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

重整非網路服務系統

成為一個網路服務系統

Re-engineering Non-Web Services

to Web Services

研 究 生：許俊雄

中 華 民 國 九 十 三 年 六 月

中文摘要

電腦從發明到現在已經有幾十年的歷史，在這之中產生了許多優秀的程式，但是隨著電子技術的進步、軟體理論的成熟、網路通訊的發達，電腦系統應用架構更從舊有核心應用(core legacy application)轉化到 Client/Server 及 web-enabled 架構，再逐步轉化至分散式運算(Distributed computing)架構，技術快速地改變造成在這每一轉換的階段裡，許多優秀的軟體由於無法適應變化，而被淹沒在時代洪流之中。一個好的建築物可以永世留存，傑出的設計建築風格更被當成後世的典範，而能不斷地被複製及延伸。一個電腦應用系統或許不能適應新環境而被淘汰，但是在這系統內部或許也存在足可為後世典範的設計及可被再發展的概念，如果讓這些設計及概念隨著系統淘汰而遺失，是一件非常可惜的事。

在過去的 10 年裡，全球化的競爭環境及網路溝通無國界化，使得軟體系統需要負擔著從前更龐大的資料流量及更繁複的業務處理邏輯，造成系統設計因此也越趨複雜化。在這情況下，系統開發者追求著一可快速地開發、可便利地擴充、可簡單地維護的高品質軟體似乎是天方夜譚。但若以另一個角度思考，一個具可延展性、可維護性、再使用性的高品質軟體開發是從舊有基礎上來建立，則開發者可以縮短許多開發時間，則追求高品質且低時程的軟體開發過程依然是有可作為。

此篇論文根據重整工程原理提供一連串步驟來萃取舊有系統(legacy system)的設計元素，並依據軟體再使用(software reuse)理論為基礎，以 N-tier 架構建立一軟體架構，使架構其上的系統是可以被延展且易於維護的，而搭配網路服務(web services)更可為應用系統提供增加元件重複使用及系統整合的特性。

關鍵字： 軟體重整工程、軟體再使用、N-tier、網路服務

英文摘要

The computer from invented already has several dozens years history to the present, has had many outstanding formulas in this, but along with the electronic technology progress, the software theory was mature, network communication developed, the computer system from old has the core using the overhead construction to apply core legacy application to transform to Client/Server and the web-enabled overhead construction, again gradually transformed to the disperser -like operation overhead construction, fast changed creates in each stage many outstanding software because were unable to adapt the change, but is submerged in the time mighty current. A good building may always preserve, the outstanding design construction style is regarded as later generations' model, but unceasingly is duplicated and extends. Perhaps a computer application system cannot adapt the new environment to eliminate, perhaps but also exists in this system interior fully may for the later generations model design and may the concept which develops again, if lets these designs and the concept loses along with the system elimination, is a matter which extremely was a pity.

In the past 10 years, the globalization competition environment and the network communication did not have the national boundary, caused the software system to shoulder compared to the before more massive data processing and the complicated handling of traffic, the system design more hasten the complication. In this situation, the system development pursue one may fast develop the high quality software which, may conveniently expand, may simply maintain as if is Arabian Nights. But pondered by another angle that, may the ductility, maintainable, the again use high quality software development be from old has the foundation to come up the establishment, then may achieve the development demand.

This paper basis rallies the project to provide a succession of step to come the extract old has the legacy system the design element, and uses the theory based on the software is a foundation again, establishes a software overhead construction by the N-tier overhead construction, causes above the overhead construction the system is may delay also be easy to maintain, but matches the web services to be possible to provide the increase part repetition use and the system conformity characteristic for the application system.

Keywords: Re-engineering, Software Reuse, N-tier, Web Services

目錄

圖目錄

表目錄

第 1 章　序論

1.1 前言

時代的遷移及資訊技術日新月異，讓軟體開發(software development)面對跟以前不同的挑戰，其中網路化及全球化影響最為深遠。1995 年後 Internet 的普及，給於任何人都有面對全世界的機會，此時 WWW 伺服器網站的建立蔚為風潮，這個契機造成大多數的公司捨 Client/Server 架構程式而逐漸朝向網路架構程式發展，猶以跨國性公司最為積極。但是跨國性公司及組織並不能滿足只單純在網路上建立單向性的網站，更希望為分散在世界各地據點建立統一化的應用及管理互動性系統，如企業內部知識管理系統。可是在此全球化的發展中又有時候需要考量在地化(localization)需求，如多國語言的設置、各地稅制等問題，讓系統架構也逐漸趨向複雜化。系統開發需求(Requirements)常常都是在使用者實際遇到問題的時候才會被提出，使得現在中大型的系統已經無法在一開始便可以完整定義系統需求，面對不斷地需求改變我們無法再一開始的時候就所以造成我們的系統不斷地改變及擴充。

現今的軟體開發需要考慮軟體是否能夠適應不同的地域，作用於不同的平台(platforms)，適用於各種不同的設備及支援不同的語言，讓軟體系統複雜程度提高，此時如果我們還是使用以前手工式軟體開發方式，則將會被全球化的競爭環境給淘汰。軟體工程[23]則是希望提供一個有效率且經過驗證的開發流程及控制的研究，如 Rational Unified Process (RUP)[28]、Extreme Programming (XP)[27]、OPEN 等軟體開發流程方式，讓開發者可以根據不同的開發環境及目的選擇適合的開發流程。RUP 幫助我們在製作大型軟體及管理大型團隊時，提供一系列的流程及規定，並利用階段報告來幫助軟體的維護及承接，以確保開發軟體的品質。RUP 屬於重量級的開發方式，當我們需要快速地產生系統原型(prototype)及

產品(product)時，則可以採用輕量級的開發方式。XP 是屬於一種輕量級的軟體開發流程，透過客戶駐點、持續測試以及獨特的規劃方式，帶來快速的回饋及大量的溝通，來滿足客戶的需求。這些方式靠著控制開發過程中的變數來獲得高程度的軟體品質，但是過長的開發時間讓系統依然存在很多不確定感及危機。

一再地重複執行軟體開發流程可以得到具品質保證的軟體，但是否有方法能利用已存在的軟體來發展新軟體而不用一再地從無到有？這個作法概念稱為軟體再利用(software reuse)。軟體再利用的目的是希望能重複使用所有之前軟體開發過程中的一切資源。在流程方面，可以透過流程再造找出最適合運作的精簡流程。在文件方面，可以對產生的文件製作共用模板(model)來直接套用。在程式方面希望能程式碼再使用，而在資料方面希望資料能建立一次就能在各地存取，這樣一方面可以保證軟體的品質，另一方面也可以縮短開發時間。軟體再利用可以有效地解決需要快速地且有品質地軟體系統製造問題。網路服務便是軟體元件的一種，它能不斷地被重複使用，其在軟體再使用可達到兩個再使用的目的，一是原始碼再使用，另一是資料再使用，透過這兩種再使用的方式讓公司可以更快速地開發軟體，其間也能更有效的溝通。

網路服務(web services)在各軟體組織的大力支持下，如微軟(Microsoft)、IBM等，已經逐漸成為新一代分散式環境運算標準。在應用服務方面，可以用來處理公司之間(Business to Business, B2B)的服務，公司可以把關鍵的商業應用提供給上游的供應商及下游的客戶，透過彼此的資料交換分享，讓資訊透明化，節省溝通的時間及成本。在軟體工程(software engineering)方面，也具體實現軟體再使用(software reuse)的目的，Web services 的技術除了讓優良的元件可以不斷地重複使用，並且也能達到資料重用的目的。

1.2 研究動機

軟體發展從 1950 年代至今，已經有 50 多年的歷史，在各年代期間產生了許多優秀的軟體，深深影響當時軟體設計及電腦應用，但是隨著電腦技術的進步、軟體理論的成熟、網路通訊的發達，電腦軟體系統應用程式架構更從舊有核心應用(core legacy application)轉化到 Client/Server 及 web-enabled 架構，再逐步轉化至分散式運算架構[17]。架構的改變，使得許多優秀的軟體由於無法適應變化，淹沒在時代洪流之中，這是非常可惜的一件事。

由於全球化競爭的環境愈來愈激烈，軟體技術更新的週期越來越短，讓許多優秀軟體還沒有發揮其功能效益，就已經被時間給淘汰。更甚者，猶在開發階段就已胎死腹中，這些情形都讓軟體品質的能量無法被累積。我們常常把軟體比喻成建築物，一個好的建築物可以永世留存，傑出的設計建築物更被當成後世的典範，而不斷地被複製及延伸，或有機會成為一流派。但是一個優秀的軟體，相對於建築物而言，卻有總是曇花一現之憾。

1.3 研究目的

在[3][16] [17]研究裡，對於重整舊有系統至使用現代技術之系統已經有不錯的成果，但對於重整至網路服務架構的研究方法，並沒有提出適當的方式及驗證是否可行的方式。在此篇論文裡根據重整工程流程提出一套方法，來重整一個舊有軟體，希望提供相關經驗及作法讓類似的軟體系統能重新被開發出有用的軟體元件，減少新軟體設計的錯誤及大量資源的投入。在這篇研究中預期可以得到下面幾項目的：

- 建立網路服務環境
- 獲得可再利用的網路服務

- 系統轉移程序
- 另一個形式存在的新系統

1.4 章節安排

論文後面的章節安排分述如下：

第二章　背景知識及相關研究，此章將說明網路服務的相關技術及介紹軟體重整及軟體再利用的概念，並也會對此篇論文所要重整的對象將會做一簡單描述。

第三章　重整方法，此章將提出以網路服務所構成的系統架構，並也會說明整個重整工程的步驟進行的方式。

第四章　案例研究，此章舉出以一個例子執行上章中所提出的步驟，並重新得到系統的使用案例及類別圖以新系統架構製作轉化應用程式。

第五章　此章將會根據論文中所得到的結果，判斷是否此項研究是否有達到此篇論文的目的。

參考文獻列出論文中所使用的資料來源，如文句、圖表等。在之後的附錄中提供論文裡進行過程中所產生的使用案例圖、類別圖及其他文件的詳細參照。

第 2 章　背景知識及相關研究

2.1 軟體重整 (Software Re-engineering)

當過去所開發出來的軟體漸漸的已經不適合現在的環境、硬體設備及技術時，為了讓過去開發軟體時所投資的人力、物力不至於浪費，這時我們就需要做重整工程(Re-engineering)[23]。早期的系統由於硬體能力不強，所以設計人員在設計程式的時候，需要注意到記憶體的利用，因此結構寫得非常簡潔，不容易讓維護人員了解。但是近來電子技術的進步，硬體能力的限制已經比從前的少了很多，且也由於應用環境的改變，為了延續原來程式的使用期限，則必須進行軟體重整，使得系統更容易維護。圖 1 為一重整工程的判斷矩陣，從圖中我們可以可以判斷系統被重整的時機。

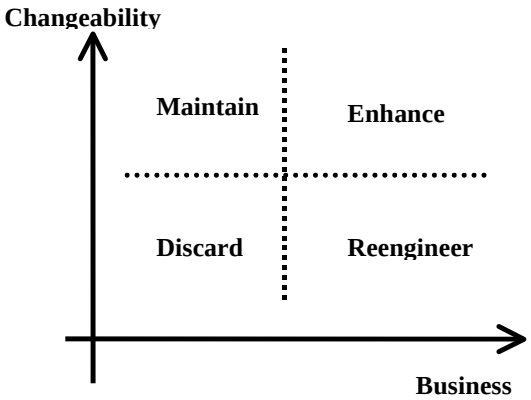

圖 1 重整工程判斷矩陣

傳統上對於軟體系統不敷使用時的處理方式，大致可以分為兩種方式，一是將舊有的軟體系統捨棄，重新開發一個全新的軟體系統，而這種方式的優點是可以避免舊有系統中可能的缺點，擺脫舊有系統中的包袱，同時對軟體工程師來說，等於是重新開發一套系統，無需去瞭解舊有系統中的程式結構，然而這種方式對公司或組織而言，等於是將舊有系統中的投資完全捨棄，同時對使用者來說，也必須重新去學習新系統的使用方式。另一種方式則是在舊有系統的基礎

上，加入新的功能，這種方式的優點是舊有投資可以保留，同時對使用者來說，不必重新適應新系統，減少重新學習過程中生產能力降低的問題，但對於程式設計師而言，就必須去瞭解整個原始程式碼、設計概念及系統架構，增加軟體開發週期的時間。

傳統程式設計方式通常無一致的規則性，往往都是依程式設計師個人風格的不同而有不同的程式撰寫方式，所以若是相關的文件說明不夠完整或與原始程式不符的話，就會造成程式解釋上的困難。因此一般程式設計師面對這種問題，往往不敢更動原有程式核心，通常都是直接將新功能以包裝的方式加入舊有的系統中，但這種包洋蔥似的程式發展方式，可能會使程式中的相似變數或函式愈來愈多，增加系統的複雜度，也降低了系統的效率，同時也會將舊有系統中的缺點完全的繼承，造成日後維護上的困難。

重整工程的步驟是一連串包含分解(Decomposition)及瞭解現在程式碼的逆向工程(Reverse engineering)階段，和重新組譯(Re-interpretation)在逆向工程階段中所獲得的資訊，進而產生新程式碼的順向工程(Forward Engineering)階段的過程。在整個重整過程中，可以有效地將舊有系統的缺點一併消除，並將整個系統結構重新調整以適應新的環境，減少日後維護上的困難，同時對於舊有系統中成熟的可再用軟體元件盡可能地再使用，降低系統開發時的成本。

圖 2 是軟體工程中瀑布式軟體開發模型，正常的軟體開發方式一開始起於需求分析(Requirement analysis)流程得到系統目標；然後經過架構設計(Architecture design)流程規劃出系統的軟、硬體架構；再來系統設計(System design)流程則定義模組中的元件；最後經過系統實作(System implement)來完成整個軟體。上面步驟的順序組合稱為順向工程。重整工程則是在進行順向工程產生新軟體之前，先執行反轉工程(Reverse engineering)，主要的目的是在於回覆系統原來的設計規格及需求規格，使得順向工程時不必在需求分析及架構分析花費過多的時間。因

為一個執行多年的軟體要重新尋得當時的設計分析人員不是一件容易的事。

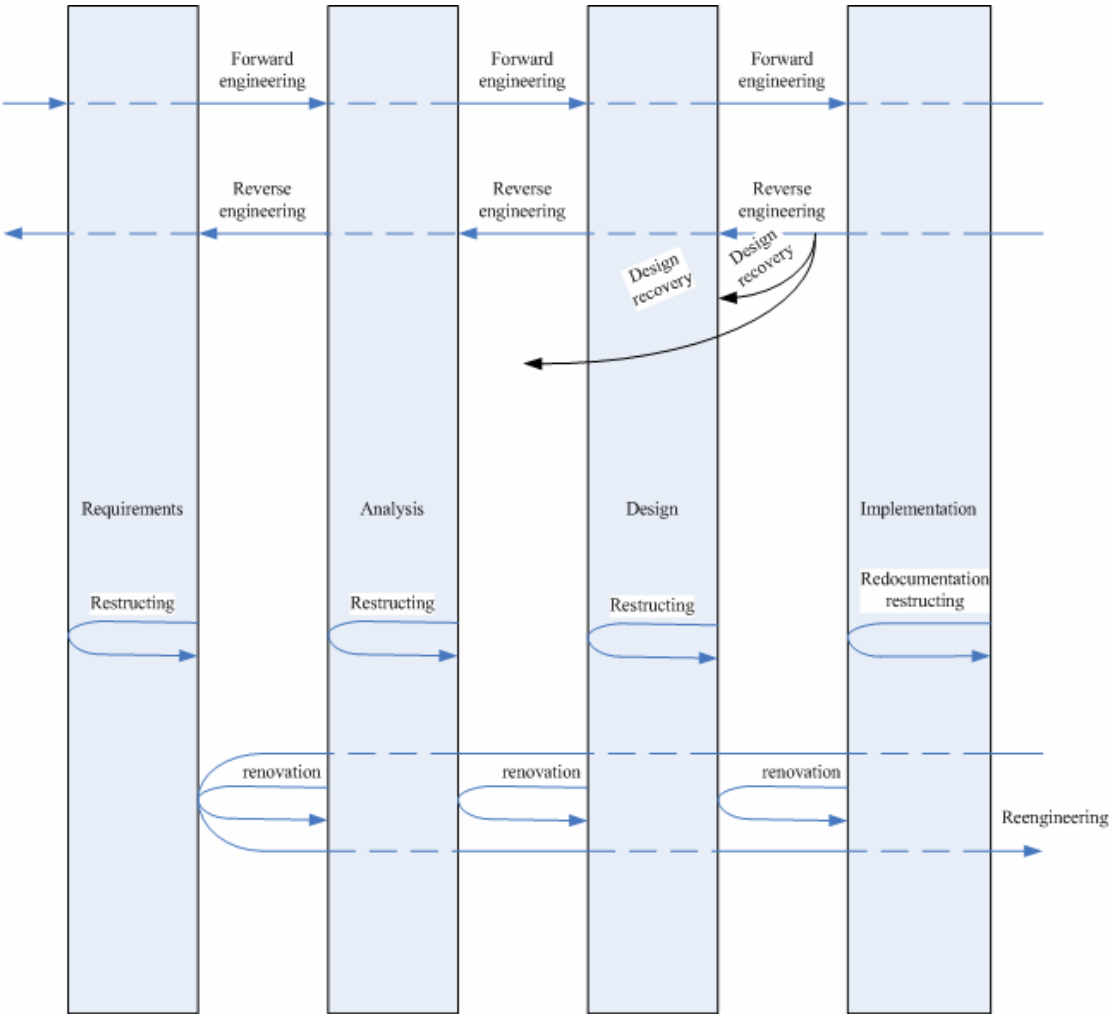

圖 2 The model of forward/reverse engineering

在重整工程中，設計階段過程的重整步驟除了程式碼的重整，就是資料重整。資料重整的目的是由於電腦系統資料存在的方式，從最初以檔案形式存放、具特殊結構檔案型態，如 dbx 檔案，與程式綁在一起的方式逐步形成資料庫的方式。以資料庫形式而言有可分為網路型資料庫、分散式資料庫、物件式資料庫、關聯式資料庫等等。在新系統裡由於資訊技術的更新、系統結構的改變或搭配程式語言的都會影響新系統的資料存放方式。因此資料有時候需要經過轉換，如從非關聯式資料庫轉化成關聯式資料庫。在資料重整部份，又可以分為兩個部份來討論，一是資料內容，另一是資料結構。由於存放資料的空間在轉換前後是無法完全的適合新的資料結構，如 char 對應成 string 型態、integer 位元問題等。資料

結構注重於資料綱要(data scheme)設計重整，而在不同資料庫的轉換尤其重要。

在重整過程中也會遇到其後這些問題，如不適合文件(insufficient documents)、功能重複 (duplicated functionality)、缺乏模組化(lack of modularity)及不適合的分類方式(improper layering) [20] 等，也都是需要在過程裡一併改善的地方。

2.2 軟體再利用 (Software Reuse)

軟體再利用(software reuse)概念[6][22][12]是對於品質與生產力的追求提供了一個具有吸引力的遠景，希望有朝一日將不需要軟體發展人員撰寫程式、設計模型及分析問題，也就是能夠重複利用已經發展過的軟體元件來建構所需要的軟體系統。

在傳統的軟體發展過程中，整個軟體發展流程大部份是針對該特定目標，整體地計劃並進行，所設計出來的架構、模組，及所實作的程式碼皆是包含於該軟體中而成為一個封閉完整且邏輯獨立的單元。而所得到的軟體雖然採用整體角度來設計，使其具備較高的內聚力(cohesion)及強固性（robustness），但是該軟體許多可重用的部份，如設計方法、程式碼及測試資料等，卻被隱藏在軟體內部，甚至是糾結在一起。這使得即使是屬於同一公司或同一領域的軟體發展者，也無法在日後有效地重用這些有用的資源，這些可重用資源的閒置無疑是一種浪費，而重複的開發成本更是一大損耗。

重用現有軟體以達到快速地建置新的軟體系統是目前研究領域與業界裡最重要的軟體發展方法之一[9][1]。相較於從軟體之規劃、設計、發展全部重新做起的方法，重用已開發完成的軟體將可大幅地降低開發成本及時間，並提昇軟體產品的品質[6]，更可間接地降低開發過程中產生的可能風險。軟體重用的方式

來發展軟體其主要的理論基礎在於，多數採用軟體所解決的問題實際上都一再地重複發生，尤其是一些基本應用，或是較小的問題，例如排序、搜尋、檔案處理等，而某些較特殊或較大型的問題，在經過分析與細分（decomposition）之後，通常會發現這些特殊或較大型的問題事實上也是由許多以前已解決過的問題所組成，只有其中一小部份是全新的問題。因此用以解決曾經發生過的問題的方法、相關的文件、設計及程式碼都可以再被使用，而不需再花費心力重新來過。

2.3 Web Services

網際服務(Web services) [33]是目前實現軟體重用概念最新的技術。在圖 3 中顯示我們可以任意挑選適當的軟體元件組合成唯一新的應用程式。重用軟體元件除了讓我們能減少開發新元件的風險，更讓原本的元件可以不間斷的改善而得到更好效能及可靠度。

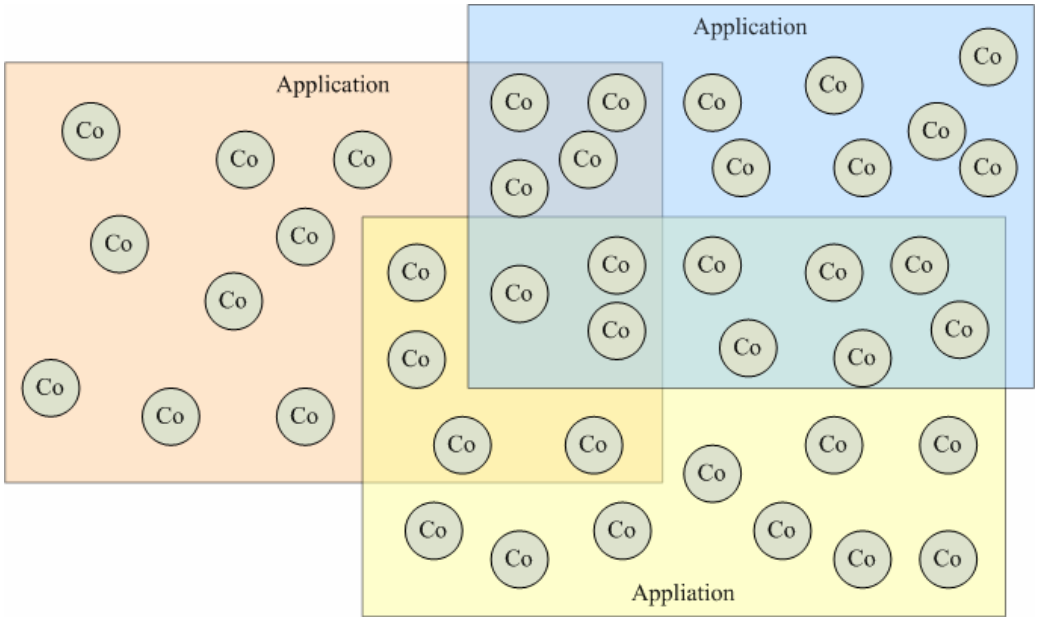

圖 3 Component-based System

網際服務可以做為整合組織內部的應用程式及整合企業夥伴之間的電子訊息溝通應用程式，而它能達到如此效果的方式是網際服務平台獨立及分散式元件

的特性，當開發者把組織內部的應用程式轉換成網路服務，應用程式變成分散式元件的組合。這些不同的分散式元件可不相關於運作平台互相結合成新的應用程式。由於專注於某一領域的軟體元件只有唯一的存在，因此可以不斷地被改進與應用，也使得結合軟體元件而成的新應用程式具有一定的品質(quality)。網路服務可以達到元件共享實現了軟體重用中的程式碼重用目的。

我們也可以利用網路服務來溝通組織之間的訊息，在真實世界裡公司的運作少能單獨而行，常常需要與其他公司合作，其商務報表及收據的往來可想而知是非常地頻繁，為了簡化之間的文件交換動作及人力浪費，而有電子檔案交換(EDI)系統用來溝通公司間的資訊，取代人力完成公司之間資料交換的動作，可是電子檔案交換系統由於建造的成本及所帶來的效益並無法讓所有的公司都能投入，造成公司電子化的一個阻礙。但是網路服務架構的系統可以用少量的成本就可以在原網路系統上建置，它可以運行於 Internet 上，在世界各地都可以輕易地存取，公司可以開發私有對外的網路服務，負責與其他公司分享及交換資料，來簡化頻繁的溝通所浪費的物力及人力成本，如此一來也可以達到軟體再利用上另一個目的－資料重用。

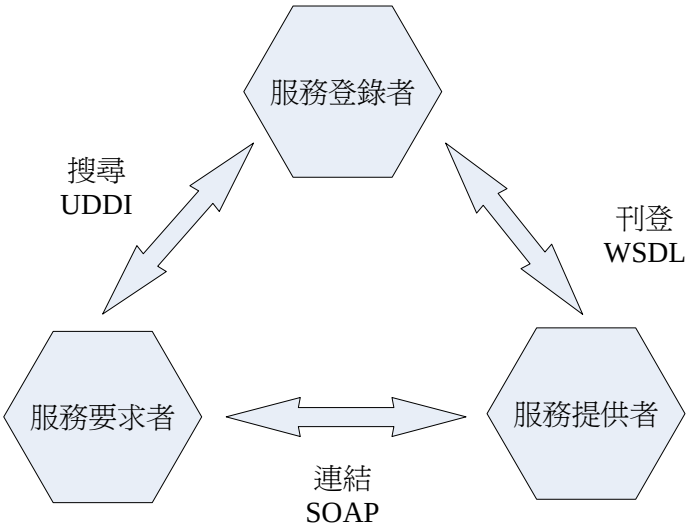

圖 4 Service-Oriented Architecture

以服務的觀點來看網路服務架構可以表示如圖 4，在此架構中有三個角色，分別是服務登錄者(service register)、服務提供者(service provider)及服務要求者(service requester)。服務要求者是向外界要求網路服務的前端使用者；服務提供者是把內部服務給外界的資源提供者。服務登錄者則是一方面提供網路服務的資訊給服務要求者，另一方面則是讓服務提供者註冊本身所能提供的外部服務功能至服務登錄者。以一個實際例子來說，當我們住家裡要安裝一桶罐裝瓦斯，在本身無法得知任何關於瓦斯行任何資訊情況下，我們會利用電話簿來查詢住家附近的瓦斯行。在這個例子裡, 我們住家就是服務要求者，電話簿相當於服務登錄者, 而瓦斯行則是服務提供者。

在此架構中，我們最常擔任兩個角色－服務要求者及服務提供者。服務要求者如果有一個服務要求，可以先跟服務登錄者搜尋所能提供的此項網路服務的所有服務提供者。在網路登錄者回傳此部分訊息後，服務要求者更進一步選擇其中之一最適地服務提供者，透過網路連線利用 SOAP (Simple Object Access Protocol)[33]包裝訊息(message)互相溝通資訊。當為服務提供者則在製作新的網路服務時，用 WSDL (Web Service Description Language)[34]描述所持有的網路服務並註冊至服務登錄者，讓之後有需要此服務的需求者能索引到此服務，提供再利用的機會。服務登錄者就像是一本電話聯絡簿，載明服務提供者的通訊方式、地址、服務內容等訊息，讓要求者能一目了然。

Web service 所提供的技術主要由 WSDL、SOAP、UDDI (Universal Description, Discovery, and Integration)[32]三個部份組合而成。圖 5 是網路服務架構堆疊圖，從圖中可以較清楚地了解這三項技術在網路服務架構中所呈現的意義。首先，SOAP、WSDL 及 UDDI 內部資料定義都是基於 XML、DTD、schema 的技術所發展出來，所以內容格式皆以 XML 形式呈現。SOAP 在其中所扮演的角色為訊息包裝者，處理資訊在服務要求者及服務提供者間的流通，由於採用 XML 的描

述方法，使得可以獨立於平台之外，任何支援 SOAP 的技術都可以來解譯其包裝的資訊。這也使得服務要求者及提供者所在的平台並不會影響網路服務的運作，所以網路服務可以在異質平台上合作。這個特性也是促進網路服務能成為新一代分散式運算機制的原因之一。SOAP 規格的介紹將在 2.4 節中來更進一步討論。

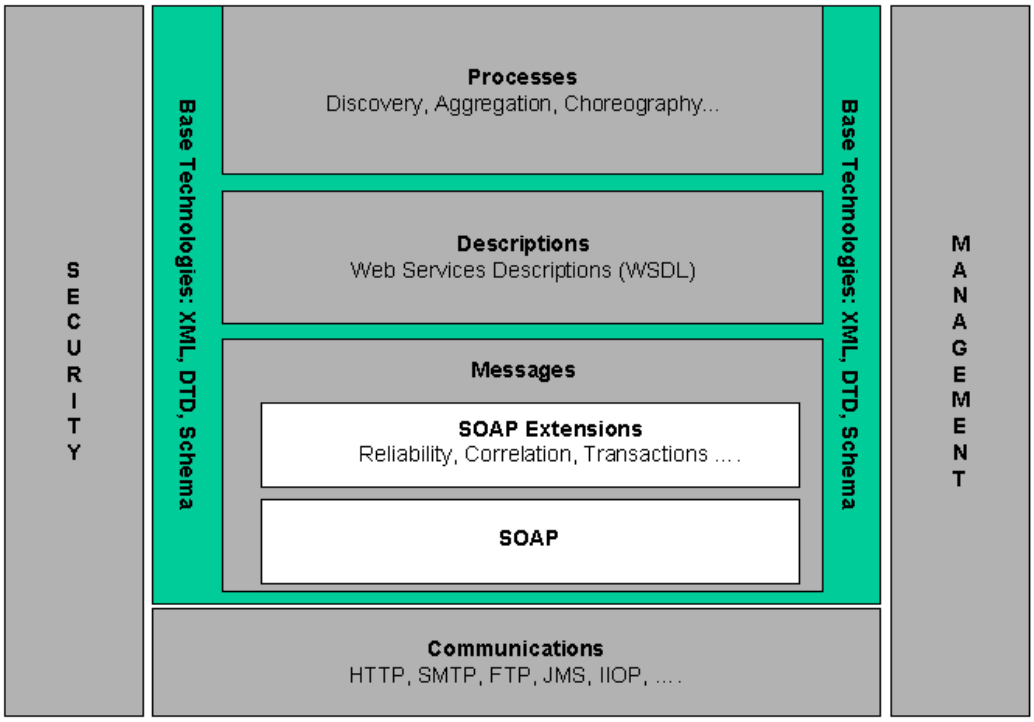

圖 5 Web Service Architecture Stack[33], from IBM

WSDL 則是一種描述網路服務的技術語言，主要用來描述網路服務位在哪裡、如何溝通、及有什麼樣的服務內容等資訊。一樣是以 XML 形式來呈現及定義，而此部份更詳細的資訊將在 2.5 節中介紹。UDDI 在圖 5 中並沒有被明顯的表現出來，其擔任的位置在 Processes 此區塊中。它是一個交易平台，用來註冊網路服務及搜尋網路服務，並建立讓服務提供者及服務要求者資訊傳遞的平台。除此之外，在圖中我們也得知 web service 的管理及安全，在架構中並沒有是沒友直接規範，如果要確保交易通訊安全，則需要透過網路加密技術對通訊(communication)或訊息(message)封裝提供安全機制才能確保資訊不會外露。

表 1 分散式元件比較

	CORBA	DCOM	EJB	Web Service
Message format	Private format	ORPC	RMI-IIOP	SOAP
Component describtion	IDL	Server stub	Remote/Home interface	WSDL
Component register	ORB	SCM	JNDI	UDDI
Component deployment	CORBA server	COM server	EJB container	Web service application server

Web service 是一種軟體元件，其架構更是一種分散式運算機制，在與其它目前現流行的分散式技術比較,在訊息格式方面利用 SOAP 包裝的訊息具平台獨立特性，不會像 DCOM 及 EJB 一樣，兩邊的訊息傳送皆需要同樣的元件，也不像 CORBA 其格式是私有格式不能共享。在元件描述上，WSDL 跟 IDL 一樣用來描述元件本身的特性，而不同的地方是在於 WSDL 是利用 XML 定義出來。元件描述在 DCOM 及 EJB 兩種軟體元件則是比較單純，傳接雙方都是相同元件，所以只要知道元件位置及元件介面就可以直接呼叫，但是其缺點是難以跨平台呼叫。UDDI 及 ORB(Object Request Broker)都是一種中介軟體(middleware)，用來溝通要求者及提供者。ORB 可以接受用 IDL 來定義元件介面的協議，如同網路服務之於 WSDL。DCOM 架構下，Client 遠端啟動 Server 是透過區域 SCM 與遠端 SCM 協商而成，所謂 SCM 乃 Service Control Manager 的縮寫，其為支援啟動程序(activation)的 COM 元件，而當 Server Object 繫結至 Client 後，雙方便不再需要 SCM 的支援。其運作的原理也跟 UDDI 大同小異。JNDI (Java Naming and Directory Interface)用來找尋一個特定名字所對應的物件，把物件名字與實際位置、服務、訊息及資源連接起來。ORB、SCM、JNDI 及 UDDI 簡單說起來就是溝通服務者及被服務者兩端，只是實做技術及服務對象不同的差別。最後，各類型元件伺服器為一置放元件的容器，只因所製放元件不同而會有差異，但是一種元件伺服器只能服務一種類型元件，不能交錯置放。我們可以發現網路服務優於其他分散式機制的地方，在於除了可在異質平台上溝通外，更因簡單且開放的系統架構，讓一般程式設計人員也能快速實現網路服務。

2.4 SOAP

SOAP (Simple Object Access Protocol)[33]的概念起源於 Dave Winer 的 XML/RPC，目的為了開始研發一個彌補分散式元件 COM 不足的標準時產生的。當初研發的動機是由於 COM 的機制對系統資源索求過重、不夠輕巧，難以應付未來需求。所以其主要目的透過 XML 的方式建立遠端呼叫的方式，為分散式環境下的程式和系統之間，提供了一套簡單的訊息傳呼協定。在企業軟體日趨網路化、分散化的環境下，不論是企業內或跨企業的資訊整合，以及新一代整合工程所偏好的多層式(N-tier)運算架構[1]，伺服器對伺服器、程式對程式間的互動均大量增加，而透過 SOAP 在此類系統互動的能跨越程式（processes）、伺服器軟硬體及企業的限制。

