中間應用層資料服務物件模型之改進研究

學生：曾清義　　指導教授：王昌斌

南華大學資訊管理學系碩士班

摘要

三層式架構資訊系統之中間層服務模型是以服務物件化、網路化的資訊系統的核心。目前資訊科技（IT）界在強調的網路服務物件（WEB SERVICE）概念既是中間層服務模型的延伸，對於網際網路環境中使用者多、資料存取量大、變異性高與要求快速回應的需求特性上，中間層服務模型如何增進資料庫的存取效能、善用主機的資源，是 IT 部門重要的課題。我們利用分散式物件模型，提出一改進的服務物件模型稱為兩件式服務模型，類似以 Worker/Service Threads 的服務模型，使用一代理物件負責與前端訊息溝通，利用代理物件可以提供更多使用者的同時連線的需求。而服務物件則利用 Pool 的管理模式管理，改進資源的使用效率。我們並提出一動態調整系統資源使用的數學模式，讓系統依最新的狀況自動配置，使整體服務效能達到最佳化。

我們的模型已實際應用於南華大學校園資訊系統中，利用此模型整合 WEB 與傳統 Client/Server 架構，建構出新的服務形式。

關鍵詞：三層式系統、應用伺服器、WEB SERVICE、Distributed Objects

Improved Data Service Object Model In Middleware Service

Student：

Chin Yi Tseng

Advisors：

Dr. Chin Bin Wang

Institute of Department of Information Management

The M.B.A Program

Nan-Hua University

ABSTRACT

Three-tier model is the most popular architecture in current application information system. It is an important issue that the performance will be effected significantly under heavy load. Moreover, for the traditional system, it restricted the concurrent internet user due to resource limitation..

We propose a model, called worker/service threads. Our model has two stages. In the first stage, an agent response to connect to the internet user, the stage allow more user connect to the system at the same time. At the second stage, we provide a dynamic resource allocation model. The model can improve the resource utilization base on system status. The proposed model has been applied in Nan Hwa university and show the approach run efficiency

Keyword： Three-tier model ；Application server；Distributed-Objects；Web Service；Distributed System

誌　謝

本論文，非常感謝王昌斌教授費許多心力的指導批示，提供許多寶貴的修正意見，使得本論文能順利完成，並感謝各位口試委員的建議與指正，最後感謝資訊管理學系全體教職員兩年來之提攜與幫助，使得能如期完成學業，在此深深致謝。

目錄

摘要 ..... **I**

ABSTRACT ..... **II**

誌　謝 ..... **III**

目錄 ..... **IV**

圖表目錄 ..... **VI**

符號說明 ..... **VII**

符號說明 ..... **VIII**

第一章 緒論 **1**

第一節 研究動機 1
第二節 研究目的 2
第三節 研究方法與效益 2

第二章 文獻探討 **3**

第一節 三層式軟體架構 3
第二節 伺服型應用程式(SERVER APPLICATION) 6
第三節 分散式物件模型 8

第三章 問題分析 **12**

第一節 傳統的服務處理流程 12
壹、 階段一：建立連線 13
貳、 階段二：請求服務 14
參、 階段三：請求斷線 14
第二節 問題分析 15
壹、 耗用大量的記憶體資源 15
貳、 可提供連線數有限 16
參、 Client 端的連/斷線頻率太高, 影響效能 16

第四章 改良式資料服務模型 **18**

第一節 改良的資料服務模型結構 18
第二節 改良式資料模型處理流程 20
壹、 階段一：建立連線 20
貳、 階段二：請求服務 20
參、 階段三：請求斷線 21
第三節 整體效能的最佳化策略 21

第五章 效能改進分析 **24**

第一節 反應時間效率改進分析 24
壹、 傳統的反應時間 24
貳、 新模型之反應時間 25
參、 效率改進分析 26
第二節 資源使用效率改進分析 27
壹、 新模型之資源使用分析 27
貳、 資源差異比較 28
第六章 實作模型設計 29
第一節 類別定義與說明 29
壹、 TCom 類別: 29
貳、 TFO：代理物件工廠類別 29
參、 SI:服務介面的定義 30
肆、 TASO:代理物件的父類別 30
伍、 TSO:服務物件父類別 30
陸、 TSOInfo:服務統計資訊類別 31
柒、 TSOM：服務物件管理者類別 32
捌、 TTimerManager：時間週期控制物件類別 35
第二節 類別關係圖 36
壹、 類別 36
貳、 代理物件類別關係圖 37
參、 服務物件與管理者類別關係圖 38
肆、 時間週期控制類別關係圖 39
第三節 循序圖（SEQUENCE DIAGRAM） 40
壹、 前端連線 40
貳、 服務請求 41
參、 ASO 取得一可用的服務物件 42
肆、 ASO 歸還服務物件 43
伍、 計時控制事件 44
第四節 類別方法虛擬程式碼 45
壹、 TASO:代理物件類別 45
貳、 TSO：資料服務物件類別 46
參、 TSOInfo：服務統計資訊類別 47
肆、 TSOM：服務物件管理者類別 49
伍、 TTimerManager: 時間週期控制類別 55
第七章 結論與建議 56
參考文獻 58
自傳 60

圖表目錄

圖表 1 三層式架構 3
圖表 2 南華大學網路選課系統部署架構 5
圖表 3 工作者與服務者執行緒模型 8
圖表 4 Microsoft DCOM 分散式物件模型 10
圖表 5 三層式資料服務流程 13
圖表 6 行程耗用之時間因素 15
圖表 7 連線與資料存取之時間關係 17
圖表 8 改良的資料服務模型 19
圖表 9 需求與等待曲線 21
圖表 10 類別關係圖 36
圖表 11 代理物件類別關係圖 37
圖表 12 服務物件與管理者類別關係圖 38
圖表 13 時間週期控制類別關係圖 39
圖表 14 連線循序圖 40
圖表 15 服務請求循序圖 41
圖表 16 SOM 分派服務物件循序圖 42
圖表 17 SOM 收回服務物件循序圖 43
圖表 18 時間事件處理循序圖 44

符號說明

系統最佳化策略符號定義：

Δt：**系統時間週期**

t_n：**系統第**n**個時間週期**, $n > 0$

t_{n+1}：$t_n + \Delta t$

Q_{rn}：t_n**時服務需求量**

Q_{wn}：t_n**時等待服務量**

q_{son}：*SO***物件於**t_{n-1}**至**t_n**的**Δt**內平均服務量**

Δq_{rn}：t_n**時的需求改變量**

ΔQ_{sn}：t_{n-s}**至**t_n**的前**s**個狀態需求改變量的累計**

K_n：t_n**時需求增量率**

N_n：t_n**時,系統存在的***SO***物件數**

$N_{reserve}$：**服務物件最小保留量**

符號說明

系統效能分析符號定義：

t_{sc}：*Client*建立連線

t_{sd}：*Client*斷線時間

T_s：*Service*本身處理時間

T_{sp}：*Pool*模式的*Service*處理時間

t_{st}：*Client*與*Service*資料傳送時間

t_{bc}：*DBMS*建立連線時間

t_{bd}：*DBMS*斷線時間

T_{bs}：*DBMS*資料存取與傳送時間

T_{bsp}：*Pool*模式的*DBMS*處理時間

且T_{bs}恆大於$(T_s + t_{st})$,t_{bc}恆大於t_{sc}

N_c：第*C*次連線,累計服務次數

U：服務利用率 = 連線與服務總時間比

E：服務效率改進比

第一章　緒論

第一節 研究動機

三層式軟體架構(Three Tier Software Architecure)於 1990 年代被提出時，是為改進二層式(Client/Server)架構的限制，於使用者介面(Client)與資料管理伺服器(Server)，新增加一層所謂的「中間層伺服器」(Middle Tier Server)，或稱應用伺服器(Application Server)，此層最主要是在處理商業邏輯與應用規則的能力，使得在分散式的處理環境中，能容納更多的使用者，增加處理效力、使系統更有彈性、更易維護與程式再使用、並更具透明性。

但傳統三層式的資訊系統架構，對於網際網路使用者多、存取的資訊內容變異性高及要求快速反應的需求特性上，中間層的資料服務模式遭遇相當的困難，尤其對瞬間大量需求湧入時，服務品質迅速降低。

另一方面，由於網路的普及，資訊處理環境走向分散式，主機、資料庫、各種服務分散於複雜的網路世界中，使得資訊系統的發展，朝向分散式物件化的服務模型建置，因此如何將中間層資料服務模式，以服務物件化的模型設計並改進資訊處理的效能，善用系統的資源，增進伺服主機的處理效率，提供更好的服務品質，是資訊科技（IT）部門重要的研究課題。

第二節 研究目的

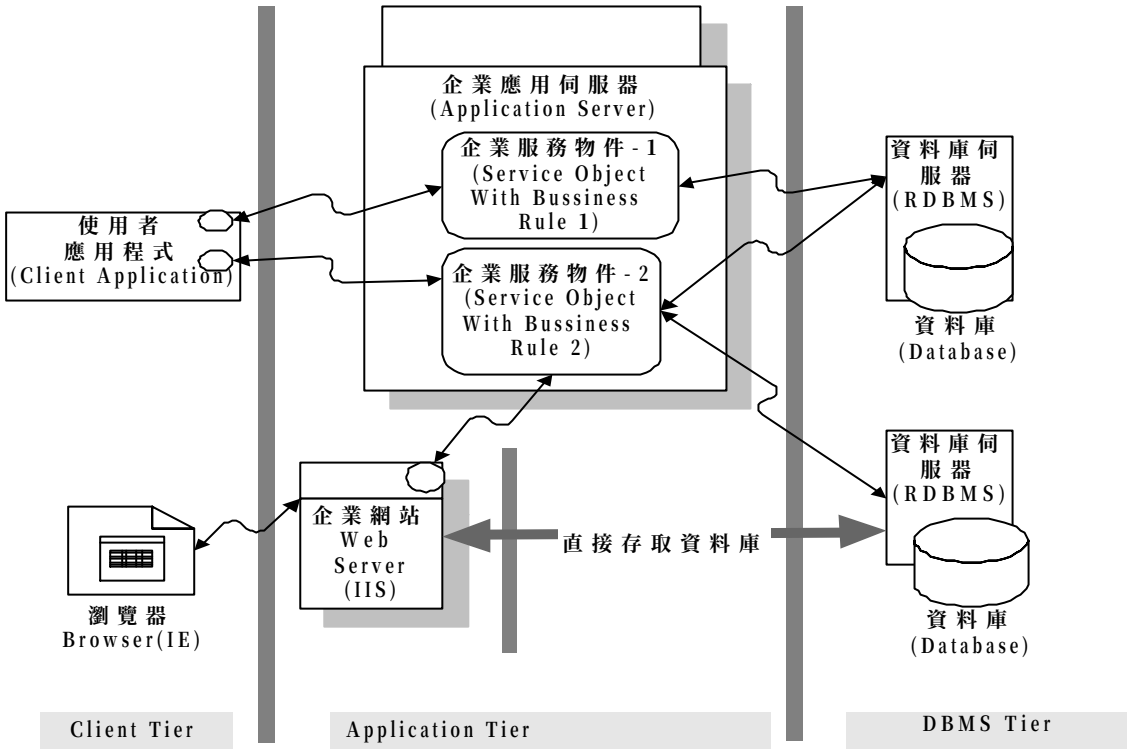
在傳統三層式的資訊架構中中間層資料服務模式所面臨的問題是：耗用大量的記憶體資源、可提供連線數有限、Client 端的連/斷線頻率太高，影響效能。此皆肇因於未能妥善處理與資料庫管理系統間資料存取問題。在本論文中，我們提供一改進傳統三層式的中間應用層資料管理的模式，使能適用於分散且多變的網際網路環境特性。並利用分散式物件模型設計中間應用層，使系統發展更具有彈性與調變能力。

第三節 研究方法與效益

我們提供改進的中間應用層資料管理的模式，此模式類似以 Worker/Service Threads 的服務模型，應用分散式物件模型（Distributed Object Model），如 DCOM、CORBA 的實作技術，專注於存取資料庫資料之服務物件的管理，此模式最大的效益可改善傳統三層式架構的缺點，提高執行效率，我們的模式可改進資料庫的資料存取速度，與動態管理系統資源（如記憶體，資料庫系統連線數）的能力，使資源的利用最佳化。

此一模式，我們已實際應用於南華大學校園行政自動化系統，實證其可有效的改善服務的反應時間及減少資源的使用。

第二章　文獻探討



圖表 1 三層式架構

第一節 三層式軟體架構

為改進二層式(Client/Server)架構的限制，在使用者介面(Client)與資料管理伺服器(Server)間，新增加一層所謂的「中間層伺服器」（Middleware Server），或稱應用伺服器 (Application Server)。此層最主要是在處理商業邏輯與應用規則的能力，使得在分散式的處理環境中，使系統更具透明性、彈性、更易維護及程式再使用【15】。因此三層式（Three Tier Software Architecure ）軟體架構，於 1990 年代被提出（圖 1），目前已成為企業應用的標準架構。現

在更進一步利用分散式物件模型（Distributed Object Model），以 SERVICE 的觀念，實作企業服務物件（Service Object），應用於網際網路的資訊處理環境中，如 WEB SERVICE。

