南　華　大　學

資訊管理學系碩士論文

建立以 Web Services 為基礎的應用整合架構

Development of a Web-Services Based

Application Integration Framework

研 究 生：黃 書 慶

指導教授：王 昌 斌

中華民國　九 十 二 年　六 月

南　華　大　學

碩 士 學 位 論 文

資訊管理學系

建立以 Web Services 為基礎的應用整合架構

Development of a Web-Services Based Application Integration Framework.

研 究 生：黃書慶

經考試合格特此證明

口試委員：

指導教授：

所　　長：

口試日期：中華民國　92 年　6 月　19 日

誌　　謝

不知不覺，從大二甫插班進入南華這個美麗的校園至今，已經第五個年頭過去了，而進入研究所兩年來的研究生涯，就在論文完成之際畫上了個句號，這些日子以來，除了結識了許多真心相待來自全省各地的朋友齊聚一堂之外，所上各位教授們的苦口婆心，不管是在生活上的提點，以及對學問上的鼓勵；都深深的銘記在心。

首先，由衷感謝恩師王昌斌教授，不論在生活上或是研究資源上都給我相當程度的支持，讓我能夠在不受限制的環境中持續發掘自己的潛能，以及不斷提升面對困難時解決事情的能力；而高雄應用科技大學何正得老師對產業界的豐富經驗與資訊技術的深入了解，更是大大的給予本篇論文的相關概念的啟發，若沒有諸位教授的指導跟成大製造工程所的同學們彼此切磋，本篇論文將無法順利完成。

再者，十分感謝口試委員成大製造工程所的陳裕民老師跟所長吳光閔老師，由於老師們嚴謹的治學態度跟思緒，提供我諸多寶貴的意見與指正，讓我在論文跟做研究的過程中受益良多。

感謝所上的助理伊汝從我進入研究所到畢業，一直盡力給予非常多的協助跟鼓勵；還有實驗室的助理淑媛跟月琴，當我們遇到任何疑難雜症時，都能夠幫我們適時化解；還有我遠在台北的好友耿豪，謝謝你總能不辭勞苦的幫我找到我需要的資訊。此外，若非玉真平日費心幫忙打點生活上的瑣事，無法想像我的日子會變成什麼樣子。

兩年多的生活中，同學們互相扶持與鼓勵令我難忘，與在研究所中結識的諸多好友共同分享彼此的興趣喜好，都將成為我美好的回憶。

最後，更要感謝我的家人，尤其是我的父母，在我離鄉背井這麼多年，除了讓我經濟上沒有煩惱之外，他們默默的支持與鼓勵，讓我有更大的自信能夠往前邁進。

黃書慶　謹識
於　南華大學資管所
九十二年六月

建立以 Web Services 為基礎的應用整合架構

學生：黃書慶　　指導教授：王昌斌 博士

南華大學資訊管理學系碩士班

摘　要

隨著資訊技術的不斷發展，產業電子化的議題也伴隨著資訊技術的不斷推陳出新而有不同的面貌，同一產業的企業夥伴合作關係中，普遍存在著系統平台的異質性問題，在同一企業之內，也可能面臨不同應用服務供應商的新舊系統無法互相溝通，分享資訊的繁瑣問題。另一方面，隨著全球化發展，企業中商務運作往來已不受時間空間的限制，當業務之中的組織成員在進行共同的任務或交易流程時，組織間的距離將不再侷限於同一間企業或是同一棟大樓；因此存在著任務組織成員之間所提供的服務資訊與系統加以整合與協調的重要性。

近年，軟體業界推出 Web Services 技術標準，主要透過 XML 為基礎的標準協定諸如 XML、SOAP、WSDL 等將企業間的服務加以描述與包裝，本研究希望能夠以 Web Services 技術為基礎，透過統一塑模方法分析建立起一個應用整合的平台架構；期能整合企業間原先建構在新舊系統與異質性平台之中的企業應用服務元件，在平台中進行提供資訊整合與任務協調共享，提供不同的應用服務間能有一個溝通的管道，並產生一個整合後的使用介面，讓企業中的使用者能夠簡單的在單一介面操作之下達成數個不同系統的任務。

關鍵字：Web Services、XML、企業應用整合、中介軟體。

Development of a Web-Services Based Application Integration Framework

Student：Shu-Qing Huang　　Advisors：Dr. Chin-Bin Wang

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

ABSTRACT

The interesting of E-business is discussed widely in various dimensions with the development of information technologies. The heterogeneity among systems have persecuted businesses and their partner in several industries. These problems will also face the problem that systems between two application service providers can't communicate and share information to each other. On the other hand, globalization enable businesses can operation in 24/7 and no limitation in space, it has a significant implication that distance among businesses are unlimited. Therefore, the service information and system integrations among various businesses are important in digital age.

Recently, software industry established the web-services standard, it's packages business rules and processes by XML, SOAP and WSDL. Our research is expect to development a web-services based application integration framework by using UML, we hope to provide a new solution to the application of business services component integration among heterogeneous systems. In our framework, it's providers a channel of communication between two different systems, that they can to proceed to data integration and coordination of assignment; and generate interface dynamically, that User can accomplish an assignment among different systems in only one simple interface.

Keywords: Web Services、XML、EAI、middleware.

目錄

圖目錄

第一章、緒論

第一節　研究動機

隨著資訊技術的不斷發展，以及網際網路的加速擴張使得資訊的傳遞與交易更加快速，連帶的造成企業組織的不斷變革與調整，產業電子化的議題也伴隨著資訊技術的不斷推陳出新而有不同的面貌，在同一產業的企業當中，普遍存在著彼此間系統平台的異質性問題，即使專注在同一企業之內，也可能在適應資訊處理速度越來越快，資訊技術的不斷更新與變革；進而面臨不同應用服務供應商的新舊系統無法互相溝通，分享資訊的繁瑣問題。

而另一方面，隨著全球化發展的趨勢，企業營運版圖的逐漸擴大，在網際網路的發展下，企業中商務運作往來已不受時間空間的限制，當業務之中的組織成員在進行共同的任務或交易流程時，組織間的距離將不再侷限於同一間企業或是同一棟大樓；因此存在著任務組織成員之間所提供的服務資訊與系統加以整合與協調的重要性。

雖然軟體業界目前採用的技術諸如 DCOM、RMI、COBRA 等技術來支援此類分散式的服務進行溝通與協調，但是當遭遇到先前所述之系統平台異質性問題時，往往要額外投注人力進行特定轉換介面的開發和維護，造成效率的不彰與資源的浪費。

近年，軟體業界推出 Web Services 技術標準，主要透過 XML 為基礎的標準協定諸如 XML、SOAP、WSDL 等將企業間的服務加以描述與包裝，本研究希望能夠以 Web Services 技術為基礎，建立一個應用整合的平台架構；期能整合企業間原先建構在新舊系統與異質性平台之中的企業應用服務元件，在平台中進行提供資訊整合與任務協調共享，提供不

同的應用服務間能有一個溝通的管道，並產生一個整合後的使用介面，讓企業中的使用者能夠簡單的在單一介面操作之下達成數個不同系統的任務。

第二節　研究目的

本研究主要探討建立一個整合 Web Services 技術為基礎的應用元件仲介平台架構，首先我們利用 Web Services 技術跨平台以及訊息標準化的特性，期望找出企業日常業務操作系統的應用模組之間所包含的服務元件能在此平台中進行整合與溝通協調的機制，並進一步定義出能將整合與溝通的方式能夠加以定義儲存在本平台的格式，最後再設計一個能動態依照前述格式產生整合操作介面的模型，將此架構的完整運作達成企業應用整合的目的。

本研究目的可分為以下幾點：

1. 蒐集及整理相關文獻資料，探討 Web Service 技術的特性與仲介服務平台架構的結合。
2. 建構一可行的元件服務仲介平台架構。
3. 分析與設計平台系統的可行性。
4. 針對進行雛型測試以驗證架構可行性。

第三節　研究範圍與限制

本研究擬就以下範圍加以探討：

1. **Web Services** 技術發展部份，由於 Web Service 屬於目前新興的技術，尚有許多附屬的技術尚未通過成為技術標準，諸如關於服務品質陳述的 WSEL，服務流程概念的 WSFL 等，因此本研究僅

針對該技術的概念作為架構的考量，而並不實際引用該項技術，待業界標準發展成熟後再加以改良導入。

2. 分散式物件技術部分，本研究即在改進目前分散式物件技術的異質性系統問題，因此將探討分散式物件技術與 Web Services 技術之間的比較與改進，重心則是以 Web Services 為基礎所設計的溝通機制，最後研究結果則不再針對現有技術為基礎作效率比較。
3. 應用服務仲介平台部分，本論文主要在探討一個完整的應用整合架構中，匯整企業應用服務的平台，如以下圖中一個完整的整合架構中，本研究實作的系統部份屬於中介軟體層。

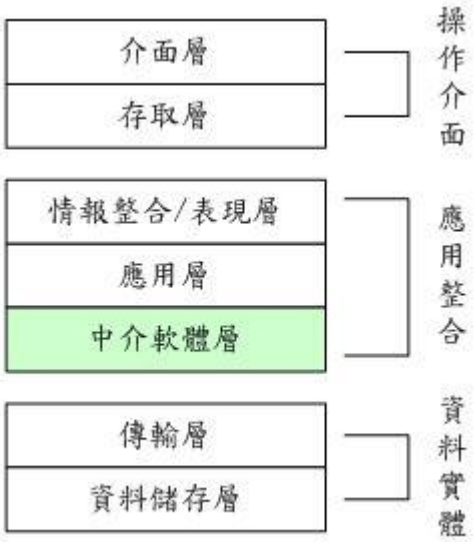


圖 1、應用服務整合架構技術層次圖[8]

4. 應用案例部分，本研究以追求彈性化的平台架構為主，但特定個別應用整合案例受限於內部系統的封閉或企業資源的缺乏而或有無法適用的情形，本研究並不針對特定案例進行分析設計。[8]

第四節　研究流程

本研究的研究流程可分為以下幾個部分：

1. **確定研究主題：**

   確定研究範圍、目的和主要架構等。

2. **資料收集：**

   搜集建構應用服務仲介平台系統所需要的相關資料與工具。

3. **核心技術探討：**

   搜集過去與本研究相關研究與文獻。本研究核心技術與文獻主要分為四類，分別為「企業應用整合」、「分散式物件技術」、「Web Services**」、「統一塑模語言**-UML**」。**

4. **資料分析：**

   針對相關文獻中所發表之 Web Services **訊息溝通模式以及如何建立** Web Services **仲介互動機制所需的功能性需求於架構中加以分析與整理。**

5. **架構建立與發展：**

   建立一個仲介 Web Service **與管理的平台架構，使得各類獨立開發應用模組能在其上將各別模組內的元件以** Web Services **方式包裝，在平台中對每個** Web Service **所擁有的屬性與方法加以登錄設定在平台上；供平台上定義的其他程序叫用。**

6. **系統分析與設計：**

   根據所建立之架構加以細部分析並選用適當的技術，以利程式的規劃與撰寫。

7. **系統雛型實作與測試：**

   根據所分析出來的架構，進行雛型的開發與測試。

8. **結論與建議：**

   提出本研究之結論與未來研究之方向。

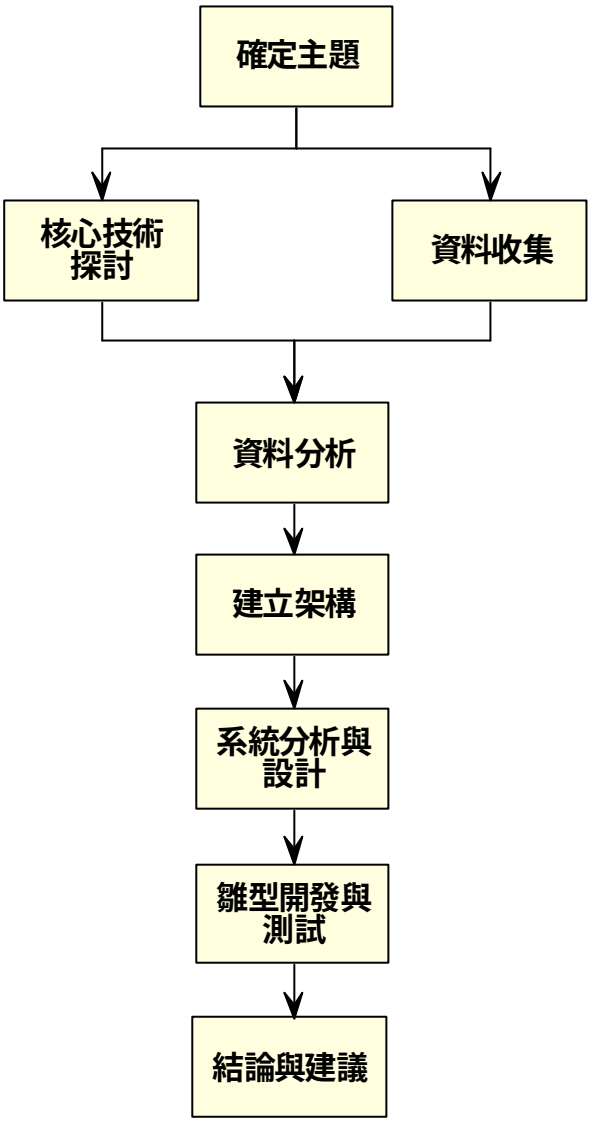


圖 2、研究流程圖

第二章、文獻與核心技術探討

第一節　企業應用整合

在企業資訊科技的應用之中，互不連結的應用系統、企業內部的資訊孤立、各個應用程式各自對應各自的資料庫有也具有不同營運目標，資料重複性與資訊處理程序分散不連貫，往往是阻礙企業資訊流通的絆石。

因此新技術的選擇與應用，必需考慮到跨平台的異質性、不同的資料來源與格式處理、安全控管與結合互不關連的網路環境、促使母公司與子公司跨越企業界線，進行資訊交流及隨時依需求新增功能的適應度…等，於是企業應用整合之觀念便應運而生，其功能便是統整不同類型之應用程式、確定其資料一致性、提供企業一個仲介平台將分散於各部門之有用資訊整合並連結，讓資訊在各系統間得以相互流通，以使商業流程籍由企業應用整合而更有效率的應用。

David S.Linthicum 認為企業應用整合是「企業體中任兩個以土的連接應用程式和資料庫無限制地共享彼此的資料和營運流程。』[11]Boris Lublinsky 對企業應用整合所下的定義為：企業應用整合是一種策略、技巧或是作用過程針對整合分隔的資訊和建立企業營運流程[21]。另外，OVUM 顧問公司對此的定義為：企業應用整合乃是結合技術與流程的策略性整合方案，能將客製化、套裝軟體與 web 的功能做一有效結合，以企業間熟悉的文件及檔案格式進行商業資訊的交換。』[1]其所使用的整合方法乃以標準化中介軟體(middleware)架構與分散式物件技術將不同應用程式做安全且有效率的整合。[9]

企業應用整合的價值在於有效協助企業達到風險、軟體開發和維護

成本的降低，並能進一步整合各供應商閥的合作關係。以下簡單敘述企業應用整合對企業的效益[22]：

(1)降低企業既有應用程式之維護成本：

企業系統之維護成本原先必須每一系統分別維護，而採用企業應用整合後，使企業僅針對整合之應用程式系統加以維護，有效降低現有應用程式系統的維護成本。

(2)降低企業既有應用程式之整合成本：

企業運用企業應用整合能將新的技術快速且容易的整合在原先的應用系統架構下，在資料分析和應用程式整合過程當中，所花費的時間和成本有顯著的降低，使企業不同部門間應用程式整合更為容易且更具成本效益。

(3)增加企業既有應用程式之整合彈性：

企業應用整合的解決方式提供彈性的整合方法，以支援分散時進行不同系統間的系統與資訊整合交換。

(4)提高企業間跨組織之溝通效率：

企業間合作關係日趨緊密，因此企業導入企業應用整合能有效提升企業問供應鏈系統跟資訊傳遞的即將性與協調性。

總而言之，企業導入 EAI 概念後，針對既有應用程式而言，企業可進行整合，不僅統一資料格式及型態，對於企業營運流程以加以整合，使高階管理者更容易掌握企業現況，針對開發新的應用程式而言，其開發成本將會大量降低，且對於新系統的開發應用將會更加容易、快速，進而縮短開發的時間，使企業更容易增加自身的競爭優勢，提高企業競爭力。

第二節　分散式物件技術

電腦軟體的發展伴隨著網路及作業系統的不斷進度，目前有愈來愈多的軟體或系統是採用了主從式(Client/Server)的架構進行設計，主從式架構基本上是由 Server 端及 Client 端兩部分的程式所組成。Server 端程式負責在網路環境中提供服務，而 Client 端的程式則是負責向 Server 請求服務，使用 Client/Server 架構可以將原本的集中式系統分割成 Server 及 Client，使許多 Client 的應用程式可以由遠端連接到一個 Server，並同時取得服務。然而，為了處理大量且快速的資料流量及 Client/Server 之間的交易，Client 和 Server 兩端不可避免地變得愈來愈複雜且龐大，這將意味著愈來愈難去建構、維護和擴充原來的 Client/Server 系統。[17]

解決上述問題的趨勢之一就是採用分散式物件技術，其包含了一個中介層 (Middleware)位於 Client 與 Server 兩端之間，專司建立與處理 Client 端與 Server 端物件之間的關係，在分散式物件技術裡，將資料及內部邏輯都被封裝在物件中，並允許物件可以位於分散式環境中的任一處，因為這個特性，在建立彈性化的 Client/Server 系統上顯得很容易。且藉著分散式物件技術，我們可以將單一的 Client/Server 系統再分割成許多聰明的、易於管理的元件，使這些元件可以跨網路及平台而以適當的方式共同合作。其中元件式軟體發展技術(Component-based Development)是近年來軟體產業最重要的革命技術，元件化技術主要應用於分散式應用程式，分散應用程式主要由分散物件模組管理，例如 Microsoft 的 Distributed Component Object Model(DCOM)、共通物件存取仲介架構(Common Object Request Broker Architecture; CORBA)。或是 SUN 的 Remote Method Invocation(RMI)等。

然而這些元件技術在網際網路發展的過程中遭遇瓶頸，許多企業系統過去一直都是採用此類模組開發，然而，這種模組並非長久之計。當企業為了保持競爭優勢而進行組織的改革、合併或是隨著資訊技術的提升而系統有所更動時，企業就很難保證能繼續維持單一完整的架構，因為企業所陸續採用的新舊系統、物件模組或程式語言都可能不盡相同。相反地，本研究所提及的 Web Services 技術，其在網頁上分散及整合應用程式的架構則顯得更有彈性，提供服務描述(Service Description)及服務搜尋(Service Discovery)等功能;最重要的是，透過使用 SOAP 及 XML 技術以達到跨作業平台、跨程式語言及跨任何接取設備的境界，讓企業應用系統的發展能更加完整，並可運用有效的資源管理來滿足客戶的需求。

一般來說，分散式物件技術所帶來的好處主要有以下幾點[15]：

- 隨插即用(Plug and Play)：所有軟體元件有如積木般，藉由定義完整的介面，可以輕易的裝上或拆下，並且裝上即可使用，不必再經過編譯(Compile)的過程，根據這種特性，即使再複雜、龐大的系統也可以是由許多元件組裝而成，組裝軟體成為軟體發展的趨勢。
- 訊息互通(Interoperability)：經過完整介面定義的元件可以藉由 ORB (Object Request Broker) 互相傳遞訊息共同合作，且傳遞的訊息不僅是資料還包含處理資料的方法，故當一個物件插入分散式系統後，它就可以提供服務給其他物件，並且也可以在需要時向其他元件請求服務。
- 可移植性(Portability)：在分散式系統中，所有的物件以介面(Interface)來定義其所提供的服務，而不需要考慮物件的位置或物件以何種方式實現(Implement)，因此在分散式系統中元件可以在不同的作業平台上工作，而具有良好的可攜性。

- **並存**(Coexistence)：**傳統非物件導向式可以經過包裝，而成為新的元件，舊有的程式可以再被使用。**

分散式物件技術成為新一代的主從式架構程式的發展依據，於是各種分散式物件標準紛紛出現以下將分別針對 Java **中的** RMI **機制、**OMG (Object Management Group)**所制定的** CORBA **規格，及微軟的** DCOM **此三種分散式物件標準稍作說明。**

壹、RMI [14]

以往當我們談到 RPC **的時候，是以函式為單位，這主要是配合過去程序式語言（**procedural programming language**）的作法。到了現在，物件導向語言充斥，所以這** RPC **的對象就變成了「物件」，叫用**(invoking)**的就不再是函式、而是物件的方法**(method)**了。所以** Java **遠端物件的叫用才會稱作** Remote Method Invocation**。**RMI **為** JAVA **中所提供的遠端程序呼叫機制，使** Client **與** Server **可分散於不同的電腦，在** RMI **中** Client **與** Server **之間的溝通。**

要瞭解 RMI **的架構，我們先從一般架構看起。**圖 3 **就是** RMI **一般的簡略架構。這個架構其實非常簡單，當使用者要叫用一個伺服端的函式時，真正叫用的是由伺服端的** skeleton **程式所提供的假函式，這個函式會複雜兩個重要的工作，第一個是找到所要叫用函式的所在位置，第二個就是幫您將叫用該函式所需要的參數作轉換，以便透過網路傳輸。等到到達伺服端時，伺服端這邊的** skeleton **程式會負責相反的動作，將參數拆解開來，然後叫用真正的** RPC **函式。函式執行完後，再透過伺服端的** skeleton **程式以相反的順序將傳回值丟回去給客戶端 因此當客戶端的**

stub 程式接收到後，再傳回給使用者的程式。透過了中間這兩個 stub / skeleton 程式的運作，使用者的程式就可以完全不管中間這層網路傳輸的問題，而當成是在叫用本地端的一個函式。換成遠端物件，而所謂的 skeleton 只不過是 RPC 架構中伺服端 stub 程式的新名字而已。這樣的架構表示現在伺服端提供出來的不再是以函式為單位，而是物件（事實上，如果要講的嚴謹一點，應該說是物件的一組方法）。

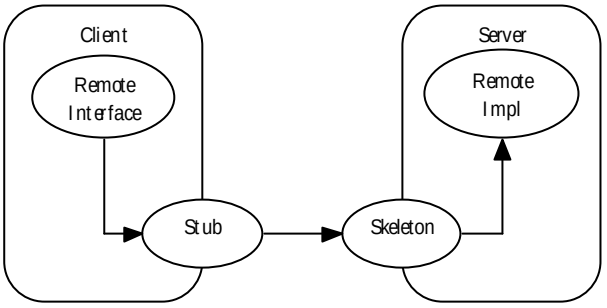


圖 3、Client & Server Communicate through stubs and skeletons [16]

貳、CORBA [10]

CORBA 乃是由 OMG (Object Management Group)所定義，提供 Client 與 Server 間溝通的機制，CORBA 規格對一個物件請求仲介者 (ORB; Object Request Broker) 之標準介面作了詳細的規定。ORB 單純從字面上看來是一種可以連接各種物件的架構。再說清楚一點就是這種架構可以讓分散在不同機器上的物件互相做溝通，概念上跟 RPC 有點像，不過 CORBA 傳送的是物件，簡單的說就是在 A 端的程式可以使用放在 B 端的物件。根據 OMG 的定義，一個物件請求仲介者(Object Request Broker)為可提供分散式環境上各個物件透明化 (transparent)的請求服務與回應接收功能的應用程式建構工具。透過 CORBA 規格中所定義的 ORB 及 Object Services，使遠端的 Client 可以很容易地與 Server 溝通。

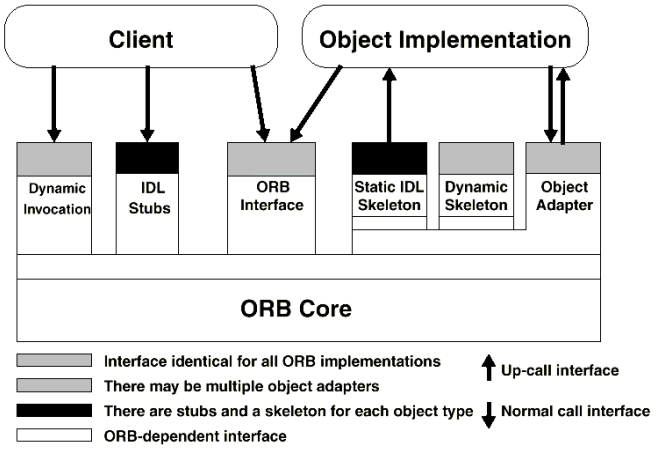


圖 4、CORBA 架構(CORBA 2.0/IIOP Specification) [29]

- Dynamic Invocation **的功能是用在動態建立物件和對物件行使要求。**
- Dynamic Skeleton Interface (DSI) **的功能是** Server **端用來接收** Client **端的要求，這種要求在** compile **的時候型態還不能確定，所以需要** DSI **來做處理。**
- ORB Interface **的主要功能是用來讓** Client & Server **可以互相傳送** Object Reference。
- Object Adapter(OA) **的主要功能有：**
  1. Generation and interpretation of object references
  2. Method invocation
  3. Security of interactions
  4. Object and implementation activation and deactivation
  5. Mapping object references to the corresponding object implementations
  6. registration of implementations

參、DCOM [13]

DCOM 是 Distributed Component Object Model 的縮寫，是由 Microsoft 所提出之通訊協定，DCOM 係建構 COM 技術架構之上，它是微軟將元件的應用層次提升至網路的重要技術規範。從許多方面來看，DCOM 與 DEC RPC 的關係相當密切。DCOM 讓某一應用程式能夠引發遠端的物件。此外，DCOM 大量借用 DEC RPC 的 Object Remote Procedure Call (OPRC)協定來遙控物件方法。簡言之，DCOM 的技術核心是由 OPRC 和 COM 所構成，如欲應用 DCOM，無論是 Server 端或 Client 端的平台皆須支援 COM。[17]使 Client 與 Server 可以在網路上以可靠的(Reliable)、安全的(Secure)且有效的(Efficient)方式直接溝通。如圖 5 所示：

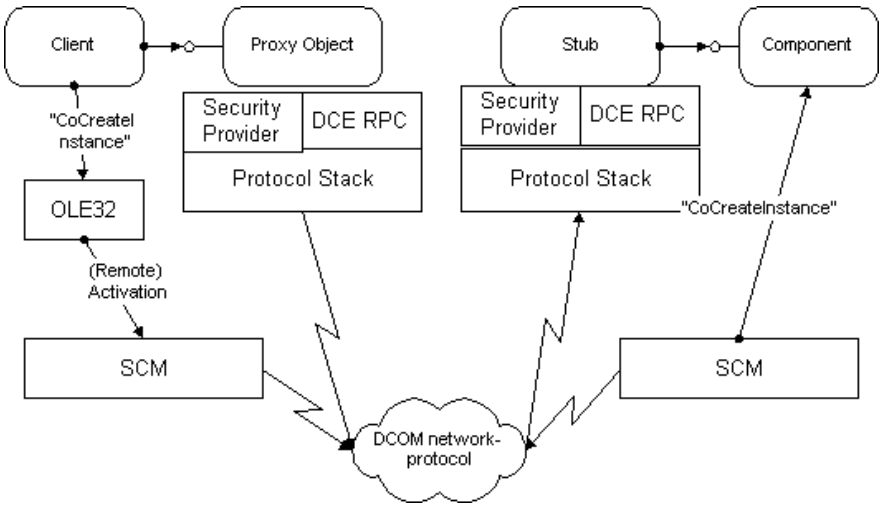


圖 5、DCOM [33]

肆、小結

上述這些分散式物件技術最大問題是它們已經不能延伸至網際網路，在企業內部網路(Intranet)上使用這些分散式物件傳輸協定有很好的效果。但在公眾的網際網路上使用這些協定就有很多問題：

- **無法穿越防火牆[2]**

  要了解為何防火牆會造成分散式物件通訊協定的問題，必須先了解到防火牆是如何分辨協定之問的不同。在 TCP/IP 的架構下，每一個被廣泛使用的協定都被賦予一個特殊的埠號(Port number)而每一個使用該協定的需求封包都帶著這個埠號。例如 HTTP 協定的埠號是 80、FTP 是 21 等等。大部分的防火牆可以用來防止某個特殊協定的方式就是針對埠號拒絕某種協定的通訊。通常防火牆是被設定成允許埠號 80 的運作(如果該公司不拒絕使用 HTTP 的話)。

  大部分的防火牆會擋住其他的埠，因為它們假定利用其他的埠對公司內部網路的運作都是有危險的。但這也正是造成分散式物件通訊協定無法運行的原因。不像 HTTP / FTP 等其他著名的通訊協定，分散式物件通訊協定通常沒有使用一個著名的大家都知道的埠號來溝通。相反地，這些通訊協定通常動態地被賦予埠號，埠號碼在被需求時任意產生。如果沒有防火牆擋在使用者端與伺服端之間，這種方式將可以很有效地運作。但若加了防火牆，則該通訊協定會因為防火牆不允許兩端任意使用任何埠號來溝通而中斷。

  當下存在很多種解決方式，例如某些防火牆可以被設定成允許某個範圍的埠號碼可以進行溝通，若該分散式物件通訊協定也可以被設定成只用這個範圍的埠號碼，則這個方案便可行，使用者端與伺服端

之間可以進行溝通。但比較注重安全的網路管理者將不會贊成開放任意一組埠號碼而導致這個方案並不完美。另一個選擇是採用 COM 網際網路服務，這讓傳統的 DCOM 封包在 TCP 上透過埠號 80 來傳遞。這在某些方面很有用，但這項技術只有微軟的 Internet Information Server 和 DCOM 在使用，而不是一項完整的解決方案，所以我們需要一個更普遍一般性的解決方案。

- **需要緊密連結**

  它們需要緊密的與服務的顧客連結，服務本身意味相互連結的架構。這往往也代表著這樣的連結是非常脆弱的，假使當連結的一方突然改變時，另外一則會中斷。例如，當主機的應用程式介面改變時，使用者端就會被迫中斷連線了。

  它們相當的不符合實際的功能，也就是說當一邊已經更改內容的時候，一邊卻不知道。例如，當服務應用程式的介面改變時，使用者端就會中斷了。需要緊密相連的架構並沒有什麼本質上的錯誤，許多應用程式一直都是採用此模組開發。然而，這種模組並非長久之計。當公司之間相互購併或當提供資訊服務的廠商結束營業的時候，公司就很難保證能維持單一完整的架構。也很難保證說你要的服務，在遠方的另一端能有你需要的完整架構：因為你不知道它們採用什麼作業系統、物件模組或程式語言。

- **應用程式整合困難[4]**

  無論目前的應用程式、作業系統、網路架構如何，在企業內部透過公司政策可以強制要求或是調整使用相同的平台、系統及規格，但是到 Internet 的環境，是個開放自由的環境，無法要求也無法預期交易夥伴或客戶與企業內部使用相同的資訊產品及技術。所以資訊要流

通必然會遇到企業內部整合、供應鏈整合及異質系統問的問題，在應用程式溝通時還會遇到的問題，例如：資料傳輸方式、呼叫規則及資料格式。

■ 傳輸方式：

在各種異質的環境中，各自使用不同的方式傳輸資料，像是 RPC(Remote Procedure Call)，或是不同的通訊協定：HTTP、SMTP、FTP，或是以訊息的方式傳送：MSMQ、MQ Series、IIOP，要研究這麼多的底層傳輸協定及技術，會消耗掉許多專案開發人員寶貴時問及精神，也不一定能順利整合。

■ 呼叫規格：

透過目前的分散式物件技術像是 DCOM、CORBA，只要應用程式提供完整的函式呼叫說明，描述要傳入那些參數、資料型別，應用程式便可以呼叫外部的應用程式，因為作業系統中的分散式的運算環境，已經幫我們處理掉底層的呼叫連結及參數傳遞。但事實上其中隱藏了許多假設性的預設情況，像是呼叫雙方要執行在相同的安全權限控管的環境中。同時也要參考相關的呼叫方式 (calling convention)、以相同的方式表示資料。同時所有的軟體元件也要支援使用相同的定址及啟動方式，才能以一致的步驟找到要呼叫的軟體元件並將它啟動。

■ 各種資料格式的轉換：

不同軟體公司所開發出來的應用程式，就必須考慮資料格式轉換的問題。XML 及 XSL 可彈性簡化資料轉換的工作。有了這些 XML 規格，軟體開發廠商及企業的 IT 人員就可以透過 XSL，進行資料格式的轉換。[25]

以上這些分散式物件技術所做的假設條件，讓分散式運算的理想在真正跨平台的環境中似乎有點難以實現。在整合的過程中，為了要讓這些應用程式協同運作，負責系統整合小組的成員便要自行解決這之間的差異性。為了使上述困擾減到最低，最近新一代的分散式技術網路服務(Web Services)因而出現。

第三節　Web Services 技術

從上一節的問題來看，Web Services 正是發展用來解決上述問題的，Web Services 將交換與溝通的層次拉高到程式邏輯，把 Internet 當成一個大的執行平台，大家都可以在平台上提供或使用服務。如圖 6 所示，Web Services 主要有下列成員[32]：

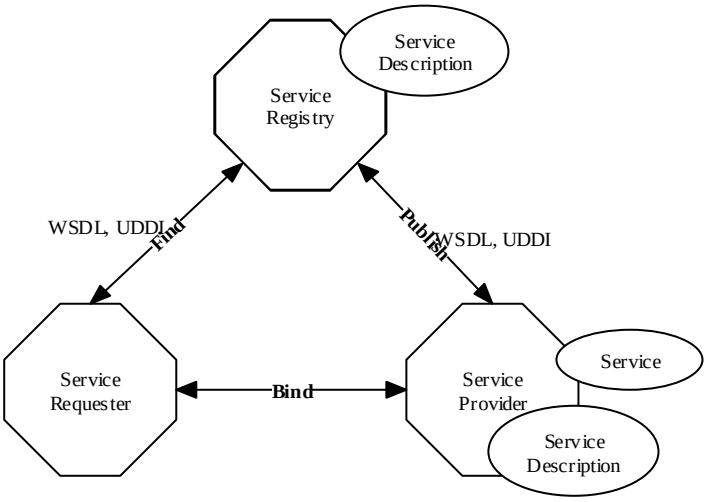


圖 6、Web Services 操作角色與行為

1. Web Services：Service 是一種應用程式，提供者將它公佈於 Internet 上提供服務。
2. 服務提供者(Service Provider)：負責供應商業處理功能讓別人來使用。藉由使用 WSDL，服務提供者可描述在這項服務中所含的功

能、要使用此功能(function)所需輸入的資料(input)以及輸出的結果(Output)，並提供 URL 使服務要求者可呼叫這項服務。最後，服務提供者透過 UDDI 把這些資訊公告在 UDDI 伺服器。

3. 服務要求者(Service Requester)：透過 SOAP 訊息處理，服務要求者考已送出格式化的查詢字串給服務登入資料庫，當查詢結果傳回後，服務要求者可以準備要求所需輸入的資料，並透過傳回的 URL 位址啟動該服務。重要的是，在 SOAP 架構中，雙方的溝通都是透過 SOAP 訊息機制。
4. 服務登錄資料庫(Service Registry)：此資料庫的功能就是 UDDI，一種儲存 Web Services 資訊的環境，讓服務提供者能公告其服務，而服務要求者能找到這些服務並取得和 Web services 溝通的相關資訊。