其他知名的 RPC 機制包括 CORBA、Java RMI 及 COM，Java RMI 是 J2EE 及其主角 EJB (Enterprise Java Beans) 框架的重要基礎科技；COM/DCOM/COM+ 則是 Windows DNA 賴以發展的命脈。這些機制對平台或程式語言一致性，都有一定的要求：RMI 用在 Java 語言，COM 則是用在 Windows 作業系統，CORBA 則需要適當的運行環境和平台來支援。過去企業為達到內部資源整合(EAI)，其應用系統(Application)或許統一採用 DCOM、CORBA，再搭配某種 message broker 的傳訊機制，並透過該 broker 作傳遞樞紐即可達成。然而，網路所帶來的資訊革命及電子商務，加上企業併購風潮的盛行，促使企業對系統整合的需求日殷，異質性系統間的互動亦隨著大量增加，CORBA、Java RMI，COM 等 RPC 機制，開始顯得僵硬、彈性不足。而這正是 SOAP 優於其他機制最大的好處。

SOAP 利用 XML 純文字的特性，提供一套機制，以 XML 來包裝方程式呼叫

[1]如微軟的 DNA 及昇陽的 J2EE。

和訊息。由於採用了 XML，SOAP 順理成章地繼承了許多 XML 的優點，可輕易透過 HTTP、SMTP 等網路上最常使用的通信管道來來挾帶，更能穿越企業的防火牆(firewalls)，還可透過 SSL、S/MIME 等機制加密，安全性高。透過 XML 來傳訊，還有一項更大的優點是 CORBA、Java RMI，及 DCOM 這些以專屬 binary 格式傳送資料所不及之處，那就是對程式語言、作業平台的獨立性。由於是純文字 XML 格式，SOAP 訊息可由任何一種程式語言所產生，被任何程式語言、甚至肉眼所解讀，而這也是目前網路服務時代所迫切需要的。

SOAP 主要由四部分組成：

1. SOAP envelop，定義了一個整體的表示框架，可用於表示在訊息(message)中的是什麼，誰應當處理它，以及這是可選的還是強制的。
2. SOAP encoding rules，定義了一編碼機制用於交換應用程式定義的資料類型的實例。
3. SOAP RPC representation，定義了一個用於表示遠端程序呼叫和回應的約定。
4. SOAP binding，定義了一個使用底層傳輸協定來完成在結點間交換 SOAP 信封的約定。

下圖 6 為一段簡單的 SOAP 訊息，這是我們利用 SOAP 發送附著名字的訊息給伺服器，伺服器處理回傳的 SOAP envelop 訊息。

```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <SOAP-ENV:Envelope
3       xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
4       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5       xmlns:xsd="http://www.w3.org/2001/XMLSchema">
6       <SOAP-ENV:Body>
7             <ns1:sayHiResponse
8                   xmlns:ns1="urn:HelloWorld_SOAPService"
9                   SOAP-ENV:encodingStyle="http://schemas.xmlsoap/org/soap/encoding/">
10            <return xsi:type="xsd:string">Hello my friend, Gaiba! Glad to see you!</return>
11            </ns1:sayHiResponse>
```

```
12     </SOAP-ENV:Body>
13 </SOAP-ENV:Envelope>
```

圖 6 SOAP 訊息

第 1 行宣布此文件的格式；第 2 行至第 13 行是 SOAP envelop 定義區塊；第 3 行至第 5 行宣布此區塊的環境及語法規則來源；第 6 行至第 12 行則是此訊息重要資料區，以標籤<SOAP-ENV:Body> </SOAP-ENV:Body>來包含資料。在第 10 行：

<return xsi:type="xsd:string">Hello my friend, Gaiba! Glad to see you!</return>

是網路服務回應回來的訊息，我們可以直接明白其意思 ，傳回一個字串”Hello my friend, Gaiba! Glad to see you!”，非常容易被程式及程式設計師所了解。從這裡我們可以更清楚地知道 SOAP 是一個開放的網路標準，以 XML 格式包裝訊息資料，並且可以在任何程式語言及平台上運作。

2.5 WSDL

WSDL (Web Services Description Language)[34]是用 XML 格式來描述網路服務的語言，可以用來找尋網路服務及定義服務介面，從節 2.1 表 1 中可以知道它如同其他分散式環境之下的物件描述語言，其定位成一種網路服務的 IDL。一個用來定義網路服務的 WSDL 檔案會包含下面五個主要元素(elements)，分別為 portType、message、types、binding 及 service 等，其功能定義如表 2 所示。

表 2 WSDL 元素

元素	定義
<portType>	描述網路服務的抽象操作介面
<message>	網路服務的訊息

<types>	網路服務的資料型態
<binding>	網路服務的溝通協定
<service>	定義服務的位置

利用下面圖 7 的例子來解釋元素的意義，這個例子是由一個 JAVA 程式轉化成包裝成網路服務時所產生的 WSDL 檔案，以<wsdl: portType>標籤所含括的部份說明此網路服務中有一個 operation 叫做 Hello。此 operation 被分成兩個部份來描述，分別為 HelloRequest 及 HelloResponse，前者接收參數資料，後者傳回運送結果。在<wsdl: message name=”HelloRequet”>標籤中表示 HelloRequest 有一個屬性，其屬性名稱為 name 且格式是字串(string)型態。而<wsdl: message name=”HelloResponse”>籤中表示 HelloResoinse 有一個屬性，其屬性名稱為 HelloReturn 且格式是字串(string)型態。至於從<wsdl:binding name="HelloSoapBinding"> 標籤中的<wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />可以看出程式是利用 RPC 來傳送資料的方式。整份檔案架構裡，在<portype>描述服務的抽象介面，如訊息與操作。而具體訊息與操作實作則在<binding>說明。經過 WSDL 我們可以知道服務是什麼、如何存取服務及服務在何處的資訊，如服務所提供的操作、存取操作的資料格式和通訊協定及其位址。

```
<?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions targetNamespace="http://localhost:8080/axis/Hello.jws" ...>
+ <wsdl:message name="HelloResponse">
      <wsdl:part name="HelloReturn" type="xsd:string" />
      </wsdl:message>
+ <wsdl:message name="HelloRequest">
      <wsdl:part name="name" type="xsd:string" />
      </wsdl:message>
+ <wsdl:portType name="Hello">
   - <wsdl:operation name="Hello" parameterOrder="name">
      <wsdl:input message="impl:HelloRequest" name="HelloRequest" />
      <wsdl:output message="impl:HelloResponse" name="HelloResponse" />
```

```
      </wsdl:operation>
      </wsdl:portType>
+ <wsdl:binding name="HelloSoapBinding" type="impl:Hello">
        <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
       +<wsdl:operation name="Hello">
       …
+ <wsdl:service name="HelloService">
…
</wsdl:definitions>
```

圖 7 WSDL 的例子

2.6 UDDI

UDDI 全名為 Universal Description, Discovery and Integration[32]，其被定義為「主要架構於 XML 技術之上，其設計目的係為構築條列電子商務表單之單一標準。藉由此一標準，業界可列述電子商務網站、聯絡資訊、付費選項與 B2B 相容之各類電子商務表單，並可增進線上異動之效率，且促使業者更易於透過網際網路搜尋引擎尋找其他相關資源。」[22]這是針對 UDDI 的功能所能達到的目的而言。

此外 UDDI 也可以是個網路服務的註冊中心(Registry)，這是因為其中包含了兩個部份功用－服務描述及服務尋找規範。當組織以網路服務來發展自我的應用程式時，需要去了解哪裡有可以利用的網路服務及此服務是否適合自己來使用，這時候就需要透過 UDDI 的幫助來達到這個目的。UDDI 的配置也可以分為兩個部份，一是公用的，另一是私用的。公用的 UDDI 為一開放的服務，提供任何人於任何時間及地方可以註冊個人的網路服務給任何人來使用，如 IBM、微軟等公司所設置 UDDI 伺服器；私用的 UDDI 則為組織內部所使用，個別組織可以架設屬於自己的 UDDI 註冊中心，在裡面建構不分享的網路服務。

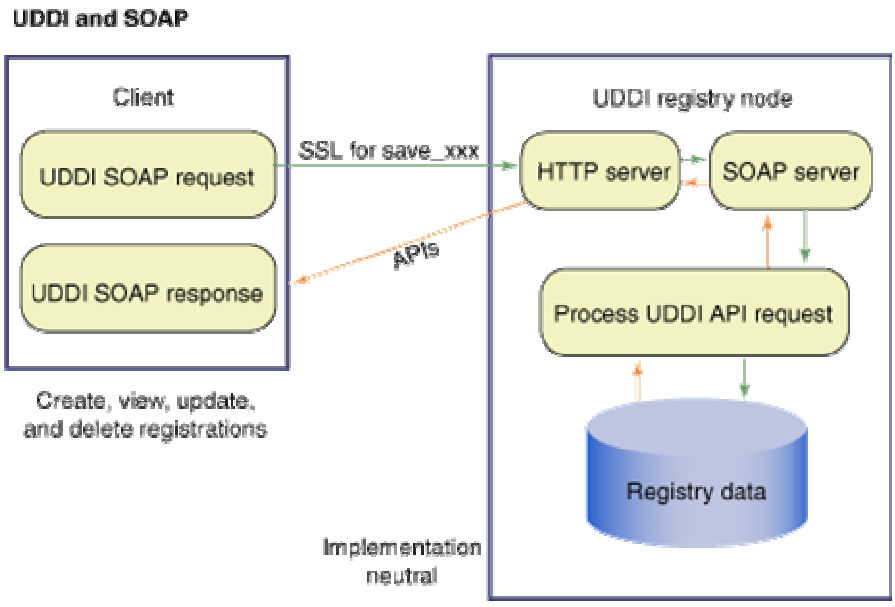

圖 8 UDDI and SOAP

上圖 8 展示客戶端(Client)與 UDDI 註冊中心之間產生、檢視、更新及刪除註冊訊息動作間的關係。在客戶端存在兩個模組 UDDI SOAP request 及 UDDI SOAP response，前者負責傳送資料，後者接收資料，兩者都以 SOAP 封裝資料並可用 SSL 來增加傳送的安全性。在 UDDI registry node 中 HTTP server 接收訊息，判斷為 SOAP 訊息後讓 SOAP server 接手負責解譯此訊息。SOAP server 解譯後判斷此為 UDDI 之註冊資料，透過 UDDI API request 來處理此資料，來加入、修改、或者刪除於資料庫中的註冊資料。在節 2.1 的圖 4 中，這個動作主要發生在服務登錄者及提供者間，至於服務要求者如何跟服務登錄者查詢所註冊的網路服務，我們可以從下圖 9 看到其工作原理。

圖中第一步，軟體公司、標準組織及程式設計師公佈不同界面描述規範，此時建立分類的方式及服務可再用定義。第二步，公司發布所提供的服務及描述訊息於 registry 上，這時候豐富了服務的選擇性。第三步，UDDI registry 給每個介面描述及公司登記資訊一個 UUID (Unique Universal Identifier)並存在網路註冊中心裡。UUID 用來辨別所有的實體都是唯一存在，相當於身分證的區別作用。第四步，公司透過電子交易市場(e-marketplace)、搜尋引擎及商業程式尋找所需的

服務。此時，公司可以根據需求找尋相關的網路服務。最後，可以簡單整合形成網路上應用工具或服務。

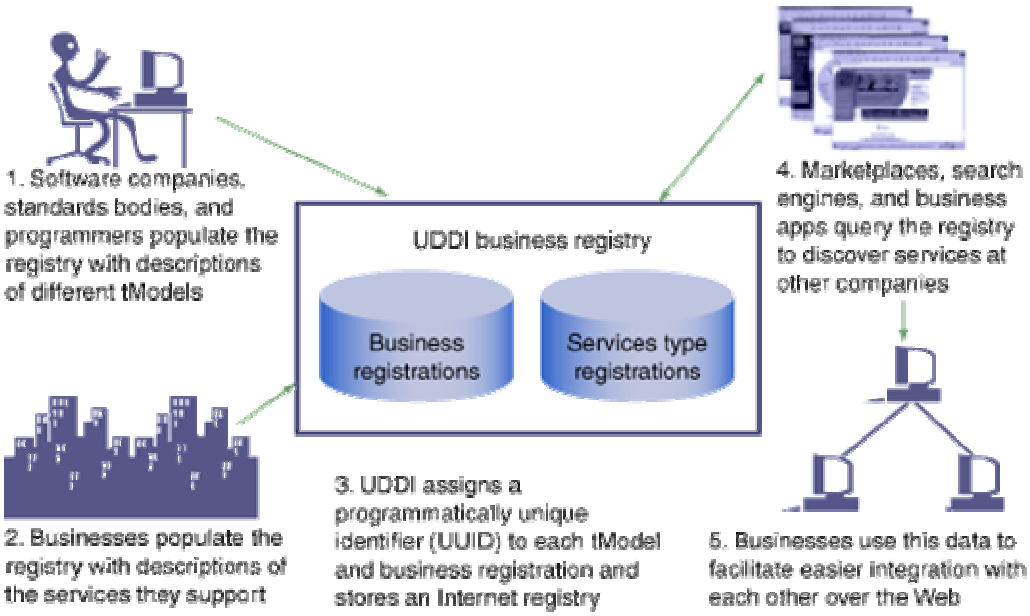

圖 9 UDDI 工作原理 (source from IBM)

為了讓服務提供者所提供的網路服務能快速地被尋找及利用，UDDI 中定義了內部的分類方式。UDDI 對於服務的分類規範目前有數種分類方式，在下表 3 中我們比較並列出目前公用 UDDI registry 所支援的分類方式。當註冊網路服務時，則便需要從各類方式中選擇適當的放置位置。

表 3 UDDI 服務分類法

分類法	描述	伺服器支援
NAICS[2]	北美行業分類法 1997 版，如製藥、光學用品	IBM, MicroSoft
UNSPSC[3]	通用標準產品和服務代碼	IBM, MicroSoft
ISO-3166-2003	ISO 3166 地理分類	IBM, MicroSoft
NAICS02	北美行業分類法 2002 版	IBM, MicroSoft
UNSPSC6	通用標準產品和服務代碼，第 6 版	IBM, MicroSoft
UDDITYPE	UDDI 分類，如 transport、wsdlSpec	IBM

[2] North American Industrial Classification System
[3] Universal Standard Products and Services Classification

SIC[4]	標準行業分類	MicroSoft
GCS/GEO[5]	地理分類系統，以國家或區域、州、省及地區的代碼來分類。	GCS: MicroSoft GEO: IBM

2.7 Office Organizer

Office Organizer 是『華人數位股份有限公司』在 2001 年開發的辦公司應用軟體，提供許多模組支持公司一般運作需求，其架構在網際網路上具跨平台的彈性，能夠滿足不同階層企業的作業需求。它是完全地 JAVA 開發，前端使用網路瀏覽器(web browser)來顯示資料與互動操作，後端則是把資料放在關聯式資料庫(relational database)。Office Organizer 配置在應用伺服器上，透過執行內部 Enterprise Java Bean (EJB)元件來服務前端要求。

Office Organizer 提供的模組，如圖 10 所示，共可以分為規章制度管理系統、人力資源系統、財產管理系統、會議排程系統、文件管理系統及簽證系統。細分六大功能模組，在每個模組下又有數個較小的模組，規章制度管理系統又分成公司組職、上下班時間及公司規則三個模組。人力資源系統可以分成人事資料與請假管裡兩個部份。財務管理再分成資產管理及庶務管理。會議排程管理包括會議排程及會議室管理兩個模組。文件管理系統主要有檔案管理。簽證系統則有簽到系統。系統總共分為六大功能模組及十一小功能模組。

根據系統使用情形及使用者說明書中，得到系統網路架構圖如圖 11 所示，系統的運作方式是將此應用軟體配置在應用伺服器(Application server)上，讓使用者可以從前端網路瀏覽器(web browser)經過 Internet 或 Intranet 存取應用伺服器上的 Office Organizer 服務，根據使用者的要求，配合資料伺服器上所提供的資料運算處理後回傳資訊給前端使用者。

[4] Standard Industrial Classification
[5] Geographic Classification System

圖 10 Office Organizer 功能模組

從實體環境架構看來，我們可以知道應用伺服器需長時間且大量地處理前端使用者要求，所以常常會有資源不足而必需定期清理內部資料及造成管理人員的不便。因此必須有一套方式或機制能用來分散應用伺服器的負擔。

另一方面從系統內部架構來看，此系統也有個問題，由於每家公司都有不同的功能需求，使得系統需要客製化才能符合客戶需求，而客製化常需要重新編譯原始程式碼，此項動作容易造成一份軟體卻有多種版本，致使維護不易。而且，當客戶提出不屬於原本開發之功能套件則也需要重新開發連結才能加入原系統內。種種的不便不只造成應用程式管理人員的不便，也讓後端程式開發人員及維護人員需要在投注大量心力於這軟體上。

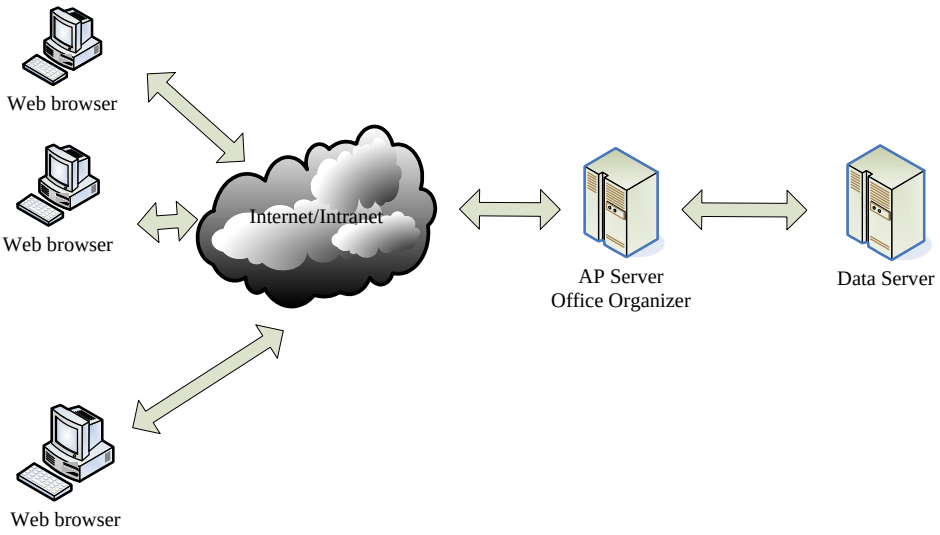

圖 11 目前系統架構

透過此次研究，希望把 Office Organizer 中的軟體元件重整成網路分散式元件，如網路服務形式來分散應用伺服器的負載，取得更好的效能利用並希望系統也因此具有易於修改的特性。此外，分散式的網路元件能重複使用的特性，也將幫助新系統的開發時間可以減少，風險因此而降低。

第 3 章　重整方法

在這個章節裡將把重整步驟分成五個階段，並在 3.1 節描述每個階段所要執行的工作。在 3.2 節中則詳述一個在分散式環境之下導入網路服務的一系統架構及之中各模組間的關係及內容。

3.1 重整過程

整個研究過程裡依據重整工程方法將過程分為五個階段，分別是初期分析、功能分析、應用分析、設計重獲及系統轉換階段，並在之後將各階段所需進行工作敘述如下：

- **初期分析**：初期分析是此研究的第一個步驟，此步驟最主要的目的是在收集後面過程中所有的相關使用資訊，尤其是功能分析及應用分析階段中所使用的資源，所以此階段也可以說是功能分析與應用分析的先行步驟。在此階段中先初步的檢視過整個轉換過程的流程，並概要似地標出需要執行的項目。這個階段可以收集的資料可以有原始程式碼(source code)、需求文件(Requirement specification)、設計文件(Design specification)及系統配置手冊等。從需求分析文件中，可以了解到原始系統設計的目的，得到系統使用人員身分及其需求；從設計文件裡，可以了解軟體架構，幫助我們了解系統設計目標；原始程式碼則是可以了解整個系統執行邏輯及其流程，越多的參考資料將可以縮短系統開發的時間。此步驟研究流程規劃如圖 12 所示。在這個步驟過程裡，所取得的文件是要幫助我們從不同的角度對系統的了解，但是在實際的狀況下除了原始程式碼可以非常順利的取得，其他的文件可能因原來開發的時候就沒有製作或是在不斷的程式改版裡而被廢棄，造成文件遺失。對於

缺少的文件，我們依然需要透過其他的方式來還原文件的內容，例如在缺少需求規格文件時，我們可以透過直接訪問終端使用者來復原需求規格文件；缺少設計規格文件是，我們同樣可以直接透過訪問程式設計人員來復原設計規格文件。但是我們也會遇到原始開發人員都已經離開的情形而現成的維護人員也無法提供詳細的設計內容，則我們便需要透過程式碼分析來間接獲得系統設計規格。

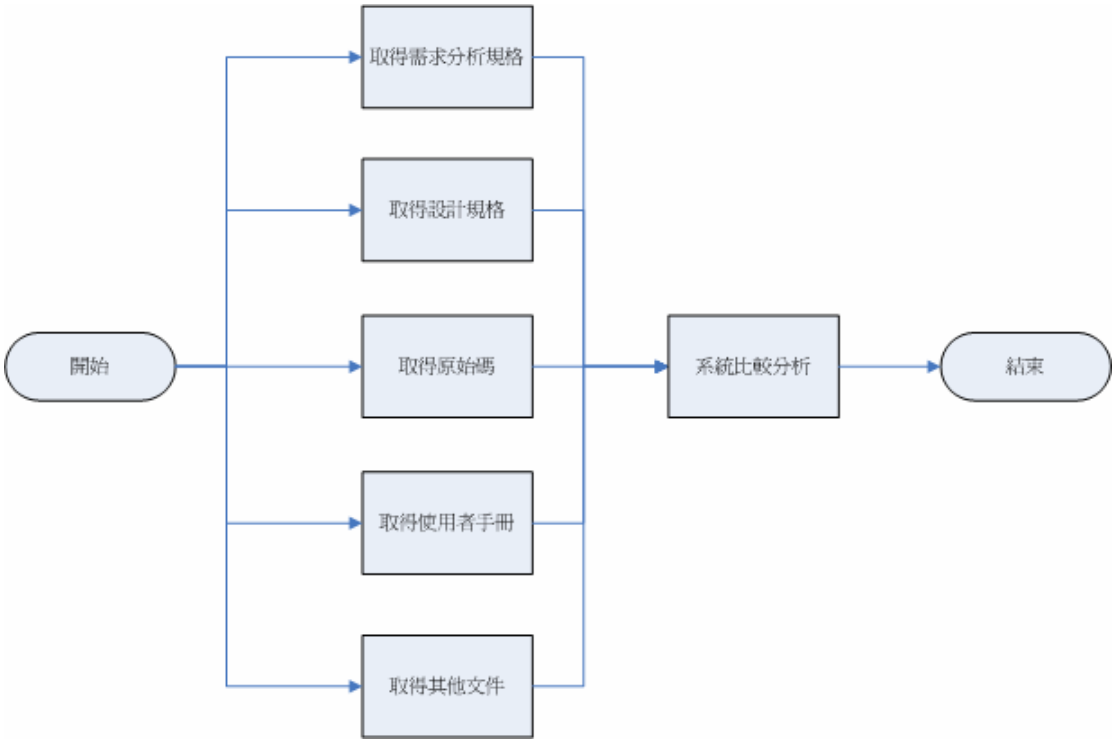

圖 12 初期分析流程

- **功能分析**：這個步驟在研究過程裡也是必要的，此階段裡找出所有的系統內部元件，並希望在進入下一個步驟前，能精確的定義所有元件。當我們得到所有的系統元件，就需要經過分析對照，當開始執行時便可以把不需要的元件提前去除，減少整個過程中時間的浪費。Mayrhauser[12]認為原有系統的改善與維護最困難的地方亦是在恢復遺失的資訊，但是這個動作卻也是決定事後過程能否順利進展的關鍵，所以更詳細的資訊收集是不可忽略的。此步驟研究流程規劃如圖 13 所示。在圖 13 裡，

在上一個階段裡，我們可以根據系統設計規格及程式碼把原始程式裡面的函式及元件分類成三大類：

- 資料處理元件
- 企業邏輯元件
- 使用介面元件

若函式及元件是與資料處理相關，如檔案輸入/出、資料庫存取等，歸類其為資料處理元件 (data access components)；若函式及元件主要負責計算，如薪資計算、資料統計等，則歸類為企業邏輯元件 (business logic components)；此外提供與使用者操作的介面，則是歸類成使用介面元件 (use interface components)。

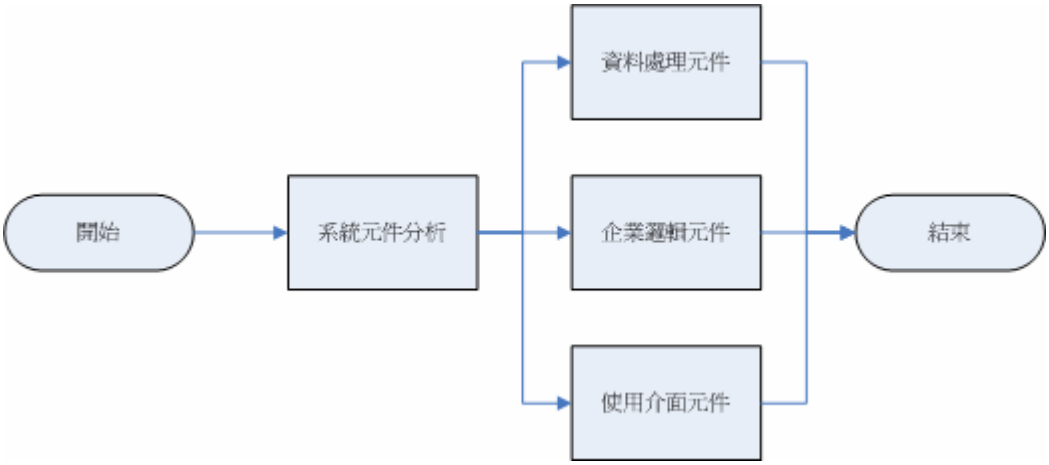

圖 13 功能分析流程

- **應用分析**：在早期的研究[6][8]裡已經有提供許多門徑(approach)及工具來成功地獲得資料流程圖(data flow diagram)、控制流程圖(control flow diagram)及其他相關資訊。可是這些方式尚不能從原始程式碼分析中得到高階(high-level)的資訊，使用者仍需要其他輔助才能進一步深入系統。在此這階段之前利用自動工具輔助及人工方式加強系統了解，此時最重要的事必須分析是否此時系統元件是否已經有能力應付新系統環境需求，如認知新系統設計架構及使用技術等。此階段主要的任務便是找出任何可以在重整過程中增加改善系統的因素，使得系統品質可以被更進一步地增進。在此步驟裡，除了刪除對新系統沒有作用的元件及函

式外，我們也在進行函式及元件應用分析的同時，找尋函式及元件被轉換成網路服務的可能性。我們依據軟體重用及網路服務的特性定義下列規則，只要符合下列規則便可以將原始對象轉換成網路服務：

- 是企業邏輯元件但無關於流程控制及特定物件產生；
- 為資料處理元件。

此步驟研究流程規劃如圖 14 所示，其中可適性分析的目的是去除在未來轉換後系統中不必要存在的元件。而元件轉換分析的目的則是針對留存的元件進行更進一步地分析，得到可以轉換成網路服務的元件並定義其表現方式。在這個過程裡，還內含一些細部流程，就是判斷當函式及元件在可轉換成網路服務時，可以先到公開的網路服務註冊中心尋找已經被製作完成的服務是否可以重複使用的資源，在沒有適合的網路服務之下才需要自己製作網路服務程式。

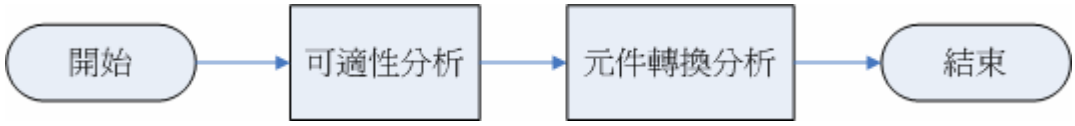

圖 14 應用分析流程

- **設計重獲**：設計重獲(Design Recovery)[4]這個過程是把前三個步驟所得到的資訊加以整合並對於新的系統提出新的結構，此階段的重點是針對完成的系統作新的設計規格書，描述新系統設計的元素，如系統元件描述、使用技術介紹、系統應用說明、系統介面使用方法、系統架構呈現等等。此步驟研究流程規劃如圖 15 所示。從這個階段之前我們進行的是重整過程中反向工程步驟，此階段之後則是正向工程步驟。在這個階段裡，我們要重新描述新的系統，所以在舊有的基礎上，增加各類文件的對新系統的描述及定義。

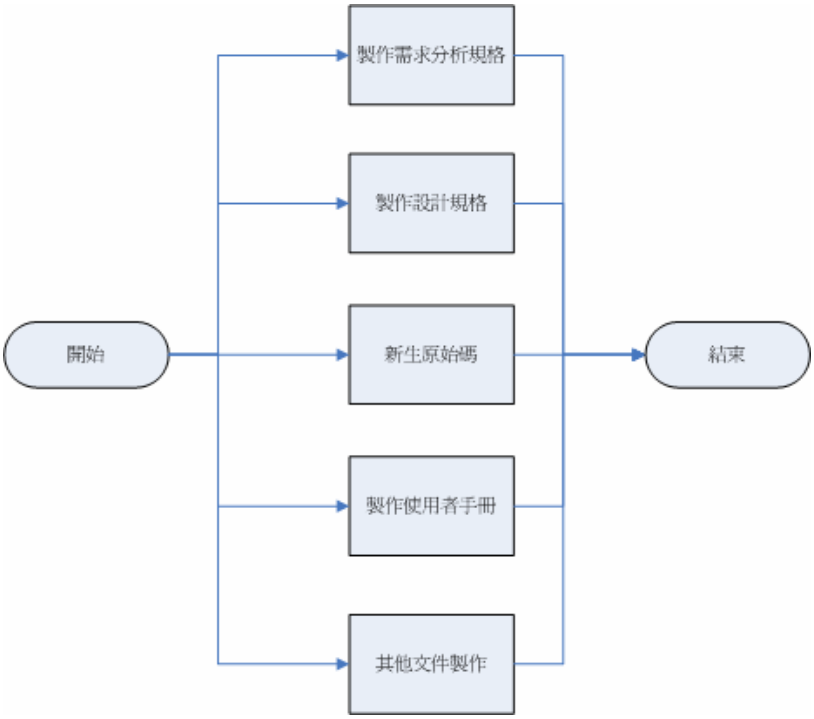

圖 15 設計重獲流程

- **系統轉換**：在此步驟開始針對程式來做改善，加入新要素。在前四個步驟注重分析，此階段之則是注重系統的呈現，將系統轉換成為另一個更好形式的系統，使得更容易操作及維護。在這轉換過程裡分為內、外部的轉換，內部的轉換意指元件的轉換，外部的轉換則是環境的轉換。此步驟研究流程規劃如圖 16 所示。

圖 16 系統轉換流程

初期分析步驟找出系統所能達到應用的範圍；功能分析步驟分析得到系統中組成元件及其用途；應用分析步驟則探討對於資料及程式改善的方式；系統轉換則把系統改變，形成另一個新形式的系統；最後設計重獲步驟主要則是回覆系統設計規格，例如使用 UML 中的案例圖(Use case diagram)及類別圖(class diagram)來表現系統需求及內部設計架構。下一節中將展現系統架構，配合此五個步驟。

3.2 系統架構

建立以網路服務為基礎的應用程式，首要的是其軟硬體使用環境，在新系統統硬體架構可分成三個部份，分別是前端使用者(Client)、中端應用伺服器(Application Server)及後端資料伺服器(Data Server)。後端資料伺服器職司提供資料給中端伺服器及做為存放系統資訊的集散地，所以這邊可以是一個資料庫管理系統(Database management system, DBMS)；中端應用伺服器為接收前端使用者要求，並提供要求的解決方式，而包含此解決方式的便為網路服務，所以此處存在有 HTTP 伺服器(HTTP Server)收取使用者要求，及 SOAP 引擎(SOAP Engine)提供網路服務的動力；前端使用者則可以直接透過瀏覽器(Browser)來操作及使用應用伺服器所提供的功能，如圖 17 所示。

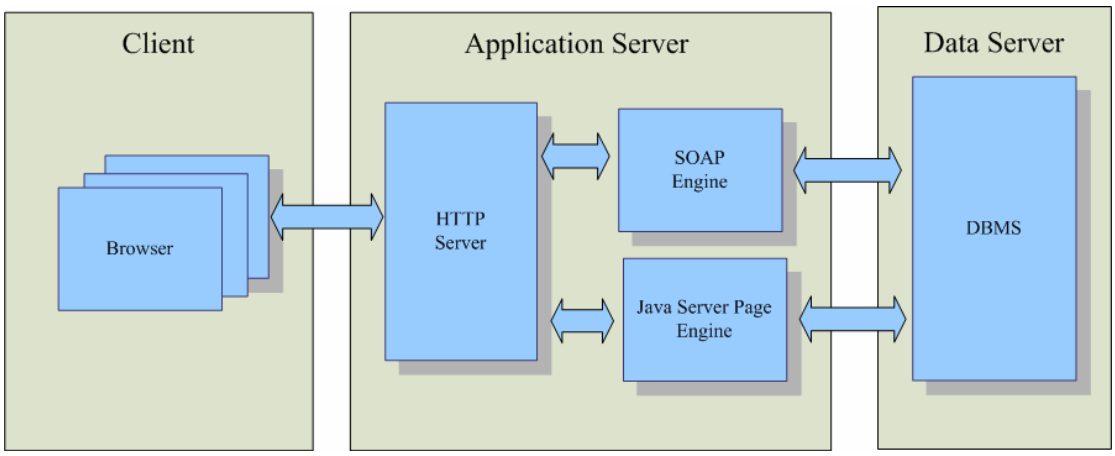

圖 17 系統概念架構

在圖 17 裡中資料伺服器裡是由 HTTP server、Java Server Page Engine 及 SOAP Engine 三塊所構成，其中 Java Server Page Engine 模組是可有可無，其判斷標準端看是否此應用伺服器需要提供 Java 伺服器網頁服務作為使用者操作圖形化介面，如果不需要則此模組在網路服務網路系統架構則不需要被建立。

