Part 1 of 3

序中有亂：
系統整合之思維和技術

大綱

前言

整合之目的是創造圓滿的整體(whole)，圓滿的整體來自和諧互動的個體(part)。例如，汽車和行人是兩種個體，經常在路上發生衝突，導致社會不和諧了。

為了創造和諧，人們在追求「整」體圓滿之下，努力安排個體的溝通「合」作。因而設計出紅綠燈與斑馬線，成為汽車與行人的接口，只要確實遵守接口(的規則)，個體自然整合起來，社會圓滿和諧了。

無論在自然界或我們生活中，「接口」處處是整體和諧之基礎，也是系統整合之關鍵。因此，本書針對信息系統，闡述接口之涵義，從實務解析中，深刻體會接口設計之真諦。

無論是接口之涵義或是接口設計之真諦，都必須在陶醉在實際案例和情境中，才能心領神會之。所以，

1 新的整合工法與心法

信息系統愈來愈複雜，然而整合工法愈來愈進步，信息產業逐漸成熟，就如同建築業一般，萬丈高樓如雨後春筍般，井然有序的鋼骨結構，支撐著複雜多元之功能。新的整合工法有如下特色：

1. 合乎人們習慣，
2. 發揮人的天賦，

3. 有力落實心法，
4. 人人皆能 DIY。

簡單而有效，人人能之。本章將介紹精緻的進步工法，包含：

工法 1 ----- 善用接口(Interface)，
工法 2 ----- 以 Façade 樣式(Pattern)實現接口，
工法 3 ----- 不斷重複 Façade 樣式，無限成長。

同時，也說明工法如何落實心法(即接口設計之決策過程)，包括：

心法 1 ----- 亂中有(找)序，抽象出組件(Part)之共同接口。
心法 1 ----- 序中有變化(亂)，締造整體(Whole)之接口。

俗語說：科學家從亂中找序，而設計師(藝術家)則是創造序來容納變化(亂)。無論是「亂中有序」或「序中有亂」，兩者都要呈現序(Order)，並包容亂(Change)，只是手段不同而已。兩種手段都能帶來巨大經濟價值，精通這兩種心法，是接口設計和系統整合之成功關鍵。

亂中有序

「亂中有序」意味著，從一群有些相似的物件中，分析其變與不變，然後抽象出一致的接口，呈現出某種序。例如：

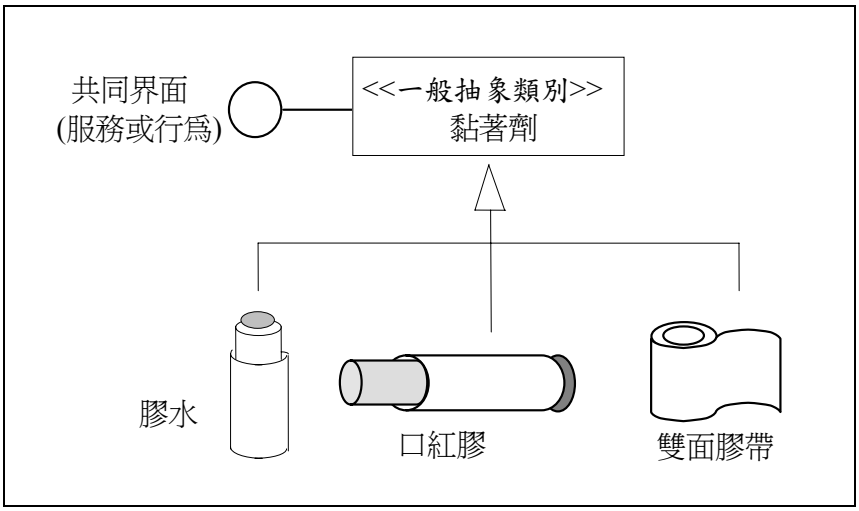


圖 1　從族群找出序(一致的行爲)

這是針對一個族群 (Family) 找出共同的服務 (Service) 或行爲 (Behavior)，然後強調這同一族群之兄弟姊妹之間相互替換性。於是不同家族之間，各成員就能透過接口而一拍即合了。這種分析與抽象是科學家們「亂中找序」的慣用手法。

序中有亂

設計師(含建築師、藝術家)，正好與科學家互補，他們透過綜合或整合的手藝，而創造序來容納亂(變化)。例如，信封袋可以把信紙、楓葉、照片等異質性的東西包裝起來，提供一個簡單的整合接口、一致的格式，讓郵差容易攜帶和處理。再如，上圖裡的膠水家族的成員，可以跟剪刀、針線等異質性的物件整合起來，成爲一個工具袋，然後以「Facade 樣式」落實之，如下圖：

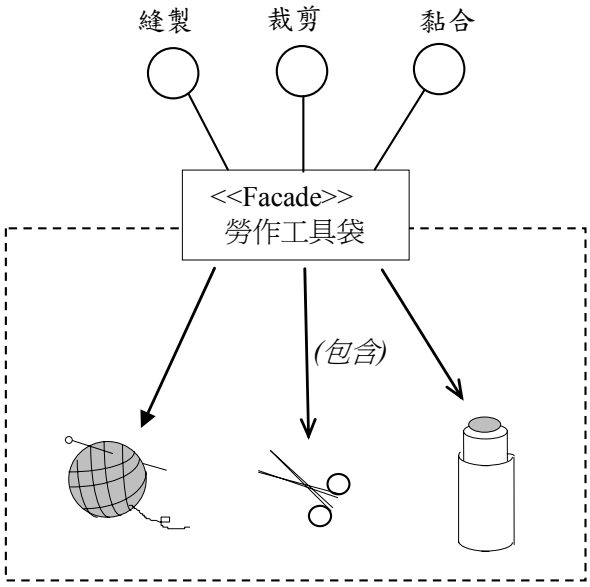


圖 2　異族合作之整體行爲(整合之序)

在設計接口時，上述兩種方法常常交替使用。「亂中有序」注重於抽象(Abstraction)與分析(Analysis)，其目的在於「分」，分出變與不變，表現於抽象類別與具體類別之「分」，也表現於具體類別之「分」(例如分成吹風機、電磁爐等)。「序中有亂」注重於綜合(Synthesis)，其目的在於「合」，創造某種序(Order)來包容或整合混雜多變的內涵。

例如貨櫃、電視機外殼等，都是「序中有亂(變化)」的作品。這如同自然界的蝦子等動物，在蝦殼之內充滿複雜的蘊藏。再如高樓大廈的鋼骨結構(Architecture)，更是「序中有亂」的代表作。這如同自然界的脊椎動物，以脊椎支撐複雜多變的機制或活動。

[創造「序」來容納變化]

誕樹(序)上掛上各式各樣的聖誕禮物、燈飾(變化)。還有，當您設計一個衣架或書架時，就是創造了序；而當您掛上衣服或擺上書籍時，就是以序包容變化了。

2 系統整合工法之演進

在 1970-80 年代裡，是資料庫(Data Base)流行時期，從資料庫來整合各應用系統(Application)，是當時流行的整合技術。茲做個比喻，兩個系統互相整合御溝通，就如同兩座四合院互相整合溝通，後院就如同 Data Base，於是這個時期，是藉由「後院相連」方式達到整合溝通之目的。如下圖：

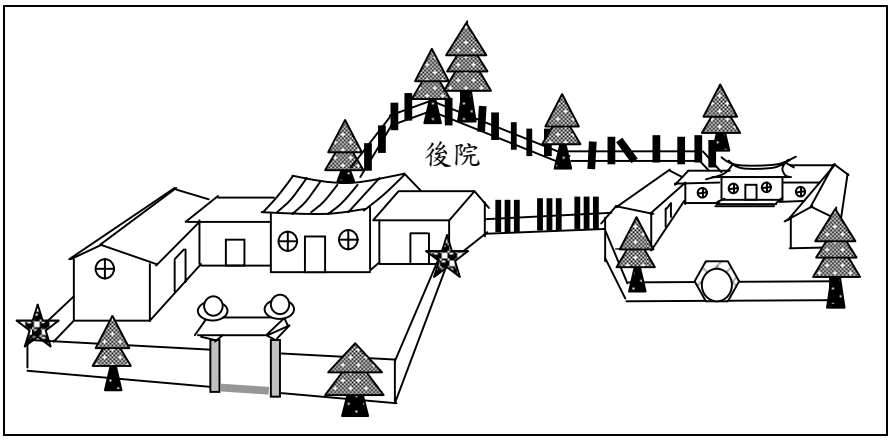


圖 3　資料庫層(Tier)之整合

由於資料庫整合方式，需要修正資料庫的藍圖(Schema)，不得不更改眾多相關的應用程式，費時費錢又費力。到了 1990 年代，CORBA 等標準提出之後，逐漸流行以 Middleware 將兩個系統的 AP Server 互相溝通；中間層的 AP Server 就如同四合院的中庭，於是這個時期，是藉由「中庭相連」方式達到系統整合與溝通。如下圖：

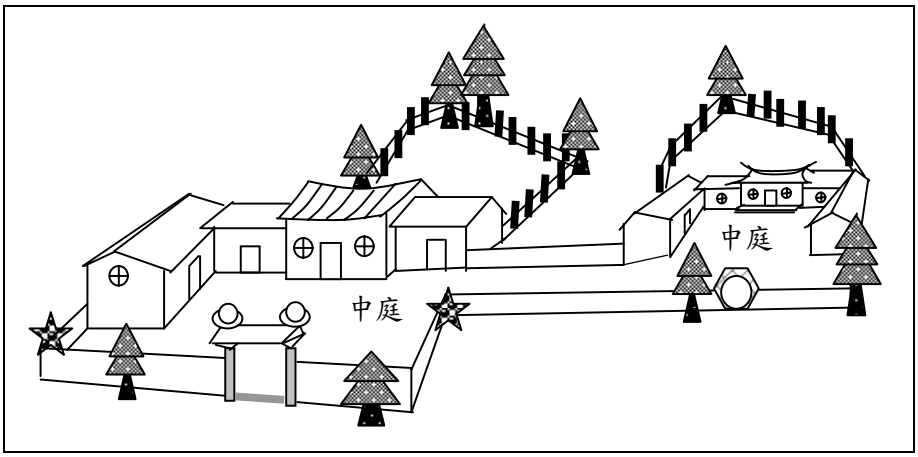


圖 4　中間層之整合

由於中間層 AP Server，都是由幾家互相競爭的大廠所提供，其設計架構及外接接口都差異很大。因之像 Orbix 等 Middleware 變得複雜又昂貴，系統整合費時又費力。到了 1997 年 Internet 流行起來，逐漸轉變爲從前端溝通整合，如下圖：

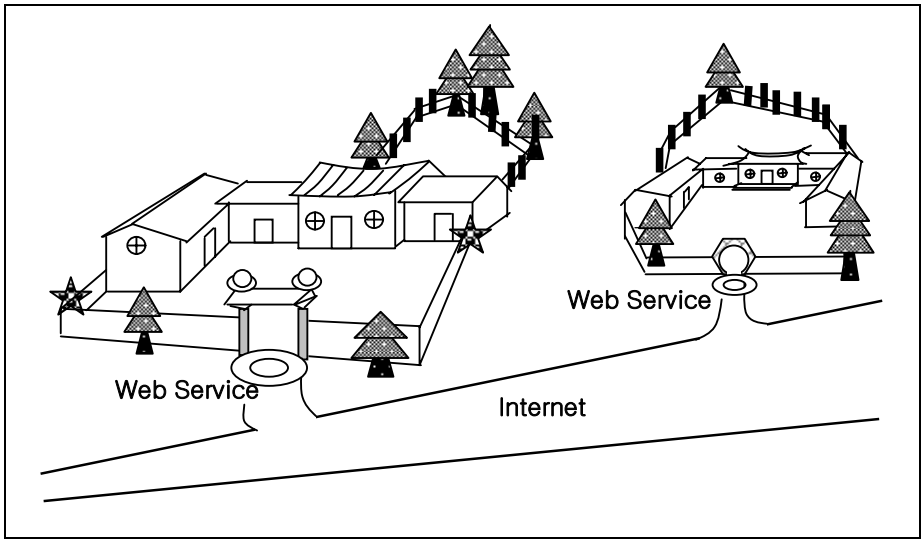


圖 5　Web 層之整合

這是藉由「前門相連」方式達到整合溝通之目的；家家戶戶都有標準的 Web Service 接口，並經由門前的 Internet 高速公路互相溝通，通稱爲 SOA(Service-Oriented Architecture)整合方法；這也就是本書現在採用的整合方式。

1.3 Web 時代的整合工法

在 1990 年代流行的 Client-Server 系統環境，是透過共用的資料庫來整合各軟體系統，一個系統做完其任務時，就將處理完成的資料存入資料庫，間接地傳遞給別的系統。如下圖所示：

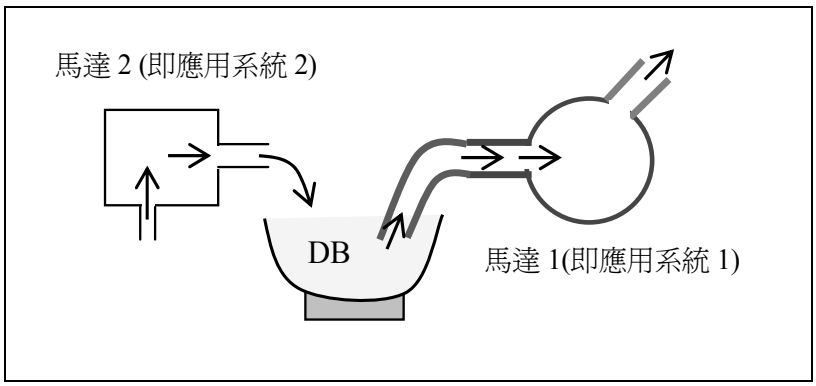


圖 6　藉由 Database 整合

因此許多人習慣先整合出一個整體性的資料庫(Universal Database)，然後進行分工，由不同開發團隊各自開發不同的系統或子系統(Subsystem)。此種軟體思維讓我們建構出眾多的『草本植物』型的系統，像稻子一般許多葉子(即 Application)直接銜接到在泥土裡(即 Database)。此種思維習慣和分工策略並不能充分發揮 Web 的巨大聯結(Connection)潛能，沒有充分利用門前的 Internet 高速公路。

在當今的潮流裡，卻反過來認爲各系統或自系統完全封裝(Encapsulate)自己的資料庫或資料來源(Data Source)，並不跟別的系統或子系統共用。然後，藉由 Web Service 來促成系統之間的直接傳遞資料與溝通。如下圖：

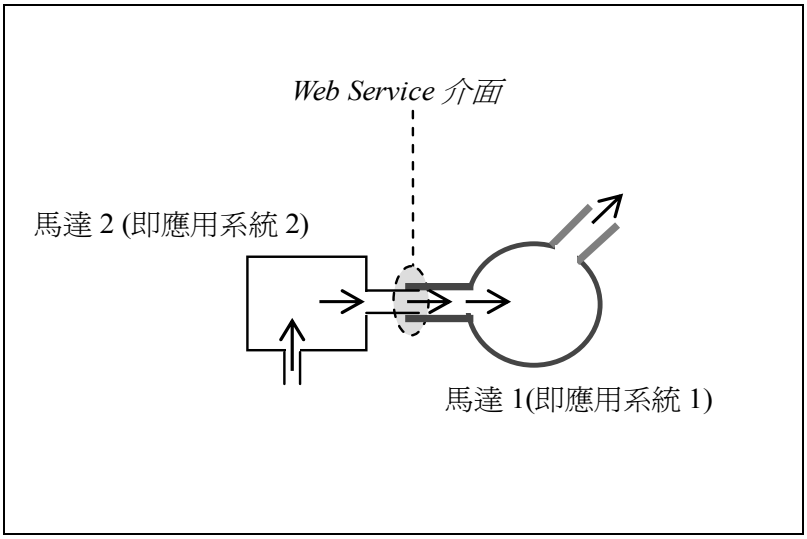


圖 7　藉由 Web Service 整合

基於此新的願景(Vision)，讓我們能採取新的系統開發途徑，建構出眾多的『阿里山神木』型的系統。此新願景帶來了許多益處，包括：

1) 各系統易於分散到不同地區，讓企業能彈性成長。
2) 各系統易於視其需求而選擇不同的平台(如.Net 或 J2EE)或資料庫系統。
3) 各系統易於外包分工、採取平行開發(Parallel Development)，大幅縮短系統開發時間。

Web Service 是一種軟體系統對系統之間(Program-to-Program)的溝通接口。它不是打前鋒的使用者接口(User Interface)，而是擔任後衛角色，提供後勤的服務來協助潛在的 Client 程式。就如同電腦網路專家 Billy Hollis 所說：

> "In Web Services, software functionality becomes exposed as a service that doesn't care what the consumer of the service is."
> (從 Web Service 來看，軟體系統的各項功能，都成爲公共的服務，通常並不知道該服務的使用者是誰。)

Web Service 非常有助於整合後端分散於各地的資源，同時也提供標準的接口，讓其資源也易於跟別的資源相互整合，匯集出更爲巨大的服務能量。Billy Hollis 又說：

> "Web Services allow developers to build applications by combining local and remote resources for an overall integrated and distributed solution."
> (Web Service 讓軟體人員能聯結本地及遠距的各種資源，成爲整合性及分散性的應用系統。)

這種「只賣牛仔褲，但不淘金」的心境，跟古典 Client-Server 系統的建構程序有極大差別。過去，開發 Client-Server 系統時，都是先由分析使用者的需求開始，並假設能捕捉到相當完整而穩定的需求，然後根據這些特定需求而開發出軟體系統的功能和服務。如今這項思維習慣已經不再適用了，而必須在未確知 Client 是誰的情境下開發出 Server 端的共用性服務；這種習慣和心境的轉換之衝擊是蠻巨大的。

雖然剛才提到過 Web Service 並不知道特定的使用者是誰，但是跟裁縫師一樣地會針對特定的顧客群，而不是完全不知道。設計 Web Service 就是要把服務賣給這個特定的客戶群，Web Service 幕後的標準協定 (Protocol)，如 XML/SOAP 等讓服務的供需雙方都共蒙其利。如下圖：

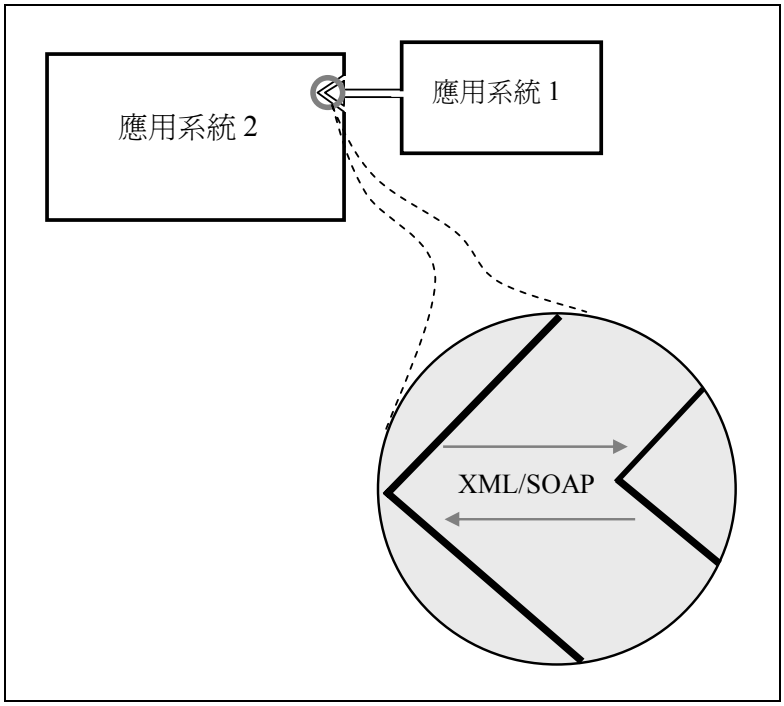


圖 8　XML/SOAP 標準協定支持 Web 接口

無論是在 Internet (如 HTTP)或專線網路(如 RPC)上，位於不同地區的異質性系統都能互相溝通與整合。請注意，一般談到 Web Service-based 接口整合時，其意味著是採取 Web Service 標準協定，其通訊並不限定於 HTTP，在一般的 RPC/LAN 環境裡也能使用 Web Service 標準協定。當整合的雙方都能接受 Web Service 標準協定時，就能一拍即合、分合自如了。

4. 人人皆會的整合工法

4.1 工法 1：善用接口

接口(Interface)是一種規範(Specification)，表達兩個系統如何互相整合的一種共識(Agreement)。例如十字路口的「紅綠燈與斑馬線」，就是行人與汽車雙方的接口；當此接口發揮其權威時，社會(即人、車的整合體)呈現出井然有「序」(Order)；反之，當接口失去權威時，整合就出問題了。因之，善用接口是系統整合的基本工法。

1.4.1.1 接口與整合

系統整合之目的是：讓系統模組一拍即合，而且易於新陳代謝。接口(Interface)機制是實現這項承諾的基礎。從我們週遭的生活實例去觀察，會較容易瞭解接口之涵意。例如家中牆壁上的插頭接口：

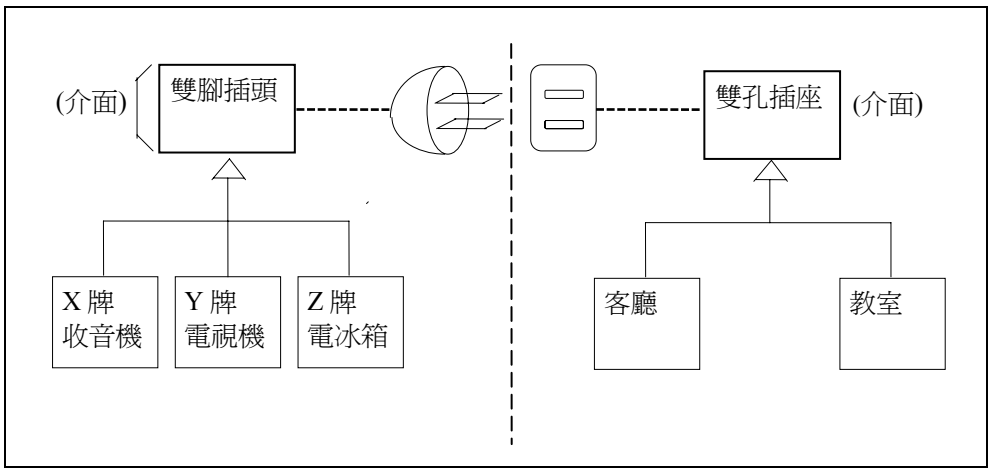


圖 9　您所熟悉的「插頭/插座」接口

上圖裡，X 、Y 、Z 牌的電視機皆提供雙腳插頭。而 A 、B 皆爲雙孔插座。透過接口能帶來許多方便，例如 Z 可插在 A 上，也可將 Z 拔出來換插在 B 上； X 可隨時插到 A 或 B。所以共可有 XA、XB、YA、YB、ZA、ZB 六種搭配，極具彈性。

X 、Y 、Z 三者具有共同接口。A 、B 具有共同接口。雙方接口能相容合作，就創造出極大彈性了。插頭與插座兩個族群能各自發展，只要接口一致，就能相輔相成。例如電器業者可以推出更多種的電器，則電器體系就會無限成長了，如下圖所示。

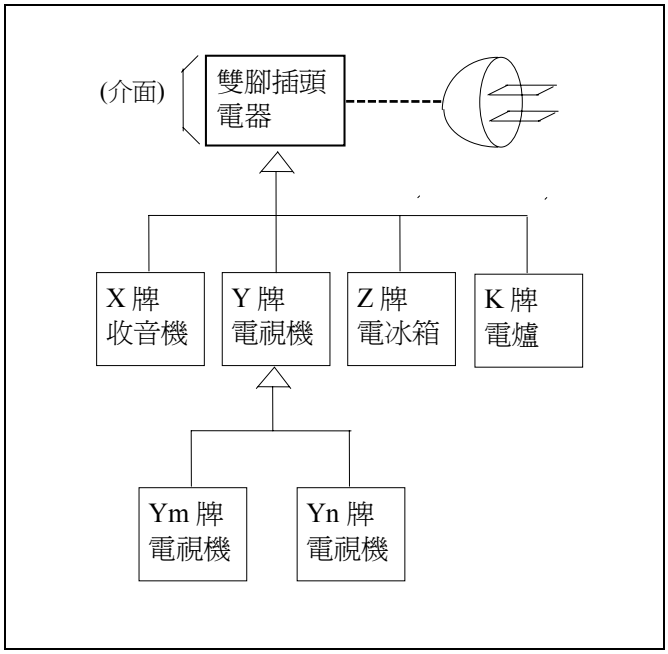


圖 10 「插頭」體系之擴充

由於 Ym、Yn 及 K 皆與 X 、Y 、Z 的接口一致，都提供雙腳插頭。所以皆可買回家插在 A 或 B 牌插頭上。同理，插座族群亦能擴充如

下圖 11 所示。這兩族群在獨立成長的過程中，其接口維持不變，保持其相容性。所以新型電器如 Ym，可插在新型的 Bq 插座上。當然，新舊也仍可搭配，如 X 可插在 Br 上。這是從消費者的立場著想的，此外對生產者也很有好處，電器設計者不必考慮到插座背後的構造及變化，只要接口一致就行了。另一方面，插座背後的電源結構設計者不必費心去瞭解電器的種類及構造，只要提供一致接口：兩支腳的插頭就行了。

