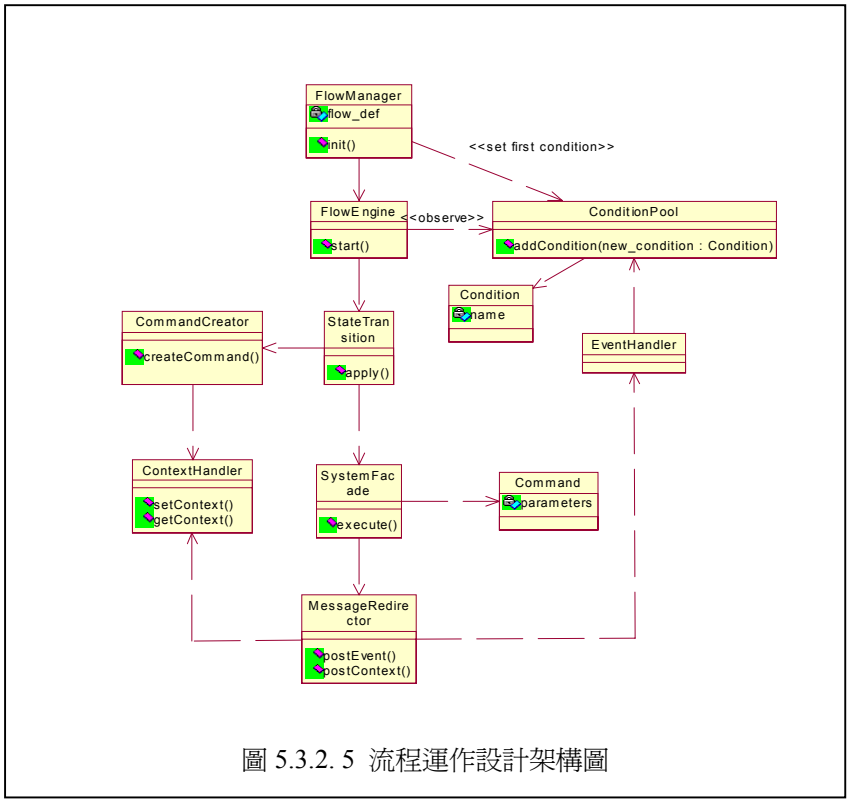


圖 5.3.2. 5 流程運作設計架構圖

存取 Context Storage，新增一項共享參數，並將 Event 交給 Event Handler，Event Handler 呼叫 Condition Pool 中的 addCondition 新增條件，Flow Engine 發現 Condition Pool 中有新的條件產生，便呼叫適當的 State Transition，這又是下一個循環的開始，如此遞迴的執行直到完成整個工作流程，完整的流程圖如圖 5.3.2.5 所示。

5.3.3 如何建立合作層機制

在了解流程的定義及其運作機制後，接下來的內容是討論以系統整合人員的立場如何定義服務系統間合作的流程。定義流程的第一步是建立流程的運作過程，如圖 5.3.3.1 是用 UML 的活動圖定義出的一個學生選課流程。接下來是找出運作過程中有哪些活動如圖 5.3.3.1 中的學生認證(Verification)、課程註冊(Register Class)、發送通知(Notification)等等。接著是找出每個活動的前置條件(Pre-condition)，如認證成功(Verification success)、認證失敗(Verification fail)等等。

最後的動作是定義每個活動使用到哪個軟體元件的 SystemFacade 及執行的命令(command)。綜合活動(Activity)、前置條件(Pre-condition)、軟體元件(SystemFacade)及命令(command)可以定義出流程中的 State Transition，如圖 5.3.3.2 中的 B 所示。