在系統設計上，利用 Model-View-Controller (MVC)架構的概念，用來配合網

路服務提供基礎資料的功能，也可以把系統分成三個部份，分別為商業邏輯(business logic)、資料及顯示邏輯 (presentation logic)提供一個彈性的架構。在此 MVC 架構裡，Model 顯示一個應用程式的資料，如可利用網路服務來當作資料來源；View 可以讓系統在同樣的 Model 下有不同的表現形式，如 JSP 及其他 stand-alone 程式做為使用者介面；Controller 的功能則控制 Model 內的資料並且使資料在適當的時候能被展現在 View 裡。這樣的設計方式使系統開發可以是彈性的(flexibility)、可管理的(manageability)及責任分離，如圖 18 所示。

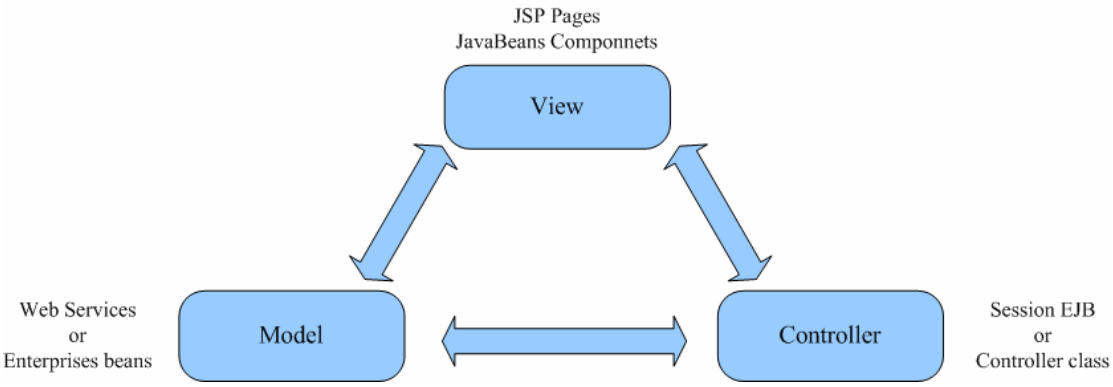

圖 18 Model-View-Controller Architecture

網路服務做為資料提供者的角色，其資料來源是主要來自資料端的資料庫。在簡單的應用環境之下，系統通常只會有唯一的資料庫，但是在企業應用環境之下，通常是許多套應用程式並存，而其所屬的資料也通常需要在彼此之間流通，但是資料流通方式通常是應用程式再獲得其他應用程式的資料庫使用權限後，以直接的方式來存取內容，這種方式有可能會破壞資料的一致性及造成資料庫安全的問題，但是經由透過網路服務我們可以把資料控制方法全部封裝在一起，而個別應用程式若要取用資料則可以透過對應的網路服務來達到目的，降低其中風險。

在圖 19 中為一資料庫多重存取的例子，Application server 上面存在有 AP X、AP Y、AP Z 三個應用系統，分別以 Database A、Database B 及 Database C 為主資料來源庫。AP X 及 AP Y 為獨立應用系統其資料參考來源都是單一位置，

此類系統如人事管理系統、工作事項管理系統等。AP Z 為一統合型應用系統，其資料來源除了原屬資料庫外，尚需要參考其他資料來源才能運作，此類系統如會計系統。

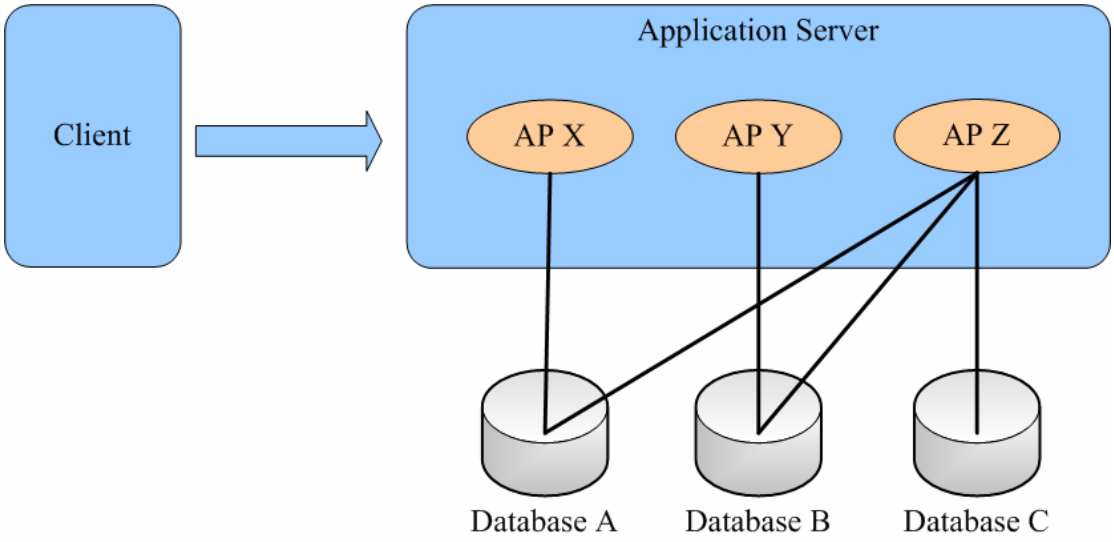

圖 19 多系統下的多重資料庫取用

此資訊系統配置方式可以看出每套應用系統皆需要重新開發或重複配置連線模組增加開發成本，此外由於資訊系統隨著時間推展及企業成長，越來越多應用系統及資料庫將讓這樣架構底下的資訊系統越來越雜亂，維護成本也會隨之增大。假設每個應用系統的連線資料庫模組其維護成本是 Capp，應用系統數目是 Napp，資料庫數目是 Ndb，則其維護成本最高成本可達

Capp * Napp * Ndb

在資料庫不改變之下增加一個應用系統其維護支出以 (Capp * Ndb) 的倍數增加，換句話說如果最初一套應用系統時其維護成本是 10 萬元，在 n 套之下則需支付 10*n 萬元，若再考慮資料庫數目的成長，其最大成本則將更高。

在此研究裡中所建立網路服務的有一個特性－資料重用，這是指其他應用系統只要知道網路服務的 WSDL 就可以取用這個服務，並獲得所提供給外部的操作(operation)。改變圖 19 的資訊系統架構方式，分離出每個應用系統取得資料

來源元件，製作成網路服務，使應用系統的資料來源轉而從網路服務取得，改變後的架構如圖 20 所示。

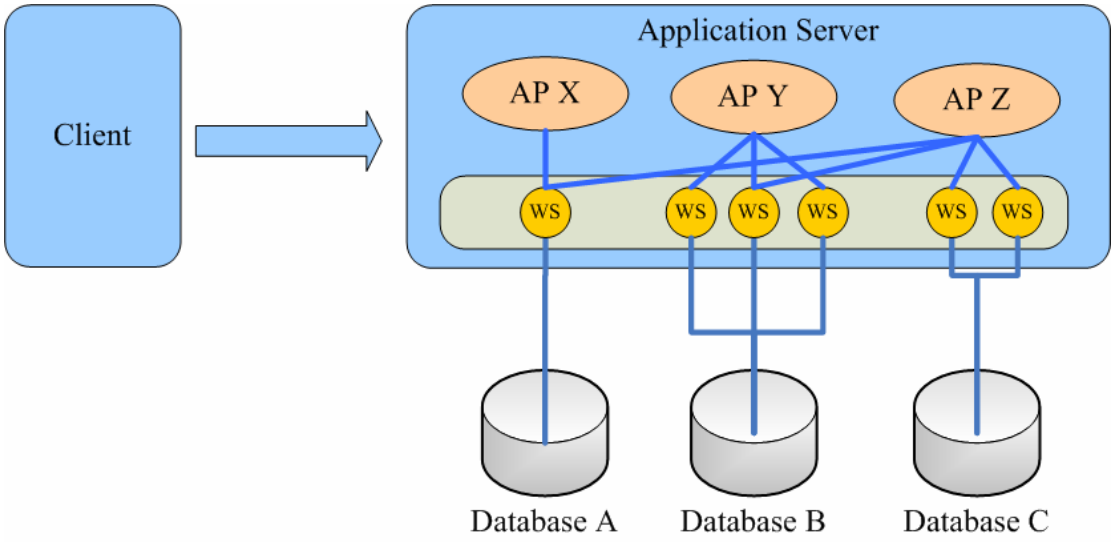

圖 20 以網路服務連接資料庫

上圖中分離出各應用系統的資料取得方式，並轉化成網路服務，而各應用程式所形成的網路服務除了能提供原來應用系統的資料來源外，也可以把此服務提屋給其他應用系統使用。假設每個網路服務連線維護成本是 Cws，應用系統的連線網路服務模組維護成本是 Capp，應用系統數目是 Napp，資料庫數目是 Ndb，網路服務數目是 Nws，則其維護成本最高成本為

Capp * Nws * Napp + Cws * Nws

但是由於網路服務上的應用系統只需要建立呼叫網路服務的方式並不涉及資料處理，所以 Capp 的成本近乎 0，在此情形下其維護最高成本為

Cws * Nws

與之前架構下的維護成本(Capp * Napp * Ndb)作相對性比較，從變數數目上可以知道網路服務的方式比應用系統直接連線取得資料的方式更為簡便，後者複雜性高於前者，所以可以推測後者可維護性高於前者。另外一般情形下網路服務的維護成本(Cws)絕大部分都遠低於應用系統維護成本(Capp)，所以其所需要維護成本實值也是低於前者。而在同樣的情形之下新增加一個應用系統使用原來的資料

庫，所以也不需要增加網路服務數目，其維護成本增加的幅度趨近於 0。當系統越益複雜的時候，採用網路服務架構系統可以讓我們有一個成長較緩慢的維護成本成長曲線。

存在網路上的網路服務，只要知道 WSDL 的描述內容就可以取得資源，這點讓我們在新系統的開發獲得一個很大的好處，對於以前所研發的網路服務我們可以直接在新應用系統直接使用，而不需要再重覆開發相同功能的程式或產生一份相同複製的程式在新應用系統上而造成人力浪費及維護成本增高的情形一再出現。如圖 21 顯示，假設在兩部應用伺服器上各有其應用系統存在，在 Application Server II 上新開發一個應用系統 AP Z，如果此應用系統的資料來源都可以 Application Server I 上取得適合的網路服務，則可以透過支援的傳送方式，如 HTTP/HTTPS、FTP、SMTP 等傳送以 SOAP 封裝的資料內容，而不需要製作新的資料元件，可利用再使用元件增加開發時間。

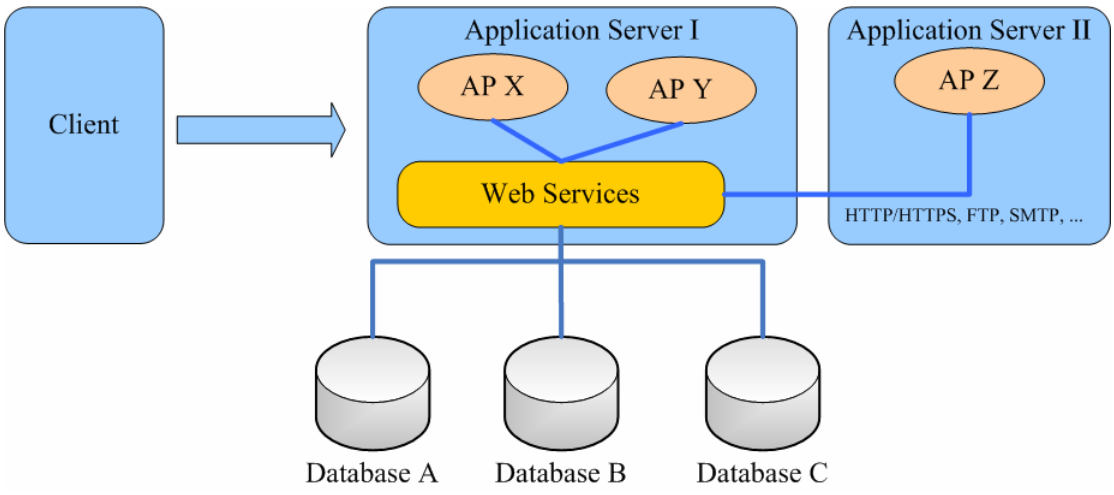

圖 21 多應用系統下的網路服務架構

作為一個網路應用程式最常遇到一個問題，便是服務品質隨者伺服器訪問人數增加而下降，換句話說，也就是系統現有效能已無法應付外在需求。這時候解決的方式有兩種，一是換裝系統配備增強系統本身能力，如更高時脈的中央處理器、大容量的記憶體及使用磁碟陣列等，這種方式稱為向上延展(scale-up)。這種

方式雖然可以即時方便地提升系統能力，但是受限於當時電腦硬體技術及擴充能力，系統效能的加強是有限的。相對於自我提升的方式，另一種方式是分散外來要求，利用相同功能的伺服器分散服務來源，使得單一獨立系統都能控制在最適量的訪問要求下運作，這種方式稱為向外延展(scale-out)，但是此種方式的缺點就是每台獨立系統下的程式(process)無法跨伺服器即時互相溝通，只能透過後端資料庫建立非同步溝通方式。若應用系統對外限制單一入口點，這時候有訪問都從同樣的服務接口進來，要讓系統不會因為大量的訪問要求而失效，便可以利用向外延展的概念建立分散式元件，把元件分散至內部網路上其他伺服器上，使得應用系統效能的消耗不會集中在一台機器上。

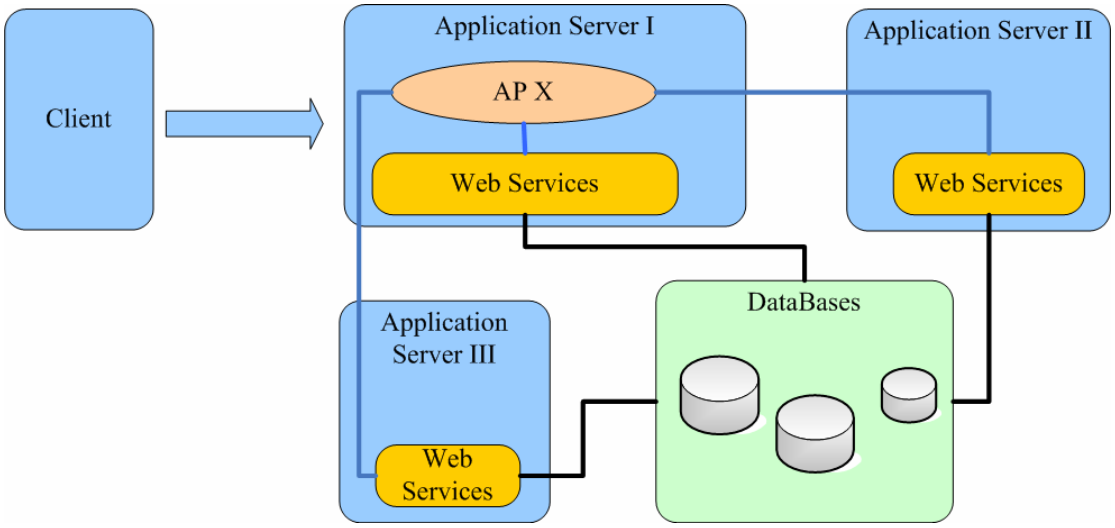

圖 22 應用系統下的分散元件架構

在圖 22 中顯示一個應用系統使用網路上網路服務資源的情形，結合向外延展的建置方式配合網路服務的配置，混合兩者的優點使得系統負載可以分散而服務更多前端要求。我們可以從圖中看出 Application I, II, III 中都存在有網路服務，這些服務可以是建置 AP X 應用系統時所開發，也可以是原本就已存在而被 AP X 應用系統所利用，但是不管哪種方式此架構可以把系統耗能較重的網路服務配置另一台獨立的伺服器，使此伺服器專門處理此一動作而達到耗能分散的目的，作為直接對 Client 端提供入口伺服器而言也就能提供更大量的服務要求。

在內部應用系統架構的設計需提供一彈性易於擴充的，容易被了解及維護機制。N-tier 分層設計方式[9]把系統分成數層，元件依據所屬功能集中於同一層中並配合設計樣板設計隔離內部元件複雜性，使得應用系統即使需要重新編譯漣漪效應不會擴散至其他層中。當應用系統提供給其他服務及應用系統使用時，最常使用的方式是以一組 APIs(Application Interfaces)方便外部服務能不用了解原系統架構便能使用應用系統大部分功能，但是 APIs 方式由於要統合原來複雜地應用系統內部物件需要對系統相當熟悉，開發 APIs 常常是非常耗費時間且當物件介面有所變化也會影響 APIs 使用，所以系統改版時也會增加系統維護的複雜度。網路服務的導入讓系統能提供一個更方便的方式讓其他服務及應用系統使用，由於網路服務提供註冊功能使開發者可以根據元件特性發布註冊資訊，使得其他應用系統從註冊資訊中便可以從網路上找到網路服務，進一步使用其所提供的功能，完全不需要增加一組額外的 APIs 程式。網路服務在改變其內部功能時，也可以使用策略讓服務不間斷，如當網路服務需要被修改的時候可以維持舊有的網路服務，當修改完成後配置新的網路服務至其他伺服器，再重新 UDDI 註冊則便可以使得服務要求導向新的網路服務伺服器位置，讓前端使用者使用新的服務功能而不需要中斷。

搭配著外部網路服務作為資料存取的媒介，並按照 N-tier 架構設計方式，應用系統的架構設計模組也區分成三個主要部份，分別為展現(Presentation layer)、商業邏輯(Business logic layer)及資料(Data layer)三層[9]，展現層負責訊息顯示及外部資料輸入資料等對外溝通介面；商業邏輯層提供問題的解決辦法；資料層則是提供資料的提取及存放的一個可與外部溝通的介面，分層設計方式把一個大型系統分成好個較小且獨立的部分，讓軟體易於建置、重複使用以及修改，有助於與不同的技術或領域知識相互配合。

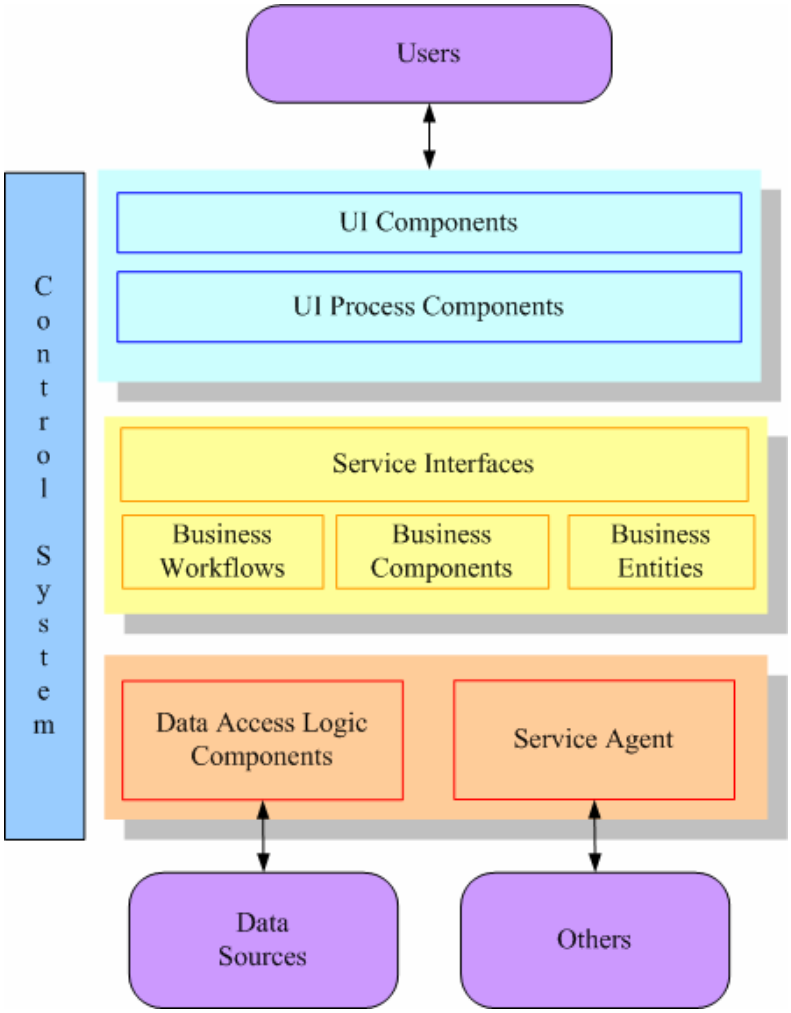

圖 23 應用系統 N-tier 架構

在圖 23 裡，為網路服務應用系統 N-tier 設計架構圖，圖中主體架構分為三區分別為展現層、邏輯層及資料層，而在此之外另有三個獨立角色存在，User 代表前端使用者，意即所有與這套系統相關的人。Data Source 則是資料來源提供者，提供有用的資料給應用系統邏輯元件參考，另一方面也用來儲存計算過後的資料，而其實體可以是一般資料庫或是檔案。Others 則是其他服務或應用程式，相較於 Data Sources 提供資料給應用系統，此類多屬於不能直接由系統控制其後端資料，所以在應用系統中有一 Service Agent 專職與其溝通。

在圖中表現層又可以分為上下兩層各含有一個模組，一個是 UI Components 主要提供使用者人機圖形化操作介面，負責資料規則化的輸出及便利地操作環境，如視窗(window)介面或瀏覽器介面。除此之外，獨立此層的目的，在未來在

新的技術及設備下使用者改變新的存取方式時可以提供擴充機制，如手機介面。另外一個在表現層的模組是 UI Process Components，這個模組主要負責多個 UI 的一致化控制，現今的系統複雜化，大量資料的參考使得決策選擇不容易，往往一個流程的進行，不是一個視窗或是瀏覽器介面就可以完成的，如線上購物系統中購物流程的完成使用者需要經過選擇產品目錄、檢視商品詳細資料、選購至購物車、選擇金錢交易方式、再來選擇送貨地點等等一連串的程序，明顯看出非是單一視窗即可提供全面化的資料訊息，所以需要一 UI Process Components 負責管理 UI 的流程及各種使用者介面與使用者互動的情形。

在邏輯層裡也可再分為兩層共四個模組，分別是 Services Interface、Business Workflows、Business Components 及 Business Entities。在這個四個模組中，Service Interfaces 的功用類似於展現層裡的 User Interfaces，其主要目的是提供適當的介面適合各種不同用戶端呼叫，使得能在未來符合跨系統整合需求。Service Interfaces 的作用是替子系統或其它層級元件裡的一推介面定義一套統一的高階介面讓子系統更容易使用，此部份利用 GOF 設計樣板[22]中的 Façade 樣板來完成，此舉可以降低系統或元件之間的溝通及依存關係，減少展現層中元件的直接觸及此層元件，降低所要面對的物件數量，讓此層元件能更容易使用，也降低兩層彼此之間的耦合性，在屬於同樣層級裡的元件其耦合性往往都很高，此樣設計方式為系統在未來改變元件功能時，因元間之間的低耦合性，可使得程式的漣漪效應(rippling effect)不至擴大而變得難以維護。Business Components 為領域知識的實作，其所包含的是在一特定領域的知識技能的運算，如線上購物系統中帳單稅額計算、物品配置費用等工作，或者是 ATM 系統中的轉帳費用拆帳處理等工作都可在 Business Components 中完成。Business Entities 封裝系統中各種物件成為獨立元件，如線上購物系統中的訂單及商品，這樣做的目的可以讓每個物件都能擁有私有獨一無二的識別編號，幫助物品或人在系統中方便流通及資料交換。Business Workflows 負責與其他系統互動，完成交易處理，如線上購物系統中，

物流系統與金流系統。

在資料層裡的模組主要是做為資料來源的提供者，由於資料來源的不同而分成兩個部份－Data Access Logic Components 及 Others。Data Access Logic Components 負責處理來自於完整地結構化資料，如可透過結構化查詢語言的資料庫系統。Service Agent 模組為其他不屬於 Data Access Logic Components 的其他資料來源獲取媒介，提供一套處理機制來得到其他系統所提供的資料，如由另一金流服務系統得到處理銀行的交易金額。在此架構之外，另外也建立一控制系統(control system)，此控制系統用來管理展現層、邏輯層及資料層等三層中的元件及介面功能，及調整此系統的初始資料功能等基礎設定。

在圖 23 中最適合以網路服務的形式存在有 Data Access Logic Components 及 Business Components，由於資料的提供方式往往容易被所使用的程式語言及資料來源的形式等等因素而被限制服務的對象，導入以 XML 為基礎的網路服務可以使得資料來源的提供可以完全獨立，Data Access Logic Components 可不限制於特定對象的存取方式，也增加了元件及資料的重覆使用率。另一個適合以網路服務形式存在的模組則是 Business Components 模組，這個模組的功能是計算領域知識的實作，其主要功能大都為數值計算，可被封裝於單一元件裡，也適合以網路服務的形式存在，提供其他應用系統透過互動機制能容易的存取此應用系統的功能，如在圖中 Service Agents 便能直接與網路服務溝通取得資訊。

在下一章裡將以一舊有程式(legacy system)為例子，依五個研究步驟順序重整這個系統為新的應用系統架構使其能具有延展性(scalability)，並獲得能再使用的網路服務。

第 4 章　案例研究及分析

在此章裡將以一個舊有程式做為例子，把此程式轉移成上章中的新應用系統架構。這個舊有程式如 2.7 節中所描述的是一個線上辦公室系統，其架構為標準的網路三層式架構，原系統建立此架構最大的目的是提供線上遠端使用者可以不限地點的存取服務，但是在系統成長至某個階段就產生效能瓶頸，在功能上既不方便擴展本身功能也難以與其他應用程式互相整合；在效能上則也不容易透過元件佈署不同伺服器上達到效能平衡(balancing)的作用，所以將利用本論文所提及的研究步驟及系統架構來轉變此系統成為一個具有可擴充性、可維護性、具延展性的低維護成本之應用系統。

4.1 案例研究

在五個步驟，最先執行的是初期分析步驟，此步驟的任務是收集資料，在資料收集完畢後並就所得資料進行分析，此外，也需對重整後的新系統需求進行了解。收集整理所收集的資料及比較新建系統的需求，幫助了解新系統目標及提供後續階段參考資料來源。在此步驟裡從『華人數位股份有限公司』得到 Office Organizer 的原始程式碼(source code)和系統使用指導書(user guide)。程式碼裡包含兩種類形的程式檔案，一種類型是 Java 編譯程式，另一種類型是 Java Server Page (JSP)格式的動態網頁語言，雖然檔案格式不同，但是都是基於 JAVA 語言所延伸的檔案類型。系統使用指導書則是針對一般使用者及系統管理員來說明系統中每一項功能的操作方法介紹及系統數值設定所代表的意義說明。這兩部分文件分別提供最高階及最低階的系統了解，但是由於缺少連結此兩文件的系統設計文件及系統需求分析規格描述，使得無法容易地了解整個系統設計架構及內部結構，所以在第三步驟應用分析步驟裡從現有文件中取得此類資訊，利用 United Modeling Language (UML)[18][30]來還原系統使用案例圖及系統內部類別圖，提

供後續系統開發能方便地瞭解整個系統。

在同一步驟中，新應用系統架構不同於舊系統是增加了網路服務，為了讓新的應用系統架構能支援網路服務,所以應用系統需要增加 SOAP 引擎來處理網路服務資訊的處理，考慮能與原系統 Java 應用程式相容，在新應用系統裡選用 Apache 組織[23]開發的 AXIS (Apache eXtensible Interaction System)[25]。此外為了讓應用系統能更具有強固性，HTTP server 及 JSP container 則是從原系統使用的 Enhydra[23]改換為同是 Apache 組織所提供開放網路應用程式 Tomcat。表 4 是變化前後兩個系統的執行環境比較，從表中兩個系統中最前端的 web browser 及最後端資料庫伺服器的執行環境並沒有改變,變化的是中間層裡的應用伺服器基礎建設(Infrastructure)。

表 4 Office Organizer 執行環境

	Exiting System	Reengineered System
Client	Internet Browser	Internet Browser
Application server	JDK1.2.2 later Enhydra	JDK 1.4.2 AXIS 1.1 Tomcat 5.0.19
Data Server	MS-SQL 7.0 or Postgre SQL 7	MS-SQL 7.0 or Postgre SQL 7

根據系統環境架構設計，新的系統形成如圖 24 的環境架構。其運作方式是應用伺服器上的元件會先接受使用者的要求(Request)，然後系統根據要求協同數個網路服務提供者(web service provider)運算所得資訊回傳給前端，然後結束整個程序。這個系統的運作架構不同於原先系統在 2.7 節圖 11 的運作模式，主要差別在於前系統對使用者的處理運算從集中在一台應用伺服器，而後者在此架構能把運算工作分散給其他能提供網路服務的其他應用伺服器，進而提升網路效能。在圖中 UDDI Registry 則是提供程式設計時網路服務註冊之用，註冊過後的網路服務可以被其他程式給尋獲進而使用，如果少了這 UDDI registry 伺服器則會降

低網路服務軟體再利用的特性。

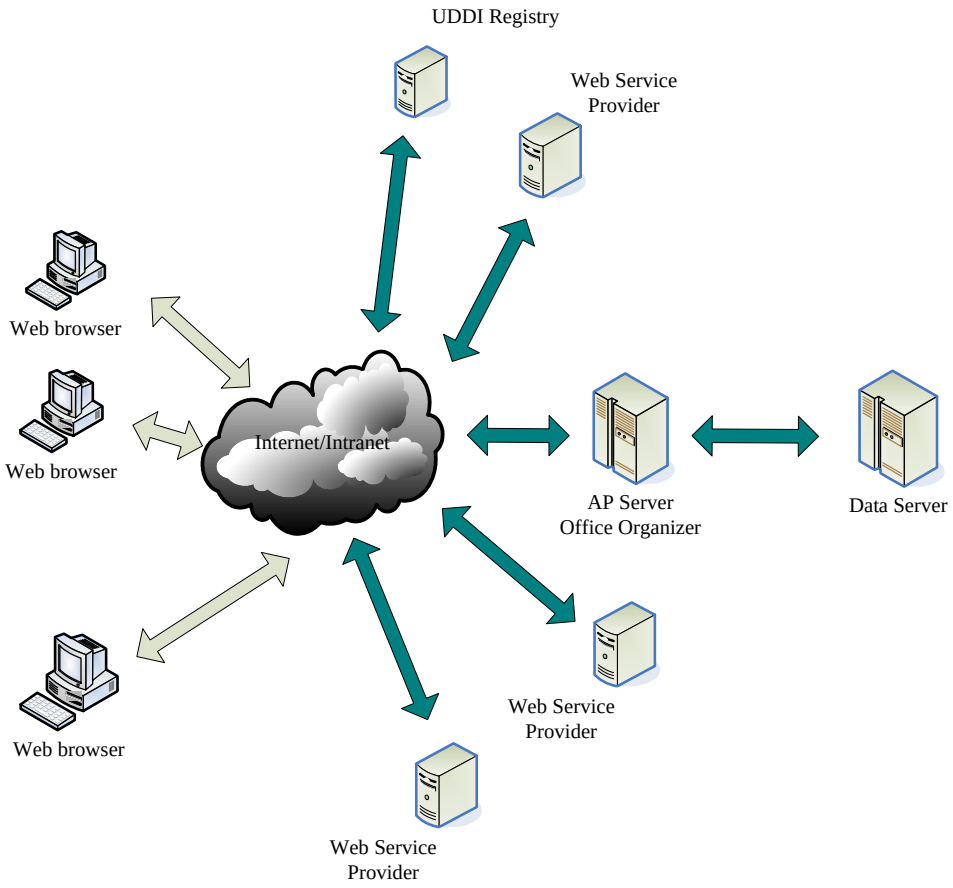

圖 24 網路服務為基礎的系統架構

第一個步驟建立了對重整前後系統的初步認識，接下來的步驟為功能分析階段，在此步驟中會詳盡地分析在 Office Organizer 的所有模組及組成元件，在了解元件的功能後，也會判斷哪些現有元件適合轉化至網路服務的分析研究，另一方面對於系統中已經不適合存在或沒有作用的元件也會進行篩選，這個動作可以幫助系統能保有最簡單、乾淨的程式碼，增加執行效率。

在 2.7 節中 Office Organizer 系統的組成是由 JSP 網頁語言、HTML 網頁語言、Java class 檔案及部分 APPLET 所組合而成。從原始程式碼裡知道支撐此系統的類別可以分成 com、stl、eoffice、java、org、javax 等六大套件。此六大套件所有套件內部結構如下表 5 所示，Level 1~4 表示套件命名空間的階層，最後一欄則

是系統類別圖，所有類別圖可以參考附錄 C。

表 5 系統套件

Level 1	Level 2	Level 3	Level 4	Diagram
com	Zen	Eoffice	Account (14)	圖 47 圖 48 圖 49
			Check_attendance (34)	圖 50 圖 51 圖 52
			Company (11)	圖 53 圖 54 圖 55 圖 56
			Conference (8)	
			Control (5)	
			Cooperator (14)	
			Gen_routine (38)	
			Mailer (6)	
			Personnel (26)	
			Sing_in (4)	
			Util (17)	
Eoffice	Bean (11)			圖 57
	Company	Department (5)		圖 58
		Emp (4)		圖 59
		Visitor (1)		圖 60
	Docm	Doc (7)		圖 61
		DocIdf (6)		圖 62
		User (1)		圖 64
		Group (5)	Visitor (2)	圖 63
	Message (7)	FlowMessage		圖 65
		GeneralMessage		圖 66
	Pa	Job	FileChecker (5)	圖 67
			MessageHandler (2)	圖 68
		Mediator (3)	Handler (4)	圖 69
		Ui (2)	Basic (7)	圖 70
			Icq (17)	圖 71
			Login (2)	圖 72
		User (1)		圖 73
	Service	Login (2)		
	Util (1)			
	Visitor (1)			
	Workflow	Metaflow (1)		
Java	Io			圖 74

Level 1	Level 2	Level 3	Level 4	Diagram
	Rmi			圖 75
Javax	Activation			圖 76
	Ejb			圖 77
	Swing			圖 78
Org	Xml	Sax	Helpers (1)	圖 79
Stl	Framerwork	Bean (2)		
		Com	Db (3)	
			Ftp (4)	
			Public_ (4)	
			Util (9)	
		Message (5)	Visitor (3)	
		Schedule (6)		
		Service (1)	Message (6)	
			Tree (2)	
		Tree	View (12)	
			Visitor (3)	
		Xml	Db (3)	
			Form (2)	
			Util (2)	
			View (1)	