Web Services 主要的運作行為有：[42][43][44]

1. 描述(Description)：要讓 Web Service Requester 知道 Web Service 提供服務的內容，以及和其溝通的方式，需要有一種描述 Web Service 的語言。
2. 公佈(Publish)：透過註冊機制將 Web Service 的描述資訊登錄於公開的 Web Services Register。
3. 找尋(Find)：Service Requester 向 Web Service Registry 送出 Query 訊息，取得 Web Service 的相關資訊。
4. 繫結(Bind)：根據取得之 Web Service 描述資訊，建構一個連結到該 Web Service 的代理程式，透過 Internet 傳遞適當訊息參數，呼叫該 Web Service。

要使應用程式能跨越網域及不同異質平台架構的關鍵，是簡易資料描述格式，這個格式就是 XML。然而要讓 Web Services 在開放的 Internet 上實際動態地發佈、搜尋並執行服務，勢必得依存某些技術標準：基本線上傳輸格式(basic wire format)、服務描述(service description)、及服務搜尋(service discovery)。如圖 7 所示：

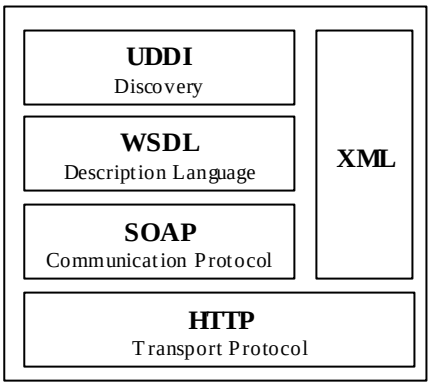


圖 7、Web Services 組成元素

- SOAP(Simple Object Access Protocol)：傳輸架構的最底層，系統需要藉由共同的語言來溝通。特別是，應用程式問的溝通需要有共通的規則，以做為不同資料型態之間的表示方法(例如，數字或陣列)，以及藉由共同的指令來溝通(處理資料的方式)。必要的話，應用程式也需要延伸本身語言。而架構在 XML 之上的 SOAP 即代表了一套資料與指令如何呈現與延伸的共用規則。
- WSDL(Web Services Description Language)：一旦應用程式有了共同的語法以呈現資料型態與命令，他們需要有一個共同的方式來描述資料與命令。但對於應用程式而言，只認得數字這是不夠的，甚至於應該說，如果有兩個數字的話，它還要能夠會運算。而 WSDL 以 XMIL 語法讓開發人員及開發工具得以展現網站服務

的能力。

- UDDI(Universal Description Discovery Integration)：最後一項需要設定規範的是如何確定服務描述的所在位置，UDDI 提供一組規範，能讓個人或程式開發工具可以自動的發現服務的 WSDL 描述。

一旦這三層架構完成，開發人員可以很輕易地找到網站服務，並視其為一物件，將之整合人他們的應用程式中，同時建立完善的基礎架構以便將應用程式執行結果傳送至網站服務上。在分散式的架構下，使用 XML 的環境中，SOAP 提供兩個電腦系統之間交換的架構與資料型別。在過去五年來透過網際網路存取已經變成是較進步社會的基礎需求。在其上執行著各式各樣的通訊協定。但是直至目前為止最廣泛被接受的通訊協定依然是 HTTP(Hypertext Transfer Protocol)，它於瀏覽器與 Web 伺服器之間溝通時被使用，對於文字、圖形以及其他資訊的傳輸具有很好的效率與彈性，而且它簡單易懂。接下來，我們針對 SOAP、WSDL 及 UDDI 這三種技術一一作介紹。

壹、SOAP[26]

SOAP(Simple Object Access Protocol)，它是一種語言中立、架構簡單的 Light-Weight 資料傳輸協定，用於分散式網路環境下作資料訊息交換。

SOAP 包含三個部分：

- SOAP 信封：定義了表達訊息中包含什麼的基礎結構;誰應該處理這個訊息;無論訊息是選擇性或強制性。
- SOAP 程式碼撰寫規則：定義了一個序列化的機制，這個機制用

於交換應用程式定義資料型別的執行個體。

- SOAP RPC：定義用於描述遠瑞程序呼叫和回應的協定。

除了 SOAP 信封、SOAP 程式碼撰寫規則和 SOAP RPC 協定以外，SOAP 也定義了兩種通訊協定的連結方式，其中包含描述 SOAP 訊息如何在 HTTP 訊息中傳送。

HTTP+XML 的 S0AP 訊息，首先要包裝 HTTP 的 Header，接下來的部分就是 SOAP 本身傳送的訊息內容，稱為 SOAP 信封。SOAP 信封是一份標準的 XML 文件，分為 SOAP Header 及 SOAP 中 Body 兩部份。Header 一般會定義一些 SOAP 內文、SOAP 資料型態、SOAP 編碼等之名稱空間 (NameSpace) 位址。Body 部分就是傳送 ClientRequest 與 ServerResponse 的訊息內容。

SOAP 訊息結構如圖 8 所示：

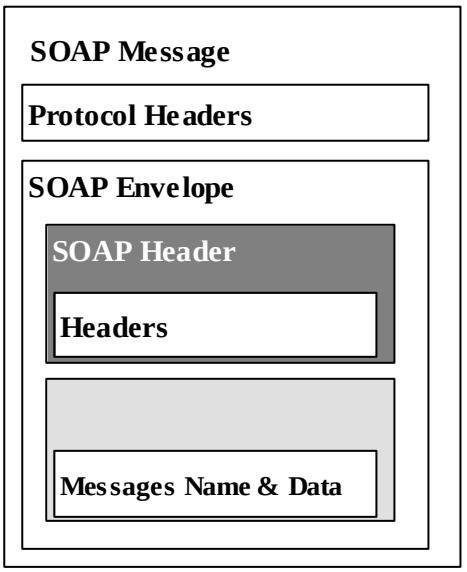


圖 8、SOAP 訊息

● 嵌在 HTTP 請求中的 SOAP 訊息[18]

```
POST /HelloWorld HTTP/1.1
Host: localhost
Content-type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://im.nhu.edu.tw/HelloWorld/HelloWorld"

<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelop xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
              xmlns:xsd=http://www.w3.org/2001/XMLSchema
              xmlns: soap=http://schemas.xmlsoap.org/soap/envelope/ >

<soap:Body>
    <m:HelloWorld xmlns=http://im.nhu.edu.tw/HelloWorld/" />
        <msg>HelloWorld!</msg>
    </m:HelloWorld>
</soap:Body>
</soap: Envelop>
```

● 嵌在 HTTP 回應中的 SOAP 訊息[6]

```
HTTP/I.I 200 OK
Content-Type：text/xml; charset="utf-8"
Content-Length：length

<? xml version="1.0" encoding="utf-8" ?>
<soap:Envelop xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              xmlns:soap="http://schemas.xmlsoap.org/soap/envelop/" >

    <soap:Body>
        <HelloWorldResponse xmlns="http://im.nhu.edu.tw/HelloWorld/ ">
            <HelloWorldResult>Hello World!</HelloWorldResult>
        </HelloWorldResponse>
    </soap:Body>
</soap:Envelop>
```

SOAP 的原理基本上是從傳送者至接收者的單方向傳輸，但是如同上述範例所示，SOAP 訊息交換通常會和商業上的夥伴以請求、回應的方式建置。接收 SOAP 訊息的 SOAP 應用程式必須依次執行下列的動作：

1. 識別供該應用程式處理的 SOAP 訊息每一個部份。
2. 確認該應用程式支援步驟 1 中識別的每一個部份，某些不會影響輸出的選擇性部分會被忽略。
3. 如果 SOAP 應用程式不是該訊息的最終目的，那麼在轉傳訊息之前先移除步驟 1 中所識別的部份。

SOAP 是利用 XML[30][31]的彈性和延伸性來封裝和交換 RPC 呼叫。使用 HTTP 為通訊協定連結的情況下，RPC 呼叫會自然對應 HTTP 請求，並且 RPC 回應會自然對應 HTTP 回應。然而，使用 SOAP 執行 RPC 並不只限於和 HTTP 通訊協定連結，例如 SOAP 訊息亦可透過 SMTP 傳輸。

SOAP 訊息需要下列資訊以使用一個方法呼叫：

- 目標物件的 URI
- 方法的名稱
- 選擇性的方法標記
- 方法的參數
- 選擇性的標頭資料

RPC 方法呼叫和回應都使用下列方式於 SOAPBody 元素中傳送

- 方法呼叫模式化為一結構。

- **方法呼叫視為單一結構，該結構中每一個** in **或** out **參數都包含存取器，該結構的命名和型別都和方法名稱一致。**
- **每一個** in/out **參數視為一個存取器，具有對應於參數名稱的名稱和對應於參數型別的型別。**
- **方法回應是模組化為一結構。**
- **方法回應視為單一結構，該結構中包含傳回值和每一個** In/out **參數存取器，第一個存取器就是參數後面傳回的值，次序和方法標記中的次序一樣。**
- **每一個參數存取器都有一個相對於參數名稱的名稱和相對於參數型別的型別，傳回值的存取器名稱並不明顯，同樣的，結構名稱也不明顯，在方法名稱後面會附加字串** Response"。

貳、WSDL[27]

WSDL(Web Service Description Language)**是一種以** XML **格式來描述** Web Services **的語言，簡單的說就是** Web Service **的** IDL(Interface Description Language)o **當** Web Service Provider **欲對外公佈其提供之** Web Services**，就必須以** WSDL **來建置描述檔案，描述內容包括**[36]**：**

1. **描述** Server **所提供的** Web Services**，以及** Web Service **可進行的各種** Operations。
2. **描述** Services Requester **如何和** Web Service **的** Operation **溝通，內容包括傳輸協定、格式、參數等。**

WSDL **內容的主要構成元素為：**

1. <Types>**：定義資料型別，各** element **實際對應之資料型態。**

2. <Message>：定義所要傳輸的訊息，各輸入、輸出 message 由那些參數 element 所組成。
3. <Operation>：定義服務所支援的動作，描述 Service 所提供的 Operation，包括輸入、輸出所需的 message。
4. <PortType> ：某一端點所 支援的 Operation 集合，此 service 所有 port 提供之全部 operation 的集合。
5. <Binding>：某一 Port 型態的通訊協定與資料格式，以及提供之 Operations。
6. <Port>：定義某端點所使用的 Binding 與網址，每一個 Port 代表外界 Client 可以和此 Service 溝通的一個進入點，一個 Port 會指定一個 Binding 的方式。
7. <Service> ：相關端點的集合，即此 WSDL 文件所要描述的 Web Services 集合。

下面為一份 WSDL 文件的範例，我們可以從 *<port> HelloWorldServerSoapPort* 所指向的 *<binding>HelloWorldServerSoapBinding* 看到 Port 之 Binding 通訊協定為 SOAP，並可找到 *<operation> HelloWorld* 接著在 <portType>找到對應的 operation，取得輸入、輸出的 Message 名稱 *HelloWorldServer.HelloWorld* 及 *HelloWorldServer.HelloWorldResponse*，然後在 <message>找到對應 的 Message，取得回應資料型態 *string*，這就是這份 WSDL 所要描述的大致內容。[6]

```
<?xml version='1.0' encoding='UTF-8' ?>
<definitions  name ='HelloWorld'  targetNamespace = 'http://im.nhu.edu.tw/wsdl/'
      xmlns:wsdlns='http://im.nhu.edu.tw/wsdl/'
      xmlns:typens='http://im.nhu.edu.tw/type'
      xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
      xmlns:xsd='http://www.w3.org/2001/XMLSchema'
```

```
        xmlns:stk='http://schemas.microsoft.com/soap-toolkit/wsdl-extension'
        xmlns='http://schemas.xmlsoap.org/wsdl/'>
    <types>
      <schema targetNamespace='http://im.nhu.edu.tw/type'
        xmlns='http://www.w3.org/2001/XMLSchema'
        xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/'
        xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
        elementFormDefault='qualified'>
      </schema>
    </types>
    <message name='HelloWorldServer.HelloWorld'>
    </message>
    <message name='HelloWorldServer.HelloWorldResponse'>
      <part name='Result' type='xsd:string'/>
    </message>
    <portType name='HelloWorldServerSoapPort'>
      <operation name='HelloWorld' parameterOrder=''>
        <input message='wsdlns:HelloWorldServer.HelloWorld' />
        <output message='wsdlns:HelloWorldServer.HelloWorldResponse' />
      </operation>
    </portType>
    <binding name='HelloWorldServerSoapBinding' type='wsdlns:HelloWorldServerSoapPort' >
      <stk:binding preferredEncoding='UTF-8'/>
      <soap:binding style='rpc' transport='http://schemas.xmlsoap.org/soap/http' />
      <operation name='HelloWorld' >
        <soap:operation soapAction='http://im.nhu.edu.tw/action/HelloWorldServer.HelloWorld' />
        <input>
          <soap:body use='encoded' namespace='http://im.nhu.edu.tw/message/'
            encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
        </input>
        <output>
          <soap:body use='encoded' namespace='http://im.nhu.edu.tw/message/'
            encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
        </output>
      </operation>
    </binding>
    <service name='HelloWorld' >
      <port name='HelloWorldServerSoapPort' binding='wsdlns:HelloWorldServerSoapBinding' >
        <soap:address
location='http://im.nhu.edu.tw/WebServices/HelloWorld/Service/Rpc/VbSrv/HelloWorld.WSDL' />
      </port>
    </service>
</definitions>
```

參、UDDI [26][28]

UDDI(Universal Description, Discovery and Integration) 是一套基於 Web 的、分散式的、為 Web 服務提供的資訊註冊中心的實現標準規範，同時也包含一組使企業能將自身提供的 Web 服務註冊以使得別的企業能夠發現的訪問協議的實現標準。圖 9 描述了 UDDI 規範、xml Schema 和 UDDI 商業註冊中心集群之間的關係， UDDI 商業註冊中心集群能為 Web 服務提供"一次註冊，到處發佈"的功能。

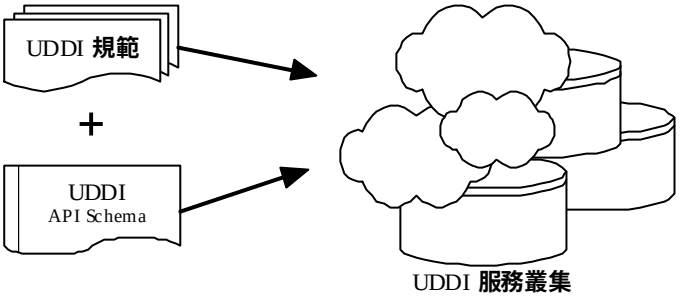


圖 9、UDDI 組成元素之間的關係[23]

UDDI 商業註冊中心在邏輯上是集中的，在物理上是分散式的，由多個根節點組成，相互之間按一定規則進行資料同步。當一個企業在 UDDI 商業註冊中心的一個實例中實施註冊後，其註冊資訊會被自動複製到其他 UDDI 根節點，於是就能被任何希望發現這些 Web 服務的人所發現。UDDI 規範和調用模式用來在 Internet 上建立起發現服務。這些發現服務提供了一致的發佈介面，以使得能通過編程進行發現。

UDDI 是以 XML 語法來描述登錄的 Web Services，並成為登錄檔與資料庫的總集。概念上來說，UDDI 可以譬喻為網上 Web Services 軟體元件與模組的電話簿，用標準的方式來登錄 Web Services 元件。在 UDDI 的技術規格中，有黃頁以分類的方式來登錄 Web Services;有白頁來記載

Web Services 之提供者、網址、聯絡方式等資料;有綠頁來登錄 Web Services 的技術細節。這三種由公司、企業所註冊想要對外公佈之訊息的 Page 主要包含：[30]

1. 白頁(White Pages)：此部分包括公司名稱、公司描述、聯絡資訊、公司代表標示。
2. 黃頁(Yellow Pages)：記載公司登錄於 UDDI 之分類，目的是讓 Service Requester 可以由分類找尋到目前有登錄之相關 Business White Page.
3. 綠頁(Green Pages)：描述 Internet(使用者如何和該公司進行電子商務，包括流程、提供服務之描述及 Service Requestor 如何 Bind 服務。

通過 UDDI 註冊，使得公司能夠把自身的描述、服務描述以及服務訪問方式的描述公開發佈。已註冊的企業能夠被潛在的交易市場或採購商所搜索到，同時，對於合作夥伴之間協調與整合也能更為動態與方便。目前參與 UDDI 的組織來自很多的行業，同時也都是這些行業的具有重要影響力的企業，而 UDDI 也將為所有行業和專案中對此標準的廣泛使用，而為全球電子商務的飛速發展帶來難以估量的價值。

肆、小結

Web Services 主要有下列優點：

- 可容易穿越防火牆[2]

  網際網路上的伺服器都是可被任何其他網際的使用者所存取的，因此需要較審慎的安全考量。為了避免企業的主機受到攻

擊以及內部資訊的洩漏，大部分的企業都在它們內部與外部網路之間加裝防火牆以防止網際網路上的大眾存取企業內部的伺服器。這些防火牆，例如代理伺服器，可以經由條件設定以阻止一些想進企業內部來的公眾網路需求，這可以大幅提昇內部系統的安全。

雖然防火牆是提供接上網際網路安全的基礎機制，但它卻會降低分散式物件通訊協定的使用效能。為了要解決這個問題，有識之士紛紛提出了各自的解決方案。在 1998 年，UserLand 公司的執行總裁 Dave Winner 提出透過 XML 讓 RPC 的通訊方式透過 HTTP 協定在網際網路上執行。這個想法經由微軟公司加以改良，提出了實際可行的 Simple Object Access Protocol (SOAP)通訊協定。由先前探討可知，SOAP 是一個像 DCOM 或其他分散式物件通訊協定的協定，讓使用者端與伺服端的 RPCs 可以溝通。但與其他類似協定不一樣的地方是，它支援防火牆的使用。同樣重要地，SOAP 不是只設計用來針對某種物件技術的協定，它不像一些時下的分散式物件通訊協定會被綁死在某一種特定的物件規格上，這個協定將可以被任何的物件便用。所以它將是兩大物件陣營 COM 和 CORBA 最好的溝通橋樑，讓彼此的物件程式可以跨平台透過網際網路呼叫。

因為幾乎所有的防火牆都允許透過埠號 80 來溝通，所以透過埠號 80 來溝通的分散式物件通訊協定將是一個較好的方案。但這並不是說說那麼容易，因為埠號 80 已經被設定給 HTTP 協定。所以 SOAP 這個分散式物件通訊協定是架在 HTTP 協定之上的。HTTP 通訊協定相當簡單，僅僅以少數基礎的動詞所組成，

如 GET、PUT、POST 等等，而這些動詞在瀏覽器與伺服器之間傳遞。而每一個動詞之後跟著一些資訊，而這些資訊通常以簡易的字串方式傳遞。

- **非緊密連結**

  SOAP 的訊息傳遞方式是簡單聯繫機制，因為 SOAP 使用 HTTP 來做為它基本的傳輸協定，所以 SOAP 本質上來說算是無狀態的協定，換句話說，它是一個 1ooselycouple 的通訊協定，這樣顯得較有彈性。這意味著您隨時可以更改任何一端的連結，但應用程式仍可以繼續運作。

- **易於進行跨異質平台的整合**

  網站服務轉換成以信息為基礎(message-based)，高效能的同步技術，及使用 HTTP 或 SMTP 的 Web 通訊協定;最重要的是，使用 XML 以達到全球通用。XML 是 W3C 所管理的開放式業界標準，它可讓程式開發人員描述交換於 PC、智慧型裝置、應用程式與網站之問的資料，因為 XML 將基本資料與該資料顯示的方式予以分離，因此它可輕鬆地在任何網站、應用程式與各種裝置進行組織、編輯、交換，所以 XML 可說是 Internet 的通用語言，藉助 SOAP 與 UDDI 等 XML 架構的技術，建立可使用 XML 來提供 Web 形式服務的全新型態軟體，這些 XML Web Service 和軟體元件非常類似，均可程式化並重複使用，不同的是你可在 Internet 的任何地方存取他們，並自每個網站擷取資訊與服務，然後以自訂的形式進行合併，並可傳送給任何裝置。由於 XML Web Service 打破了 Internet、應用程式與各類運算裝置的區隔，他們可讓企業通力合作，提供空前的高度整合與可自訂的解決方案，

可讓客戶們在任何時問、任何地點與任何裝置上處理資訊。當各家企業逐漸以 XML Web Service 作為新世代的 Internet 架構運算之後，即可開發一個平台，以便更輕鬆地建立解決方案，並提供可靠的架構來進行整合與交互操作，而這類平台必須奠基於開放式的標準，以適用於所有的程式語言、作業系統與應用程式。

第四節　統一塑模語言

在軟體業中，曲於許多軟體系統無法符合顧客的需求、超出時程或預算而宣告失敗，而有軟體危機的形成，為了增加系統開發的品質，並減少開發成本及時問，新的軟體開發技術如：元件技術 (Component Technology)、以視覺化程式設計(visua1 Programming)、樣板(Patten)、結構(Framework)等技術紛紛被提出，然而，當系統的範圍(Scope)及規模(Sca1e)變大時，尋求一簡單的方法來管理複雜的系統變得益形重要。

故有許許多多物件導向的方法被提出，如 Booch、OMT、OOSE 以及 Fusion，以用來協助開發軟體系統，解決軟體危機，然而，這些方法很難去區分出那一個是最好且最正確的，而統一模型語言 UML(Unified Modeling Language)統一了 Grady Booch (Booch)、James Rumbaugh (OMT) 以及 Ivar Jacobason (OOSE)這三位所提出的方法，並引入其他方法中的一些概念，成為一物件導向分析與設計的標準方法，在 1997 年底，OMG 決定將 UML 當作其標準之一[3]。UML 的主要目標在於

- 提供即時可用的視覺化表達之模式化語言(Modeling Language)，讓使用者可以發展與交換模型。
- 提供一可擴充的、規格化的機制去擴充核心概念。

- 與特定的程式語言及發展程序無關。
- 提供模式化語言正式的基礎。
- 激勵物件導向工具市場的成長。
- 支援高階的發展概念，如合作(Co1laboration)、結構(Framework)、樣板(Patten 及元件(Component)。
- 整合已實踐的軟體開發中最好的方法。

UML 可用來詳細說明(Specifying)、視覺化(Visualizing)、建構(Construction)與文件化(Document)人造的各類系統，UML 代表最佳的模型開發活動之集合，且這些實踐已證明出可以成功地模式化大型且複雜的系統。UML 並且也提供了許多種類的圖形表示方法來協助程式開發者分析與設計系統的各個層面，以模型組成符號的排列來展示出系統的某一特定部分運作或功能，UML 中的的圖形包括了：使用者案例圖(Use Case Diagram)、循序圖(Sequence Diagram)、合作圖(Collaboration Diagram)、類別圖(Class Diagram)、狀態圖(State Diagram)、活動圖(Activity Diagram)等，在發展系統時並不一定要用到所有的圖形，系統發展者只需要選擇其所需的圖來描述即可。[1]

以下說明本論文後面章節中會用到的幾種圖示

- 使用者案例圖(Use Case Diagram)：由 Actor 及 Use Case 兩個元件所組成，主要是用來表達系統外部所見的功能，以系統的參與者觀點去表達系統的行為。

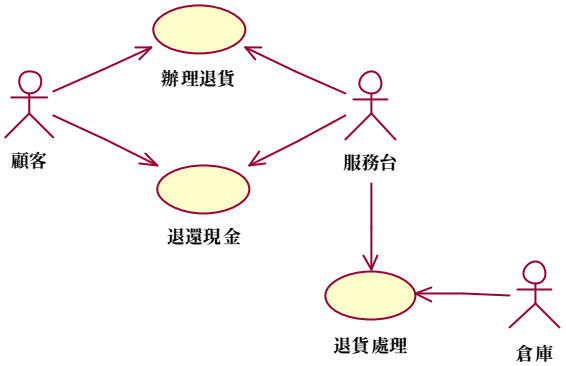


圖 10、使用案例示意圖-以退貨流程為例

- **循序圖**(Sequence Diagram)：**以時間的順序來表示物件之間訊息的傳遞，其捕捉一個使用案例的行為模式，並幫助瞭解控制流程。**

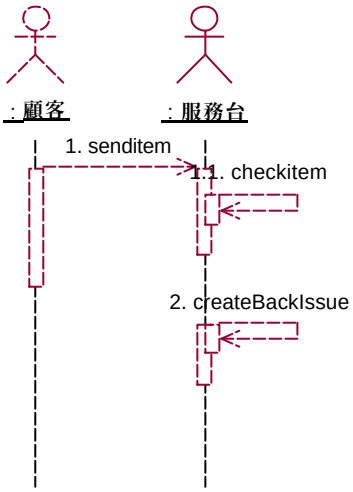


圖 11、循序圖示意-以辦理退貨為例

- **類別圖**(Class Diagram)：**表達物件及物件之問的靜態關係，從類別圖層可以看出軟體的組織架構。**

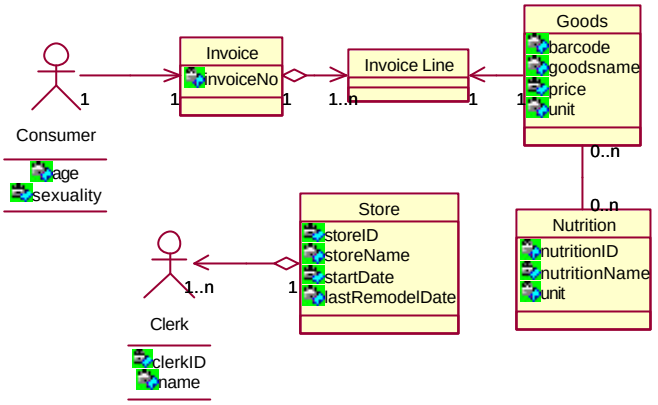


圖 12、類別圖示意-以 Sell good Model[4]為例

在本論文的物件導向分析與設計當中，我們主要以三種類別來表示三層式架構中各類別的特性：

- **邊界類別**(Boundary Class)：**與其他系統之間互動的實際界面，也有可能是使用者界面，用以界定使用者在系統邊界上的需求。**
- **控制類別**(Control Class)：**用以協調物件間的運作，也可將複雜的企業邏輯封裝起來。**
- **實體類別**(Entity Class)：**代表保存的資訊，亦即負責資料庫的物件。**

第三章、以 Web Services 為基礎的應用整合架構

本章將介紹本論文所提之以 Web Services 為基礎的應用整合架構，第一節將先介紹整個系統的發展步驟，然後再第二節進行系統的需求分析；以便於第三節進行架構設計；而進一步的系統元件分析與設計則將於下一章說明。

第一節　系統發展步驟

本研究採用物件導向方式的分析與設計來開發系統架構，其通用的物件導向發展流程如圖 13 所示。其中主要的步驟分為以下幾項，說明如下：

1. **系統需求分析**

   本研究依據第二章所作關於企業應用整合的相關考量作為系統需求分析的主要依據，另外考量目前現有分散式物件技術的優缺點，對於建立一個改良式應用整合系統的架構進行需求分析；並考量 Web Services 技術中對於支援企業應用整合的優點以進行結合。

2. **應用整合架構設計**

   在進行整合架構設計時，我們採用 UML 作為本研究進行物件導向式分析與設計時的塑模語言，在建構實際系統之前先將其模型化；UML 是 OMG 標準，目前正準備成為 ISO 標準；我們試著以標準的圖形與語言去描述研究中的系統架構；以方便讀者閱讀理解與後續發展時的陳述和溝通，在這個步驟中；針對前一步驟的需求進行功能的規劃與設計，然後我們將先依這些系統外部所

能見的功能進行使用案例的規劃；以外部功能的觀點去表現出系統內部類別應有的整體輪廓，並依不同的事件流程繪製成使用案例圖以供後續元件類別分析時能夠取用。

3. **系統元件模型設計**

   經由上一步驟的使用案例分析之後，將會產生許多不同的使用案例；在此步驟再針對每一個使用案例進行類別的發掘與建構，初步找出系統之中所可能會有的所有類別之後；並繪製出初步的類別互動循序圖，然後；我們將再「漸進式」、「反覆式」的將各類別之間的靜態關係加以修整；設計，並對類別重新找出最佳的循序圖方式進行類別的妥適性驗證；以期能找出最適的類別模型。這個部分我們將在第四章專門進行討論。

4. **應用系統實例建構**

   在這個步驟中我們將開始考量當部署這樣的一個架構在真實環境中時將會面臨那些問題，並以知識管理[19]具備的七層架構來做為探討本研究中的系統架構所扮演的角色的實例；找出當導入本架構之時將會面臨什麼樣的改變與效益。

5. **系統整合測試**

   最後依本系統架構的應用實例進行系統整合與測試，以展現本論文的研究成果。

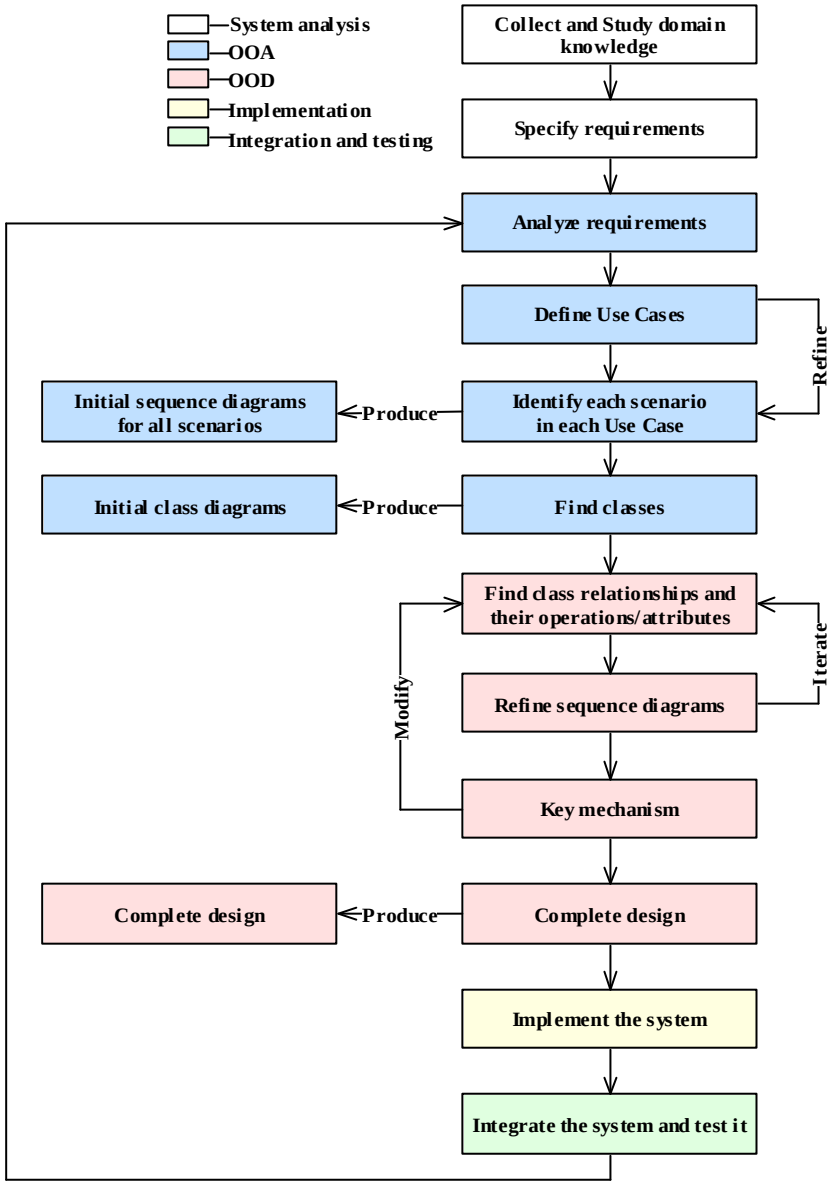


圖 13、通用物件導向方式分析與設計流程圖[20]

第二節　系統需求分析

在第二章時我們綜合各學者之論點彙整得知出；企業應用主要整合資料、訊息和程序流程、發展一個適用於不同平台的應用程式，包括企業自行開發的軟體或套裝商用軟體，目的是為了整合各種在異質環境下的軟硬體，來產生一個好用又有效率的應用系統整合。而且企業應用整合的基本需求是希望能夠分享資料而不改變本身之應用程式和資料儲存

結構[34][45]；企業應用整合發展趨勢在於企業對企業(B2B)的整合以及 Web 功能應用企業的需求。關於如何評估企業應用整合是否符合企業之需求，我們整理了 Fisher 所提供六點關於企業進行企業應用整合時，所應考慮的要素[17]：(1).選用之軟體開發技術是否適當?(2).如何整合資料之間的傳遞?(3).時間與成本的預算支出為何?(4).是否已針對核心系統進行風險評估?(5).目標市場的設立是否明確?(6).系統整合度如何，效率才會提高?