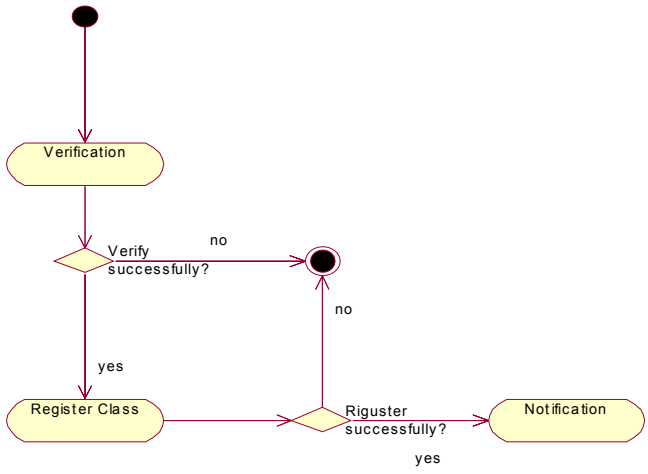


圖 5.3.3. 1 學生選課流程

一個 SystemFacade 執行完後往往產生一些事件(event)這些事件會對應到新的條件(condition)，因此流程的定義裡還包含一個事件的發生對應到哪個條件(condition)，如圖 5.3.3.2 中的 A 所示。

```
<flow>
…
 <event>
  <name>verification_fail</name>                    A
  <condition>verification_fail</condition>
 </event>
…
<state_transition>
 <name>Verification</name>                          B
 <condition>
  <condition>start_verification</condition>
 <condition>
 <system_facade>
    <system_name>Membership Mamagement System</system_name>
    <command>
      <name>Verification</name>
      <parmeter>
        <name>studentId</name>
        <type>String</type>
       </parmeter>
      <parmeter>
        <name>Passwd</name>
        <type>String</type>
      </parmeter>
    </command>
    <event>verification_success|verification_fail</event>
 </system_facade>
 </apply>
</state_transition>
…

</flow>
```

圖 5.3.3. 2 流程定義

5.4 包裝層(Wrapper Tier)

網路服務提供現有系統合作整合的空間,在系統進行整合的同時必須考量舊有系統的屬性。同時,系統適度的重整與包裝是完成系統整合在所難免的事情。所以本設計架構提供一個封裝層,專門負責包裝異質系統,以軟體元件化的方式為傳統系統提供溝通機制,一個軟體元件面對多重合作元件,複雜的存取會導致系統設計維護困難,所以良好的存取介面控管可以提昇系統維護性,在包裝層利用 Façade Pattern 建立系統溝通窗口,加上一系列客制化機制,同時也對元件內部的事件處理提供一個較好的設計方式。

這個階段利用了 Gamma et. al.在 1994 年提出設計樣板(Design Patterns)中的 Proxy Pattern[12]、Decorator Pattern[12]、Chain of Responsibility Pattern[12]、

Douglas Schmidt 在 1999 年提出的設計樣版 Wrapper Façade Pattern[23]及[24]中所提到的 Message Redirector Pattern。以下將針對各個設計樣版及整個包裝層所提供的框架做說明。

5.4.1 建立對外統一窗口—導入 Façade 樣版

一般程式設計目標都是減少系統與系統間的複雜存取關係和相依性，在這個階段本研究使用 Façade Pattern 架構，封裝了網路服務內部的軟體元件，這些元件來自於待整合的應用系統，這些應用系統分享部份系統內部資源，透過 Façade Pattern 包裝這些資源形成獨立的軟體元件，，對外提供單一的存取介面，呈現一對多的運作關係，網路服務內部系統運作因此而達到元件化。

網路服務的內部元件可以處理多種不同的事件，也因軟體元件可以處理的命令相當多元，降低事件處理的複雜度也是此階段的設計重點。所以使用 Chain of Responsibility Pattern 的觀念以責任制的方式分配命令處理者，一則可以降低命令處理複雜度，二則可以增加元件內部事件處理擴充性。

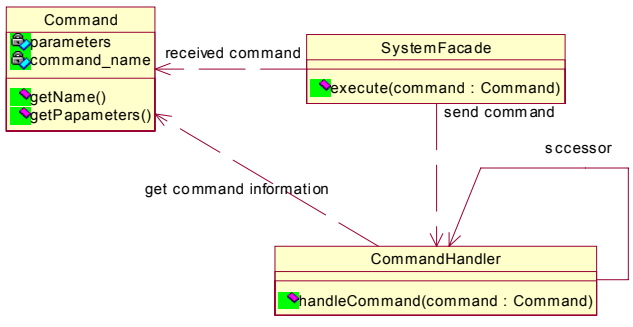


圖 5.4.1. 1 元件溝通介面與事件處理之設計架構

圖 5.4.1.1 表示網路服務內部軟體元件溝通介面與事件處理之設計架構，State Transition 呼叫 System Façade 並將 Command 傳給 System Façade，System Façade 表示網路服務內部元件的對外溝通介面，每個元件可以接收多種不同的命令，本論文利用 Chain of Responsibility Pattern 的觀念設計元件內部事件處理機制。多個 Command Handler 組成一個元件內部的 Command Chain，Command Chain 中的每