在上表裡，Com 套件為「華人數位股份有限公司」內部開發元件的集合，主要有權限檢查、使用者控制、會議功能、信件功能、身分驗證功能等等，為系統主要部份之一。Stl 套件中物件同樣屬於內部開發完成套件，主要功能是串連整個系統，其中有 XML 應用元件、樹狀選單、行事曆排程等功能。Eoffice 套件則是系統中最前端作用元件，有文件管理、訊息傳送、工作流程、登入服務、公司組織等等。另外，其他三組套件 java、javax 及 com 則都是 sun 公司的 java 程式開發套件，為整個系統的基底及元件介面實現主要來源。

在系統角度來看整體功能，可以得知在此系統上共有三種主要角色－使用者(user)、進階使用者(power user)及管理者(administrator)，如圖 25 所示。管理者可以設定各類系統參數，及定義系統規則。進階使用者擁有核可單據的權限，如

請假申請單、庶務申請單等等。使用者則是公司一般僱員可以操作由管理者所定義好的系統規則。基本上，使用者可以操作所有的系統功能而管理者可以設定所有系統功能規則及權限，進階使用者只有當系統功能需要有文件核可的角色時才被建立。

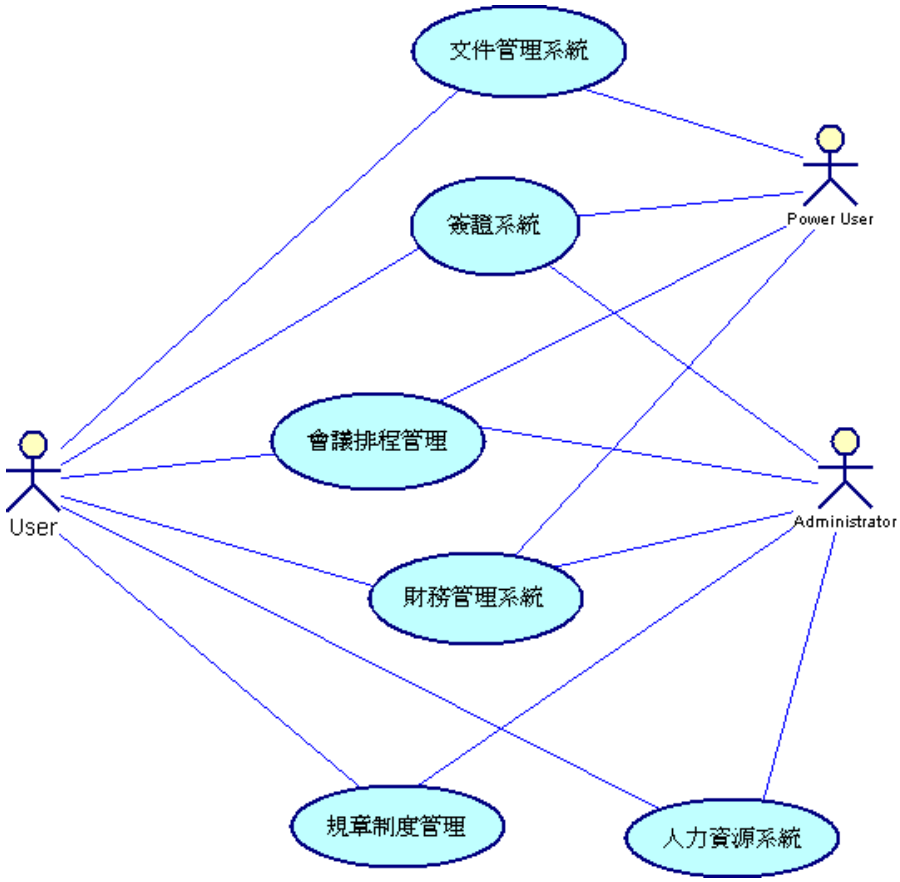

圖 25 Office Organizer 系統使用案例圖

從系統使用者案例圖可以了解系統功能模組與使用者關係，而從類別圖可以瞭解程式碼中物件間的關係並知道各元件或子系統功能及作用，完成此步驟後，下一個步驟是應用分析階段，在此階段主要的目的是分析從原系統中得到的元件在轉為新系統後，是否需要設計新的元件及新製作的網路服務需要提供的操作也是在此階段完成。

進入系統轉換步驟，將更詳細的深入系統中每一個功能模組中，並把程式轉

移至新的系統架構裡，在此步驟另一重點在於如何轉移(migrate)目前程式成為網際服務。其中對於網路服務的開發有下列幾項執行步驟：

1. 分析系統中所有的元件；
2. 找尋 UDDI 註冊中心是否有可利用的網路服務；
3. 若無，則製作網際服務；
4. 配置網路服務至伺服器上。

在第二項步驟裡，搜尋是否有類似功能的網路服務已完成且註冊於 UDDI 註冊中心，如果有的話可以直接獲得其 WSDL 後使用，可以減少開發時間及增加元件重複使用。

根據原系統的功能模組分類方式，以系統中最先接觸的簽證子系統做為此步驟範例。簽證系統的功能主要管理使用者上下班時間，其與系統角色的互動如圖 26 所示。其行為是當使用者登入系統後根據個人權限模版獲得系統操作權力，使用者進入此系統上可以執行簽入及簽出功能，簽入功能是向系統登記此時為使用者上班時間；相反的，簽出功能是向系統登記此時為使用者下班時間。在使用者不清楚個人當月或其過去某段時間的上下班情形，簽證系統也提供出勤紀錄管理功能，使用者可以隨時透過此功能查詢之前上下班情形。出勤紀錄管理對於使用者為限制只有查詢權限，若因為特殊原因需要修改資料，則需要求助於進階使用者或系統管理者。系統管理者在此系統中尚有一項特別功能為設定工作時段功能，此功能是為系統定義何時為上班時間及下班時間，以因應不同地域及文化需求。在出情紀錄管理除了資料查詢功能外還包括統計及表單列印功能，此兩項功能皆為進階使用者及系統管理者的所擁有的功能，統計功能可以計算個人出勤狀況，累計曠工時數及上班狀況比較等；表單列印則依據需求格式化資料輸出成 XML 檔案資料或表格。

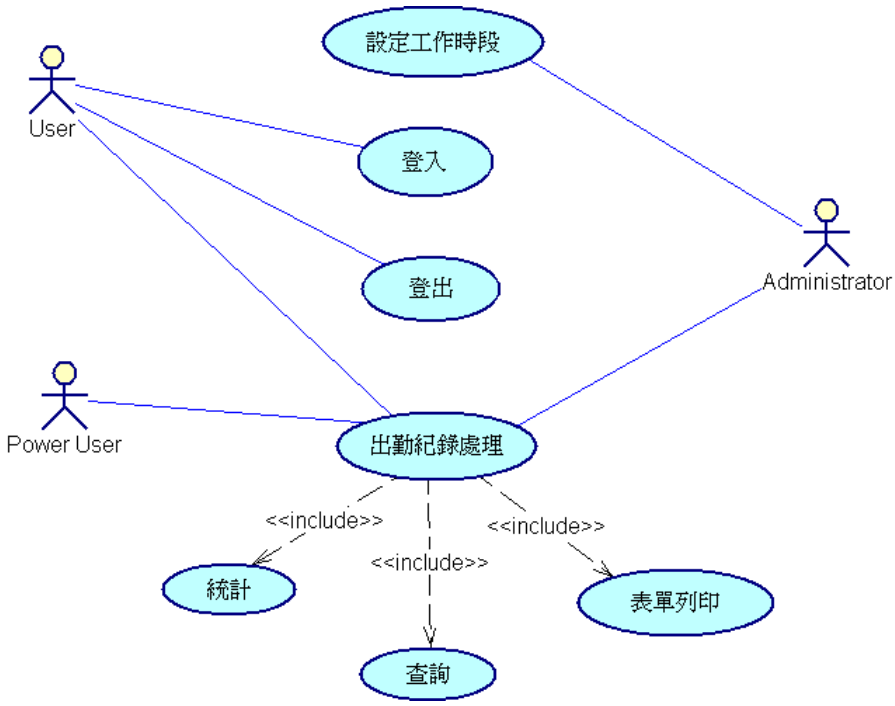

圖 26 簽證使用案例圖

從圖 26 簽證使用案例圖裡了解此系統功能，根據物件導向分析與設計(OOA&D)方法，定義此系統裡共有四個主要類別：

- Card 類別，為記錄員工上下班紀錄卡片。
- Card Puncher 類別，為登記時間點至 Card 的動作者。
- Card Manager 類別，為統計員工 Card 資訊的管理者。
- System 類別，為應用系統的屬性集合。

在這四項類別中 Card、Card Manager 及 System 類別都需要與資料溝通，如 Card 類別裡 ToSaveToDatabase 操作，Card Manager 類別裡 ToSearch 操作，System 類別裡 ToSetTimeOfStart 及 ToSetTimeOfClose 操作皆是，所以在這些操作及資料來源之間可以定義數個網路服務來連繫兩邊，如圖 27 所示。

在圖 27 中，綠色的類別為應用系統簽証子系統的主要類別，橙色的類別則是配合子系統所獨立出來的專門與資料庫溝通的網路服務。在這個系統裡，共有 Card Banker 及 System Banker 兩個網路服務，分別處理資料庫裡的 card 資料表(table)及 system 資料表，由於此兩個服務為了系統安全性，所以在此是有限制的

只能處理資料庫中的指定資料表，但是其他子系統及其他應用系統如果需要獲得同樣類似的資訊可以直接透過 Card Banker 及 System Banker 來取得，不需要重新建立私有的資料庫連線功能。

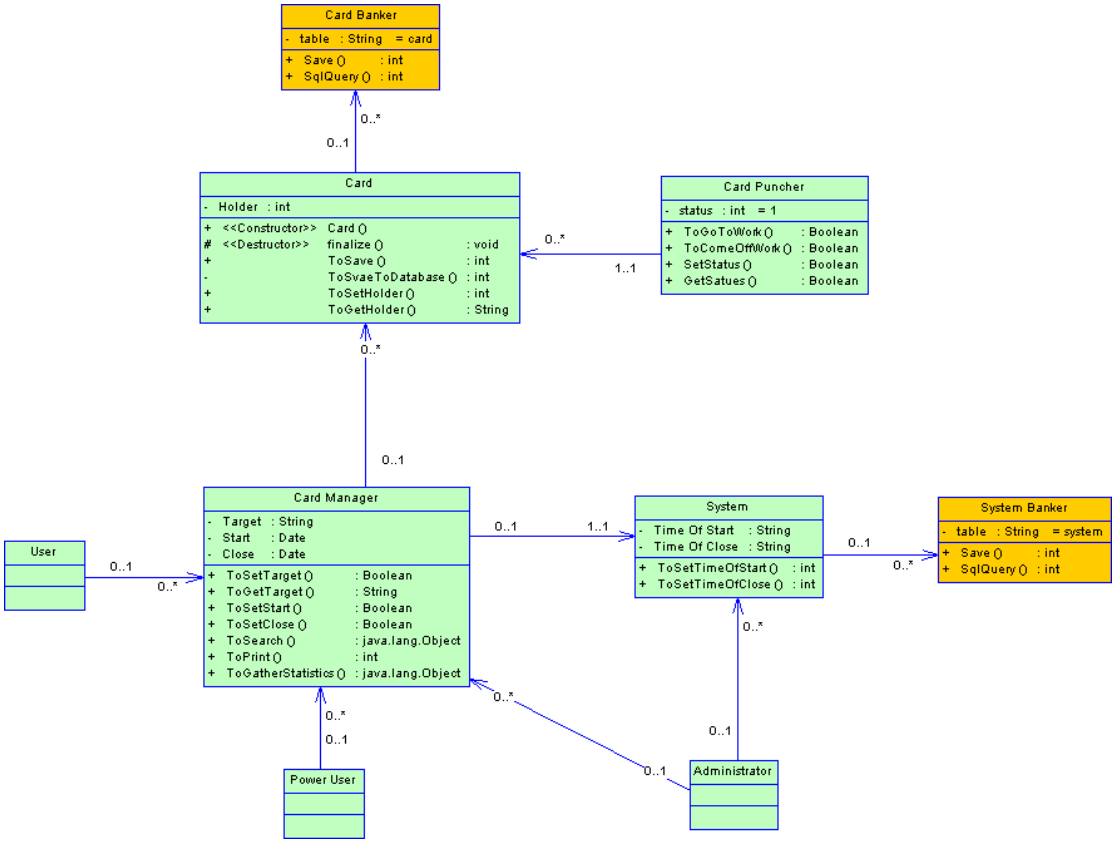

圖 27 簽證系統類別圖

對應至應用系統 N-tier 設計架構裡，System Banker 及 Card Banker 連接資料來源屬於 Data Access Logic Component，System 及 Card 是獨立的商業元件屬於 Business Entity，Card Puncher 屬於領域知識是 Business Component，而 Card Manager 則是 Service Interface 整合了 Business Components 及 Entities 提供一簡單的介面給 User、Power User 及 Administrator 使用。其他諸如文件管理系統、會議排程管理系統、財務管理系統、規章制度管理系統及人事管理系統也是以同樣的方式分析其使用案例圖，如附錄 C，從其中得到各類別圖如附錄 D 中所示，並分離出各自需要的網路服務而建置之，而整合六個子系統得到完整一個應用系統內部結構如圖 28 所示。

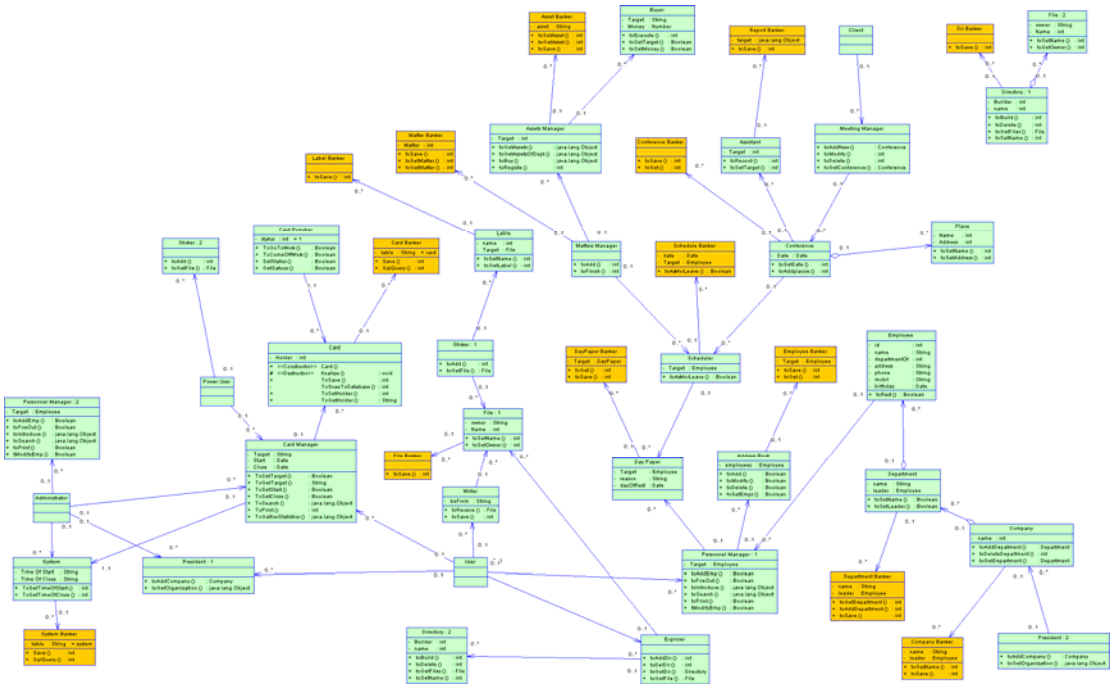

圖 28 Office Organizer 系統類別圖

系統中其他五個子系統以同樣的方法完成後，我們可以形成一個以 Java Server Pages 作為使用者圖形介面及以網路服務為核心來提供資料，而系統企業邏輯知識包圍其中的一個架構，其概念圖如下**錯誤! 找不到參照來源。**所示。在圖中六個子系統可以共用最外層的圖形介面及最內部的資料來源元件，但是企業邏輯的部份是各自獨立而不相關的，這樣子的好處是減少系統之間的耦合性，當邏輯元件變動時縮小了連漪效應範圍。若系統元件需要參考其他子系統資料結果，我們可以透過共有的網路服務來提供這方面的資訊，而若是需要搭配其他子系統來共同運作才能完成的程序，則是使用應用系統 N-tier 架構中的 Service Agents 來幫助系統間的運作。

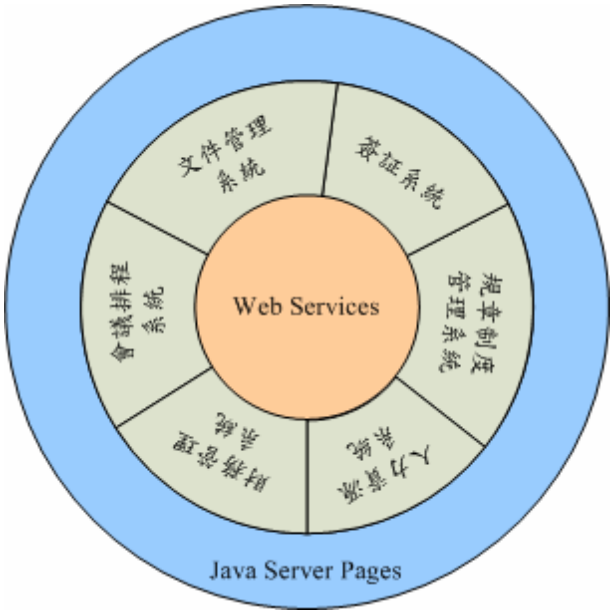

圖 29 系統構成圖

依此方式逐一對整個系統分析，把適合轉換的程式變成網路服務，逐步代替原始程式碼部份物件的功能。最後呈現的系統的內部結構圖，如圖 30 所示。此時，原來使用者仍然可以利用 Browser 觀看原來的 JSP，但是其後端的作法增加網路服務這個部分。在 JSP 上面可以呼叫 web service proxy 使用網路服務的介面，使其能透過 web service router 取得正確的網路服務位址。另一個部份，系統改善並非完全取代原先系統中 Java bean 的功用，所以在系統裡仍存在 session bean 及 entity bean 持續提供效用。當系統功能改為網路服務也帶來另一項好處，就是前端介面不會被限制成只有少數工具才能開發，現在只要有支援網路服務通訊且可以解譯 WSDL 的開發工具便可以使用網路服務所提供的功能及資料，如可以使用 Visual Basic[6]、PowerBuilder[7]等工具開發前端應用程式。

當系統轉換至新的應用系統架構後，步驟也進入最後一個設計重獲階段，這個階段的工作是針對新型態的系統重新製作新的說明文件，如使用者操作手冊、設計規格文件、系統函式介紹手冊、系統轉移技術書、系統建置手冊等文件，讓

[6] 微軟公司所發行視窗開發環境工具。
[7] Sybase Inc. 開發的資料庫前端應用工具。

後續的開發者及使用者能有參考的資料，知道如何使用及維護新的系統。

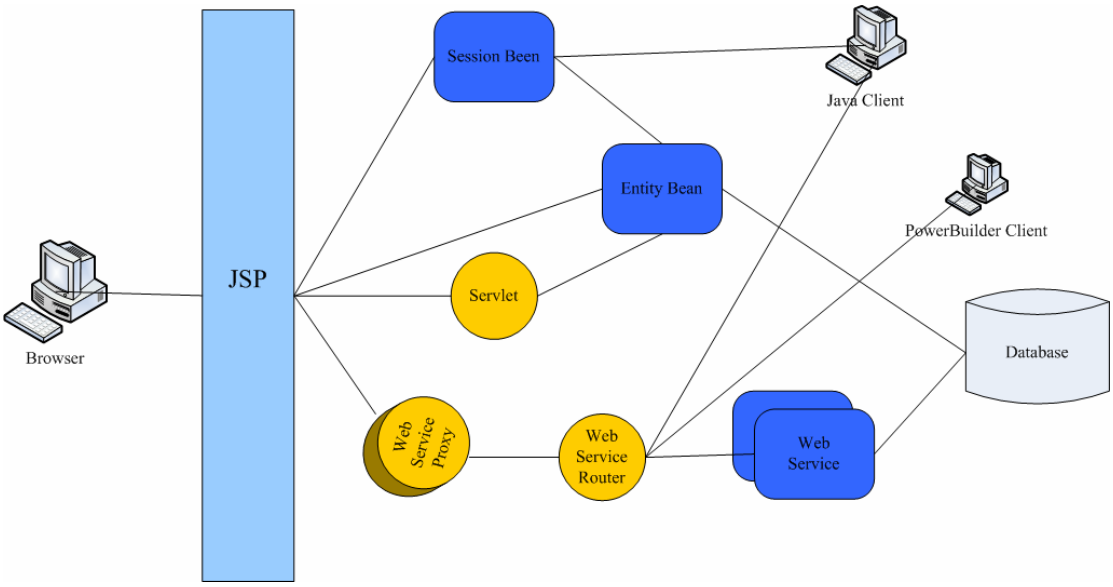

圖 30 Office Organizer 內部運作圖

4.2 案例分析

在這章中將針對重整過後的新系統分析，分析的主要內容有網路服務可再利用性(Reusability)分析、系統可延展性(Scalability)分析及系統可維護性(Maintainability)分析等特點來分析。首先在可利用性分析上，應用 Sybase 公司的 PowerBuilder 前端應用程式開發工具再重新開發一個 Client 端視窗系統，當前端使用者只把 Office Organizer 當成單純的線上打卡工具時，不需要使用其中複雜的功能，所以我們可以提供一簡單的視窗應用程式，此應用程式只需做到兩項主要功能，分別是系統的登入及登出及上班時間的簽入及簽出功能即可。開發 Client 端視窗程式所有的 User Interfaces 皆直接在 Client 端完成，在程式直接中我們建立 web services proxy 並製作 SOAP 處理元件發送及接收網路服務的資料，形成如下的程式架構圖。

在圖 31 裡，利用快速的應用程式整合發展環境(IDE)開發在獲取 Card Banker

及 System Banker 網路服務的 WSDL 即可使用服務所提供的操作，利用 Card Banker 來記錄使用者登入的時間，並從 System Banker 取得上下班設定時間點，如果在上班在規定間之內則以藍色字體標示進入時刻，若超過則以紅色字體顯示。當前端程式完成可以處理 SOAP 資料元件，利用已存在的網路服務可以幫助我們快速地開發新的應用程式而不需要了解後端系統實作，所以可以確定網路服務的重覆使用性可以增加程式開發速度。

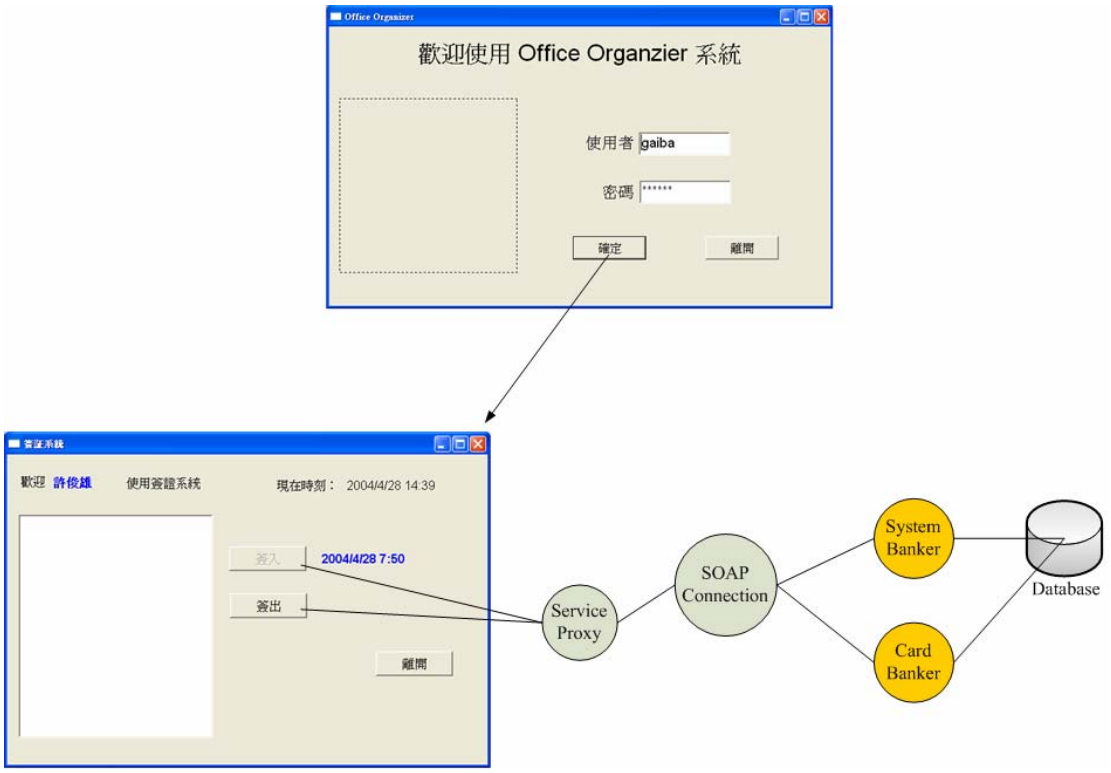

圖 31 簽証系統視窗程式架構

具可延展性的電腦系統（scalable computer system）可以讓使用者依需求與經費而彈性的擴充其運算資源以期增加其計算能力。在新的 Office Organizer 應用系統裡，於 Intranet 環境之下建置數個同樣的伺服器只需要配置 SOAP 引擎，如 AXIS，可容易轉移系統內部份或全部網路服務至新的伺服器上，如此一來本在應用系統 N-tier 架構九塊中的 Data Access Logic Components 模組及 Business Components 模組的運算就可以移至其他伺服器執行，如商業元件伺服器及網路服務伺服器，新的應用系統可以把網路服務配置在其他機器上，對系統本身而言

可以利用向外延展的方式增加其運算能力且不破壞系統架構，如圖 32 所示。

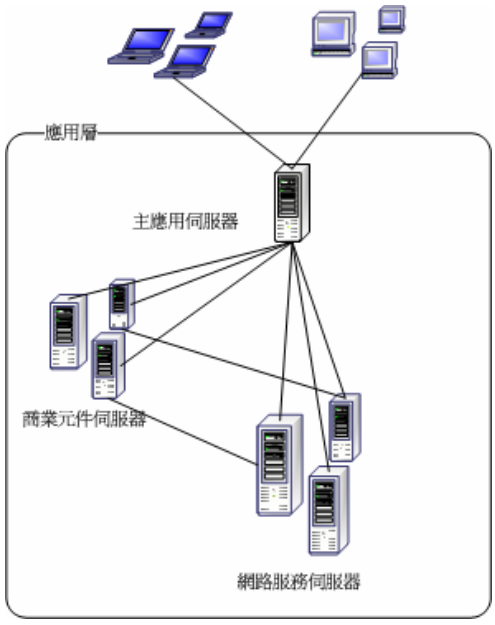

圖 32 可延展的多伺服器應用系統

在新應用系統裡轉移系統內部份或全部網路服務至新的伺服器上，如 Data Access Logic Components 模組及 Business Components 模組的運算移至其他伺服器執行，如商業元件伺服器及網路服務伺服器，使得原應用伺服器可以得到更好的效能，如圖 32 所示。

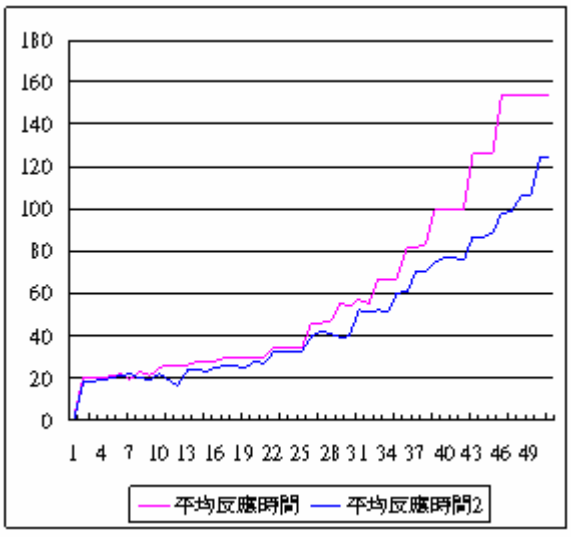

圖 33 系統反應時間圖

一個應用系統開發完成後，其後最重要的工作即是系統維護，如果以漂浮在海上的冰山來比喻的話，露出海面部份代表系統開發成本，而在海面下的就是系統維護成本，我們常常只注意系統開發所帶來的影響，可是當一套系統開發完成後常常會被使用一段非常久的時間，也就是有一段非常長的維護時間，所以其維護成本是大於開發成本的，一般而言，系統維護成本約佔系統生命週期(system life cycle)的 60%，在這個情況下，系統的可維護性就非常重要了。應用新架構系統 N-tier 的分層設計方式，使得系統內元件都能依照定義有適合的位置，增加了系統元件的企業應用集成可理解性；網路服務是可獨立於應用系統之外，每一服務都可以視之為單一個體，而這也讓此部份程式容易被修改及測試，可為整個系統增加可測試性及可修改性，所以新系統的可維護性是有被增強的。

第 5 章　結論

資訊技術的進步帶給許多生活上的變化，如手機改變了人類通訊的方式，網路服務是軟體技術與理論結合的另一項成就，推動網路世界資訊獲取的方式不限定只是人對電腦，也成為是電腦對電腦，應用程式對應用程式的溝通。

最早之前，人類交換資訊的方式是面對面同步化的溝通方式，但是在人類活動範圍擴大後，直接面對的溝通方式已逐漸不可行而轉化成以郵件方式傳遞非同步化訊息方式代替之，但是資訊卻無法及時性地被處理，為此方式的缺點。電話及傳真的發明解決了這個問題，讓資訊可以即時化不限地點地被傳送出去，可是對於利用這些資訊轉化成商業用途仍是不方便。網路的建置讓資訊傳遞方式又進入一個嶄新的時代，讓資訊傳遞更免除了時間的限制。但是之間資訊處理的過程仍需要相當程度的人類行為輔助，才能把資訊轉化成可被利用的型態。網路服務架構的提出及定義，又讓網路發生另一次質的變化。

網路服務透過 SOAP、WSDL、UDDI、XML 等技術，讓使用者可以利用這些技術創立私有的在網路上的服務功能。網路服務主要服務的對象是另一個網路服務或程式，使用 SOAP 的通訊方式傳遞資料，讓資料可以同步地即時被處理，大大地增進了企業資訊處理的能力。

以軟體技術本身來說，網路服務是以軟體元件佈置在網路上，使用者可以透過 WSDL 及 UDDI 來發現及呼叫元件，元件的重複使用縮短系統的開發時程，並使用被認可的元件也讓系統具有更佳的強韌性。但是網路服務是否都需要都是重頭開始製作成完全新的服務？在過去，我們已經產生了許多優秀的軟體，可是這些軟體在新技術的出現後，通常就被隱沒了。在這些軟體系統中，存在許多優秀的設計模組，在今日網路時代仍是可以被利用的，只是受限於無法被改變成以

網路為基礎(web-based)的程式放置於網路上供大眾利用，這是非常可惜的一件事，因此透過軟體重整工程方法整理出舊有系統的設計規格等原始知識，以元件方式重新設計，並從中得到可利用的元件轉化成網路服務來被重複利用，使得我們可以擁有更長使用週期的應用程式，及逐漸建立的元件庫(component library)也讓多個應用程式能共用核心應用元件，降低維護成本及新應用程式的開發週期，如容易透過新的元件組合方式來獲取新的應用程式及服務。

在這篇研究裡，提供一個重整的方法來轉化非網路服務程式成為網路服務。我們把整個過程分為五個階段，分別是：初期分析、功能分析、應用分析、設計重獲及系統轉換。從收集系統相關資訊、分析現有資料、擬定轉化方針、轉換系統及最後成果展現，一步步地實現。之中也遇到一些決擇，諸如：

1. 是否需要對整個應用程式流程重新建造？
2. 是否整個應用程式都以 web service 組成？

第一個問題，雖然流程重整可以讓整個系統更簡化，但這並非我們此次研究的目的，所以我們仍然保持系統原有的流程。對於讓網路服務取代所有的元件功能也並不是需要的，由於製作網路服務最主要的目的就是希望所製作的元件能被其他應用程式所重用，所以內部元件轉化成網路服務，以元件功能為提供內部資料的元件為主。這個部份的元件轉換也符合網路服務的資料重用的需求，所以並不需要對整個應用程式都利用網路服務組合而成。改變前後的系統並沒有對一般應用環境下的效能產生極大的幫助，可是採取分散式元件及 N-tier 架構元件可以獨立於其他伺服器上，讓複雜運算功能元件可以擁有獨立的運算能力讓系統增加了延展性，應用系統可以在極端環境之下還能擁有不錯的執行速度。而新架構所形成的應用系統除了製作可被重複使用的元件之外，也為以後企業應用系統整合提供良好的基礎。

在正式進入 21 世紀的這個時候是繼往開來的時機，除了對以前發展的軟體

技術、軟體理論、程式設計等內容保留精華部份並在新世紀中發揚光大，未來新世紀軟體開發的重點之一則是整合(Integration)，整合上個世紀留下的各種技術、理論及應用程式。在資訊普及化的現在，企業及個體使用者最害怕的是自己的電腦系統是個獨立的系統不能與外界溝通而成為資訊孤島(Informational island)，網路服務提供一個以 XML 為基礎的整合所有應用程式的方式，相信網路服務是在保留舊有技術之下，也能銜接上新的觀念。

在本篇論文中未來可以改進的地方有下面幾項，由於 UDDI 註冊中心上的網路服務會有更動的機會,所以建立一個隨時反應機制能動態支援網路服務 WSDL 的變動而確保系統能正常的運作，不會因失去網路服務而癱瘓。又如對於網路服務方面的加強方面，就是在面對多個相同功能網路服務的抉擇問題，這些都是在網路服務方面可以再加強的部分。而對系統本身而言，系統元件的移動也需要建立一個管控機制,能監測系統效能並能直接移動網路元件至其他負載較輕的伺服器上，使系統一直保持高性能狀態。在重整過程裡，可以在更多的研究對象中，訂立更詳細的執行步驟也是未來研究所需要建立,並針對步驟中的項目能自動化或半自動化執行，這些都是在未來需要更努力的目標。

參考文獻

[1] J. Boese and E. Reid, "Software reuse: a challenge," *Conference Record of the IEEE International Conference on Communications*, Vol. 3, 1991, pp. 1496-1499.