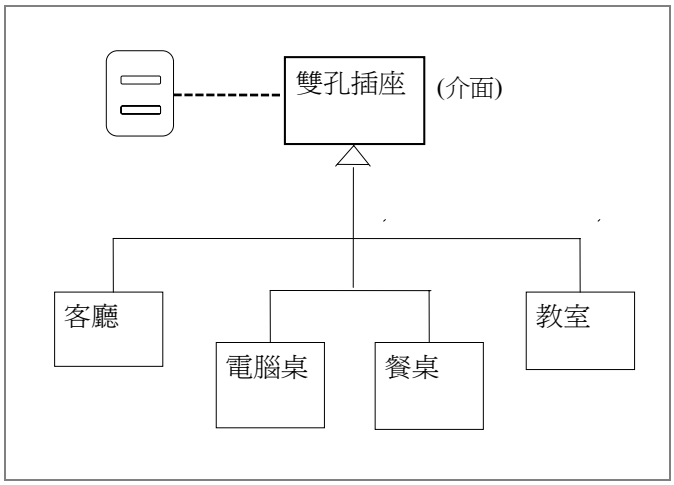


圖 11 「插座」體系之擴充

因此，消費者、電器製造者及插座製造者三方皆獲得益處。

4.1.2 DIY：動手落實爲 VB.Net 接口

我們可以輕易地以 VB.Net 落實上述的接口，茲以販賣機爲例：

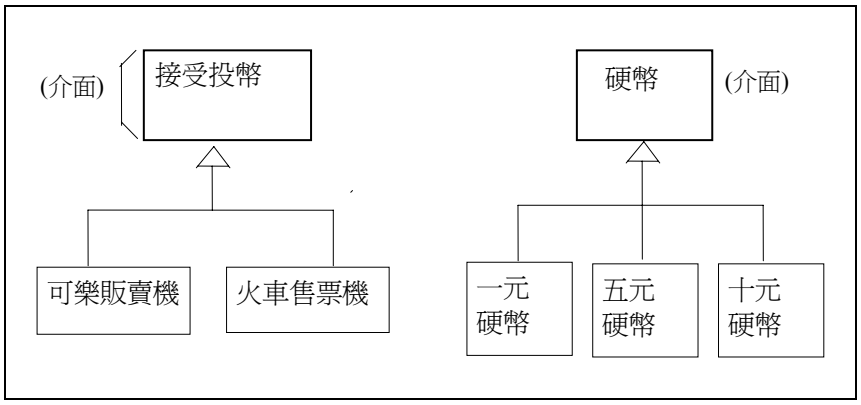
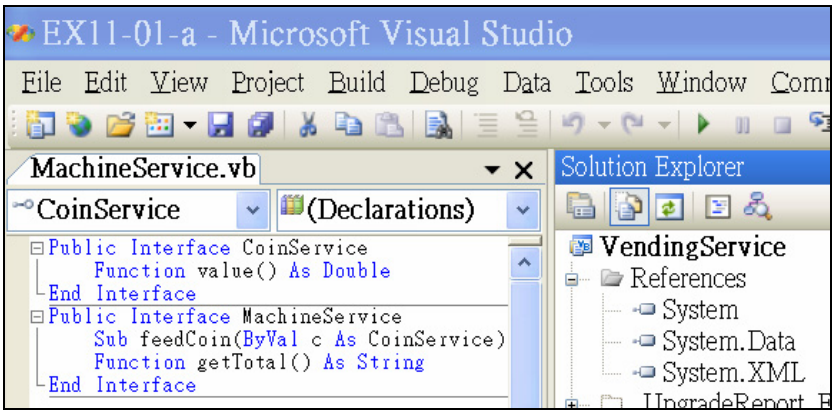


圖 12 「販賣機與硬幣」體系之接口

「可樂販賣機」和「火車售票機」有共同的接口──可讓消費者投入硬幣，且會計算投入的總金額。所以硬幣與售票機、販賣機都能一拍即合。現在就來動手 DIY 吧! 將上圖落實爲 VB.Net 程式碼的步驟如下：

Step-1: 先設計及定義 MachineService 和 CoinService 接口

於是，建立一個 Class Library: EX01-01-a 項目，內容爲：



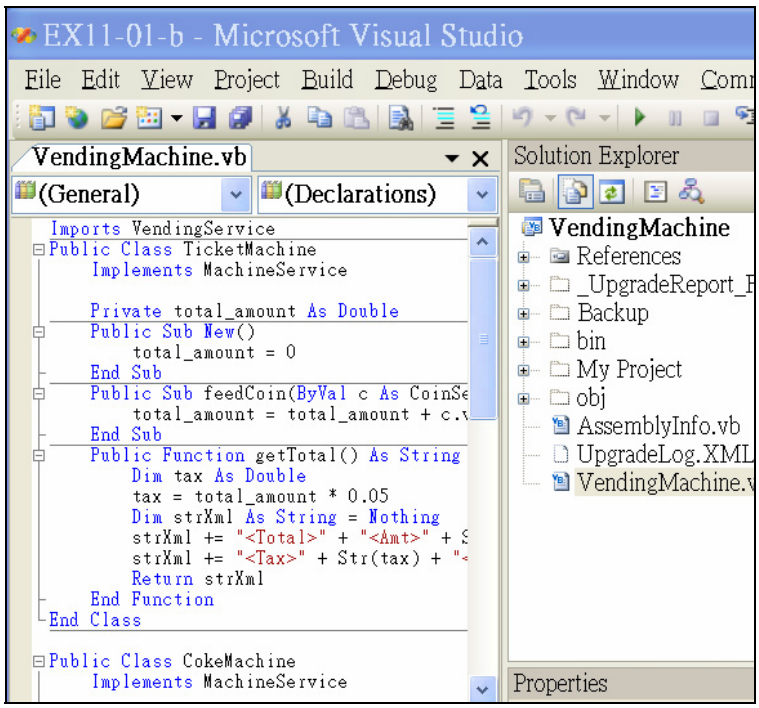
VB.Net 程式碼如下：

```
' EX01-01-a
' MachineService.vb
Public Interface CoinService
    Function value() As Double
End Interface
Public Interface MachineService
    Sub feedCoin(ByVal c As CoinService)
    Function getTotal() As String
End Interface
```

使用 VS.Net 從這項目產生一個 VendingService.dll 程式庫文檔。

Step-2: 定義販賣機類別來支持 MachineService 接口

於是，建立一個 Class Library: EX01-01-b 項目，內容爲：



使用 Add Reference 來參考 VendingService.dll 接口定義檔。VB.Net

程式碼如下：

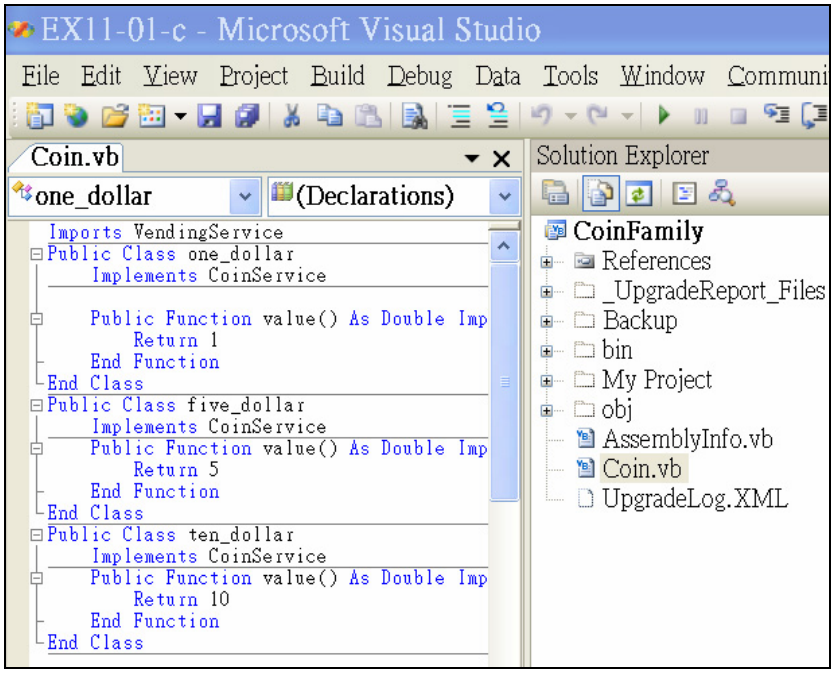
```
'EX01-01-b
'VendingMachine.vb
Imports VendingService
Public Class TicketMachine
    Implements MachineService

    Private total_amount As Double
    Public Sub New()
        total_amount = 0
    End Sub
    Public Sub feedCoin(ByVal c As CoinService)
                Implements MachineService.feedCoin
        total_amount = total_amount + c.value()
    End Sub
    Public Function getTotal() As String Implements MachineService.getTotal
        Dim amt, tax As Double
        tax = total_amount * 0.05
        Dim strXml As String
        strXml += "<Total>" + "<Amt>" + Str(total_amount - tax) + "</Amt>"
        strXml += "<Tax>" + Str(tax) + "</Tax>" + "</Total>"
        Return strXml
    End Function
End Class

Public Class CokeMachine
    Implements MachineService

    Private total_amount As Double
    Public Sub New()
        total_amount = 0
    End Sub
    Public Sub feedCoin(ByVal c As CoinService)
                Implements MachineService.feedCoin
        total_amount = total_amount + c.value()
    End Sub
    Public Function getTotal() As String Implements MachineService.getTotal
        Dim strXml As String
```

```
            strXml += "<Total>" + "<Amt>" + Str(total_amount) + "</Amt>"
                    + "</Total>"
            Return strXml
        End Function
End Class
```

使用 VS.Net 的 Rebuild 功能，從這項目產生一個 VendingMachine.dll 程式庫文檔。

Step-3: 定義合乎 CoinService 接口之販賣機類別

於是，建立一個 Class Library: EX01-01-c 項目，內容爲：



使用 Add Reference 來參考 VendingService.dll 接口定義檔。VB.Net 程式碼如下：

```
'EX01-01-c
```

```
'Coin.vb
Imports VendingService
Public Class one_dollar
    Implements CoinService

    Public Function value() As Double Implements CoinService.value
        Return 1
    End Function
End Class
Public Class five_dollar
    Implements CoinService
    Public Function value() As Double Implements CoinService.value
        Return 5
    End Function
End Class
Public Class ten_dollar
    Implements CoinService
    Public Function value() As Double Implements CoinService.value
        Return 10
    End Function
End Class
```

使用 VS.Net 的 Rebuild 功能，從這項目產生一個 CoinFamily.dll 程式庫文檔。

Step-4: 設計 Windows 應用程式，以整合販賣機和錢幣物件

於是，建立一個 Windows Application: EX01-01-d 項目，並使用 Add Reference 來參考 VendingService.dll、VendingMachine.dll、CoinFamily 三個類別庫。VB.Net 程式碼如下：

```
'EX01-01-d
'Form1.vb
Imports System.Xml
Imports VendingService
Imports VendingMachine
Imports CoinFamily
```

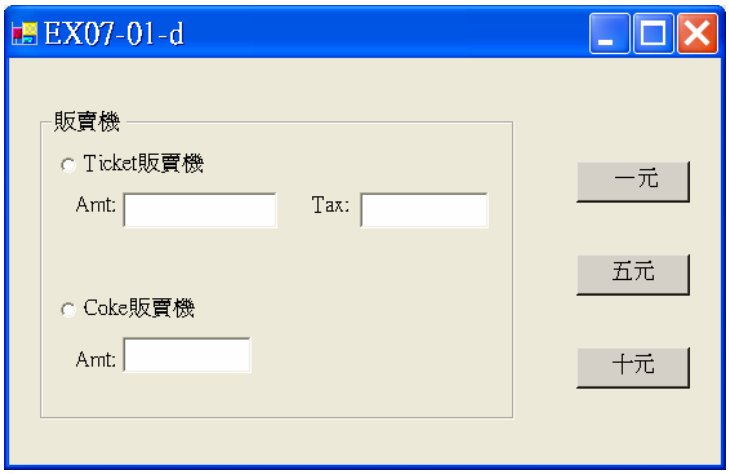
```
Public Class Form1
    Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "
  ……
#End Region
    Private ticket, coke, current As MachineService
    Private coin As CoinService

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e
                           As System.EventArgs) Handles MyBase.Load
        ticket = New TicketMachine
        coke = New CokeMachine
        current = Nothing
    End Sub

    Private Sub TicketRB_CheckedChanged(ByVal sender As System.Object,
          ByVal e As System.EventArgs) Handles TicketRB.CheckedChanged
        current = ticket
    End Sub

    Private Sub CokeRB_CheckedChanged(ByVal sender As System.Object,
           ByVal e As System.EventArgs) Handles CokeRB.CheckedChanged
        current = coke
    End Sub

    Private Sub OneBtn_Click(ByVal sender As System.Object, ByVal e
                    As System.EventArgs) Handles OneBtn.Click
        If current Is Nothing Then
            MessageBox.Show("No Machine is selected!")
            Return
        End If
        coin = New one_dollar
        current.feedCoin(coin)

        Dim strXml As String
        strXml = current.getTotal
        Me.parseXml(strXml)
    End Sub
```

```
Private Sub FiveBtn_Click(ByVal sender As System.Object, ByVal e
                          As System.EventArgs) Handles FiveBtn.Click
    If current Is Nothing Then
        MessageBox.Show("No Machine is selected!")
        Return
    End If
    coin = New five_dollar
    current.feedCoin(coin)

    Dim strXml As String
    strXml = current.getTotal
    Me.parseXml(strXml)
End Sub

Private Sub TenBtn_Click(ByVal sender As System.Object, ByVal e
                         As System.EventArgs) Handles TenBtn.Click
    If current Is Nothing Then
        MessageBox.Show("No Machine is selected!")
        Return
    End If
    coin = New ten_dollar
    current.feedCoin(coin)
    Dim strXml As String
    strXml = current.getTotal
    Me.parseXml(strXml)
End Sub

Private Sub parseXml(ByVal xx As String)
    Dim xml As New XmlDocument
    xml.LoadXml(xx)
    Dim nlist As XmlNodeList
    Dim m_node, mChildNode As XmlNode

    nlist = xml.SelectNodes("/Total")
    'Loop through the nodes

    Dim amt, tax As String
    tax = Nothing
    amt = Nothing
    For Each m_node In nlist
```

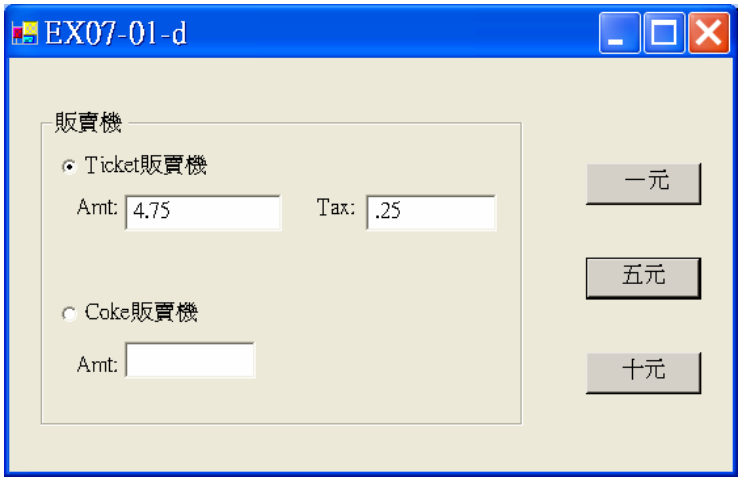
```
                For Each mChildNode In m_node.ChildNodes
                    Select Case mChildNode.Name
                        Case "Amt"
                            amt = mChildNode.InnerText
                        Case "Tax"
                            tax = mChildNode.InnerText
                    End Select
                Next
            Next
            '-------------------------------------
            If tax Is Nothing Then
                CokeTx.Text = amt
            Else
                TicketTx.Text = amt
                TaxTx.Text = tax
            End If
        End Sub

        Private Sub TicketTx_TextChanged(ByVal sender As System.Object, ByVal
                e As System.EventArgs) Handles TicketTx.TextChanged
        End Sub
End Class
```

Step-5: 執行上述程式

此程式執行時，出現起始畫面：

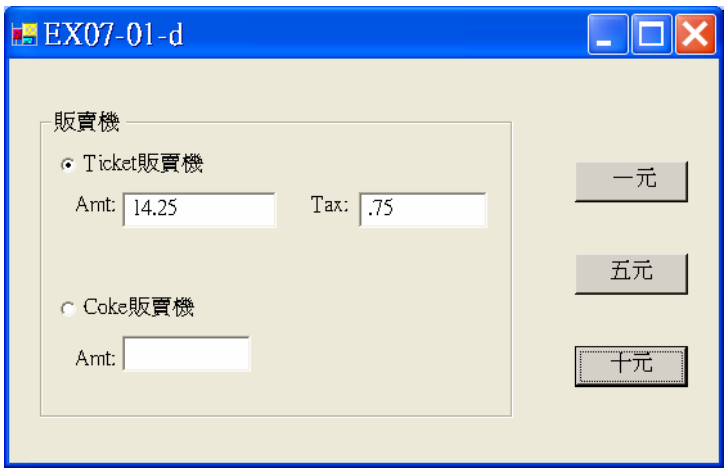


左邊有兩個販賣機，從 Radio Button 你可以選擇欲使用之販賣機，然後按下右邊的 Command Button，代表投錢之動作。例如，按下「Ticket 販賣機」Radio Button，然後按下「五元」Command Button，就將五元餵入 Ticket 販賣機了，販賣機計算出 Tax 金額，並顯示出來：

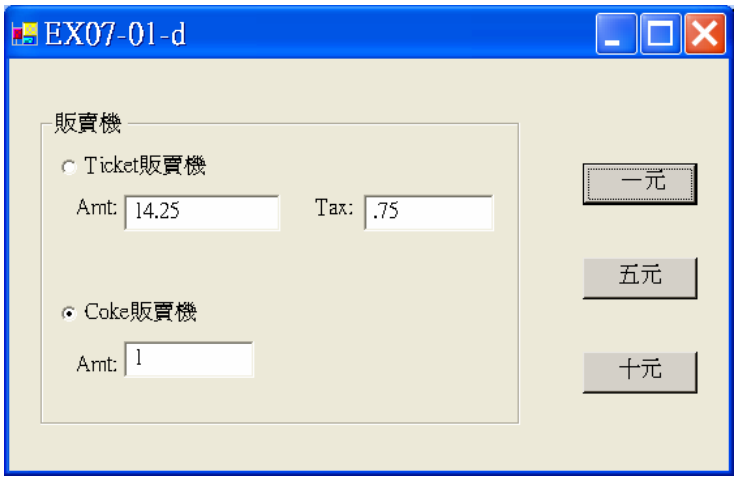


當您繼續投入錢幣時，販賣機會累積而顯示出總金額，和總 Tax 金

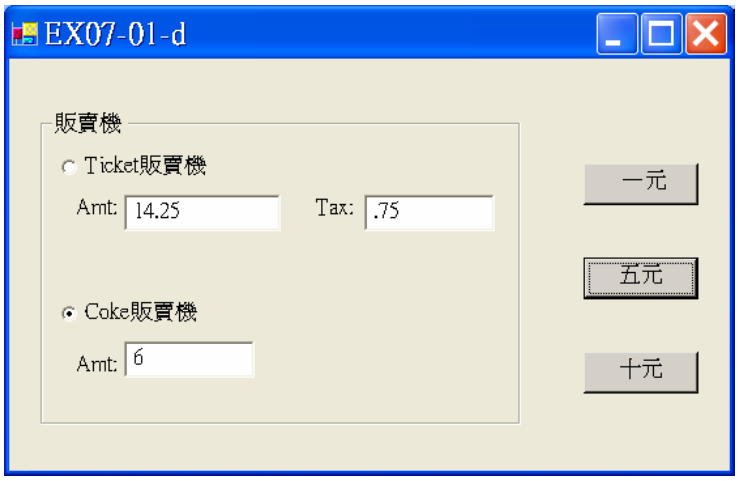
額。例如，按下「十元」Command Button，就將十元餵入 Ticket 販賣機了，販賣機計算出 Tax 金額，並顯示出來如下：



您也可以按下「Coke 販賣機」Radio Button，然後按下「一元」Command Button，就將一元餵入 Coke 販賣機了，販賣機顯示出來如下：



當您繼續投入錢幣時，販賣機會累積而顯示出總金額。例如，按下「五元」Command Button，就將五元餵入 Coke 販賣機了，販賣機顯示出總金額如下：



這項工法是來自一般的 OOAD(Object-Oriented Analysis)，以繼承關係建構類別體系，再以界面將之連結起來，這是系統整合的初步工法。

4.2 工法 2：以 Façade 樣式(Pattern)實現接口

在軟體業裡，這算是一項先進的工法，但是在其他成熟產業已經是典型的整合工法了。最膾炙人口的是貨櫃(Container)，從世界航運鉅子---- 長榮海運的迅速茁壯，人人皆能領會到這項新整合工法的威力，能帶來的經濟效益和大商機。

於此，請您先分享長榮因貨櫃而繁榮的歷史經驗，體會出貨櫃(是一種接口物件)如何帶來巨大經濟價值。貨櫃的外表簡單有序，能疊得很高，而且井然有序；貨櫃的內部是空的，用來容納多樣化而繁雜的物

品；這種情形稱爲序中有亂(變化)。序(即接口)中有亂的巨大威力，改變了整個運輸產業，也改變了人們的生活。就如 Intel 公司總裁 Andrew Grove 曾舉貨櫃爲例說道[San98]：

> 「短短的十年，實際上只是航運史上一個極短促的時間，造船設計即走向標準化，冷凍運輸船也誕生了，最重要的是貨櫃化作業的興起，這種容許船貨更快速裝卸的技術在航運的生產力上引進了『十倍速』的改變，逆轉了節節升高的成本趨勢，..... 有些港口做了改變，有些雖盡力改革，卻無法做到，許多則堅持抗拒這種潮流，結果，這些新技術引發了全世界貨運港口的重新洗牌，.... 沒有採行新技術的港口（如新加坡及西雅圖）可能被重新規畫，成爲購物中心、休閒場所或河岸公寓大廈。」

長榮海運公司總裁 張榮發 先生在其回憶錄上寫道[張 97]：

> 「爲了配合貨櫃化運輸時代的來臨，海運事業的整體運作型態也產生了重大的轉變，無論在海上運送、碼頭作業以及陸運轉接上，都有革命性的改變。陸上拖車運輸業應時而興，扮演極爲重要的角色，以貨櫃拖車配合貨櫃船運輸，具有簡化包裝、防止竊盜、加速貨物搬運及便利關務檢驗等優點，使貨櫃運輸作業更加靈活。爲了能確實掌握海上及陸上的運送服務品質和效率，就率先於 1973 年 9 月成立了長榮運輸公司。」

貨櫃的威力是來自於它提供了一致的『接口』，讓物品與輪船能一拍即合，而且分合自如。異質性物品可以整合於貨櫃中，貨櫃與貨櫃之間能輕易整合(疊得很高)，貨櫃與輪船、拖車、港口碼頭的卸貨吊車、倉庫等皆能一拍即「整合」。因而這是一項威力強大的整合工法，對內整合

複雜的物品，對外提供標準接口，易於自我重複組合，也易於跟輪船等整合。透過貨櫃提供之標準接口，讓物品與輪船等能一拍即合、分合自如，這種整合創造了輪船的巨大「重用」(Reuse)機會，物品也因而暢其流，處處整合處處商機。

貨櫃不一定給物品(如汽車、木材、玩具等)產業帶來方便(如大汽車必須拆解開來才能裝進貨櫃裡)。但是運輸業等獲得好處，其樂意把貨櫃標準接口拱起來，促成貨櫃的革命性風潮。