綜上所述，在企業環境中依據其不同產業特性，安全或是成本的考量底下；各自採用不同的作業平台、程式執行環境、資料庫系統以及網路架構；在如此異質性環境中，不論是彼此間資料的傳輸與交換或者是彼此間應用服務的相互溝通協調都是相當困難且複雜的問題，若藉由第二章所談到的 SOAP 協定所具備的可擴充性、跨平台、可穿越防火牆等種種優點，將能解決此類溝通問題。因此在研究中我們使用 SOAP 作為最基礎的 Web Services 叫用協定，並利用 Web Services 當中的 UDDI，WSDL 等技術加入系統的設計，協助我們能夠方便的找尋到網路上其他夥伴的應用服務，而為了進行整合服務時額外撰寫程式造成整體系統的複雜度與後續的維護成本；本研究所提出的架構要能夠讓使用者透過最簡單的介面點選動作完成整合與應用模組的建構，最後再由系統依模組的設定，產生一個整合型的操作介面，供使用者輕易的操作跨系統的服務。

第三節　應用整合架構設計

本研究經由前面章節對應用整合系統之功能需求上以及架構上討論，在此節根據之前討論結果建立起應用服務架構，本系統架構如下圖 14 所示，主要分為伺服端、服務端、客戶端；將說明於後，並進一步針

對各種情境進行使用案例分析與說明。

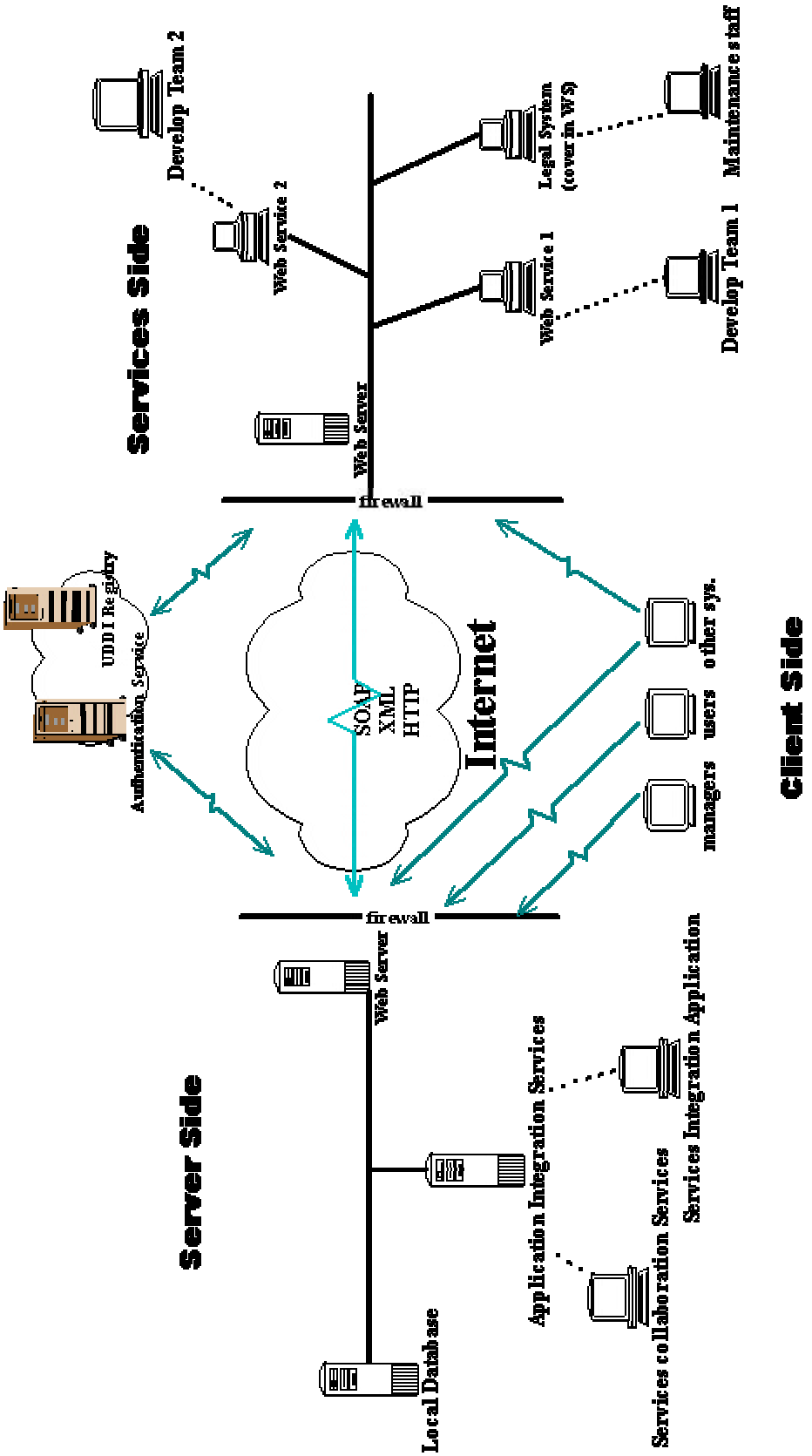


圖 14、Web Services 服務應用整合系統架構圖

壹、服務端(Services Side)

服務端在本研究架構中的定義是指企業內部或者是夥伴之間現有的應用系統或是服務元件；經由 Web Services 技術以 WSDL 加以包裝、陳述；然後透過 Web Server 散佈在網際網路上或者是登錄於 UDDI 伺服器上供該企業服務的使用者查詢與叫用。由於服務端可能位於本系統架構中的其他組織或是跨企業的電腦系統之中，主要使用 SOAP 透過網際網路傳遞叫用，而其他服務使用者只能依據其所散佈的 WSDL 陳述進行叫用；並不會知道該系統的實際商業邏輯，而實際系統的維護仍然是由原公司組織的發展小組負責，又因為 XML 格式的通用性，在訊息的傳遞上並不會受限於相同的環境或平台而進行緊密的結合。

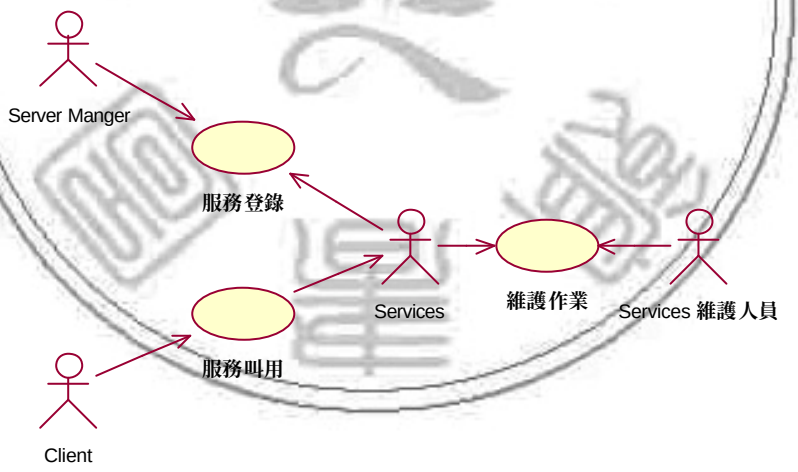


圖 15、以 Service 端為主的使用案例示意

貳、客戶端(Client Side)

主要定義為應用整合系統的管理者，應用服務的使用者，以及可能

會需要這些服務的其他系統；管理者主要負責將服務端的資訊在伺服端進行登錄、整合；而服務使用者則依伺服端整合後產生的介面進行服務的操作，其他的系統部份則是一方面可以直接叫用服務端的服務，另一方面則依將伺服端產生的公開叫用介面進行呼叫。

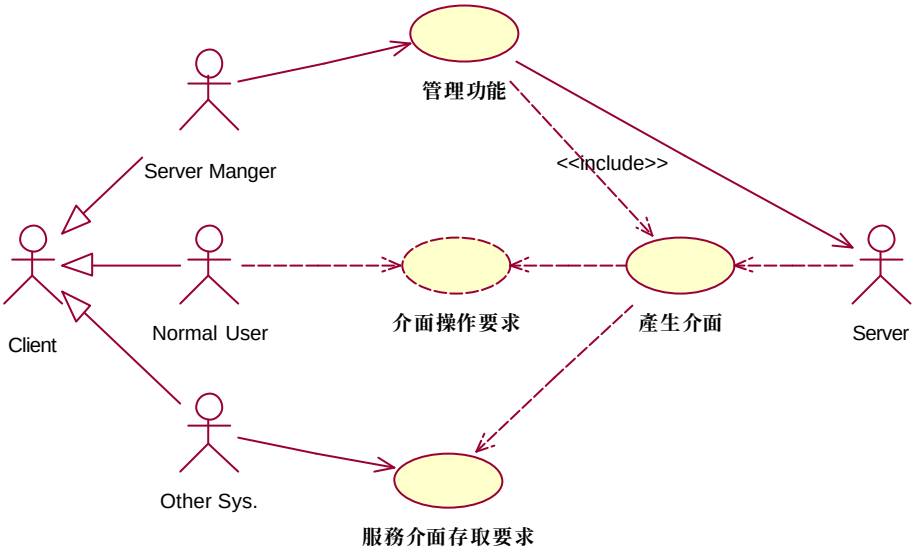


圖 16、Client 端使用案例示意

參、伺服端(Server Side)

主要提供管理者的操作介面，將伺服端的服務進行登錄，整合；並產生一個整合式的操作介面，供一般服務使用者透過單一的介面存取服務，再依介面與實際登錄服務的對應關係去叫用服務，使用者將省去直接連結服務端時的不方便；另一方面伺服端的介面亦可以考慮再包裝成一個整合型的 Web Service 供其他系統呼叫。

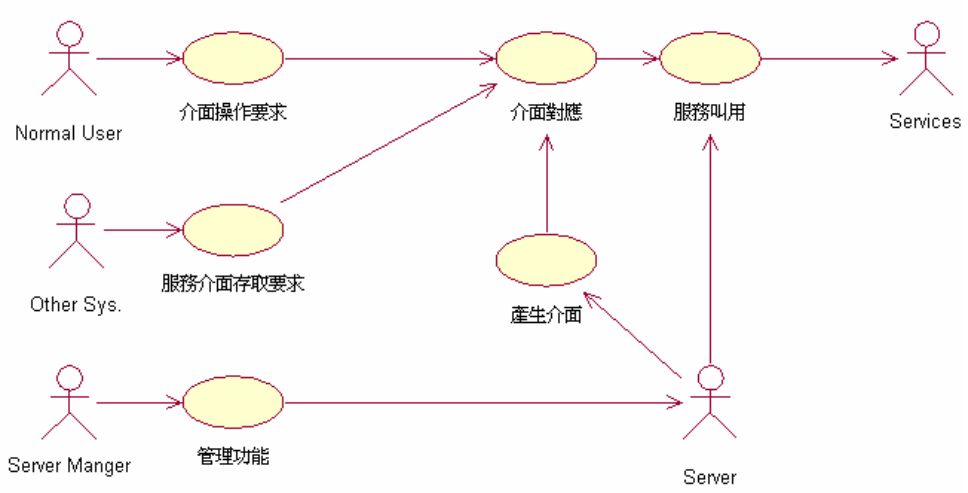


圖 17、Server 端使用案例示意

下圖為本研究中建立的應用服務伺服器相關層次圖，依其任務的不同分為(1)介面層；主要處理與使用者間資訊呈現的格式、(2)企業物件層；放置伺服器主要的邏輯與運算物件、(3)實體層；應用服務平台的實體部署方式、(4)傳輸交易層；依任務需求專司與資料庫系統進行查詢、編修等要求，並將運算結果傳回給平台。

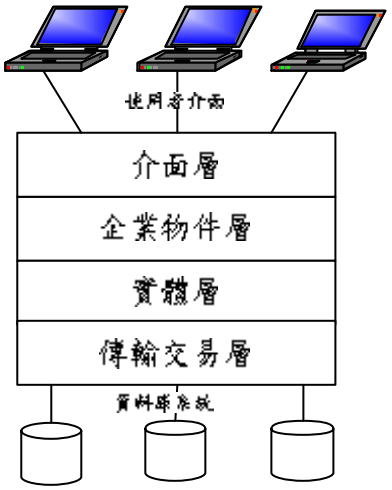


圖 18、應用服務伺服器層次圖

至於應用平台在企業中的實際環境部署則由下圖可以看出，主要希望能夠透過 Web Server 作為介面呈現的方式，而後端各層則希望能以 Servlets 的方式去實做。

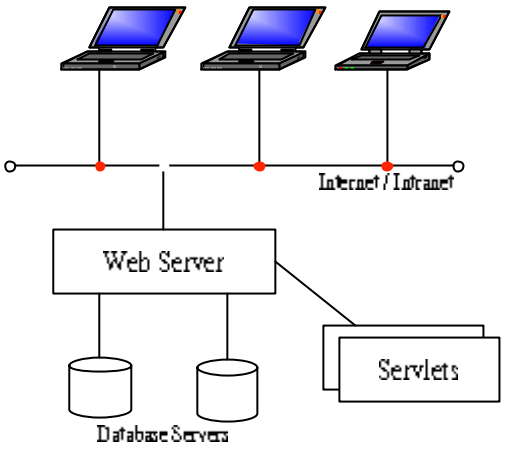


圖 19、企業內操作環境圖

肆、系統架構使用案例分析

我們依照系統在運作的時候所需要的操作主要分為以下幾項機制，並將對其一一說明如下：

- **Web Services 瀏覽與登錄機制**

  本機制主要將 Web Service 格式之元件對應到系統之中，並給予一個特別的元件代碼供自身辨認。由於 Web Service 特性，內部元件代碼跟外部元件實體可以是一對多關係；本機制主要的操作包括遠端 Web Services 的查詢瀏覽，然後針對所查詢的資訊進行解析後；決定是否儲存於系統之中；另一方面則是提供現有系統中登錄的紀錄查詢；以便對其進行刪除與修改等作業。

  - Web Services Explorer

    此使用案例用來表示系統提供對於遠端 Web Services 的即時查詢服務，使用者可以輸入 Web Services 的 WSDL 檔網址，

由系統進行內涵的解析，辨識出該服務擁有的屬性與行為等等的資訊。

- Web Services Record Saving

  此使用案例是當使用者查詢到遠端 Web Services 的部分，針對該遠端 Web Services 進行相關資訊的登錄，諸如服務的分類；群聚，屬性，方法，參數名稱，以及其他的補充資訊如服務品質，替代服務的索引值等，都透過這個使用案例進行紀錄；以供後續回溯叫用時使用。

- Web Services Record Querying

  針對系統目前的紀錄進行細節資訊的查詢。

- Web Services Record Modifying

  針對系統目前的紀錄進行細節資訊的編修動作。

- Web Services Record Deleting

  針對系統目前的紀錄進行細節資訊的編修動作。

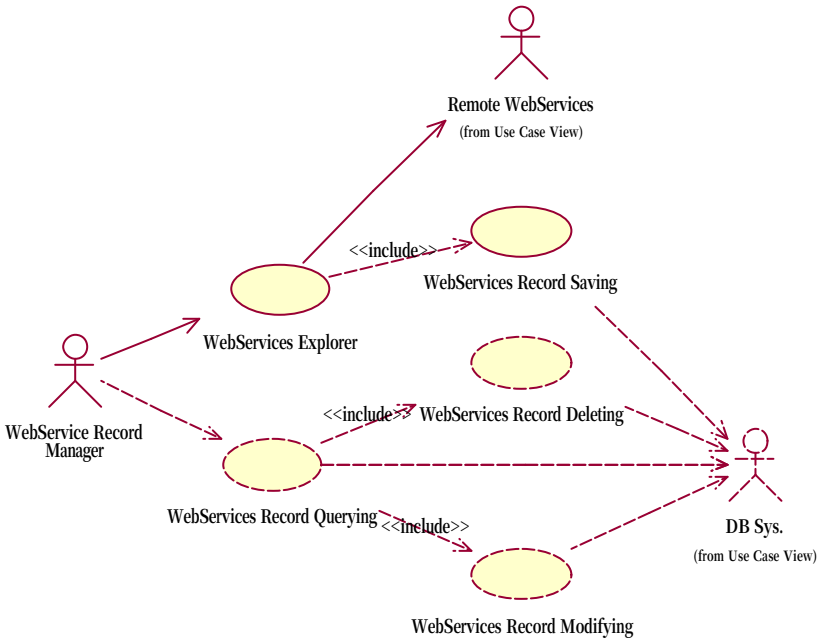


圖 20、Web Services 瀏覽與登錄使用案例圖

- **動作定義與登錄機制**

定義單一元件間的動作，並給予內部代碼。本機制主要為定義個別元件的動作，並將其對應到系統紀錄中，定義動作紀錄前需先對系統內的 Web Services **紀錄進行查詢，選定所要定義的動作所屬服務，才能進行定義以及之後的新增動作。**

- Action Record Define

  在管理者對於服務紀錄進行查詢的時候，將能夠進一步針對服務所屬的動作與屬性進行定義與描述的動作，以供後續存檔利用。

- Action Record Saving

  當管理者完成服務相關屬性的定義與描述之後，便會產生一個暫存的定義格式，然後進入到這個使用案例，作進一步的確認後，再加以紀錄儲存。

- Action Record Querying

  針對系統目前的屬性與動作紀錄進行細節資訊的查詢。藉由關鍵字串進行相符的動作及屬性名稱的搜尋，或者是依照登錄時的描述進行全文檢索的動作，然後再列出相符合的服務紀錄相關細節資訊。

- Action Record Modify

  針對系統目前的紀錄進行細節資訊的編修動作，使用者須先查詢到該筆紀錄之後才能夠進行修改。

- Action Record Deleting

  針對系統目前的紀錄進行細節資訊的編修動作，使用者須先查詢到該筆紀錄之後才能夠進行刪除。

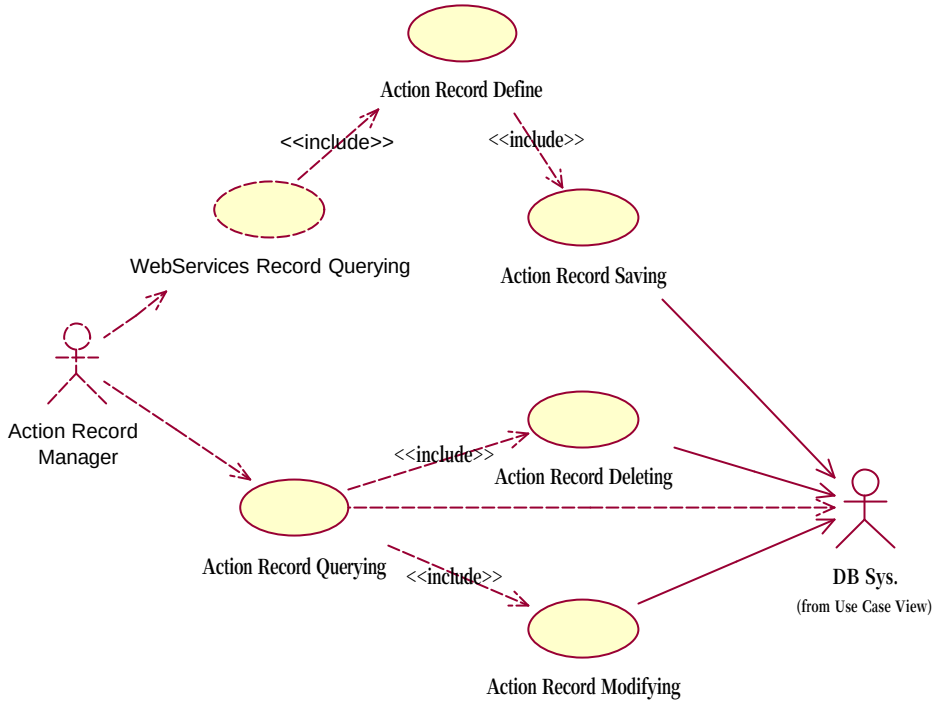


圖 21、Web Services Action 登錄使用案例圖

- **程序定義**

定義單一元件間互動行為，並給予內部代碼。由於先前已經紀錄了服務元件所屬的屬性跟動作，在此則是將動作定義為一個程序，而意指單一動作行使時的參數，以及動作完成後的回傳值，都可由另外一個屬性進行銜接，另外程序的定義也可以是包含了兩個不同的屬性之間的運算關係。

- Process Record Define

  程序的定義分為動作跟屬性兩種類型，動作程序分別有(A).**純動作，不回傳值** (B).**參數動作， 回傳值** (C).**參數動作，不回傳值、**(D).**純動作，回傳值；此類動作程序的處理在於將動作的參數來源與其回傳值的去向與前述的** Action Record **作一個關聯性，定義其關聯性關係。。**

- Process Record Saving

  經由上述動作定義完成之後，在此進行存檔確認後進行程序的紀錄。

- Process Record Querying

  此處提供一個能輸入索引鍵值，進行相關資訊查詢或是描述性資訊檢索的功能。

- Process Record Modify

  經由查詢所得的紀錄以及相關細節，於此處能夠進行進一步的編修。

- Process Record Deleting

  經由查詢所得的紀錄以及相關細節，於此處能夠予以刪除。

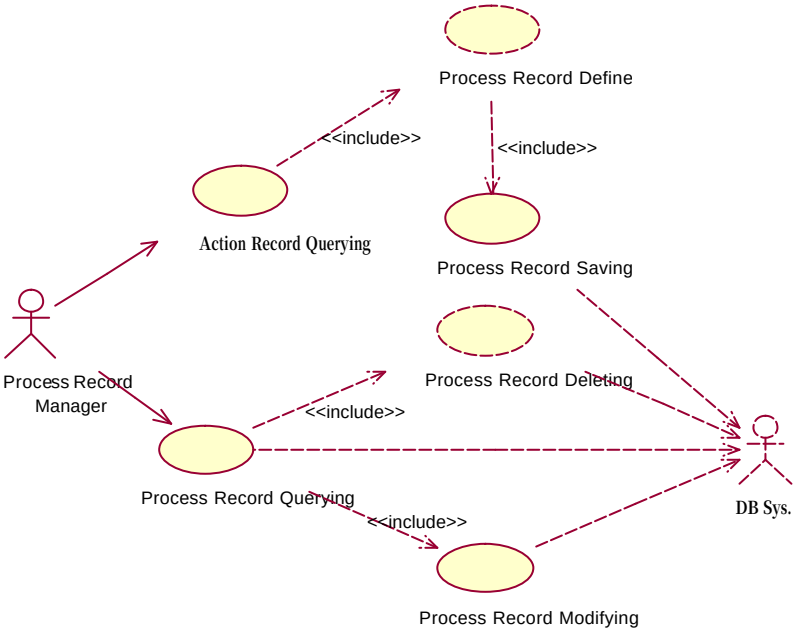


圖 22、程序管理介面使用案例圖

- **程序排程定義**

  定義若干基本程序的工作排程，並給予內部指令代碼，此處包含

了工作流程[24]的概念，主要是希望將先前所定義的基本程序，其有連續性質關係存在者，在此進行排程的設定，亦即一個排程當中紀錄著許多不同的程序運作，最後在紀錄於排程紀錄中時，也同時存入程序紀錄之中，並提供一個欄位供分辨與基本程序的不同，供程序回溯呼叫時使用。

- Procedure Record Define

  此處將對若干有連續性或者有類似目標的程序進行排程，並紀錄成為一個排程紀錄，其可包含了前述各類排程的運作，相互關連性，並可定義過渡傳值所需的變數，以及排程的相關細節描述與最後回傳值的去向都做一個完整的定義。

- Procedure Record Saving

  將先前定義後的格式予以確認後存入排程資訊紀錄，並且將相關紀錄另存一份至程序紀錄當中，方便程序代理人統一管理；但是仍將在紀錄中給予一個辨識欄位以利區分。

- Procedure Record Querying

  針對所輸入的索引鍵進行查詢以及相關的資訊檢索。

- Procedure Record Modify

  根據查詢所得出的程序紀錄進行相關資訊的修改。

- Procedure Record Deleting

  根據查詢所得出的程序紀錄進行相關資訊的刪除，並同步刪除原先建置在程序紀錄中的對應紀錄。

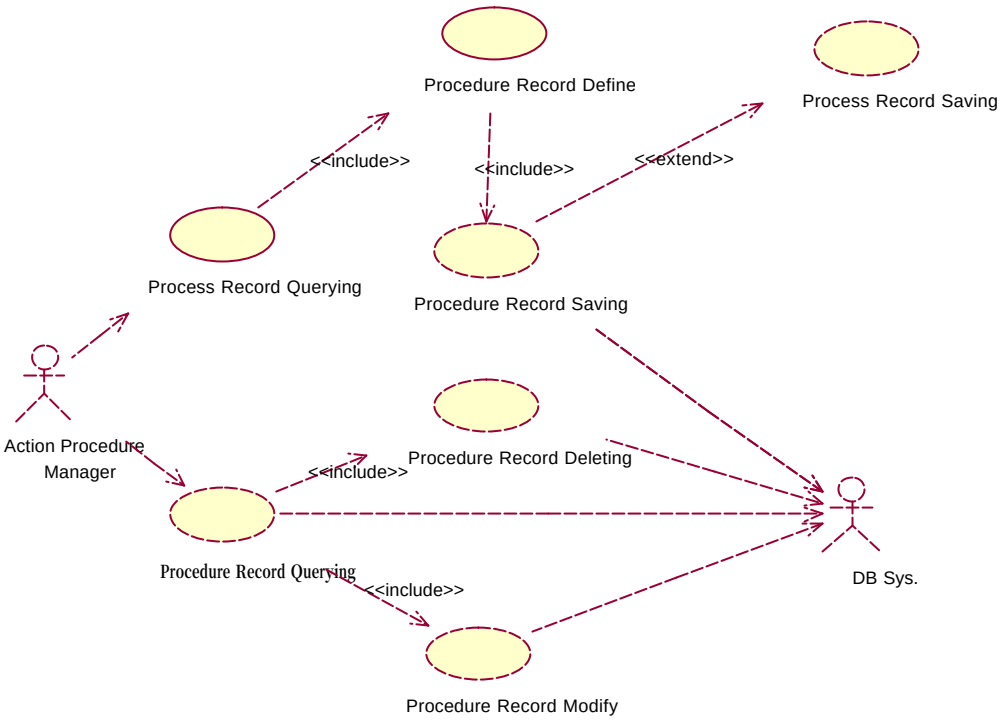


圖 23、程序排程登錄使用案例

- **模組建構機制**

彙整系統各項內部元件資訊(包含單一動作或屬性)、互動與排程等，總合成一完整功能的管理模組。模組即為執行相同任務的服務元件組合，模組紀錄能夠擁有自己的屬性，行為；以及其任務的相關描述。

- Model Define

  模組本身擁有自己的屬性，行為，方法；因此在定義時亦分為若干動作，屬性設定主要分為，模組屬性紀錄(定義模組叫用屬性時所需使用的代碼或名稱)，模組屬性取得對應(定義當欲取得模組屬性時所需進行的程序)，模組屬性存入對應(定義當模組屬性值被存入時需對應存入何處?)，動作紀錄

(定義模組的動作代碼跟名稱), 動作叫用參數對應(定義模組的動作所需的參數取得後傳遞至何處?), 動作回傳值對應(定義當模組動作執行完畢後回傳值需要作何種處理?)等相關動作。

- Modeling Record Saving
  經過確認之後，對所定義的語法合法性進行檢測，然後將先前所做的定義依一定格式存入模組紀錄檔中。

- Modeling Record Querying
  讓管理者依關鍵字進行索引的搜尋或者是描述性資料的全文檢索。

- Modeling Record Modify
  針對查詢到的紀錄進行編修與調整。

- Modeling Record Deleting
  針對查詢到的紀錄進行整個模組定義的刪除的作業。其中可以選擇模組中的個別行為，屬性等相關細節紀錄進行刪除，或者是要求與模組相關的其他樹狀節點的相關紀錄全數刪除作業。

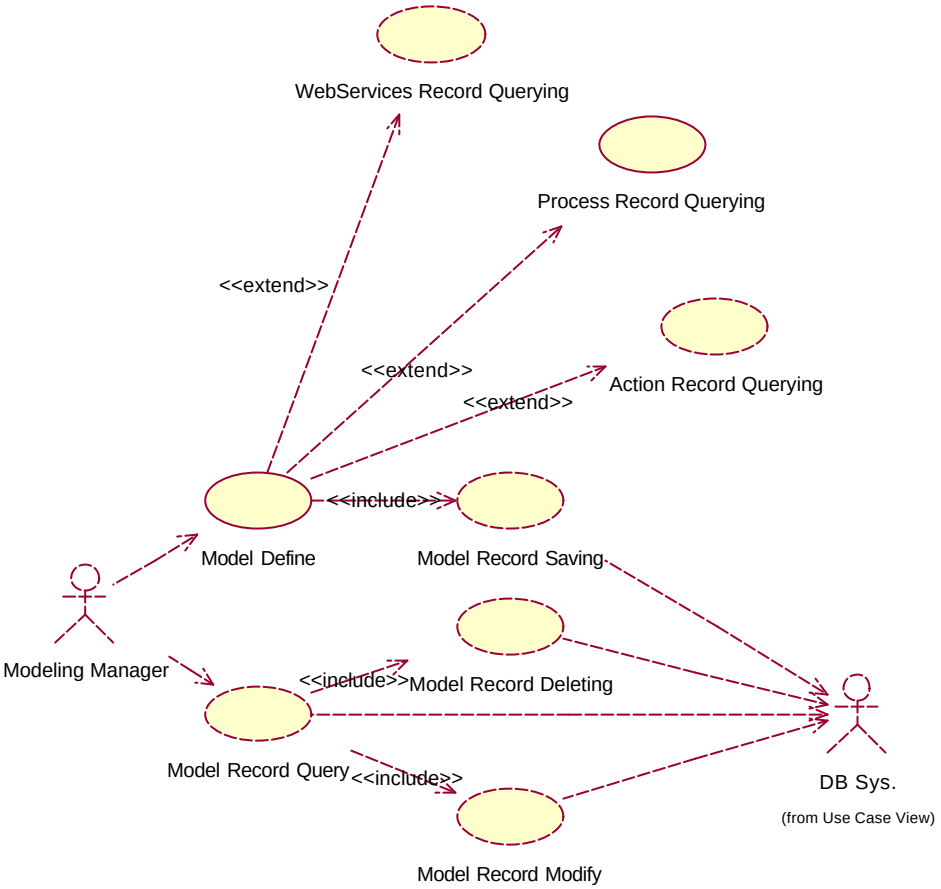


圖 24、模組建構機制使用案例

- **操作介面定義與模組關聯對應**

定義系統內所有模組的功能定義成統一操作介面。未來介面本身可能有不同的呈現方式，例如網頁結構的介面(ASP, ActiveX, **或是** Java Applet)**，或者是** Web Services **中** WSFL **形式的介面，經由匯整不同模組而組合成單一的操作介面，並且紀錄介面操作上的種種動作與模組間對應的關係。**

- Interface Define

  此處主要是將特定介面的呈現方式與模組行為的對應關係

做一個明確的定義，使得後續當使用者在介面上操作時的資訊能夠傳遞到個別模組上。目前可行的做法包括將含有特定標籤的 HTML 介面檔上傳後與模組資訊進行組合；或者是由系統產生一份標準包含特定呼叫標籤的表單(form)供使用者自行組建操作介面，再將操作介面的檔案上傳至系統開放的資料夾中。另一種較高階的做法則是讓使用者自行在系統管理介面上透過簡易的步驟對新的介面進行動態的產生；編排與定義；但這部分由於技術上受到系統開發時所選用的語言限制需要再進行考量。

- Interface Record Saving
  將所定義的對應格式進行存檔的工作，其中包括了介面的定義檔，與模組的對應關係檔，介面的其他相關描述等資訊。

- Interface Record Querying
  針對對應格式的紀錄進行查詢與調閱，以利後續編修作業進行。

- Interface Record Modify
  對查詢出來的介面對應紀錄進行編修作業。

- Interface Record Deleting
  對查詢出來的介面對應紀錄進行刪除作業。

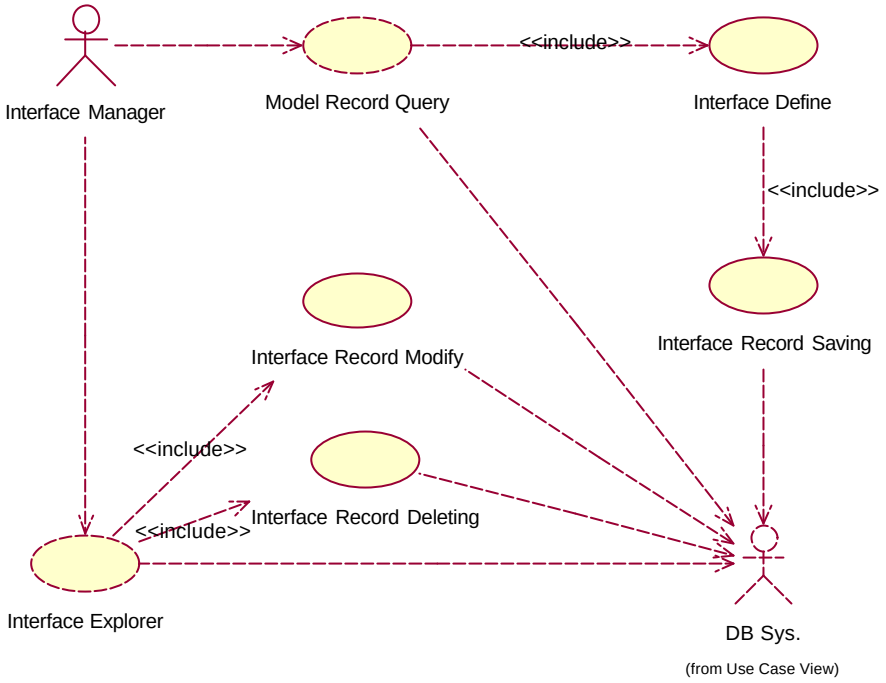


圖 25、操作介面定義與模組關聯使用案例圖

第四節、小結

本章希望透過循序的建構過程，由最基礎的服務屬性與行為；逐步將程序、排程、模組等紀錄建構完成，最後再進行與介面的對應定義，希望能夠產生一個單一的存取介面，讓使用者能夠在此介面的操作過程中，不需要去考慮其背後的服務實際所在的地點或者是服務實際運作的方式為何；另外我們在程序的定義上引進了排程的觀念，當我們這個系統中的服務元件主要目的是虛擬組織或是協同式商務環境中的時候；這將會是非常有幫助的一項機制。

我們目前已經討論完畢關於整合應用服務的相關機制，一直到介面的對應提供使用者呼叫的部分，接下來我們將在下一章開始討論其中的類別細節，以及當我們從介面對應模組的回溯過程。

第四章　系統元件之分析與設計

在第三章的使用案例分析中，針對個別的使用案例進行類別的發掘與分類；對各類系統機制中的元件進行分析與設計，其中找出了服務整合元件、訊息傳遞元件、程序代理人元件與介面對應元件等類型。關於使用案例圖中的各項類別之間的循序圖請參照附錄；以下將依這些元件所進行的活動以類別泛化、服務整合、介面對應回溯等三個層面進行說明。

第一節　類別泛化

由於某些類別具有相同的操作與屬性，因此我們將他們集合成一個介面；我們將管理介面，代理人，資訊實體這三個類別都定義了一個通用的介面，而相關的類別在依其處理的目標不同而自行額外擴充新的屬性與方法。即是表達了各管理介面都是繼承同一個介面類別，這在我們進行類別的循序圖訊息指定時將會很有幫助。關於上述各類別中操作方法的運作請參閱附錄各循序圖以及第二節的各泛化類別圖。

第二節　服務整合類別

在本節中將討論到關於進行服務整合時所需之元件，主要包含了幾種類型的類別，分別是管理介面類別、訊息傳遞類別、代理人類別以及實際存放資訊的實體資訊類別。以下先對其進行定義與說明：

壹、管理介面類別

管理介面類別主要是提供管理者進行管理操作時所面對的畫面，管理介面類別必須要能夠提供管理者操作所需的資訊，並且能夠將所做的操作對應翻譯成一定的訊息格式，在經由訊息傳遞類別傳遞給代理人進