三層式架構【13】，如圖 1、被分成：

1. **Client Tier：又稱** Presentation Logic Layer，負責提供與使用者互動的相關操作，如文字輸入、對話窗、顯示、列印等等，是實作使用者介面程式（GUI）。

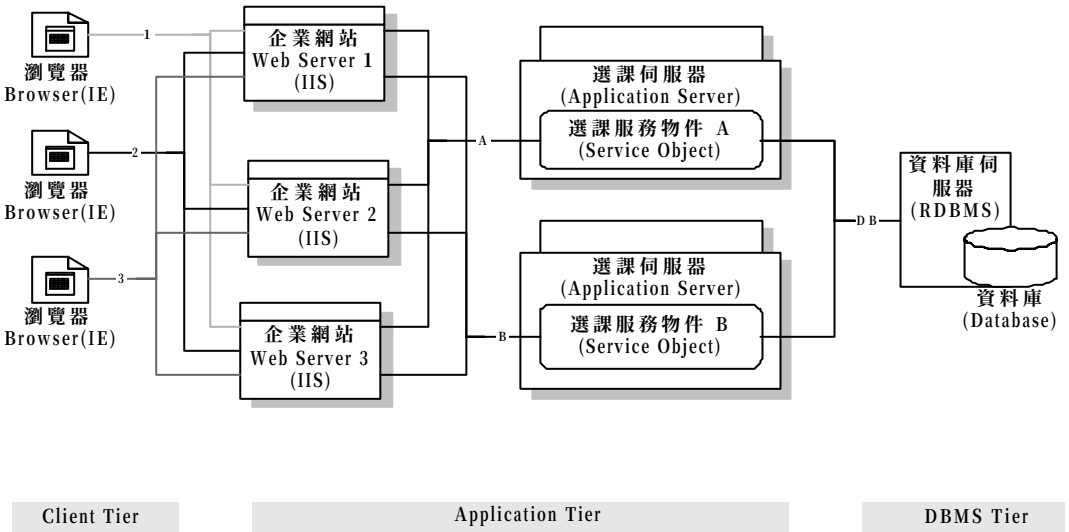
2. **Application Tier： 又稱** Bussiness Logic Layer **或** Domain Server，負責處理商業服務的邏輯與交易應用的規則，提供一致性的服務介面給許多 Client Applications。在圖 1 中，Web Server **可能直接存取資料庫伺服器（**DB Server）或再要求其他 Application Server **服務或某一** Application Server **要求另一** Application Server **服務，不管中間層經過多少** Servers **的服務轉介，只要實作商業邏輯處理，則通稱之** Middleware Server。

3. **DBMS Tier：又稱** Data Storage Layer **，負責資料庫或檔案儲存管理的服務，資料管理元件能過保證於分散式的環境下，資料的一致性與複製能力。為使資料庫達到通透性（**Transparency），資料庫管理層不使用廠商特定的處理語言（SQL）。如圖 1 所示，資料可能分散於不同的 DB Server 中，根據 Application Server 的不同服務物件整理、更新、轉換、分發企業的資料庫。

三層式軟體架構之所以能更有處理效力、更有彈性、更易維護與重複使用、更具透明性，是因為將商業邏輯集中化於中間層

處理，集中化處理的結果，使得商業邏輯功能的管理與改變，更容易施行。當商業處理部分規則變更，只要調整一次中間層相關的子系統功能，則所有 Client Applications 得到相對一致性反應。

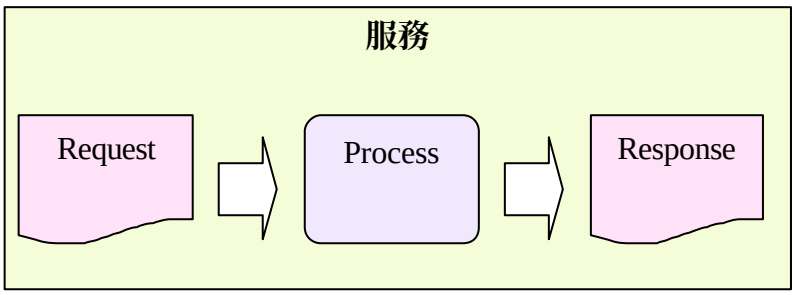
有時中間層根據不同的功能而分割成多個不同服務應用單元，這個架構又被稱為多層式架構（Multi Layer），典型的例子，是一些網際網路的應用系統（Internet Applications），這些應用系統，通常具有一由 HTML 撰寫的 Thin Clients【14】，與由 C++或 Java 撰寫的 Application Server，而 Web Server 介於中間負責產生 HTML 網頁，當 Web Server 接受到前端的要求時，解譯需求，呼叫相對應的 Application Server 服務，當資料由 Application Server 傳回後，再產生 HTML 回應前端完成服務的要求。南華大學網路選課系統，既是以此架構實作完成的。如圖 2 所示，使用二部選課應用伺服器與三部 Web Server 是為系統的負載平衡與設備備援計劃考量的。



圖表 2 南華大學網路選課系統部署架

構

第二節 伺服型應用程式(SERVER APPLICATION)



應用伺服器(Application Server)**【17】，基本是一個業務導向的伺服型應用程式**(Server Application)**，所以所面臨的問題或環境因素與一般伺服器是一致的。一般伺服型的應用程式，通常接受從不特定來源的** Client **端來的請求（**Request**），經過處理程序（**Process**），提供一些特殊資訊（**Response**）稱為服務（**Service**）。通常這些請求的數量與到達率（**arrive rate**）是不受控制的。所以它們必須長時間持續的執行，以應付來自網路（區域或廣域）上各個** Client **的要求，它們必須能有效的處理大量的要求與最小延遲時間的反應能力。典型的伺服器如** DataBase Servers,Web Servers,Directory/Name Servers,Mail Servers **等等都屬之。**

伺服型應用程式，依服務的需求或作業環境的不同，有各種不同的實作方式與分類，如我們依伺服器對 Client **端要求處理時狀態（**State**）維護的分類，可分為有狀態（**Stateful**）與無狀態（**Stateless**）的伺服器【**3**】【**17**】。**

無狀態伺服器：對每一 Client **端要求服務時都是相互獨立的，**

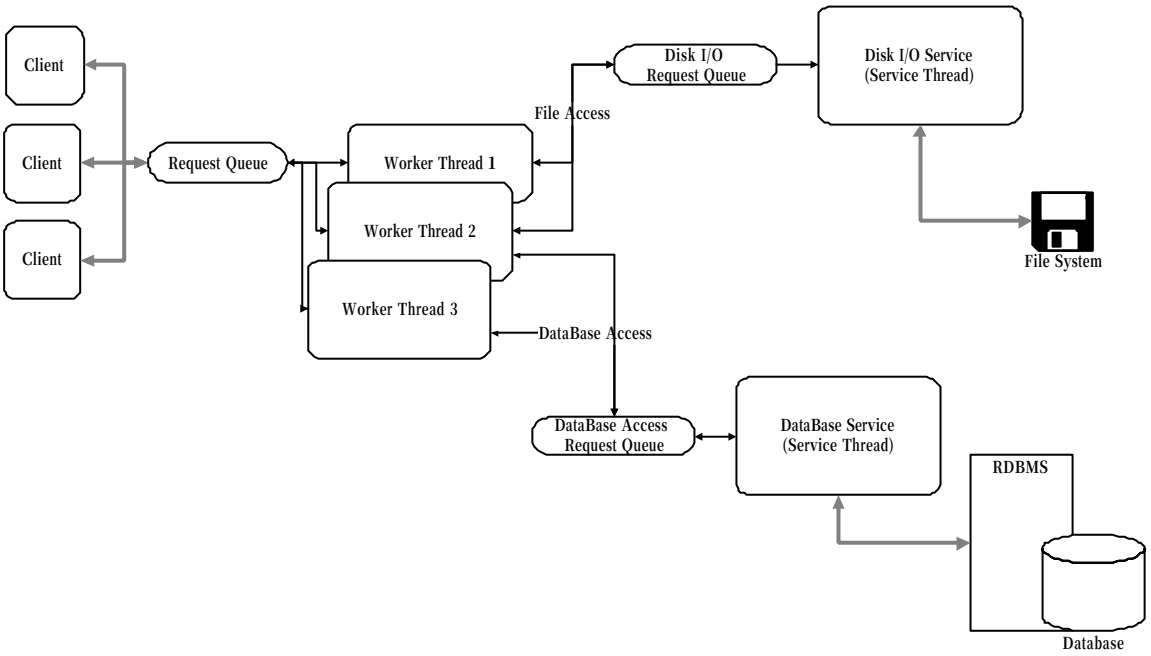
不需維護狀態資訊以供下次服務的參考，如 DNS Servers,Time Servers。

有狀態伺服器：當 Client 端要求服務時，首先伺服器開啟一連結(Connection)，伺服器一直保持此連結並維護對 Client 端的服務狀態(state)，直到全部服務結束，中斷連結。通常此有狀態服務，伺服器會對每一 Client 的連結建立一獨立的 Session 以維護 Client 的狀態資訊。如 Database Servers。

許多的伺服器在實作方面，通常會以「為每一連結（connection），獨立建立一執行緒(Thread) 服務」的架構設計，另一種改進的架構模型稱為工作者與服務者執行緒（Worker/Service threads）模型，如圖 3，此模型使用所謂 Pool【3】的方式，先建立某一數量的工作者執行緒（Worker Threads），已備使用，當 Client 端的要求到達時，會先放進等候處理序列(Request Queue)，有空的工作者執行緒會依序從 Queue 中取出要求執行。Worker Threads 的方式，不但可以避免重複於建立與釋放執行緒的程序，並可以根據並行處理（Concurrency）的需求，彈性的調整 Worker Threads 的數量。但是有限的 Worker Thread，並不適合直接等待一些較耗時間的處理作業的完成，如 Disk I/O 或是要求其他 Server 的服務等，因為如此可能造成所有的 Worker Threads 處於停滯的狀態，所以對一些特殊服務的需求，另以服務者執行緒（Service Threads）的方式來完成。當 Work Thread 需要要求特殊服務的需求時，如圖 3 的 File Access，Database Access，會將此要求放進相對應要求序列中，交付給 Service Thread 處理，Work Thread 暫停處理此項工作，繼續處理其他未完成的工作或從等候

序列再取出等待處理的要求事件。當 Service Thread 完成服務後，再通知 Worker Thread 繼續完成未完的工作。一般等候序列（Queuing）的工作緒（Thread）處理模式，在日常生活中是到處可見的，最主要乃在使服務專業化以及協調處理工作時間不同步的問題。

我們的中間層應用層的資料管理模式，既類似以(Worker/Service Threads)的服務模型，來改善資料庫資料的存取速度。



圖表 3 工作者與服務者執行緒模型

第三節 分散式物件模型

在網路技術愈來愈成熟與系統愈來愈複雜的情況下，資訊系統的發展愈來愈朝向分散式與物件化的結構發展，分散式系統的開

發，通常必須考慮系統的承載能力（Scalability）、開放性（Openness）、異質性（Heterogeneity）、資源的共用性（Resource Sharing）、容錯率（Fault-Tolerance）【10】，它通常較能充分利用電腦的資源與較快的反應能力，但相對的複雜度也將隨之增加，為了減少系統的複雜度，必須讓軟體能重複使用，為達到此目的，軟體必須類似 IC 組件方式設計。因此軟體朝物件導向的方式分析與設計乃是必然的結果，物件導向的方式分析與設計，是將系統功能與流程作適當的切割，把相關的資料與處理方法封裝成一一個功能相關或獨立的元件，將每個元件介面標準化，如 IC 的接腳。如將此系統元件分開部署於各型電腦主機中，讓它們協調運作，則構成所謂的分散式物件的處理模式。各大電腦公司或組織，有見於分散式物件的處理模式的效益，陸續提出一些分散式物件的模型，目前被廣泛使用的標準有 Common Object Request Broker Architecture（CORBA）[Object Management Group 1995] 【5】，Distributed Component Object Model（DCOM/COM+）[Microsoft] 【11】，JAVA/ Remote method invocation（JAVA/RMI）[Sun Microsystems]。