那麼，Façade 樣式就是一種軟體貨櫃，它讓軟體之間一拍即合，與硬體之間也能分合自如。

4.2.1 認識 Façade 樣式

Façade 是源自法國語，表示一個建築物的門面。就像我國傳統的四合院，就是一個「門面樣式」(Façade Pattern)的呈現，在一個方形的圍牆裡，能住進許多人，也能裝進百花齊放的東西，例如床、書桌、餐具等等，包羅萬象。Façade 樣式源自 Gamma 等人所著的 *Design Patterns* 書。Facade 樣式的主要用途是：將一個子系統內眾多物件的接口統一起來，讓子系統能不斷地成長。Gamma 在其書中提到[Gam95]：

> "The common design goal is to minimize the communication and dependencies between subsystems. .... This can be an important consequence when the client and the subsystem are implemented independently."
>
> (軟體設計的共通目標之一就是：確保子系統之間相依性減至最低，子系統之間的溝通最少。 ..... 如果您想要讓子系統與其 Client 之間能獨立發展的話，此種設計能帶給您好效果。)

如果沒有採用 Facade 樣式，則其 Client 端物件直接使用 Server 端子系

統的物件接口；

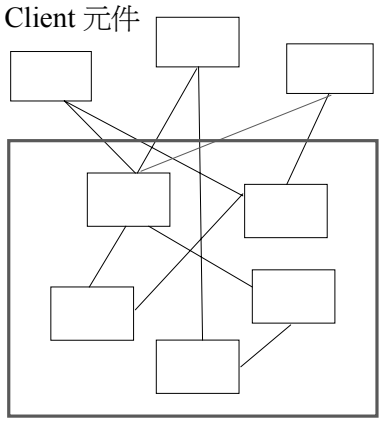


圖 13　複雜的溝通與高相依性[Gam95]

因爲 Server 端物件是共用的，一旦子系統的物件接口改變了，就會影響到許多 Client 物件。這意味著，Server 端子系統的變化成本增高，人們就不願意去更改它，則子系統就無法自由獨立地成長了。Gamma 在其書中建議[Gam95]：

> "One way to achieve the goal is to introduce a facade object that provides a single, simplified interface to the more general facilities of a subsystem."
>
> (達成這個目標的方法，就是：引進一個 Facade 物件以提供一個單純化的接口，透過它而取得子系統的服務。)

採用此項作法，就得到下圖的美好結構：

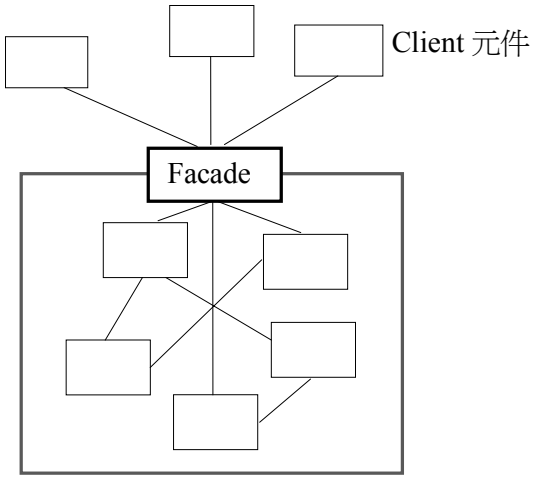


圖 14　單純的溝通與低相依性[Gam95]

當子系統內物件的接口有所調整時，常只需要調整 Facade 物件內的實作(Implementation)部份而已，不會影響到 Facade 物件的接口，也就不影響 Client 物件。於是，子系統的改變成本變低了，因而能彈性調整及獨立成長了。Gamma 也在其書中仔細說明 Facade 物件的效益：

> "It shields clients from subsystem components, thereby reducing the number of objects that clients deal with and making the subsystem easier to use."
>
> (它封藏了子系統內的物件，讓 client 不必跟許多物件打交道，而能夠容易使用該子系統。)

> "It promotes weak coupling between the subsystem and its clients. Often the components in a subsystem are strongly coupled. Weak coupling lets you vary the components of the subsystem without affecting its clients. ..."

(它降低了子系統與 client 之間的聯偶性。通常子系統內的物件之間會有很強的聯偶性。弱聯偶性意謂著您能彈性調整子系統內的物件，但不會影響到 clients。)

我們可以把 Façade 物件視爲虛的，就像四合院的大門，它也是虛的；而門內的房子才是實的，兩者虛實相依。Façade 物件塑造出一個虛的空間，能容納一群實的物件，使得實物件擁有極高的變化自由度，甚至隨時能輕易抽換掉實物件。這就是 Gamma 所說的「弱聯偶性」(Weak Coupling)。

就像四合院，其所建構出來的空間裡(包括戶外庭院及室內空間)能裝進許多多東西，而且能彈性地移進移出，因此能包容複雜的變化，蘊藏著無限的潛能，例如人們能住在四合院裡自由快樂地成長，我國數千年來就是如此，不是嗎？

1.4.2.2 Façade 提供 Web 標準接口

在 Web 時代裡，Web Service 與 Façade 樣式是「最佳拍檔」。Façade 樣式追求設計層級的「弱聯偶性」(Weak Coupling)；而 Web Service 則追求實作層級的「疏結合性」(Loosely Coupling)。兩者的目標是一致的，而且互補，堪稱爲最佳拍檔。

Web Service 技術讓我們能降低與平台的相依性，而 Façade Pattern 技術則能降低 Client 與後端 Subsystem 之間的相依性。兩者的融合將創造真正的低相依性。如下圖所示：

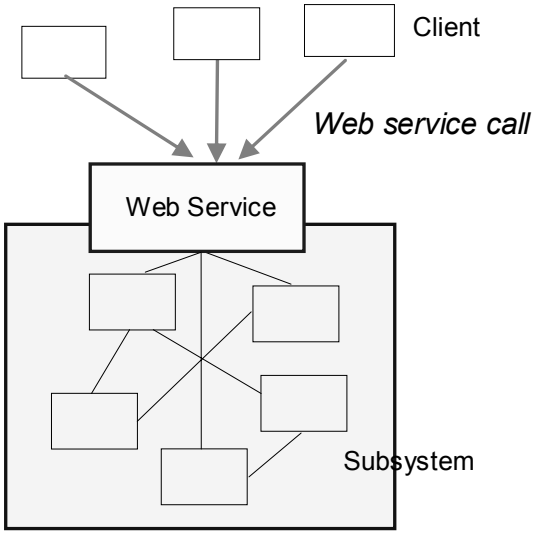


圖 15　Loosely Coupling 關係

這種 Web Service Façade 扮演兩項角色：

1. Façade 物件提供單一整合接口

---- 封裝異質性的物件或子系統，提供整體服務接口。就如 Gamma 所提，Façade 物件之目的是：

"….a facade object that provides a single, simplified interface to …"
(….façade 物件提供單一的接口.)

2. Web Service 提供標準接口協定(Protocol)

---- 提供標準的 Web Service 協定，降低 Client 與 Subsystem 所在平台之相依性。

兩者的融合成爲：

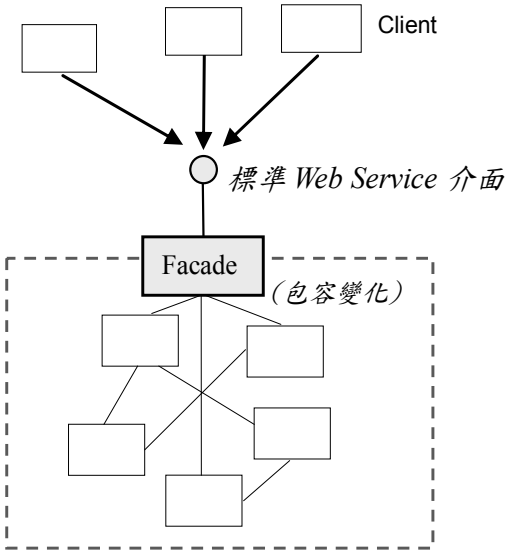


圖 16　Facade 接口物件

這 Façade 樣式，跟長榮的貨櫃一樣，對內整合複雜的軟體系統，對外提供標準接口，易於自我重複組合，也易於跟硬體組件等整合。透過 Facade 提供之 Web 標準接口，讓系統與系統等能一拍即合、分合自如，這種整合創造了巨大的軟體「重用」(Reuse)機會，處處整合處處商機。

4.2.3　DIY：動手落實 Façade 樣式

茲舉下圖爲例，我們將親手以 VB.Net 落實 Web Service-based 的 Façade 樣式，體會看看是否如貨櫃一般，能發揮系統整合的巨大功效，如下圖 1-17 所示。此 Façade 樣式含有一個 SalesFacade 物件，它擔任兩項任務：

1. 對內，它容納(整合)各種物件、模組或系統。
2. 對外，它提供 Web 標準接口，與外界系統能分合自如。

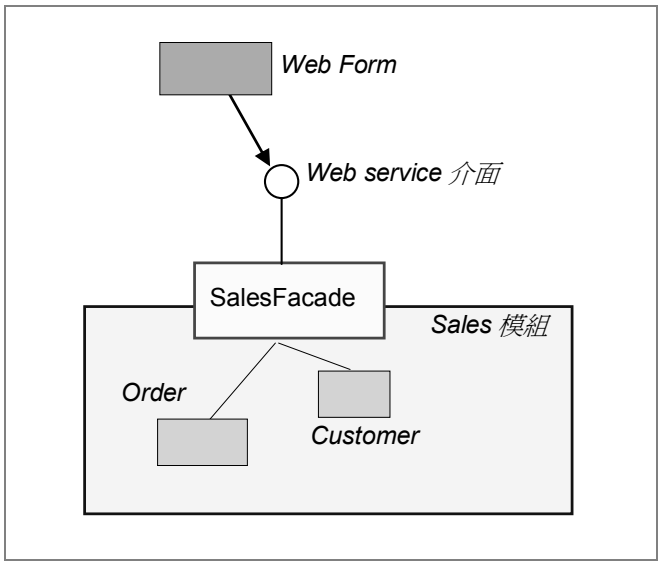
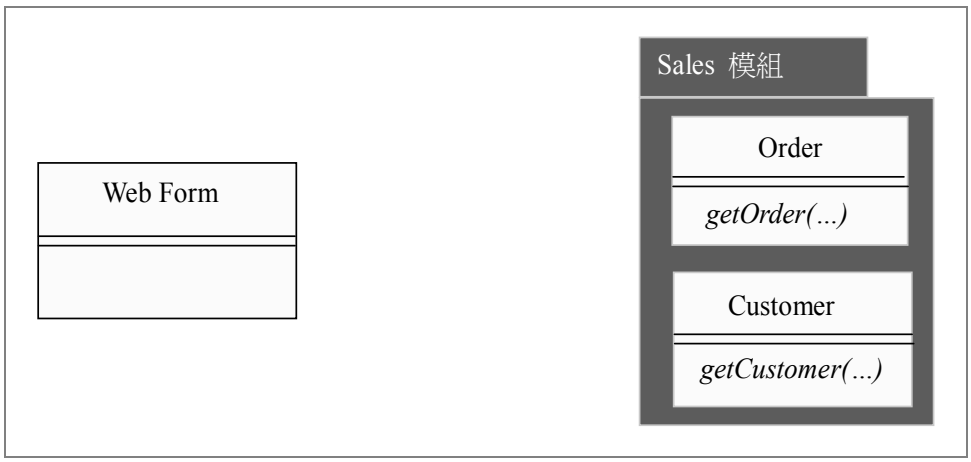


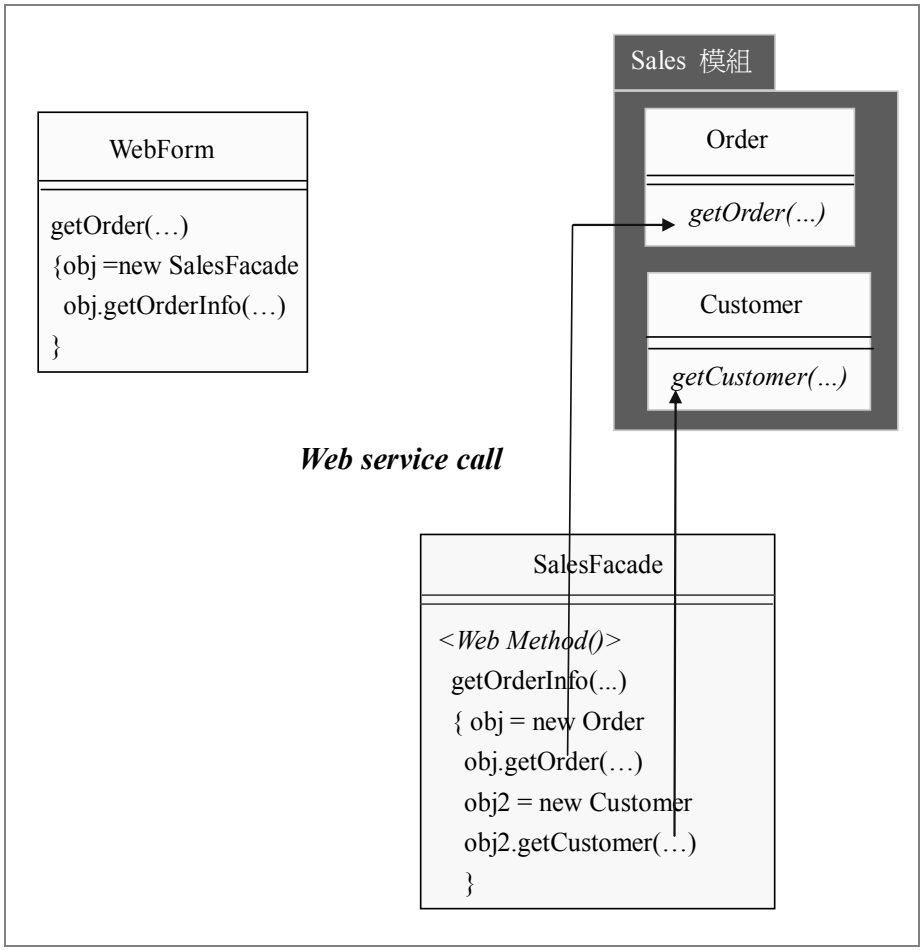
圖 17　Façade 樣式之實作範例

現在看看如何使用 VB.Net 實現之?

Step-1 首先規劃 Order 及 Customer 兩個類別，還有一個 Windows Form 或 Web Form。如下圖：

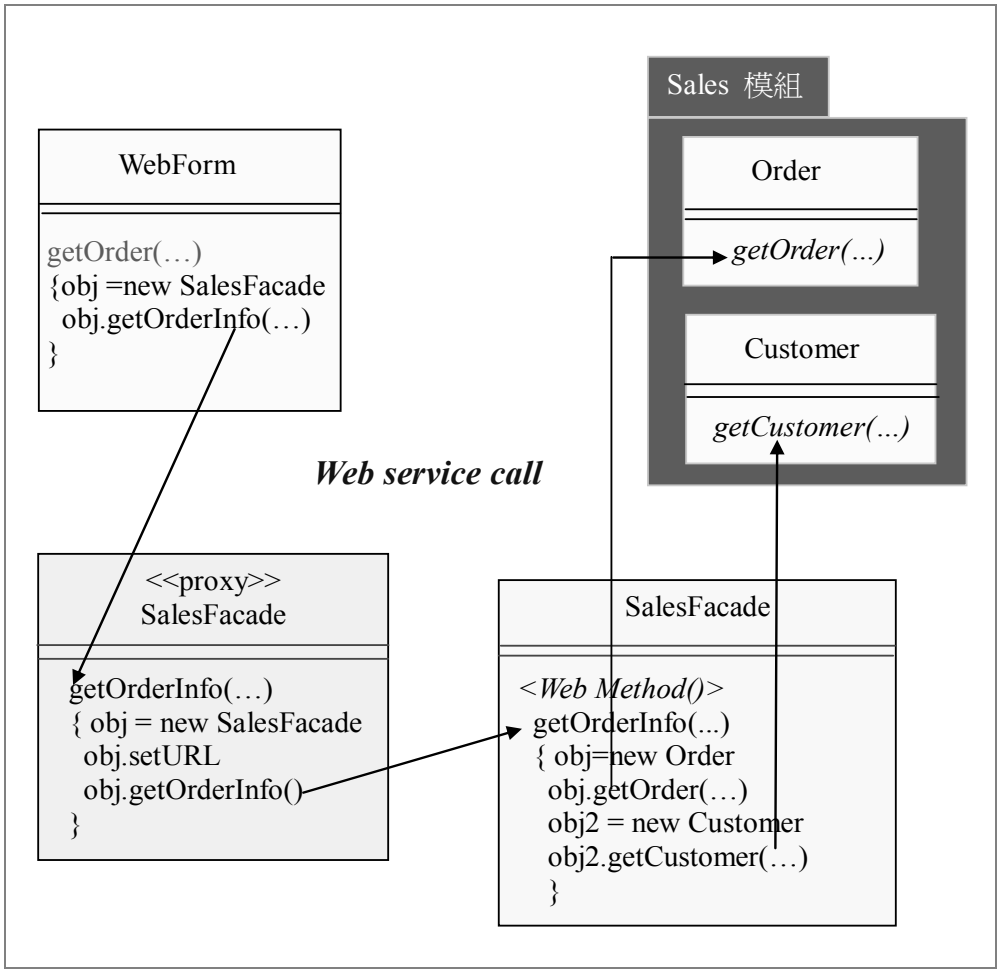


Step-2 接著建立 SalesFacade 類別，讓 Client 透過 SalesFacade 的 Web Service，間接地使用 Order 和 Customer 的服務，下圖進一步實現 SalesFacade 架構：



設計了 SalesFacade 之後，VB.Net 開發環境會產生所對應的 WSDL 文件，然後提交給 Client 的開發者。Client 的 VB.Net 開發環境讀取 WSDL 文件之後，自動在 Client 端產生一個 SalesFacade 的 proxy 類別(包

含在 Reference.vb 裡面，也稱爲 SalesFacade 類別)。在實際執行程式時，Client 端會誕生 proxy 類別之物件，例如下圖所示：



proxy 和 SalesFacade 構成 Client 與 Server 之間的溝通橋樑。如此，在世界各地的 Client 模組皆能遠距地呼叫我們開發的 Sales 模組了。請看 EX01-02 範例之 VB.Net 程式碼如下：

Step-1 設計 Order 和 Customer 兩個類別。

於是，建立 EX01_02_a 程式庫(Class Library)項目，其內容爲：

```
'EX01_02_a
'Sales.vb
Public Class Order
    Public Function getOrder() As String
        Return "<OrderDetail><OrderID>A123</OrderID>" & _
               "<OrderDate>2006/5/21</OrderDate></OrderDetail>"
    End Function
End Class

Public Class Customer
    Public Function getCustomer() As String
        Return "<CustomerName>Mike Bush</CustomerName>"
    End Function
End Class
```

Step-2 設計一個 SalesFacade 類別，將 Order 和 Customer 的服務組裝出 getOrderInfo 整合服務。建立 EX01_02_b 網路服務(Web Service)項目，其內容爲：

```
'EX01_02_b
'Service1.asmx.vb
Imports System.Web.Services
Imports MailService

Namespace EX01_02_b_ws
<System.Web.Services.WebService(Namespace:=
                    "http://LocalHost/EX01_02_b_ws/Service1")> _

Public Class SalesFacade
    Inherits System.Web.Services.WebService

#Region " Web Services Designer Generated Code "
  ……..
#End Region
```

```
    <WebMethod()> _
        Public Function getOrderInfo() As String
            Dim orderObj As New Order
            Dim custObj As New Customer
            Dim custNaXml As String
            Dim orXml As String
            orXml = orderObj.getOrder()
            custNaXml = custObj.getCustomer()
            Return "<Order>" & custNaXml & orXml & "</Order>"
        End Function
End Class
End Namespace
```

設計好了，可使用 Browser 先試試看這個 Web Service，把 http://LocalHost/EX01_02_b_ws/Service1.asmx 填入網址，執行如下：

按下 getOrderInfo 連結，出現：

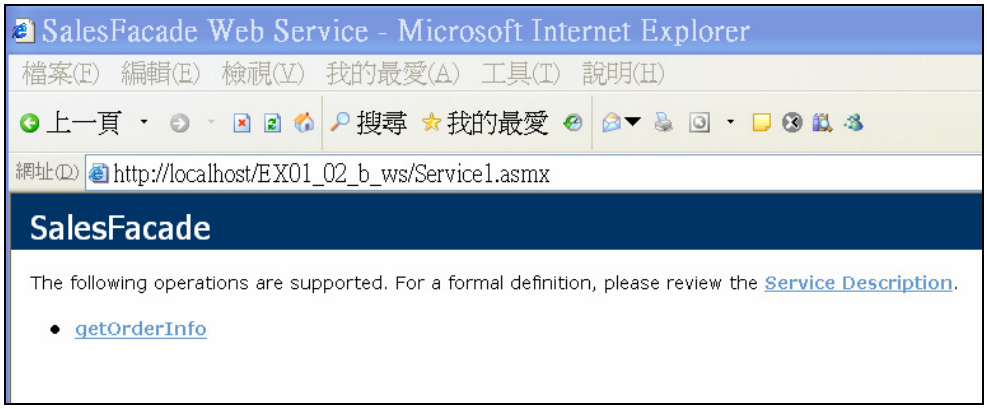
按下 Invoke 按鈕，就呼叫 getOrderInfo 服務了，這個 Web Service 執行之後傳回結果如下：

從結果看出這個 Web Service 是正確的。

Step-3 查看 WSDL 接口敘述檔。在建立 Web Service 時，可產出一 WSDL

文檔，其敘述了此 Web Service 的內涵。

您可在 Browser 上按下 Service Description 連結，如下圖：



或是在網址填入 http://localhost/EX01_02_b_ws/Service1.asmx?WSDL，就可看到 WSDL 文檔的內容如下：

WSDL 文檔：

```
<?xml version="1.0" encoding="utf-8" ?>
-<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tns="http://LocalHost/EX01_02_b_ws/Service1"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
targetNamespace="http://LocalHost/EX01_02_b_ws/Service1"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
- <wsdl:types>
- <s:schema elementFormDefault="qualified"
targetNamespace="http://LocalHost/EX01_02_b_ws/Service1">
- <s:element name="getOrderInfo">
  <s:complexType />
```

```
  </s:element>
- <s:element name="getOrderInfoResponse">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="0" maxOccurs="1" name="getOrderInfoResult"
type="s:string" />
  </s:sequence>
  </s:complexType>
  </s:element>
  </s:schema>
  </wsdl:types>
- <wsdl:message name="getOrderInfoSoapIn">
  <wsdl:part name="parameters" element="tns:getOrderInfo" />
  </wsdl:message>
- <wsdl:message name="getOrderInfoSoapOut">
  <wsdl:part name="parameters" element="tns:getOrderInfoResponse" />
  </wsdl:message>
- <wsdl:portType name="SalesFacadeSoap">
- <wsdl:operation name="getOrderInfo">
  <wsdl:input message="tns:getOrderInfoSoapIn" />
  <wsdl:output message="tns:getOrderInfoSoapOut" />
  </wsdl:operation>
  </wsdl:portType>
- <wsdl:binding name="SalesFacadeSoap" type="tns:SalesFacadeSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
- <wsdl:operation name="getOrderInfo">
  <soap:operation
soapAction="http://LocalHost/EX01_02_b_ws/Service1/getOrderInfo"
style="document" />
- <wsdl:input>
  <soap:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
- <wsdl:binding name="SalesFacadeSoap12" type="tns:SalesFacadeSoap">
  <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" />
- <wsdl:operation name="getOrderInfo">
```

```
  <soap12:operation
soapAction="http://LocalHost/EX01_02_b_ws/Service1/getOrderInfo"
style="document" />
- <wsdl:input>
  <soap12:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap12:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
- <wsdl:service name="SalesFacade">
- <wsdl:port name="SalesFacadeSoap" binding="tns:SalesFacadeSoap">
  <soap:address location="http://localhost/EX01_02_b_ws/Service1.asmx" />
  </wsdl:port>
- <wsdl:port name="SalesFacadeSoap12" binding="tns:SalesFacadeSoap12">
  <soap12:address location="http://localhost/EX01_02_b_ws/Service1.asmx" />
  </wsdl:port>
  </wsdl:service>
  </wsdl:definitions>
```