行實際的運作。

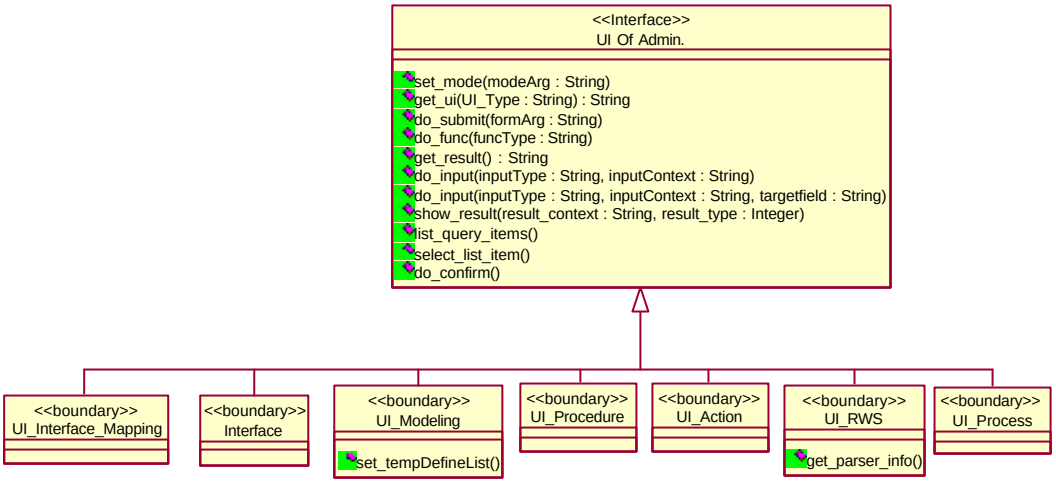


圖 26、管理介面泛化類別圖

貳、訊息傳遞類別

訊息傳遞類別主要是接受介面操作所產生的訊息，這代表了一份經過設計的格式，並且要被代理人所接受，執行，某些較高階的代理人在進行操作時也會產生訊息給低階的代理人要求服務或資訊。

參、代理人類別

代理人類別是系統中主要執行跟進行處理的類別，由接收介面層次傳遞而來的訊息所驅動；主要的運作是對資訊實體進行操作以及與其他代理人進行任務的細部分派、溝通、協調。

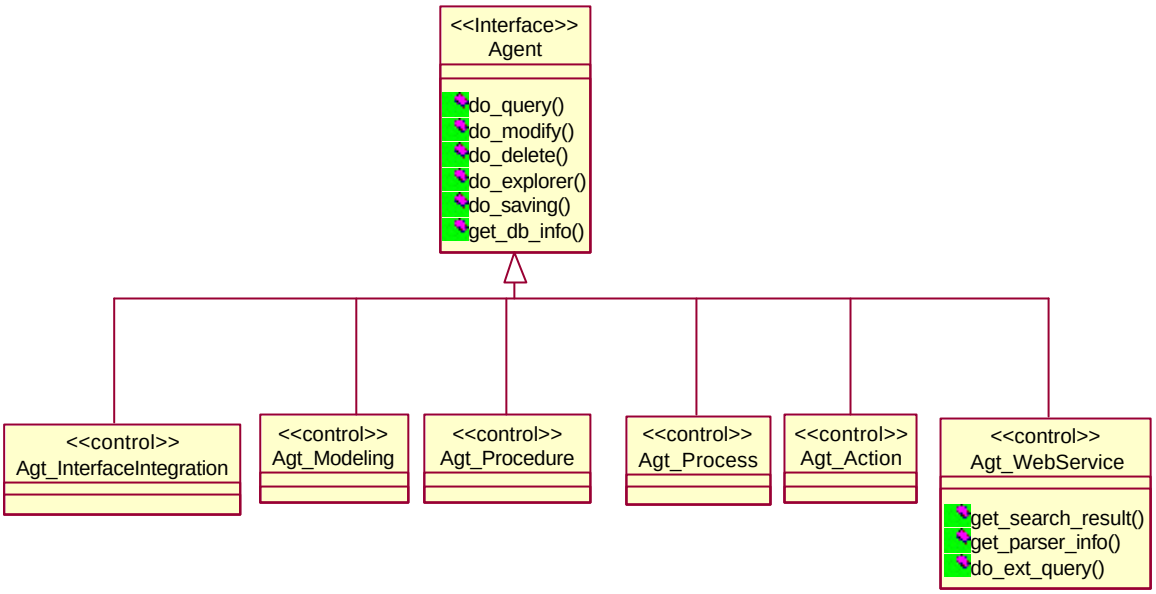


圖 27、代理人介面泛化類別圖

肆、實體資訊類別

實體資訊類別包含了進行實體資料存放的管理類別以及資訊本身兩者，在本研究系統架構中定義了若干層次的的實體資訊類別，並將其泛化成統一標準的操作介面，再依個別資訊類型的不同而各自延伸自己的特殊操作和屬性。

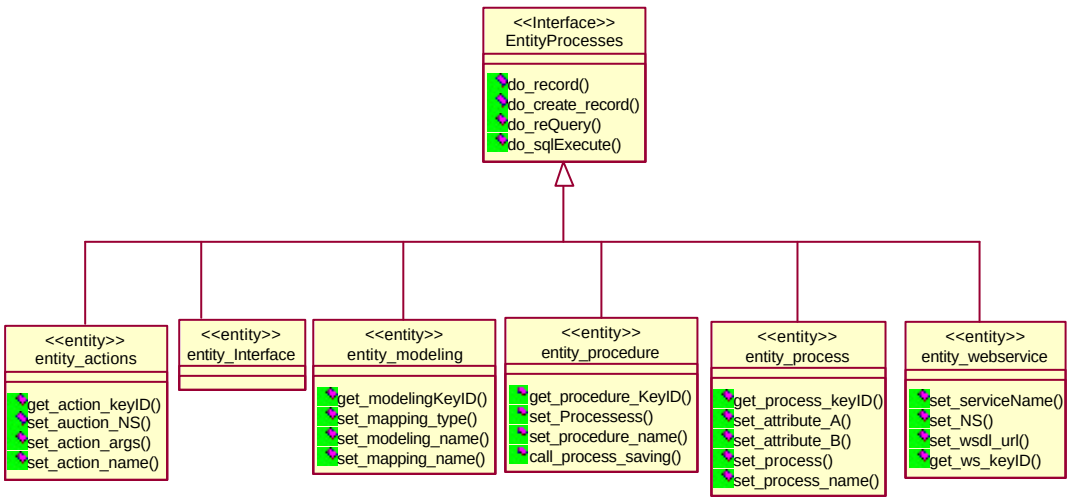


圖 28、實體資訊介面泛化類別圖

伍、服務整合類別圖

我們根據各項類別的特性與在各使用案例的角色，產生以下類別圖以表達類別之間的關係。

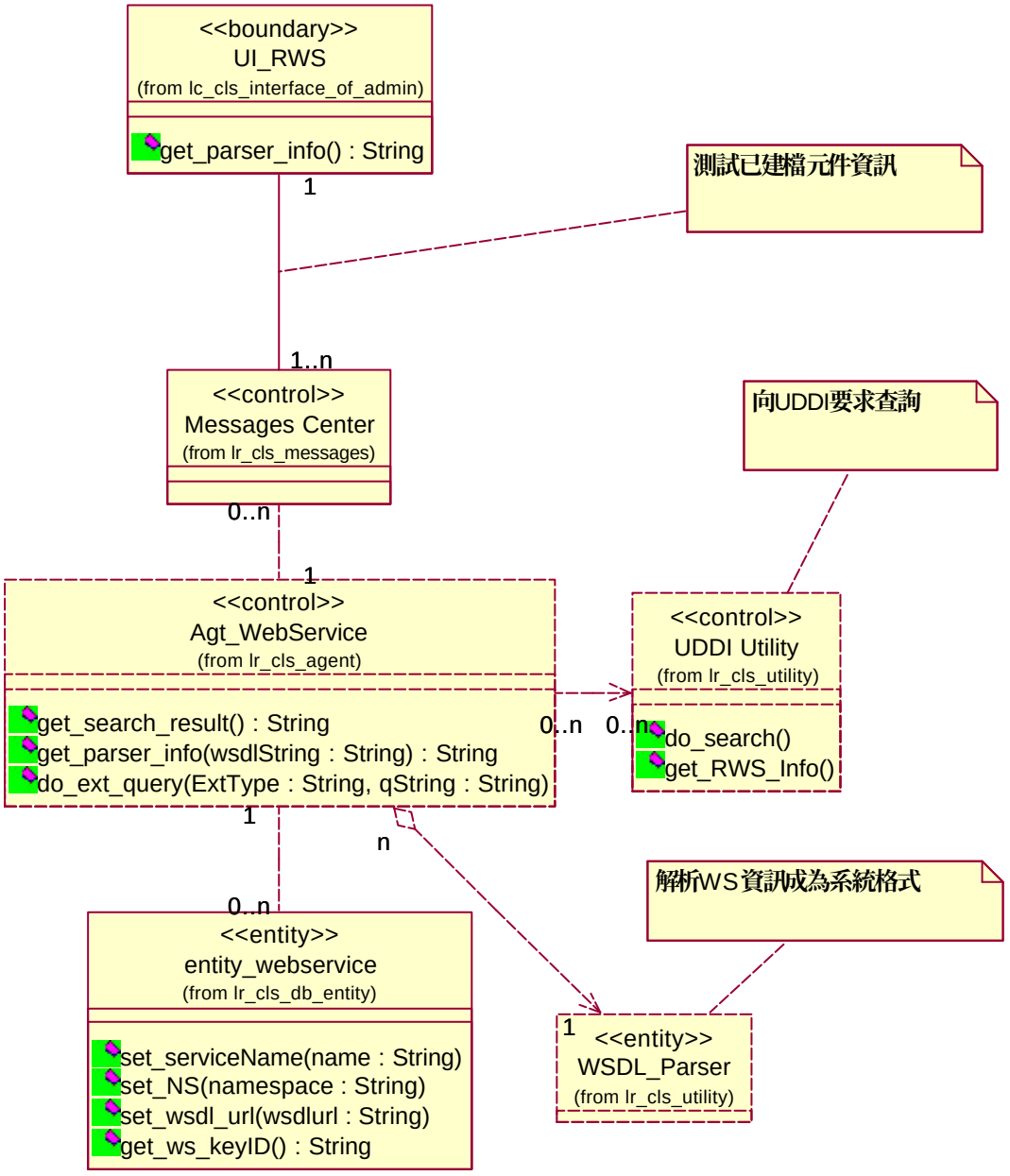


圖 29、遠端服務搜尋與登錄類別圖

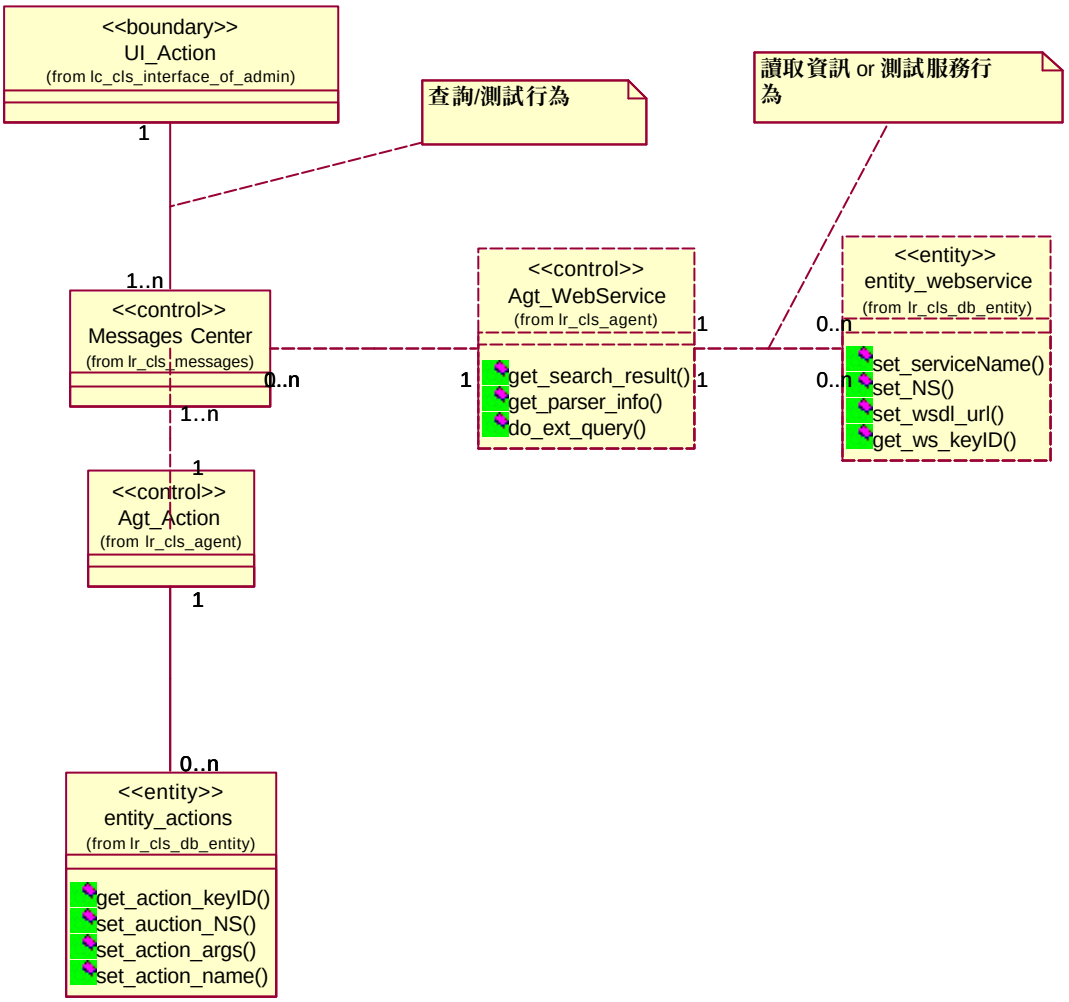


圖 30、服務行為資訊登錄類別圖

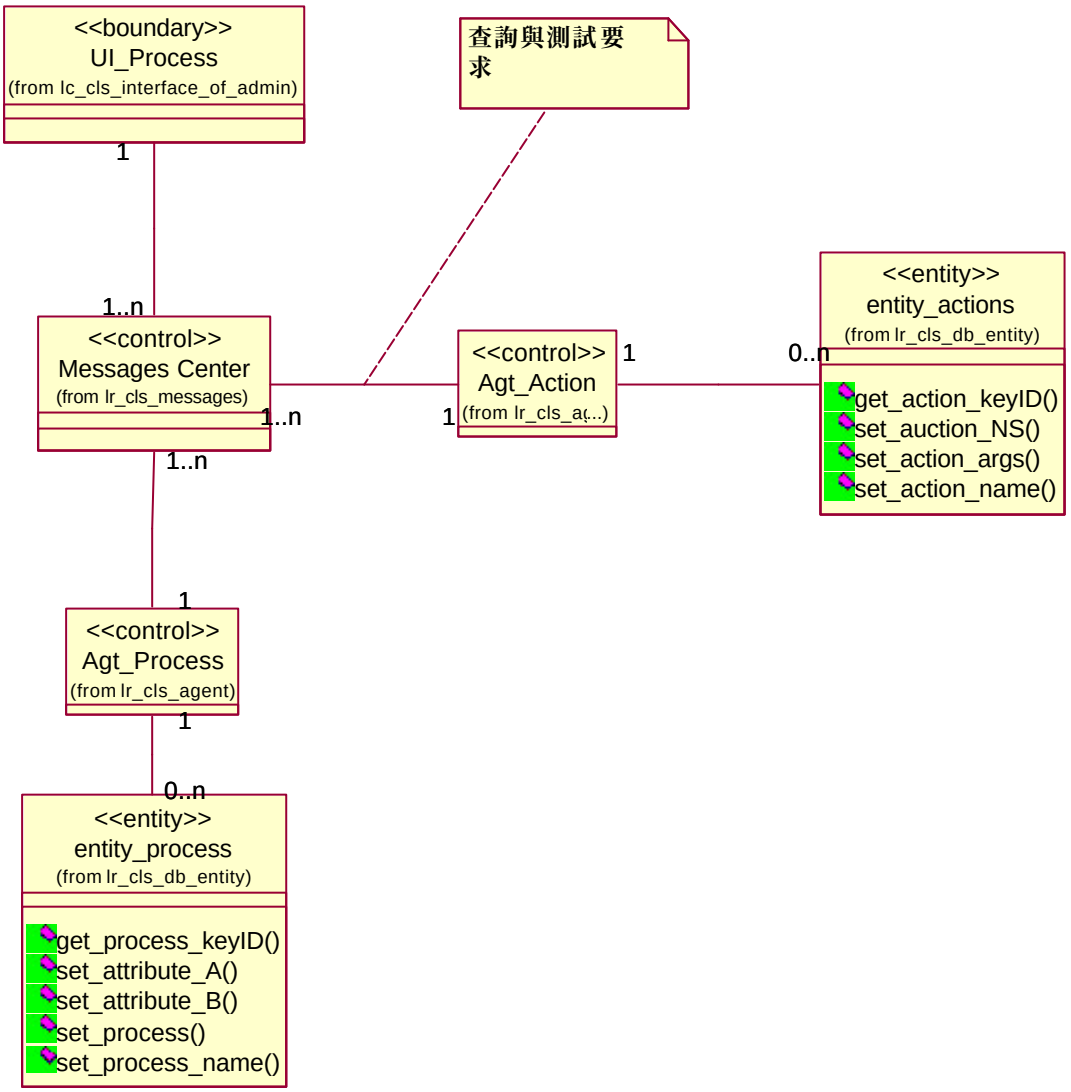


圖 31、程序資訊登錄類別圖

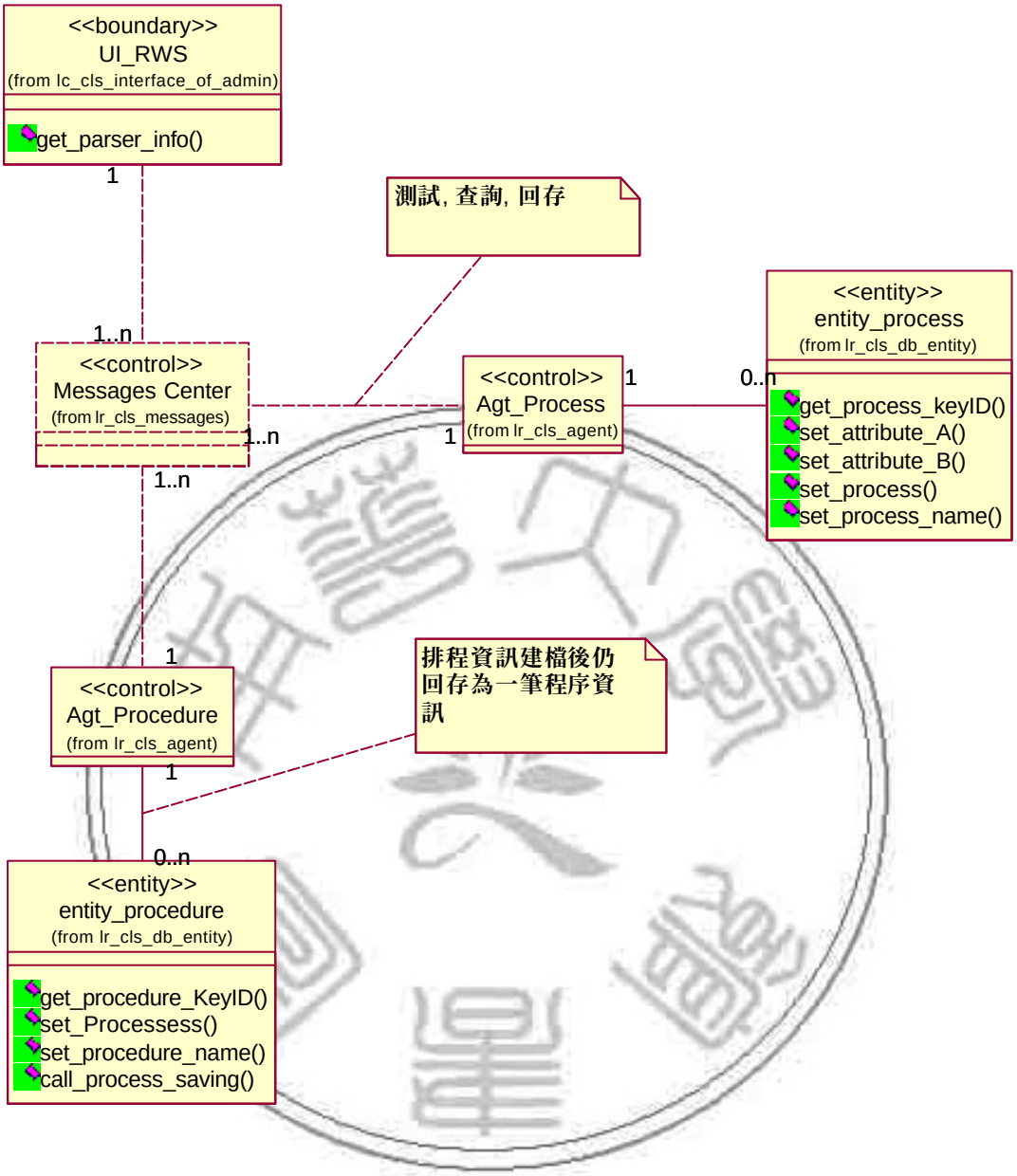


圖 32、排程資訊登錄類別圖

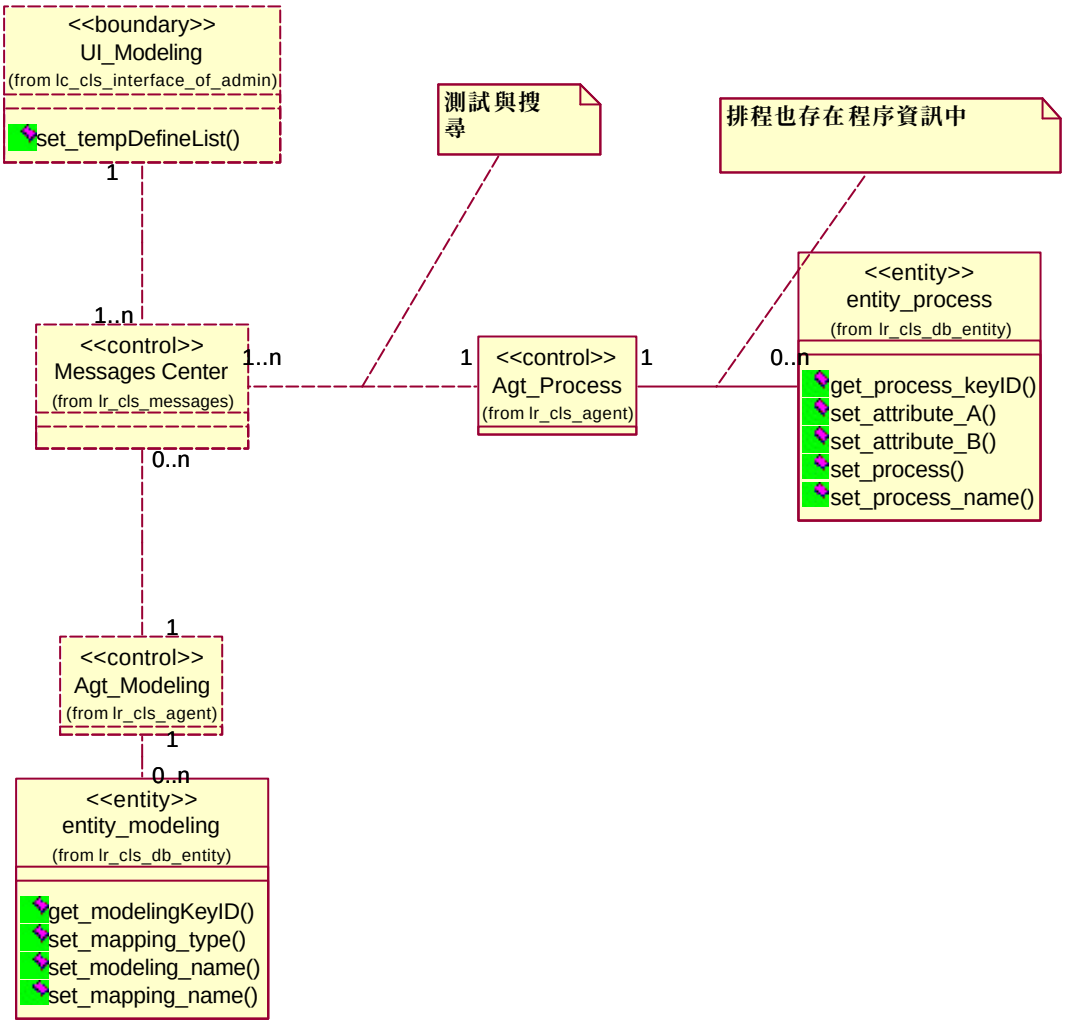


圖 33、模組組成化資訊登錄類別圖

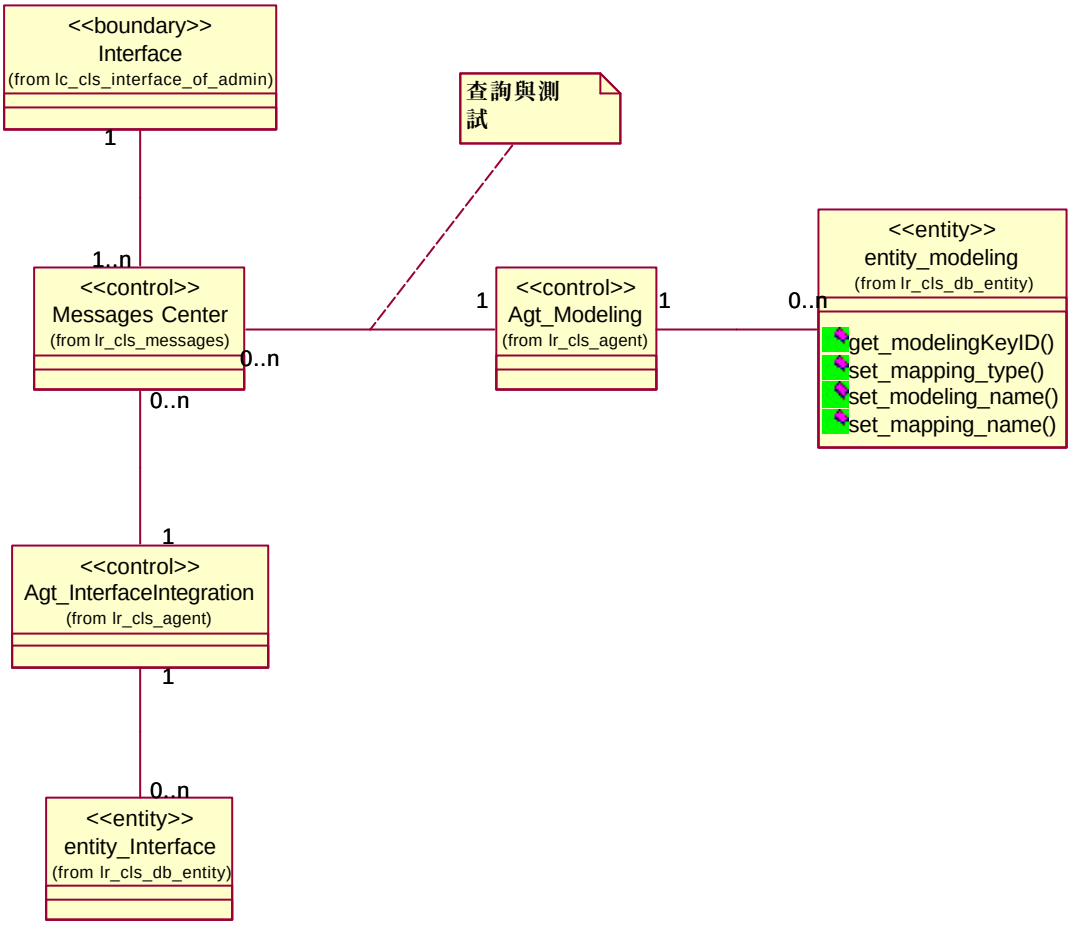


圖 34、介面整合資訊登錄類別圖

第三節　介面對應回溯類別

介面對應回溯機制的類別成員基本上與服務整合機制相同，唯一不同的是其叫用的模式；其中代理人的角色從操作資訊實體類別轉變成從實體類別叫出系統相關紀錄，然後找出任務的下層代理人，再將資訊傳遞給低階的代理人去操作，一層層解析之後；最後成為單一的 SOAP 呼叫程序送出。