[2] E. Chikofsky, "Aspects to Consider for Understanding Web Site Evolution," *1'st International Workshop on Web Site Evolution (WSE'99)*, Atlanta, GA, Oct. 1999.

[3] G. Canfora, A. De Lucia, G.A. Di Lucia, "An Incremental Object-Oriented Migration Strategy for RPG Legacy Systems", *International Journal of Software Engineering and Knowledge*, Vol. 9, No. 1, 1999, pp. 5-25.

[4] Z. Chen, J. Chen, "Object oriented methodology for switching software reuse," *ICCS/ISITA'92*, Vol. 1, Nov. 1992, pp. 178-182.

[5] E. Chikofsky, J. Cross II, "Reverse Engineering and Design Recovery: A Taxonomy", *IEEE Software*, Vol. 1, No. 7, January 1990, pp. 13-17.

[6] S. Dietrich, F. Calliss, "The application of deductive databases to inter-module code analysis," *the International Conference on Software Maintenance*, Oct. 1991, pp. 120-128.

[7] W. Frakes and C. Fox, "Quality improvement using a software reuse failure modes model," *IEEE Transactions on Software Engineering*, Vol. 22, No. 4, April 1996, pp. 274-279.

[8] G.. Gannod, B. Cheng, "Using informal and formal techniques for the Reverse engineering of C Programs," *the Third Working Conference on Reverse Engineering*, 1996, pp. 249-258.

[9] Pour G., "Enterprise JavaBeans, JavaBeans & XML Expanding the Possibilities for Web-Based Enterprise Application Developments," *31st International Conference on Technology of Object-Oriented Language and Systems*, 1999.

[10] Steiert HP, "Towards a Components-based n-Tier C/S-Architecture," *In Proceedings of the third international workshop on Software architecture*, 1998, pp. 137-140.

[11] C. Lillie, "Distributed network of reuse libraries offers the best approach to successful software reuse," *Third International Conference on Software Reuse: Advances in Software Reusability*, Nov. 1994, pp. 207-208.

[12] A. Mayrhauser, A. Vans . "Program understanding: Models and experiments", *The Advances in Computer*, Vol. 40, September 1995, pp. 1-38.

[13] I. Potts, "An object-based design system for software reuse," *the IEE Colloquium on Automating Formal Methods for Computer Assisted Prototyping*, 1992.

[14] S. Ramakrishnan, C. Mingins, B. Henderson-Sellers, R. Duke, "Planned software reuse in object-oriented software engineering education," *the Software Education Conference (SRIG-ET1994)*, IEEE Computer Society Press, Nov. 1994, pp.

250-254.
[15] M. Brodie, M. Stonebaker, *Migrating Legacy Systems: Gateways, Interface & Incremental Approach*, Morgan Kaufmann Publishers, 1995.
[16] J. Butler, *Mainframe to Client/Server Migration: Strategic Planning Issues and Techniques*, Computer Technology Research Corp, 1996.
[17] Alfts Berztiss, *Reverse engineering, Re-engineering, and Concurrent Engineering of Software*, Department of Computer Science, University of Pittsburgh, 1995, ISSN 1101-8529.
[18] G. Booch, J. Rumbaugh, I. Jachbson, *the Unified Modeling Language User Guide*, Addison-Wesley, 1998.
[19] W. Chu, "Reverse Engineering," *the Handbook of Software Engineering and Knowledge Engineering*, Vol.2, pp. 447-466, World Scientific Press, Singapore, 2001.
[20] S. Demeyer, S. Ducasse and O. Nierstrasz, "Object-Oriented Software Reengineering", 2000.
[21] I. Graham, *Migrating to Object Technology*, Addison Wesley, 1994.
[22] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1994.
[23] Sommerville I., *Software Engineering*, 5th edition, Addison-Wesley.
[24] Apache Organization, http://www.apache.org/
[25] Apache eXtensible Interaction System, http://ws.apache.org/axis/
[26] Apache Jakarta Project, http://jakarta.apache.org/tomcat/
[27] Extreme Programming, http://www.extremeprogramming.org/
[28] Rational Unified Process, http://www-306.ibm.com/software/awdtools/rup/
[29] ObjectWeb's Enhydra Server, http://www.enhydra.org/
[30] OMG's Unified Modeling Language Resource Page, http://www.uml.org/uml.
[31] Simple Object Access Protocol Version 1.2, http://www.w3.org/TR/soap12-part1/
[32] Universal Description, Discovery and Integration (UDDI), http://www.uddi.org/
[33] Web Services, http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/
[34] Web Services Description Language 1.1, http://www.w3.org/TR/wsdl
[35] XML 台灣資訊網, http://www.xml.org.tw/Function/Fglossary1.asp?key=UDDI

誌謝

在資訊工程與科學研究所三年的研究中，我首先要感謝朱正忠教授指導，是我的指導教授朱正忠老師。在這兩年的研究所生涯當中，老師的諄諄教誨，對於學生的求學態度和思想觀念上，有著非常深遠的影響。讓我在迷茫的求學生活裡，確實了自己以後努力的目標。

此外，還要感謝資訊技術實驗室張志宏學長及各位同學、學弟們，謝謝依恒不時的關懷，讓我每每在意志薄弱的時候再次升起鬥志；謝謝曉玲、士嘉、致偉、翼汶彼此的砥礪而讓我有一次比一次更好的表現，也因此讓這篇論文能順利的產生。另外，也非常謝謝「華人數位股份有限公司」提供程式及人員協助讓此篇論文能突破一次又一次的難關。

最後，要感謝我的父母在我求學的過程之中，感謝父母讓我在衣食無慮、無後顧之憂的情形下，得以專心致力於課業及研究，並且也在適當時機給予我無盡的支持、關心與鼓勵，讓我能順利的完成學業。在此謹將此論文獻給我的父母及所有幫助過我的人，感謝父母多年來的培育之恩，感謝實驗室同學們的砥礪之情，感謝我的指導教授朱正忠教授，在老師身旁學習的一切，將成為我最珍貴的寶藏。

附錄

附錄A. Office Organizer 操作畫面

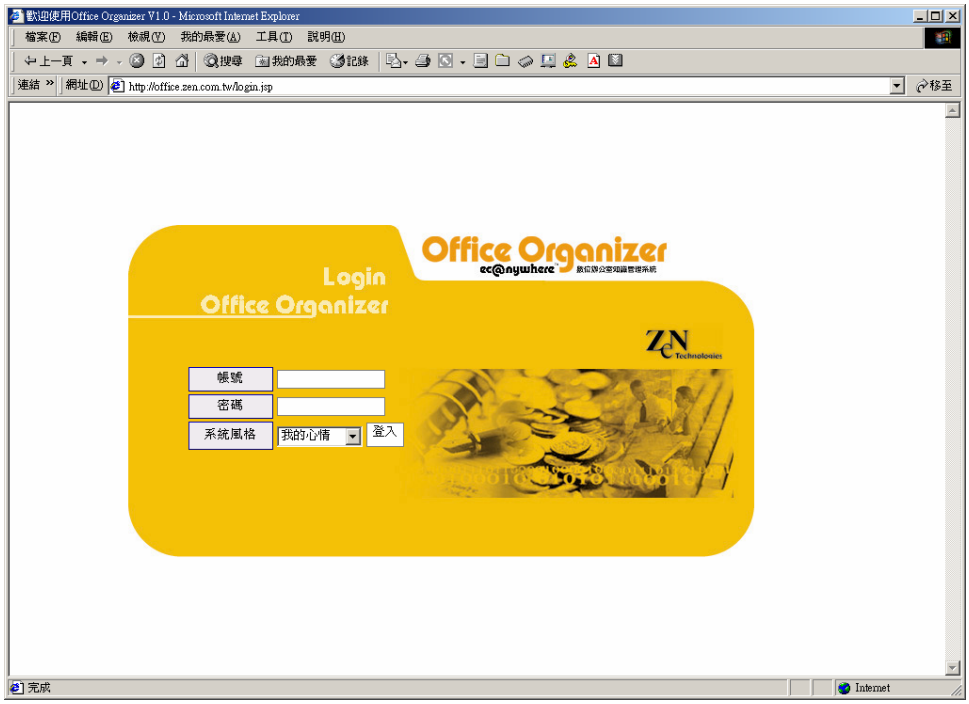

圖 34 登入

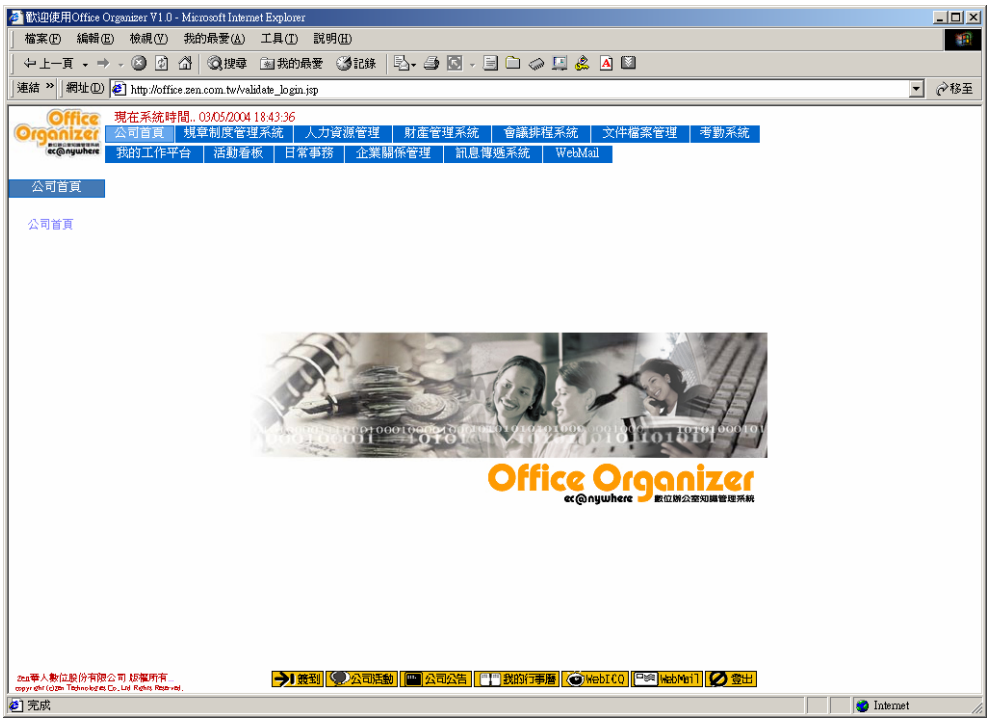

圖 35 主畫面

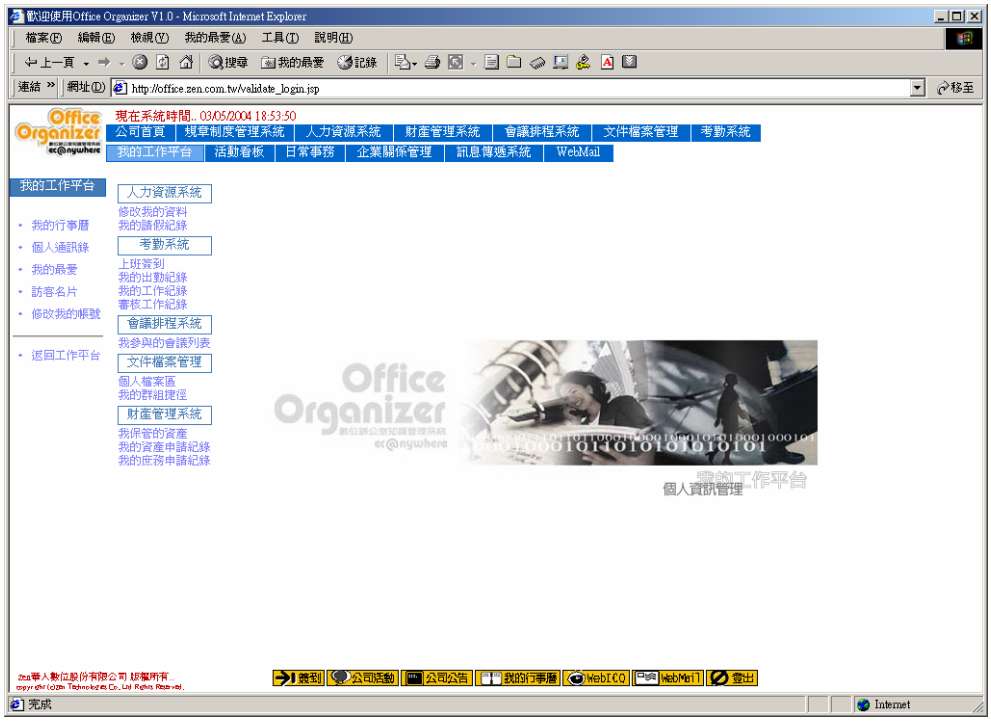
歡迎使用Office Organizer V1.0 - Microsoft Internet Explorer
檔案(F) 編輯(E) 檢視(V) 我的最愛(A) 工具(T) 說明(H)
上一頁 搜尋 我的最愛 記錄
連結 網址(D) http://office.zen.com.tw/validate_login.jsp
移至
Office Organizer
現在系統時間.. 03/05/2004 18:53:50
公司首頁 規章制度管理系統 人力資源系統 財產管理系統 會議排程系統 文件檔案管理 考勤系統
我的工作平台 活動看板 日常事務 企業關係管理 訊息簡遞系統 WebMail
我的工作平台
我的行事曆
個人通訊錄
我的最愛
訪客名片
修改我的帳號
返回工作平台
人力資源系統
修改我的資料
我的請假紀錄
考勤系統
上班簽到
我的出勤紀錄
我的工作紀錄
審核工作紀錄
會議排程系統
我參與的會議列表
文件檔案管理
個人檔案區
我的群組捷徑
財產管理系統
我保管的資產
我的資產申請紀錄
我的庭務申請紀錄
Office Organizer
我的工作平台
個人資訊管理
簽到 公司活動 公司公告 我的行事曆 WebICQ WebMail 登出
完成
Internet

圖 36 工作平台

附錄B. Office Organizer 功能模組說明

表 6 功能模組

名稱	說明
公司組織	提供公司組織圖設計，使用者可以自己設定所屬組織之階層圖。
上下班時間	設定公司上下班時段，並且可以查詢個人上下班時間紀錄。
公司規章	建立公司規章制度。
人事資料	個人資料維護。根據權限，可以修改自己的個人資料。
請假管理	在系統中直接提出請假需求，並透過網路得到結果。
資產管理	對於公司內部的有價資產作管理。
庶務管理	管理公司內部一般雜物事項。
會議排程	統一行事曆，可以安排共通的行程及獨立的工作排程。
會議室管理	設定會議室利用時間。
檔案管理	使用者可以上傳資料，並設定分享對象，以達到資料共享的目的。
簽到系統	線上打卡系統，使用者直接登錄系統簽到，並且於月末可以直接得到缺席紀錄。,

附錄C. Office Organizer 案例圖

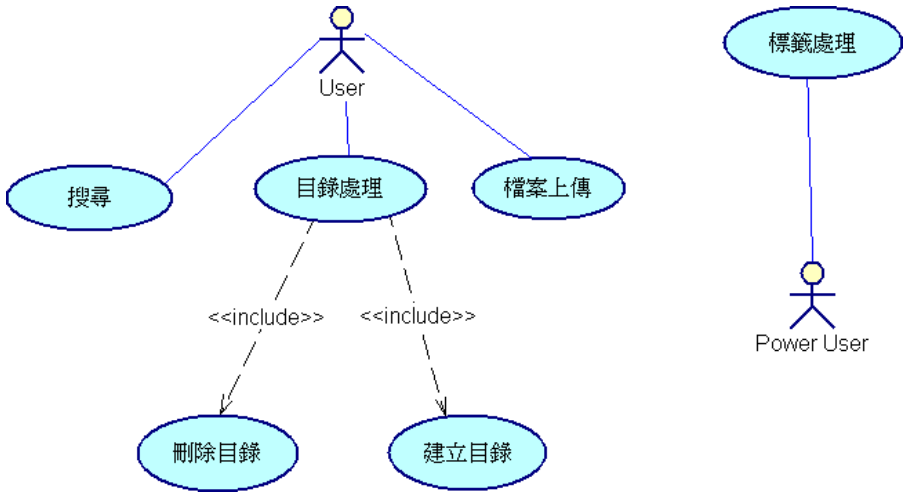

圖 37 文件管理系統

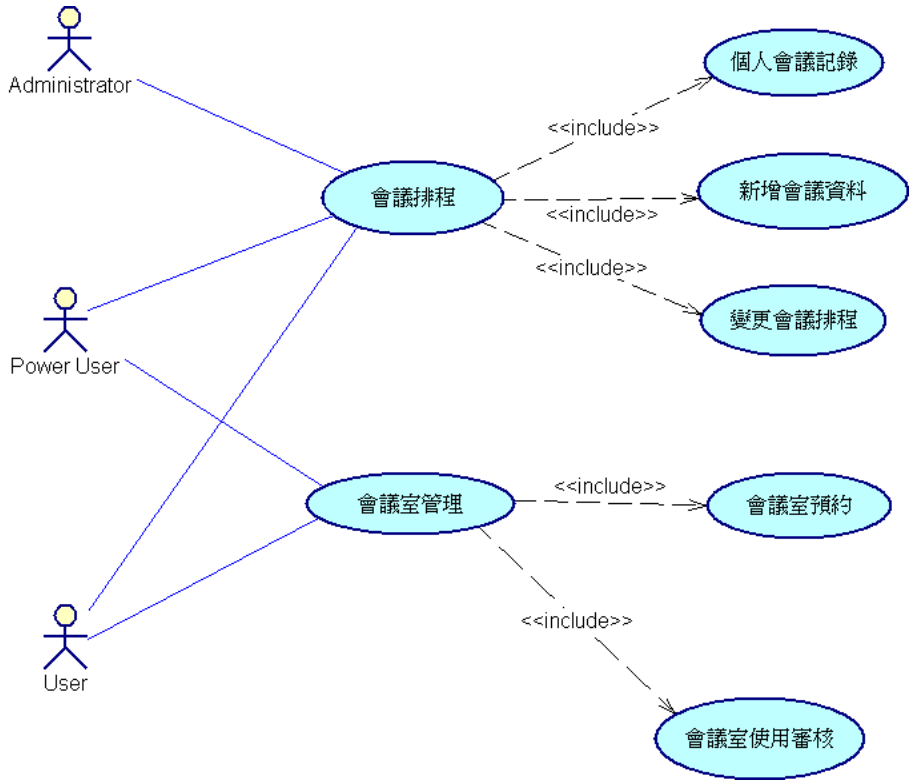

圖 38 會議排程管理

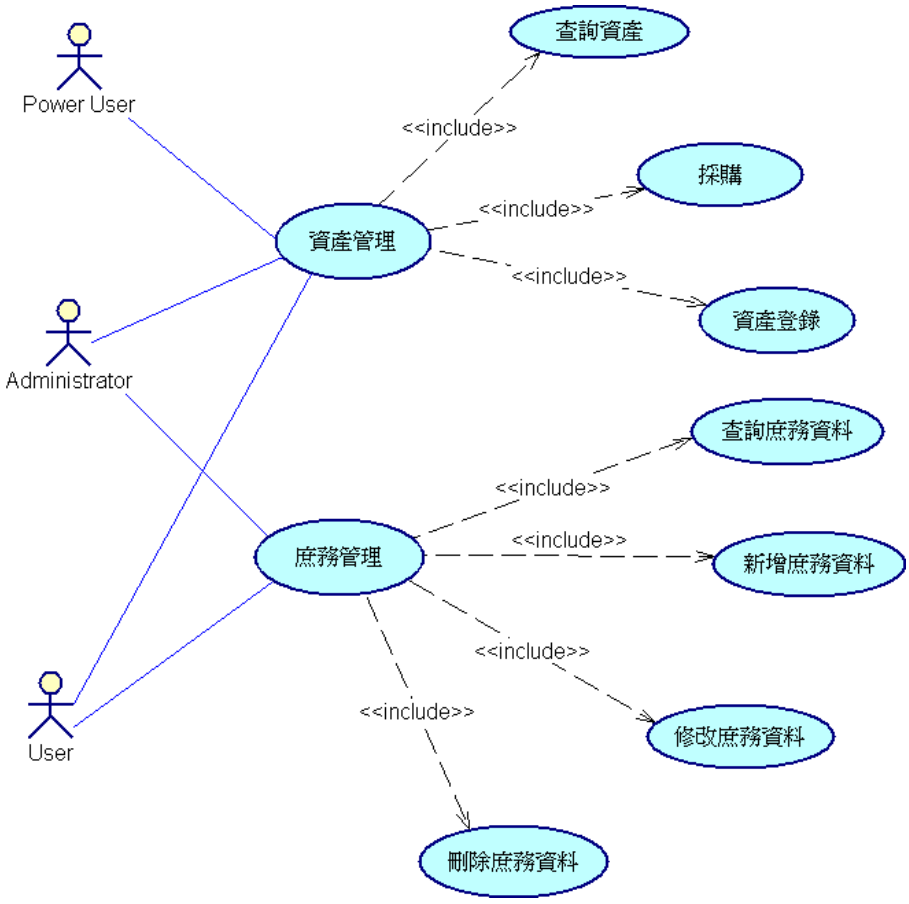

圖 39 財務管理系統

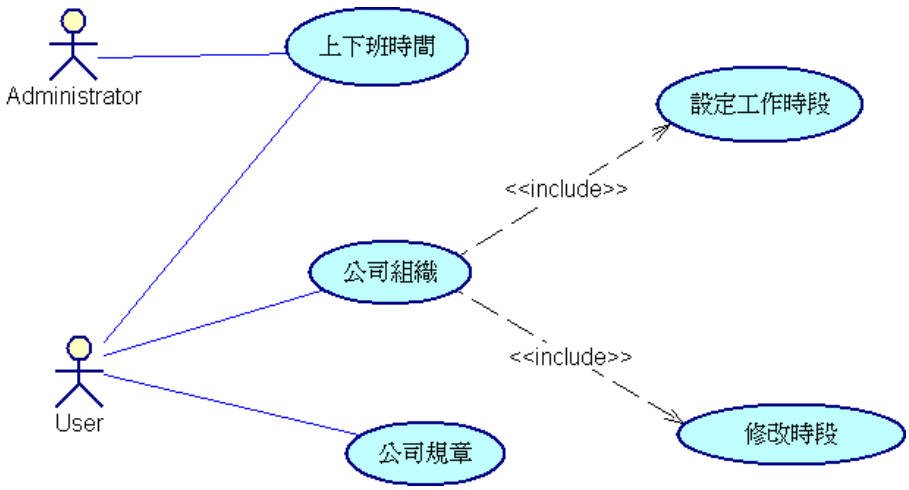

圖 40 規章制度管理

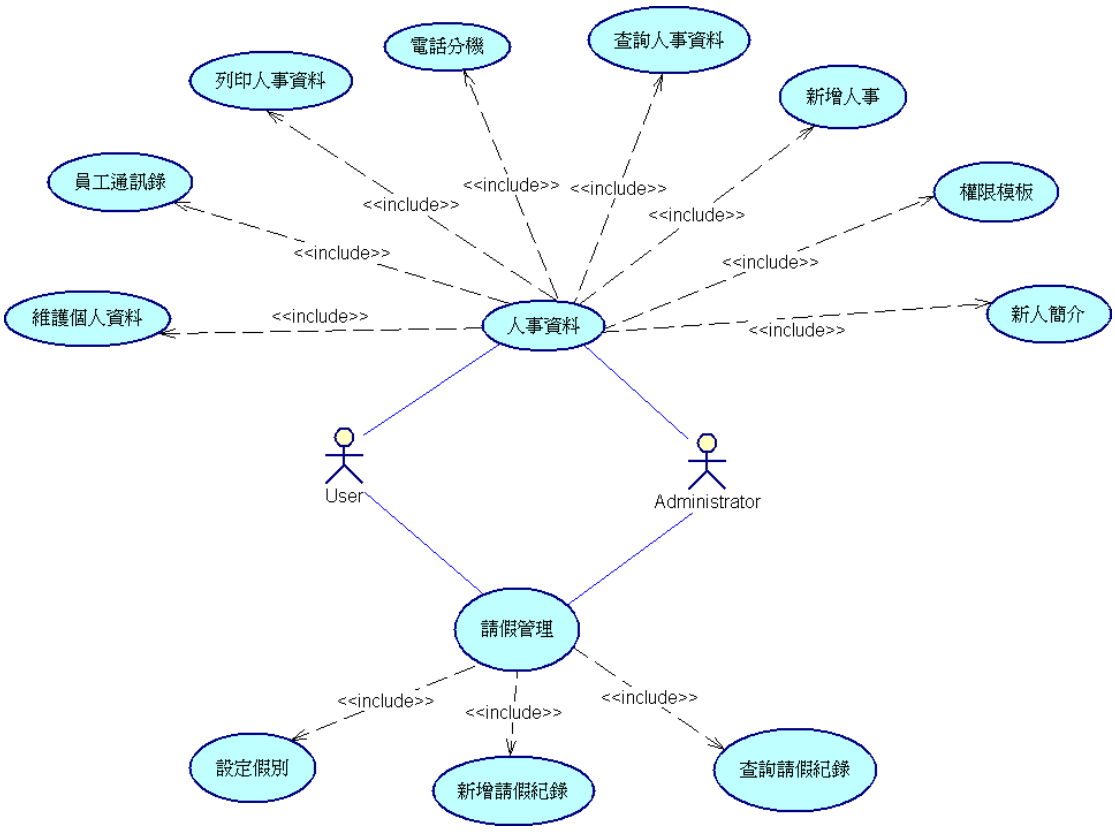
電話分機
查詢人事資料
列印人事資料
新增人事
員工通訊錄
權限模板
<<include>>
維護個人資料
人事資料
新人簡介
User
Administrator
請假管理
設定假別
新增請假紀錄
查詢請假紀錄