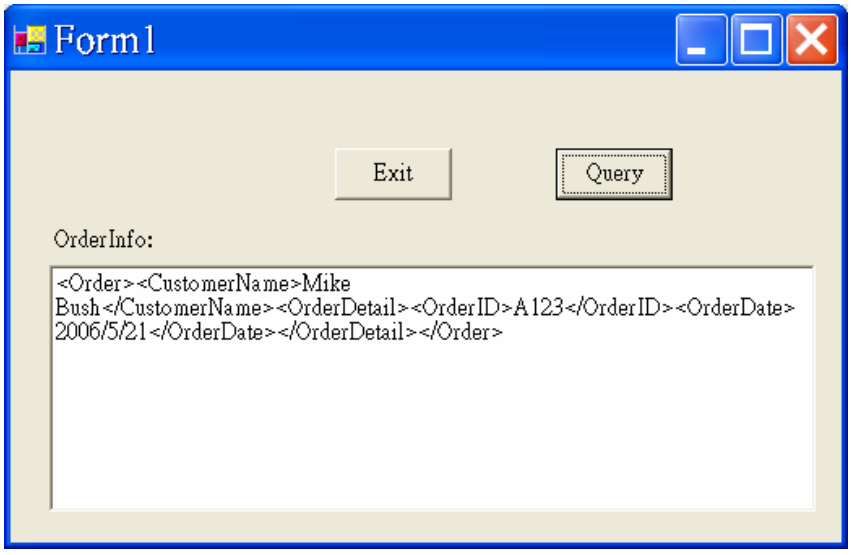
這個 WSDL 是接口敘述，是給 Client 程式使用的，Client 程式藉之了解該 Web Service 接口之內容。

Step-4 設計 Windows Form 或 Web Form 類別。也就是開發 Client 程式來使用這個 Web Service 接口。例如下述的 Windows Form 程式：

```
'EX01_02_c
'Form1.vb
Public Class Form1
      Inherits System.Windows.Forms.Form
#Region " Windows Form Designer generated code "
        ……
#End Region
      Private Sub ExitBtn_Click(ByVal sender As System.Object, ByVal e
            As System.EventArgs) Handles ExitBtn.Click
```

```
        Me.Close()
    End Sub

    Private Sub QueryBtn_Click(ByVal sender As System.Object, ByVal e
                               As System.EventArgs) Handles QueryBtn.Click
        Dim wsfa As localhost.SalesFacade
        wsfa = New localhost.SalesFacade
        Dim strXml As String
        strXml = wsfa.getOrderInfo()
        Me.InfoTx.Text = strXml
    End Sub
End Class
```

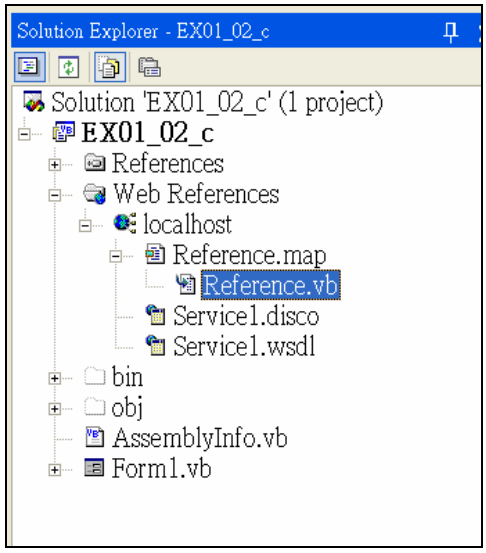
指令 wsfa = New localhost.SalesFacade 是誕生 proxy 類別之物件，它再呼叫真正的 SalesFacade 物件的 getOrderInfo()。此程式執行結果是：



Step-5 查看 SalesFacade 的 proxy 類別。

Sales 模組已經開發完畢了，能接受 Client 的遠距呼叫了。剛才提過

了，Client 開發人員在 VB.Net 開發環境上必須參考上述 WSDL 文檔內容，並自動產生一個 Reference.vb 程式檔，該程式檔含有 SalesFacade 類別(proxy)來使用。您可以從 Solution Explorer 視窗裡看到 Reference.vb 的內容。



其內容爲：

```
'Reference.vb
Opt_u111 ?n Str_u99 ?t Off
Opt_u111 ?n Explicit On

Imports System
Imports System.ComponentModel
Imports System.D_u97 ?gnostics
Imports System.Web.Services
Imports System.Web.Services.Protocols
Imports System.Xml.Serialization
'
'Th_u115 ? source code was auto-generated by Microsoft.VSDesigner, Version
1.1.4322.2032.
```

```
'
Namespace localhost
    '<remarks/>
    <System.Diagnostics.DebuggerStepThroughAttribute(),  _
     System.ComponentModel.DesignerCategoryAttribute("code"),  _

System.Web.Services.WebServiceBindingAttribute(Name:="SalesFacadeSoap",
            [Namespace]:="http://LocalHost/EX01_02_b_ws/Service1")>  _

    Public Class SalesFacade
        Inherits System.Web.Services.Protocols.SoapHttpClientProtocol

        '<remarks/>
        Public Sub New()
            MyBase.New
            Me.Url = "http://localhost/EX01_02_b_ws/Service1.asmx"
        End Sub

        '<remarks/>
        <System.Web.Services.Protocols.SoapDocumentMethodAttribute(
            "http://LocalHost/EX01_02_b_ws/Service1/getOrderInfo",
            RequestNamespace:="http://
            LocalHost/EX01_02_b_ws/Service1",
            ResponseNamespace:="http://
            LocalHost/EX01_02_b_ws/Service1",
            Use:=System.Web.Services.Description.SoapBindingUse.Literal,
            ParameterStyle:=System.Web.Services.Protocols.
            SoapParameterStyle.Wrapped)>  _

        Public Function getOrderInfo() As Str_u110 ?g
            Dim results() As Object = Me.Invoke("getOrderInfo", New Object(-1) {})
            Return CType(results(0),Str_u110 ?g)
        End Function

        '<remarks/>
        Public Function BegingetOrderInfo(ByVal callback As
            System.AsyncCallback,
            ByVal asyncState As Object) As System.IAsyncResult
            Return Me.BeginInvoke("getOrderInfo", New Object(-1) {},
                callback, asyncState)
```

```
            End Function

            '<remarks/>
            Public Function EndgetOrderInfo(ByVal asyncResult As
                        System.IAsyncResult) As Str_u110 ?g
                Dim results() As Object = Me.EndInvoke(asyncResult)
                Return CType(results(0),Str_u110 ?g)
            End Function
        End Class
End Namespace
```

Reference.vb 的 SalesFacade 類別如同您派駐至 Client 端的外派代表(通稱爲 Proxy)，就像派駐國外的大使館一般，提供 Client 模組一個溝通橋樑。Web Service 標準接口是系統整合的重要技術，若能親自動手，多加應用，就能輕鬆愉快地設計出新時代的軟體貨櫃(即 Façade)，將讓您在系統整合上無往不利。

(請繼續看下一節)

本文摘自　高煥堂 所著的
“VB.Net: 大型系統整合 DIY” 一書

高煥堂 著

高煥堂 著

高煥堂 著

Part 2 of 3

4.3 工法 3：以 Façade 樣式(Pattern)實現接口

在前面，已經介紹了兩項工法，您也親自動手設計軟體貨櫃(即 Façade)了。根據一般常識，人們能體會到貨櫃具有如下之特色：

貨櫃：

包容複雜，一致接口，
簡單組合，無限複製，
裝進天下所能裝之物。

同時，也看出 Façade 擁有極相似之特色：

Facade：

包容複雜，一致接口，
簡單組合，無限複製，
裝進天下所有的軟體。

以上，我們已經親手(DIY)設計出 Façade 軟體貨櫃，實現了兩項特色：「包容複雜，一致接口」。接下來，將介紹另兩項特色：「簡單組合，無限複製」，讓前兩項工法之效益無限擴大，引發爆炸性成長。

4.3.1 從「四合院」體會 Façade 之簡單組合

Façade pattern 如何組合出巨大系統的架構呢? 就跟四合院一樣，是樹狀(Hierarchical)組合，四合院的建築師也不斷運用樹狀組合，來規劃巨大

的四合院豪宅，如下圖：

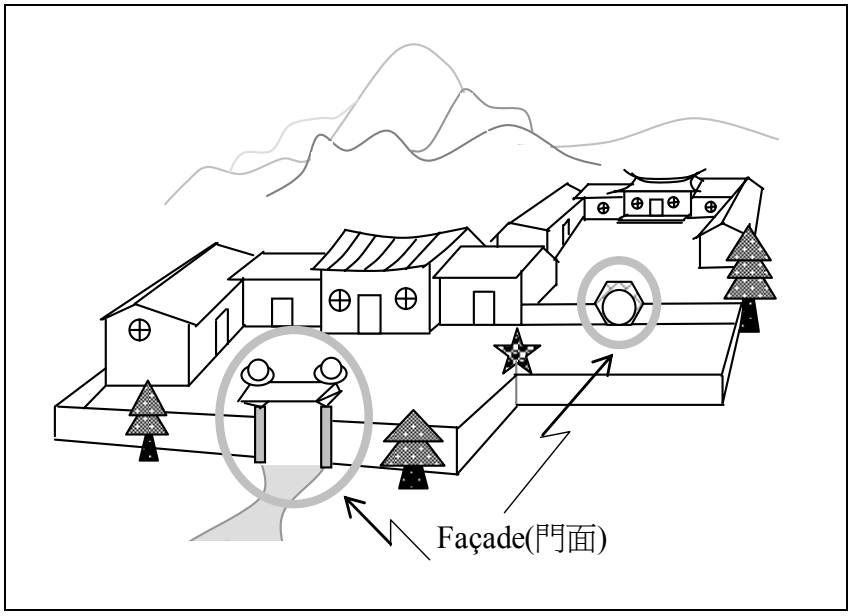


圖 18　四合院豪宅---- 單純的樹狀組合

在軟體裡，跟四合院一樣，大架構極爲相似，都由 Façade 樣式所組成。

4.3.2　從「樹木」體會 Façade 之無限複製

自然界事物之設計，其限制之一就是：「信息的有限性」。一個生物形體的造成，是出自一個概觀的計劃---- 簡單的序，隨著生物的成長，與環境互動的信息越多，逐漸在細節上修修補補，就發展出多樣化的細節。所以，我們長得一模一樣的手掌，其細節紋路並非由同樣規格所造成的。

人造系統(如房屋或軟體系統)之設計，其重要的限制也是：「信息的

有限性」。所以在系統的生命週期中，必須不斷地配合新需求而修修補補，也就逐漸發展出多樣的變化。爲了方便修修補補但又不至於失去整體性，可效法生物之設計，像海洋的波浪或是樹葉的葉緣，**來自簡單的序而無盡的重複**，在其成長過程中，隨著新信息的產生，逐漸在細節上修修補補，產生多樣化的細節。例如下圖：

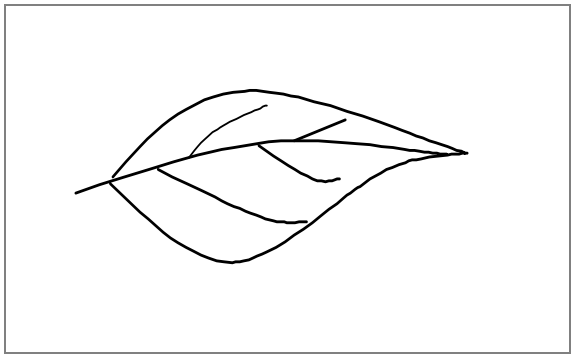


圖 19　單一元素、簡單造形、複雜內涵

葉子是一個元素或單位，其外形是簡單的，而內涵卻是複雜的。簡而言之，序的要素有三：

要素 1：　有簡單外形(接口)、複雜內涵的小單位。
要素 2：　有簡單的組合規律。
要素 3：　無盡的自我重複。

例如，從下圖 20 ~ 圖 21，很容易看出其組合的規律性，及其無限的自我類似與重複。由於每片葉子的內涵是複雜的，下圖 20 的簡單組合---- 兩片葉子爲一組，此組合結構又是簡單外形、複雜內涵的單位。

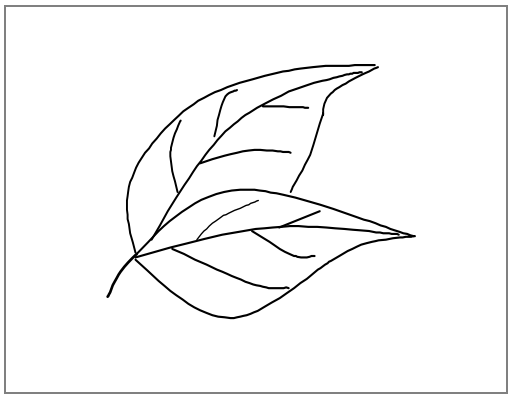


圖 20　簡單組合、自我重複

此結構單位會再無限自我類似及重複，如下圖：

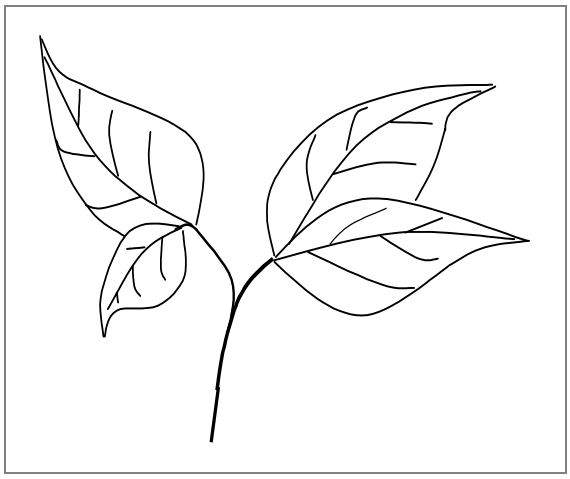


圖 21　再簡單組合、自我重複

持續自我重複，就組成較大的樹枝，如下圖所示：

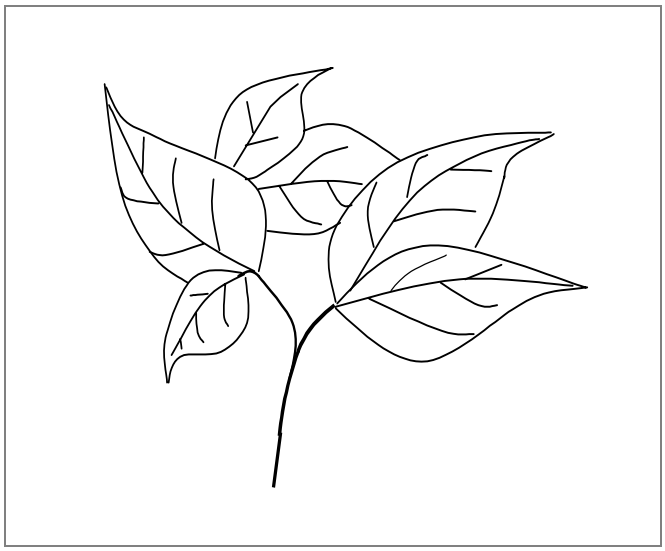


圖 22　美好的「序」來自無限重複

樹枝再自我重複，就成爲一棵樹。樹再自我重複，就成爲一座森林了。由於無盡的自我重複，同一棵樹上的眾多葉子皆有共同的結構、一致的型態。

信息系統就如同一顆楓樹一般，具有整體的和諧感覺，而且擁有該樹獨特的風味。一致的序就如同基因(Gene)，決定葉子的巨觀結構，卻也支撐並創造出每片葉子細膩的特殊內涵，此外也讓眾多樹葉能和諧地創造出無限的特殊組合體：一顆樹。

4.3.3　DIY：以 VB.Net 落實 Façade 之簡單組合

一個 Facade 樣式猶如一隻軟體貨櫃，如下圖：

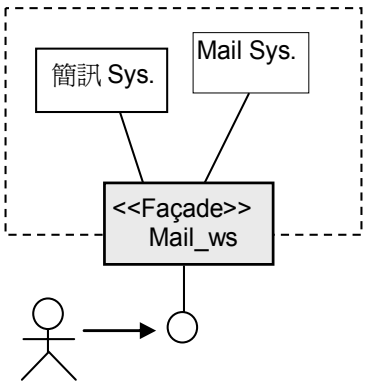


圖 23 Façade 樣式的簡單樹狀組合

茲以 VB.Net 程式碼來示範之。首先建立 EX01_03_a 程式庫(Class Library)項目，其內容爲：

```
'EX01_03_a
'MailSys.vb
Imports System.Web.Mail
Imports System.Xml

Public Class MailSys
    Private mMsg As New MailMessage

    Function sendMail(ByVal mailXml As String) As String
        Dim xml As New XmlDocument
        xml.LoadXml(mailXml)
        Dim nlist As XmlNodeList
        Dim m_node, mChildNode As XmlNode

        nlist = xml.SelectNodes("/Record")
        Dim st As String
        For Each m_node In nlist
            For Each mChildNode In m_node.ChildNodes
                Select Case mChildNode.Name
```

```
                        Case "Subject"
                            mMsg.Subject = mChildNode.InnerText
                        Case "Body"
                            mMsg.Body = mChildNode.InnerText
                    End Select
                Next
            Next
            '--------------------------------------
            mMsg.From = "misoo.tw@msa.hinet.net"
            mMsg.To = "yoo161@yahoo.com.tw"
            Try
                SmtpMail.Send(mMsg)
            Catch ex As Exception
                Return "<MailStatus>" + ex.Message.ToString +
"</MailStatus>"
            End Try
            Return "<MailStatus>ok</MailStatus>"
        End Function
End Class
```

設計 MailSys 類別之後，再添加一個 Web-based 的 Mail_ws 類別。也就是建立 EX01_03_b_ws 網路服務(Web Service)項目，其內容爲：

```
'EX01_03_b_ws
'Mail_ws.asmx.vb
Imports System.Web.Services
Imports MailService

<System.Web.Services.WebService(Namespace
                :="http://localhost/EX01_03_b_ws/Mail_ws")> _
Public Class Mail_ws
    Inherits System.Web.Services.WebService

#Region " Web Services Designer Generated Code "
    ……
#End Region

    <WebMethod()> _
```

```
        Public Function Notify(ByVal strXml As String) As String
        Dim ms As New MailSys
        Return ms.sendMail(strXml)
    End Function
```

一個 Façade 樣式就像一棵小樹的基因，依據樣式而不斷自我重複，逐漸成長，成爲巨大的樹狀體系，如下圖：

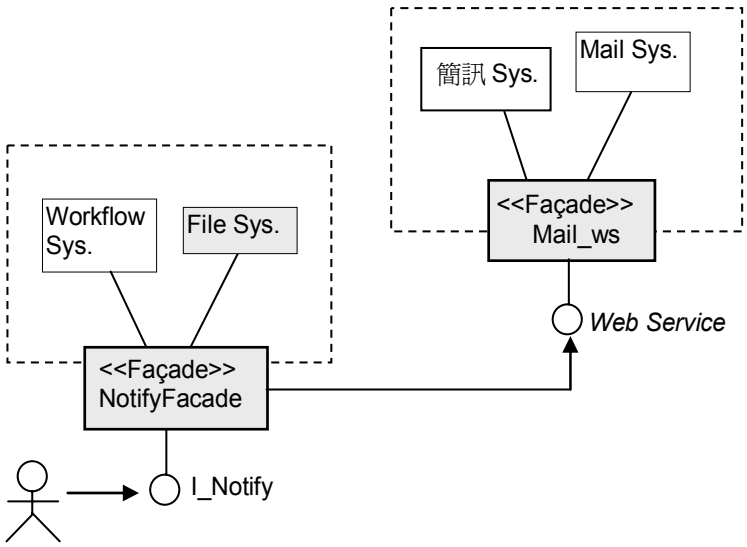


圖 24　簡單組合、自我重複、逐漸成長

既然是自我重複，那麼就重複設計更多的 Façade 樣式：Notify。於是，建立 EX01_03_c 程式庫(Class Library)項目，其內容爲：

```
'EX01_03_c
'Notify.vb
Imports System.Web.Mail
Imports System.Xml
```

```
Imports NotifyService

Public Interface INotify
    Function Notify(ByVal strXml As String) As String
End Interface

Public Class NotifyFacade
    Implements INotify

    Public Function Notify(ByVal strXml As String) As String
                                        Implements INotify.Notify
        Dim replyXml, ss As String
        Dim ms As New FileSys
        replyXml = "<Reply>"
        ss = ms.Notify(strXml) '呼叫 FileSys 建立.Xml 文檔
        replyXml += ss
        '-----------------------------------------------
        Dim wso As New localhost.Mail_ws '呼叫 Mail_ws 網路服務
        ss = wso.Notify(strXml)
        replyXml += ss
        replyXml += "</Reply>"
        Return replyXml
    End Function
End Class

'NotifyFaçade 內含之 FileSys 系統
Public Class FileSys
    Private mMsg As MailMessage
    Private mailFrom, mailTo As String

    Sub New()
        mMsg = New MailMessage
        mailFrom = "misoo.tw@msa.hinet.net"
        mailTo = "yoo161@yahoo.com.tw"
    End Sub
    Function Notify(ByVal notifyXml As String) As String
        Try
            Me.writeRecord(notifyXml)
        Catch ex As Exception
            Return "<FileStatus>" + ex.Message.ToString + "</FileStatus>"
```

```
        End Try
        Return "<FileStatus>ok</FileStatus>"
    End Function

    '寫入.Xml 文檔
    Sub writeRecord(ByVal recXml As String)
        Dim xml As New XmlDocument
        xml.LoadXml(recXml)
        Dim nlist As XmlNodeList
        Dim m_node, mChildNode As XmlNode
        Dim subject, body As String

        nlist = xml.SelectNodes("/Record")
        Dim st As String
        For Each m_node In nlist
            For Each mChildNode In m_node.ChildNodes
                Select Case mChildNode.Name
                    Case "Subject"
                        subject = mChildNode.InnerText
                    Case "Body"
                        body = mChildNode.InnerText
                End Select
            Next
        Next
        '-------------------------------------
        Dim dtime, ss As String
        dtime = Me.getStrTime
        ss = "c:/temp/" + "Alice-" + dtime + ".xml"
        Dim objXMLTW As New XmlTextWriter(ss, Nothing)
        objXMLTW.WriteStartDocument()
        objXMLTW.WriteStartElement("MailRecord")

        objXMLTW.WriteStartElement("Time")
        objXMLTW.WriteString(dtime)
        objXMLTW.WriteEndElement()

        objXMLTW.WriteStartElement("From")
        objXMLTW.WriteString(mailFrom)
        objXMLTW.WriteEndElement()
```

```
        objXMLTW.WriteStartElement("To")
        objXMLTW.WriteString(mailTo)
        objXMLTW.WriteEndElement()

        objXMLTW.WriteStartElement("Subject")
        objXMLTW.WriteString(subject)
        objXMLTW.WriteEndElement()

        objXMLTW.WriteStartElement("Body")
        objXMLTW.WriteString(body)
        objXMLTW.WriteEndElement()

        objXMLTW.WriteEndElement() 'End top level element
        objXMLTW.WriteEndDocument() 'End Document
        objXMLTW.Flush() 'Write to file
        objXMLTW.Close()
    End Sub

    '取得自訂格式之 DayTime
    Private Function getStrTime()
        Dim now As DateTime
        now = DateTime.Now
        Dim ss, yy, mmm, ddd As String
        Dim mm, dd As Integer
        mm = now.Month
        If mm > 9 Then
            mmm = mm.ToString
        Else
            mmm = "0" + mm.ToString
        End If
        dd = now.Day
        If dd > 9 Then
            ddd = dd.ToString
        Else
            ddd = "0" + dd.ToString
        End If
        ss = now.Year.ToString + "-" + mmm + ddd
        Dim min, sec As String
        min = now.ToShortTimeString.Substring(3, 2)
        sec = now.ToShortTimeString.Substring(6)
```

```
            Return ss + "-" + min + sec
        End Function
End Class
```