以下幾張類別圖即是相關回溯呼叫的類別圖，關於回溯呼叫的循序圖請參考後面附錄。

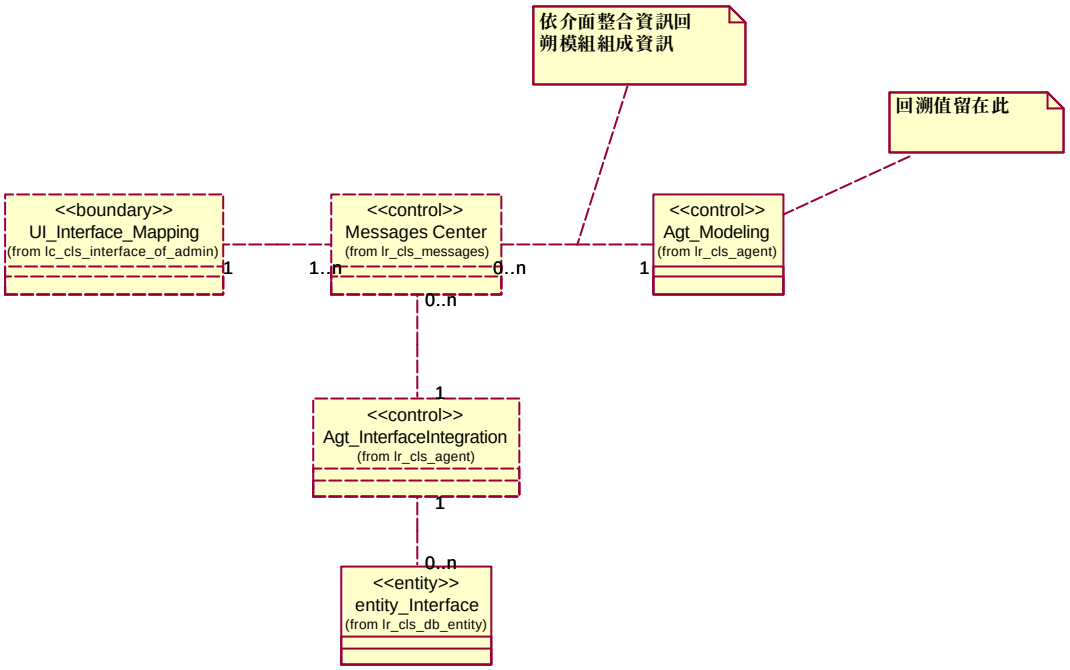


圖 35、介面對應轉換-回朔模組資訊類別圖

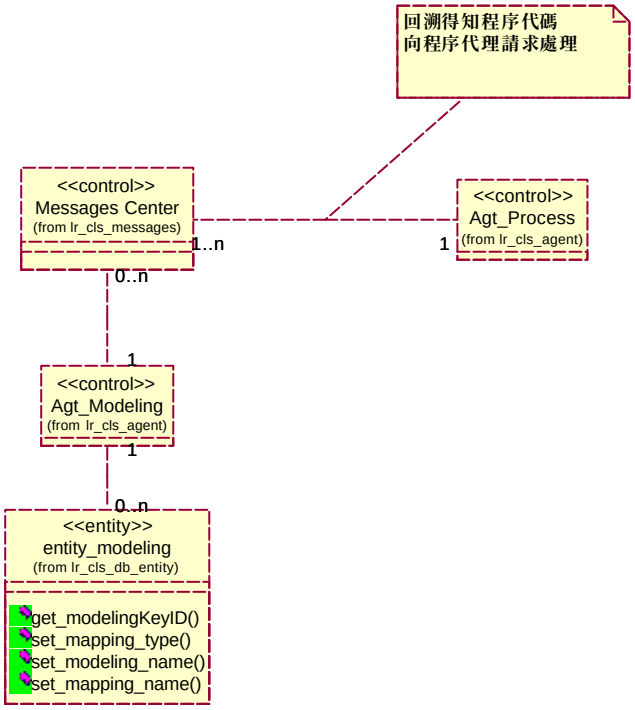


圖 36、模組化組成資訊回溯類別圖

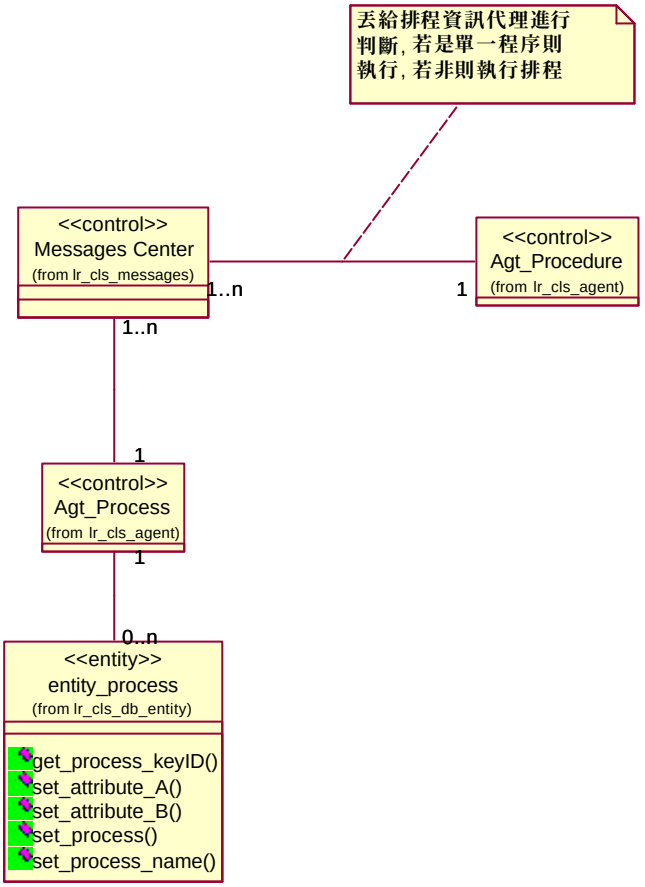


圖 37、程序資訊回溯類別圖

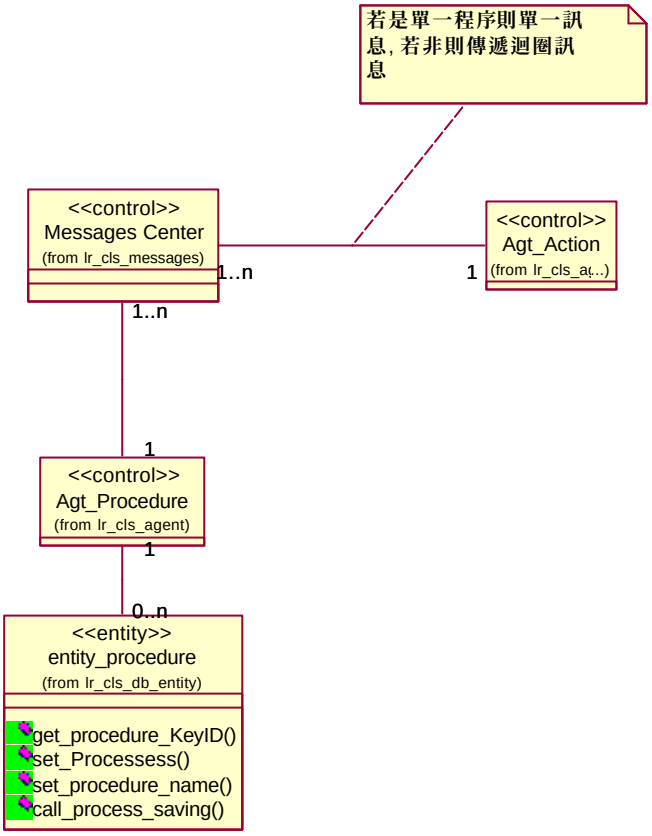


圖 38、排程訊息回溯類別圖

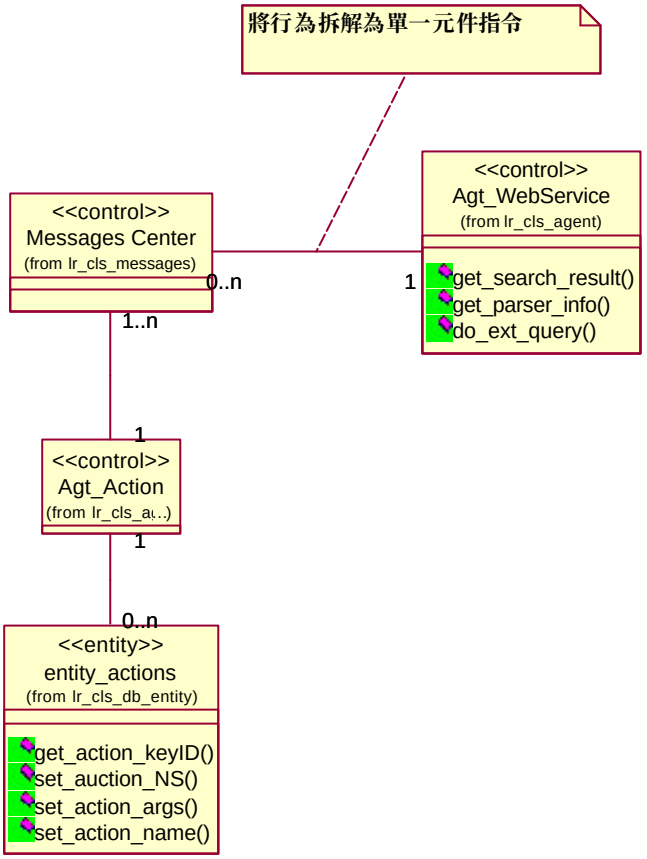


圖 39、服務元件執行回溯類別圖

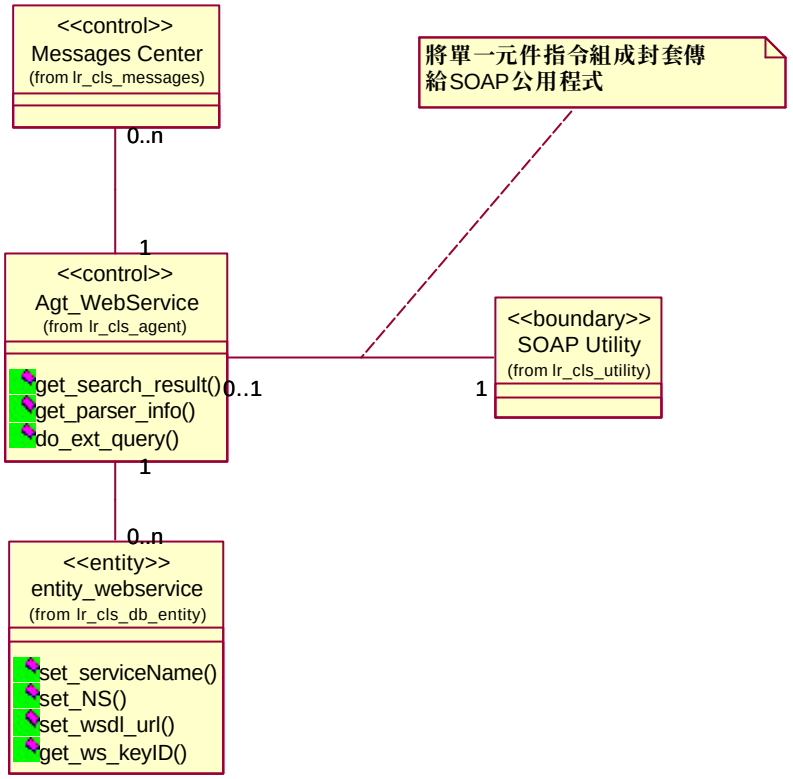


圖 40、單一服務呼叫回溯類別圖

第四節　小結

圖 41 是整個分析完成後的總覽類別圖。

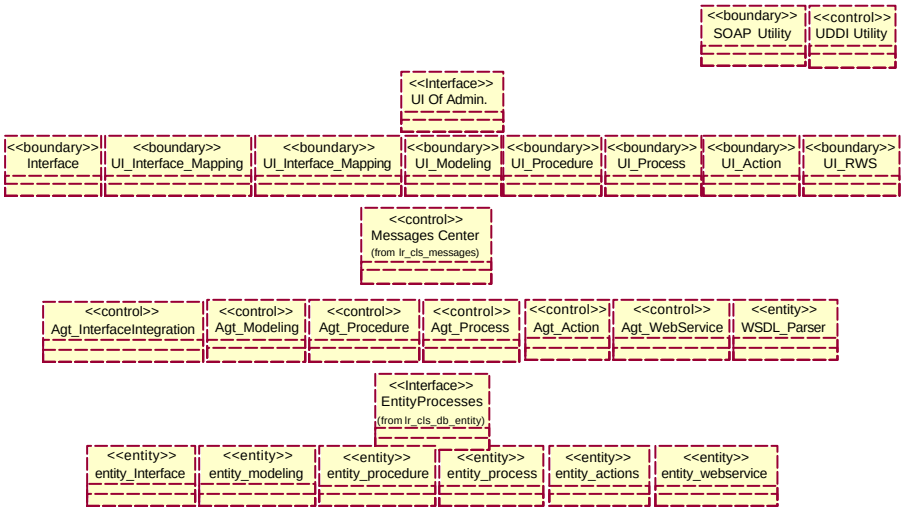


圖 41、類別總覽圖

第五章　系統雛型實作與測試

本章將依照本研究所提架構加以實作並說明所使用的工具與方法，並在第二節中陳述試作一個簡單的應用範例以便驗證本系統架構的效益。

第一節　系統之實作

- **開發環境**

  本研究中的系統架構是利用 Rational Rose 分析工具來進行各項類別元件的開發與設計，而本研究的平台端是以微軟的 Windows Server 和 IIS 伺服器提供網頁服務，後端處理部分則是利用 JDK 1.4.1.02 版中 Servlet 2.3 服務類別元件實作 JSP 頁面；而遠端 Services 端則是使用 Apache Axis 作為 Web Services 的執行環境。

- **軟體需求**

  1. **作業系統：** Windows 2000 Server sp3

     Windows 2000 是微軟公司所出的網路伺服器平台，具有高度穩定性，掛載 JVM 之後仍能夠穩定的執行使用 Java 開發的各類應用系統或程式。

  2. **網頁伺服：** Microsoft IIS 5

     本研究中主要是以 Java Server Page 作為平台操作介面的展現，但仍然使用了 IIS 網頁服務功能將使用者對系統的需求轉送到 JSP 容器的伺服端。

  3. Java 開發環境：Sun JDK 1.4.1

     本研究使用 Sun 公司所推出的 Java 2 Platform, Standard Edition 作為 Java 語言的實作開發環境。

4. JSP 容器：Resin EE 2.1.9

   Java Server Page(JSP)是 servlet 2.3 規範中的實作部份(20)，將使用 Java 語言撰寫的程式轉換成瀏覽的頁面傳回給使用者。

5. Web Services 容器：Apache Axis 1.1 rc2

   將撰寫好的 Web Service 服務掛載在 Apache Axis 伺服器上，用以模擬一般 Web Service 運作情況。

6. 資料庫：MySQL 3.23.56

   MySQL 為開發程式碼的資料庫系統，具有跨平台的特性，將很容易移植到其他執行環境與平台之中。

7. Web Services 開發工具：JBuilder 8 PE 版

   使用 Borland 公司免費提供試用的 JBuilder 8 PE 版進行 Java Bean 與其他系統所需的類別開發，並包裝成 Web Service 格式以供平台測試使用。

8. 網頁編寫： Macromedia Dreamweaver 4

   用以編寫 JSP 網頁與其他版面的設計與網站整體的規劃。

- **硬體需求**

  為了進行整合測試，我們使用了實驗室中三台 Pentium III 與一台 Pentium 4 個人電腦，其中以一台擔任主系統平台，兩台 Web Services 服務伺服器，而一台於架構建立時擔任進行開發與設計的工作平台；於雛型測試時則作為使用者端進行功能性的操作與管理用。而下圖則是我們預期在進行實體部署的時候應該會有的企業內系統環境示意：

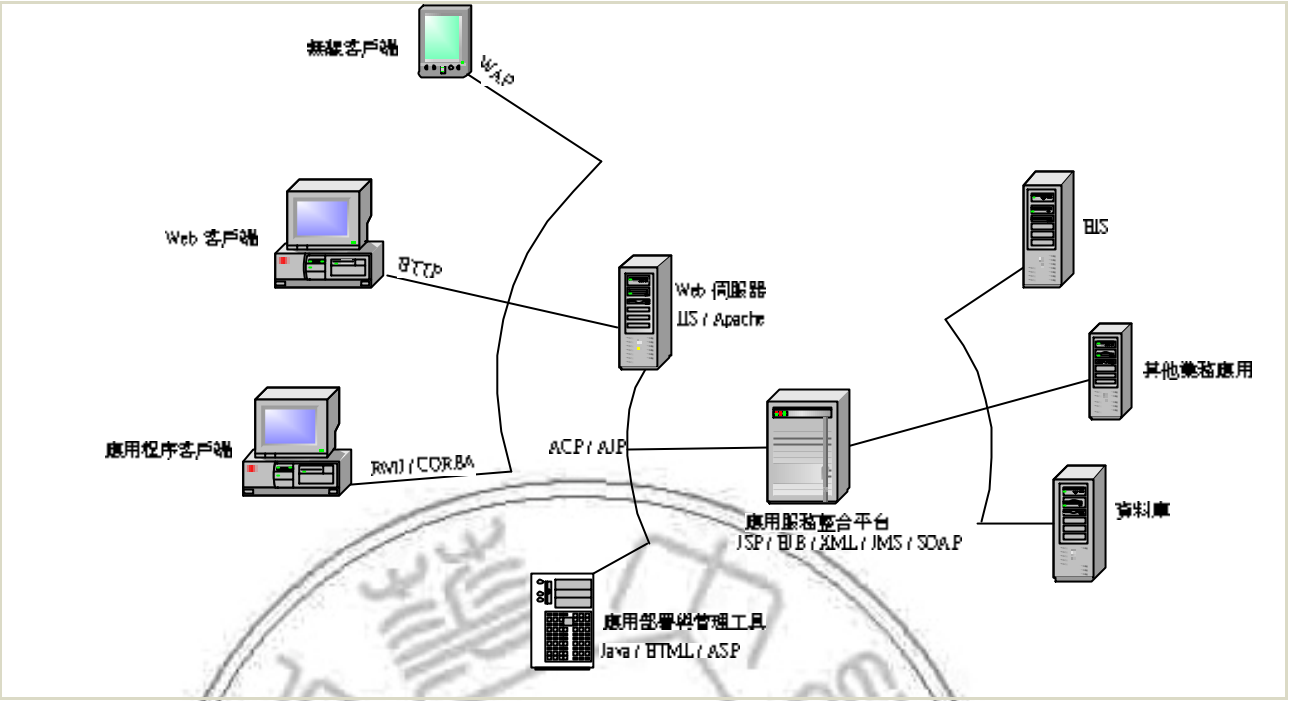


圖 42、應用服務整合平台部署圖

- **實作步驟**

  由於本系統架構必須要先進行服務的整合，才能針對服務進行回溯的叫用，因此我們首先要先從查詢遠端的 Web Services **服務元件所在，然後建立起系統所需各項紀錄，再產生一個對應介面，讓使用者能夠透過介面進行操作。歸納為以下幾點：**

  - **Web Service 紀錄格式定義與相關操作程序撰寫**

    在此步驟中，首先定義 Web Service **的描述語言**(WSDL)**對應到資料庫的儲存格式，並辨識出所屬的動作與屬性參數等資訊，分別開發紀錄基本對應資訊的函式，以及未來要叫用此服務時所需的各項必要資訊。**

  - **系統變數紀錄格式與相關操作程序撰寫**

    在此步驟定義出系統變數的格式，用來作為將來仲介程序間傳遞值時的暫存變數初始化藍圖，並撰寫相關管理函式供管理者新增，以及提供變數輸入定義功能，供管理者能

夠定義變數輸入介面的對應格式供後續的介面對應模組產生介面時使用。

- **程序紀錄定義與管理程序撰寫**

  撰寫管理程序，其要能夠對應到先前建置的 Web Service 紀錄格式, 並依程序的屬性或參數的需要而給予不同的變數對應。

- **程序排程紀錄定義與運作介面設計**

  定義程序排程專用的紀錄格式, 期能夠紀錄程序之間的循序關係，並將排程本身也定義為程序的一種，以便後續的處理調用。

- **模組紀錄設計與管理程序撰寫**

  提供使用者程序清單的檢視功能, 並允許使用者對於相同目標卻可能分散在不同服務的程序紀錄彙整新增在模組紀錄中, 當檢視模組紀錄時將能夠看到所屬的程序紀錄集合。

- **定義介面與模組對應關係紀錄和介面處理設計**

  設計一彈性化的介面對應程式，讓使用者依照自己的需求，自由選擇直接排列式、左右排列式、或者是自訂版面的方式，並引入欲在此介面操作的模組方法，再針對該方法於介面的呈現方式提供可以進行設計的機制, 在設計完成時，希望能夠依此資訊動態產生一個使用介面。

- **介面操作回溯呼叫 Web Services 服務相關程序撰寫**

  透過介面所提供的特殊格式，設計能夠接收此訊息的程序, 依介面傳遞的各種操作進行另一步的程序叫用與變數

傳遞，並依模組資訊反查回朔其所屬的 Web Service 資訊，再撰寫一支能動態產生 SOAP 封套的 Client 程式對遠端的 Web Service 進行呼叫。

- **使用者介面雛型與功能說明**

  圖 43 為平台雛型的開始畫面，以下將為本系統的實作雛型畫面一一進行解說。

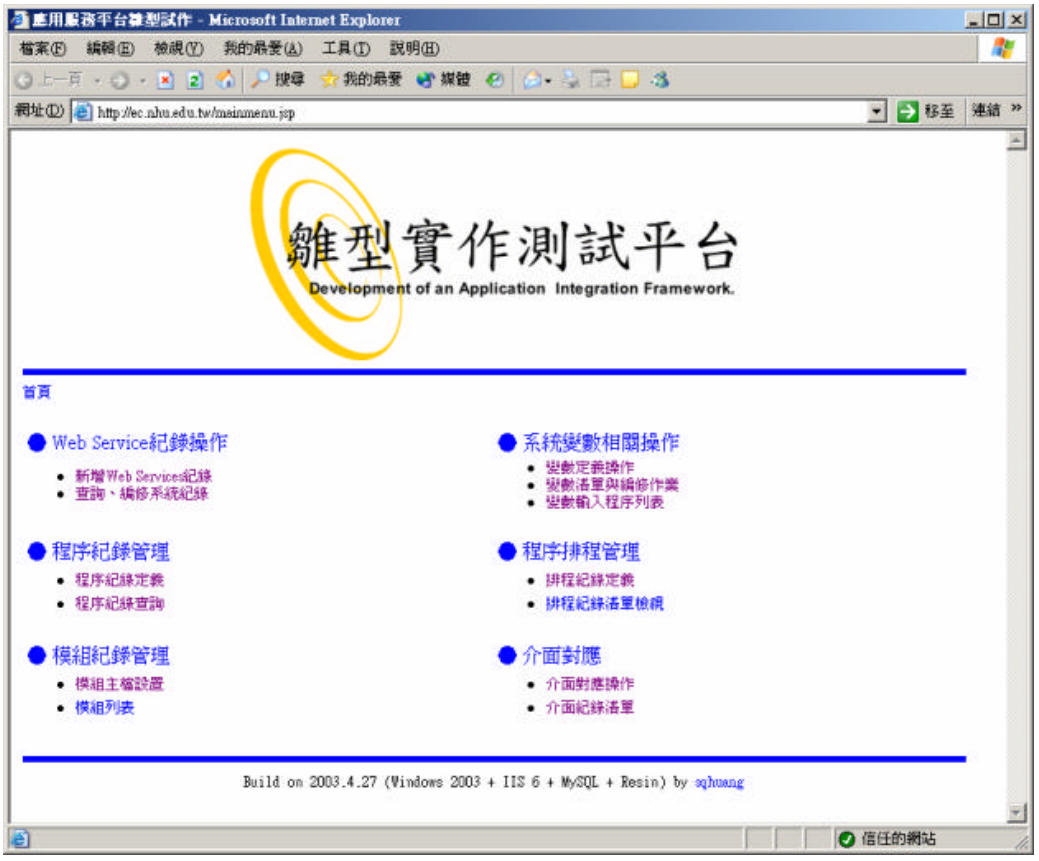


圖 43、雛形實作測試平台主畫面

- Web Service **紀錄操作**

  主要在於透過解析遠端 Web Services 的 WSDL 描述，將服務中的各項資訊建置於平台中的資料庫，如此可平台將會獲得如何叫用該服務的方法，並且供之後的其他操作使用。(圖 44)

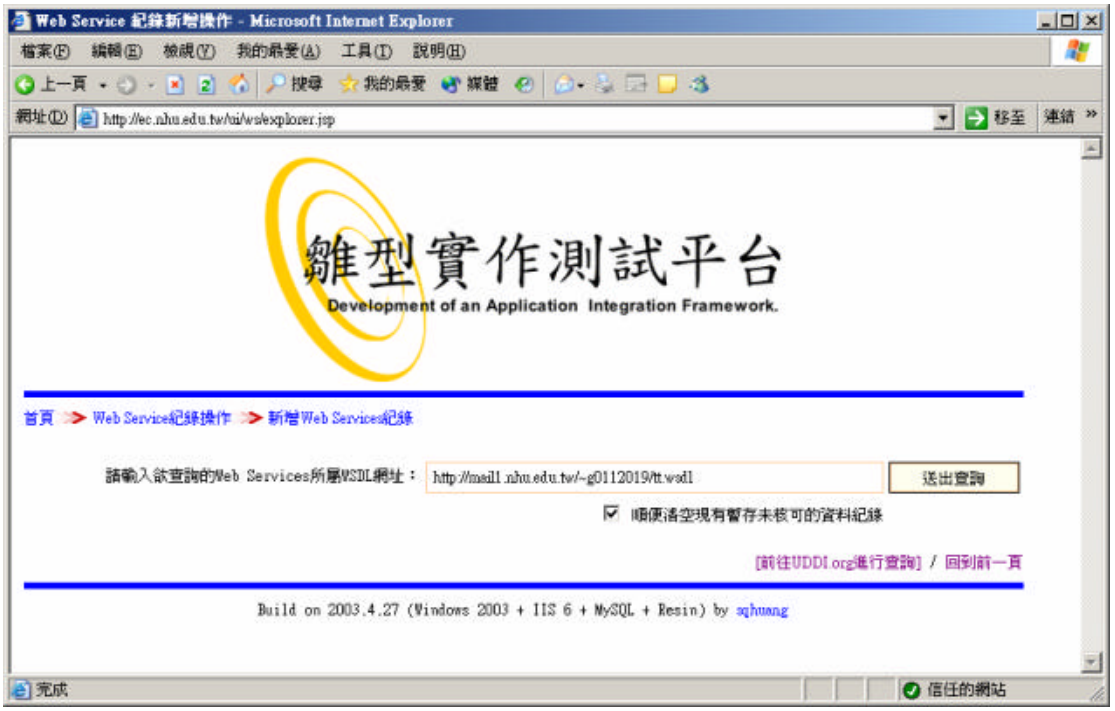


圖 44、登錄 WSDL 畫面

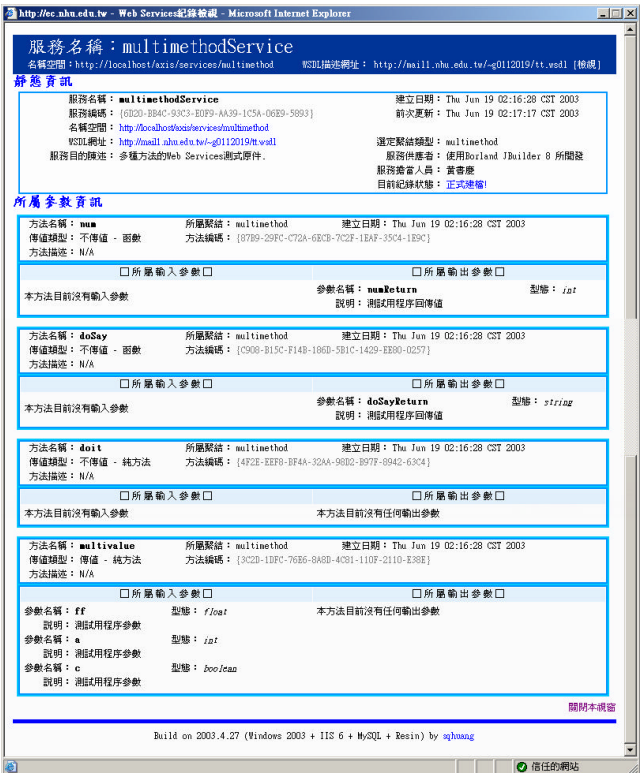


圖 45、Web Service 經系統解析後畫面

- **系統變數相關操作**

系統變數的定義是為了提供當程序跟程序之間需要傳遞

資訊時的過渡暫存區, 其中包含了使用者畫面中輸入的動作，並將其傳遞給所需要的程序，另外還包括銜接函數型程序的回應工作。

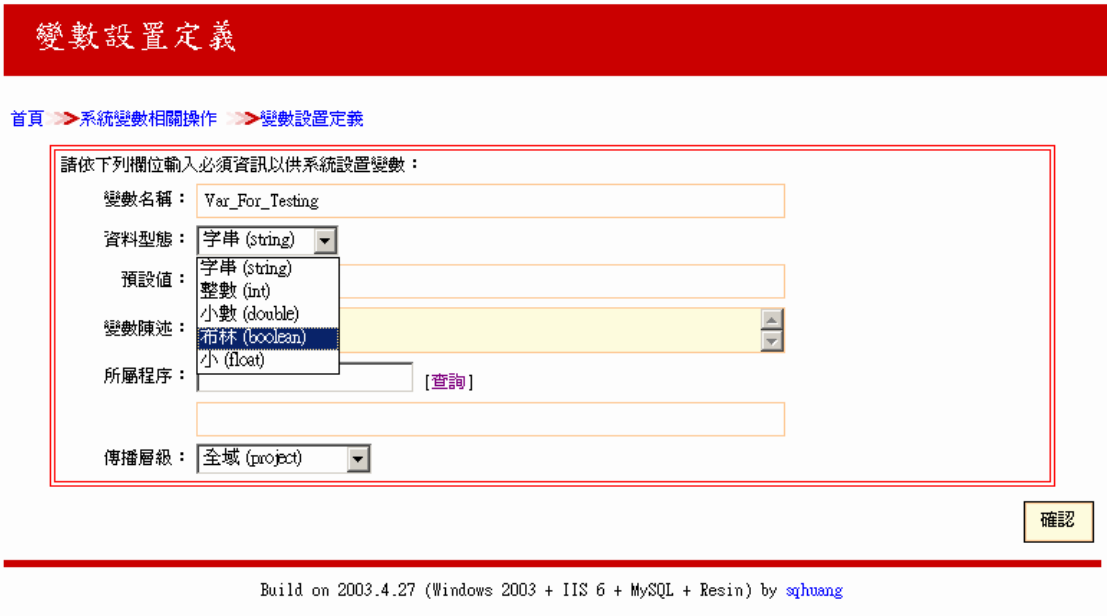


圖 46、系統變數設置畫面

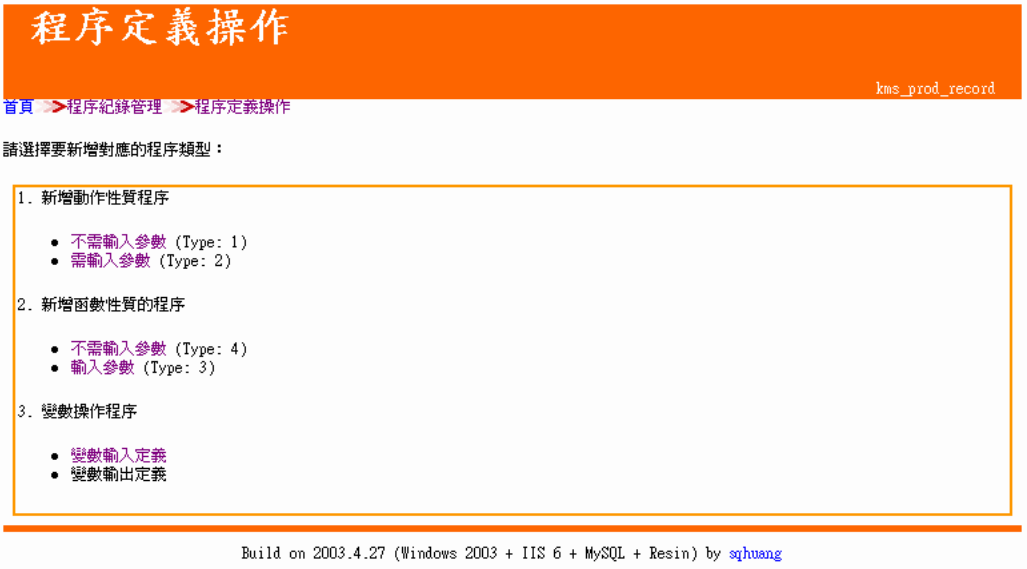


圖 47、程序定義主畫面

- **程序紀錄定義與管理**

  以 Web Service 的描述資訊作為藍本，平台定義自己的程序，虛擬的觀點可以將它視為平台所提供的服務，而實際

上在架構中的運作，其都屬於對應的 Web Services 中的一個方法，由平台送出 SOAP 封套進行調用。

圖 48、程序定義畫面

- **排程紀錄定義與管理**

  以程序紀錄為基礎的程序排程，其主要的任務是將若干程序組合成一個循序的排程紀錄，再紀錄為一筆可供使用的程序紀錄，如此當重複的動作經常的被調用時，將可以省去許多的登錄跟對應手續。

圖 49、排程定義畫面

- **模組管理功能**

  在本研究中定義模組是由一群相同任務的程序所組成，模組紀錄中包含了各種程序與變數，分散在不同的系統上面分別獨立的運作所屬的少部分特定功能，但是透過本平台的模組建置進行整合過後，將能夠有統一方便的模式去進行操作與使用。

圖 50、模組定義主畫面

圖 51、模組定義畫面

- **定義使用介面與模組對應關係**

  希望透過介面的定義, 提供使用者一個直接叫用平台中所定義的方法的管道, 並且能夠透過介面輸入程序在運作時所需的參數，進而再動態產生 SOAP **封套，呼叫** SOAP Client **去聯繫遠端的** Web Service。

  在定義之初可以要求使用者設置介面所需的相關資訊, 其中包含(1)**頁面的呈現方式，可由系統依選擇自動產生介面、**(2)**頁面的下一頁，定義頁面的參數要傳遞到那一個頁面、**(3)**頁面標題、**(4)**頁面基本陳述、**(5)**使用者自訂標題與版面排版, 提供一個簡易的自訂標題與排版機制, 供**

使用者安排畫面與操作介面的設置, 再由系統每次叫用時依設定產生頁面。〔圖 53, 54, 55)

圖 52、介面定義畫面

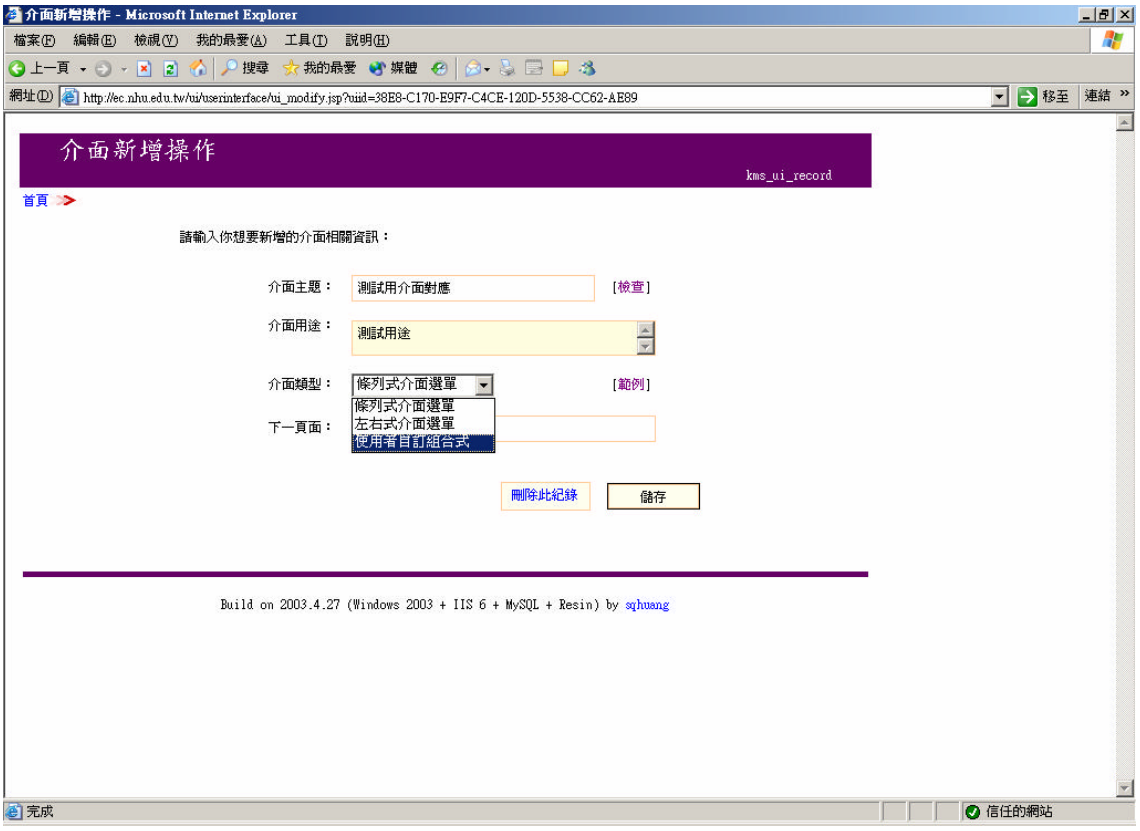


圖 53、介面對應新增畫面

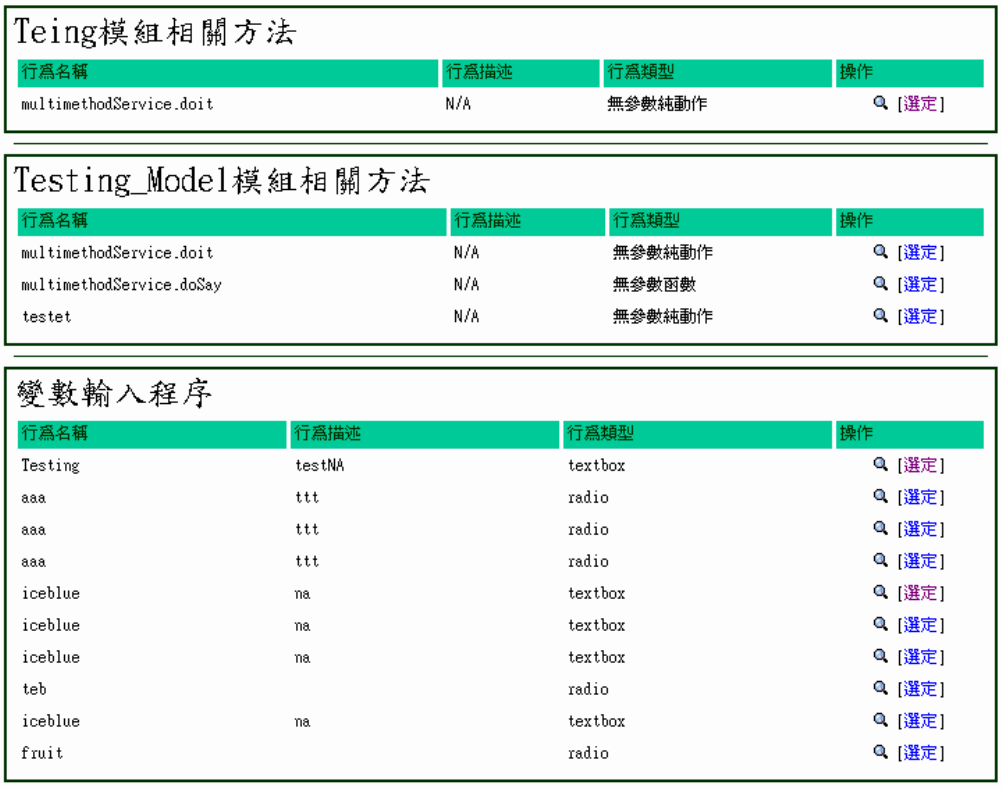


圖 54、介面對應程序清單畫面

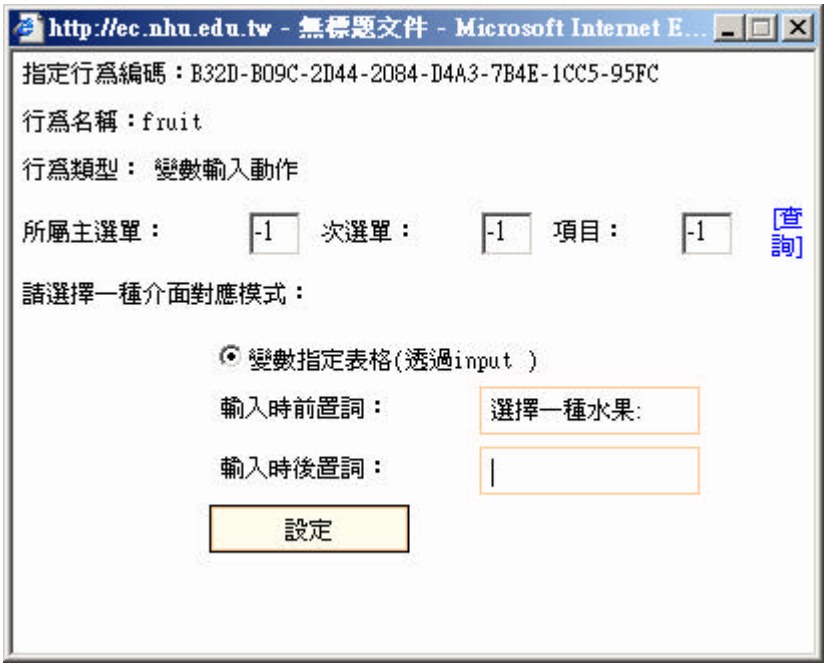


圖 55、介面對應程序相關設定畫面

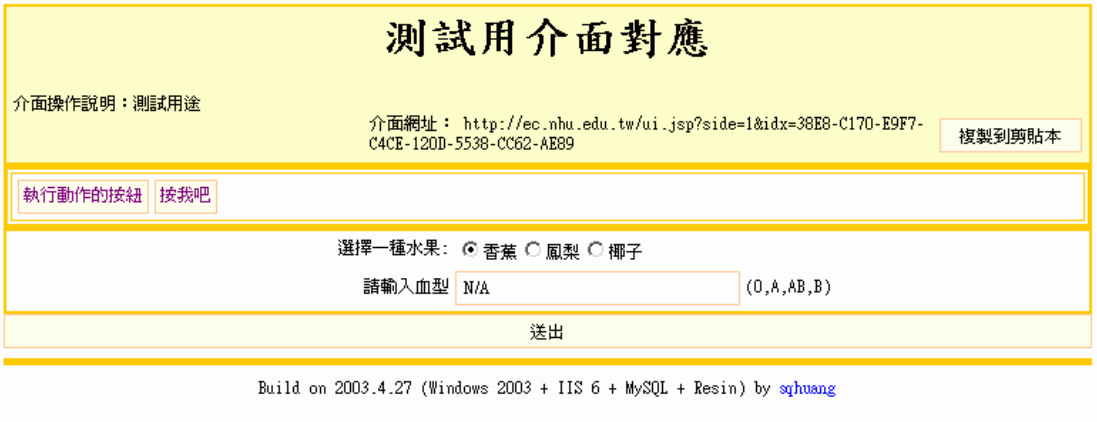


圖 56、介面自動產生-條列式

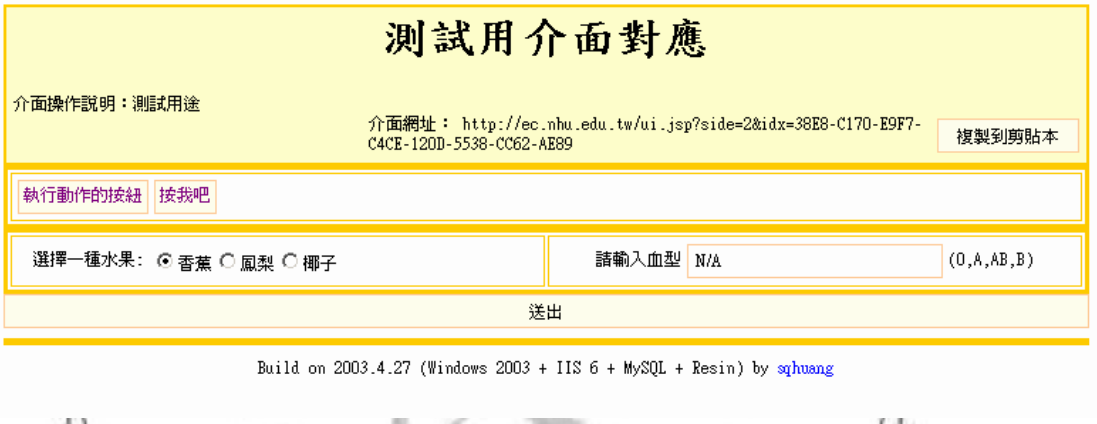


圖 57、介面自動產生-左右並排式

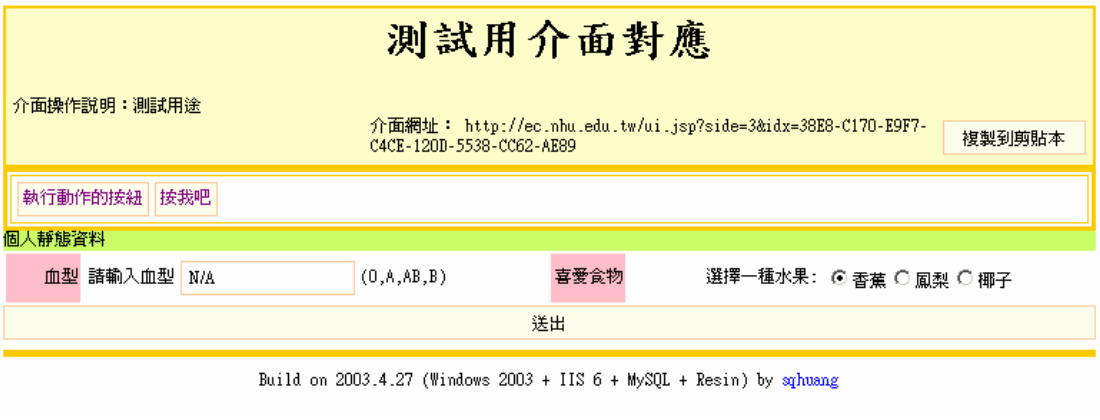


圖 58、介面自動產生-使用者自訂式

第二節　系統之整合與測試

- **測試環境**

  本研究的測試環境是以先前圖 14 作為模擬測試的硬體配置，共分為 Server 端, Service 端及 Client 端, Server 端採用 Windows 2000 + IIS 5+ Resin 2.19 作為掛載平台介面伺服環境，而 Service 端則由兩台不同的主機掛載 Resin2.19 + Apache Axis 引擎作為 Web service 的服務提供器, client 則只需要一般個人電腦，能夠上網且使用 IE 5.0 以上作為網頁瀏覽之用。