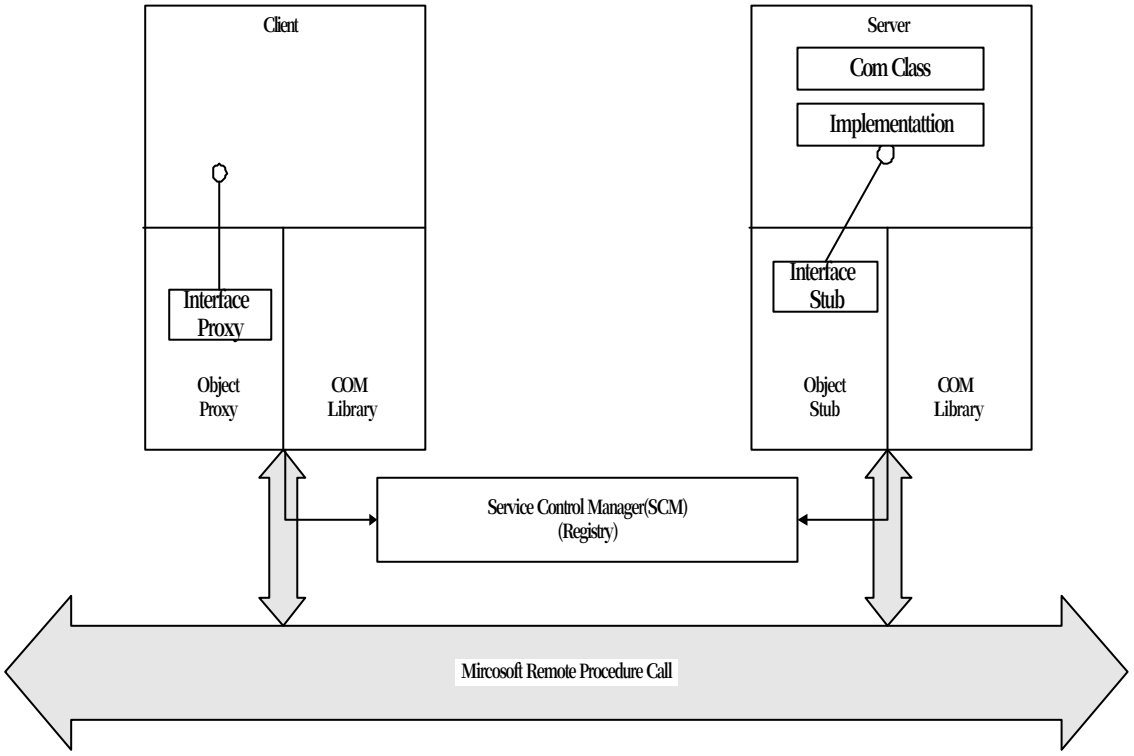
分散式物件的處理模式，最基本的要完成下列四項任務的定義：

1. 物件命名原則（Naming Space）：使得分散於各處的物件不會衝突。
2. 物件介面的定義方式（Interface）：定義物件所提供的服務。
3. 物件庫物件登錄管理模式（Registry）：定義物件的使用與管

理方式。

4. 物件呼叫與回應的程序（invoke）：定義物件的溝通方式。

在圖 4 是 Microsoft DCOM 模型，其命名原則，是 Windows OS 根據需求配置，給於物件一組世界唯一的 GUID 物件代號，物件介面定義於型別庫中（Type Library），物件登錄於 Microsoft Windows OS 的系統登錄資料庫（System Registry），由系統登錄器管理，DCOM 模型利用 Object Proxy 與 Object Stub 透過 RemoteProcess Call （RPC）完成物件呼叫與回應的任務。本論文實作的服務物件，既運作於 COM 的物件模型。



圖表 4 Microsoft DCOM 分散式物件模型

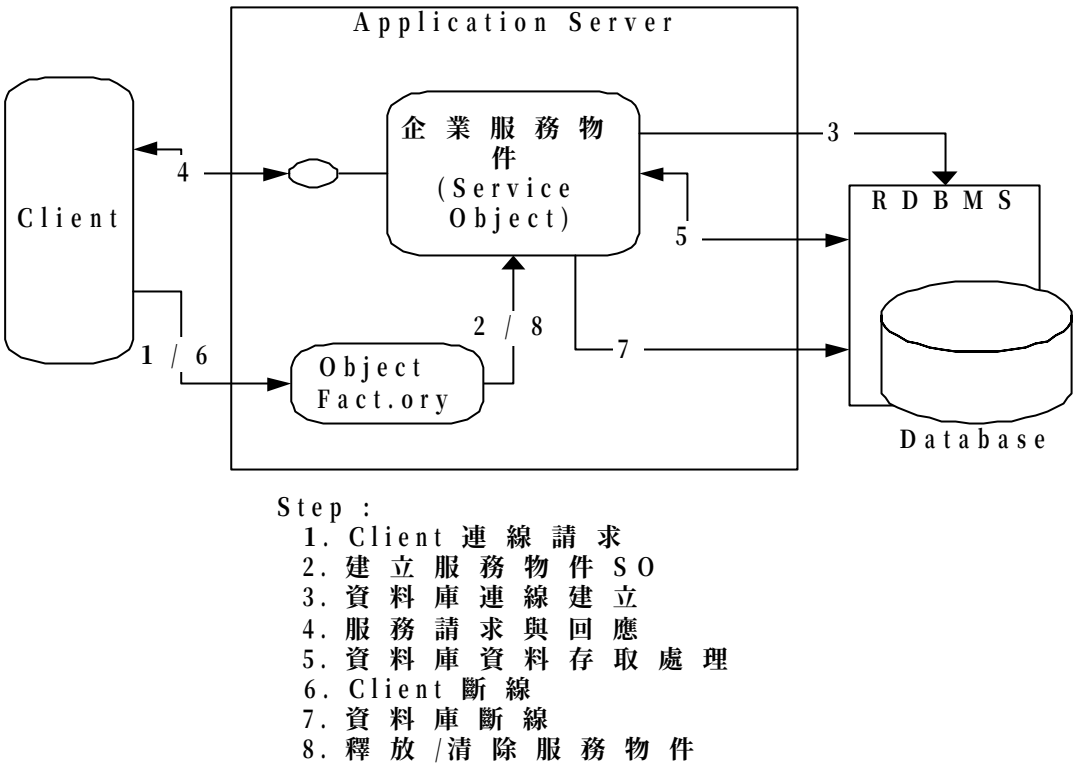
下面的章節首先介紹分析有關傳統中間層資料服務模型與面臨的問題及我們改進的模型，接著分析效能的改進程度，包含資源使用與反應時間的改進。

第三章　問題分析

三層式軟體架構之所以能更有處理效力，更具彈性、透明性，更易維護與重複使用，是因為將商業邏輯集中於中間層處理。集中化處理的結果，使得商業邏輯功能的管理與改變，更容易施行。當商業處理部分規則變更，只要調整中間層相關企業服務物件（Service Object -SO）的功能，則所有 Client 端立即得到一致性的服務。同時在一般的資訊系統中，資料庫的應用是不可或缺的一環，因此介於使用者端與資料庫系統間的應用伺服器中的企業服務物件，不可避免要與資料庫系統溝通，以便存取資料。

第一節 傳統的服務處理流程

在傳統三層式軟體架構的資料服務流程，基本上經過三大階段八個小步驟，如圖 5。Client 端能請求的服務是根據服務物件提供的介面（Interface）定義，此介面也相對說明服務物件的功能範圍。跟著我們簡略說明資料服務各階段的處理程序。



圖表 5 三層式資料服務流程

壹、 階段一：建立連線

當 Client **有服務請求時，應用伺服器接收到請求後，由一稱為物件工廠元件（**Object Factory-FO**）負責建立一獨立的企業服務物件，完成連線請求以準備提供服務。當連線建立後，所有同一連線來的** Client **端請求，一律都由此服務物件服務，服務物件可以保持服務的狀態資訊，直到服務結束。**

```
Function FO.ClientConnection:SO
Begin
   SO:=CreateSO.
  Add( SO ).
   Result:=SO.
End
```

貳、 階段二：請求服務

當 Client 端根據作業需要請求服務時，服務物件根據服務的商業邏輯規則，存取資料庫的資料。其步驟如下：

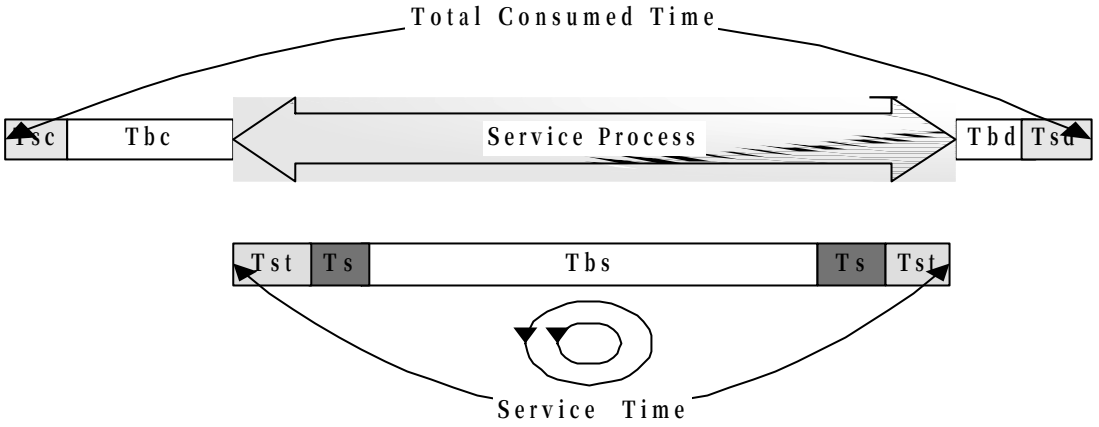
```
Function SO.DataAccess( Parameter )
begin
   if Not DB.Connected then
         DB.Connection.
    Result:=DB.Data Access( Parameter ).
End.
```

服務物件為保持服務的品質，當資料庫連線被建立後，將被維持直到 Client 請求斷線。而服務物件的資料處理策略，通常會利用資料暫存的機制（Cache），保留常用的資料於服務物件中，以便增快回應速度。

參、 階段三：請求斷線

當 Client 結束服務時，發出斷線請求，應用伺服器接收到後，由物件工廠元件將其管理的服務物件消滅，釋放資源。在服務物件消滅之前會先觸發與資料庫管理系統請求斷線事件。

```
Procedure FO.ClientDisconnection(SO).
Begin
   Remove( SO )
    SO.DbDisconnection.
    SO.Free.
End.
```



Consumed Time :
1.Tsc :Client 連線建立時間
2.Tsd :Client 斷線處理時間
3.Tbc :Database 連線建立時間
4.Tbd :Database 斷線處理時間
5.Tst :Client 與 SO 資料收送時間
6.Ts :SO 資料處理時間
7.Tbs :Database 資料處理與收送時間

圖表 6 行程耗用之時間因素

第二節 問題分析

在傳統資料服務模型中，服務物件是有狀態性（Stateful）獨占式的服務型態，每一 Client 的連線，佔用一資料服務物件，直到離線為止，因此面臨的問題有三：

壹、 耗用大量的記憶體資源

資料服務物件必須與資料庫伺服器連線，以便存取大量的資料，因此會耗用大量的記憶體資源以暫存資料表（Table），尤其當多人同時連線時，資源將快速耗盡，影響執行速度甚鉅。

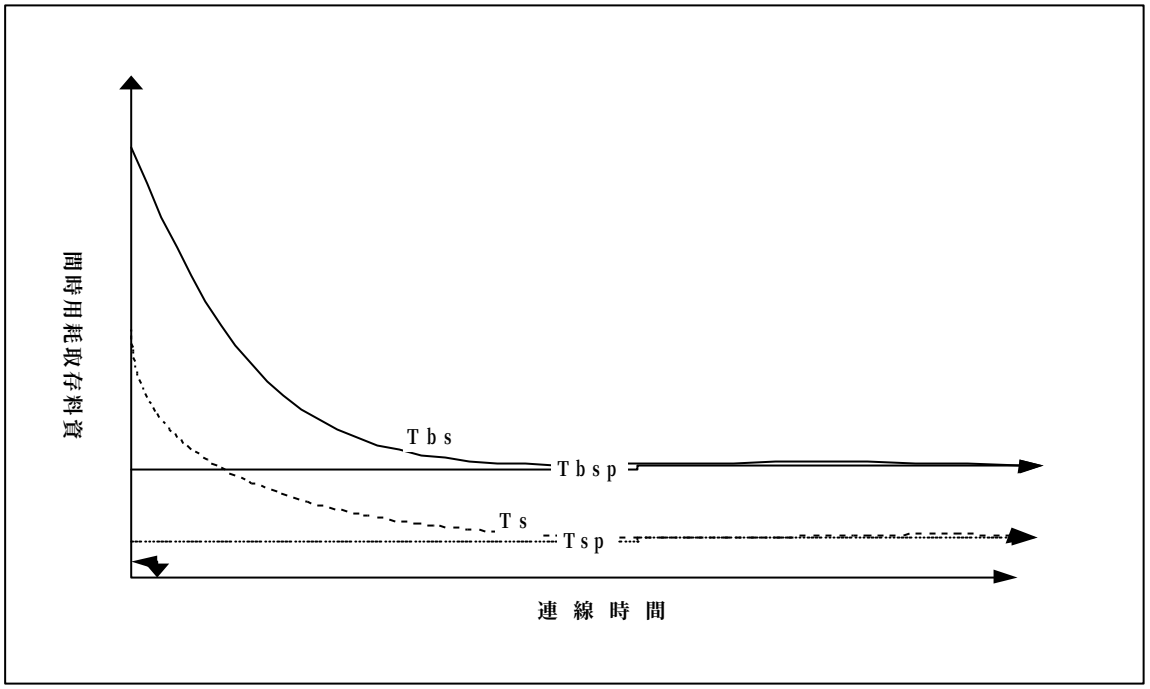
貳、 可提供連線數有限

中間層提供 Client 同時連線數目，受限於資料庫的連線數。當資料庫連線不受限時，系統又受限於服務物件本身記憶體資源的耗用程度，當記憶體耗用大時，限制了系統可建立的服務物件數。