圖 41 人力資源管理

附錄D. 重整後 Office Organizer 類別圖

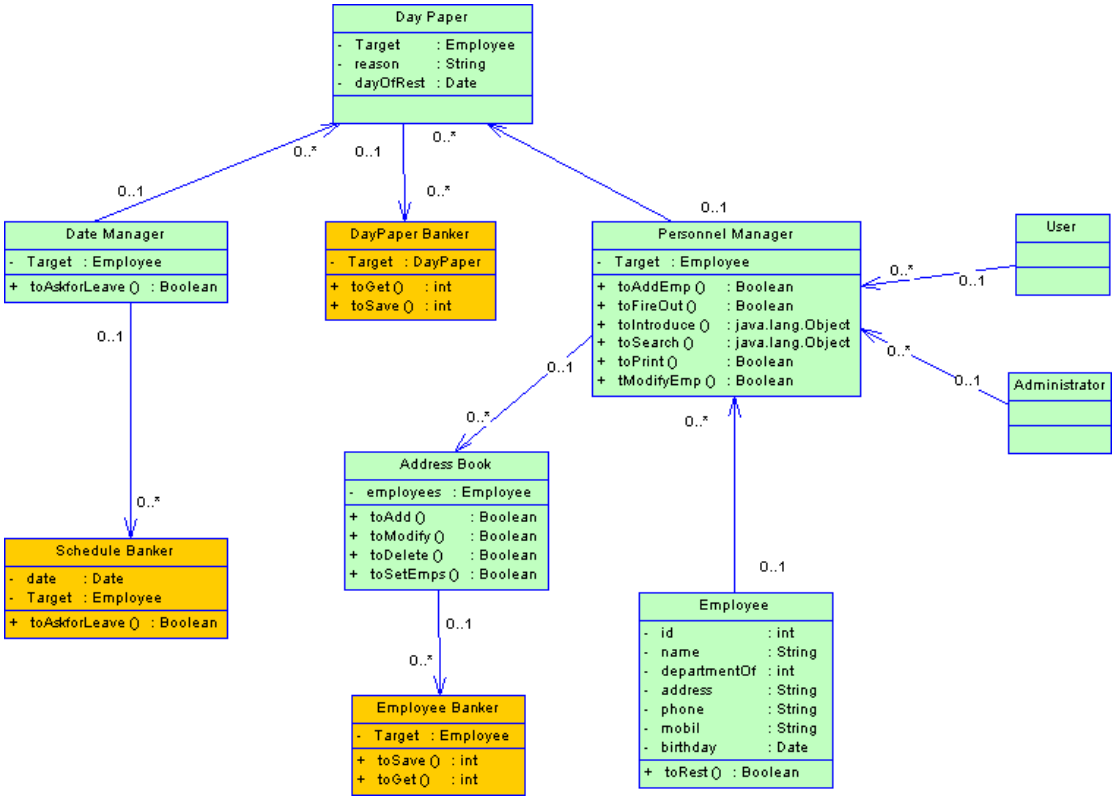

圖 42 人事系統類別圖

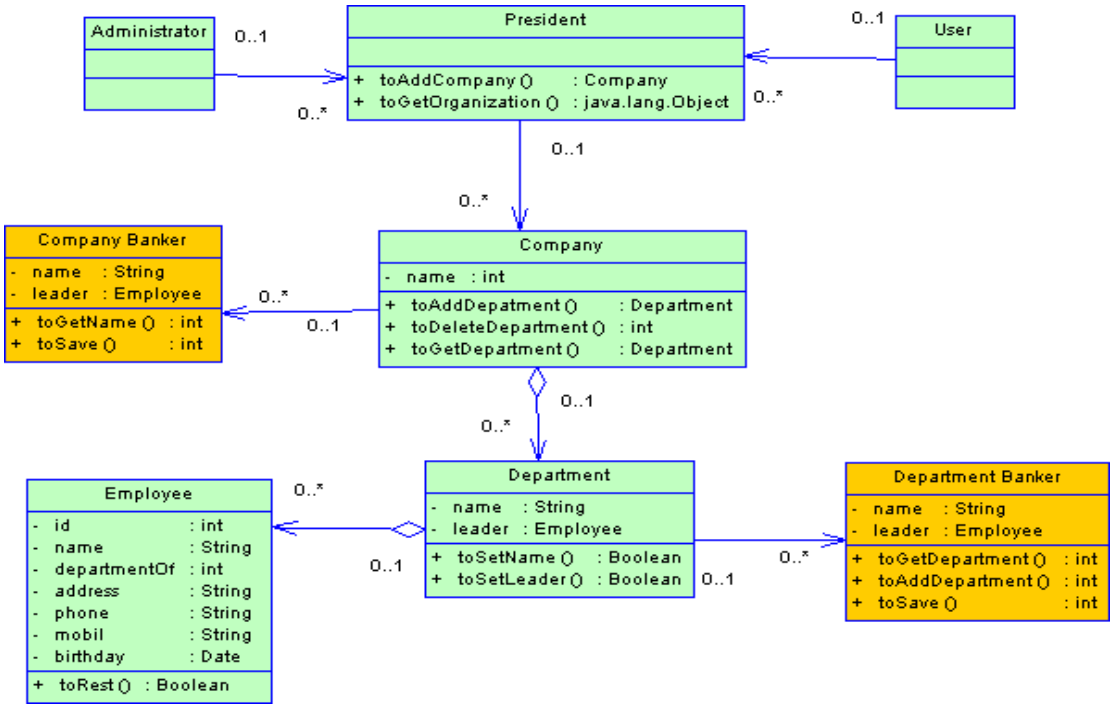

圖 43 規章制度系統類別圖

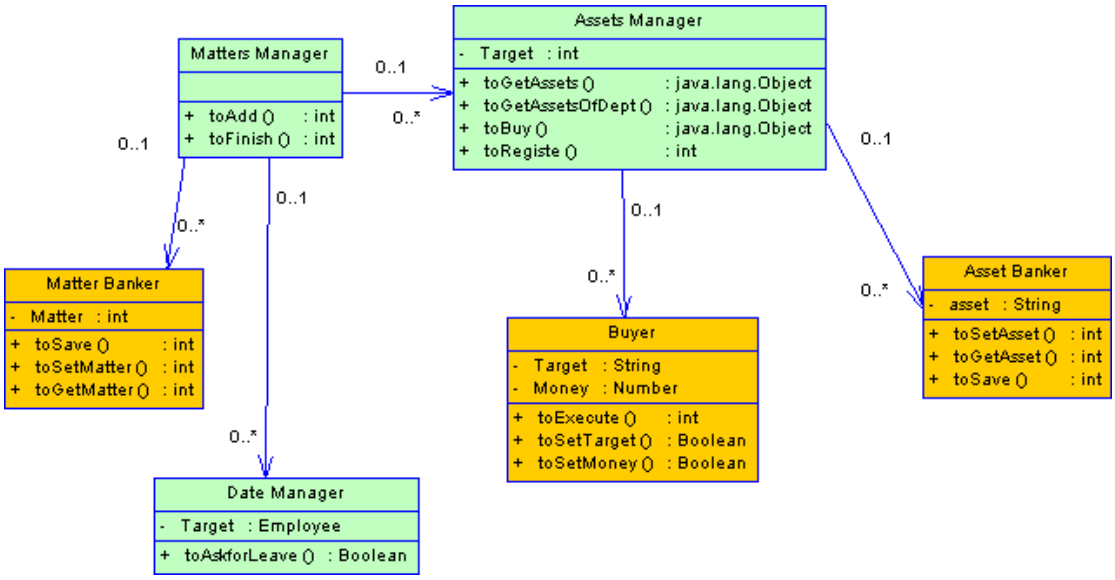

圖 44 財務系統類別圖

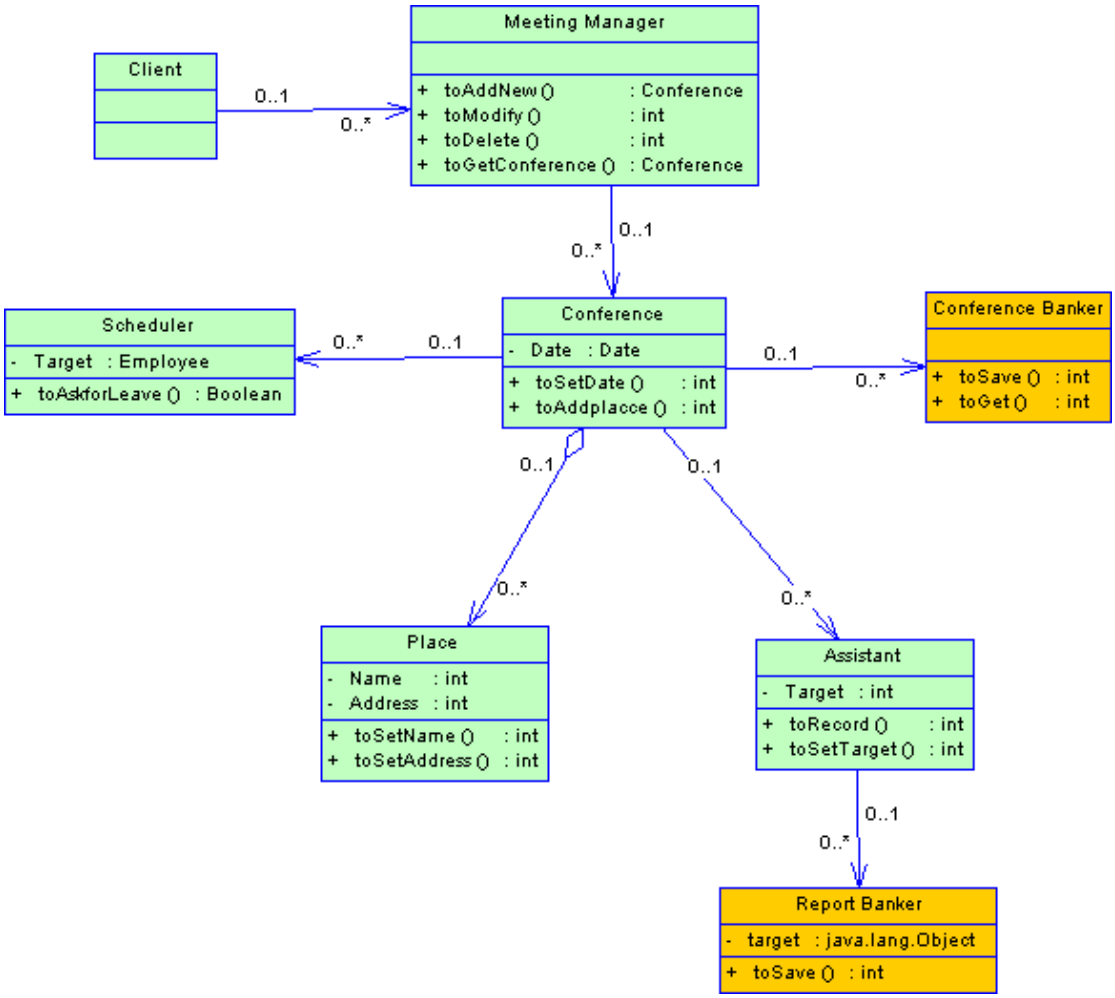

圖 45 會議管理系統類別圖

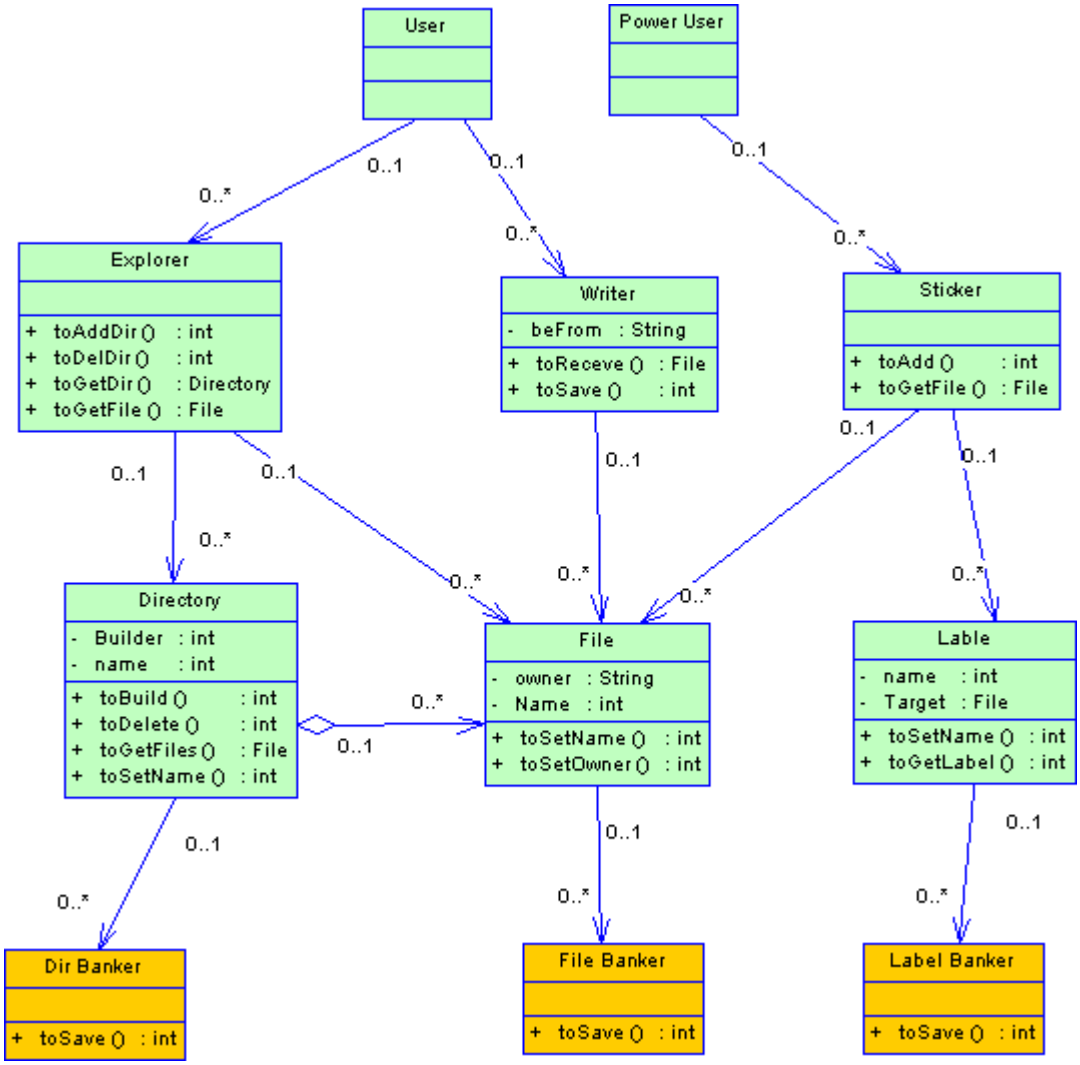

圖 46 文件管理系統類別圖

附錄E. Office Organizer 類別圖

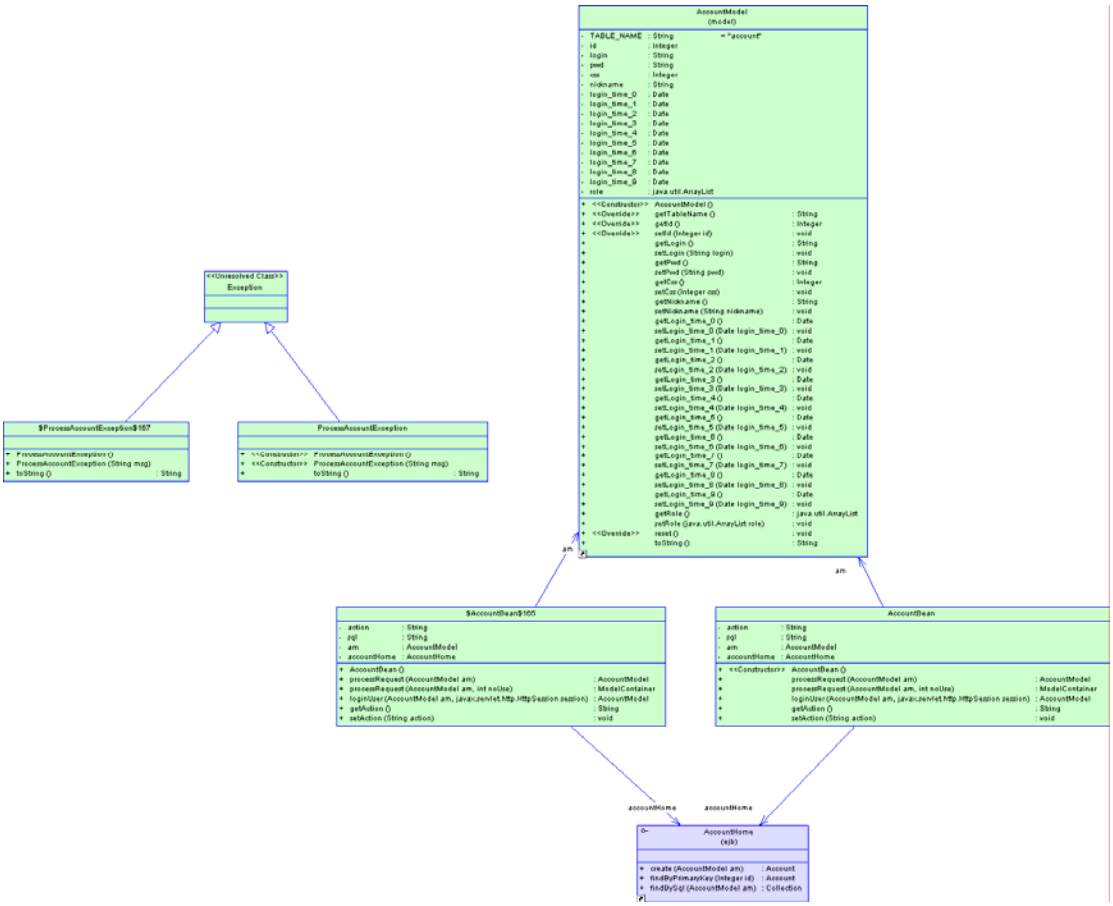

圖 47 com.zen.eoffice.acoount.adapter 類別圖

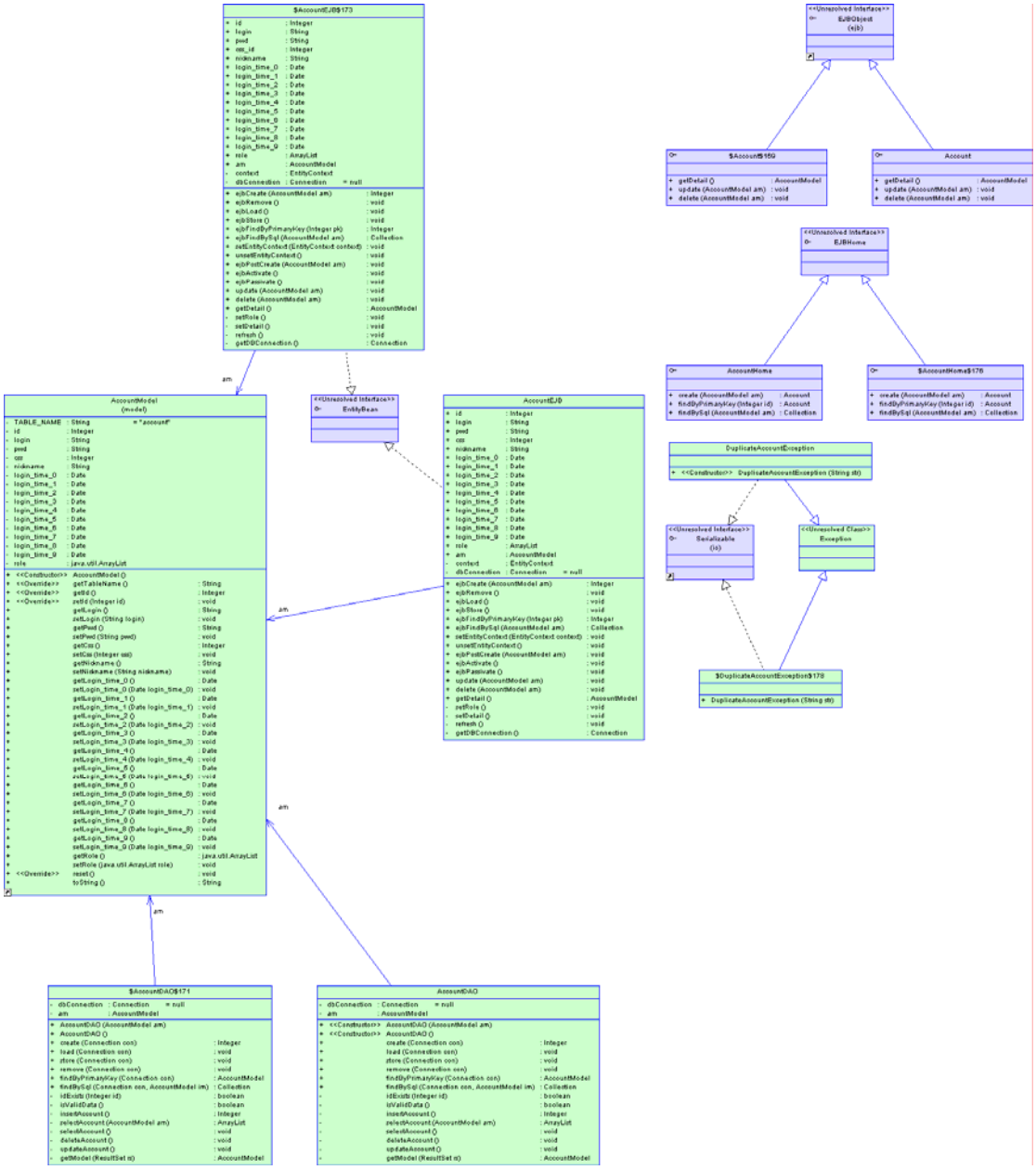

圖 48 com.zen.eoffice.acoount.ejb 類別圖

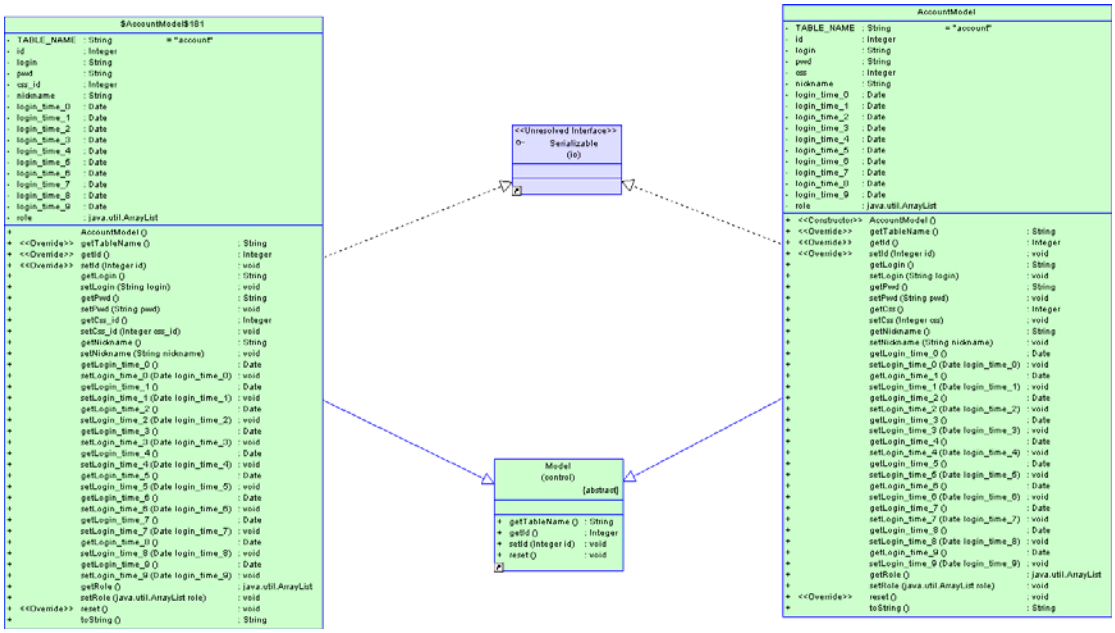

圖 49 com.zen.eoffice.acoount.model 類別圖

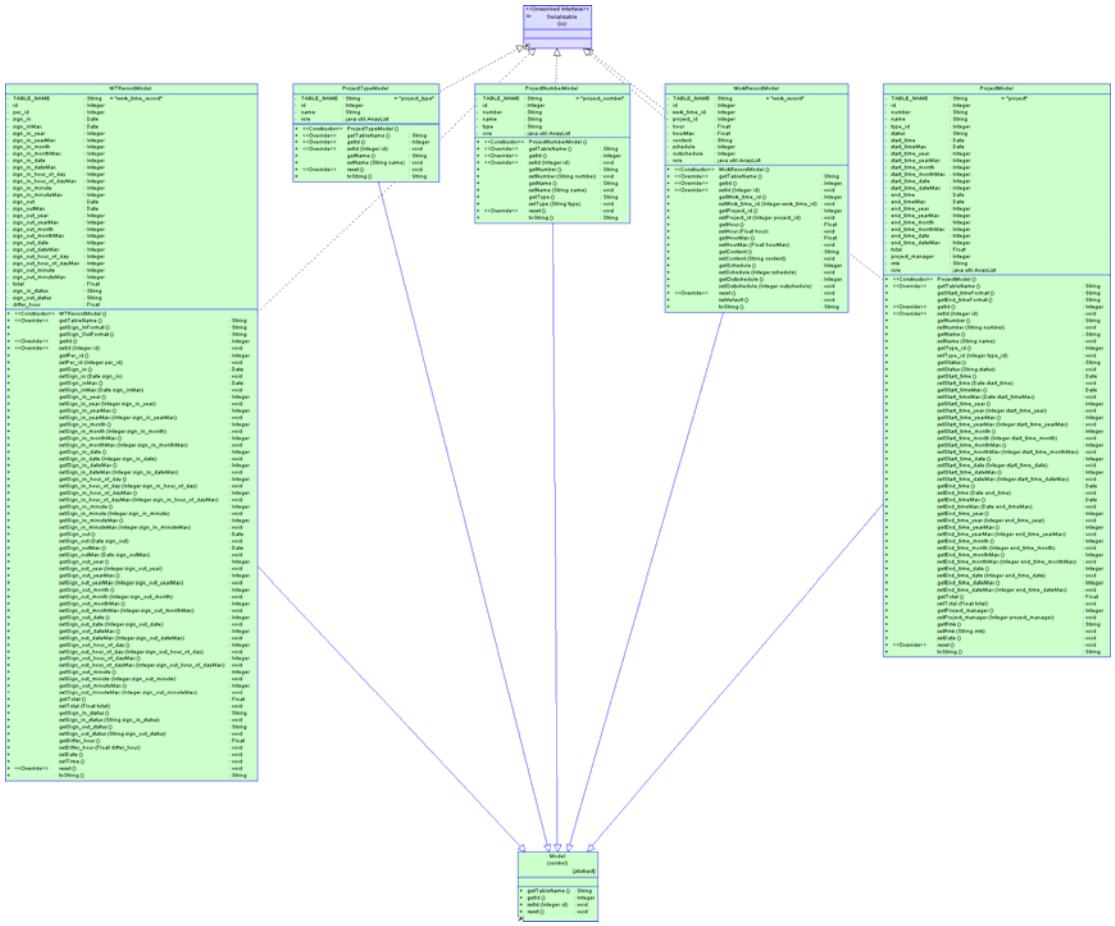

圖 50 com.zen.eoffice.check_attendance.adapter 類別圖

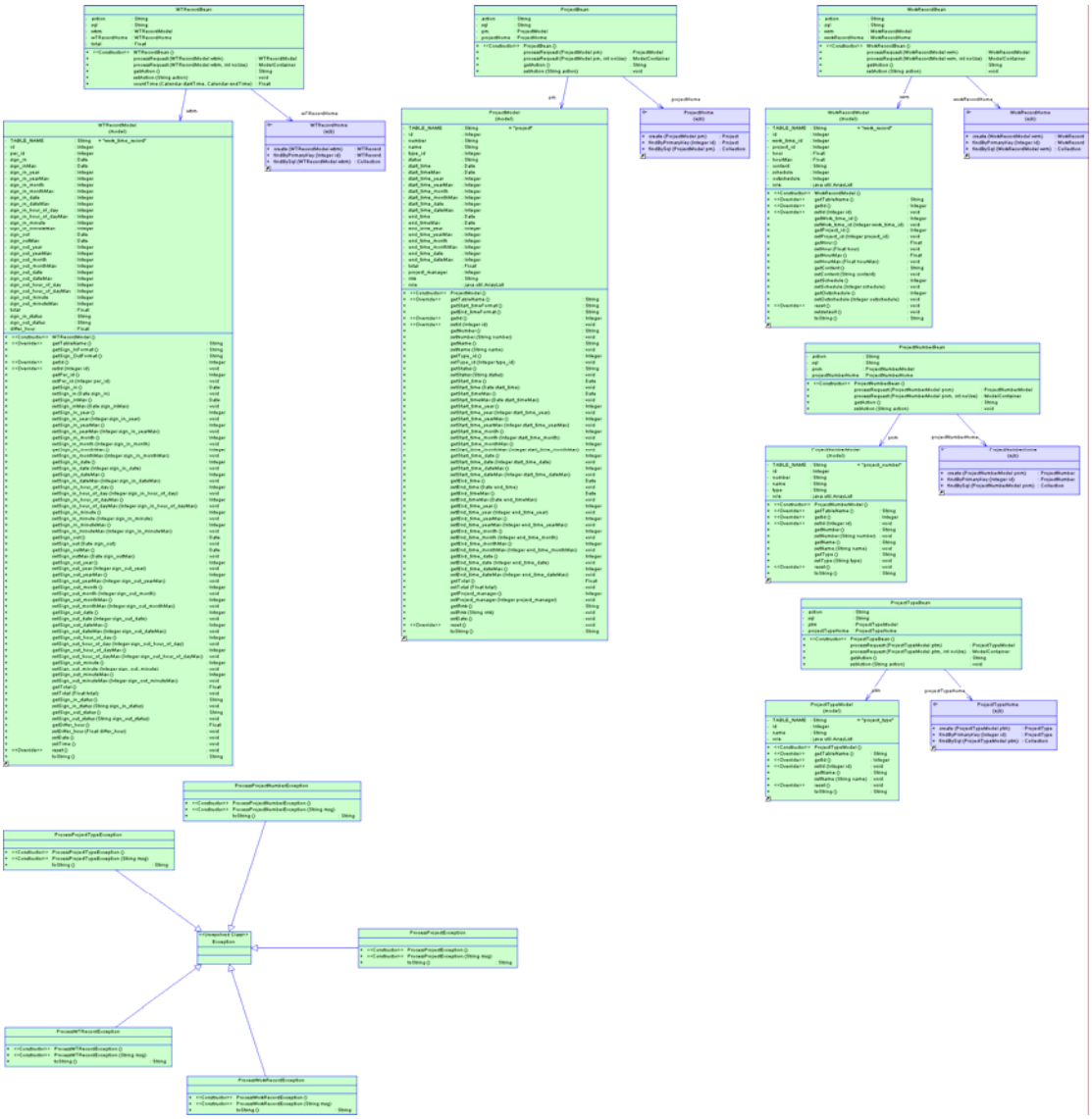

圖 51 com.zen.eoffice.check_attendance.ejb 類別圖

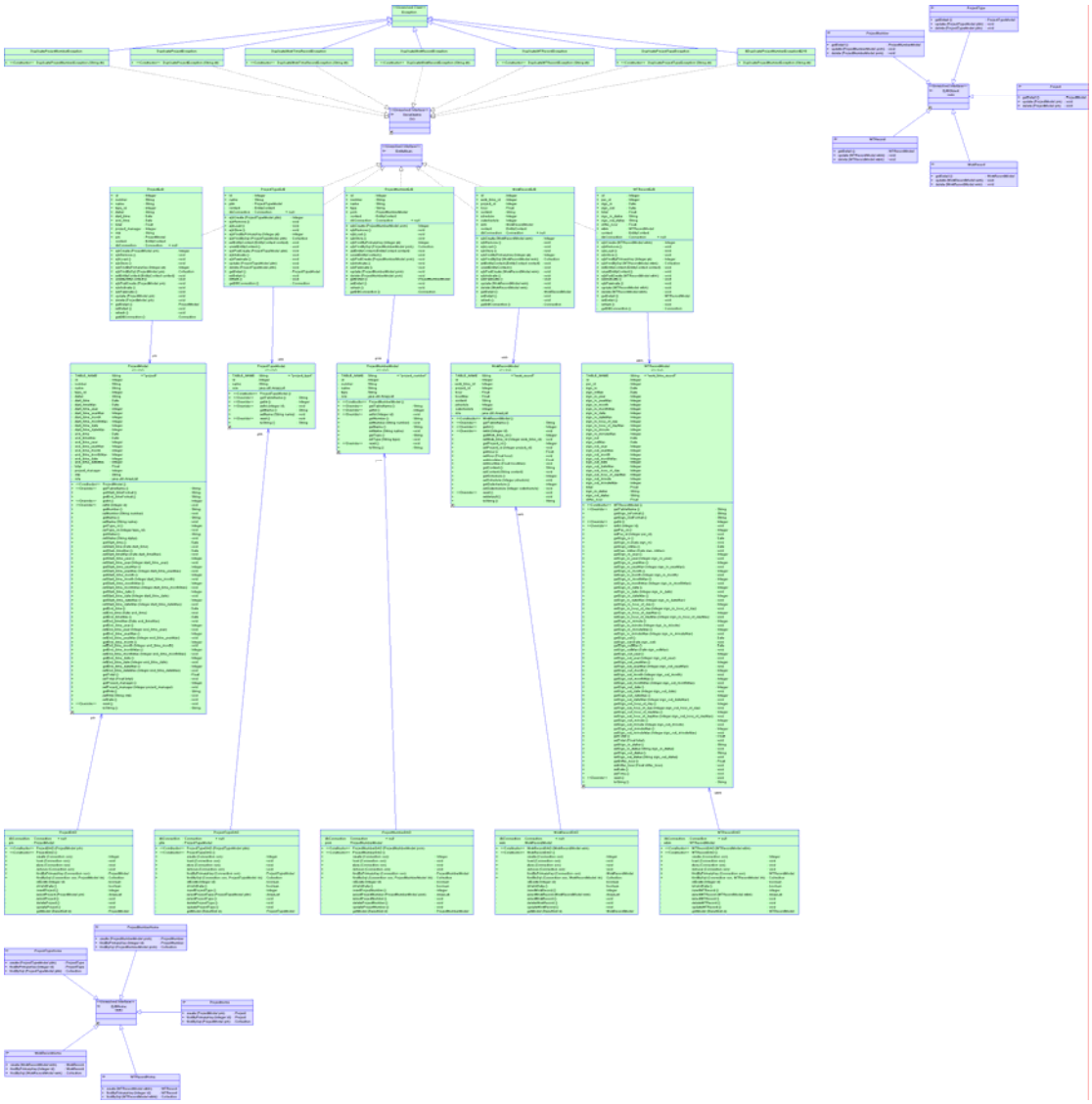

圖 52 com.zen.eoffice.check_attendance.model 類別圖

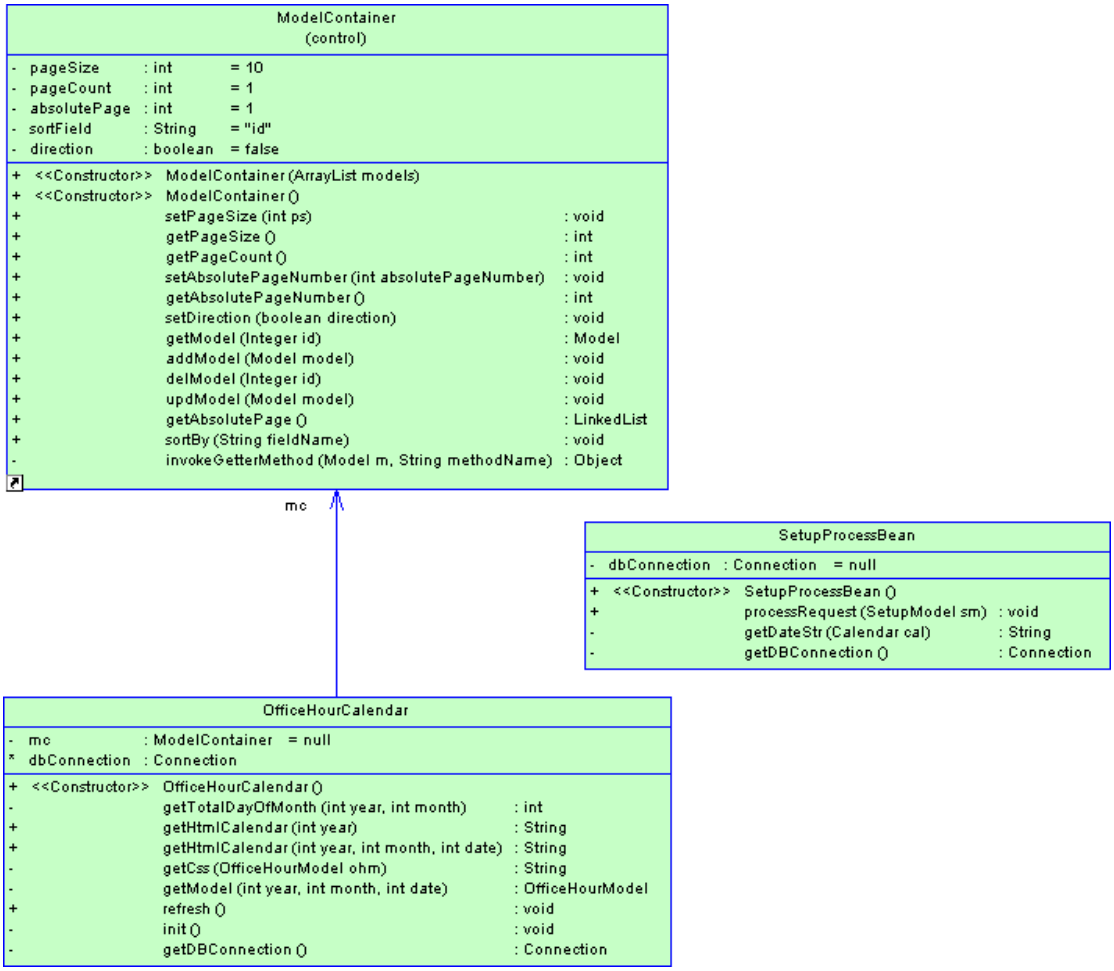

圖 53 com.zen.eoffice.company.adapter 類別圖

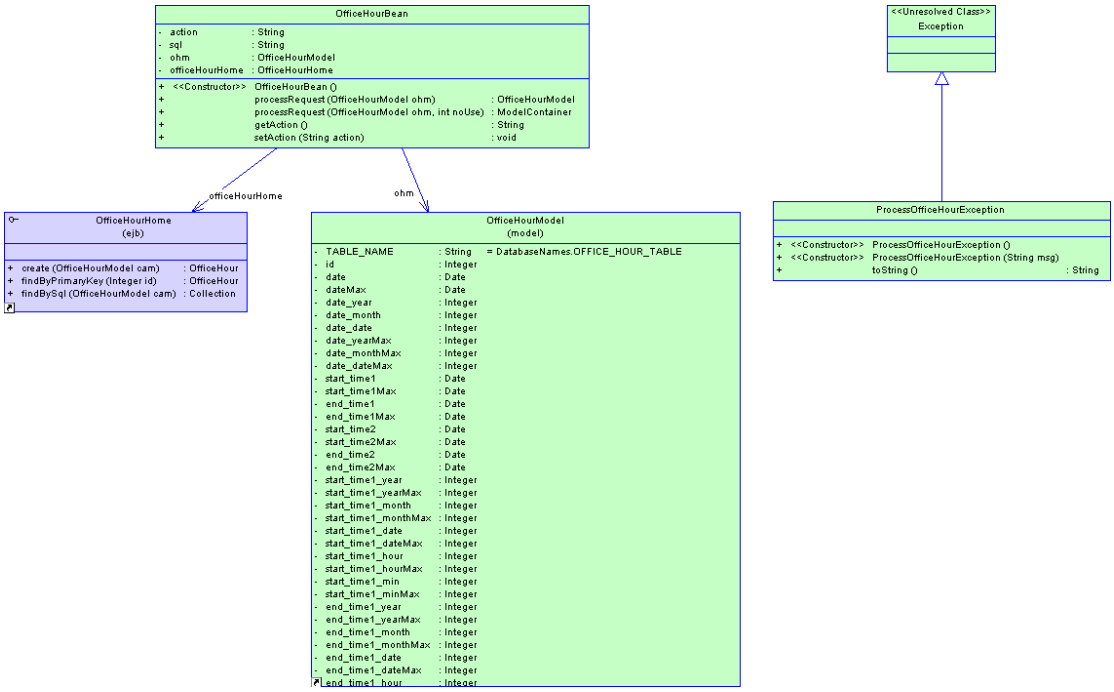

圖 54 com.zen.eoffice.company.ejb 類別圖

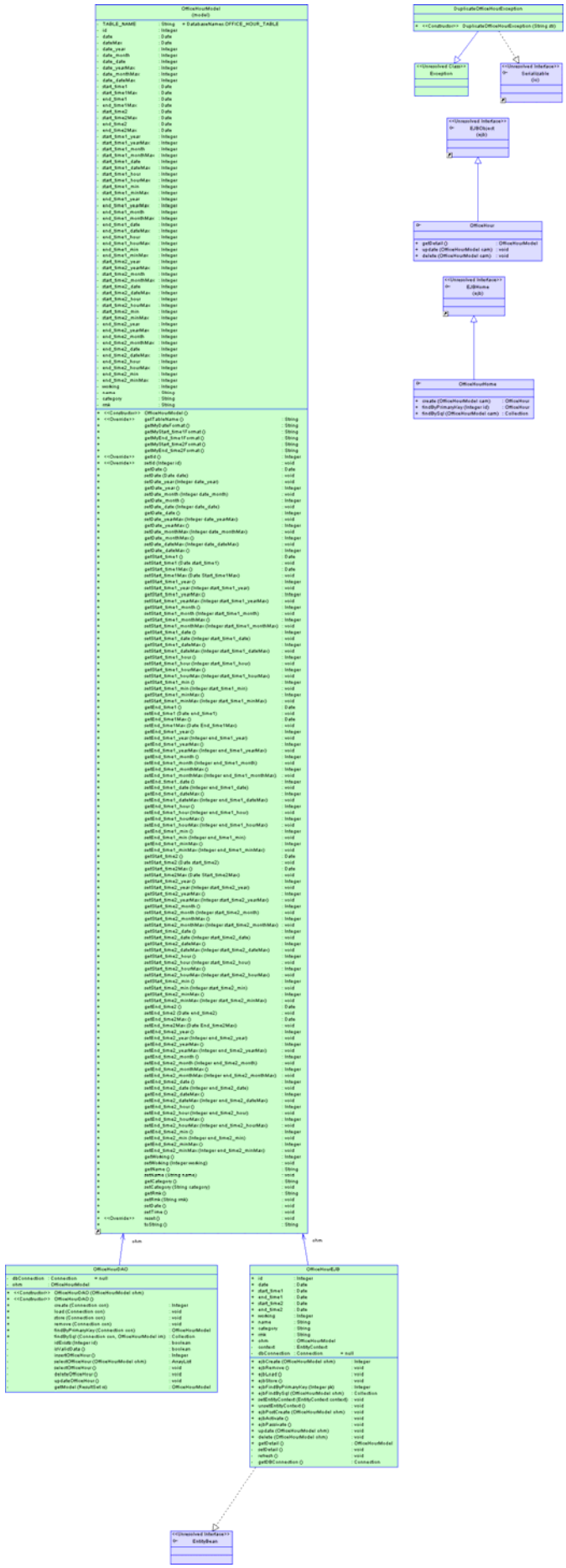

圖 55 com.zen.eoffice.company.model 類別圖

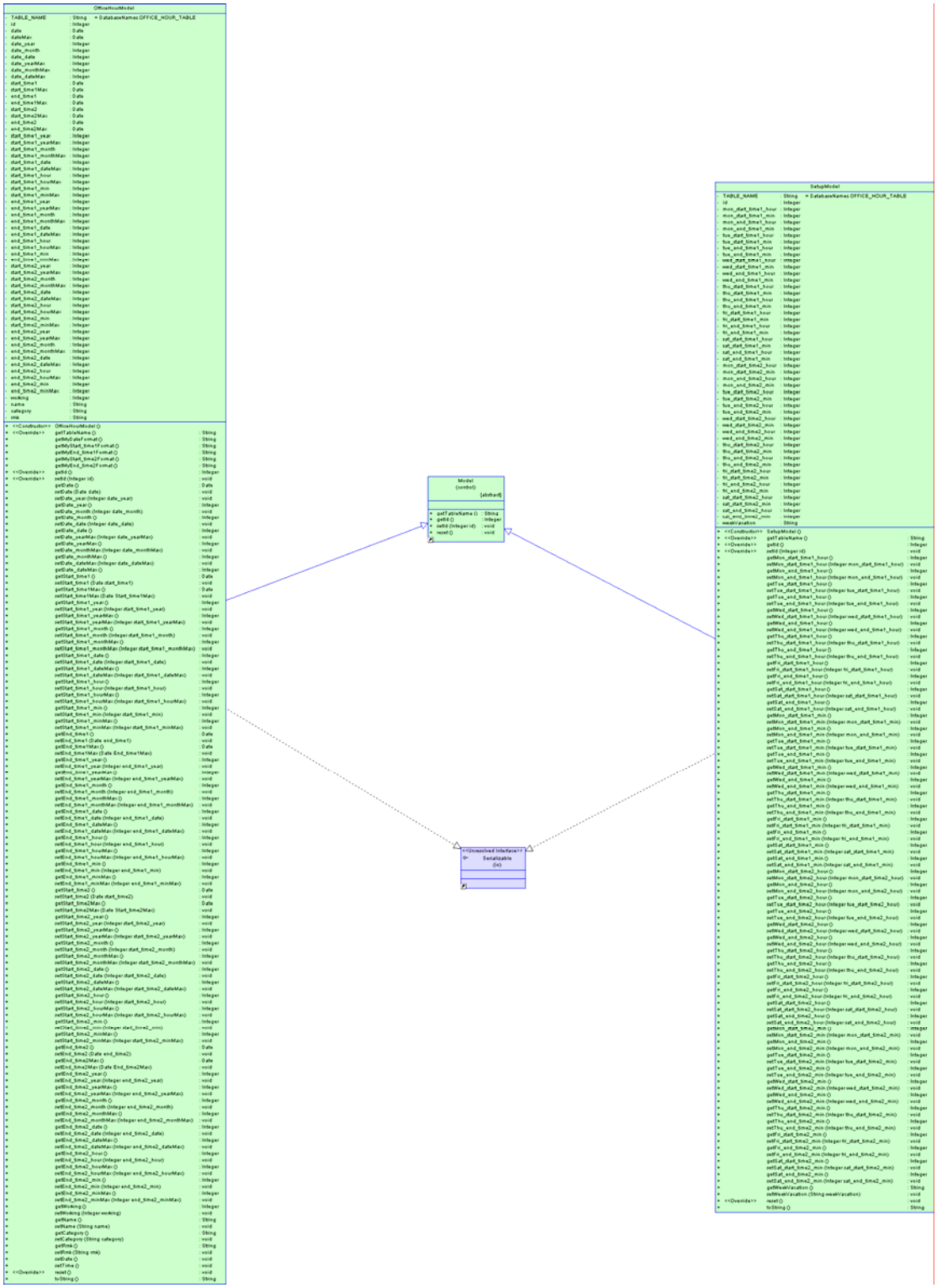

圖 56 com.zen.eoffice.company.util 類別圖

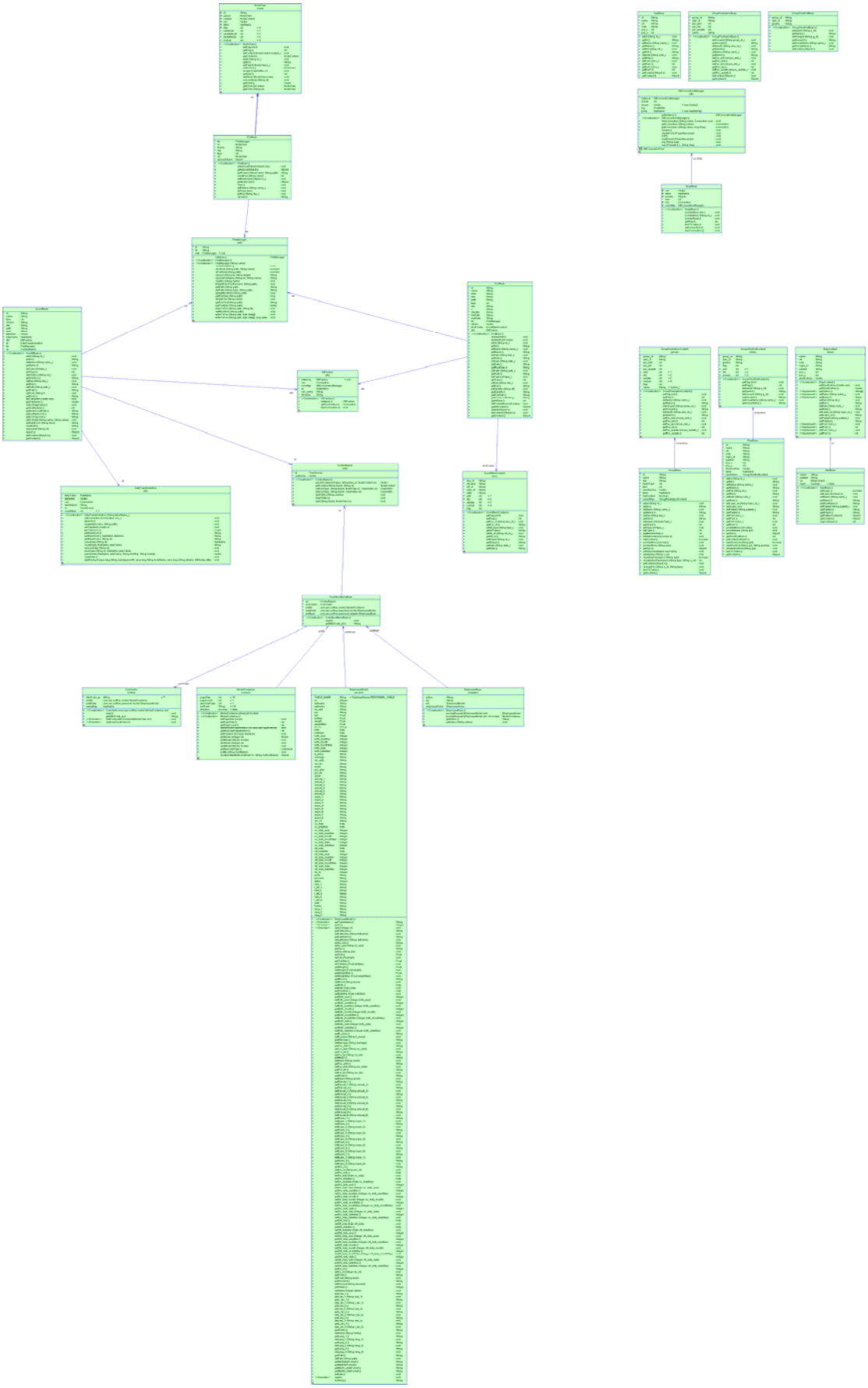

圖 57 eoffice.bean 類別圖

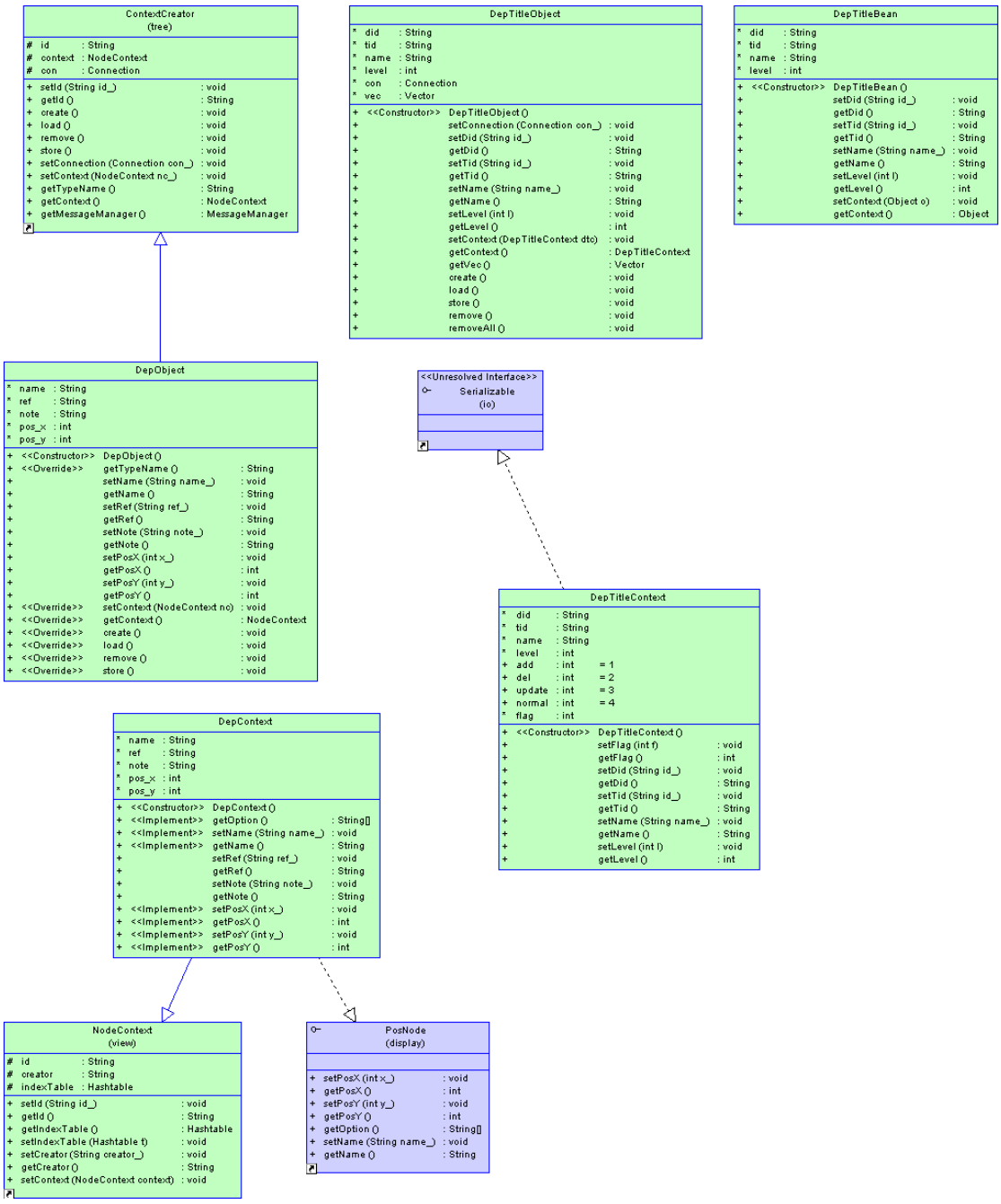

圖 58 eoffice.company.department 類別圖

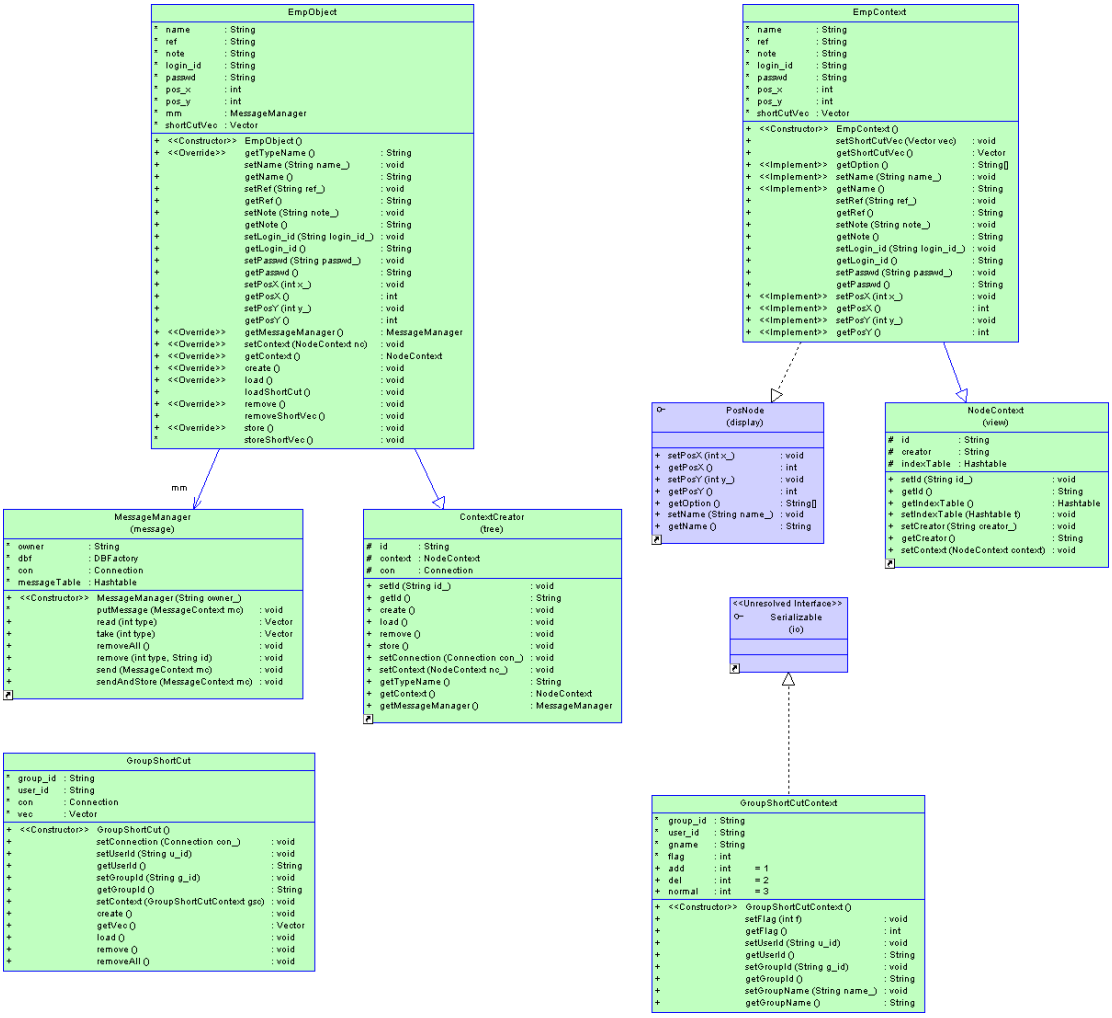

圖 59 eoffice.company.emp 類別圖

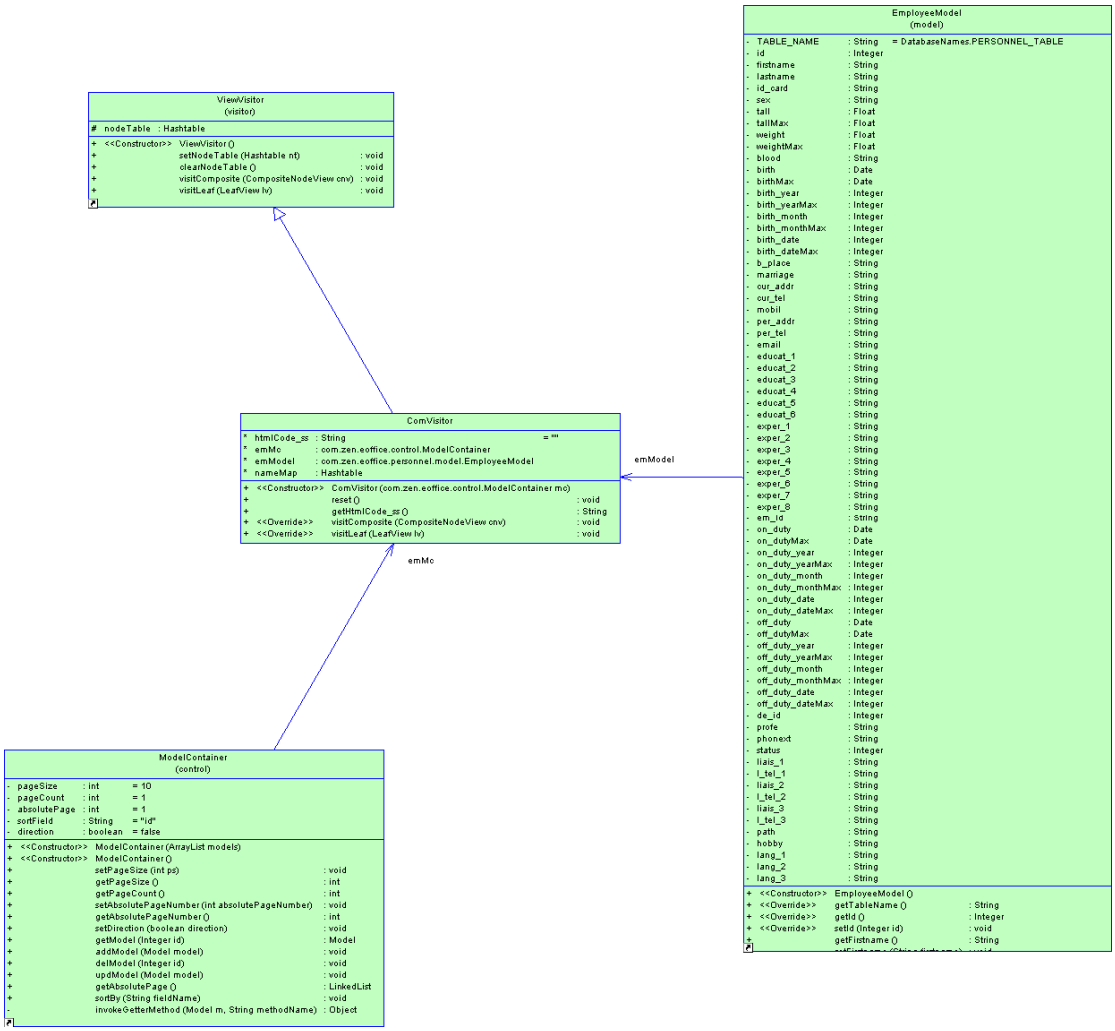

圖 60 eoffice.company.visitor 類別圖

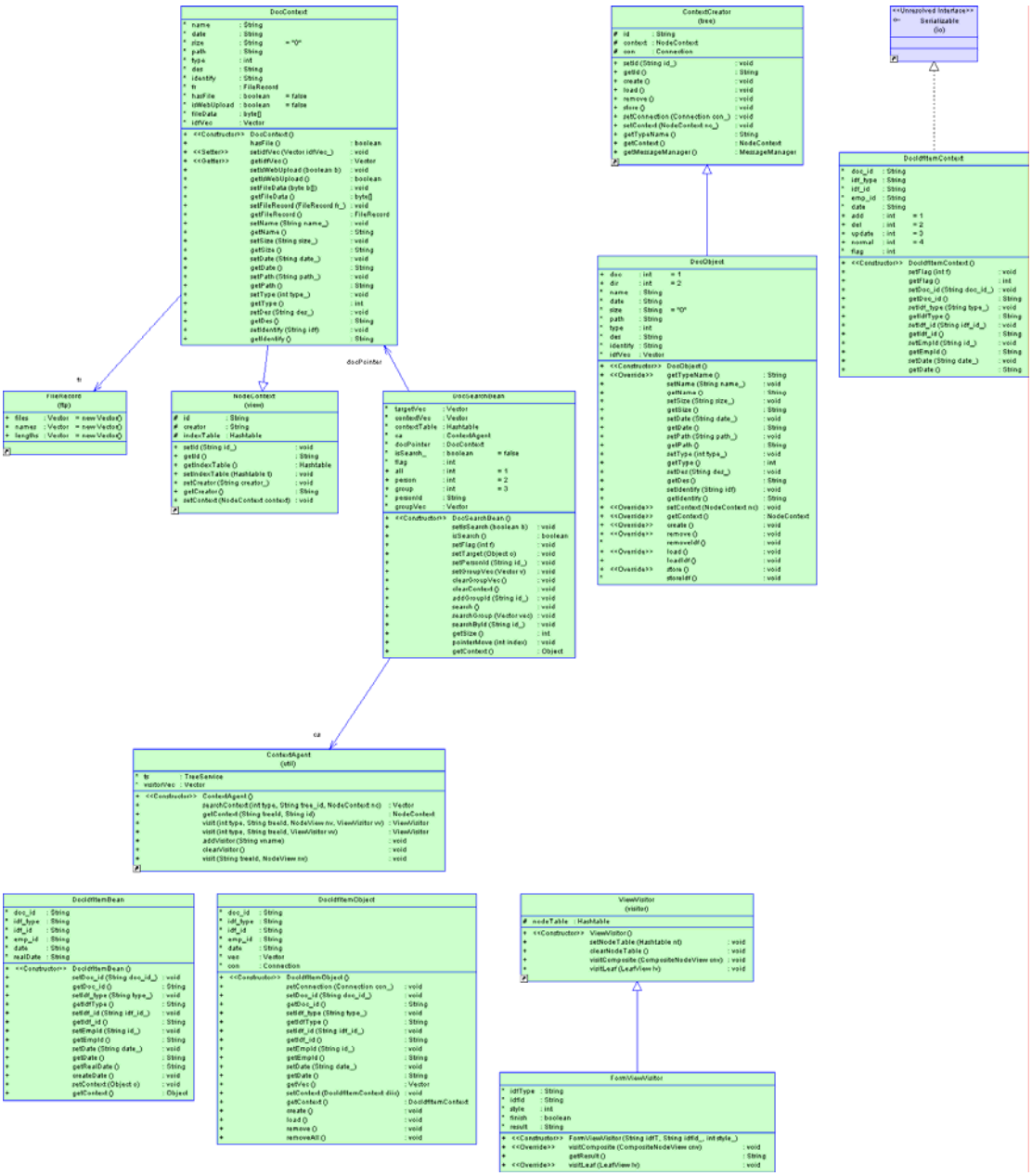

圖 61 eoffice.docm.doc 類別圖

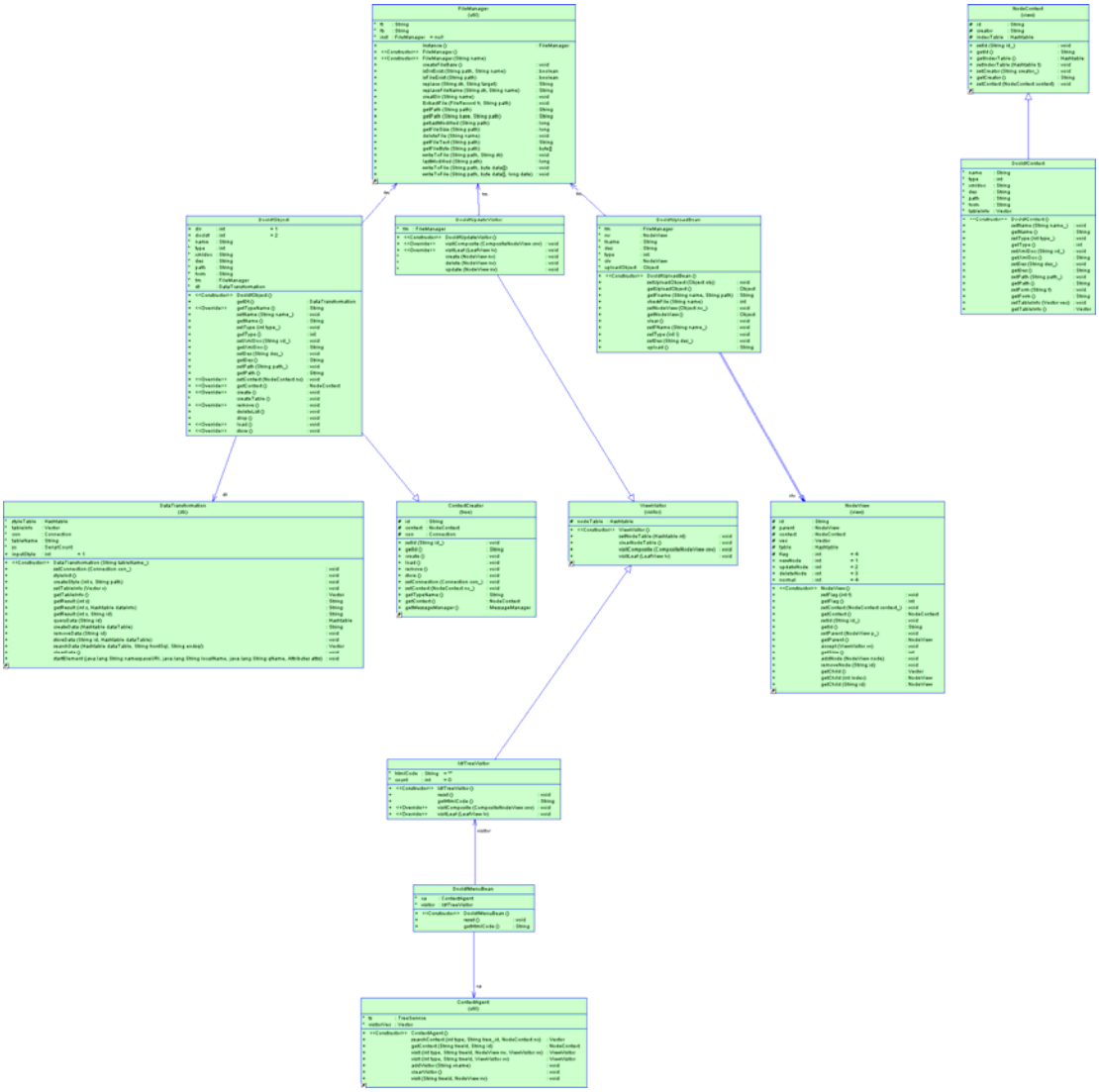

圖 62 eoffice.docm.docIdf 類別圖

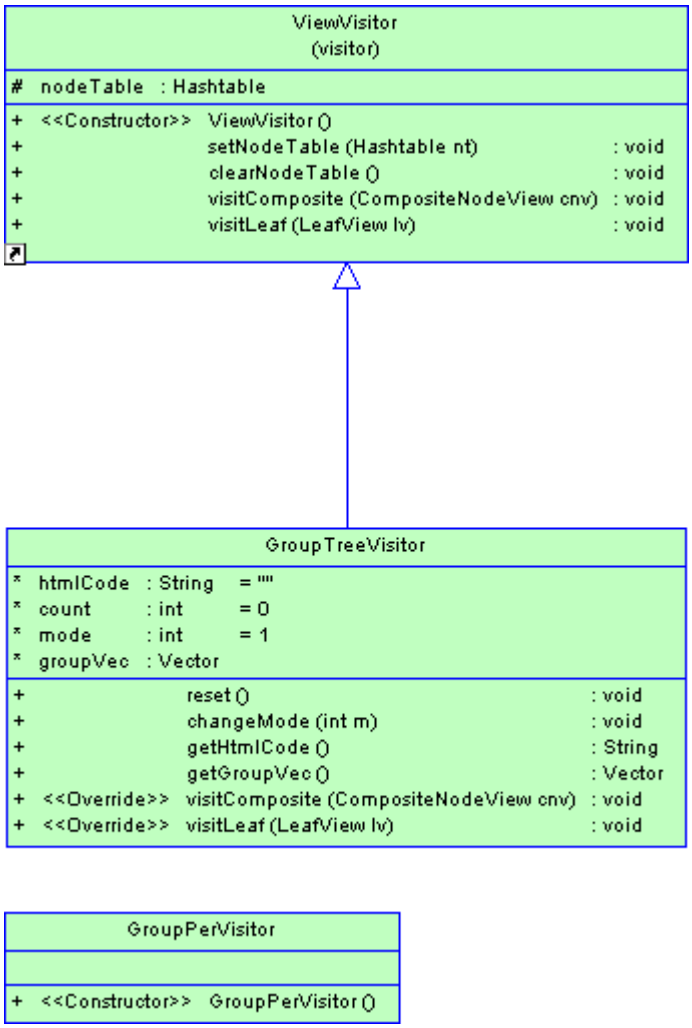

圖 63 eoffice.group.visitor 類別圖

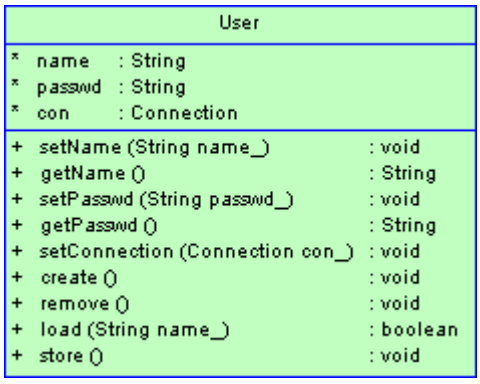

圖 64 eoffice.docm.user 類別圖

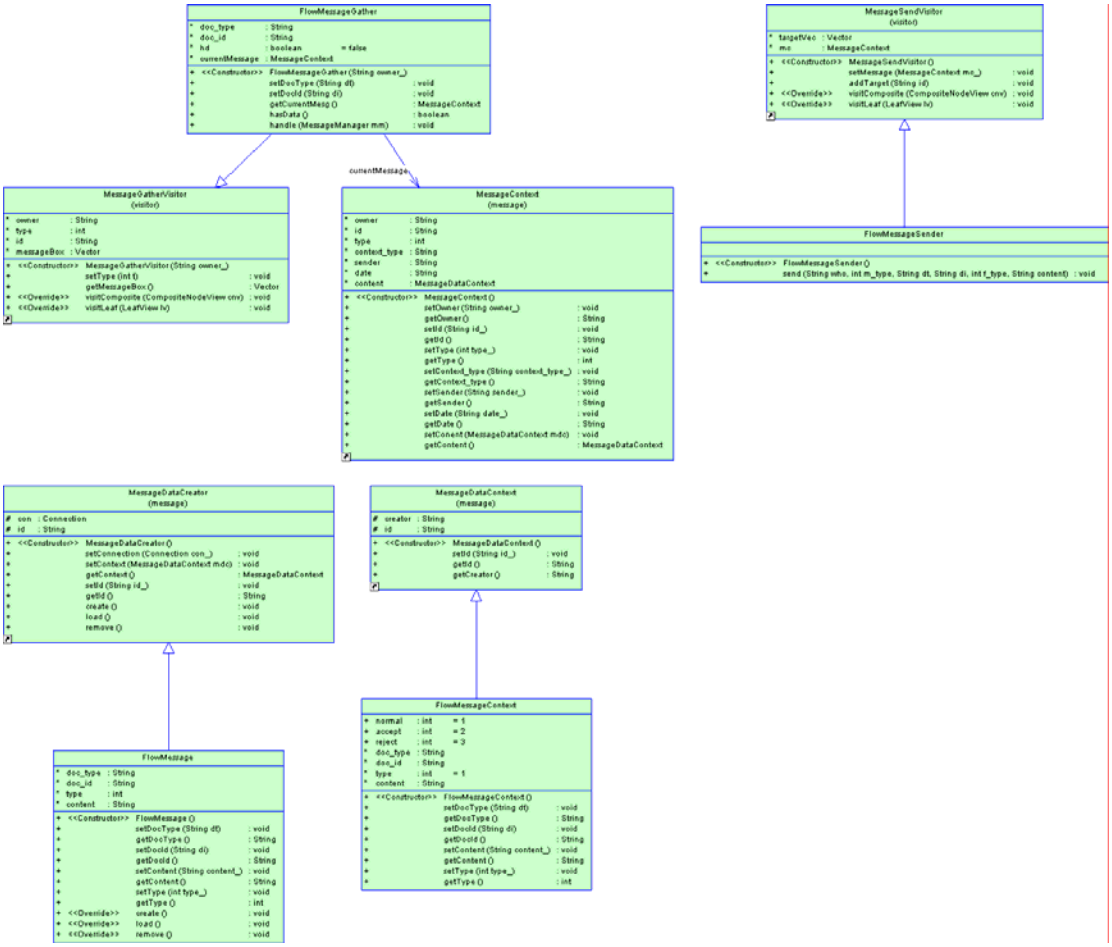

圖 65 eoffice.message.flowMesg 類別圖

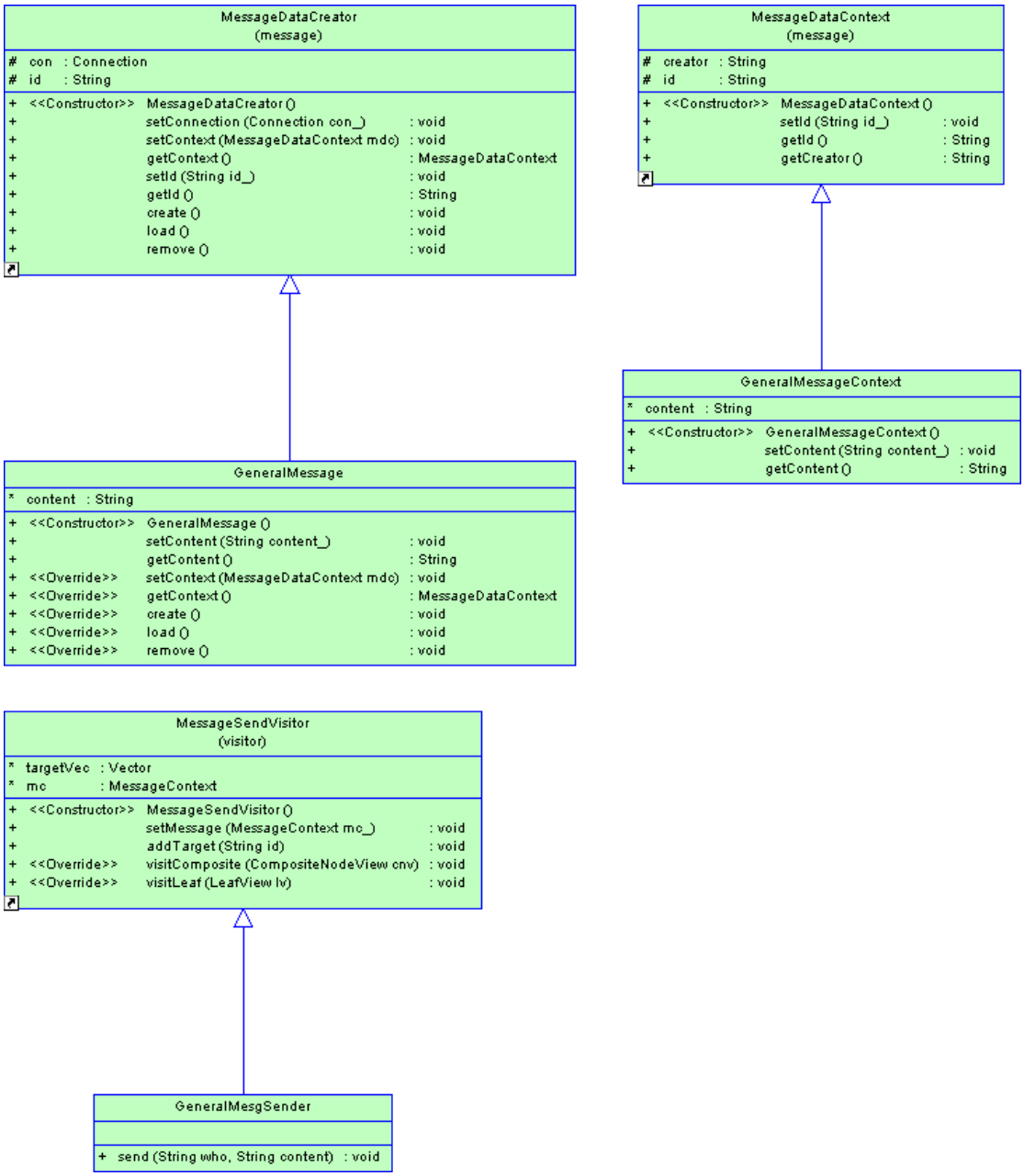

圖 66 eoffice.message.generalMessage 類別圖

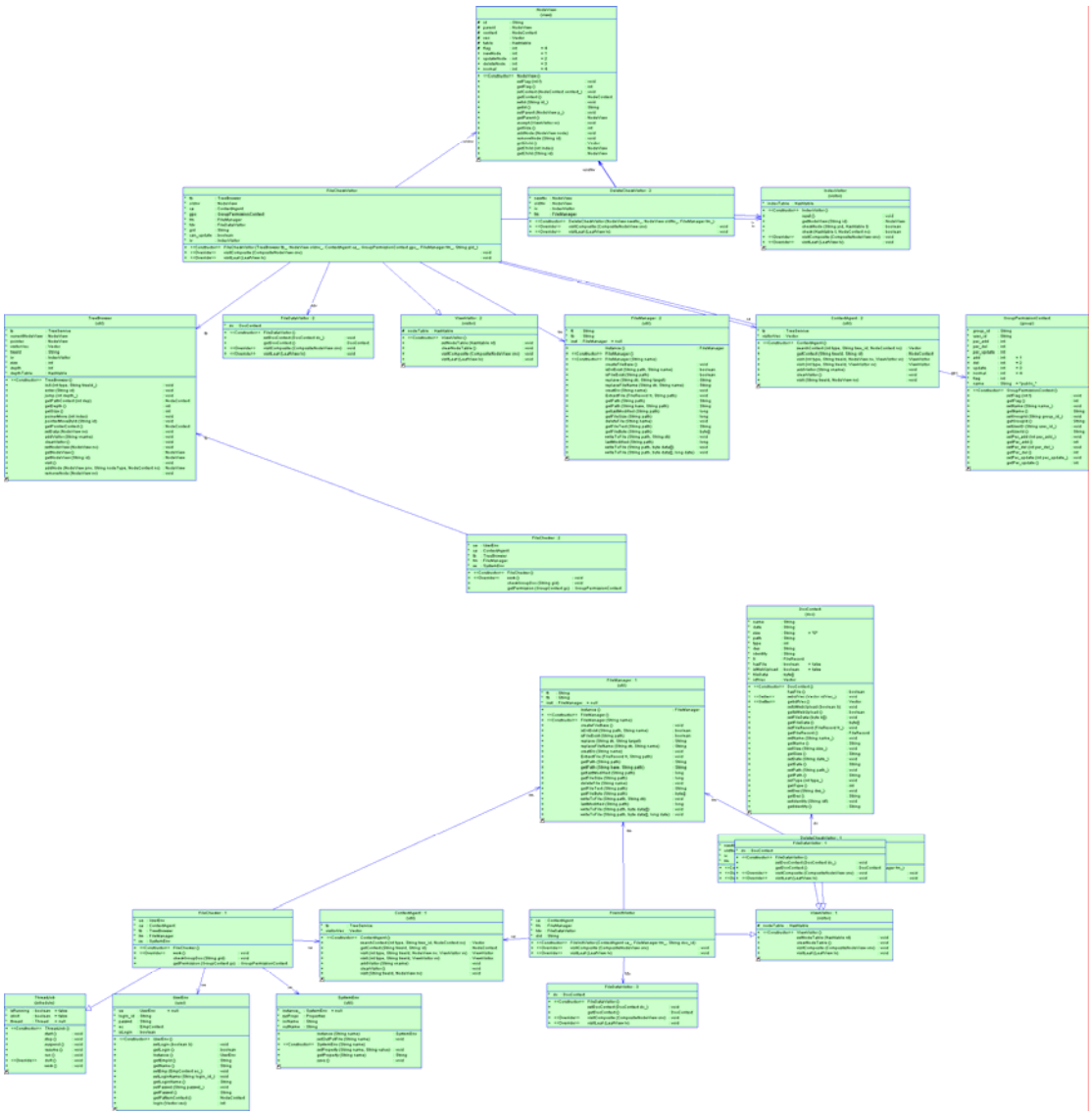

圖 67 eoffice.pa.job.filechecker 類別圖

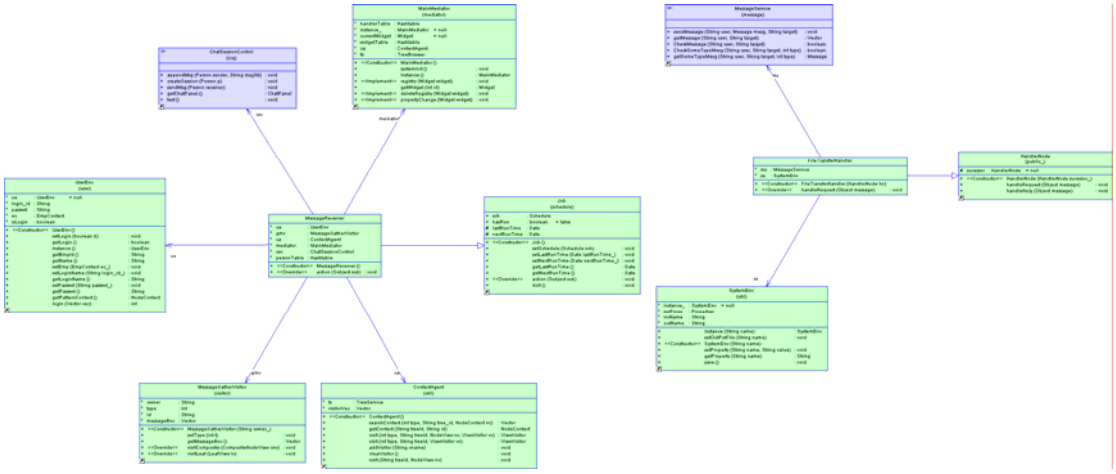

圖 68 eoffice.pa.job.messageHandler 類別圖

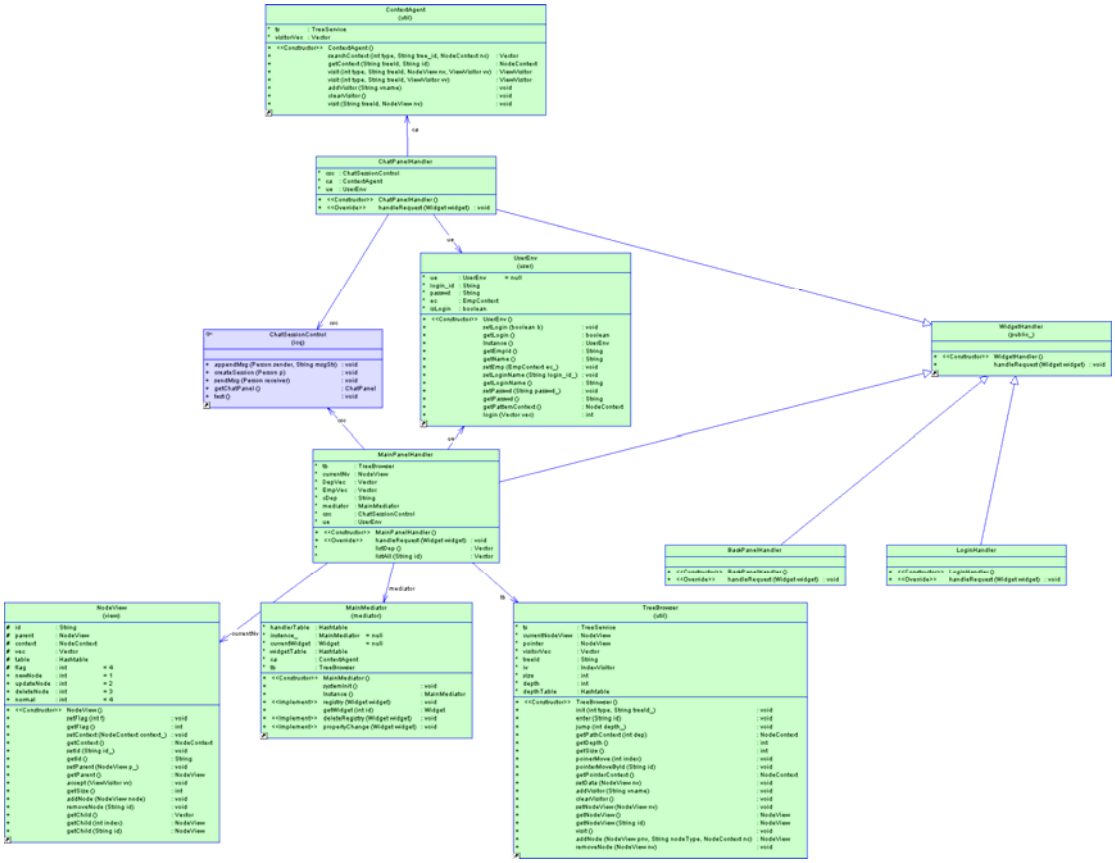

圖 69 eoffice.pa.mediator.handler 類別圖

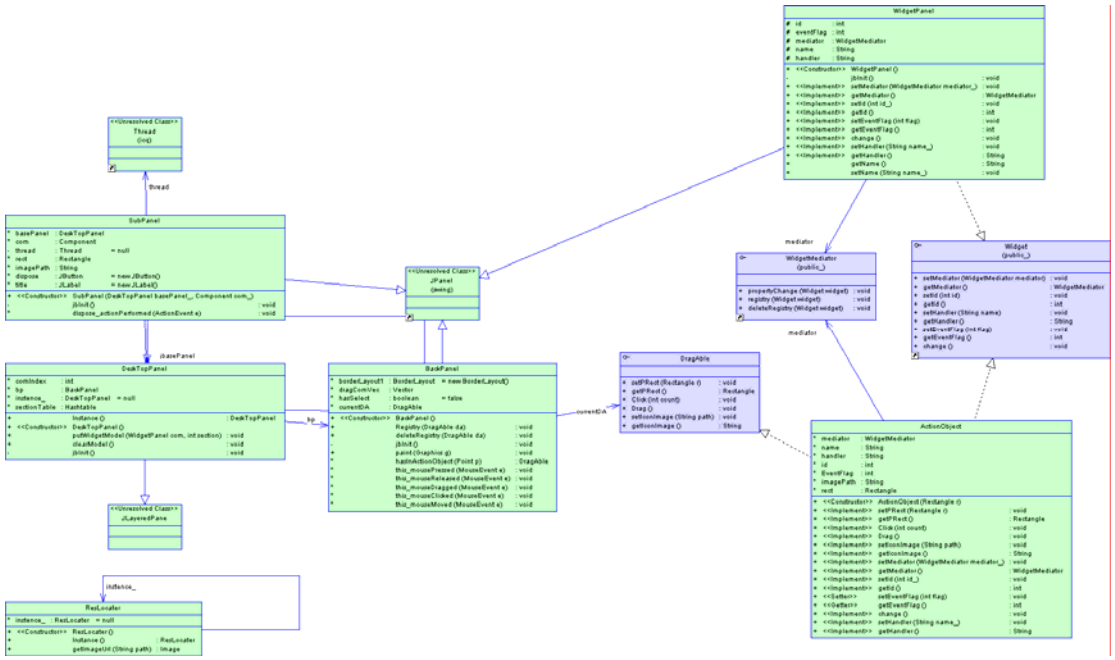

圖 70 eoffice.pa.ui.basic 類別圖

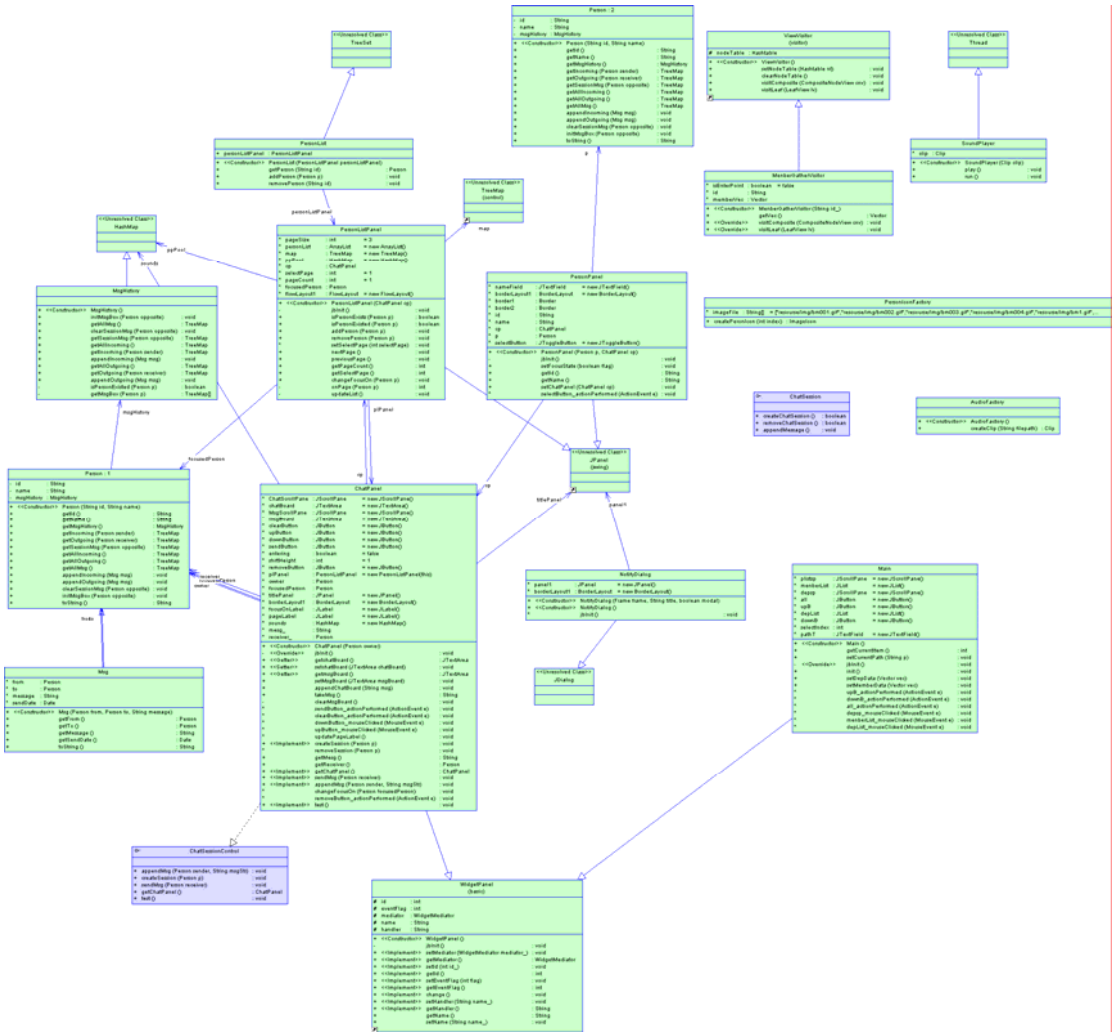

圖 71 eoffice.pa.ui.icq 類別圖

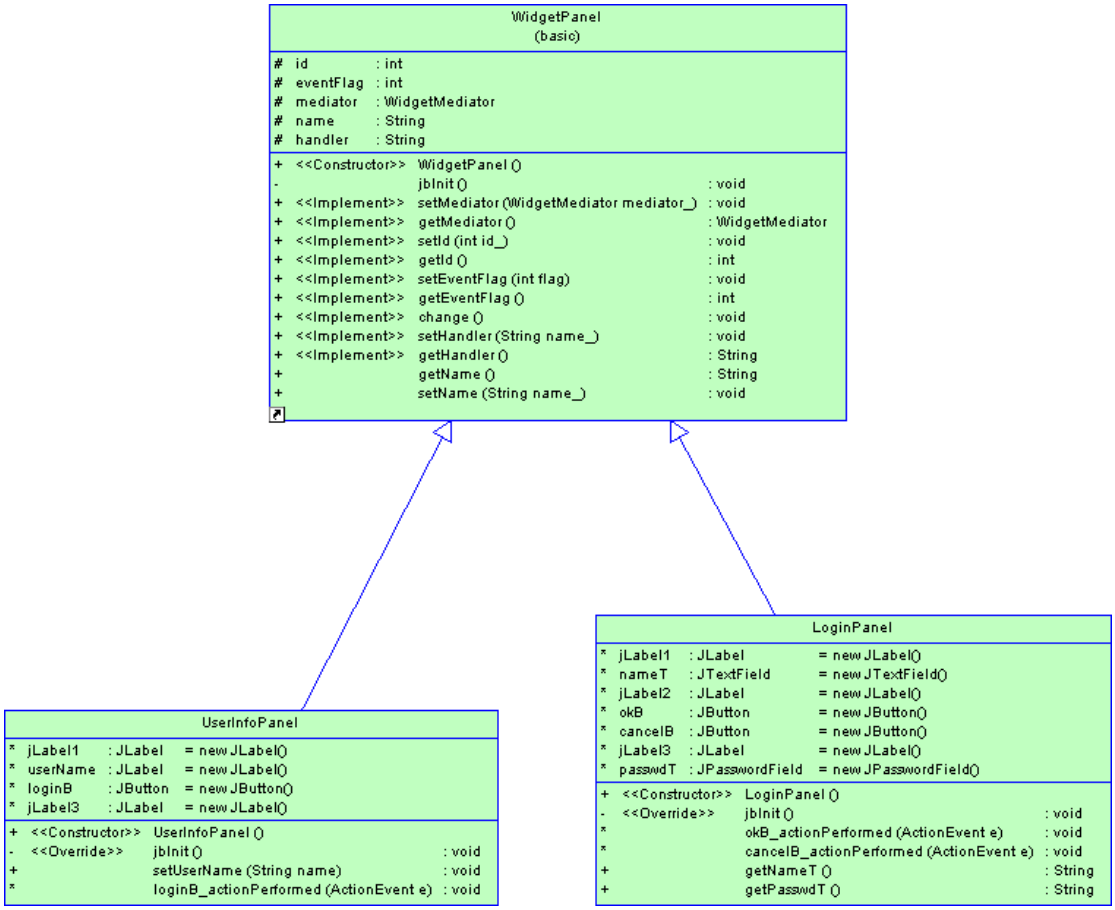

圖 72 eoffice.pa.ui.login 類別圖

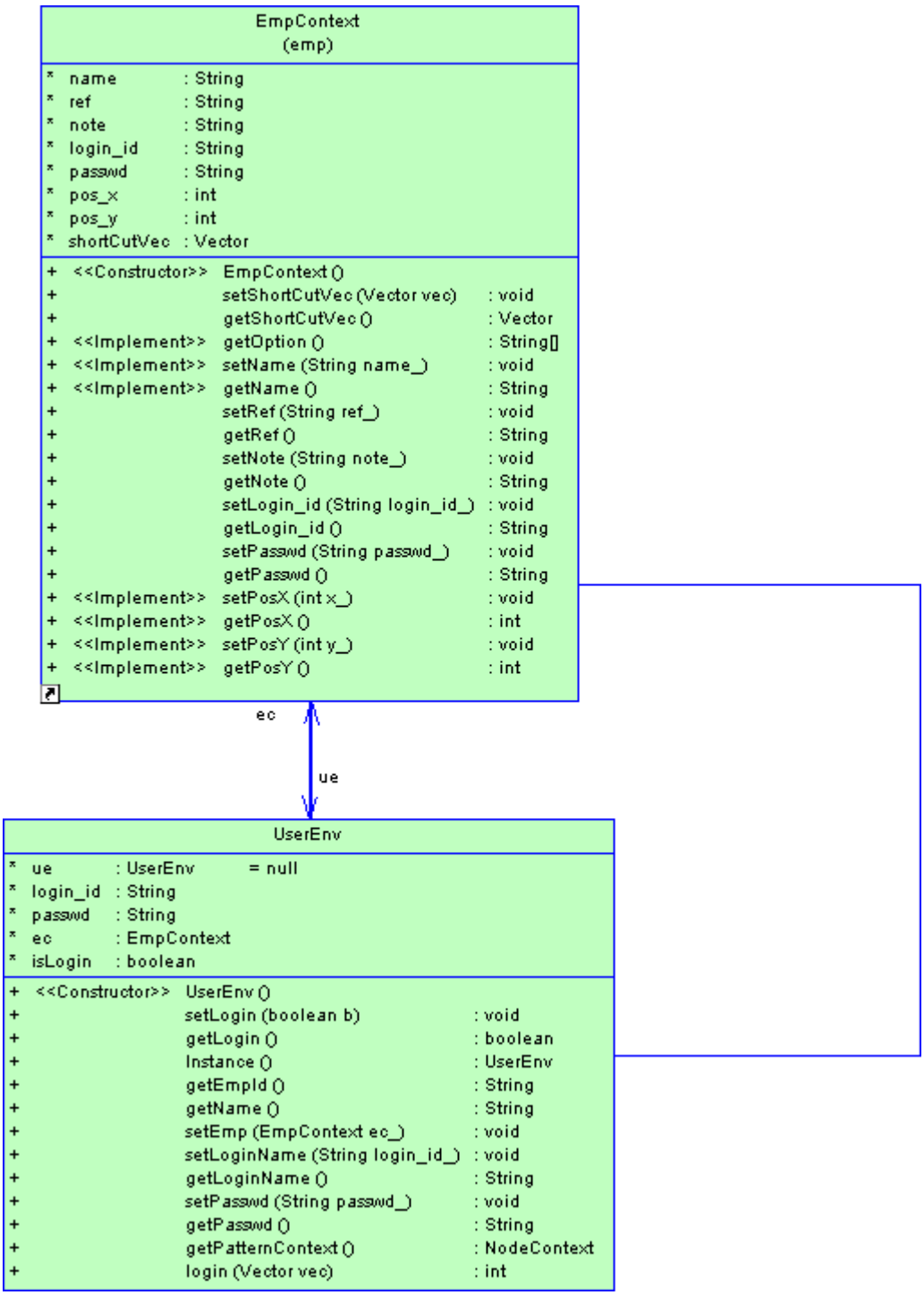

圖 73 eoffice.pa.user 類別圖

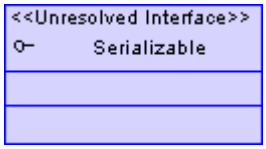

圖 74 java.io 類別圖

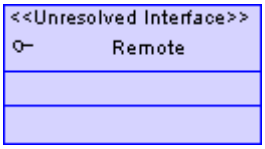

圖 75 java.rmi 類別圖

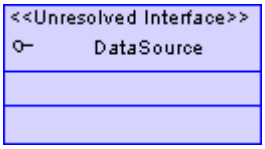

圖 76 javax.activation 類別圖

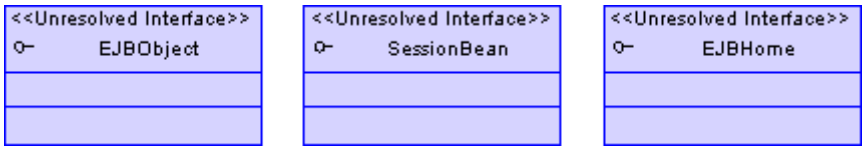

圖 77 javax.ejb 類別圖

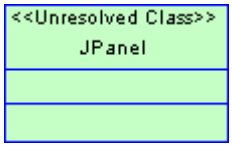

圖 78 javax.swing 類別圖

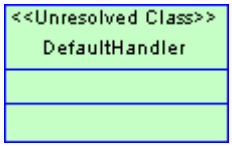

圖 79 org.xml.sax.heloers 類別圖

附錄F. 簽證系統物件

表 7 validate 元件使用物件

頁面: validate_login.jsp	描述: 檢查使用者使用資格
物件	
Am: com.zen.eoffice.account.model.AccountModel Sysam: com.zen.eoffice.account.model.AccountModel accountBean: com.zen.eoffice.account.adapter.AccountBean userBean: eoffice.bean.UserBean accountMc: com.zen.eoffice.control.ModelContainer sysaccountMc: com.zen.eoffice.control.ModelContainer em: com.zen.eoffice.personnel.model.EmployeeModel employeeBean: com.zen.eoffice.personnel.adapter.EmployeeBean employee_mc_temp: com.zen.eoffice.control.ModelContainer login_employeeModel: com.zen.eoffice.personnel.model.EmployeeModel	

表 8 sign 元件使用物件

頁面: Sign_in.jsp	描述: 簽證主頁面
物件	
無	

表 9 sign_in_process 元件使用物件

頁面: Sing_in_process.jsp	描述: 簽到、退處理
物件	