一個 node 都是一個 Command Handler，每一個 Command Handler 負責處理特定的一種命令，當 System Façade 接收到外部命令之後會導入 Command Chain，這個命令會在 Command Chain 中傳遞，直到傳至可以處理它的 node，這個 Command Handler 接收 Command 相關資訊，並進行處理。

5.4.2 整合物件導向式服務系統—導入 Proxy 樣版

當整合的目標系統本身為物件導向式系統，但卻缺乏讓外界使用的公開介面，此時就可以導入 Proxy 設計樣版，此設計架構為實際系統提供外部存取介面，此介面扮演系統代理人的角色，外部事件可以透過此介面與系同實體進行溝通。

在此我們以課程管理系統為例，如圖 5.4.2.1 所示，此課程管理系統對外提供一個 ClassManagerProxy 存取介面，此介面為課程管理系統的遠端存取介面，Command Handler 可以透過呼叫 ClassManagmentProxy，間接執行 ClassManagment 的 getClassInformation method 得到課程資訊，如圖 5.4.2.1 所示。

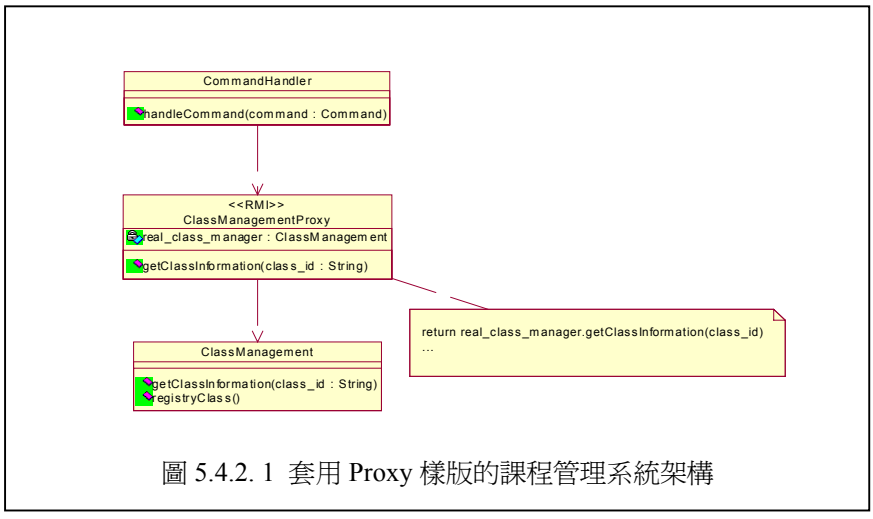


圖 5.4.2. 1 套用 Proxy 樣版的課程管理系統架構

5.4.3 建立對外訊息發送—導入 Message Redirector

網路服務中的軟體元件進行合作的時候，單一軟體元件需要取得其它元件的服務，或是通知其他元件進行另一個動作，或者驅動合作層中新的流程運作。但由於軟體元件本身在設計時並未考慮對外發送訊息，因在本論文提出了一個事件