- **測試步驟**

  系統的操作流程主要是先從遠端服務的探索並登錄，再進行程序的整合，建立模組，產生操作介面，再來針對服務進行回溯的叫用，因此我們進行測試的時後必先要先在 Services 端建立起若干包裝為 Web Services 的服務，再查詢遠端的 Web Services 服務元件資訊並予以登錄，然後建立起系統所需各項程序紀錄，對應介面的產生，再來依造系統產生的介面進行操作，最後再檢查透過本平台雛型所叫用 Web Service 達到的結果跟使用一般程式語言開發方式編碼的方式所產生的結果，是否有所出入。歸納為以下幾點尚需要進行測試與檢查，茲將檢查時考量跟其他發現的問題陳述如下：

  - **Web Services 的製作與掛載**

    利用 JBuilder 開發 Java 為基礎的類別元件，再將其以 WSDL 包裝後掛載於 Services 端的 Apache Axis 引擎中提供服務。

  - **Web Service 探索與紀錄操作**

在 Server 端針對前一步驟的服務網址進行探索與資訊的解析，檢查相關資訊是否都已經完整的對應到系統資料庫中，不同平台所開發的 Web Service 是否會有解析上的問題值得注意？

- **系統變數設置**

  設置系統變數，並於系統中安插若干標籤，於各個關鍵點追蹤系統變數遭到程序所呼叫或是傳遞回傳值時是否正確無誤？

- **程序紀錄定義與管理操作測試**

  測試當程序紀錄建檔時是否已經儲存足夠的資訊供後續的程序叫用，以及程序紀錄的叫用是否能夠不受 Web Service 格式與來源的限制？

- **程序排程定義與管理操作測試**

  檢查當程序被登錄於排程中後，是否能夠在排程叫用的時候依序沒有錯誤的執行完畢，而當錯誤發生時，同一排程中的其他已經執行程序是否需要有其他還原的動作？

- **模組紀錄管理操作測試**

  檢查模組的建置是否能夠順利完成，而當模組新增程序的時候，程序本身是否會侷限於單一模組，亦即是程序的運作範圍是否應該受到限制？

- **定義使用介面與模組對應關係**

  執行多次且不同的介面設置以測試介面的產生是否會因為某些特殊的組合或是使用者的定義不當造成介面的異常，並檢查透過介面所叫用或是傳遞的參數是否都能夠有

效的運作？

- **介面操作回溯呼叫 Web Services 服務**

  檢查透過本平台介面所叫用遠端 Web Services **而得出的結果，與使用一般程式語言開發工具所編碼的方式調用** Web Service **結果是否一致**, **若為一致**, **則相比較之下本平台所提供的便利性與優缺點為何？**

第六章、結論與未來發展方向

第一節　結論

根據 Gartner 的研究報告指出，到了 2007 年，約有 75 % 的企業將建立多種網路服務機制，其中以網路服務（Web Service）、或是網站、網路服務兩者混合服務為主, 這顯示未來「網路服務」市場一片大好。IDC 認為不僅是網路服務將形成主流標準，網路服務相關專業服務未來也將大行其道，IDC 預測美國「網路服務」（Web Service）建置專案之專業服務，將從 2002 年 3.29 億美元每年以 116 % 驚人成長率成長，至 2006 年將達 71 億美金。 IDC 認為網路服務之專業服務，種類包括：顧問、開發應用服務、系統整合服務等，這些服務將由外部第三者提供。由於建置「網路服務」所牽扯的議題相當複雜，預料顧問服務（Consulting）、及系統整合對於有意採用網路服務的企業來說，是相當重要的。

本研究主要提供一個能夠讓管理者在不需額外撰寫程式碼的前提下，透過操作介面而輕易整合企業內或者是跨企業間的各項應用服務，由於本研究系統架構的彈性；其中的應用範圍則不只侷限於單一個資訊管理模式；本系統架構也蘊含了工作流程的概念，因此在實際的應用上；也可能用於處理不同組織之間的任務協調過程。

第二節　未來發展方向

目前 Web Services 尚有許多附屬的技術尚未通過成為正式業界的技術標準，諸如關於服務品質陳述的 WSEL，服務流程概念的 WSFL 等，待業界標準發展成熟後若能在將上述相關技術的概念加以改良導入；將

會更加完整。另一方面，本研究並未針對現有技術與 Web Services 為基礎的系統架構作效率比較；本研究奠基於 XML 化訊息為基礎的技術，諸如 SOAP、WSDL 等；未來 XML 可能會朝向內嵌式發展，進而提供一個更快速的訊息傳遞環境，則系統的訊息傳遞效率將會更好。

另外，在進行測試的時候我們發現，雖然以 XML 為基礎格式編碼的 Web Service 本質上是用來解決異質性系統的互相溝通問題，但是在特定平台所開發出來的 Web Service 服務，其 WSDL 對於 types 部分的規格與定義，卻仍出現明顯與其他各家不同的情況，此等試圖破壞標準的舉動，在我們進行系統開發的時候造成很大的困擾，由於 Web Service 仍是方興未艾的一門技術，未來預期各項相關標準將會更加完備。屆時將仍更容易開發出真正相容於各類作也平台系統。

參考文獻

中文部分

[1] 王昌斌、黃書慶、吳俊毅、胡莉玲 「利用 Web Service 技術架構知識管理系統-以企業應用整合觀點」，電子商務經營管理研討會，逢甲大學，民國九十年。

[2] 胡百敬，簡易物件存取協定 Simple Object Access Protocol(SOAP)，恆逸資訊，Sep. 2000。

[3] 楊正甫，物件導向分析與設計，松崗電腦股份有限公司，2000 年一月。

[4] 鄭淑芬，Biztalk Framework 電子商務架構與 XML，恆逸資訊，Sep. 2000。

[5] 林弘之，Web Services 實作-使用 SOAP ToolKit 與 Visual Studio.NET，文魁資訊，2002 九月。

[6] 張裕益，UML 理論與實作--個案討論與經驗分享，博碩文化，2002 年二月。

[7] 徐堯，學 UML 的第一本書，博碩文化，2003 年二月。

[8] 葉斯建，「發展以網路服務為基礎之電子診斷資訊整合架構」，成功大學製造工程研究所碩士論文，民國 91 年。

英文部分

[9] Chris Fisher, “Unlocking The Role Of Middleware In e-Commerce”, EAI Journal, 2000.

[10] H. Balen, Distributed Object Architectures with CORBA, SIGS BOOKS, May 2000.

[11] Linthicum D. S. ,Enterprise Application Integration, Addison-Wesley, November, 1999

[12] M. Fowler, UML Distilled Second Edition, Addison Wesley, 2000.

[13] R. Sessions, “COM and DCOM：Microsoft’s Vision for Distributed Objects”, JW, June 1997.

[14] R. Cattell, J. Inscore, and E. Pa,”J2EE Technology in Practice：business

Applications with the Java”, AW, June 2001.

[15] R. Orfali, D. harkey, and J. Edwards,The essential distributed Objects Survival Guide, John Willy & Sons, 1996.

[16] Rick Leander, Building Application Servers, SIGS books, 2000.

[17] Robert Orfali, Dan Harkey, Client/Server Programming with Java and CORBA, John Wiley & Sons, Inc, 1997.

[18] Steve Graham, Simeon Simeonov, Toufic Boubez, Doug Davis, Glen Daniels, Yuichi Nakamura, Ryo Neyama, Building Web Services With Java, SAMs, 2002.

[19] S.Narayanan and Junhe Liu, Enterprise Java Developer's Guide, McGraw-Hill companies, Inc. 1999.

[20] Duane K. Fields, Mark A. Kolb, Shann Bayern, Web Development with JavaServer Pages, 2nd Edition, Manning Publications Co. Nov, 2001.

網站相關論壇與技術標準部分

[21] **林玉凡** “**企業應用程式整合**(EAI)**產品之探討**(**上**)”，http://www.find.org.tw/trend/disp.asp?trend_id=1126。

[22] **林玉凡** “**企業應用程式整合**(EAI)**產品之探討**(**下**)”，http://www.find.org.tw/trend/disp.asp?trend_id=1127。

[23] **柴曉路** “UDDI **服務實施的體系架構**”，IBM developerWorks; http://www.uddi-china.org/research/paper/uddi/uddi02.html, 2001 **年** 5 **月** 30 **日**。

[24] **鍾翠玲** “**輕便省力的另類方法**-Web Services”，http://taiwan.cnet.com/enterprise/technology/story/0,2000040497,20052781,00.htm，2000 **年**。

[25] **台灣資訊網**，http://www.xml.org.tw。

[26] W3C, “SOAP Specification”，http://www.w3c.org/tr/soap。

[27] ---,“WSDL Specification”，http://www.w3c.org/tr/wsdl。

[28] UDDI Specification，http://www.uddi.org。

[29] CORBA/IIOP Specification，http://www.omg.org/corba/corbiiop.htm。

[30] W3C,”Extensible Markup Language (XML)”，http://www.w3.org/XML/。

[31] ---,”XML Protocol Working Group”, http://www.w3.org/2000/xp/Group。

[32] IBM,“Web Services Zone”，http://www-106.ibm.com/developerworks/webservices/。

[33] Markus Horstmann and Mary Kirtland,“DCOM Architecture”, MSDN, July 23, 1997,http://msdn.microsoft.com/library/en-us/dndcom/html/msdn_dcomarch.asp?frame=true。

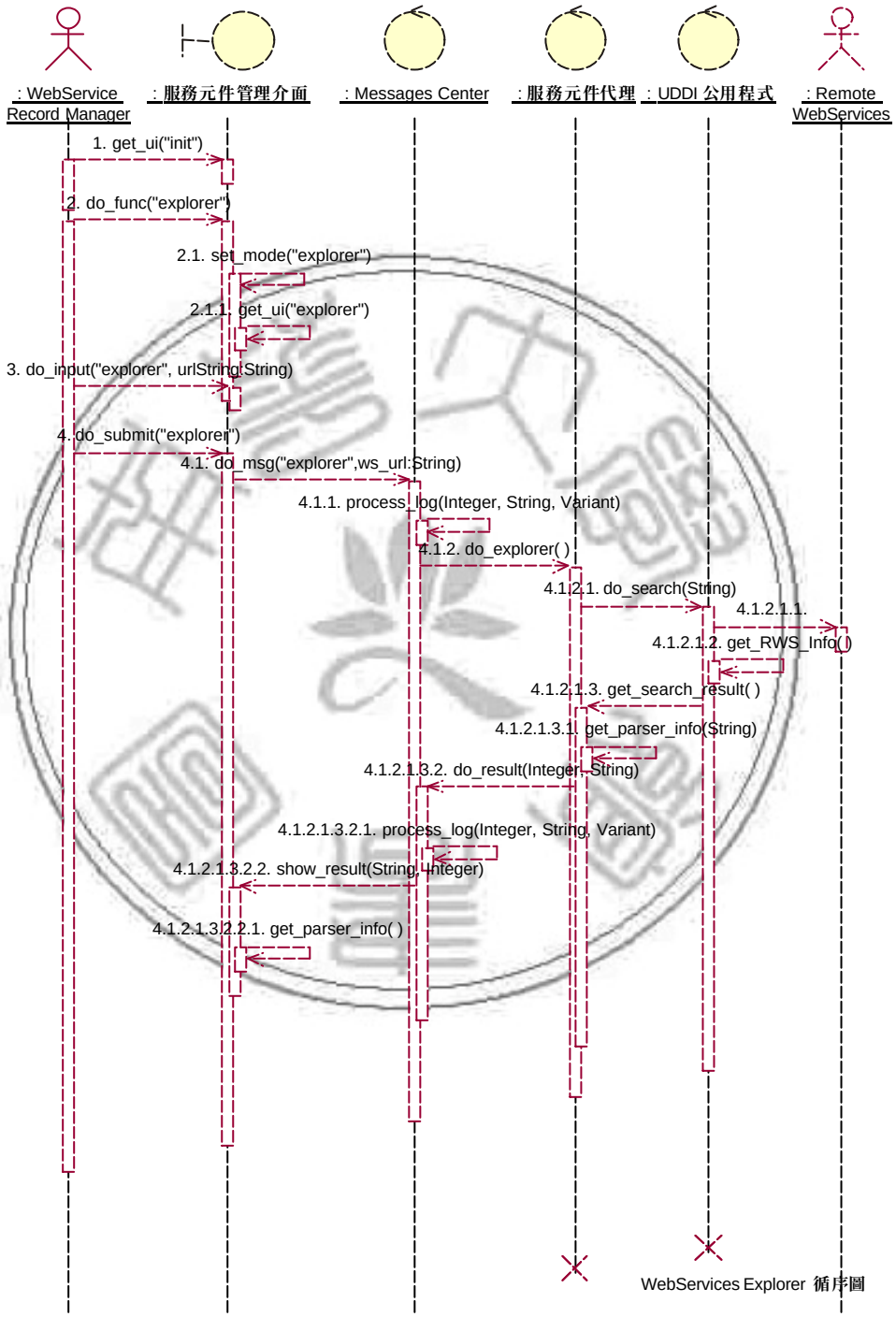
[34] Ilidio Ferreira and Neil Ward-Dutton, "Ovum Evaluates: Enterprise Application Integration", http://www.ovum.com/go/content/005004.htm。

[35] Microsoft, “XML Web Services Developer Center Home”，http://msdn.microsoft.com/webservices/default.aspx。

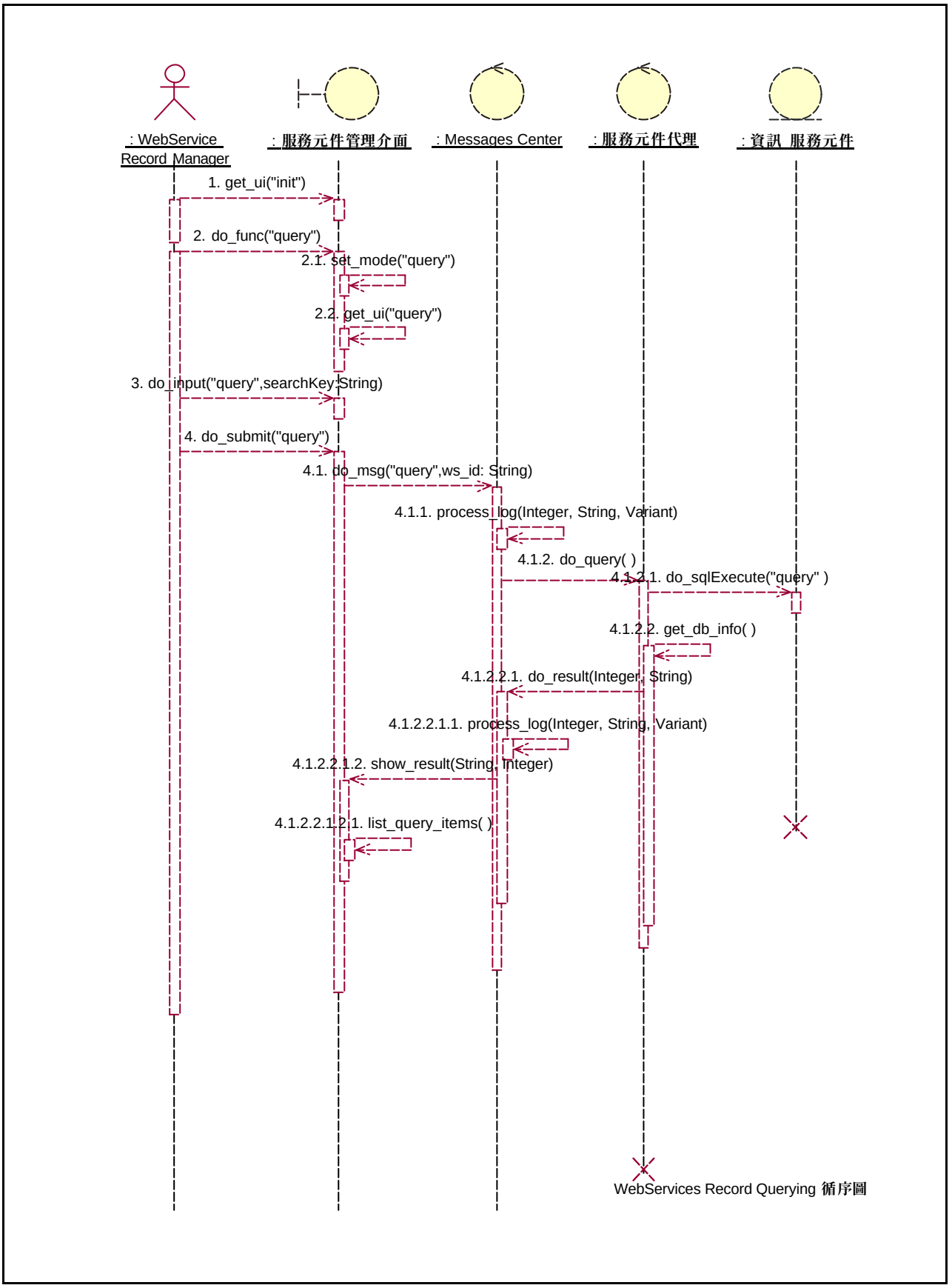
[36] Doug Tidwell, “Web Services- The Web’s next Revolution”，http://ibm.com/developerWorks。

[37] Heather Kreger ,“Web Services Conceptual Architecture”, IBM CHINA , May.2001. http://www-900.ibm.com/developerWorks/cn/webservices/ws-wsca/part1/index.shtml。