參、 CLIENT 端的連/斷線頻率太高，影響效能

由服務物件的服務狀況分析，最耗費時間是在跟資料庫建立連線與初次存取資料，這乃是受資料庫系統管理策略的影響。對資料庫管理系統（DBMS），每一連結會建立一獨立的 Session 來管理此一連結的資料存取，根據其資料庫資料管理策略【7】，資料庫管理系統會利用暫存區（Cache）暫存最近或最常用的資料或解譯過的 SQL 語言，來加快資料存取或交易的速度。資料庫存取時間與連線時間的關係，如圖 7。

連線時間短時，Tbs 的時間長，所以高頻率的連/斷線，將使連線時間變短， 影響資料庫管理系統整體的效能甚大。



Tbs ：資料庫服務時間曲線

Tbsp :Pool 模型的資料庫服務時間曲線

Ts :服務物件的服務時間曲線

Tsp :Pool 模型的資料庫服務時間曲線

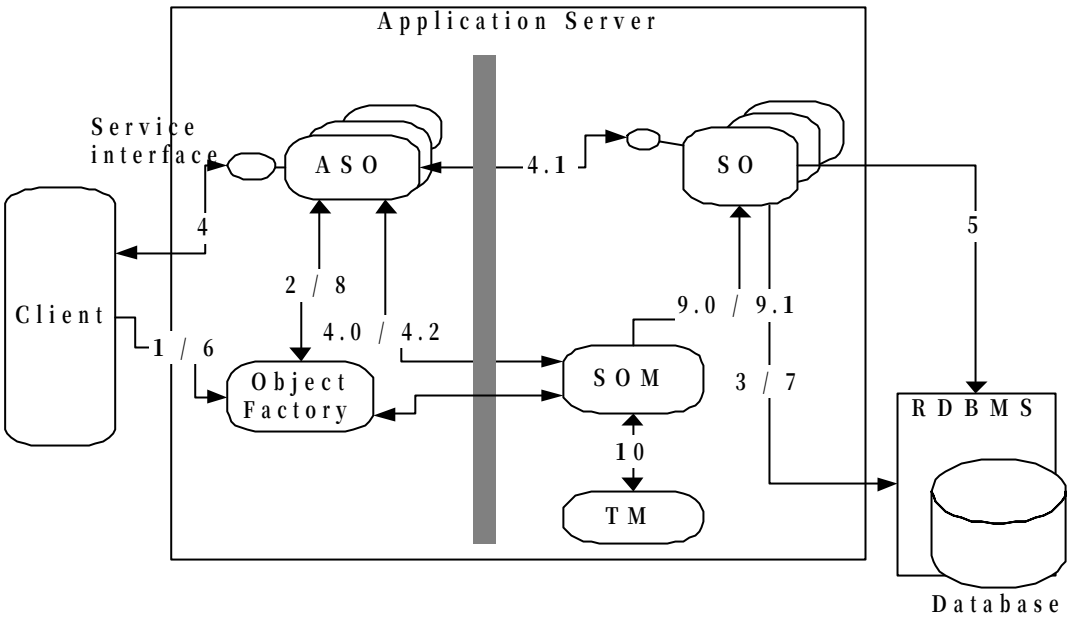
圖表 7 連線與資料存取之時間關係

第四章　改良式資料服務模型

第一節 改良的資料服務模型結構

在網際網路的環境下，當不定對象的使用者增加，將需要大量的服務物件數目服務，且造成系統連/斷線頻率升高，使整體的服務效能降低。為解決上述問題，我們提出一改良的服務模型，其結構如圖 8。

為解決前端使用者的增加時，需要服務物件，我們設計一輕量的代理服務物件（Agent Service Object－ASO）取代原來的企業服務物件 SO，專注於與 Client 端的連線，而將原來的服務物件，由服務物件管理者（SOM）利用 Pool 的管理模式控管。當 Client 端有服務請求時，由 ASO 物件負責將前端的請求轉介至 SO 服務物件服務。為了能利用物件 Pool 的管理模式，使已建立的 SO 物件，能夠重覆分派給有服務請求的 ASO 物件，必須將原來是有狀態性物件（Stateful）的 SO 物件改良設計成無狀態（Stateless）物件，而原來 SO 的服務狀態可記錄於 ASO, 或 Client 端。由於 SO 物件，能夠重覆分派給有服務請求的 ASO 物件使用，因此可減少 SO 物件的使用，節省資源。由於 SO 物件不隨 Client 端的連/斷線而生滅，故可增快反應速率。下面將說明新模型之服務處理流程。



ASO：代理服務物件
SO：服務物件
SOM：物件管理者
TM：計時管理者

Service Step：
1 > Client 連線請求
2 > 建立代理服務物件 ASO
3 > 資料庫連線建立
4 > 服務請求與回應
4.0 > ASO取得 SO 物件
4.1 > ASO轉介服務請求至 SO
4.2 > ASO歸還 SO 物件
5 > 資料庫資料存取處理
6 > Client 斷線
7 > 資料庫斷線
8 > 釋放/清除 ASO 代理物件
9.0 > 建立 SO 服務物件
9.1 > 釋放/清除 SO 服務物件
10 > 服務負載 Monitor

圖表 8 改良的資料服務模型

第二節 改良式資料模型處理流程

壹、 階段一：建立連線

當 Client 有服務請求時， Application Server 接收到請求後，由一稱為物件工廠（Object Factory-FO）的物件負責建立一獨立的代理服務物件 ASO，完成連線請求，準備提供服務。當連線建立後，所有同一連線來的 Client 端請求，一律都由此代理服務物件 ASO 服務，ASO 可以保持服務的狀態資訊，直到服務結束。

貳、 階段二：請求服務

當 Client 端根據作業需要請求服務時， ASO 收到後，首先向 SOM 取得一 SO 物件，再向 SO 請求服務，SO 將處理結果回應給 ASO，ASO 在回應給 Client 端之前，先將 SO 物件歸還 SOM 掌控後，再將最後結果回應給 Client 端。

每次服務請求時，才動態的向 SOM 要求配置 SO 物件服務，是重要的處理改變，因 SO 是一個無狀態物件（Stateless），所以 SO 可以重覆配置給有服務請求的不同 ASO 物件使用。

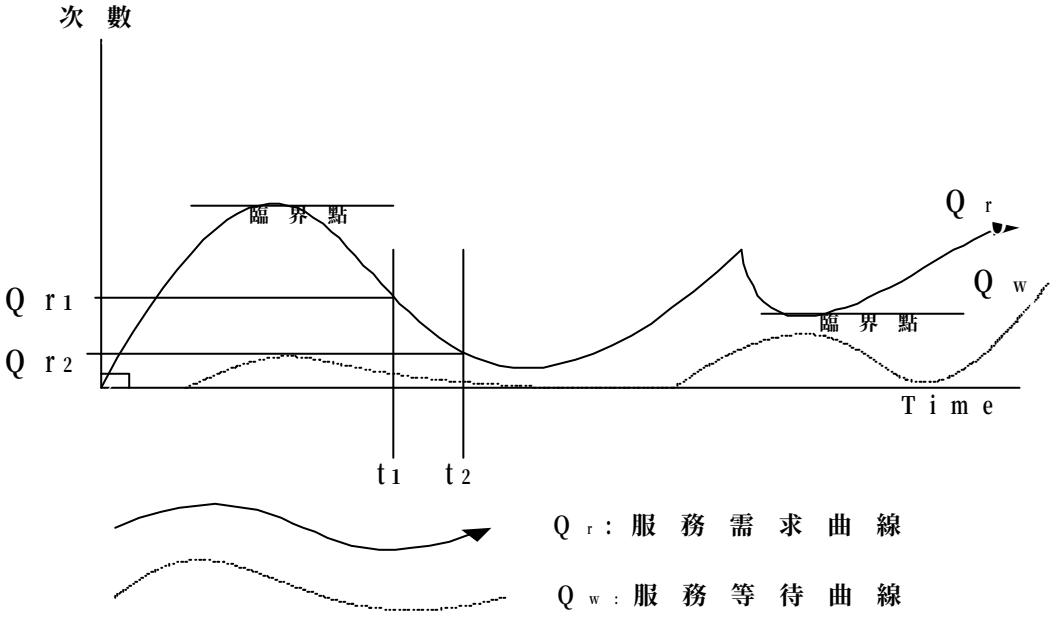
SOM 本身會根據整體最佳化策略（2.4 節）公式，管理配置待命的 SO 物件，以便能隨時應付 ASO 的需求。

參、 階段三：請求斷線

當 Client 不再需要服務時，發出斷線請求，當 Application Server 接收到後，由物件工廠將其管理的 ASO 物件消滅。此時 SO 受 SOM 管理與 ASO 的生滅無關。

第三節 整體效能的最佳化策略

根據前面章節，有關新模型利用物件 POOL 的管理策略，減少 SO 物件的使用，改進反應時間，已提昇整體相當的效能，但如何在處理效率不變的原則下，使整體資源的使用達到最佳化，我們提出一個動態管理資源的公式。



圖表 9 需求與等待曲線

當我們以分散式，負載平衡的方式，建置叢集主機（Clustered

Servers）服務時，就一般服務需求的頻率統計分析，如圖 9，假設一：服務需求雖有高低峰的時段，但常以漸進的方式達到峰值（臨界點），瞬間需求大量湧入的機率甚小。假設二：根據馬可夫鏈（Markov Chain）【4】的理論，我們假設下一階段的狀況，可以由前一階段的情況預估，也就是t_{n+1}的服務狀況，可以由t_n時推估而得，因此當我們要使t_{n+1}資源使用最佳化時，既是使服務物件使用量最小，既N_{n+1}的值最小化。我們假設在一固定的時間週期：Δt

$t_{n+1} = t_n + \Delta t$

$Q_{rn} = t_n$時服務需求量

$Q_{wn} = t_n$時等待服務量

$q_{son} = SO$物件於t_{n-1}至t_n的Δt內平均服務量

$N_{reserve}$ = 最小服務物件保留量

$N_n = t_n$時,系統存在的SO物件數

$\Delta q_{rn} = t_n$時,t_{n-1}至t_n的需求改變量

$\Delta Q_{sn} = t_{n-s}$至t_n的前s個狀態需求改變量的累計

$K_n = t_n$時需求增量率

當於t_n時需求的改變量:

$$\Delta q_{rn} = Q_{rn} - Q_{rn-1}$$

當 $\Delta q_{rn} \succ 0$, 需求漸增

當 $\Delta q_{rn} \prec 0$, 需求漸減

當於t_n時前s個狀態需求改變量的累計:

$$\Delta Q_{sn} = \sum_{k=n-s}^{n} \Delta q_{rk}$$

t_{n-1}至t_n時之增量率:

$$K_n = \left| \Delta Q_{sn} / \Delta Q_{sn-1} \right|$$

當第 n+1 **個時間區段於** Δt **區間之預估需求量：**

$$Q_{rn+1} = Q_{rn} + \Delta q_{rn} * K_n$$

SO **物件於第** n **個時間區段** Δt **區間內最大可服務量：**

$$q_{son} = (Q_{rn} - Q_{wn}) / N_n, \text{當} Q_{wn} \succ 0 \text{時}$$

當第 n+1 **個時間區段於** Δt **區間之預估** SO **物件需求量：**

$$if\left(Q_{rn+1} / q_{son}\right) > N_{reserve}$$
$$then\left(N_{n+1} = Q_{rn+1} / q_{son}\right)$$
$$Else\left(N_{n+1} = N_{reserve}\right)$$

第五章　效能改進分析

第一節 反應時間效率改進分析

根據反應時間消耗的階段，如圖 6，我們假設：

t_{sc} : *Client*建立連線
t_{sd} : *Client*斷線時間
T_s : *Service*本身處理時間
T_{sp} : *Pool*模式的*Service*處理時間
t_{st} : *Client*與*Service*資料傳送時間
t_{bc} : *DBMS*建立連線時間
t_{bd} : *DBMS*斷線時間
T_{bs} : *DBMS*資料存取與傳送時間
T_{bsp} : *Pool*模式的*DBMS*處理時間
且T_{bs}恆大於$(T_s + t_{st})$, t_{bc}恆大於t_{sc}
N_c : 第*C*次連線, 累計服務次數
U : 服務利用率 = 連線與服務總時間比
E : 服務效率改進比