在實體上，NotifyFacade 樣式與 Mail_ws 樣式是分開的，但在邏輯上是 NotifyFacade 包含 Mail_ws，如下圖：

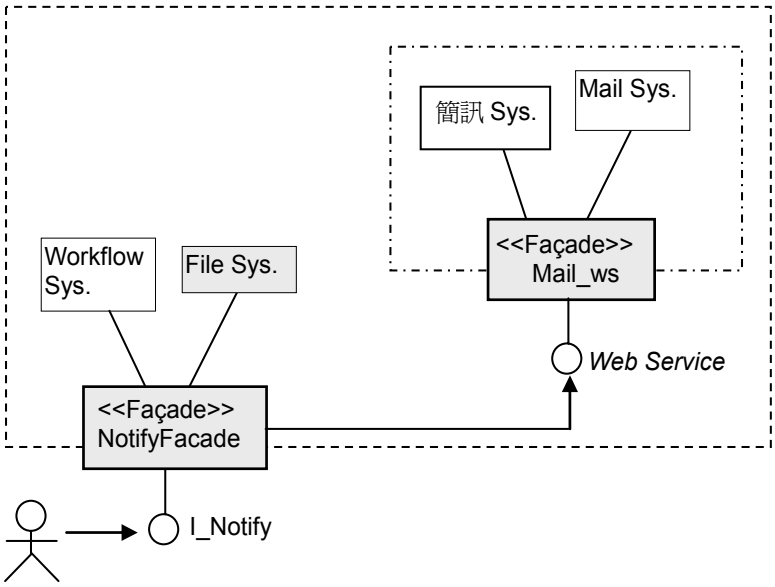


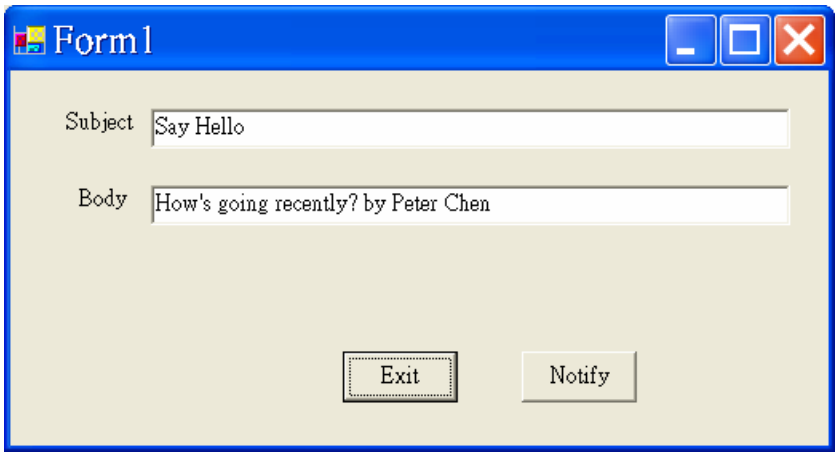
圖 25　如同踏大樹幹包含小樹枝一般

上圖 24 和 25 是兩個不同的觀點，一種實體結構，能呈現出多種觀點下的不同風貌。一般而言，設計者很重視多觀點的綜合，施工者重視實體的結構；兩者互補才能做好系統整合之任務。

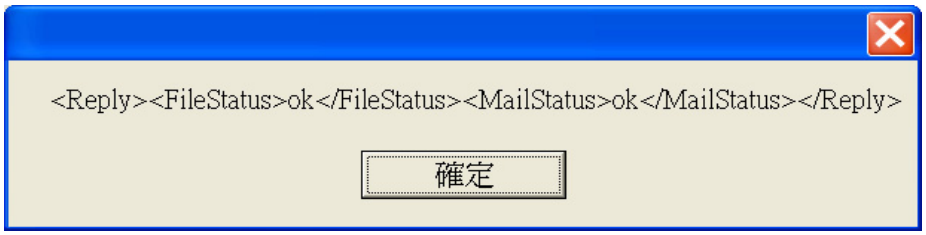
接著，設計 Windows Form 類別。也就是開發 EX01_03_d 項目，含 Client 程式來使用 INotify 接口。例如下述的 Windows Form 程式：

```
'EX01_03_d
'Form1.vb
Imports NotifyService
Public Class Form1
    Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "
.....
#End Region

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
                System.EventArgs) Handles MyBase.Load
        SubjectTx.Text = "Say Hello"
        BodyTx.Text = "How's going recently? by Peter Chen"
    End Sub

    Private Sub NotifyBtn_Click(ByVal sender As System.Object, ByVal e
                As System.EventArgs) Handles NotifyBtn.Click
        Dim x As NotifyFacade
        x = New NotifyFacade
        Dim strXml, replyXml As String
        strXml += "<Record><Subject>" + SubjectTx.Text + "</Subject>"
        strXml += "<Body>" + BodyTx.Text + "</Body></Record>"
        replyXml = x.Notify(strXml)
        MessageBox.Show(replyXml)
    End Sub

    Private Sub ExitBtn_Click(ByVal sender As System.Object, ByVal e
                As System.EventArgs) Handles ExitBtn.Click
        Me.Close()
    End Sub
End Class
```

此程式執行時，呈現如下畫面：



想像成要送出一封 E-Mail，其中的 Subject 是通知(Notification)的主題，而 Body 則是其內容。按下＜Notify>按鈕之後輸出：



表示已經完成通知之任務了。

從這個 VB.net 例子中，您會發現到，由於不斷自我重複，所以系統設計者也是不斷重複應用 Façade 樣式。每個 Façade 樣式就像茶杯(還可以內含小茶杯)，隨時能把杯內的東西(即實物件)倒掉換新。由於杯內的部分(即 Façade 所內含的子系統)能迅速變化、新陳代謝、自由成長。在設計整合系統時，必須充分發揮這種虛實相依性，讓每一個子系統都自由地新陳代謝、獨立成長，就能生生不息了。如下圖：

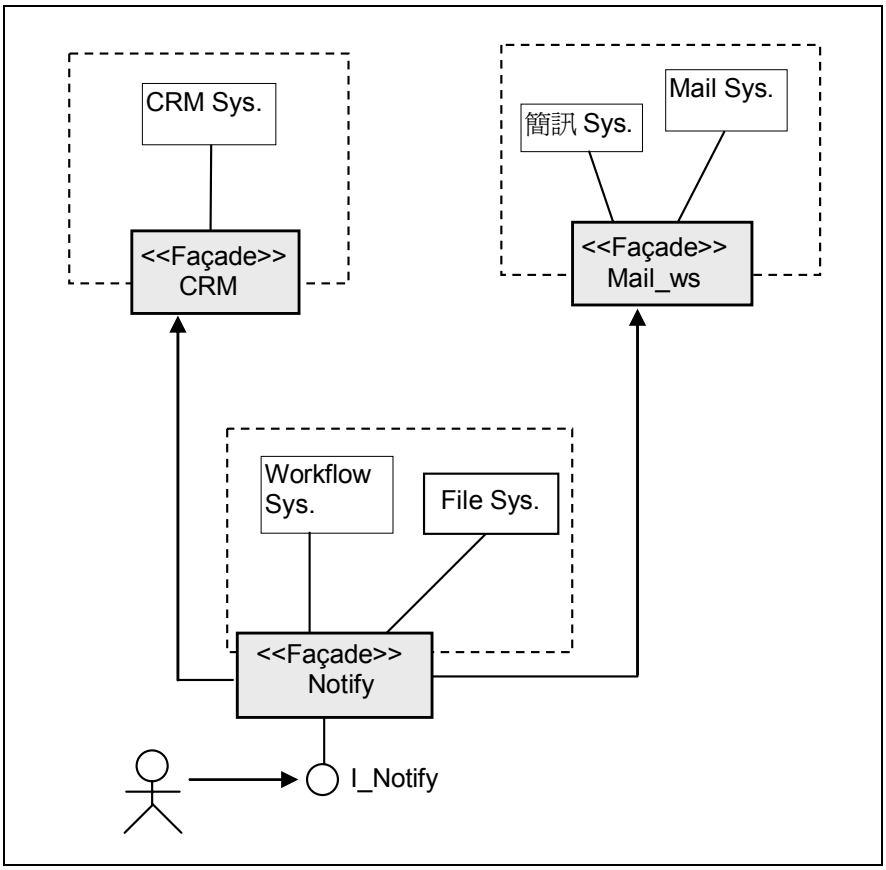


圖 26　猶如一棵樹，不斷成長與繁榮

Façade 樣式的重複，形成簡單的樹狀結構，也創造了樹狀的有機次序(Order)，此序包含了許許多多的細節變化(Change)，豐富了整體之內涵，如下圖：

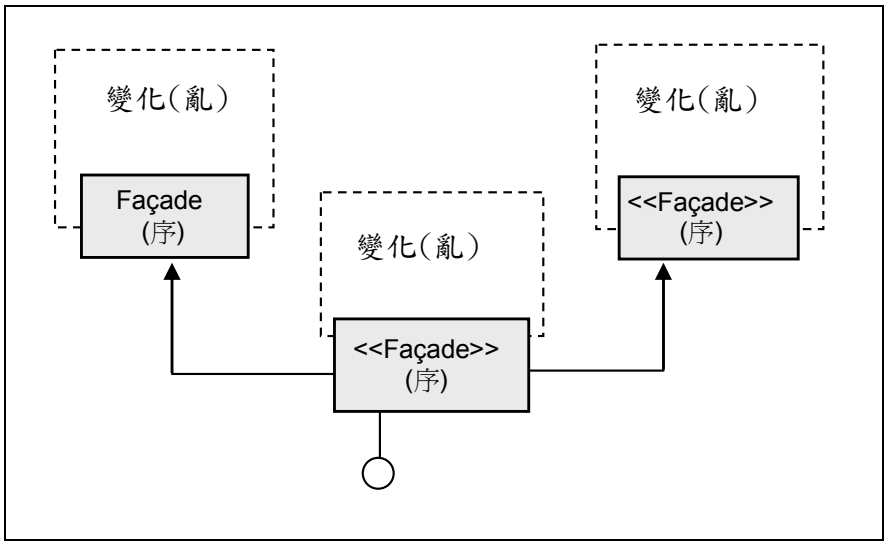


圖 27　Façade 自我重複之序

因爲 Façade 支持 Web 標準接口，而締造出「序」。猶如十字路口的「紅綠燈與斑馬線」，扮演行人與汽車雙方的接口，讓社會(即人、車的整合體)呈現出井然有「序」，帶來和諧的整體。同理，Façade 締造出的序，也能帶來系統整合的順暢與和諧。依循 Façade 的自我重複，隨著系統的整合成長，成長之後的大系統仍舊維持整體和諧。因之，擅用 Façade 自我重複是一項重要的大型系統整合工法。

1.5 人人天賦的整合心法

5.1 前言

系武林大俠用劍之道：以心使劍。意謂著心中有劍，心無旁騖，進而達到美之境界。所謂心法就是思考的樣式(Thinking Pattern)，敘述專家履試不爽的思路，引導別人舉一反三，創造自己的新思路。心法是工法的基礎，讓人們有能力去修正舊工法或創造新工法。

由於系統整合的關鍵是接口，在本節裡，所要介紹的心法是指專家設計接口時的思維過程。讓您瞭解接口從哪裡來，如何提升接口設計的品質與美感等等。

在本節裡，先介紹第 1 項心法---- **亂中找(有)序**，這是傳統系統『分』析(Analysis)的基本思維，從眾多物件之中加以分門別類，而找出「類別」(Class)，表達一群物件之共同行爲(次序)；今天的 VB.Net、C#、Java 等語言的 Class 機制就是源自於此。

然後介紹第 2 項心法---- **序中有亂(變化)**，這是一般建築師(Architect)的綜『合』(Synthesis)創意、整體規劃之思維，將複雜多樣的物件包裝起來，呈現出特定的整體行爲(序)。“**Architect**”一詞，在建築業裡稱爲「建築師」，在軟體業裡稱爲「架構師」。“**Architecture**”一詞，在建築業裡稱爲「建築」，在軟體業裡稱爲「架構」。

在當今，最著名的架構師是微軟老闆---- 比爾.蓋茲(Bill Gates)。自從 1999 年 5 月卸下微軟執行長(CEO)職務之後，他就擔任微軟的 CSA(Chief Software Architect)角色，領導軟體的整合工作。電腦業作家 David Bank 在他的書裡，描述 Bill Gates 所擔任的整合任務，他寫到[Bank02]：

「微軟是由資產、關係和人組成的有機體，經由數千、數百萬個微小的

決策，不斷適應和駕奴環境。沒有一套策略能在所有的環境中永遠有效，…..必須因時制宜。不管整套策略是什麼，它必須合乎蓋茲(Bill Gates)所想。蓋茲已經交出日常的經營控制權，不過他的長期心智模式依然用來綜合判斷所有的新信息。他是微軟單一的『整合』點，微軟依舊是 Bill Gates 的投影。」

以上兩種心法，前者重視「分」，後者重視「合」，兩者都是接口設計的基本思維，而且互補，成爲系統整合的基本心法。例如，傳統 OOAD 分析師，在複雜(亂)的企業應用領域裡，亂中找序，如下圖：

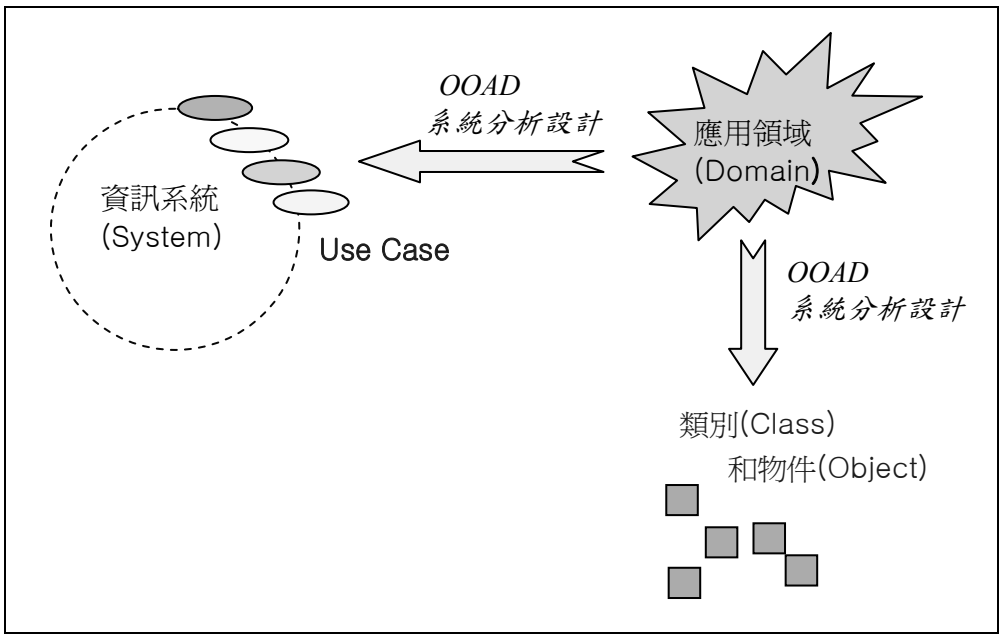


圖 28　分析師：亂中找序

OOAD(Object-Oriented Analysis and Design)是從企業領域知識(Domain Knowledge)解析出基本的元素---- 物件(Object)，然後將之「分」門別類

(Classify)成爲類別(Class)，呈現出同類物件的共同行爲(即序)，成爲同類物件的共同接口。此外，OOAD 也針對 User 的期望而解析出基本功能----即 Use Case。

接著，架構師規劃出 Façade 等軟體貨櫃，將上述不同類別之物件包容起來，形成較高層級的軟體模組(通稱爲 Component)，透過自我重複和無限複製而締造出大型系統之整合次序。如下圖：

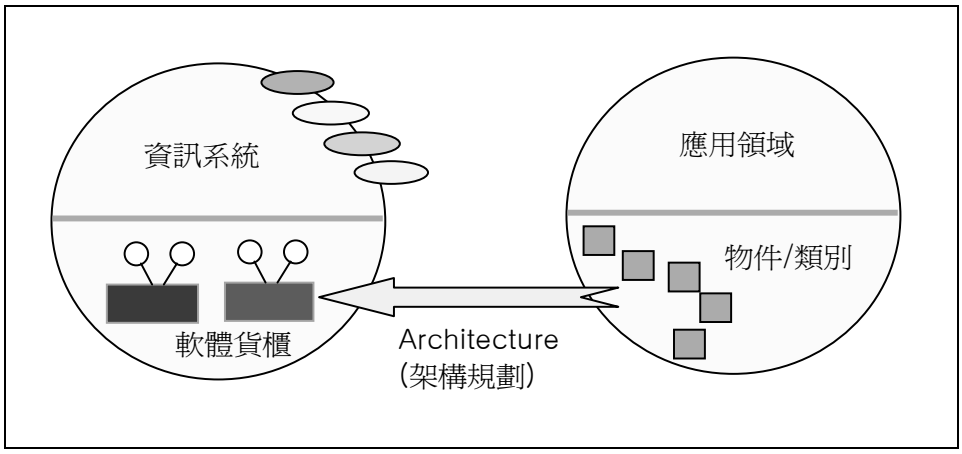


圖 29　架構師：序中有亂(締造整體之序)

從大家熟悉的建築業裡，很容易看出分析師與架構師之間的互補關係，如下圖 30 所示。涼亭是最終的整體產物(Whole)，可滿足業主的期望(Goal)。涼亭的屋頂、樑、柱、地毯、地板等都是涼亭的建築組件。自然界是建築材料的來源領域(Domain)。磁磚、大理石、木材、 布、砂石等則是從自然界取得的元素。

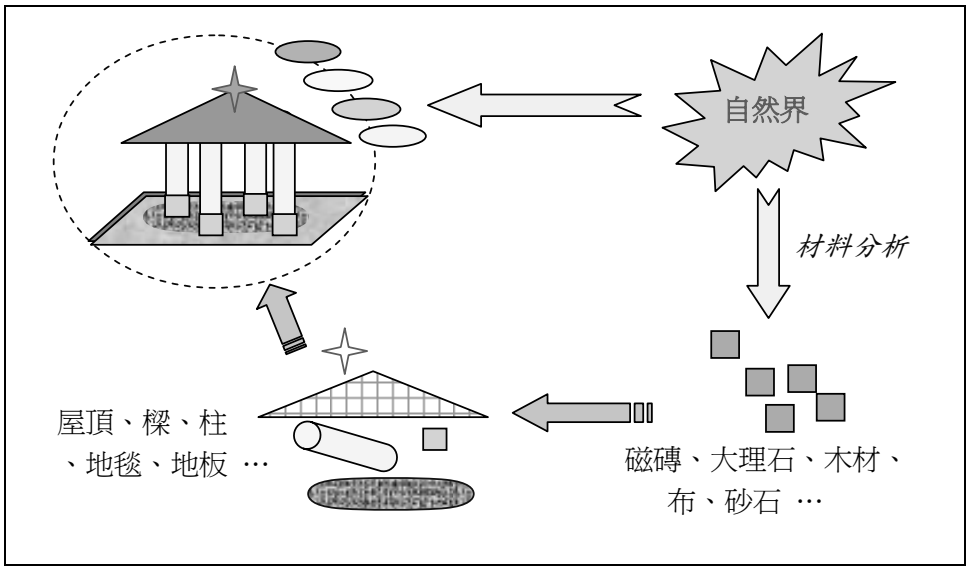


圖 30　建築業的分析與整合互補關係

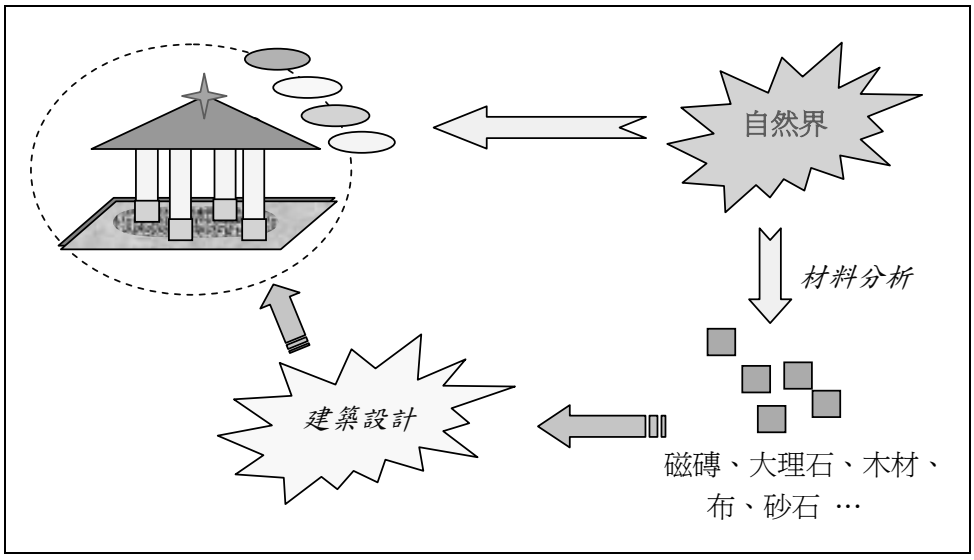


圖 31　建築師：序中有亂(締造整體之序)

在軟體業裏，一般 OOAD 系統分析技術，已經在「分」的層面提供

了高品質的建造環境，而且技術也很成熟了，讓我們無後顧之憂，能夠專注在組裝出「氣象萬千」的大型軟體系統上。因而近十幾年來，把焦點轉移到系統的架構設計(Architecture)上。例如，2003 年推出的 UML2.0，就新增了架構圖；再如，IBM 公司也將 Rational Rose 設計環境改名爲 Rational Software Architect。由於這是近年來蓬勃發展的軟體整合技術，所以本書特別注重於架構師的整合工法和心法上。

1.5.2 心法 1：亂中有序

系武林大俠用劍之道：以心使劍。意謂著心中有劍，心無旁騖，進而達到美之境界。所謂心法就是思考的樣式(Thinking Pattern)，敘述專家履試不爽的思路，引導別人舉一反三，創造自己的新思路。心法是工法的基礎，讓人們有能力去修正舊工法或創造新工法。

於此，先介紹第 1 項心法---- 亂中找(有)序，這是傳統 OOAD 的基本思維，從眾多物件之中加以分門別類，而找出「類別」(Class)；今天的 VB.Net、C#、Java 等語言的 Class 機制就是源自於此。

5.2.1 整合之意義：分合自如

系統整合不只追求一時的連結，更要追求持久的和諧與成長。爲了達成這個目標，系統的新陳代謝必須順暢，也就是，系統內之物件必須能隨時迅速汰舊換新、分合自如。此汰舊換新之效果通稱爲 PnP(Plug and Play)，即系統內之物件易於抽換與易於修改，簡而言之，就是：“Easy to change”。

爲了實現 PnP，我們必須進行接口設計，就如同汽車的輪胎能隨時抽換(PnP)，其依賴標準的輪盤(即接口)設計。也是因爲能隨時抽換輪胎

(即分合自如)，所以汽車能持久保持整體的和諧。PnP(即分合自如)是目的，而接口是手段。接口設計之優劣關係到 PnP 之效果。就如 Rogerson 在其 “Inside COM” 書裡提到[Roger97]：

> “Interfaces protect the system from being crippled by change.”
> （接口讓系統不會因某部份改變而造成癱瘓）

例如上述的汽車輪胎壞了，只要從接口(如輪盤)卸下壞輪胎，換上新輪胎就讓汽車恢復了。輪盤接口之存在價值就在於：汽車的部份(如輪胎)受損了，只要 PnP、迅速換上新輪胎，就不牽累了整部汽車，避免汽車癱瘓。他又說：

> “Interfaces allow the client to treat different components in the same manner. The capability of different components to be treated in the same manner by a client is known as polymorphism.”
> (接口讓 Client 能以一視同仁的態度來對待不同的元件，這種功能稱爲多形性。)

就輪胎的角度而言，汽車是輪胎的 Client，它正使用輪胎，接受輪胎的服務。當 NISSAN 汽車能以一視同仁的態度來對待不同廠牌的輪胎(如固特異輪胎、南港輪胎)時，也就是不同廠牌的輪胎，只要合乎輪盤接口，就能 PnP 到汽車上，這就是多形性了。所以多形性與 PnP 的涵意是一致，都意味著：做好接口設計，促進物件之分合自如，讓系統不會因某部份改變而造成癱瘓，反而新陳代謝順暢、生生不息。

5.2.2 接口：變與不變之「分」界

在上一節裏說過，人們擅長於設計各式各樣的接口，並以接口爲手段，將一群物件或系統整「合」起來，並且展現持久的和諧。此時您可能會問到：接口從何而來呢？如何設計接口呢？如果把接口當目的看，則其手段又是什麼呢？答案是：最常見的手段是---- 分離「變」與「不變」。茲拿日常生活中的例子來說明之。有一家火鍋店，店內有幾張火鍋桌子，如下圖所示。

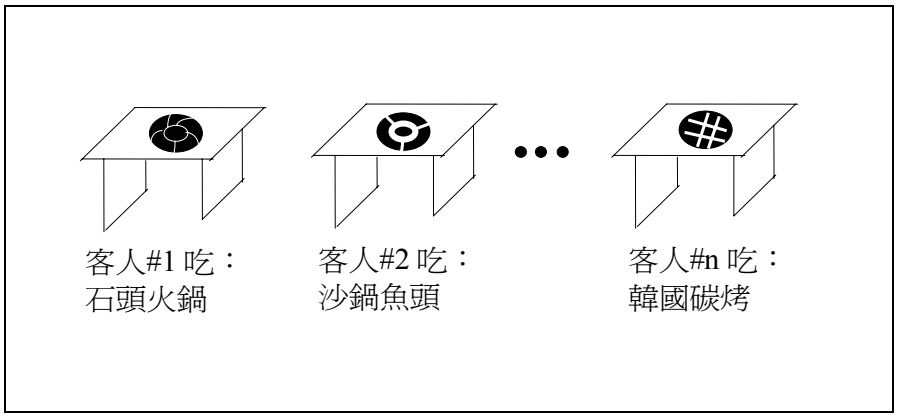


圖 32　火鍋店的桌子

有些客人吃石頭火鍋，有些客人吃沙鍋魚頭，有些客人吃韓國碳烤。有時候吃石頭火鍋的客人多，有時候吃韓國碳烤的客人多。爲了滿足這些多樣化的需求，不得不對火鍋桌子動些腦筋、仔細觀察：火鍋桌子，除了鍋子部份不一樣之外，其餘是相同的；於是將鍋子與桌子分離開來，並得出一致的接口，如下圖所示：