重導機制如圖 5.4.3.1 所示，各個軟體元件透過 MessageRedirector 對外發送訊息。其機制的設計如 5.4.3.2 所示。

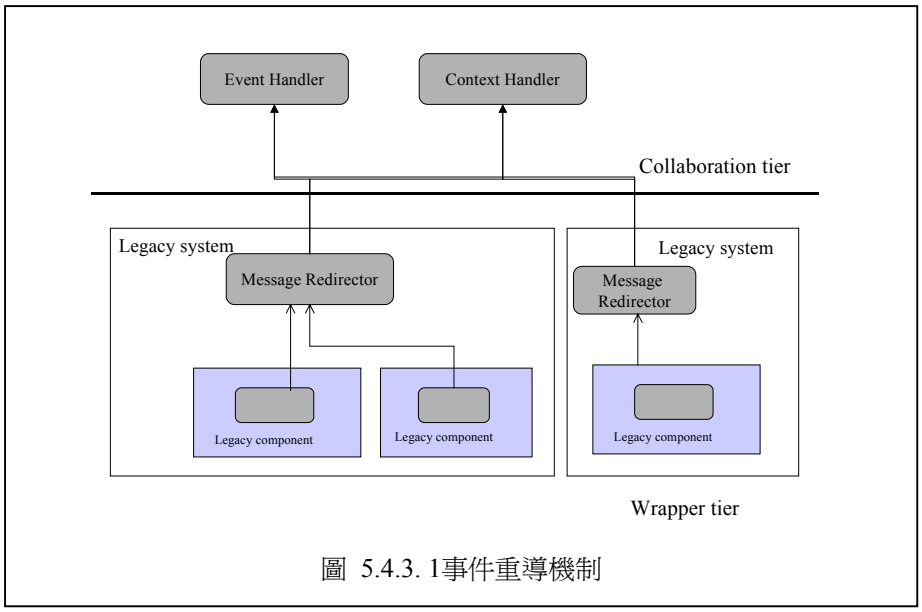


圖 5.4.3. 1事件重導機制

MessageRedirector 提供了兩個方法分別為 postEvent 與 postContext。如果軟體元件需要對外傳遞事件可以透過 psotEvent 將事件重導至合作層的 EventHandler 、 EventHandler 會將事件對應的條件 (condition) 新增至 ConditionPool，以間接驅動流程的運作。而如果軟體元件產生的資料是其他軟體元件執行時所需的參數，則透過 postContext 可以將資料傳至 ContextHandler，

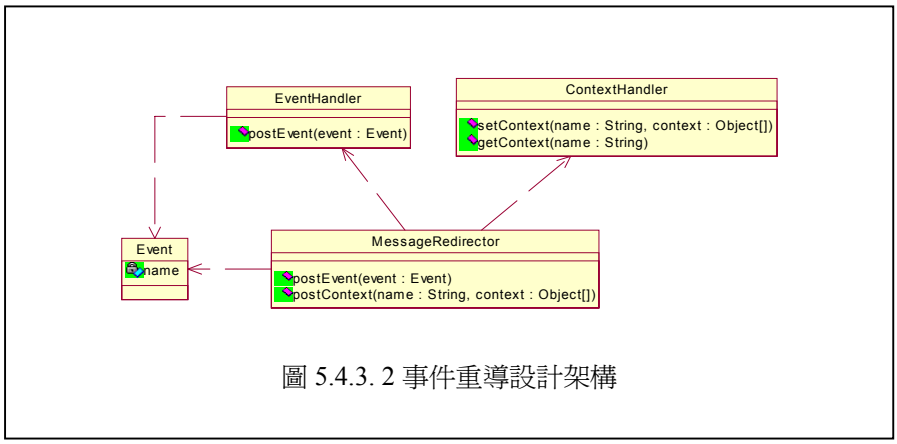


圖 5.4.3. 2 事件重導設計架構

ContextHandler 會將資料加入共享資訊區 Context storage。

5.4.4 對服務系統作客製化—導入 Decorator 樣版

進行系統整合，對現有系統進行客制化在所難免，也就是說適度的添增功能，在這個階段的設計我們利用 Decorator Pattern[12]彈性的附加網路服務內部軟體元件功能。

在此我們以課程管理系統、學生資訊系統與電子信件系統整合為例，每個系統需要載入訊息重導機制，這項機制是先前系統內部不具備的功能，所以透過 ClassManagementDecortor 可以對舊有系統進行功能擴增如圖 5.4.4.1 所示，ClassManagementDecortor 中的 registerClass 呼叫 ClassesManagement 中的 registerClass，並且執行 MessageRedirector 中的 postContext 和 postEvent 產生下一階段的參數及觸發事件，程式範例如圖 5.4.4.2 所示。

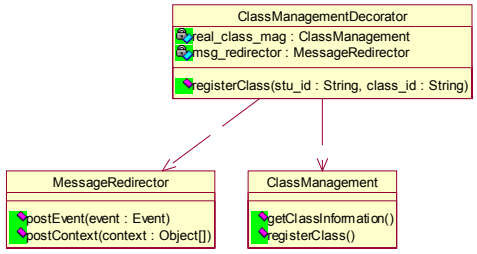


圖 5.4.4. 1套用 Decorator 樣版的課程管理系統架構

```
registerClass(stu_id:String, class_id:String) {
        Object[] obj = real_class_mag.registerClass();
        msg_redirector.postContext(obj);
        Event event=new
        Event("REGISTER_FINISH"");
        msg_redirector.postEvent(event);
}
```