壹、 傳統的反應時間

Client 一次連/斷線的平均時間：

$$TC_{so} = t_{sc} + t_{bc} + t_{sd} + t_{bd}$$

Client 一次請求服務的平均時間：

$$TS_{so} = T_{bs} + T_s + t_{st}$$

當 Client 第 c 次連線，執行 N_c 次服務請求時，總耗時：

$$T_c = TC_{so} + N_c * TS_{so}$$

當 N_c=1 **時，是最差的服務行程。**

假設連線次數為 C **時，**

系統總服務時間：

$$T_{ss} = \sum_{k=1}^{c}(TC_{so} + N_k * TS_{so})$$

系統總服務次數：

$$N_{ss} = \sum_{k=1}^{c} N_k$$

根據網際網路（WWW**），使用者瀏覽網路的特性，每一次的網頁服務請求，包含一次連/斷線的循環，故** N_k=1**。故執行** N_{ss} **次服務請求時，總耗時：**

$$T_{ss} = N_{ss} * (TC_{so} + TS_{so})$$

貳、 新模型之反應時間

根據新模型，假設永遠有 SO_p **處於待命狀態，不須再與資料庫建立連線：**

ASO **連/斷線所費的時間：**

$$TC_{aso} = t_{sc} + t_{sd}$$

SO_p **連/斷線所費的時間：**

$$TC_{sop} = t_{bc} + t_{bd}$$

SO_p **一次服務的平均時間：**

$$TS_{sop} = T_{bsp} + T_{sp} + t_{st}$$

當 Client **第** c **次連線，執行** N_c **次服務請求時，總耗時：**

$$T_c = TC_{aso} + N_c * (TS_{sop})$$

假設連線次數為 C **時，系統總服務時間：**

$$T_{ss} = \sum_{k=1}^{c} TC_{aso} + N_k * TS_{sop}$$

參、 效率改進分析

新模型的 SO_p **與原模型的** SO**，所用第** c **次服務時間之差：**

$$\Delta TS = TS_{so} - TS_{sop}$$
$$= (T_{bs} - T_{bsp}) + (T_s - T_{sp}) = \Delta t_b + \Delta t_s$$

總耗時間之差：

$$\Delta T_c = N_c * \Delta TS$$

假設連線次數為 C **時，新模型改進提昇之時間效率：**

$$\Delta T_{ss} = \sum_{k=1}^{c} TC_{sop} + N_k * (\Delta TS)$$

以網際網路的瀏覽使用特性，一對一連線服務請求時，ΔTS **的值是最大值。故當** Client **端要求** N_{ss} **次服務請求時，故節省之時間：**

$$\Delta T_{ss} = N_{ss} * (\Delta TS + TC_{sop})$$

總節省之時間是相當可觀的。

平均效率改進比 E=

傳統總服務時間-新模型總服務時間

$$= T_{ss(old)} / T_{ss(new)}$$

第二節 資源使用效率改進分析

根據上面階段的服務程序的說明，在三層式資料服務模型中，每一 Client 連線時，會獨佔一個服務物件，假設每一服務物件又會耗用一資料庫連線。一個服務物件建立時基本需要的記憶體數為 S byte，而因處理的需要，會暫存一些資料表，假設最大的服務需求時，需要使用 T 個資料表格（Table），每一 Row Data 佔用 D Byte，每一資料表暫存 R 筆，則每一服務物件需使用的記憶體數:

$$M_{so} = D * R * T + S$$

當同時存在的連線數為 Ns，則總耗用之記憶體資源:

$$M = N_s * M_{so}$$

壹、 新模型之資源使用分析

根據新模型，ASO 不負責資料庫連線與資料處理，是故 ASO 耗用的記憶體數約與服務物件（SO）建立時基本需要的記憶體數（S byte）相當，故當前端同時有 N_s 個使用者連線請求時，所需要的 N_s 個 ASO 物件所耗用之記憶體：

$$M_{aso} = N_s * S$$

又根據物件 Pool 的管理模式，假設所需的 SO_p 物件數為 N_{so}，傳統模型資源使用分析，一個 SO 物件需要 M_{so} Byte，則新模型的記憶體總耗用數：

$$M_p = M_{aso} + N_{so} * M_{so}$$

貳、 資源差異比較

新模型與傳統模型所耗用記憶體之差：

$\Delta m = M - M_p = (N_s - N_{so}) * M_{so} - M_{aso}$ **資料庫的連線數之差：**

$$\Delta N = N_s - N_{so}$$

假設平均服務利用率為 U **（服務耗用時間/連線時間），** E **：效率改進比，則** N_s **與** N_{so} **比：**

$$N_s / N_{so} = E * (1/U)$$

根據我們的實驗與經驗，此比值可達 25：1**。**

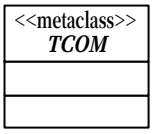
假設 Ns = 100**、** S = 500 Byte**、** Mso = 600K byte **且** Nso = 4**，則可節省之資料庫連線數**=96**，節省記憶體數約** 57.5M byte**。當** Ns **需求愈多，則資源利用改進效果更為顯著。**

第六章　實作模型設計

第一節 類別定義與說明

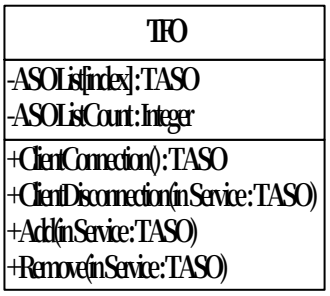
壹、 TCOM 類別:

實作 COM 物件介面與協定之類別【1】【2】【3】，此類別為 TASO（代理物件類別）與 TSO（服務物件類別）二個 COM 元件的父類別, 如果欲實作 CORBA 服務物件則可重新定義 TCOM 類別符合 CORBA 規範的介面與協定。



貳、 TFO：代理物件工廠類別

TFO 為 ASO 物件的管理者，負責管理前端的連斷線請求，基本上提供：



1. ClientConnection:前端連線請求
2. ClientDisconnection: 前端斷線請求
3. ADD:將新建立的 ASO 物件新增至物件庫（ASOList）
4. Remove：將請求斷線的 ASO 物件從物件庫（ASOList）移除

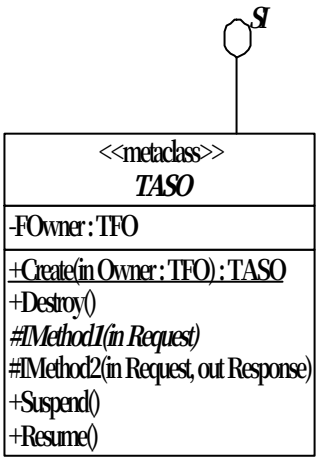
參、 SI:服務介面的定義

此 SI 介面為抽象介面（abstract Interface），根據實際之服務不同而有不同的介面定義，在 ASO 物件與 SO 物件必須依介面定義實作，也就是說 TASO 為所有代理物件的父類別，TSO 為所有服務物件的父類別。在範例中 SI.Imethod1 與 SI.Imethod2 的介面方法必須在代理物件與服務物件中實作。

肆、 TASO:代理物件的父類別

負責前端連線與服務請求，當連線時被建立，斷線時被釋放。

方法（Method）：



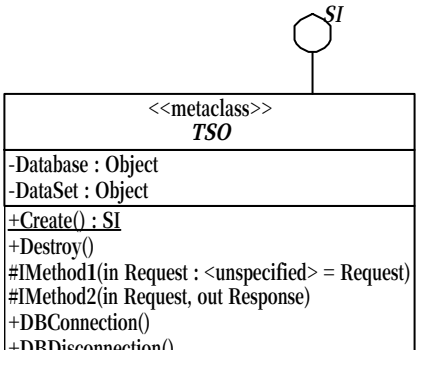
1. Imethod1,Imethod2：服務介面方法實作。

2. Suspend: 當服務請求時，SOM 無閒置的 SO 物件可用時（busy）， ASO 物件執行緒暫停執行，直到 Resume 被呼叫。

3. Resume：當 SOM 有閒置的 SO 物件且 ASO 有待處理的請求時，呼叫 Resume，Wakeup 執行緒。

伍、 TSO:服務物件父類別

負責資料庫連線與資料服務請求，由 SOM 根據服務統計資訊，控管建立、釋放與分派使用。資料存取處



理依不同的服務需求，而有不同的實作方式。

方法（Method**）：**

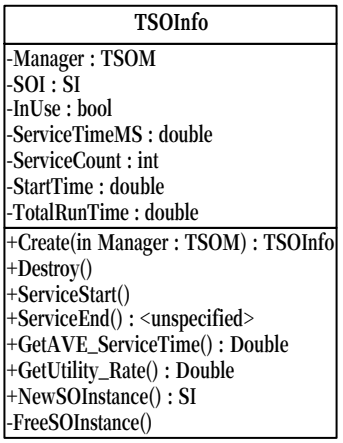
1. Imethod1,Imethod2**：服務介面方法實作。**
2. DBConnection/DBDisconnection**：為資料庫連線處理之抽象方法**
3. DataAccess**：為資料庫資料存取處理之抽象方法**

陸、 TSOINFO:服務統計資訊類別

每一資訊物件對應一服務物件, 負責記錄個別 SO **的服務狀態與統計資訊。當呼叫** Create **方法建立時，會同時建立服務物件。呼叫** Destroy **方法時，同時建釋放服務物件。**

特性值（property**）：**

1. InUse: **物件使用狀態旗標（**flag**），**True=**物件使用中。**
2. ServiceTime**：累計所有服務請求時，所使用之時間計數，單位**: **微秒（**MS**）。**
3. ServiceCount**：累計所有服務請求次數。**
4. StartTime**：服務物件建立時的時間。**



5. TotalRunTime**：服務至目前之執行時間，單位**: **微秒（**MS**）。**

6. SOI:**對應的資料服務物件（**SO**）之介面。**

方法（Method**）：**

1. ServiceStart：**服務請求時，啟動服務計時與設定使用中狀態。**
2. ServiceEnd：**服務結束時，累計統計資訊與設定非使用中狀態。**
3. New/Free SOInstance：**建立與釋放** SO **資料服務物件。**
4. GetAVE_ServiceTime：**平均每次服務時間量＝**ServiceTime÷ServiceCount**，單位**: **微秒（**MS**）。**
5. GetUtility_Rate：**服務利用率＝**ServiceTime÷TotolRunTime**。**

柒、 TSOM：服務物件管理者類別

服務物件管理者，根據系統啟動與結束而生滅，負責管理控制執行中的服務物件，統計服務資訊並監督服務狀態，動態配置服務物件數。

特性值（property**） ：**

1. SOList[Index]：**記錄目前執行中的**

TSOM
-SOList[Index] : TSOInfo -Current_Qso : Integer -WaitQueue[Index] : TASO -Current_Qwait : Integer -Current_Qr : Integer -Last_Qr : Integer -Last_DeltaQr : Integer -Max_Qso : Integer +Reserve_Qso : Integer -Qson : Integer +TotalServiceTime : double +TotalServiceCount : Integer +Standard_AVEServiceTime : double
+Create() : TSOM +Destroy() -FindUnuseSO() : TSOInfo -FindSOinfoByInterface(in SOI : SI) : TSOInfo -CaculateNext_Qso() : Integer -TimeReset() -WaitForSOObject() -SuspendToWaitQueue(in ASO : TASO) -WakeupFromWaitQueue() -AddSoInfo() -RemoveSOInfo() +LockSO(in ASO : TASO) : SI +ReleaseSO(in SOI : SI) +TimerAction() +GetAVE_ServiceTime() : Double

資訊物件。

2. Current_Qso：**目前執行中的資訊物件數。**

3. WaitQueue[Index]：**記錄等待執行服務請求的代理物件。**

4. Current_Qwait：**目前等待執行的服務請求數。**

5. MAX_Qso：**最大可建立的服務物件數，可限制總資源的使用。預估之服務物件數必須小於此數。**

6. Reserve_Qso：**保持執行狀態的服務物件數，必須小於** MAX_Qso**。**

7. Qson：**時間週期內每一服務物件的平均服務量。**

8. Standard_AVEServiceTime：**每次服務的標準時間需求，與** AVE_ServiceTime **比較，可衡量服務水準，系統可依此資訊分散服務負載。**

9. TotalServiceTime：**累計所有服務物件的服務請求，所使用之時間計數，單位**: **微秒（**MS**）。**

10. TotalServiceCount：**累計所有服務物件的服務請求次數。**

11. Current_Qr：**當下時間週期的服務請求量，** DeltaQr = Current_Qr － Last_Qr**。**

12. Last_Qr：**上一時間週期的服務請求量。**

13. Last_DeltaQr：**上一時間週期的服務請求增減差異量，提供**

下一週期的預估服務請求量的基礎值。

方法（Method**）：**

1. Lock/Release SO：**當** ASO **請求服務時，利用** LockSO **取得一可用的服務物件為其服務，當完成後呼叫** ReleaseSO **歸還並記錄服務物件的服務統計資訊。**

2. WaitForSOObject：**等待一閒置的服務物件，當無閒置的服務物件可用時，必須將請求放入等待序列（**WaitQueue**），暫停執行，直到服務物件被歸還。**

3. SuspendToWaitQueue：**將請求放入等待序列（**WaitQueue**），暫停執行。**

4. WakeupFromWaitQueue：**從等待序列中取得一待處理之請求，恢復執行。**

5. FindUnuseSO：**從** SOList **中，尋找一閒置的服務物件。**

6. FindSOInfoByInterface：**根據服務物件介面，從** SOList **中，反查** SOInfo **物件。**

7. ADD/Remove SOInfo：**將新建或消滅釋放的** SOInfo**，從** SOList **中加入或移除。**

8. TimerAction：**時間週期到期時，必須處理的事項如預估下一週期的服務請求量，調整服務物件量，監督與通知負載能力。**

9. CaculateNext_Qso：**根據記錄資訊，依公式預估下一週期的服務請求量，計算需要的服務物件量，並配合** MAX_Qso,Reserve_Qso **值，動態調整。**

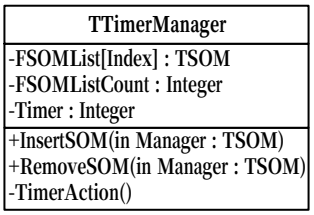
10. TimeReset：**重置時間週期的計數、記量值如** Last_Qr、Last_DeltaQr、Current_Qr。