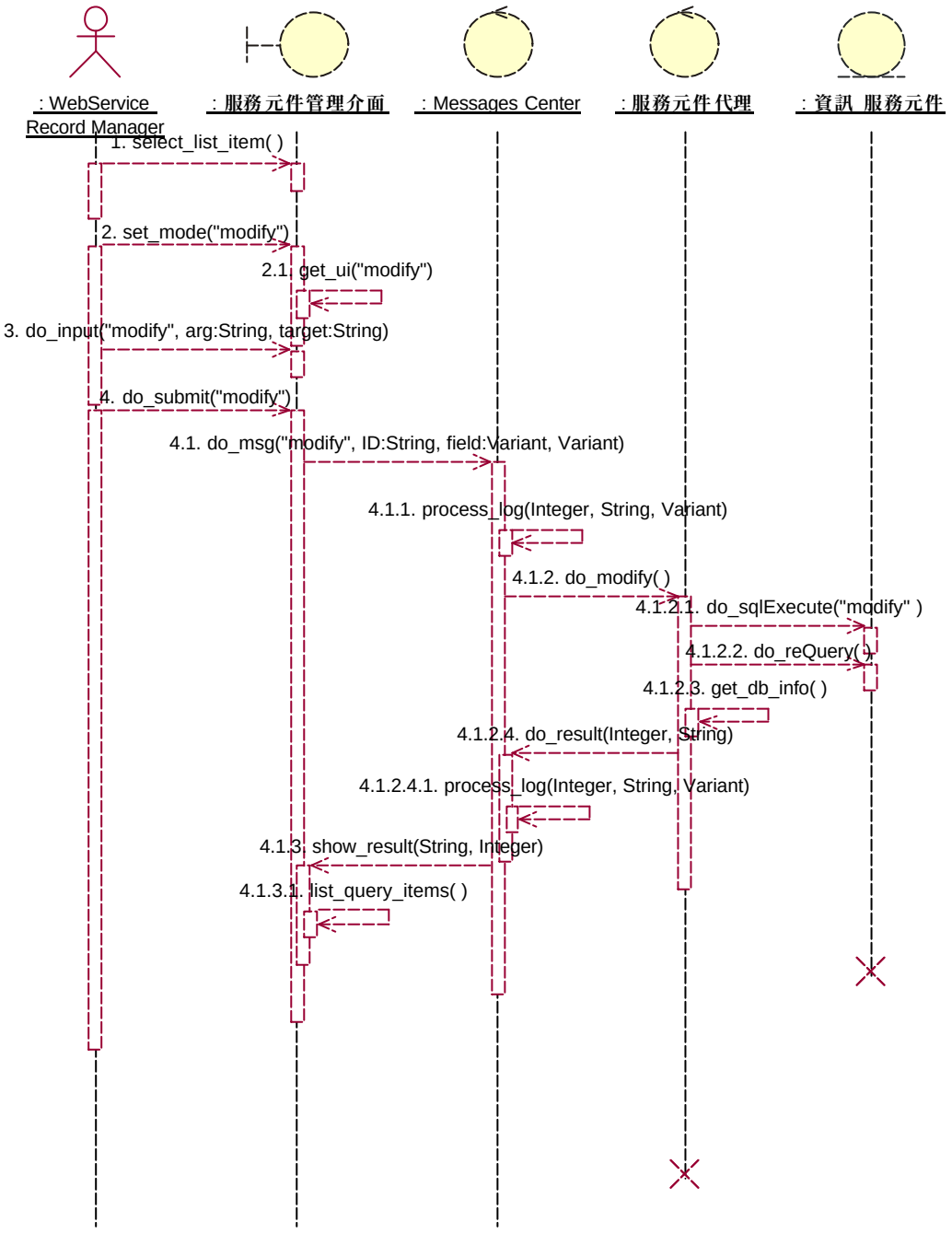
附錄一：Web Services 登錄相關類別循序圖



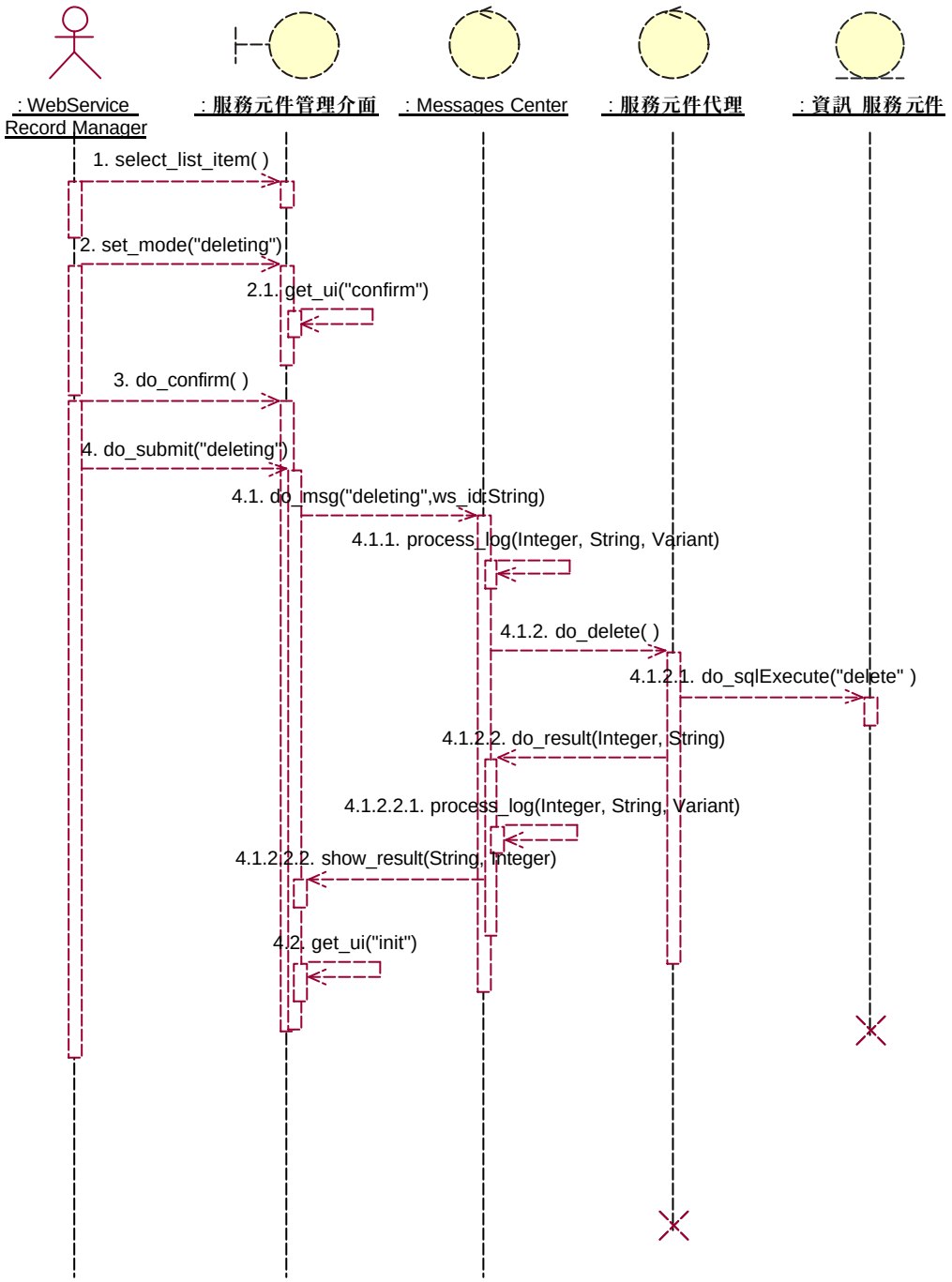
WebServices Explorer 循序圖



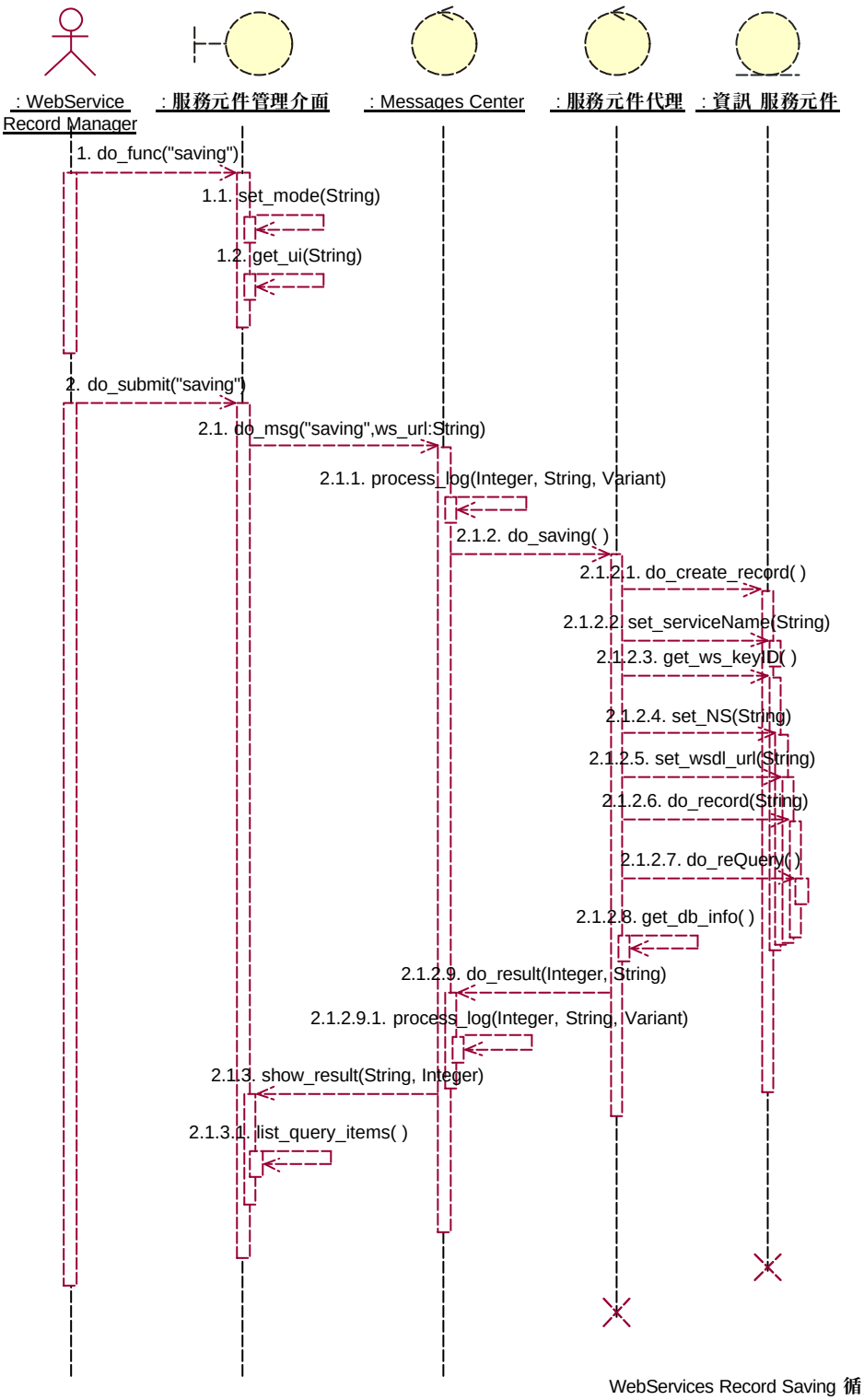
WebServices Record Querying 循序圖



WebServices Record Modifying 循序圖

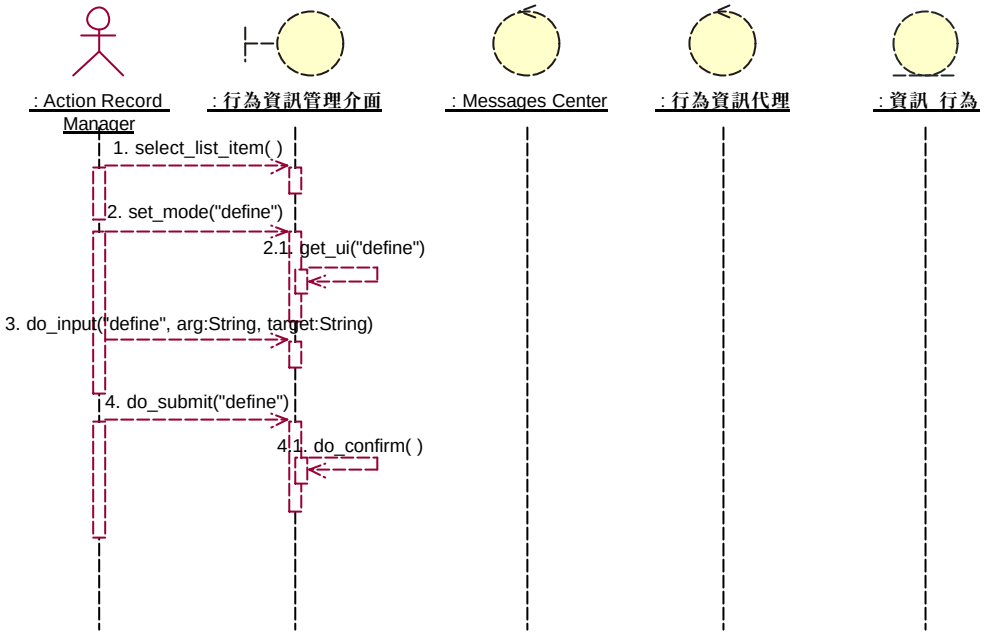


WebServices Record Deleting 循序圖

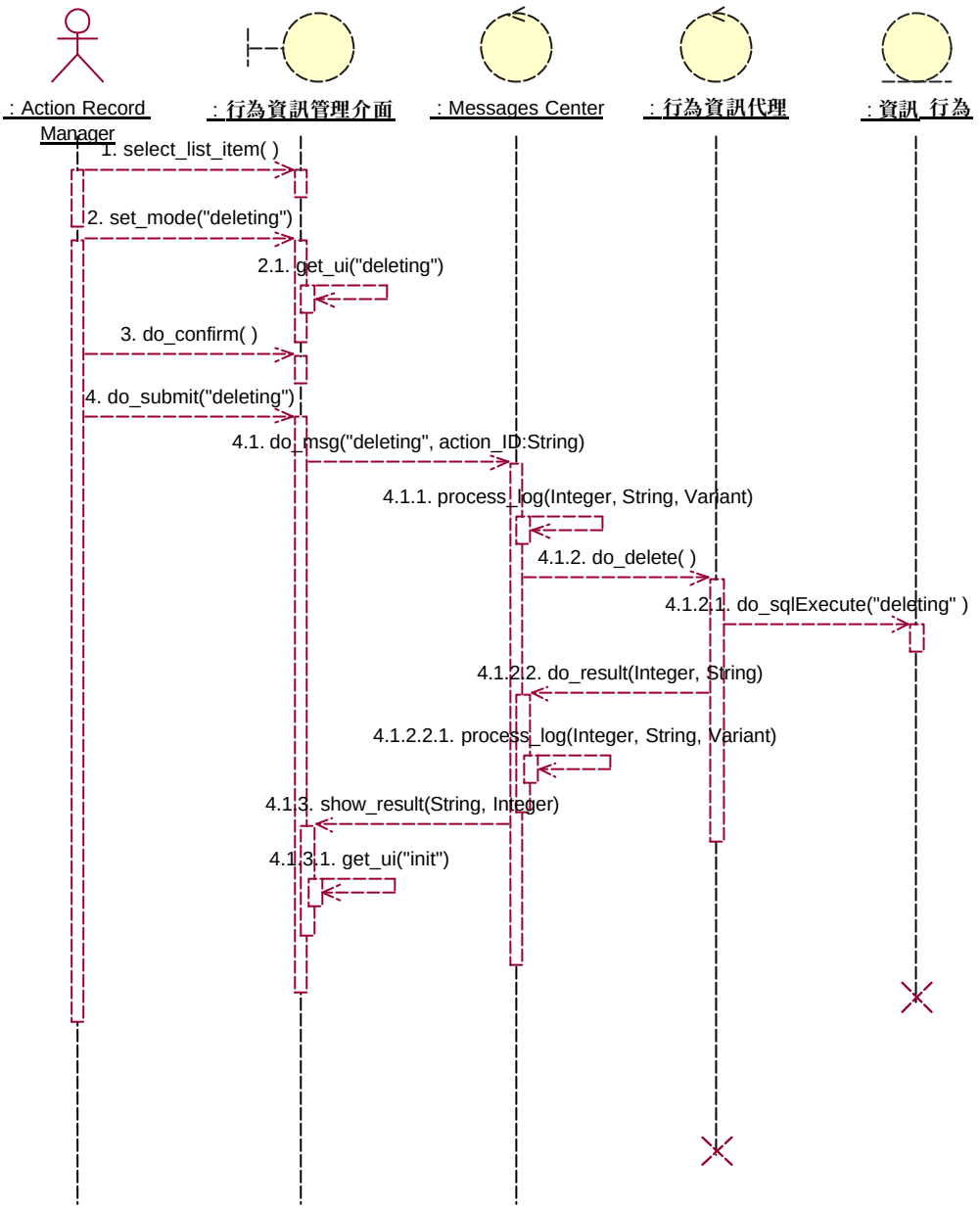


WebServices Record Saving 循序圖

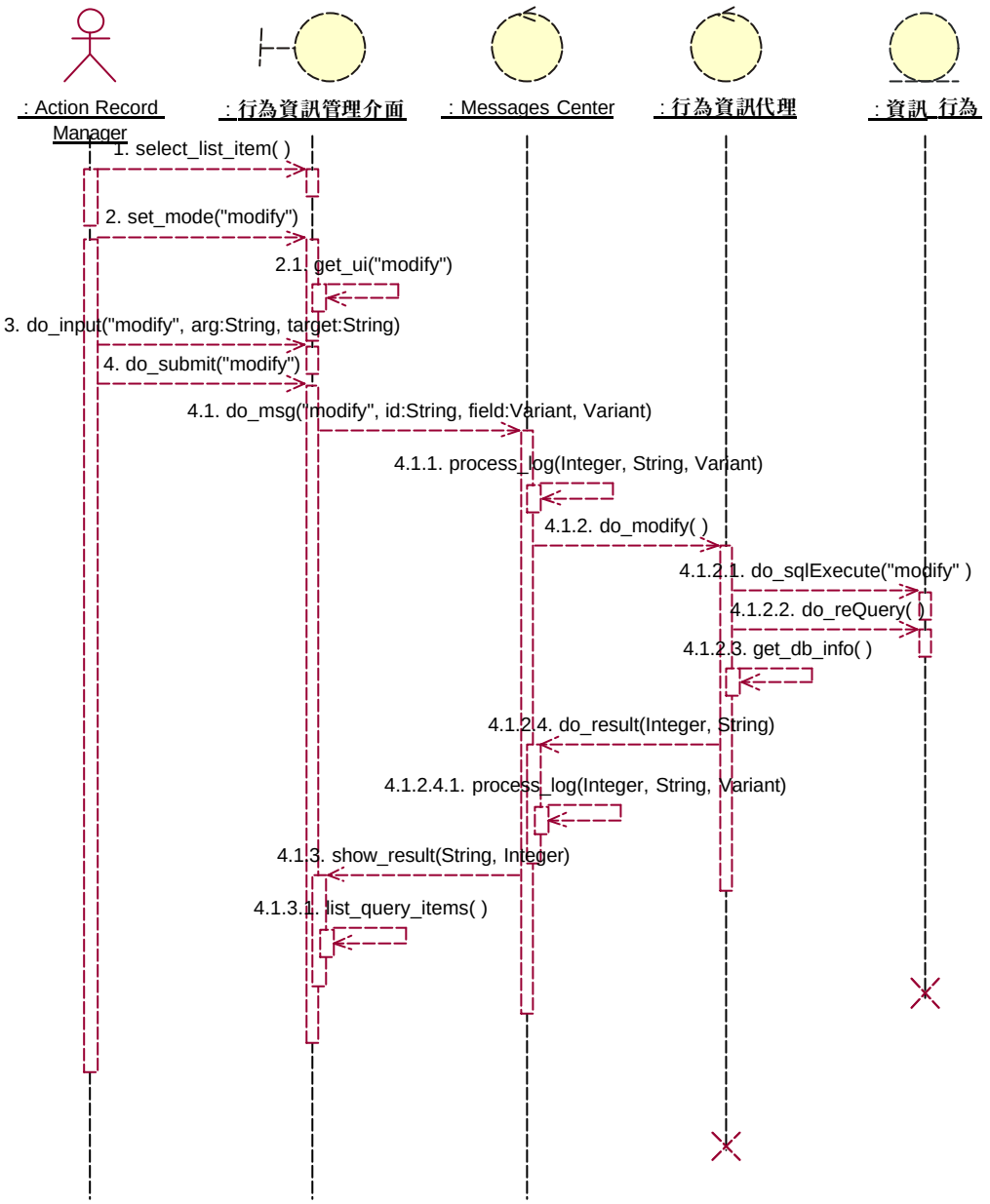
附錄二：動作屬性登錄相關類別循序圖



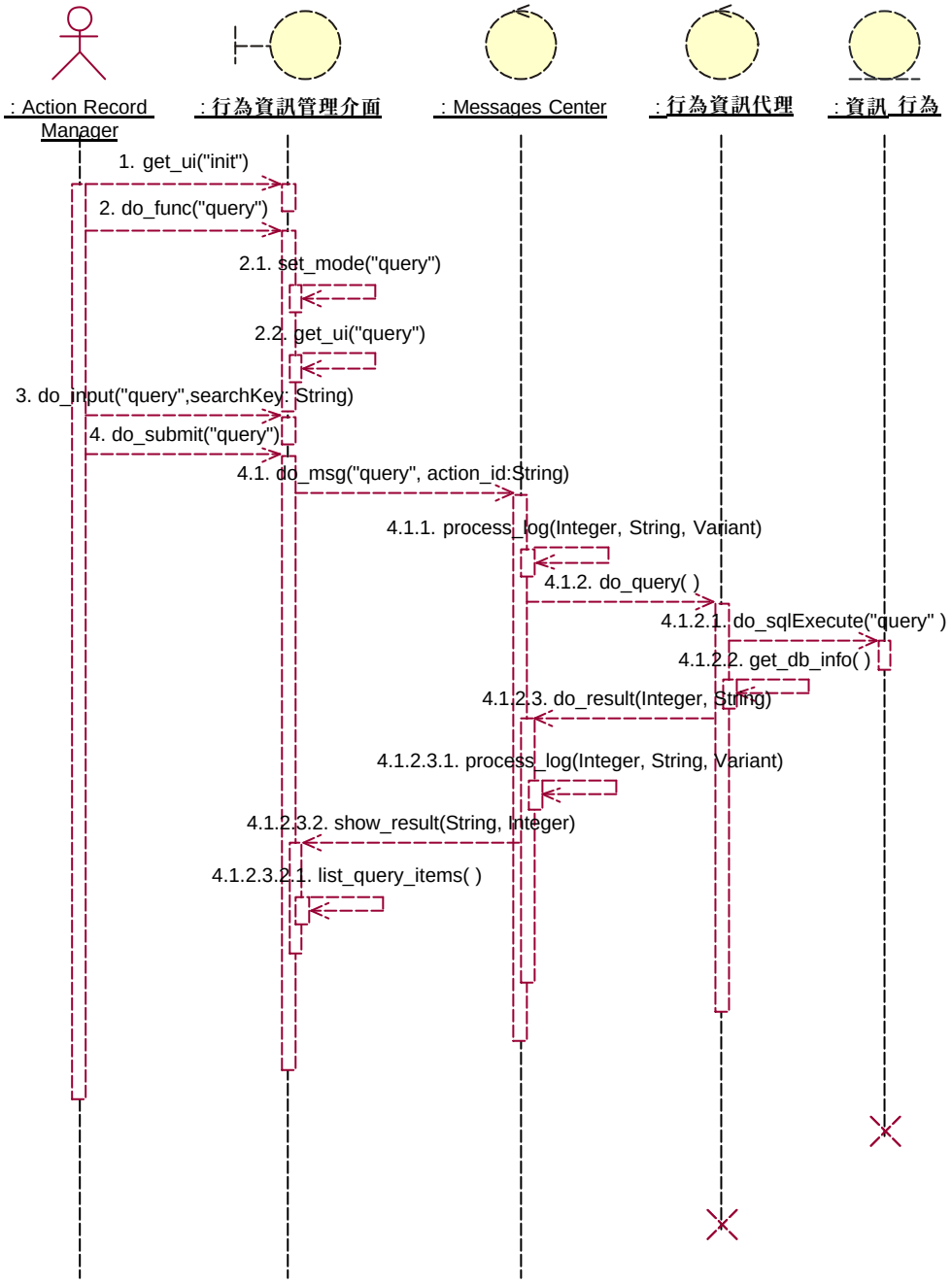
Action Record Define 循序圖



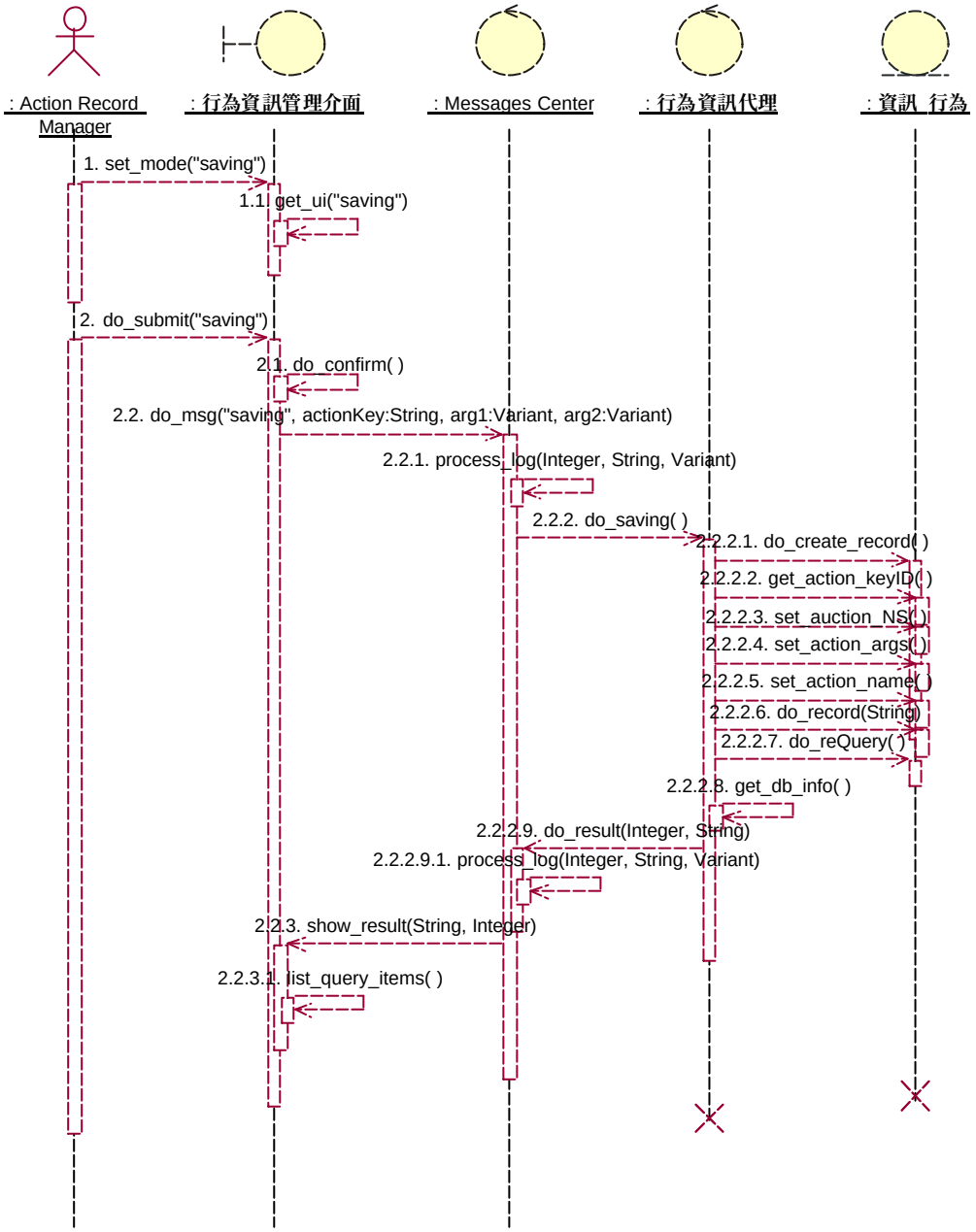
Action Record Deleting 循序圖



Action Record Modify循序圖

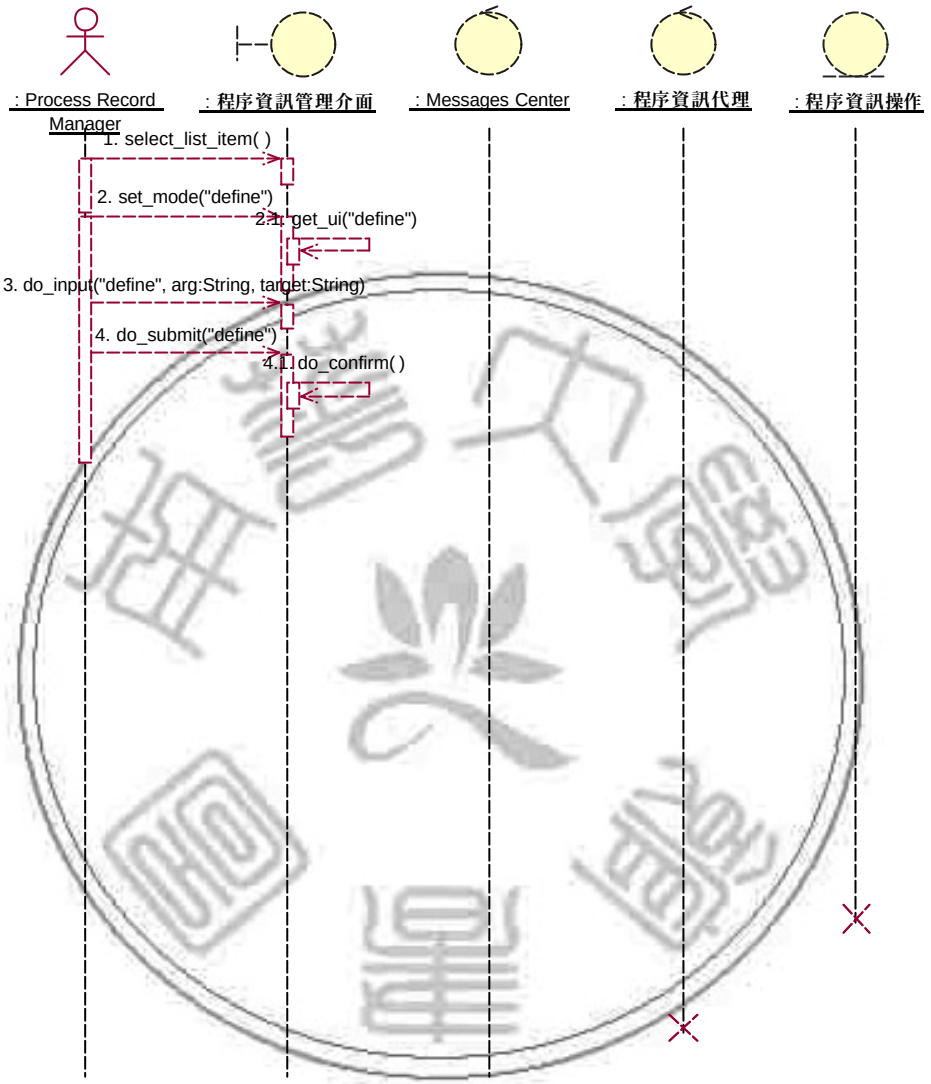


Action Record Querying 循序圖

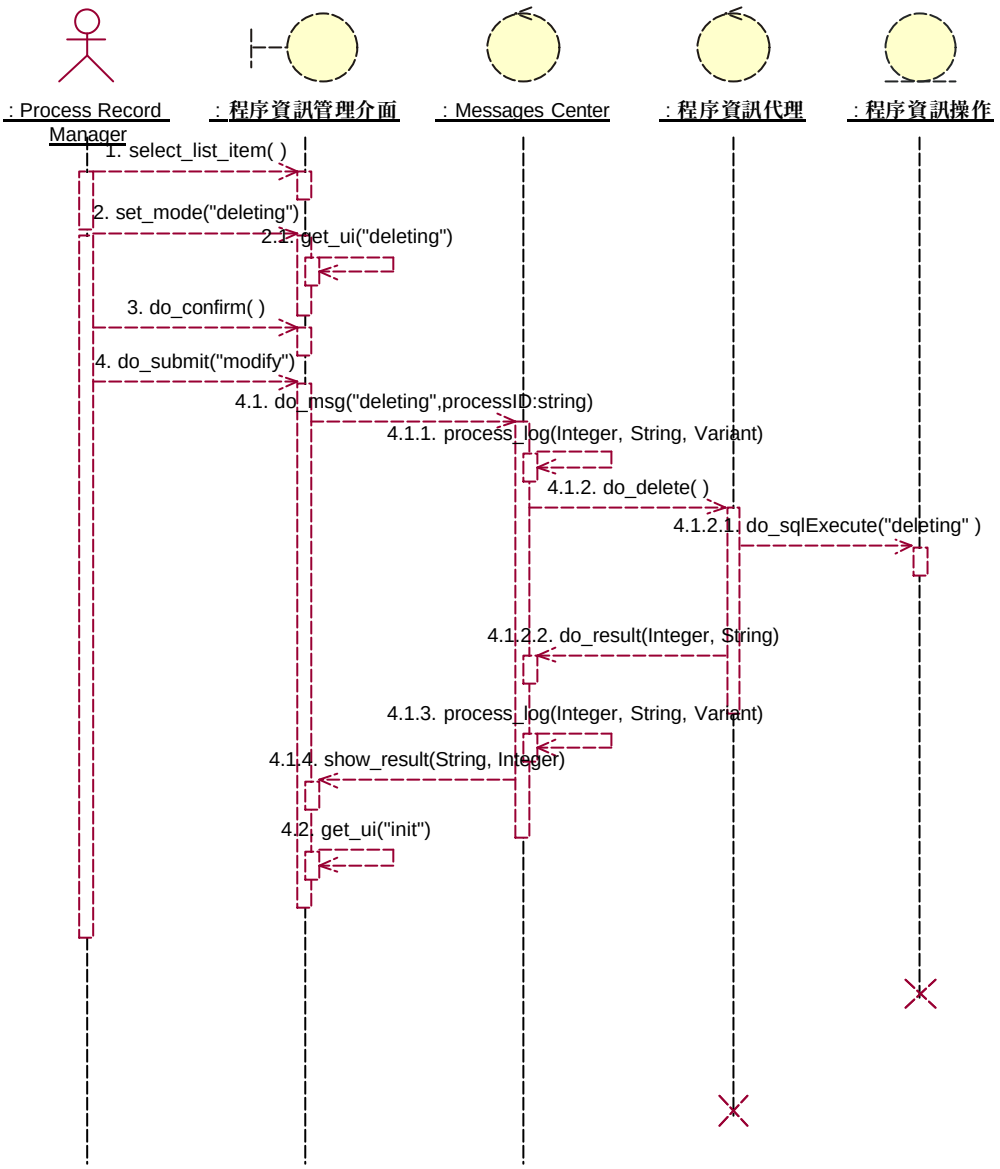


Action Record Saving 循序圖

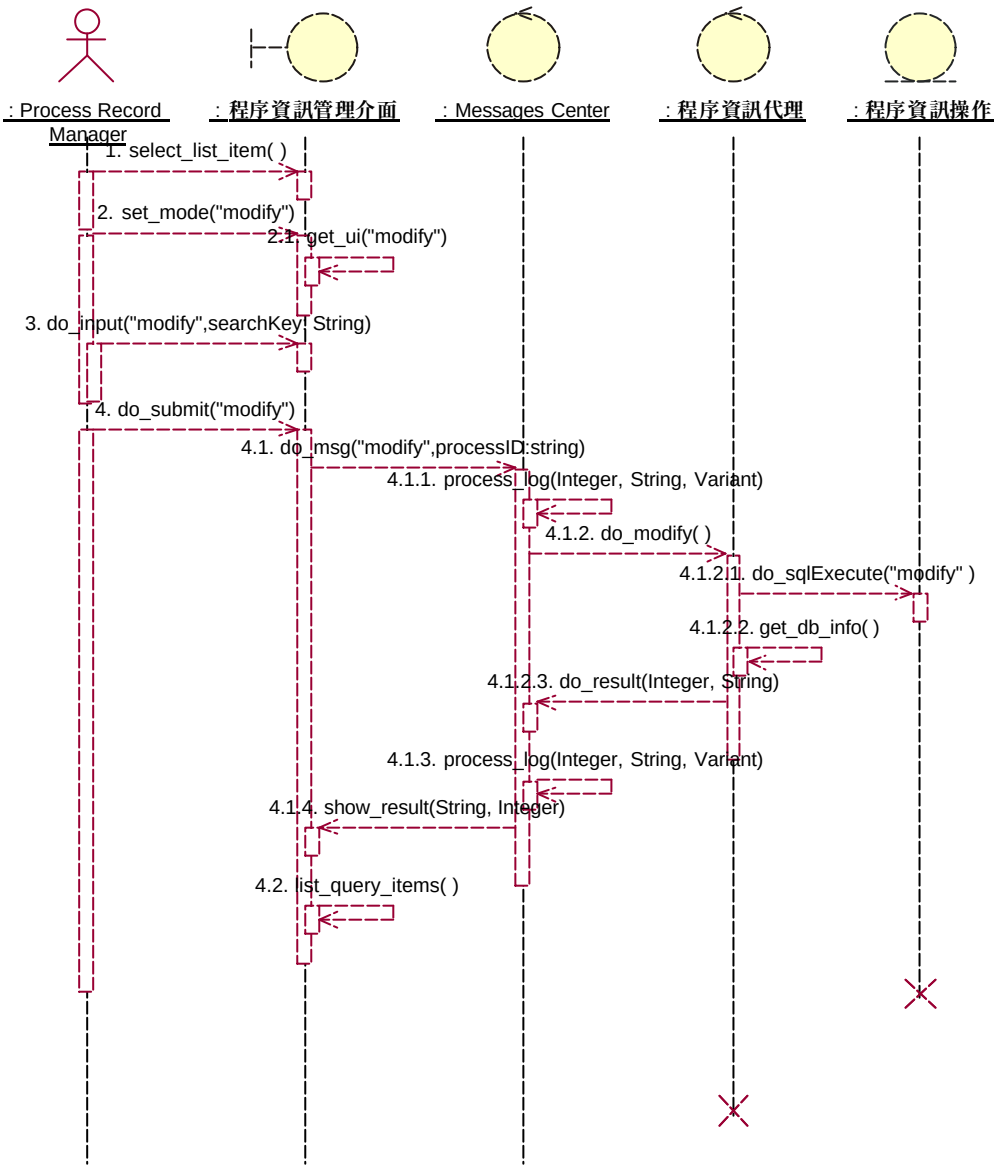
附錄三：程序資訊登錄相關類別循序圖



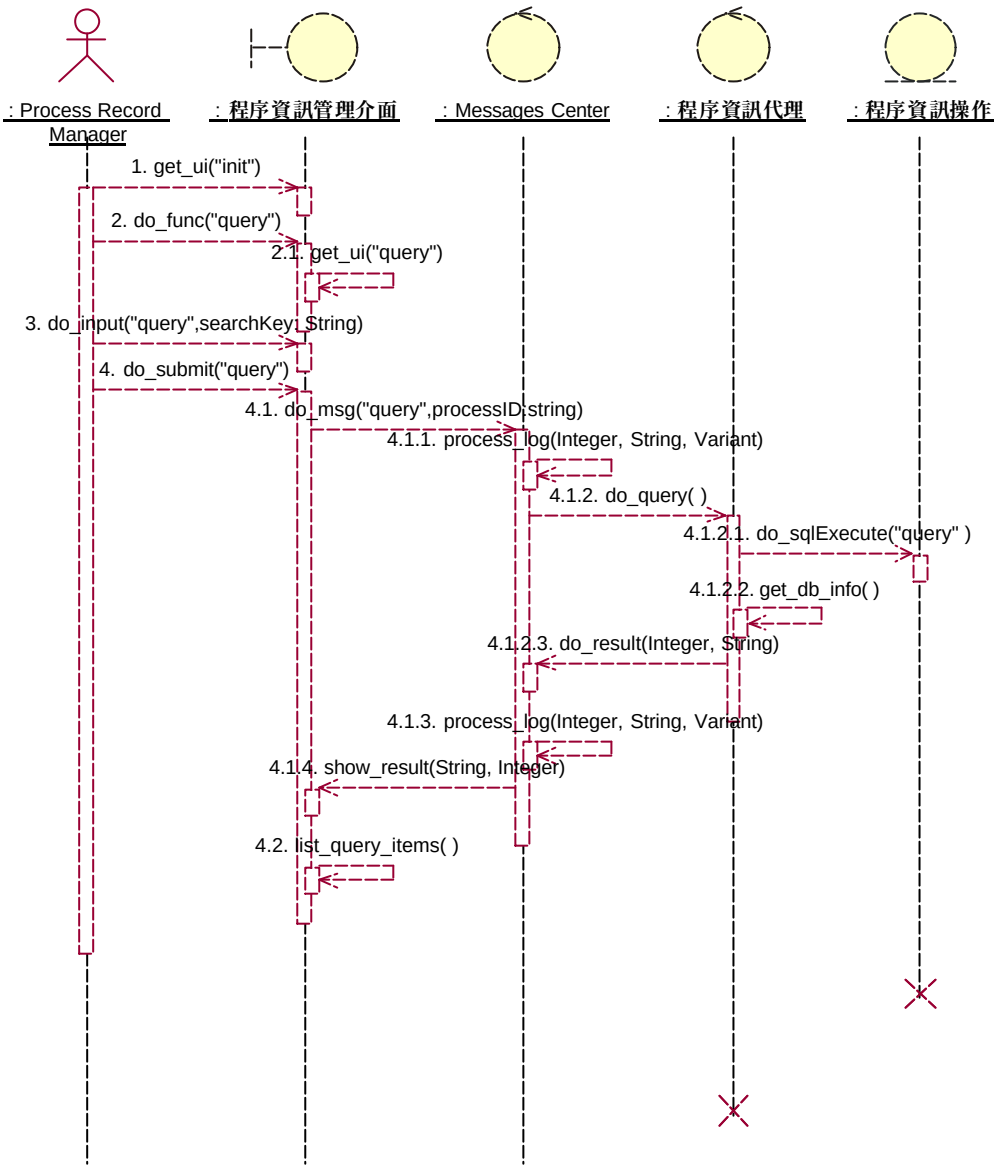
Process Record Deleting 循序圖