圖5.4.4. 2 添加訊息重導功能課程註冊程式範例

5.4.5 封裝非物件導向系統—導入 Wrapper Façade

在這個階段的設計架構利用 WrapperFacade Pattern[23]對程序式系統進行包裝，非物件導向的系統中的函式(function)和資料(data)，透過 Wrapper Façade Pattern 包裝，結合物件導向式類別介面(Object-Oriented class interfaces)，使舊有系統具可攜性(portable)導入物件導向式系統。至於讓 JAVA 程式結合異質語言如 C、C++甚至是 assembly 可以利用 Sun 所提出的 Java Native Interface (JNI) API。

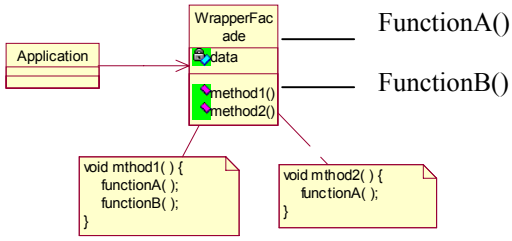


圖 5.4.5 2 封裝非物件導向式系統之架構

第六章 現有系統導入程序

經過前幾章的介紹，了解本論文多階層架構的設計之後，第六章以現有系統整合程序為方向，說明現有系統導入本論文所提出的設計架構之程序。

首先，對於現又系統進行分析，了解現有系統的組織架構關係，歸納系統間需要共享的資源，擷取這些部分，並將它們包裝為軟體元件，如圖 6.1 的第一部份所示。最後進入系統整合的第一步驟，元件封裝。

第一步驟、元件封裝

此階段最主要的目的在於封裝系統共用軟體元件，定義元件對外溝通介面。程式設計師可以依據現有系統規格，針對系統需求使用本論文提供的包裝架構。以下列舉幾項設計方法，一、非物件導向系統可以使用 Wrapper Façade Pattern 設計架構，二、分散式系統可以使用 Proxy Pattern 設計架構，三、客制化系統功能可以使用 Decorate Pattern 加上 Message Redirect Pattern。最後，針對這些 Pattern 內部所包含的角色與現有系統進行對應連結，再建立類別間的使用關係，最後連結不同的 Patterns 間的合作關係。

由於這些元件即將在第二步驟被包裝為網路服務，所以透過這些元件所擁有的功能，我們可以定義網路服務對外的溝通介面，通常這是一個較具一般性的溝通介面。

第二步驟、網路服務封裝

此階段最主要的工作在於定義網路服務對外溝通介面，因為網路服務所接收的多半是 Command 物件，所以網路服務對外溝通介面必須被定義為較通用的格式。這樣的設計架構多半套用 Façade Pattern，封裝了所有網路服務內部的系統元件。

在包裝完網路服務之後，進入第三步驟，我們對網路服務內部元件運作進行定義，也就是建立元件合作機制。

第三步驟、建立元件合作模式

在本論文中提出網路服務內部事件處理流程規格(Event Flow Template)，套用此規格可以快速的建立事件處理流程，迅速為系統導入新功能。所以在此階段我們針對事件需求定義元件合作關係，建立狀態改變流程，建立狀態改變條件，透過事件觸發及狀態改變完成事件處理。

最後完成一個網路服務，為網路服務功能進行描述，並將描述文件公告出來。使用者透過這些描述文件，擷取有效的網路服務。

第四步驟、網路服務註冊階段

網路服務建製完成後，程式設計師必須將這個網路服務公諸於世，也就是透過大家公認的描述方式(WSDL)為網路服務進行描述，此標準的文件存放在 Registry Station，使用者應用程式透過這些文件可以找到適當的網路服務。此標準的描述文件中包含有網路服務相關的描述，如 location, IP address, parameter, …等等。正確的描述格式可以參這 WSDL 相關網站。