11. GetAVE_ServiceTime：**平均每次服務時間量＝** TotalServiceTime÷TotalServiceCount，**單位**: **微秒（**MS**）。**

12. GetUtility_Rate：**服務利用率＝** TotalServiceTime÷ TotolRunTime。

捌、 TTIMERMANAGER：時間週期控制物件類別

時間週期控制物件，是一計時控制器，當計時到期時，會觸發 TimerAction **事件通知，根據登錄資料，個別通知之** SOM，**（既呼叫** SOM.TimerAction**）。**



特性值（property**）　：**

1. SOMList[Index]、SOMListCount：**記錄登錄的物件管理者（**SOM**）。**

2. Timer：**時間週期設定值。**

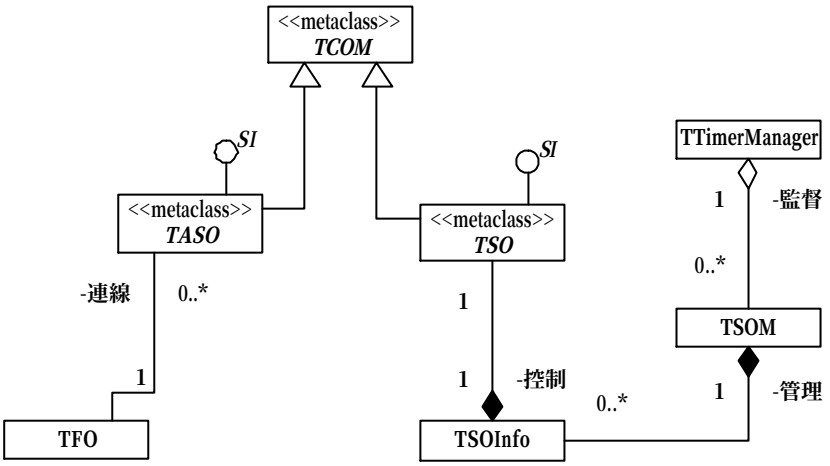
方法（Method**）：**

1. Insert/Remove SOM：**登錄/移除物件管理者。**

2. TimerAction：**時間事件處理，通知登錄之** SOM，**既呼叫** SOM.TimerAction。

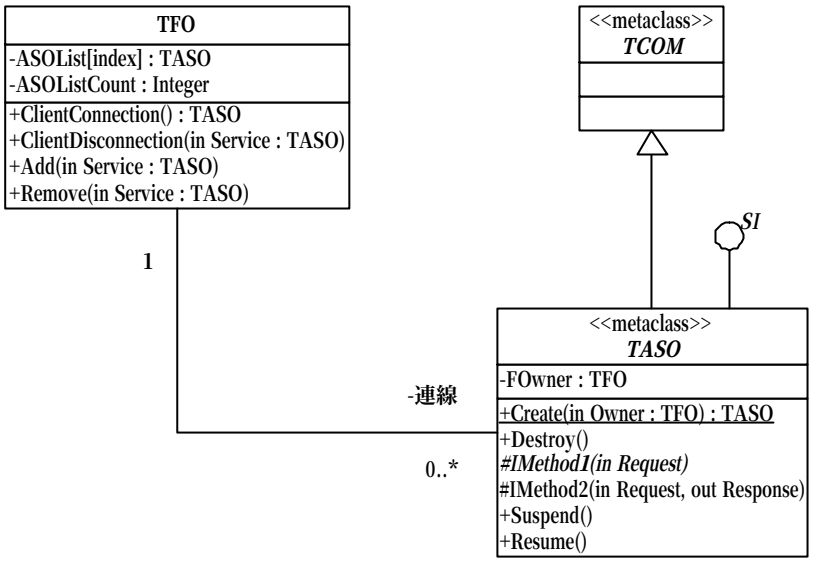
第二節 類別關係圖

壹、 類別



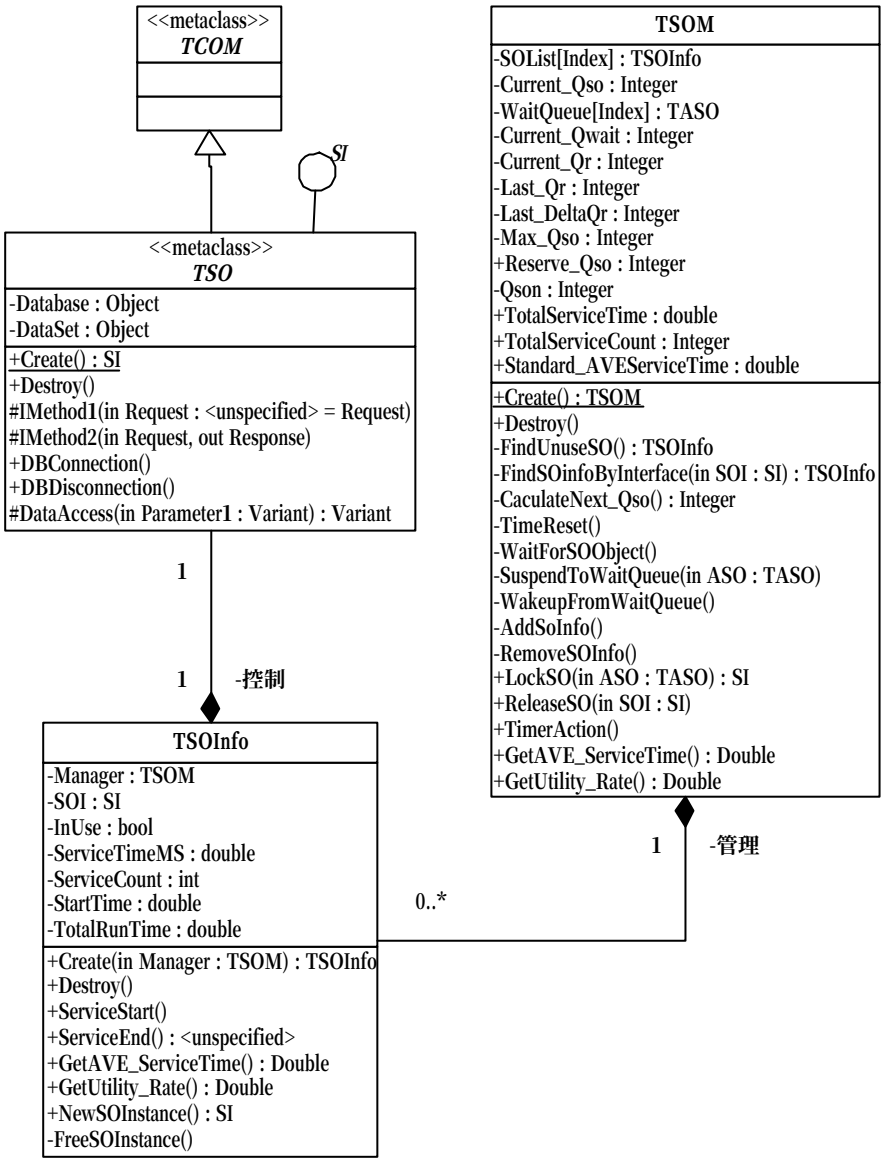
圖表 10 類別關係圖

貳、 代理物件類別關係圖



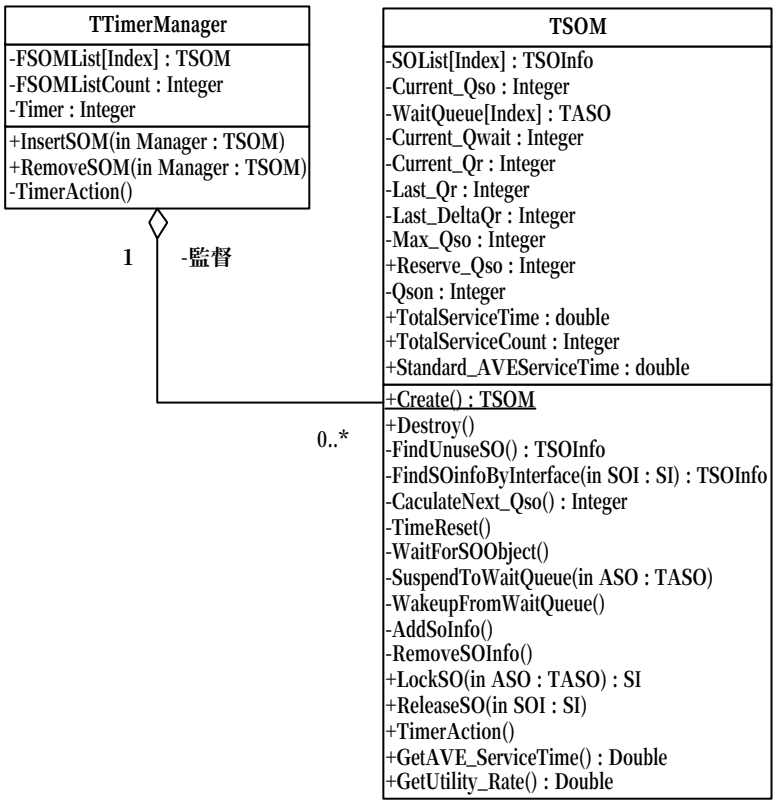
圖表 11 代理物件類別關係圖

參、 服務物件與管理者類別關係圖



圖表 12　服務物件與管理者類別關係圖

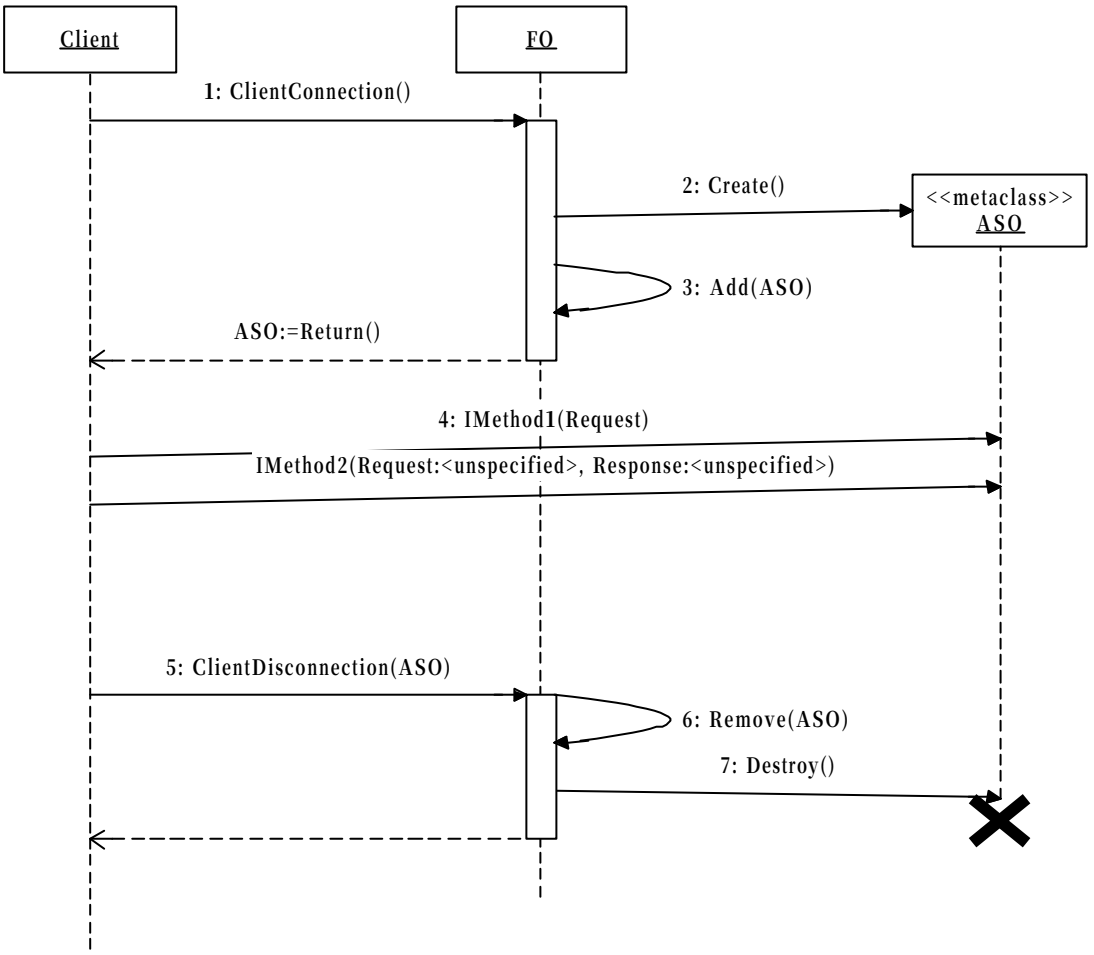
肆、 時間週期控制類別關係圖



圖表 13 時間週期控制類別關係圖

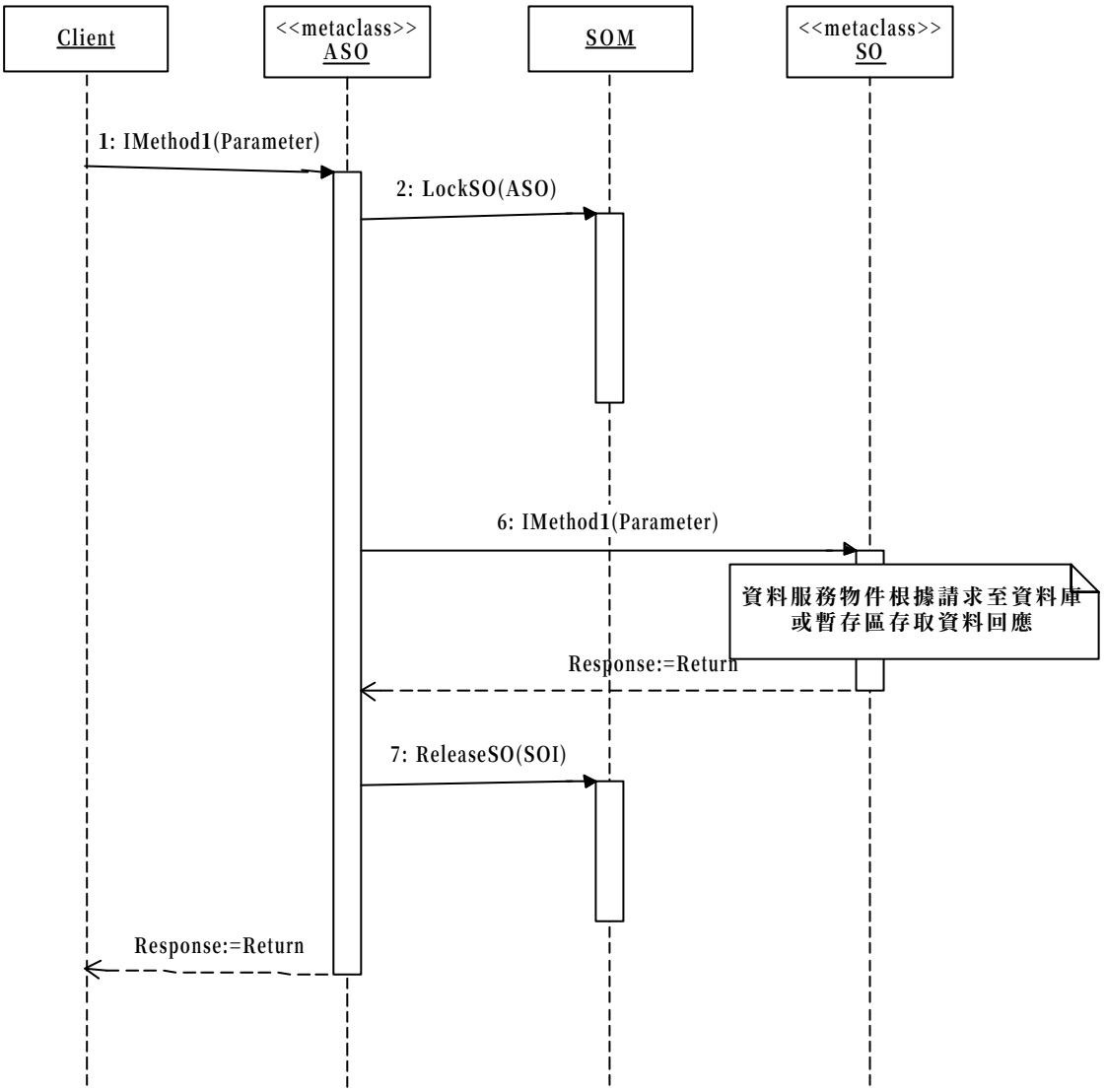
第三節 循序圖（SEQUENCE DIAGRAM）

壹、 前端連線



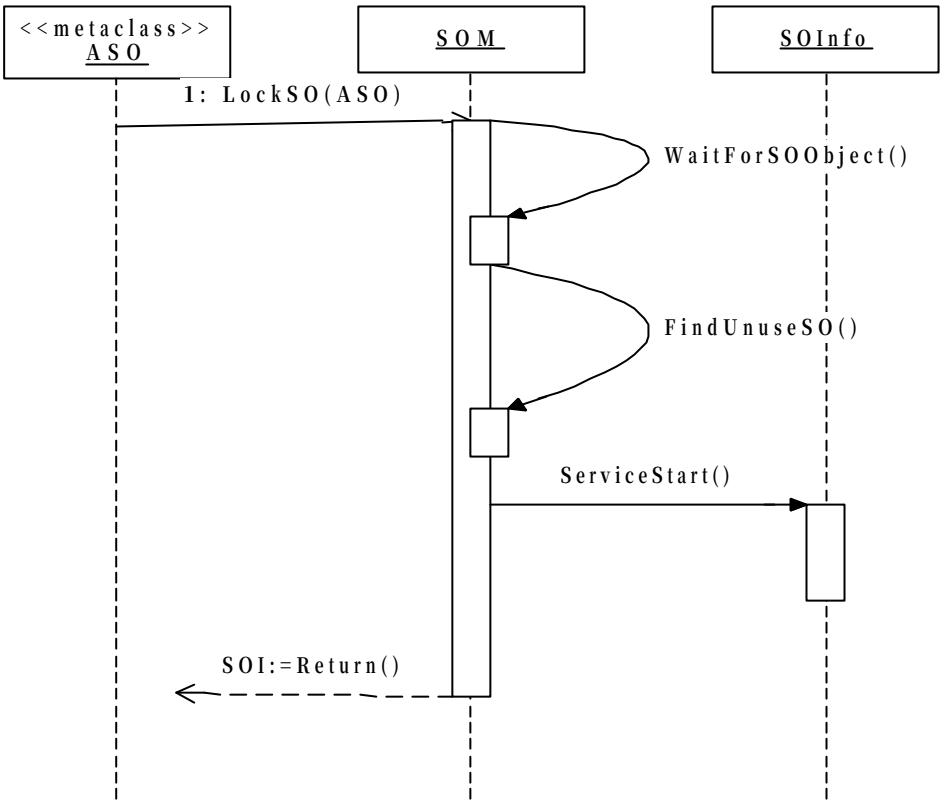
圖表 14 連線循序圖

貳、 服務請求



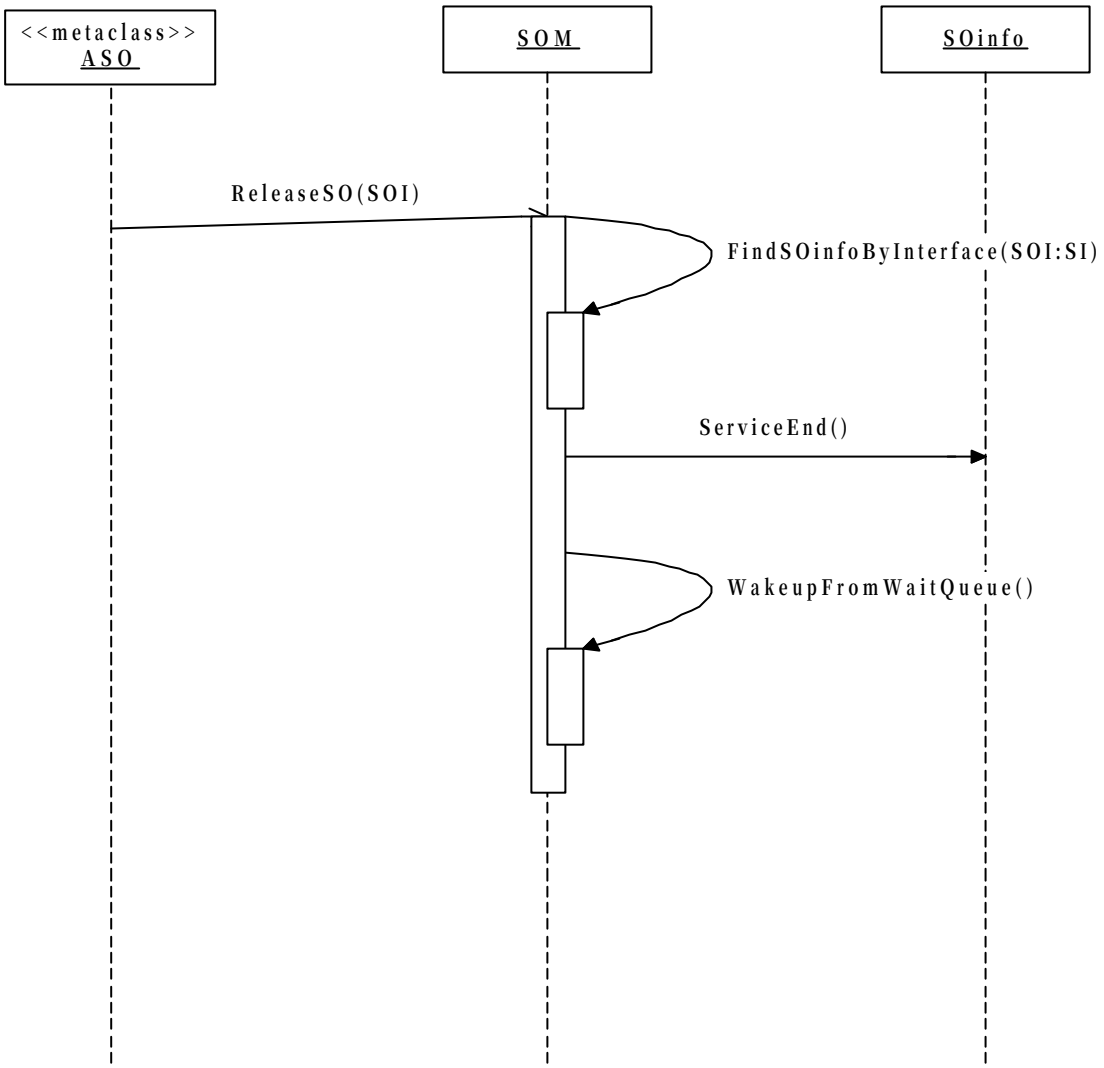
圖表 15 服務請求循序圖

參、 ASO 取得一可用的服務物件



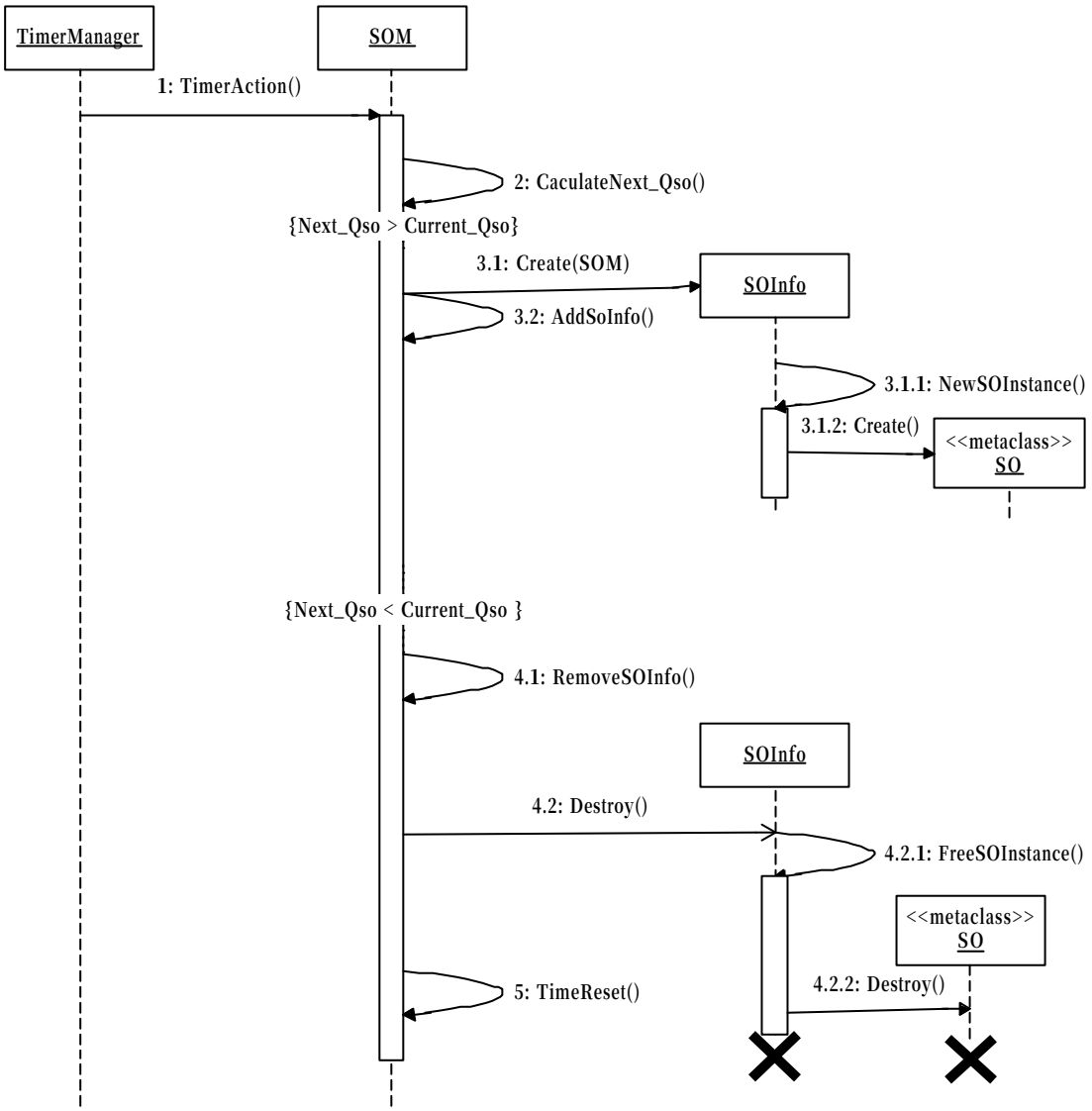
圖表 16 SOM 分派服務物件循序圖

肆、 ASO 歸還服務物件



圖表 17 SOM 收回服務物件循序圖

伍、 計時控制事件



圖表 18 時間事件處理循序圖

第四節 類別方法虛擬程式碼

壹、 TASO:代理物件類別

```
//服務介面方法 1 於代理物件的實作
procedure TASO.IMethod1(Request: Variant);
var SOI:SI;
begin
   //取得可用之服務物件
   SOI:=SOM.LockSO;
      //請求服務物件 IMethod1 服務
      SOI.IMethod1(Request);
   //歸還服務物件
   SOM.ReleaseSO(SOI);
end;
```

```
//服務介面方法 2 於代理物件的實作
function TASO.IMethod2(Parameter: Variant): Variant;
var SOI:SI;
begin
   //取得可用之服務物件
   SOI:=SOM.LockSO;
      //請求服務物件 IMethod2 服務
       Result:=SOI.IMethod2(Parameter);
   //歸還服務物件
   SOM.ReleaseSO(SOI);
end;
```

貳、 TSO：資料服務物件類別

```
//服務介面方法 1 於服務物件的實作
procedure TSO.IMethod1(Request: Variant);
begin
   //檢查連線
  if Not DataBase.Connected then DBConnection;
   //資料存取服務
  DataAccess( Request );
end;
```

```
//服務介面方法 2 於服務物件的實作
function TSO.IMethod2(parameter: Variant):Variant;
begin
   //檢查連線
  if Not DataBase.Connected then DBConnection;
   //資料存取服務
   Result:=DataAccess(parameter );
end;
```

參、 TSOINFO：服務統計資訊類別

```
//建立 SOInfo 物件
constructor TSOInfo.Create(AManager: TSOM);
begin
   Manager:=AManager;
   //物件建立時之時間
   StartTime:=Now;
   //建立服務物件
   NewSOInstance;
end;

//消滅釋放 SOInfo 物件
destructor TSOInfo.Destroy;
begin
   //釋放服務物件
   FreeSOInstance;
   Free;
end;
```

```
//建立 SO 物件
procedure TSOInfo.NewSOInstance;
begin
   SOI:=TSO.Create;
end;

//釋放 SO 物件
procedure TSOInfo.FreeSOInstance;
begin
   SOI.Free;
end;
```

```
//服務開始
procedure TSOInfo.ServiceStart;
begin
   //記錄服務起始時間
   SBTime:=Now;
   //設定未使用狀態
   Inuse:=True;
end;

//服務結束
function TSOInfo.ServiceEnd: Double;
var ST:Double;
begin
   //計算服務時間
   ST:=Now-SB;
//累計服務時間,次數
   ServiceTime:=ServiceTime+ST;
  increment( ServiceCount );
   Result:=ST;

   //設定未使用狀態
   InUse:=False;
end;
```

肆、 TSOM：服務物件管理者類別

```
//物件建立時設定初值
constructor TSOM.Create;
begin