nameMap: com.zen.eoffice.util.NameMapUtil employeeMap: com.zen.eoffice.util.EmployeeMapUtil wTRecord_mc: com.zen.eoffice.control.ModelContainer temp:com.zen.eoffice.check_attendance.model.WTRecordModel login_employeeModel: com.zen.eoffice.personnel.model.EmployeeModel tmm: com.zen.eoffice.check_attendance.model.TimeMaxModel timeMaxBean: com.zen.eoffice.check_attendance.adapter.TimeMaxBean timeMax_mc: com.zen.eoffice.control.ModelContainer timeMaxtemp:com.zen.eoffice.check_attendance.model.TimeMaxModel ohm:com.zen.eoffice.company.model.OfficeHourModel officeHourBean: com.zen.eoffice.company.adapter.OfficeHourBean officetemp:com.zen.eoffice.company.model.OfficeHourModel sim: com.zen.eoffice.check_attendance.model.WTRecordModel workTimeRecordBean:com.zen.eoffice.check_attendance.adapter.WTRecordBean	

wtr:com.zen.eoffice.check_attendance.model.WTRecordModel workTimeRecord_mc:com.zen.eoffice.control.ModelContainer

表 10 Query_wtrecord_window 元件使用物件

頁面: Query_wtrecord_window.jsp	描述: 出勤紀錄
物件	
employeeMap: com.zen.eoffice.util.EmployeeMapUtil nameMap: com.zen.eoffice.util.NameMapUtil login_employeeModel : com.zen.eoffice.personnel.model.EmployeeModel	

表 11 Query_wtrecord_process 元件使用物件

頁面: Query_wtrecord_process.jsp	描述: 查詢出勤紀錄
物件	
employeeMap: com.zen.eoffice.util.EmployeeMapUtil login_employeeModel : com.zen.eoffice.personnel.model.EmployeeModel wtrm: com.zen.eoffice.check_attendance.model.WTRecordModel wtrecordBean: com.zen.eoffice.check_attendance.adapter.WTRecordBean wtrecord_mc: com.zen.eoffice.control.ModelContainer	

附錄G. Check-In WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Web Service Description Language -->
<!-- Web Service: CheckIn -->
<!-- Author: gaiba -->
<!-- Modified: 2004 年 1 月 12 日  下午  11:01:32 -->
<definitions name="CheckIn" targetNamespace="urn:CheckInInterface"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="urn:CheckInInterface"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/">

    <!-- Schema -->
    <types>
        <xsd:schema elementFormDefault="qualified"
targetNamespace="urn:CheckInInterface">

            <!-- SOAP definitions -->

            <xsd:element name="signIn" type="tns:signIn"/>

            <xsd:complexType name="signIn"/>
            <xsd:element name="signInResponse"
type="tns:signInResponse"/>
            <xsd:complexType name="signInResponse">
              <xsd:sequence>
                <xsd:element name="signInResponseResult"
type="xsd:boolean"/>
              </xsd:sequence>
            </xsd:complexType>
            <xsd:element name="signOut" type="tns:signOut"/>

            <xsd:complexType name="signOut"/>
            <xsd:element name="signOutResponse"
type="tns:signOutResponse"/>
            <xsd:complexType name="signOutResponse">
              <xsd:sequence>
                <xsd:element name="signOutResponseResult"
type="xsd:boolean"/>
              </xsd:sequence>
            </xsd:complexType>
        </xsd:schema>
    </types>

    <!-- Messages -->
    <message name="signInSoapIn">
```

```
    <part name="parameters" element="tns:signIn" />
</message>
<message name="signInSoapOut">
    <part name="signInResult" element="tns:signInResponse" />
</message>
<message name="signOutSoapIn">
    <part name="parameters" element="tns:signOut" />
</message>
<message name="signOutSoapOut">
    <part name="signOutResult" element="tns:signOutResponse" />
</message>
<message name="signInHttpGetIn">
</message>
<message name="signInHttpGetOut">
    <part name="signInResult" type="xsd:boolean" />
</message>
<message name="signOutHttpGetIn">
</message>
<message name="signOutHttpGetOut">
    <part name="signOutResult" type="xsd:boolean" />
</message>
<message name="signInHttpPostIn">
</message>
<message name="signInHttpPostOut">
    <part name="signInResult" type="xsd:boolean" />
</message>
<message name="signOutHttpPostIn">
</message>
<message name="signOutHttpPostOut">
    <part name="signOutResult" type="xsd:boolean" />
</message>

<!-- Port Types -->
<portType name="CheckInSoapPortType">
    <operation name="signIn">
        <input message="tns:signInSoapIn" />
        <output message="tns:signInSoapOut" />
    </operation>
    <operation name="signOut">
        <input message="tns:signOutSoapIn" />