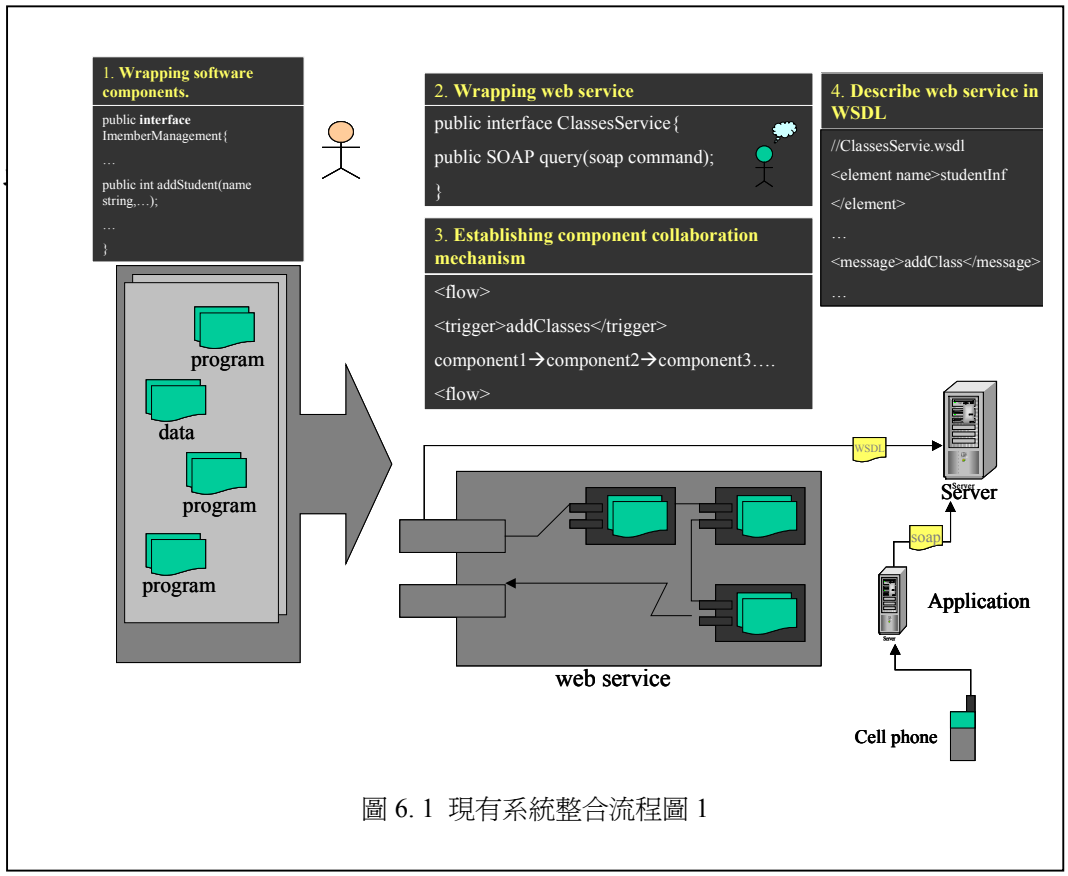


圖 6. 1 現有系統整合流程圖 1

上四個步驟是現有系統導入網路服務的基本步驟，接下來要說明的是使用者應用程式如何與網路服務建立連結溝通。

終端節點(Browser)得到資訊之前，使用者應用程式必須先得到網路服務溝通的資料格式，如圖 6.2 所示。這些格式的產生是來自於使用者應用程式抓取網路服務描述文件(WSDL)，透過這些訊息格式定義，使用者應用程式並可以於網路服務進行溝通。所以在接下來的步驟裡程式設計師必須套用本論文中 Client Tier 所使用的 Command Template Module，使用者應用程式連結到網路服務註冊中心，擷取網路服務描述，透過 Command Template Module 中的 parser，解析這份網路服務描述文件，產生網路服務的 Command DTD，再由 Command Template Module 中的 Creator 建立 Command 類別物件。如此一來使用者應用程式只要套用這些 Command Template 就可以產生網路服務可以接收的命令物件，透過 HTTP 傳送。反之，使用者應用程式也可以接收網路服務回傳回來的命令訊息，將網路服務回覆的 SOAP 封包 unwrap 成使用者應用程式可以解析的系統參數。