   //平均每一服務物件的服務量
   Qson:=(TimeManager.Timer/Standard_AVEServiceTime);
   Current_Qr:=0;
   Last_Qr:=0;
   Last_DeltaQr:=MAX_INTEGER;
   TotalServiceTime:=0;
   TotalServiceCount:=0;
   //建立 Default 服務物件
   TimerAction;
end;
```

```
//利用物件介面，反查資訊物件
function TSOM.FindSoinfoByInterface(SOI: SI): TSoinfo;
var i:Integer;
begin
  for i:=0 to Current_Qso-1 do
   begin
      if SOList[i].SOI=SOI then
       begin
          Result:=SOList[i];
          exit;
       end;
   end;
   Result:=Nil;
end;

//尋找一未使用的物件
function TSOM.FindUnUseSO: TSoInfo;
var i:Integer;
begin
  for i:=0 to Current_Qso-1 do
   begin
      if Not SOList[i].InUse then
       begin
          Result:=SOList[i];
          exit;
       end;
   end;
   Result:=Nil;
end;
```

```
//等待一可用之服務物件
procedure TSOM.WaitForSOobject(ASO: TASO);
begin
  SuspendToWaitQueue(ASO: TASO);
end;

//確認暫停服務請求
procedure TSOM.SuspendToWaitQueue(ASO: TASO);
begin
  //檢查服務物件資源是否被用完
  if Semaphore =0 then
   begin
     //資源用完,放入等待 QUEUE 中
     PutToWaitQueue(ASO);
     // ASO 暫停執行
     ASO.Suspend;
   end;
end;

//恢復待處理之服務請求
procedure TSOM.WakeupFromWaitQueue;
var ASO:TASO;
begin
  //檢查是否有請求待處理之代理物件
  ASO:=GetFromWaitQueue;
  if ASO<>Nil then
   begin
     //待處理之代理物件, 恢復執行
     ASO.Resume;
   end;
end;
```

```
//等待，尋找，分派一可用之服務物件給請求者
function TSOM.LockSO(ASO: TASO): SI;
var SOinfo:TSOInfo;
begin
   //等待一可用資源-服務物件
    WaitForSOobject;
    SOInfo:=FindUnuseSO;
    SOInfo.ServiceStart;

   //控制資源量
  Decrement( Samphore );
    Result:=SOInfo.SI;
end;

//請求者歸還、釋放使用之服務物件
procedure TSOM.ReleaseSO(SOI: SI);
var SOInfo:TSOInfo;
       ST:Double;
begin
   //服務物件被歸還
    SOInfo:=FindSOInfoByInterface(SOI);
    ST:=SOinfo.ServiceEnd;

   //累計統計值
    TotalServiceTime:=TotalServiceTime+ST;
  increment( TotalServiceCount );
  increment( Current_Qr );

   //控制資源量
  increment( Samphore );
end;
```

```
//預估下一週期之物件需要量
function TSOM.CaculateNext_Qso: Integer;
var Next_Qso:Integer;
begin
  //  平均每一服務物件的服務次數
   if Current_Qwait>0 then
      Qson:=(Current_Qr-Current_Qwait)/Current_Qso;

   //預估下一週期的服務請求量
   Current_DeltaQr:=Current_Qr-Last_Qr;
   K:=Current_DeltaQr/Last_DeltaQr;
  if K>0 then Next_Qr:=Current_Qr+K*Current_DeltaQr
   else  Next_Qr=Current_Qr-K*Current_DeltaQr;

   //預估下一週期的服務物件量,受控於最大可用資源量
   Next_Qso:=Next_Qr/Qson+Reserve_Qso;
  if Next_Qso>Max_Qso then Next_Qso=Max_Qso;
   Result:=Next_Qso;
end;
```

```
//監督資源需求，動態調整服務物件需要量
procedure TSOM.TimerAction;
begin
   //預估下一週期服務物件需要量
   Next_Qso:=CaculateNext_Qso;
  if Next_Qso>Current_Qso then
   begin
      //增加服務物件量
     while Current_Qso<Next_Qso do
      begin
         SOinfo:=TSOInfo.Create;
        AddSoinfo( SoInfo );
        increment( Samphore );
      end;
   end
  else if Next_Qso<Current_Qso then
   begin
      //嘗試減少服務物件量,(物件未在使用中方可釋放)
     while Current_Qso>Next_Qso do
      begin
         SOInfo:=FindUnuseSO;
        if SOInfo=Nil then exit;
         RemoveSOinfo(SOInfo);
         SOInfo.Destroy;