        <output message="tns:signOutSoapOut" />
    </operation>
</portType>
<portType name="CheckInHttpGetPortType">
    <operation name="signIn">
        <input message="tns:signInHttpGetIn" />
        <output message="tns:signInHttpGetOut" />
    </operation>
    <operation name="signOut">
        <input message="tns:signOutHttpGetIn" />
```

```
            <output message="tns:signOutHttpGetOut" />
        </operation>
    </portType>
    <portType name="CheckInHttpPostPortType">
        <operation name="signIn">
            <input message="tns:signInHttpPostIn" />
            <output message="tns:signInHttpPostOut" />
        </operation>
        <operation name="signOut">
            <input message="tns:signOutHttpPostIn" />
            <output message="tns:signOutHttpPostOut" />
        </operation>
    </portType>

    <!-- Bindings -->
    <binding name="CheckInSoapBinding" type="tns:CheckInSoapPortType">
        <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http" />
        <operation name="signIn">
            <soap:operation soapAction="" style="document" />
            <input>
                <soap:body use="literal" namespace="urn:CheckInInterface"
                  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
            </input>
            <output>
                <soap:body use="literal" namespace="urn:CheckInInterface"
                  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
            </output>
        </operation>
        <operation name="signOut">
            <soap:operation soapAction="" style="document" />
            <input>
                <soap:body use="literal" namespace="urn:CheckInInterface"
                  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
            </input>
            <output>
                <soap:body use="literal" namespace="urn:CheckInInterface"
                  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
            </output>
        </operation>
    </binding>
    <binding name="CheckInHttpGetBinding"
type="tns:CheckInHttpGetPortType">
        <http:binding verb="GET" />
        <operation name="signIn">
            <http:operation location="signIn" />
            <input>
                <http:urlEncoded />
            </input>
            <output>
```

```
                <mime:content type="text/xml" />
            </output>
        </operation>
        <operation name="signOut">
            <http:operation location="signOut" />
            <input>
                <http:urlEncoded />
            </input>
            <output>
                <mime:content type="text/xml" />
            </output>
        </operation>
    </binding>
    <binding name="CheckInHttpPostBinding"
type="tns:CheckInHttpPostPortType">
        <http:binding verb="POST" />
        <operation name="signIn">
            <http:operation location="signIn" />
            <input>
                <mime:content type="application/x-www-form-urlencoded" />
            </input>
            <output>
                <mime:content type="text/xml" />
            </output>
        </operation>
        <operation name="signOut">
            <http:operation location="signOut" />
            <input>
                <mime:content type="application/x-www-form-urlencoded" />
            </input>
            <output>
                <mime:content type="text/xml" />
            </output>
        </operation>
    </binding>

    <!-- Service -->
    <service name="CheckInService">
        <port name="CheckInSoapPort" binding="tns:CheckInSoapBinding">
            <soap:address location="" />
        </port>
        <port name="CheckInHttpGetPort"
binding="tns:CheckInHttpGetBinding">
            <http:address location="" />
        </port>
        <port name="CheckInHttpPostPort"
binding="tns:CheckInHttpPostBinding">
            <http:address location="" />
        </port>
    </service>
```

</definitions>