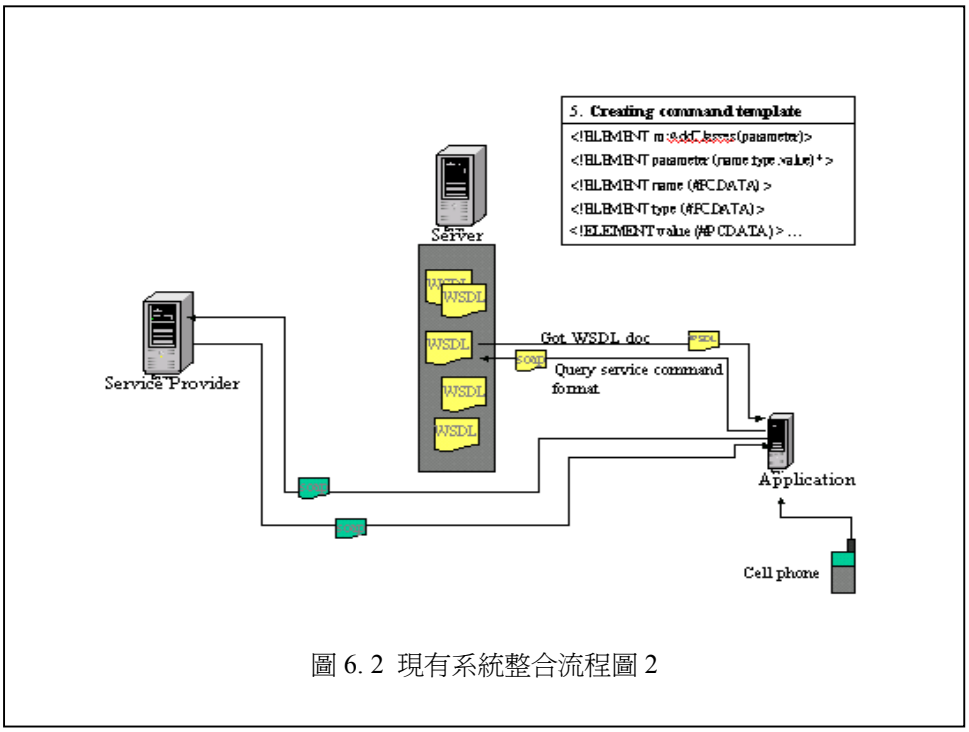


圖 6. 2 現有系統整合流程圖 2

第七章 結論與未來工作

企業組織趨向電子化增加了使用的便利性，加快了生產效率，但是這些利益專中間隱藏著很多危機。大量的數位化後，會需要有效的資源管理，妥善的管理不但可以提高工作效率，還能夠迅速的拓展事業範圍。反之，不佳的經營管理，不但促使企業電子化無法幫助企業產能，反而會造成資料混亂等現象。

經過二十世紀，企業趨向電子化，所有的企業組織一股腦的追逐潮流，在跟隨時尚脈動的同時，人們往往忽略管理的重要性。所以企業內部的整合，企業彼此之間的合作，是二十一世紀的使命。有效的企業整合，廣闊的合作規模，健全的管理機制，是這一個世紀至勝的關鍵。

所以本篇論文，有見此趨勢，提出一個幫助企業進行系統整合的架構。導入多階層架構設計於系統整合，意圖增加系統建立合作機制的便利性。利用設計樣板(Design Patterns)幫助解決程式設計問題，結合 XML 增加資料傳遞的獨立性，這個架構幫助程式開發者整合舊有系統(EAI)，導入網路服務(Web Service)，結合新興平台(Platform)。

對於一般不熟悉階層架構與設計樣版的開發人員而言，整合的過程並不容易，因此我們希望在未來對軟體技術及系統整合架構作進一步延伸，整合現有 EAI 技術，以本文章所提出的多階層架構為基礎，利用設計樣板，依循軟體設計標準，希望提供一種軟體輔助工具幫助程式設計師，有效且快速的整合現有軟體導入網路服務機制。

在製作整合工具之前，本研究對現有系統間，共享資源的擷取、元件客制化及結合網路服務其它標準等三部份還稍顯薄弱，以上數點為我門下一階段的研究方向。以下針對這幾點進行進一步的說明：

1. 現有系統間，共享資源的擷取之研究

系統整合之重要關鍵之一就是擷取系統間的共享資源，網路服務就是透過蒐集這些系統間的共享資源，並將這些資源釋放，讓所有待整合的系統都能連結到這個共享的環境裡，藉此達成整合的效果。

本階段的工作項目分述如下：

(1) 了解系統間的關連性，擷取關連性資料。

(2) 透過關連性資料取得系統間應用程式的關連性。

(3) 透過上述兩步驟，歸納系統間需要獨立的資源。

(4) 建立有效網路服務描述，讓整合的系統都能存取這個共享的網路服務資源，因此而達到整合的目的。

2. 網路服務內部元件客制化機制之研究

網路服務內部元件整合的許多現有系統先共享的資源，這些資源可以是相關的資料或是應用程式。這些元件要達到合作的機制，需要加入一些客制化的功能，才能滿足元件合作的功效，本論文所提出客制化的法則還稍顯薄弱，有待下一階段再進行進一部的研究。