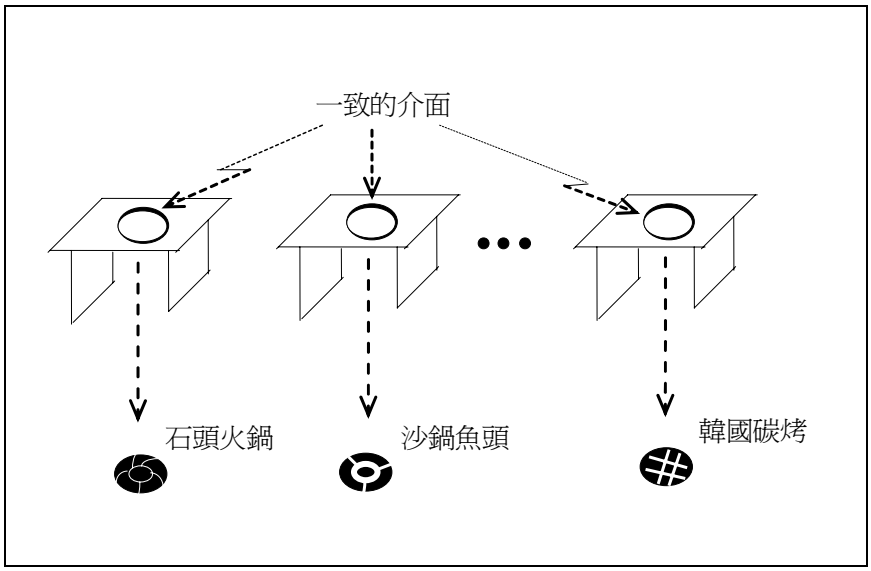


圖 33　將差異部份抽離獨立出來

把變異部份抽離出來（在桌面上挖一個洞）之後，剩下的部份就相當一致了，設計師只需要設計一款桌子就可以了。將之對應到軟體系統上，就是程式師只需要定義一個桌子類別(Desk Class)就行了，並不需要爲石頭火鍋、沙鍋魚頭、韓國碳烤等各設計其專用的桌子類別。

至於鍋子部份，因爲石頭火鍋、沙鍋魚頭、韓國碳烤等各有所不同，所以必須個別設計。也就是必須撰寫好幾個火鍋類別，如下圖 34 所示。

「抽離出多樣化部分」相當於「抽象(Abstract)出一致性部份」，兩者所指的是同一個動作。此動作(即分離或抽象)自然而然會誕生出『接口』(Interface)，精緻的抽象能得出美好的接口。誕生美好接口之後，當客人上門了，桌子與鍋子就能一拍即合。

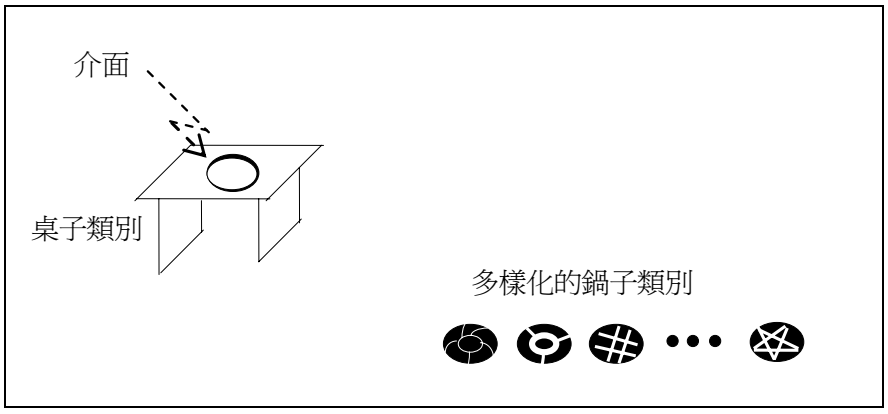


圖 34 一致性的元件與多樣性的元件

類似的接口，在日常生活中俯拾可見，例如大家最熟悉的接口：插頭與插座，如下圖所示：

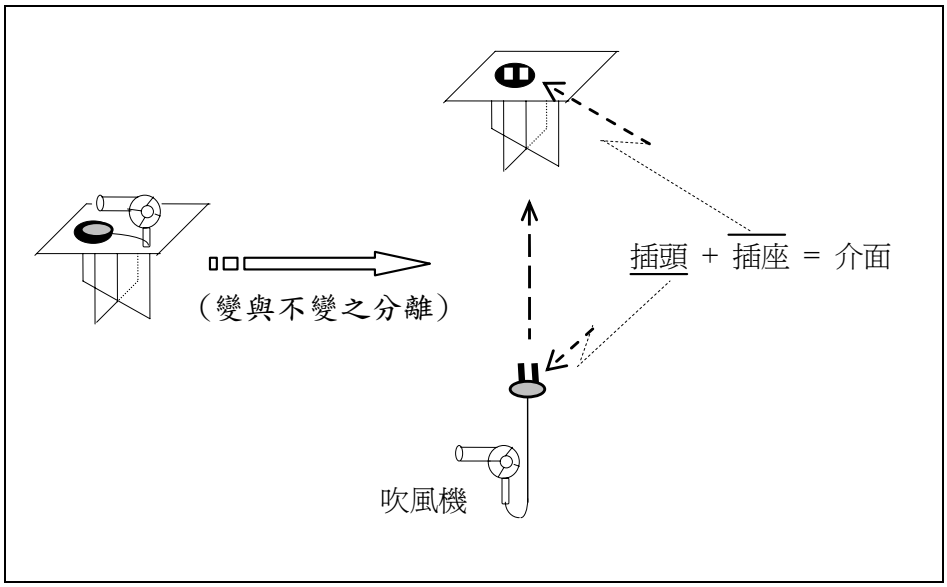


圖 35 一致性的元件與多樣性的元件

所以接口是變與不變的分界，而接口設計(Interface Design)的本質就是：變與不變的分離。此種分離就是一種「抽象」(Abstraction)的動作。因之，人們常說：抽象出穩定的接口。

5.2.3 接口：跨時空一拍即「合」

剛才說過，有了美好接口之後，當客人上門了，桌子與鍋子就能一拍即合。如下圖：

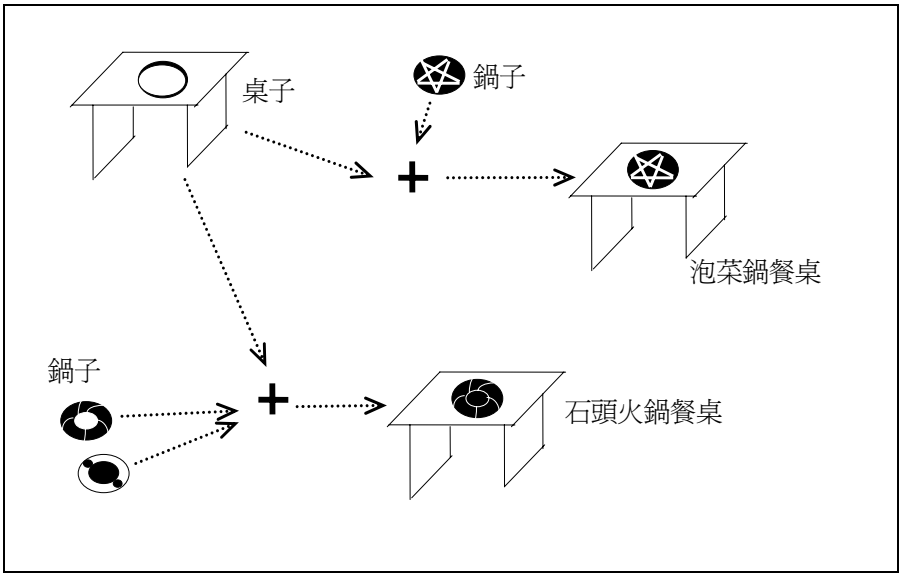


圖 36 一致性的元件與多樣性的元件

接口不僅能支持多樣化的鍋子，也能支持多樣化的桌子。任何桌子只要它也提供與上述相同之鍋子接口，則上述的鍋子都能跟它一拍即合，並且分合自如。如下圖所示：

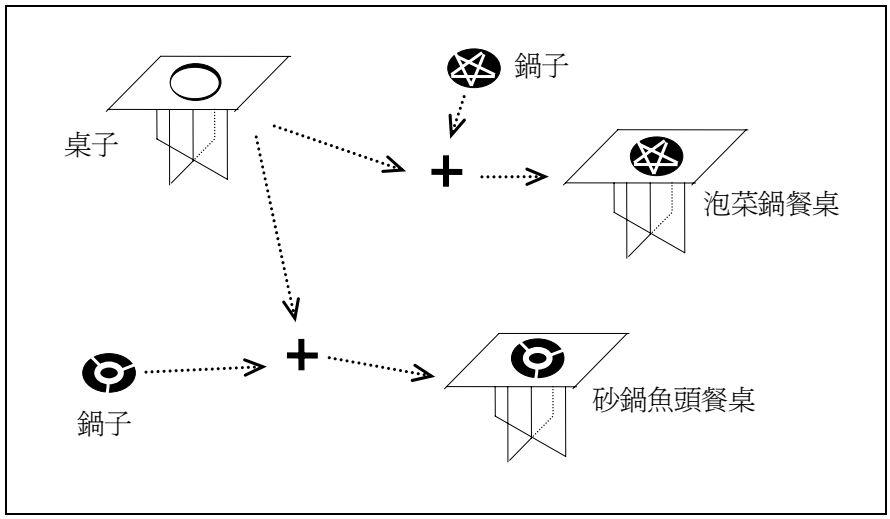


圖 37　一致性的元件與多樣性的元件

這樣的餐桌，如果也提供插頭接口的話，吹風機也能與餐桌一拍即合。如下圖：

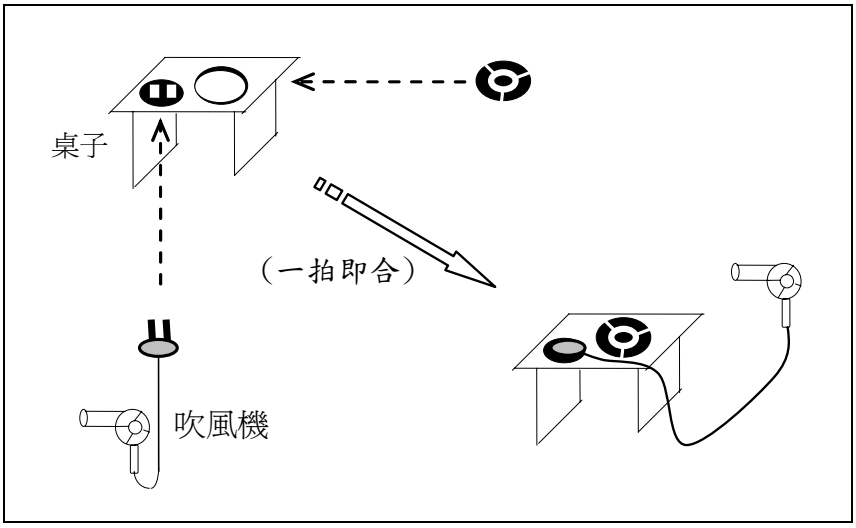


圖 38　透過各種接口整合異質性系統

上圖的整合，可滿足特殊客戶的偏好，例如：把吹風機當風扇用，將木炭的火煽得更旺。如果有些客人不喜歡吹風機，也能隨時 PnP，將吹風機抽頭拔掉，或是更換其它電器等等。

於是，您可更深刻體會到接口設計的基本技巧了：將穩定與善變「分」離開來，就在穩定的基礎結構上塑造出虛的空間，來容納多樣性的（善變的）小元件，就組「合」成各式各樣的產品。

從「虛實相依」的哲理來看，接口是實的，代表一個虛的空間，可用來容納未來的變化花樣。亦即，容納改變（容易），讓我們容易接納、適應未來，更靈活、更具生命力！例如：

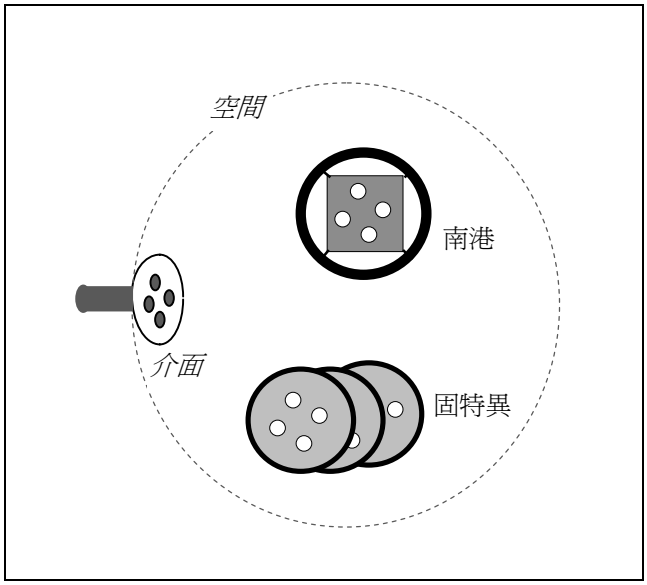


圖 39　接口代表一個空間

輪盤接口，代表一個虛的空間，凡是符合此接口的物件（包括未來可能會出現的新品牌輪胎），皆屬於該空間的元素。此空間內的元件會生生不息、推陳出新、更新版本，就是所有過去、現在、未來的多形性物件

的集合，就是此空間。這個空間是跨時間的，也是跨空間的。

透過接口空間容納更多別人未來的元件，不但整合了別人的元件，也整合別人源源不絕的設計智慧，讓自己成長更快。此外，也整合別人過去、現在、未來所設計的產品。如此，一方面讓自己的系統生生不息；另一方面讓別人的物件擁有更多用途和歷練、品質更加精美；回過頭來，更促進我們系統的品質與生命力，構成 Win-Win-Win ..... 的繁榮景象。因之，接口設計是系統整合的關鍵、是追求一拍即合、分合自如、PnP 的尚方寶劍。

5.3 抽象過程的三種產出

上一節說明了接口的設計手藝：變與不變之分離。在本節裡，將說明如何表達接口設計品，也就是說，如何呈現「變與不變分離」的結果。而且還能落實爲 VB.Net 的程式內涵。茲回憶上一節的火鍋店之例子，將其變與不變的分離過程歸納如下：

Step 1. 觀察幾個相似的東西(如火鍋餐桌)。

Step 2. 分辨它們的異同點。

Step 3. 把它們的相同點抽離出來(即是把相異點分離出來)。

這是一般通稱的「抽象」(Abstraction)過程，此過程會得到三種產出：

1. 接口：如插頭和插座。
2. 物件族群 1：如桌子族群。
3. 物件族群 2：如吹風機等電器族群。如下圖：

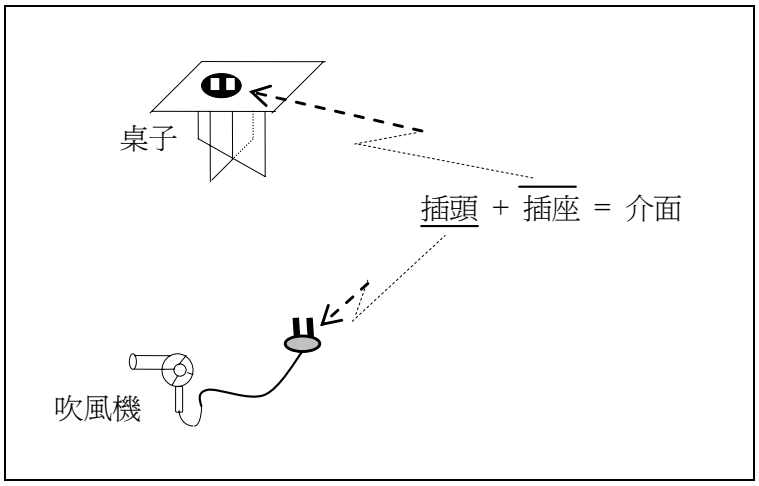


圖 40　抽象過程的 3 種產出

5.3.1　VB.Net 的表達

剛才提到的桌子和電器兩個族群，可使用 VB.Net 的一般類別繼承(Class Inheritance)來表示之。例如餐桌族群：

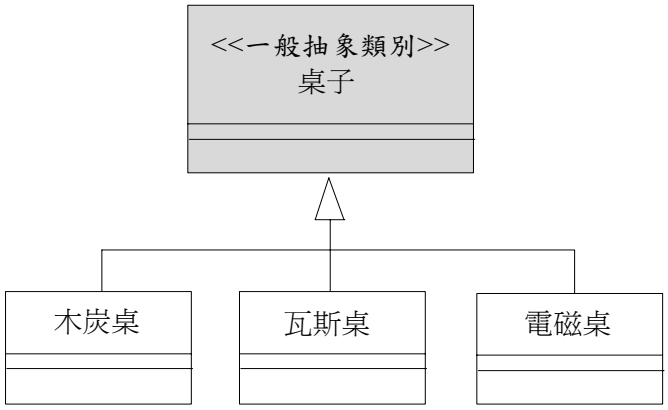


圖 41　桌子類別體系

再如電器族群：

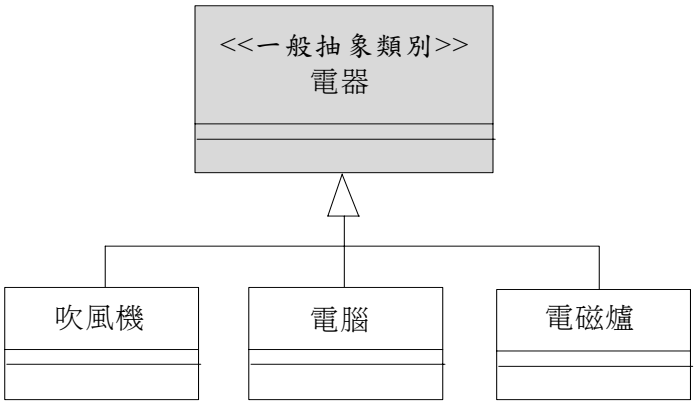


圖 42　電器類別體系

這能直覺地落實爲 VB.Net 的程式碼，如下：

```
Public Class 電器　'一般抽象類別
    Overridable Function PowerOn() As Boolean
        '……
    End Function
    Overridable Function getPowerType() As String
        '……
    End Function
End Class
Public Class 吹風機
    Inherits 電器
    ' ……
End Class
Public Class 電腦
    Inherits 電器
    ' ……
End Class
```

```
Public Class 電磁爐
    Inherits 電器
    '……
End Class
```

一般抽象類別就是一般的父類別(Super Class)，可衍生(Derive)出子類別，例如從電器父類別衍生出吹風機、電磁爐等子類別。這也是一般通稱的類別繼承關係。過去的 VB.Net 應用系統開發者大多把焦點擺在這種類別的撰寫上。現在，由於系統整合日漸重要，致焦點逐漸轉移到接口設計與表達上。在 VB.Net 裡，有兩個機制可用來表示接口：

1. 以純粹抽象類別(Pure Abstract Class)表達之。
2. 以 Interface 機制表達之。

所謂純粹抽象類別，就是該類別裡的每一個函數都是 MustOverride 函數。這種函數就是空的函數，只有定義而無實作指令。例如：

```
Public MustInherit Class 插頭    '純粹抽象類別
    Public MustOverride Function PowerOn() As Boolean
    Public MustOverride Function GetPowerType() As String
End Class

Public Class 電器    '一般抽象類別
    Inherits 插頭
    Public Overrides Function GetPowerType() As String
      '……
    End Function

    Public Overrides Function PowerOn() As Boolean
      '……
    End Function
End Class
```

以圖形表示如下：

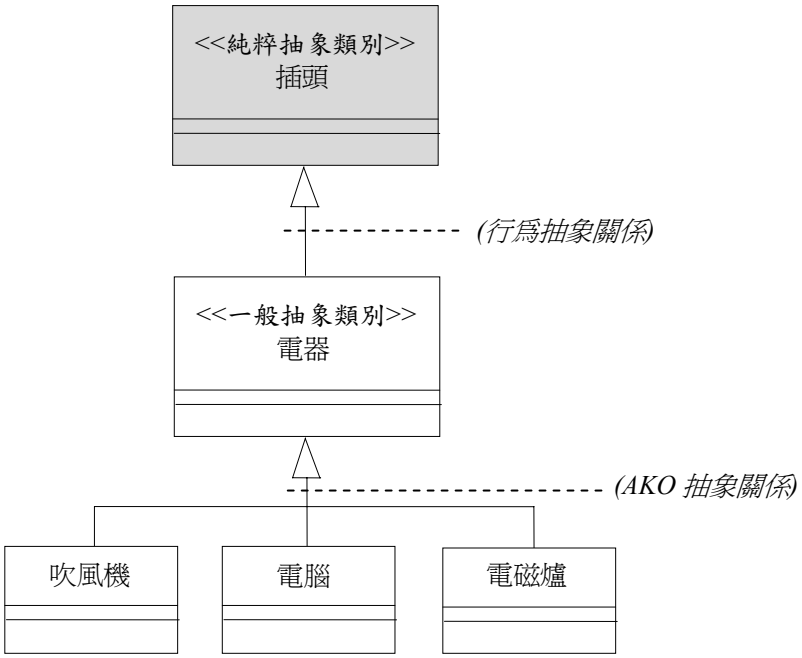


圖 43　以 VB.Net 抽象類別表示接口(一)

請留意，圖裡有兩種抽象關係。其中，AKO(A Kind Of)關係是傳統 OO(Object-Oriented)技術中的分類關係，例如電腦是一種(AKO)特殊的電器，吹風機也是一種(AKO)特殊的電器等，一般而言，這是屬於結構性的分類。至於，行爲抽象關係是屬於服務行爲的分類，通常依據服務目的而分類，提供服務給特定的顧客族群。例如，一個麥當勞餐廳可針對開車者而提供 Drive-In 的服務，而把跟開車者攸關的功能行爲抽象出來，匯集起來並取個名稱叫「Drive-In 服務台」，如下圖：

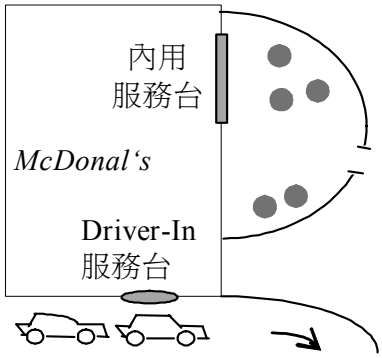


圖 44　麥當勞餐廳的兩個服務台

此外，麥當勞餐廳還能提供「內用」、「外帶」、「外送」等許許多多的服務台。再來看看另一個類別體系：桌子體系，如下圖：

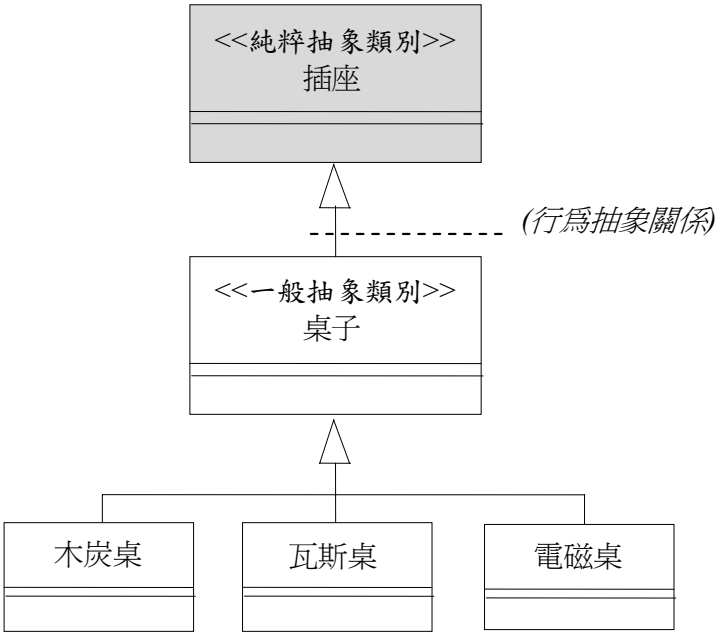


圖 45　以 VB.Net 抽象類別表示接口(二)