        decrement( Samphore );
      end;
   end;
   //重設週期變數值
   TimeReset;
end;

//重設週期變數值
procedure TSOM.TimeReset;
begin
   Last_DeltaQr:=Current_Qr-Last_Qr;
   Last_Qr:=Current_Qr;
   Current_Qr:=0;
end;
```

伍、 TTIMERMANAGER: 時間週期控制類別

```
//建立時間控制器
constructor TTimeManager.Create(ATimer:Integer);
begin
   //設定計時週期
   Timer:=ATimer;
end;

//計時週期事件處理
procedure TTimeManager.TimerAction;
var i:Integer;
begin
   //通知每一物件管理者，處理週期事件
  for i:=0 to SOMListCount-1 do
SOMList[i].TimeAction;
end;
```

第七章　結論與建議

未來以網路化為主的資訊系統中，雖然硬體效率的改進日新月異，但對各種不同的服務需求，如何在跨平台，資訊交換機制透明化下，即時的提供正確快速的服務。以及如何讓系統設計簡化，能重覆應用，是資訊科技努力的重要方向。在資訊科技領域中永遠面臨兩大問題：技術與內容。就目前，以服務內容來說欠缺像 IC 元件所定義的標準介面訊號規格書，以致不能像 IC 元件自由組裝。但就技術面來看，標準化已經有相當的進展，如最近提出的以 WEB 為資訊服務平台，讓服務物件化，透過 SOAP 的服務協定，以 XML 為文件內容或資訊交換標準，就是所謂的 WEB SERVICE 網路資訊系統服務建置架構。

WEB SERVICE，既在強化中間層資料服務專門化的設計理念，因此善用我們提出的改進模型，可以提昇整體資訊服務品質。對服務物件本身的設計來說，應要詳細考慮下列幾點因素：

1. 即時性資料的一致性，完整性。
2. 服務的量能比率關係。
3. 資料量與網路頻寬的關係。
4. 資料存取頻率與系統整體資源的關係。

在網路的環境中，服務資料的存取要朝輕量，ON Demand 的角度思考，設計以無狀態物件為主的服務物件，方能使系統運作

順暢的，提供最佳的服務品質。

參考文獻

1. Borland， Delphi 5-資料庫應用程式設計手冊
2. Borland， Delphi 5-MIDAS 入門與實作手冊
3. 李維， Delphi 4.x 實戰篇 1,2，旗標出版，台北，民國 87 年 11 月
4. 陳坤茂，作業研究，二版，華泰文化，台北，民國 87 年 7 月
5. Alan Pope， The COBRA Reference Guide Understanding the Common Object Request Broker Architecture，MA：ADDISON WESLEY，1998
6. Borland， Delphi 5 Developer`s Guide，INPRISE
7. C.J.Date ，An Introduction To Database Systems，MA:ADDISON – WESLEY ，7nd，2000
8. Kornel Terplan，Communication networks management，PTR Prentice Hall ，2nd
9. Simon Bennett，Steve McRobb and Ray Farmer，Object－Oriented Systems Analysis and Design using UML，THE McGRAW-HILL COMPANIES，2000
10. Wolfgang Emmerich.，Engineering Distributed Objects，MA：Wiley 2000

11. David S. Platt，(The Essence of COM and ActiveX) COM/ActiveX 完全自學手冊，蔡武男譯，初版，和碩科技，台北，1998/12，民國 87 年 11 月

12. Alain Conchon，"Applications and Services"，Alcatel Telecommuncation Review， 3rd，2001，p187-188

13. Amund Aarsten etal.，"Patterns of Three-tier Client-Server Architectures"，http://members.aol.com/kgb1001001/articles/threetier/threetier.html

14. Boundless Technologies International B.V，"Thin-Client/Server Computing" A New era in computing，Thin-Client/Server Whitepaper.

15. Carnegie Mellon Software Engineering Institute，"Three Tier Software Architectures" Software Technology Review，http://www.sei.cmu.edu/str/descriptions/threetier_body.html

16. P.Emerald Chung etal.，"DCOM andCORBA Side by Side，Step by Step，and Layer by Layer"，http://www.cs.wustl.edu/-schmidt/submit/Paper.html

17. Per-Ake Larson、Murali Krishnan，" Memory Allocation for Long –Running Server Applications"

18. TimesTen Performance Software，"In-Memory Data Management in the Application Tier" A new Strategy for High performance internet Computing，http://www.timesten.com

自傳

本人姓曾名清義，現就讀於南華大學資訊管理學系研究所，並任職於南華大學資訊室系統開發組組長，負責校務行政自動化系統的規劃與執行的工作，專長於制度流程的規劃、系統設計、整合工作，研究方向：物件導向的分析與設計技術，分散式多層次系統架構，商業服務物件模型分析，系統開發技術的研究。