3. 結合網路服務其它標準之研究

網路服務(Web Services)這個系統整合標準，是個相當新的議題，各家大廠有見於這塊大餅紛紛定義網路服務階段性標準如 WSDL、UDDI，WSFL、WSCL 同時提供一些幫助開發的套件如 SUN 的 Web Service Pack。本研究架構並未囊括所有的網路服務相關標準，相信日後不久必定會有更新、更完善的標準被制定，所以本研究需要跟個時代的腳步，導入各種系統整合相關標準，使整架構達到完善的地步。

參考文獻

[1] William C. Chu, L. S. Sun, and S. J. Yang, “A Formal Model to Object-Oriented Program Reuse,” Proceedings of 7th Workshop on Object-Oriented Technologies and Applications, Taiwan, pp. 65-74, Sep., 1996.

[2] C. L. Ong and W. T. Tsai, “Class and Object Extraction from Imperative Code,” Journal of Object-Oriented Program, Mar., 1993, pp. 58-68.

[3] Neighbors, J., “Software Construction Using Components,” doctoral disseration, Univ. of California, Ievine, Calif., 1981.

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

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

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

[7] William C. Chu and H. Yang, “A PRT Net to Software integration for Reuse,” Proceedings of IEEE COMPASAC'96, Korea, pp. 343-348, Aug., 1996.

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

[9] Hans-Peter Steiert, “Towards a component-based N-Tier C/S Architecture” ISAW ’98: Proceedings of the Third International Workshop on Software Architecture,

pp. 137-140, 1998.

[10] UDDI http://www.uddi.org

[11] WSDL http://www.w3.org/TR/wsdl

[12] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns, Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[13] M.Moriconi,X.Qian,and R.A. Riemenschneider," Correct Architecture Refinement", IEEE Transactions on Software Engineering, Vol.21, No.4, pp. 302-313, April 1995.

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

[15] JAXB http://java.sun.com/xml/jaxb/

[16] JAXM http://java.sun.com/xml/jaxm/

[17] JAXR http://java.sun.com/xml/jaxr/

[18] JAX-RPC http://java.sun.com/xml/jaxrpc/

[19] M. Goedicke and U. Zdun, "Piecemeal legacy migrating with an architectural pattern Language: a case study", Journal of Software Maintenancee and Evolution, Vol.14, No.1 pp.1-30, 2002.

[20] ebXML http://www.ebxml.org/

[21]Rosettanet
http://www.rosettanet.org/rosettanet/Rooms/DisplayPages/LayoutInitial

[22]James Carey, Brent Carlson, Tim Graser, "SanFrancisco Design Patterns", Addison-Wesley, 2000.

[23] Douglas Schmidt, Michael Stal, Hans Rohnert and Frank Buschmann, "Pattern-Oriented Software Architecture " Vol 2, Addison-Wesley, 1999.

[24] M. Goedicke and U. Zdun, “Piecemeal legacy migrating with an architectural pattern Language: a case study”, Journal of Software Maintenancee and Evolution, Vol.14, No.1 pp.1-30, 2002.

[25] JMS(Java Message Service) http://java.sun.com/products/jms/

[26] JNDI(Java Naming and Directory Interface) http://java.sun.com/products/jndi/

[27] JNI(Java Native Interface) http://java.sun.com/products/jdk/1.2/docs/guide/jni/

致　謝

回首這兩年東海大學的研究生生活，過的不但充實而且快樂，雖然只是短短的兩年光陰，但是學到的東西卻不少於大學四年所學，記得有人跟我說過研究所兩年會激發一個人獨立思考以及獨力作戰的能力，現在回想起來我認為研究所給我的不只這些，它還給了我許多用時間、金錢所買不到的寶貴經驗。

如今我能順利的取得碩士學位，必須仰賴許多人的幫助，在此特別感謝我的指導教授朱正忠老師，在老師身上我不但學到完整的研究過程，而且更重要的是老師圓融的處事技巧以及良好的人際關係，都是我所努力學習的對象。

最後，更感謝一群好友所給我的甜蜜回憶，包括：文達、義清、逸群、依恆、俊雄、祚民、繼賢，你們讓我的研究生生活過得多采多姿，你們讓我覺得研究是一件很快樂的事情，你們都是構築我研究生活的完美因子。

王靜慧