以 VB.Net 表達如下：

```
Public MustInherit Class 插座     '純粹抽象類別
    Public MustOverride Function GetPower() As Integer
End Class
Public Class 桌子    '一般抽象類別
        Inherits 插座
    Public Overrides Function GetPower() As Integer
        '……
    End Function
End Class
Public Class 木炭桌
    Inherits 桌子
End Class
Public Class 瓦斯桌
    Inherits 桌子
End Class
Public Class 電磁桌
    Inherits 桌子
End Class
```

從也就是說，我們可以對一個類別、物件、或系統做多種行爲觀點之抽象(Abstraction)。例如，將電器的取電源行爲抽象出來，取個名稱叫「插頭」。將電器的可攜帶行爲抽象出來，取個名稱叫「把手」。同樣地，如上圖 45 所示，將桌子的提供電源之各種行爲抽象出來，匯集起來，取個名稱叫「插座」。再如，將電器的可攜帶行爲抽象出來，取個名稱叫「把手」等等。

由於行爲抽象，是爲了服務特定顧客族群而進行的，也就是特定顧客族群取得服務的窗口，這種服務窗口就通稱爲「接口」(Interface)。所以上圖 44 就相當於：

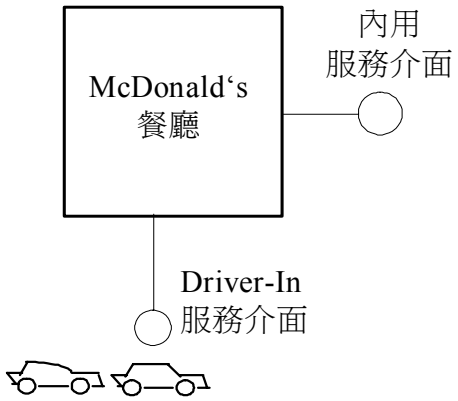
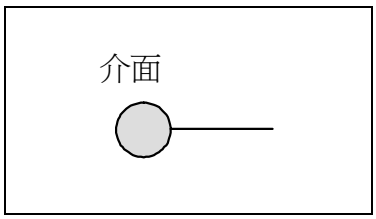


圖 46　麥當勞的服務接口

由於接口是系統整合的基礎，所以精致的行爲抽象方法可以得出好的接口，抽象是人們天賦的思維方法，因而成爲系統整合的基本心法。

在 1996 年之前，主流語言(如 C++)都只提供繼承(Inheritance)機制，而無 Interface 機制，所以當時只能用「純粹抽象類別」(C++稱之爲純粹虛擬類別)來表達接口。自從 1996 年 Java 問世之後，主要語言如 Java、VB.Net、C#等都提供了 Interface 機制來表達接口。有了新的 Interface 機制，人們也會創造使的接口符號：



於是，上圖 43 就相當於：

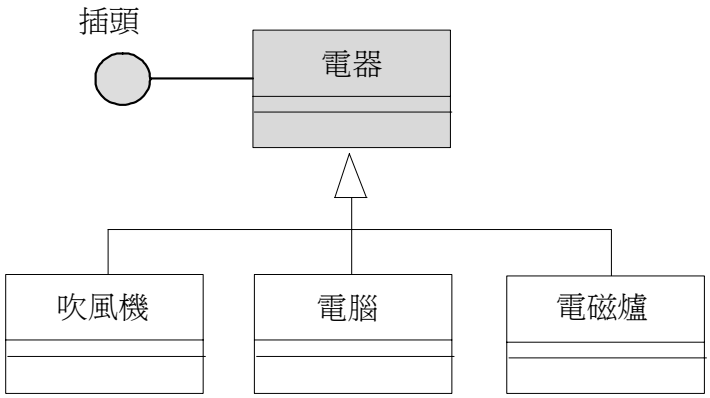


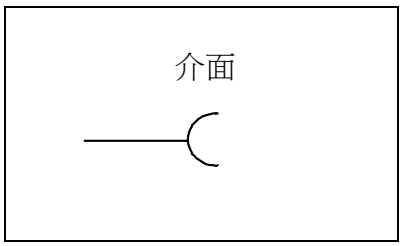
圖 47　使用 VB.Net 的接口機制

以 VB.Net 程式碼表達如下：

```
Public Interface 插頭    '純粹抽象類別
    Function PowerOn() As Boolean
    Function GetObjectType() As String
End Interface

Public Class 電器    '一般抽象類別
        Implements 插頭
    Public Function GetObjectType() As String
          Implements 插頭.GetObjectType
      '……
    End Function
    Public Function PowerOn() As Boolean Implements 插頭.PowerOn
      '……
    End Function
End Class
Public Class 吹風機
    Inherits 電器
End Class
```

```
Public Class 電腦
    Inherits 電器
End Class
Public Class 電磁爐
    Inherits 電器
End Class
```

另一個接口符號是：



例如：

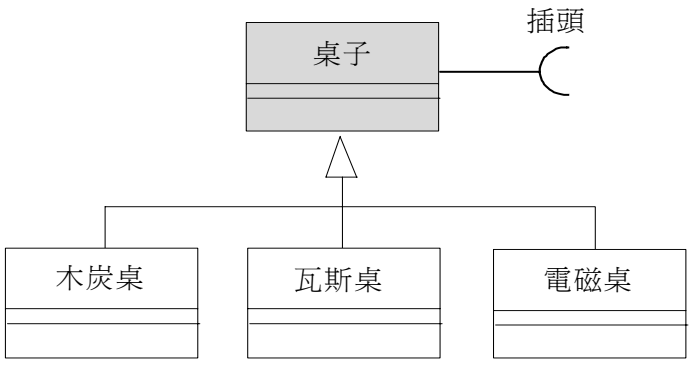
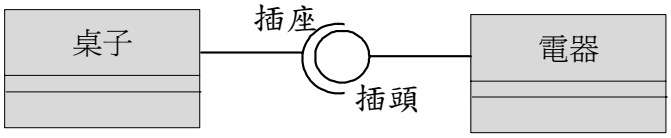
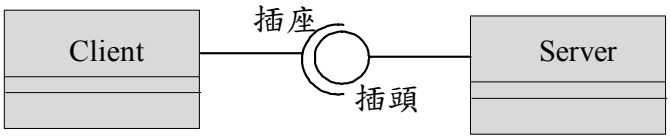


圖 48　使用 VB.Net 的接口機制

於是，桌子和電器就能透過接口而一拍即合了，如下：



兩個系統(例如桌子與電器)透過接口進行溝通與整合時，信息交流常是雙向的。信息交流事件的主動觸發者(Trigger)通稱爲 Client，而信息交流的服務提供者(Provider)稱爲 Server。如下圖所示：



Client 是服務的消費者或需求者；而 Server 是服務供應者或提供者。就好比台北市的地下鐵網路系統，以接口符號表示如下：

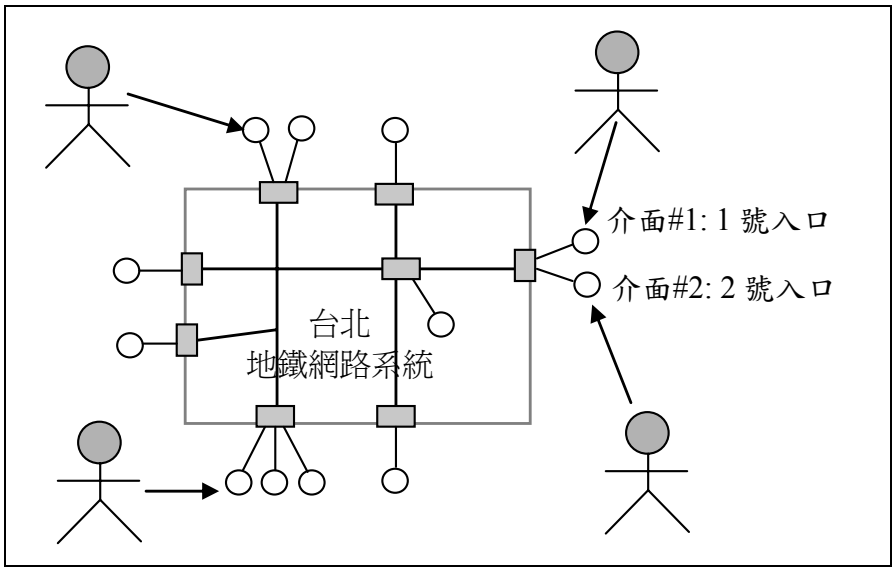
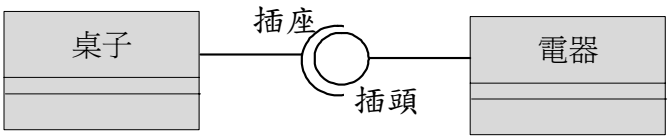
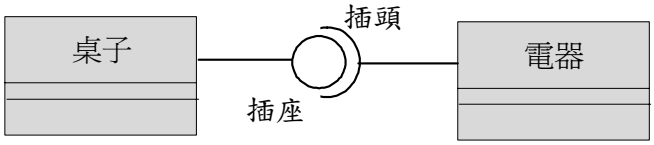


圖 49　地下鐵系統之接口

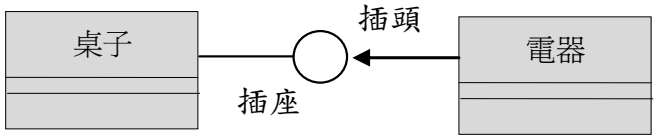
同樣地，當您認爲桌子含有電源，其透過插座/插頭界面傳遞電力給電器，而要求電器執行動作、提供服務。則桌子成爲 Client 而電器成爲 Server 了，就該畫圖如下：



反之，如果您認電器的插頭向桌子的插座取得電力的話，電器是 Client 而桌子是 Server，就可畫圖如下：



有許多人使用箭頭來取代 「)-------- 符號」。則上圖就相當於：



這表示桌子提供服務，而電器透過該接口而取得桌子的服務；例如從桌子的插座上取得電力供應。在 VB.Net 裡， 「------○符號」對應到 Interface 機制，如下之 VB.Net 程式碼：

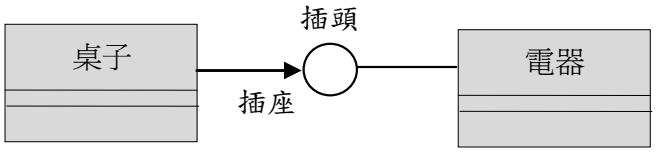
```
Public Interface  插座
    Function GetPower() As Integer
End Interface

Public Class  桌子   '一般抽象類別
    Implements  插座
    Public Function GetPower() As Integer Implements  插座.GetPower
       ' ……
    End Function
End Class
Public Class  木炭桌
    Inherits  桌子
End Class
Public Class  瓦斯桌
    Inherits  桌子
End Class
Public Class  電磁桌
    Inherits  桌子
 End Class
```

至於，「)-------- 符號」或「←」對應到如下程式結構：

```
Public Class  電器   '一般抽象類別
    Public Sub TurnOn()
        Dim  桌子物件  As  插座
        桌子物件  = New  木炭桌
        桌子物件.GetPower()
    End Sub
End Class
Public Class  吹風機
    Inherits  電器
End Class
Public Class  電腦
```

```
        Inherits 電器
    End Class
    Public Class 電磁爐
        Inherits 電器
    End Class
```

再來看看另一種結構：



「------○符號」對應到 Interface 機制，如下之 VB.Net 程式碼：

```
Public Interface 插頭
    Public Sub Activate()
End Interface

Public Class 電器   '一般抽象類別
    Implements 插頭
    Public Sub Activate() Implements 插頭.Activate
      ' ……
    End Sub
End Class
Public Class 吹風機
    Inherits 電器
End Class
Public Class 電腦
    Inherits 電器
End Class
Public Class 電磁爐
    Inherits 電器
End Class
```

至於，「)-------- 符號」或「←」對應到如下程式結構：

```
Public Class 桌子   '一般抽象類別
      Public Sub TurnOn()
            Dim 電器物件 As 插頭
            電器物件 = New 吹風機
            電器物件.Activate()
      End Sub
End Class
Public Class 木炭桌
      Inherits 桌子
End Class
Public Class 瓦斯桌
      Inherits 桌子
End Class
Public Class 電磁桌
      Inherits 桌子
End Class
```

從上所述，您可理解到：

1. 精緻行爲抽象能得到美好的接口，
2. 美好的接口確保系統整體之和諧，
3. 美好的接口確保系統元件之 PnP。

5.3.2 VB.Net 如何表達基於接口之整合與 PnP

上一節說明了一般的「抽象」過程如下：

Step-1 「分」----- 變與不變分離，分出三個要素。

例如，分出車體、接口和輪胎體系。如下：

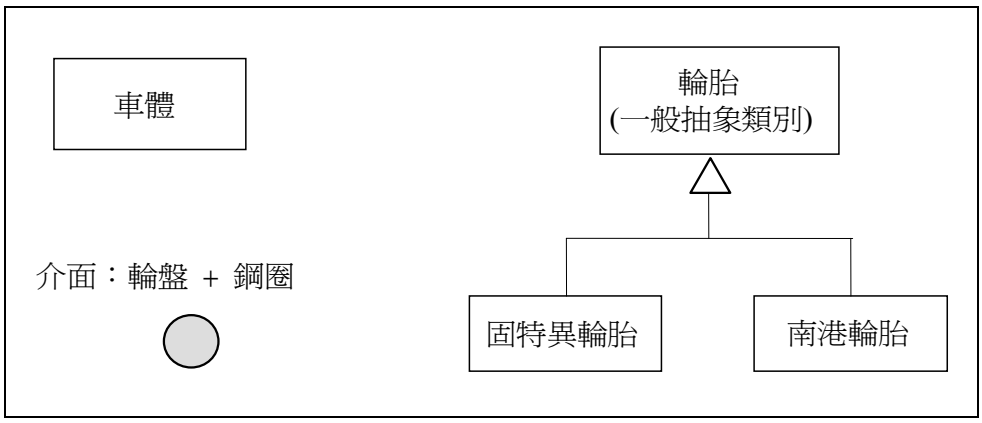


圖 50 分出三個要素

Step-2 「合」----- 透過接口，一拍即合。

例如，以車體爲 Client、輪胎爲 Server，整合如下：

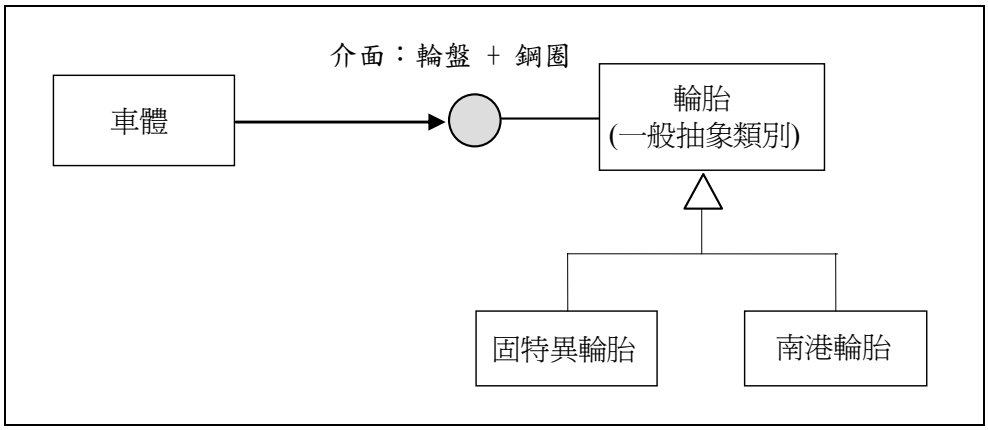


圖 51 一拍即合

再如，

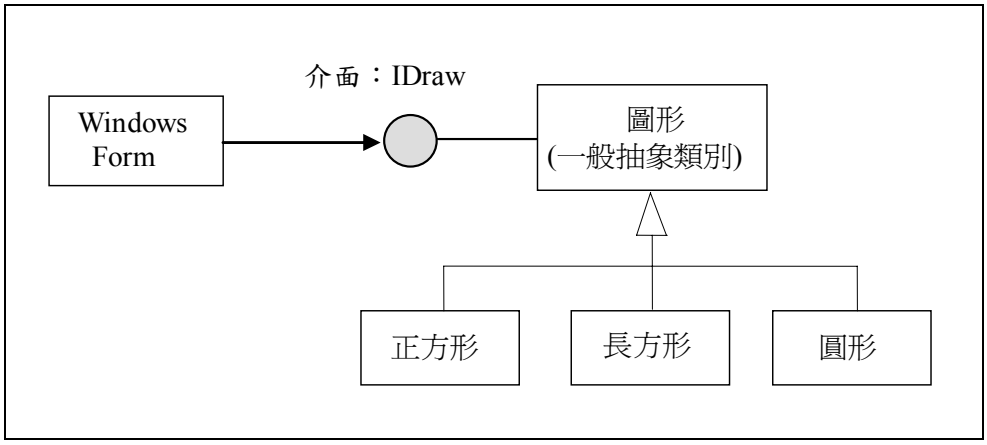


圖 52　PnP(抽換性)

Step-3. 「實現」----- 落實爲 VB.Net 程式碼。

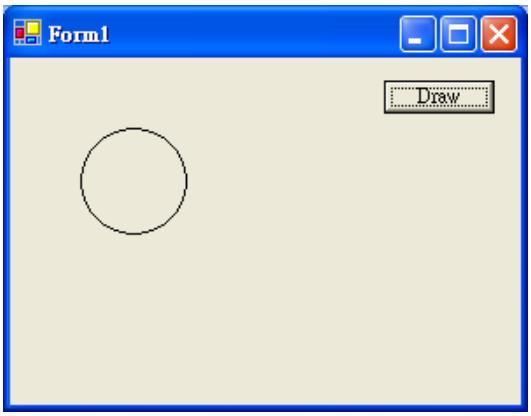
建立一個 Windows Application: EX001 項目。其 VB.Net 程式碼如下：

```
'EX004
Public Interface IDraw
    Sub Draw()
End Interface

Public MustInherit Class Shape
                Implements IDraw
    Public Shared Function CreateObject(ByVal f As Form1, ByVal x0 As Integer,
                ByVal y0 As Integer) As IDraw
        Return New Circle(f, x0, y0)
    End Function
    Public MustOverride Sub Draw() Implements IDraw.Draw
End Class

Public Class picSquare
```

```
            Inherits Shape
    Private frm As Form1
    Private x, y As Integer
    Private side As Integer
    Public Sub New(ByVal f As Form1, ByVal x0 As Integer, ByVal y0 As Integer)
        Me.frm = f
        Me.x = x0
        Me.y = y0
        Me.side = 70
    End Sub
    Public Overrides Sub Draw()
        '畫正方形
        Dim pen As New Pen(Color.DarkBlue)
        Dim gr As Graphics = frm.CreateGraphics()
        gr.DrawRectangle(pen, New Rectangle(x, y, side, side))
        pen.Dispose()
        gr.Dispose()
    End Sub
End Class

Public Class picRectangle
                Inherits Shape
    Private frm As Form1
    Private x, y As Integer
    Private width, length As Integer
    Public Sub New(ByVal f As Form1, ByVal x0 As Integer, ByVal y0 As Integer)
        Me.frm = f
        Me.x = x0
        Me.y = y0
        Me.width = 100
        Me.length = 50
    End Sub
    Public Overrides Sub Draw()
        '畫長方形
        Dim pen As New Pen(Color.DarkRed)
        Dim gr As Graphics = frm.CreateGraphics()
        gr.DrawRectangle(pen, New Rectangle(x, y, width, length))
        pen.Dispose()
        gr.Dispose()
    End Sub
```

```
End Class

Public Class Circle
        Inherits Shape
    Private frm As Form1
    Private x, y As Integer
    Private radius As Integer
    Public Sub New(ByVal f As Form1, ByVal x0 As Integer, ByVal y0 As Integer)
        Me.frm = f
        Me.x = x0
        Me.y = y0
        Me.radius = 60
    End Sub
    Public Overrides Sub Draw()
        '畫圓
        Dim pen As New Pen(Color.Black)
        Dim gr As Graphics = frm.CreateGraphics()
        gr.DrawEllipse(pen, New Rectangle(x - radius, y - radius, radius, radius))
        pen.Dispose()
        gr.Dispose()
    End Sub
End Class

Public Class Form1
    Inherits System.Windows.Forms.Form
#Region " Windows Form Designer generated code "
#End Region
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
            System.EventArgs) Handles Button1.Click
        Dim sObj As IDraw
        sObj = Shape.CreateObject(Me, 100, 100)
        sObj.Draw()
    End Sub
End Class
```

此程式繪出一個圓形：



Step-4. 「檢驗」----- 分合自如(PnP)。

前面提過，Rogerson 在其 “Inside COM” 書裡說道：

“Interfaces protect the system from being crippled by change.”

（接口讓系統不會因某部份改變而造成癱瘓）

現在檢驗看看這個 VB.Net 程式是否能發揮這種美好效果？茲將上述程式裡的片段：

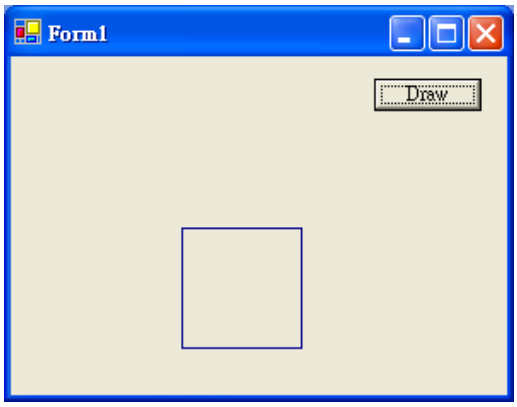
```
Public MustInherit Class Shape
        Implements IDraw
    Public Shared Function CreateObject(ByVal f As Form1, ByVal x0 As Integer,
                ByVal y0 As Integer) As IDraw
        Return New Circle(f, x0, y0)
    End Function
    Public MustOverride Sub Draw() Implements IDraw.Draw
End Class
```

將之更改爲：

```
Public MustInherit Class Shape
        Implements IDraw
```

```
        Public Shared Function CreateObject(ByVal f As Form1, ByVal x0 As Integer,
                                            ByVal y0 As Integer) As IDraw
            Return New picSquare(f, x0, y0)
        End Function
        Public MustOverride Sub Draw() Implements IDraw.Draw
End Class
```

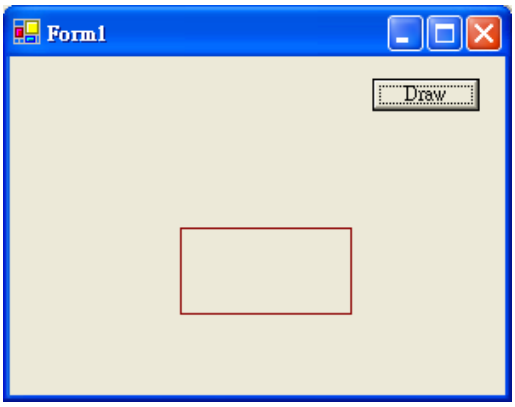
就繪出正方形：



只改變了一個指令，其他部分都沒有更動，的確達到 Rogerson 所說的效果了。再將上述程式片斷更改爲：

```
Public MustInherit Class Shape
        Implements IDraw
    Public Shared Function CreateObject(ByVal f As Form1, ByVal x0 As Integer,
                                        ByVal y0 As Integer) As IDraw
        Return New picRectangle(f, x0, y0)
    End Function
    Public MustOverride Sub Draw() Implements IDraw.Draw
End Class
```

就繪出長方形：



您可以把上述的 Form1 類別想像爲福特(Ford)汽車的車體，而 picCircle、picSquare 和 picRectangle 是三種不同牌子的輪胎，於是我們已經得到美好的滋味了：隨時輕易地將輪胎 PnP 到車體上，達到「一拍集合、分合自如」之多形性境界。再拿 Rogerson 的話：

> "The amazing result of polymorphism is that entire applications can be reused."（多形性的驚人效果就是：整個應用程式將可因而不斷重複使用了。）

請您特別留意，Rogerson 所說的是：「整體應用程式」(即車體)的重用；而不是小物件(即輪胎)的重用。許多人誤解爲接口設計是在追求物件的重用，其實是不對的。想一想，當您的檯燈燈管壞了，把壞燈管拔掉，換上新燈管，結果是：Reuse 了整個檯燈，而不是 Reuse 燈管。在想一想，一部 Benz 轎車輪胎壞掉了，把壞輪胎抽換掉，整部汽車恢復完好，所以 Reuse「整部汽車」了。

當然，輪胎也有其重用之價值，只是其價值不高；反而重用整部汽車的價值非常高。這是 Rogerson 所說："entire applications can be reused."（整個應用程式將可因而不斷重複使用了）的真諦。爲了讓您更深刻體會這個真諦，茲舉 Steven Vogel 在他的書---- 貓掌與彈弓 裡所說的[Vog99]：

> 「對人類來說，....系統功能要達到整齊劃一，通常得憑藉建造過程的持續一貫，零件(物件)要能夠彼此互換也需要高度的一貫性。自然界確正好相反，不僅對於零件互相交換毫無興趣，甚至還積極反對它。(例如免疫系統排斥心臟移植)。」

所以，接口設計之意義，並非要不斷 Reuse 它來獲利；而是追求能低成本換掉壞元件，然後裝上新元件；就等於 Reuse 整個系統裡的其它好物件，以及該新物件。所以：

- 輪胎(物件)是要換掉的，不要想去 Reuse 輪胎，其價值很低。
- 換掉壞輪胎，等於 Renew 一部汽車，等於 Reuse 整部汽車的其它好物件，價值極大。
- 換掉壞輪胎，而裝上新輪胎，等於使用(Use or Reuse)未來所有潛在可用的物件。

積極換掉舊物件，等於積極 Reuse 未來潛在的新元件。Reuse 既有的元件是有意義，但是價值不高，若因之而必須調整整個系統的其他好物件(即牽一髮動全身)，就得不償失了！最後，請您看看世界知名軟體專家 Cheesman & Daniels 的其在 2001 年的書 ---- ***UML Components****: A Simple Process for Specifying Component_Based Software* 裡所說的：

> "This may surprise some people. Many think the primary objective of components is reuse. They want to design something once and use it over and over again in different contexts, thereby realizing large productivity gains, taking advantage of best-in-class solutions, the consequent improved quality, and so forth. These are admirable objectives, but **the main driver today is that things keep changing**, and often ---- as with business-to-business electronic commerce ---- there is no longer any hope that centralized control can not be exerted.