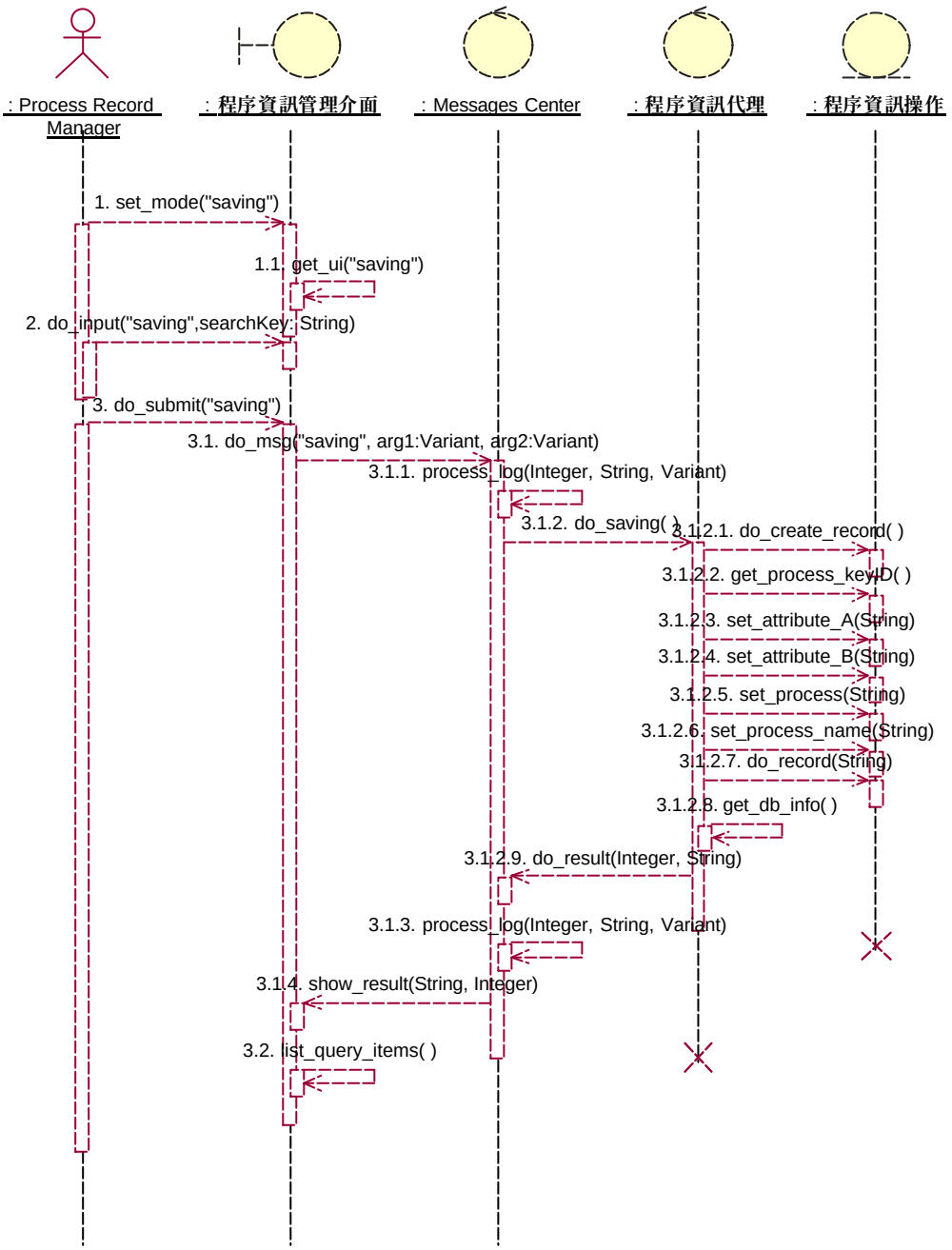
Process Record Deleting 循序圖



Process Record Modify 循序圖

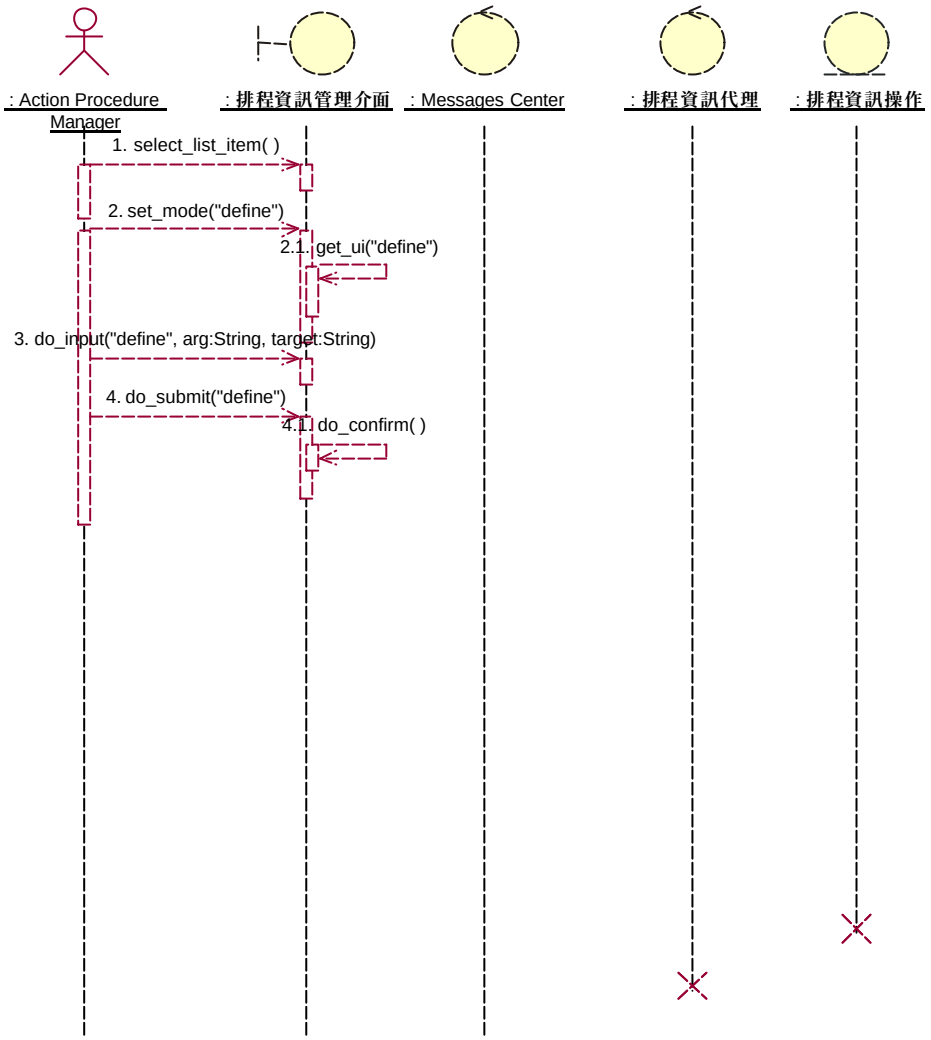


Process Record Query 循序圖

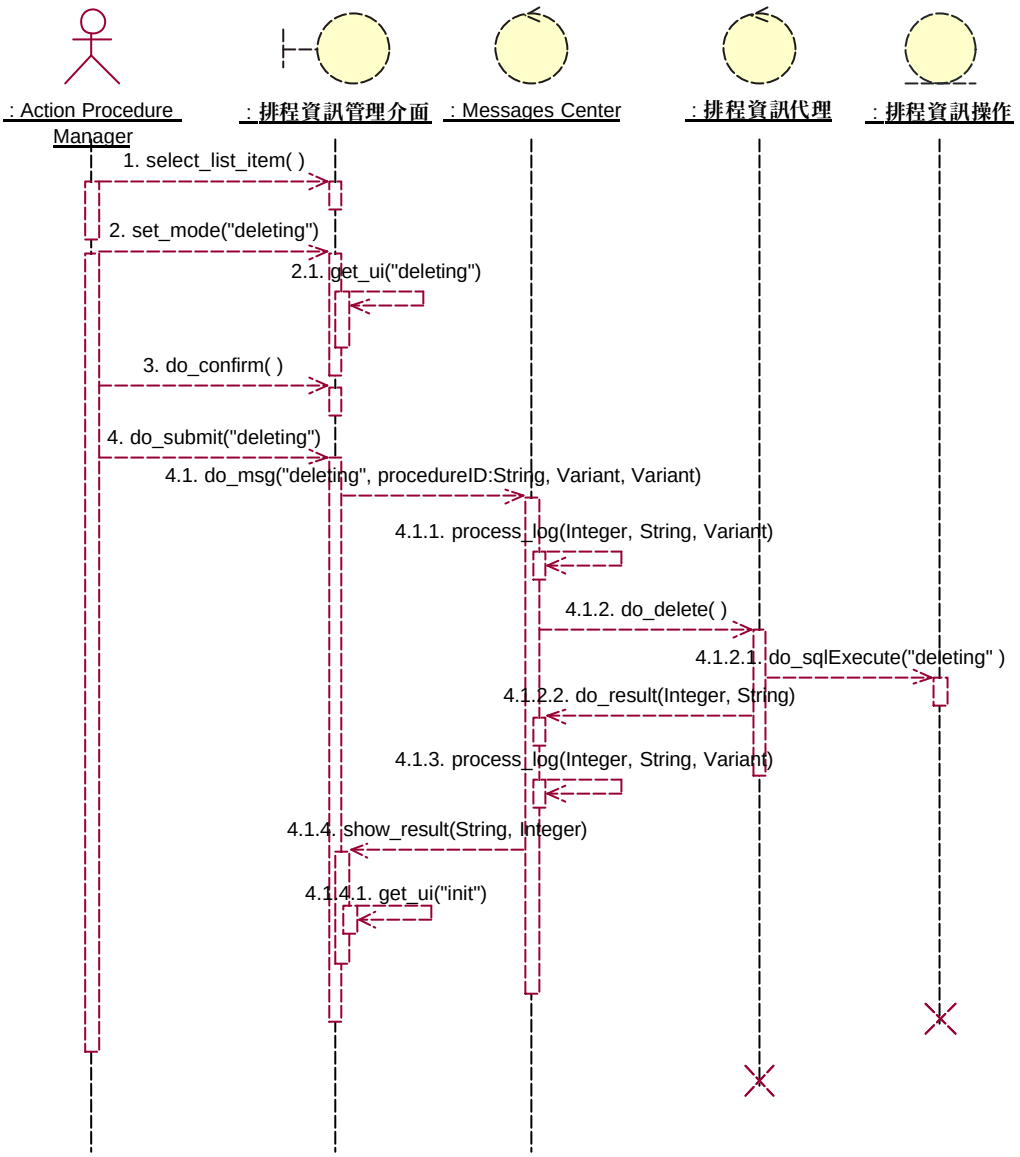


Process Record Saving 循序圖

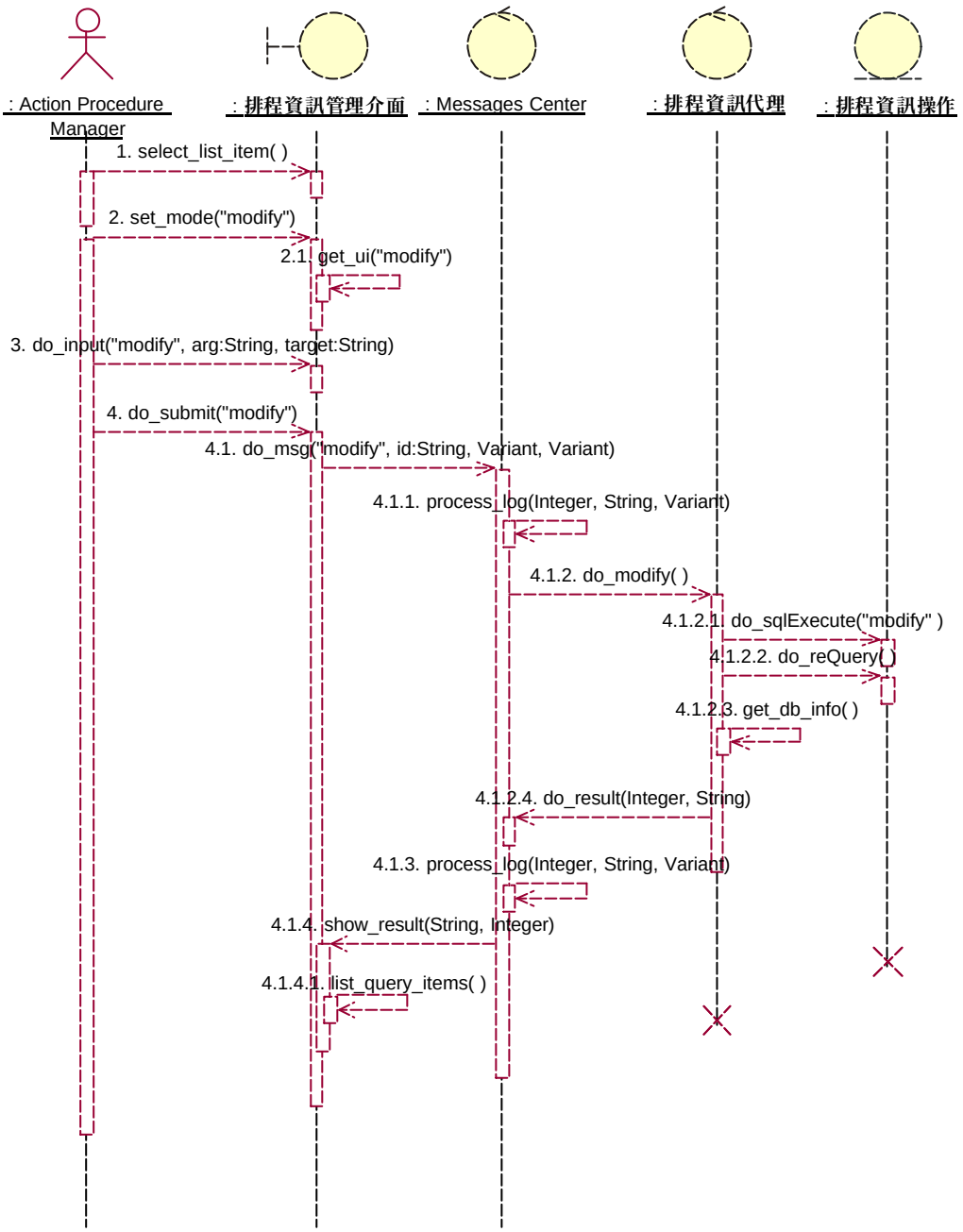
附錄四：程序排程資訊登錄相關類別循序圖



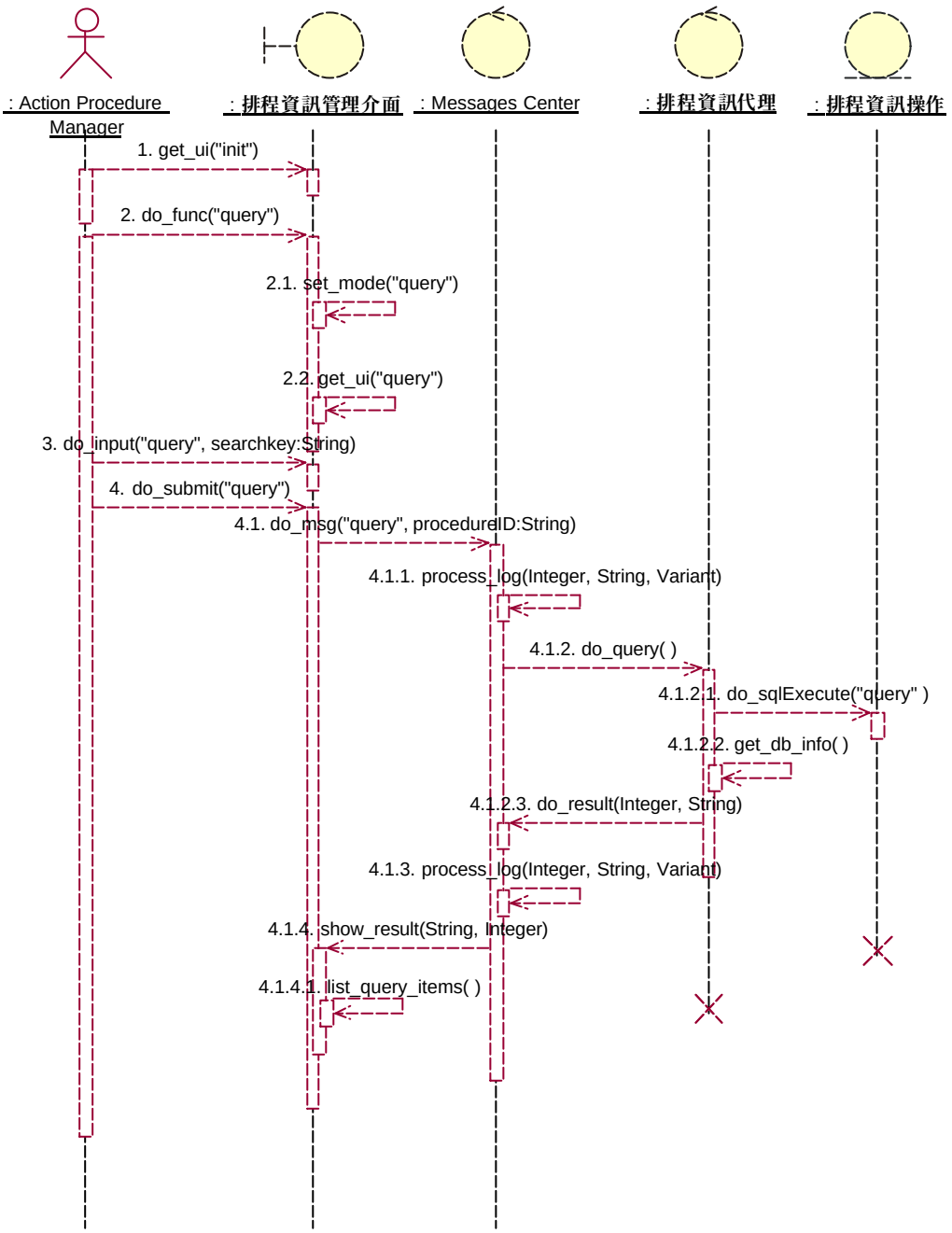
Procedure Record Define 循序圖



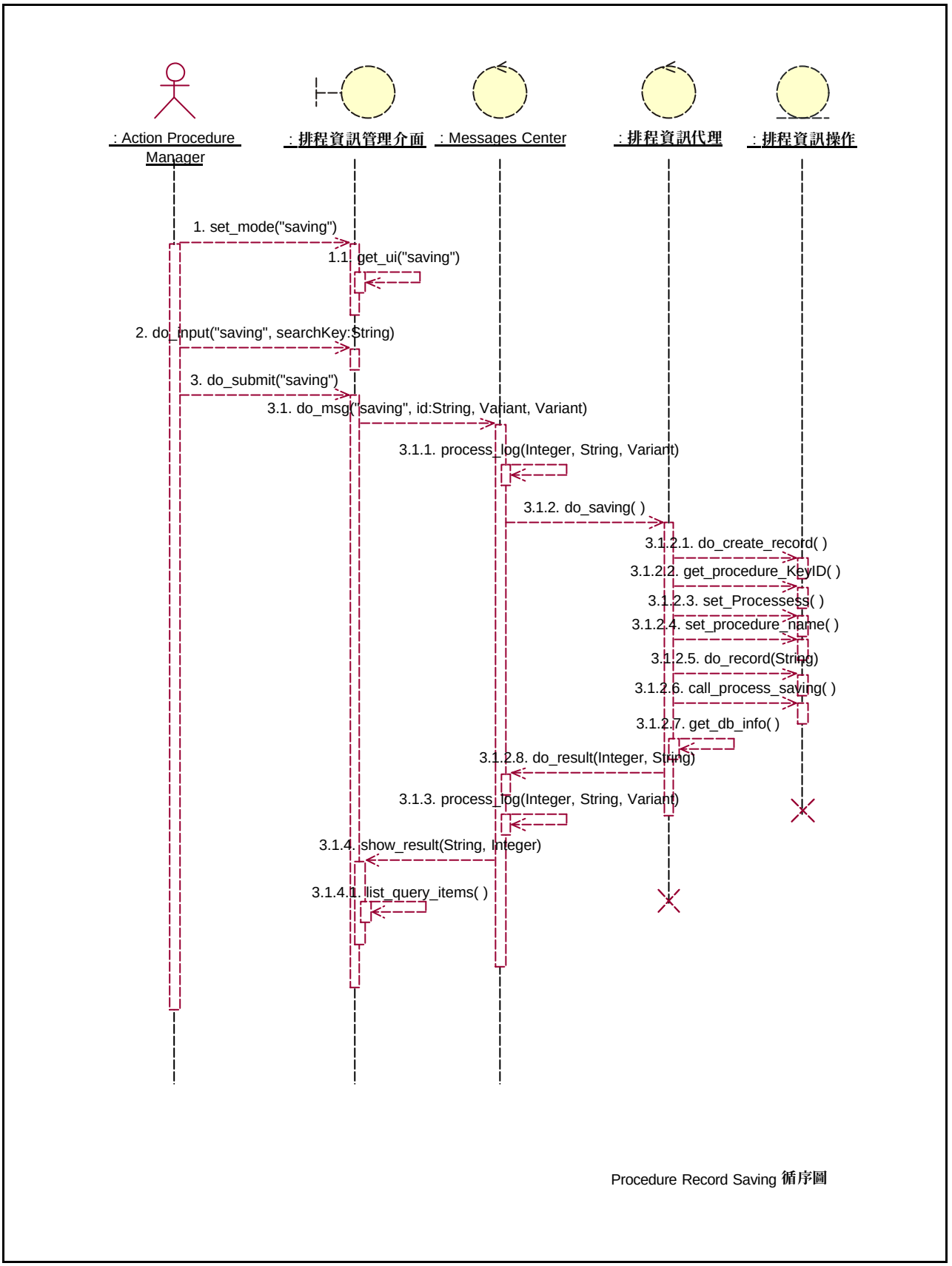
Procedure Record Deleting 循序圖



Procedure Record Modify 循序圖

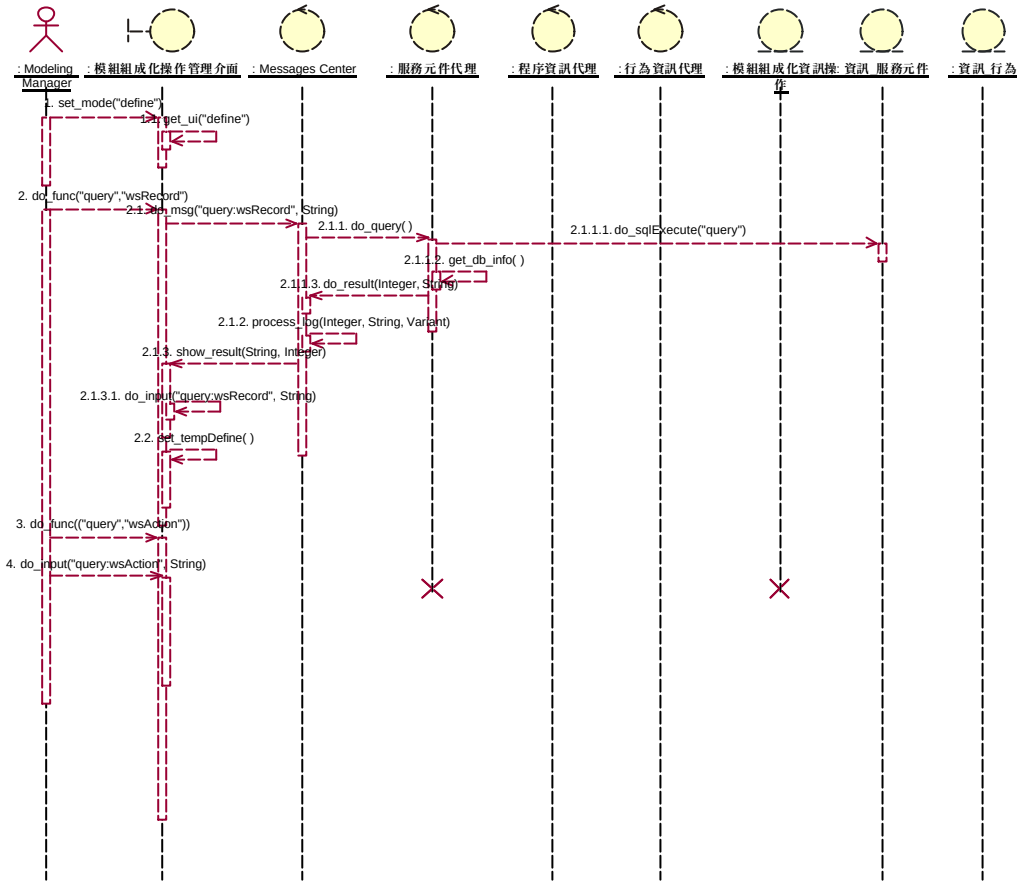


Procedure Record Querying 循序圖

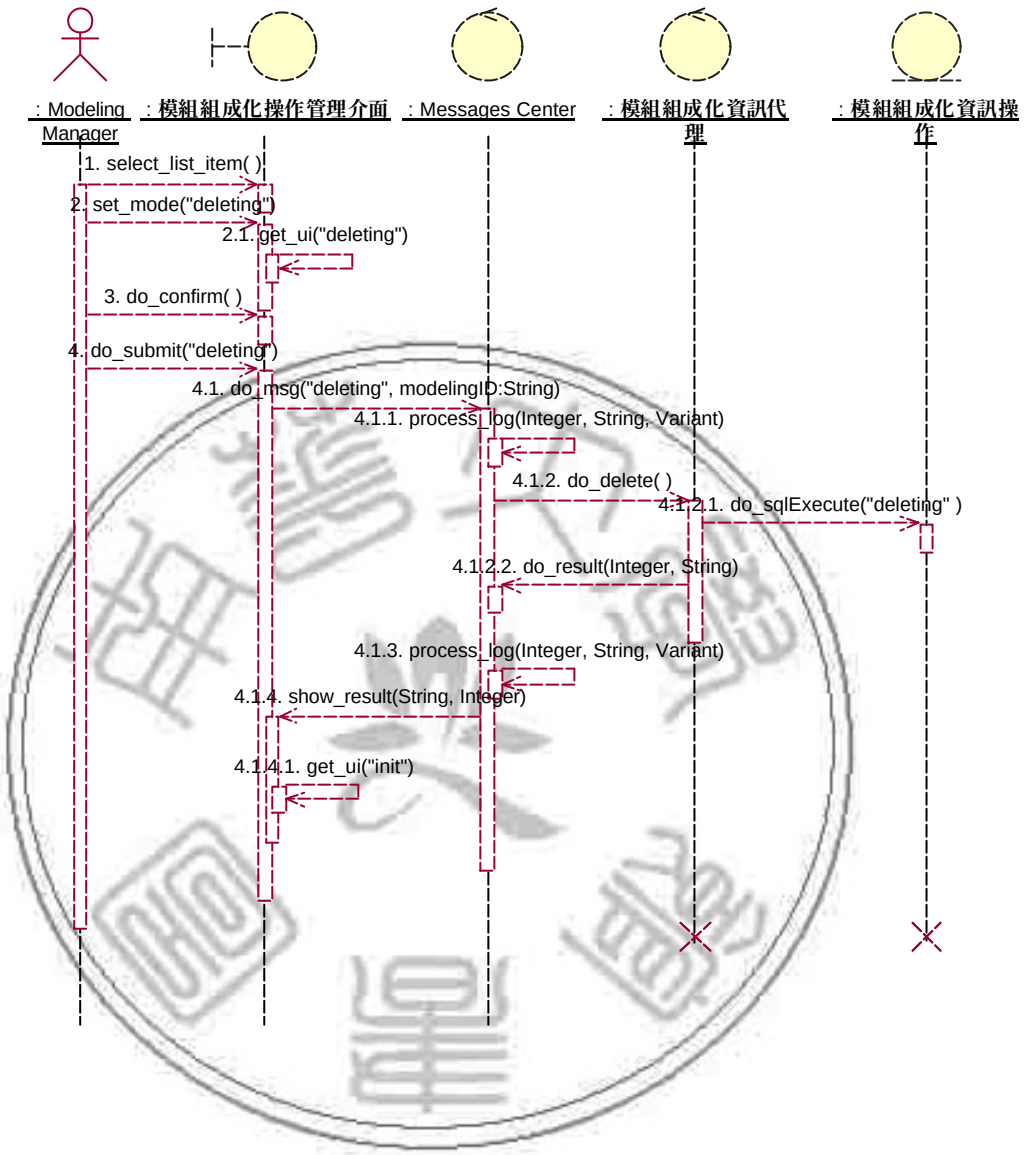


Procedure Record Saving 循序圖

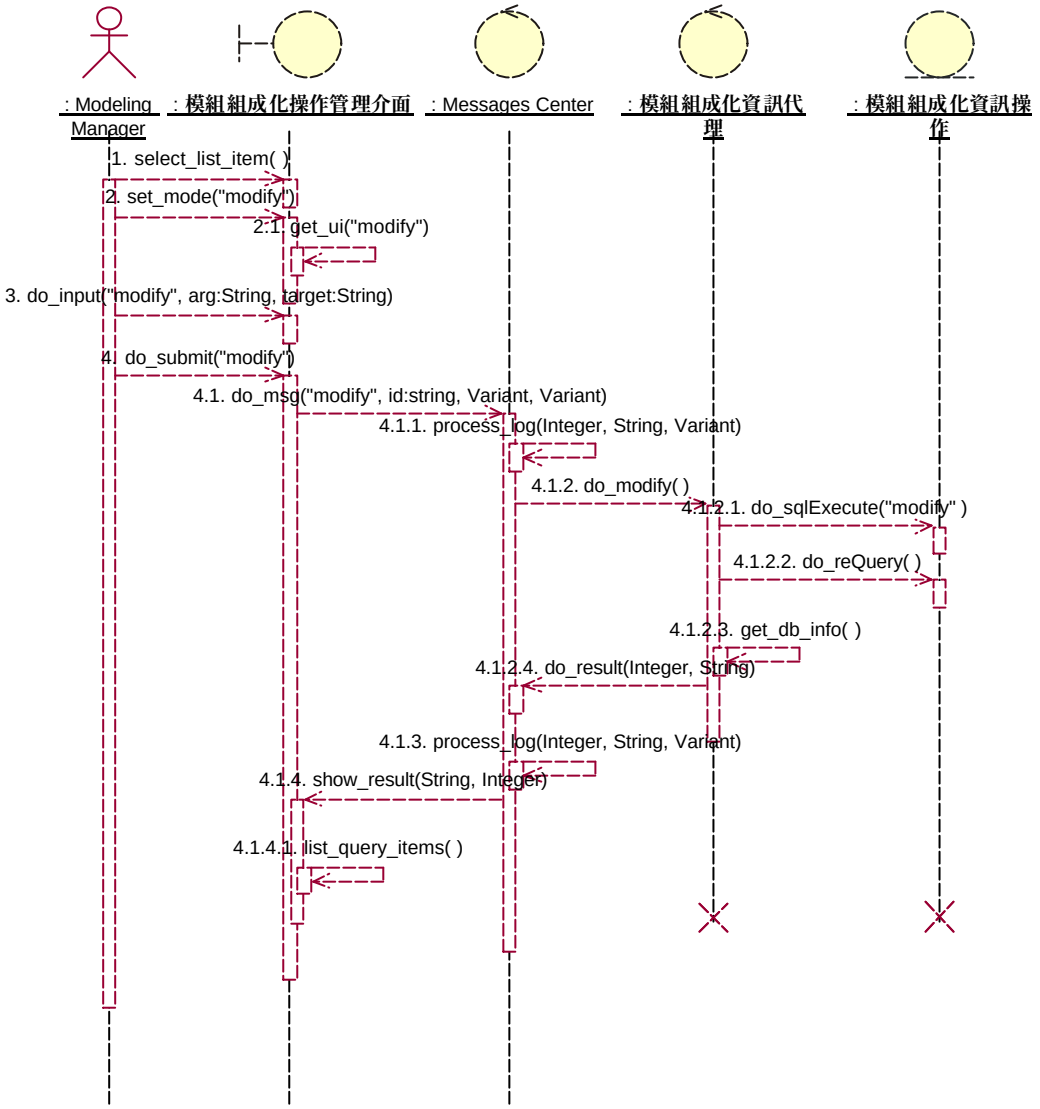
附錄五：模組化資訊登錄相關類別循序圖



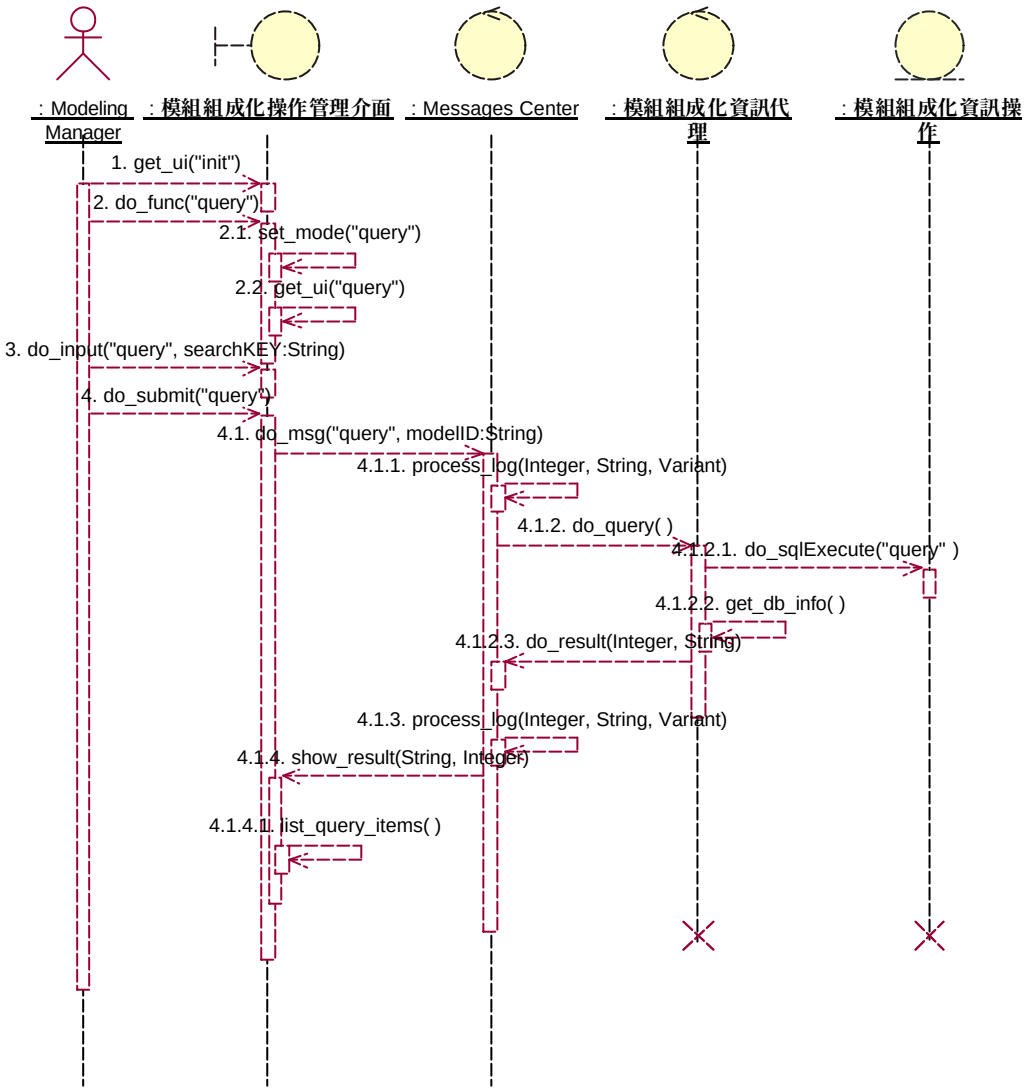
Modeling Record Modify 循序圖



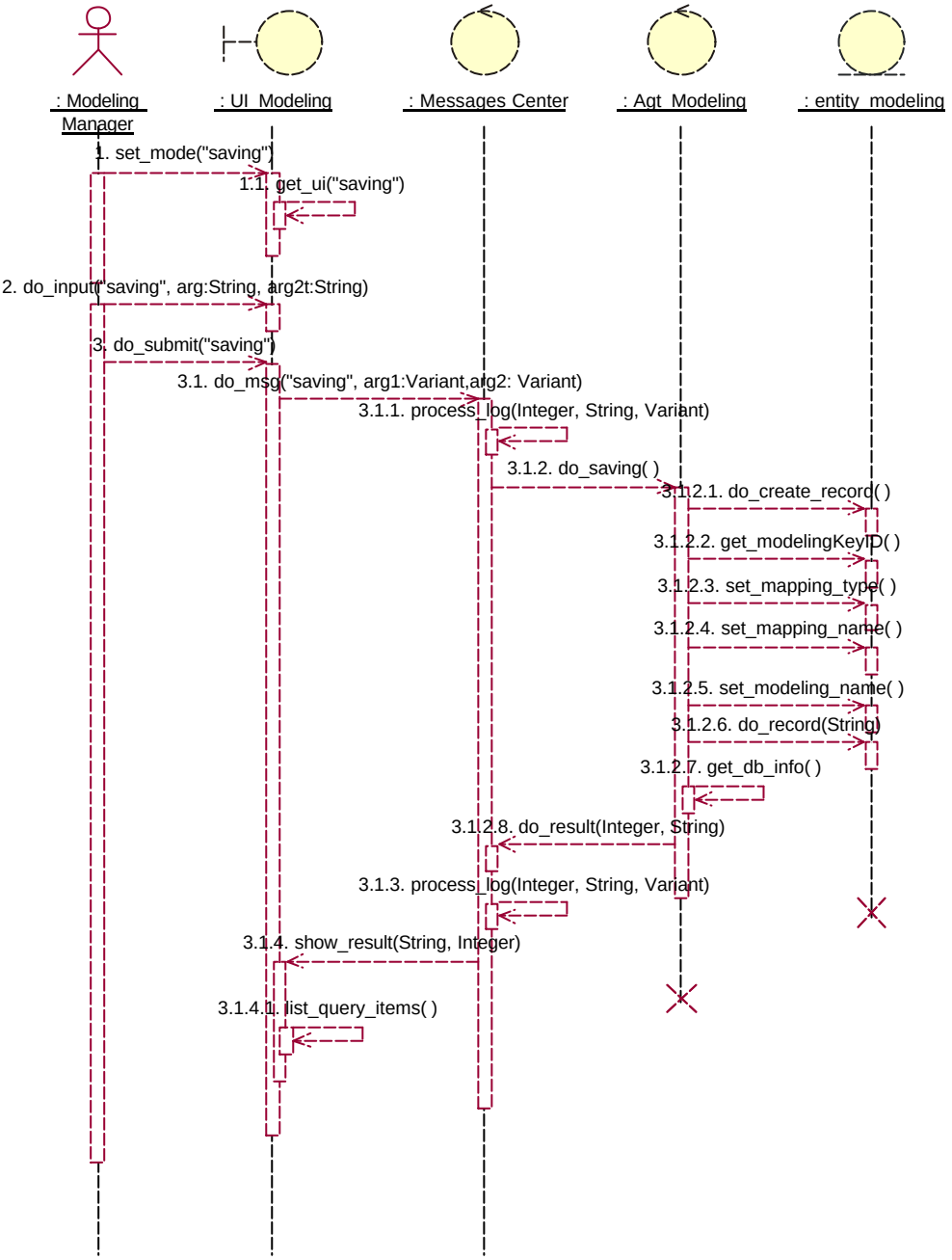
Modeling Record Deleting 循序圖



Modeling Record Modify 循序圖



Modeling Record Query 循序圖



Modeling Record Saving 循序圖