In such an environment one of **the primary objectives of a component is that it must be easily replaceable** …. ”

(這可能會讓您很驚訝！許多人認爲元件的主要目標是 reuse 元件。其希望只要設計一次元件，就能一再地在不同場合裡反覆使用它，因而提高人員產能，運用最好的解決方案，然後改善品質等等。但是，今天問題的根源在於一切事情都快速變遷 ---- 尤其是跨企業的電子商務 ---- 中央集權機制已無用武之地。在此環境裡，軟體元件設計的首要目標就是：能輕易把它換掉。)

能以極低的成本換掉壞元件，就能帶給軟體系統彈性、靈活及生命力，其元件的新陳代謝極爲順暢。元件(物件)就如同壁虎的尾巴，當壁虎的尾巴被貓咬住時，會立即斷尾逃生。壁虎迅速乾淨例落地丟棄舊物件，Reuse 沒有被咬住的身體，再生出新尾巴(物件)，恢復(Renew)成一隻活生生的完整壁虎。如果壁虎不能輕易棄尾(拋棄物件)逃生，這個已經不能正常運作的尾巴(壞物件)就會拖累整隻壁虎。

(請繼續看下一節)

本文摘自　高煥堂 所著的
“VB.Net: 大型系統整合 DIY” 一書
也請您參考
“Java: 系統整合大作戰”一書

Part 3 of 3

5.3 心法 2：序中有亂

前面說過，科學家(如分析師)從亂中找序，而建築師(或設計師、藝術家)則規劃序中有亂。無論是「亂中有序」或「序中有亂」，兩者都要呈現序(Order)，並包容亂(Change)，只是手段不同而已。兩種手段都能帶來巨大經濟價值，精通這兩種手藝，是介面設計和系統整合之成功關鍵。序中有亂的「亂」變化、成長與繁榮之意，猶如會在法治社會中享有高度的自由一般。「序」支撐「變化」(亂)，又不壓抑變化。序容許亂，又不能失序。設計者視序爲虛，視變化爲實，於是虛實相依。如老子道德經(虛用篇) 說到：

「虛而不屈，動而愈出。」
(虛的 Facade 元件內並非一片死寂，
而是散發出無盡繁榮的潛能，
發揮巨大的動能。)

道德經(虛用篇) 老子又說：

「鑿戶牖以為室，當其無，有室之用。
故有之以為利，無之以為用。」
(建造實體房屋時，必須留有虛的空間----
如門窗和室內空間，房屋才能用。所以，
有虛的空間，實體才有用，
使用實體的人才能獲利。)

如果房屋沒有空間，房屋就不能用，即使用它也無利可圖。軟體裡如果沒有虛的空間，軟體就不能用，即使用它也無利可圖。此虛的空間是多用途的，能包容各種變化 ---- 容許虛空間內的實體系統之變化(PnP)。對應到前面所舉的貨櫃之例時候，可發現：

1. 貨櫃內部就是虛的空間，可容納複雜的物品，而且能隨時清空，擺入新物品，這是物品與貨櫃之間的 PnP 效果。
2. 複雜物品被包容於貨櫃中，貨櫃呈現出簡單之介面(序)，而且能隨意跟其他貨櫃組合，這是貨櫃與貨櫃之間的 PnP 效果。
3. 各種運輸工具，凡是能接受貨櫃介面者，如貨輪、拖車等，皆能與貨櫃一拍即合，這是貨櫃與運輸工具之間的 PnP 效果。

如上述，在運輸業裡，貨櫃帶來革命性風潮。在電腦軟體產業，微軟的 Windows 是一種軟體貨櫃，發揮了巨大經濟效益，一樣帶來革命性風潮。只是它蠻昂貴的，只有像微軟之大公司才能獲益。於是，我們必須尋找通俗的軟體貨櫃，我們才能獲益。踏破鐵鞋無覓處，得來全不費工夫，在 Gamma 的 *Design Patterns* 一書裡，有一個 Façade 樣式(Pattern)，它是個絕佳的軟體貨櫃，平易近人、自我重複、無限成長。於是，我們不只能開發模組，而且可以設計 Facade 貨櫃，如 Windows 般裝進天下所有的軟體模組或物件，並無限複製，獲利無窮，不亦樂乎?

5.3.1 從 Windows 體會「序中有亂」之意義

貨櫃的威力是來自於其簡單一致的介面，簡化了運輸業的工具，運輸業就很樂意將之拱起來，蔚爲風潮，然後要求其它產業的產品(如汽車)，乖乖地塞入貨櫃裡。類似地，Windows 的威力來自於它帶給硬體業一致的介面，硬體業就很樂意將之拱起來，蔚爲風潮，然後要求各式各

樣的應用軟體系統(如 ERP、Workflow 等)，乖乖地塞入 Windows 裡。由於 Windows 發揮了「序中有亂」之巨大經濟效益，而締造了世界首富----比爾.蓋茲(Bill Gates)。微軟不僅在 Windows 上發揮「序中有亂」的巨大能量，在近年來的 XML 潮流上亦是如此。

從貨櫃、Windows 與 XML 等可體會出，舉凡能包容變化之體系，都有三項共同之特性：

特性 1：包容複雜

---將複雜包容成爲單一元素，呈現一致介面。

特性 2：容易組合

---簡單的立體、樹狀或網狀組合，呈現出美好的序。

特性 3：自我重複

---整體和諧之序，是由小單位的序自我重複而形成的。

因爲單一元素，簡單組合，具有極大的包容力和彈性，所以能組合出無限花樣。於是，就整合的角度而言，貨櫃、Windows、XML 及 Façade 皆有曲同工之妙。

貨櫃：

容納複雜，一致介面，
簡單組合，無限複製，
裝進天下所能裝之物。

Windows：

容納複雜，一致介面，
簡單組合，無限複製，
裝進天下所有的軟體。

XML：

容納複雜，一致介面，
簡單組合，無限複製，
表示天下所有的文件。

Facade：

容納複雜，一致介面，
簡單組合，無限複製，
裝進天下所有的模組。

當我們從整合觀點來看時，貨櫃、Windows、XML 和 Façade 樣式的特質真是神似極了。一樣發揮「序中有亂」的特質，帶來一樣的巨大潛能，蔚爲風潮。它們把變化封裝起來，凸顯簡單秩序，人們擁有確定感，並藉由序來掌握變化，人們比較不會害怕變化。因爲秩序簡單而令人著迷，因爲變化無窮而令人驚嘆它的無限潛能。這就是貨櫃、Windows 和 XML 具有革命性威力的來源---- 和諧而充滿能量。

5.3.2 從藝術創作體會「容納複雜」之意義

在文學及藝術創作上，大文豪但丁在其著名的創作---- 「樂園」(Paradiso)裡，一位旅途的詩人，來到一個充滿複雜而多樣變化的國度，然而在此刻，這所有的一切都被包容於『一朵簡單的火焰』之中。這種序中有亂的融合與和諧，是藝術浩瀚力量的源頭。所以 Willian Boast 也說到[Boast2000]：

「所有偉大的藝術作品，都是和諧而充滿能量的。」

簡單的序帶來和諧，複雜的變化帶來能量。藝術作品巧妙地融合簡單性和複雜性。著名畫家 劉墉 先生在研究分析黃君璧國畫大師的畫風之後，他觀察到大師創作的秘密，他說[劉 2002]：

「一向強調筆墨要乾淨的黃君璧老師，在分析之後，可能連他本人都不敢相信，他厚重的色彩，其實得力於許多『混雜的筆墨』，那複雜豐富了他的畫面。」

就如 John Briggs 所說[Brig2000]：

「任何一件令人滿意的藝術之作，像是繪畫，每一個微小部分，都能反映出整體的畫作。」

所以人類的藝術設計，與自然界的萬物設計是一致的：整體之序包容了微小組件的複雜變化，處處呈現出「序中有亂」的整合藝術。 再看看中國第一經書----易經的「易」是變化(Change)的意思。易經的乾掛代表上天之變化，坤掛代表大地的繁雜。序中有亂就是包容變化(Accommodate the change)，也就是俗話的『容易』。意味著，只要能設計出序來包容變化，則一切會順心容易。

所以，我們需要設計像貨櫃、主機板之類的東西，這種東西就稱爲「界面物件」，一方面呈現簡單的介面，一方面包容變化，發揮「序中有亂」的巨大經濟效益。善於規劃及運用介面物件，系統整合就很「容易」了，整合出來的系統會擁有整體、和諧、無限繁榮的特質。此外，還能創造革命性的風潮，帶來巨大的經濟效益。

5.3.3 從主機板體會「簡單組合」之意義

貨櫃的組合方式再簡單不過了，只是單純的疊起來而已。XML 是單純的樹狀結構而已，也是很容易理解的。至於軟體 Windows 的組合就得

仔細觀察，才能看出其也是簡單不過了。

原來 Windows 的組合是透過主機板(MB: Motherboard or Main Board)相互連結而形成的。

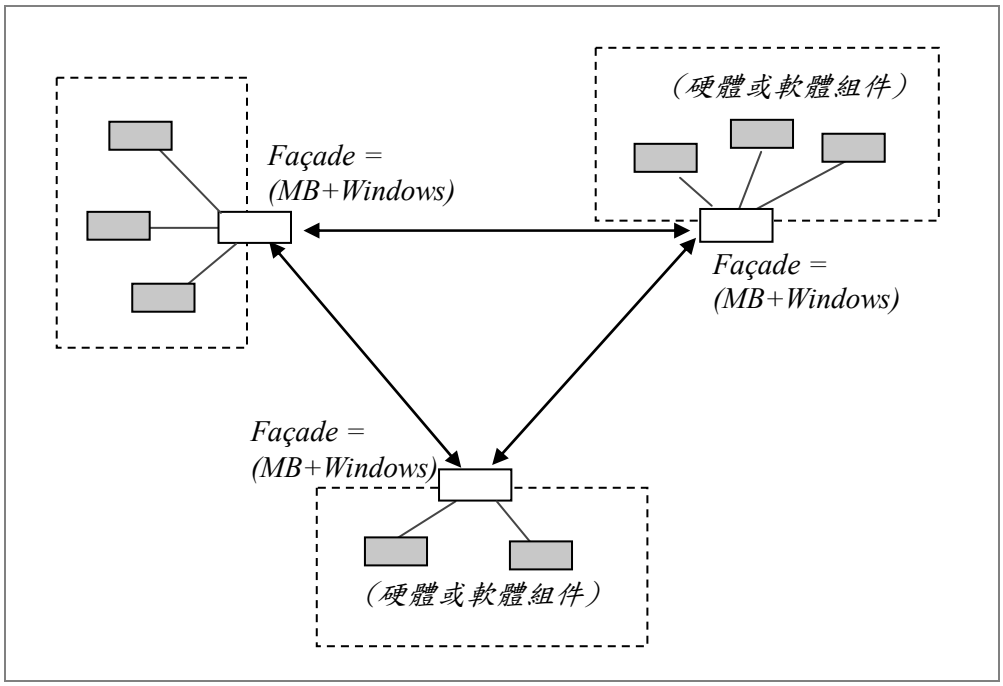


圖 53　主機板(MB)之簡單組合

主機板本身就是一個 IC 貨櫃，十足的序中有亂之作品；Windows 是軟體貨櫃，這兩個軟、硬貨櫃構成一個較大的貨櫃(Container)。然後透過 Internet 網路而形成爲簡單網狀組合(其實，網路也常是樹狀組合)。MB + Windows 成爲一個系統與其它系統接觸溝通之門面(Façade)。

5.3.4　從 DNA 體會「無限複製」之意義

樹葉是一個基本單元，經由簡單重複及組合而成爲一棵樹。當我們反向細觀葉子內部的複雜，也會看到葉子也是一個整體，也是由更小的單位、更簡單的序而形成的。一直小到 DNA 都是呈現出序中有亂的現象。例如，Steven Vogel 在貓掌與彈弓 書中寫道：

> 「一個 DNA 螺旋形的結構，可以由完全相同的小單位形成(好比一面牆是由相同結構的磚砌成)；再者，每一個小單位所插入的方式，與別的小單位一模一樣。只要你一旦明白一個 DNA 結構的裝置方式，你便可以掌握全部。」

所以我們可以從 DNA 可學習到「序」的建構途徑。茲以下圖表示出 DNA 螺旋形式：

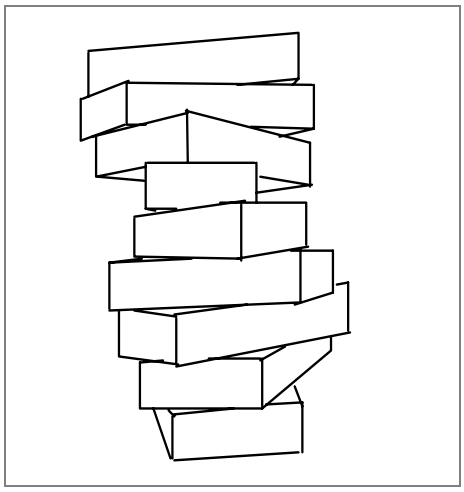


圖 54　DNA 小單位的螺旋形組合

小單位的內涵可簡單也可能複雜，其造形簡單，組合規則也簡單，不過組合出來的 DNA 螺旋結構的內涵卻極爲複雜，但結構外形仍是簡單的。生物從 DNA 開始就是依「序中有變化」的原則所規範的，因而創造

出生物的有機次序(Organic Order)，帶來和諧而多彩多姿的大自然。

6 DIY 演練：整合設計

茲舉 e-Taiwan 計畫裡的跨企業『訂單融資』系統整合爲例，說明如何運用「序中有亂」心法，擘劃訂單融資整合系統之軟體貨櫃(Façade)。

訂單融資的情境是：採購廠(如大同公司)向供應商(如大銘精密公司)發出購買馬達的訂單，供應商就立即攜帶訂單向銀行(如中國信託)辦理融資貸款，銀行向採購廠確認訂單之後就放款給供應商，供應商以該款項去購買材料來生產馬達，完成之後，運送到採購廠，驗收之後採購商才付款給銀行。如下圖：

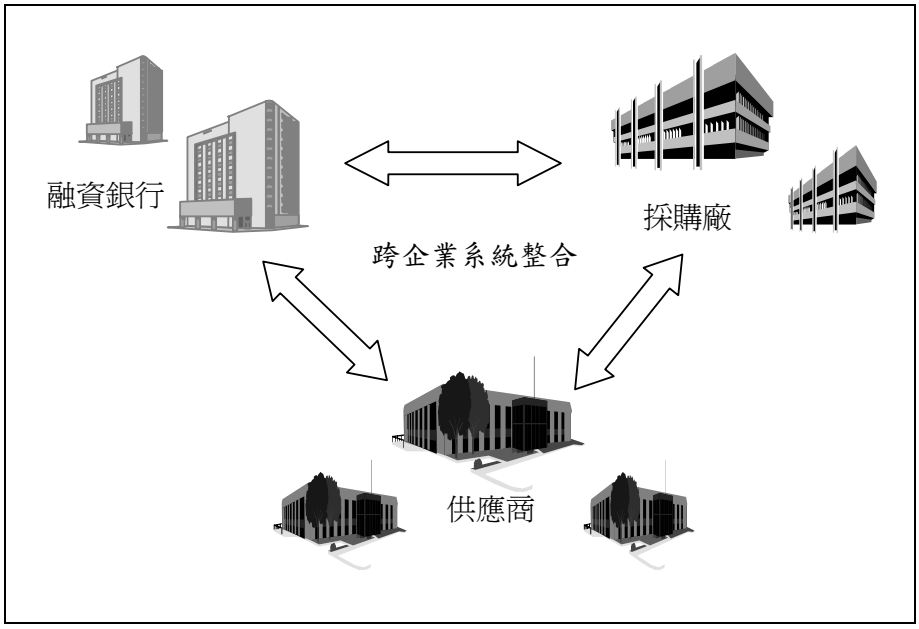


圖 55 訂單融資整合服務

這跨企業的訂單融資之整合業務流程如下圖：

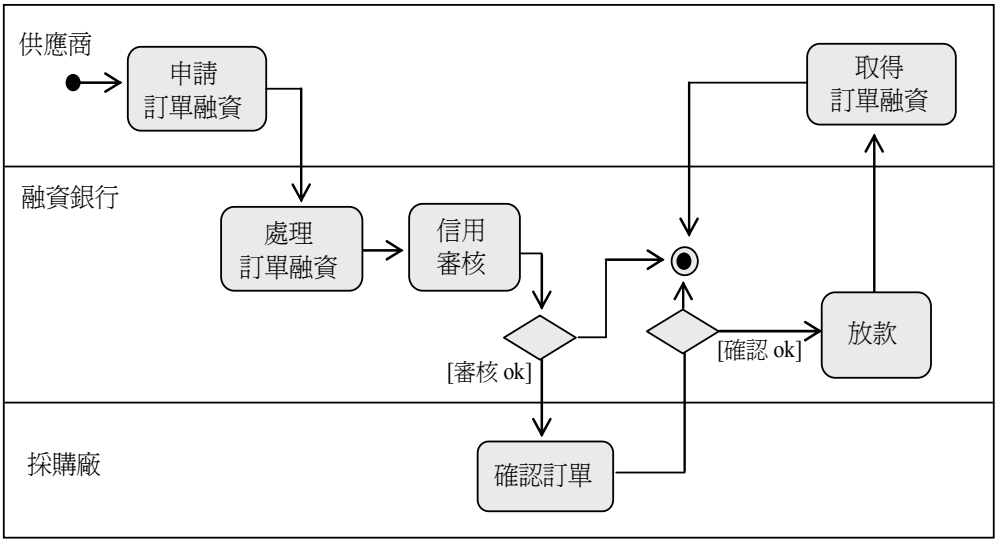


圖 56　訂單融資服務流程之例

想像在各個企業裡，各規劃一個 Façade 模組，成爲該企業的軟體主機板(Software MB)，如下圖：

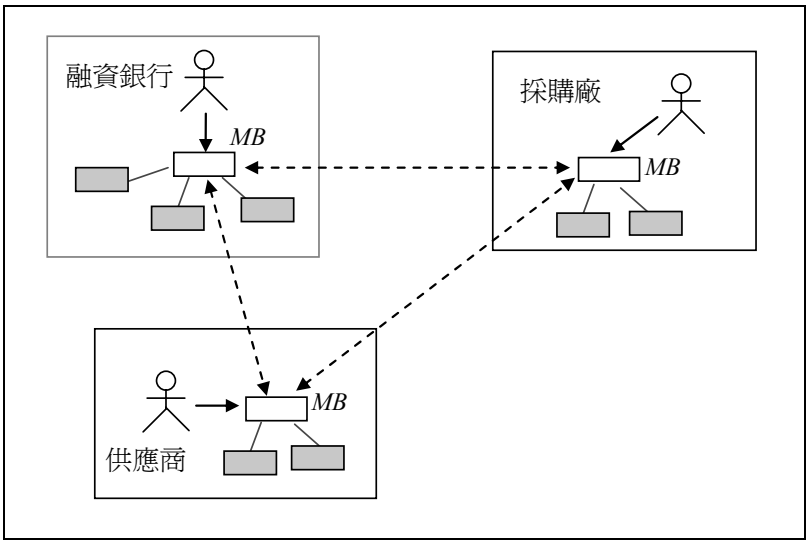


圖 57　MB-based 整合設計之一

MB 一方面整合既有系統之服務，一方面作爲對外溝通之窗口。這三個 MB 具有一致的介面能順利地互相溝通。也可以進一步規畫一個上層的 MB 來整合這三個小 MB，讓全世界各地的使用者，包括客戶、各企業的業務人員等，都使用上層 MB 的服務，如下圖：

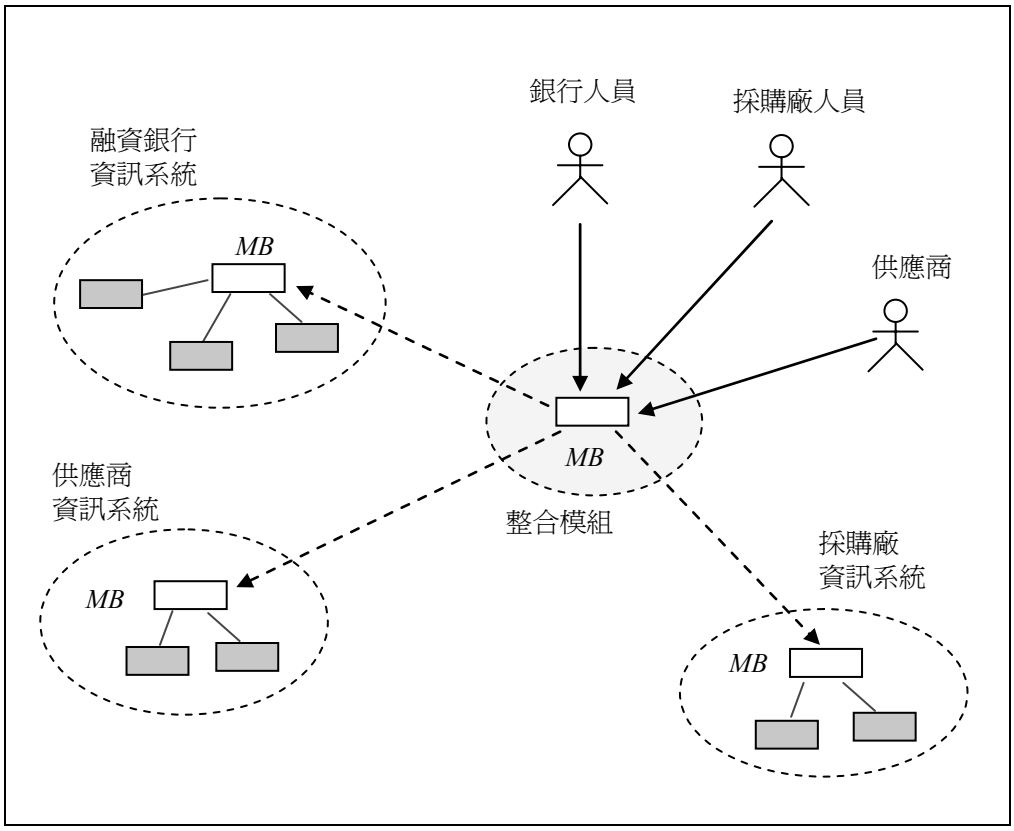


圖 58　MB-based 整合設計之二

MB-based 整合結構，提供分合自如的彈性。例如，上圖裡各實體系統是分散的，只是透過 MB 而在服務上整合起來而已。然而必要時，可以將各地的實體信息系統集中到一個地方，以便於管理和服務，如下圖：

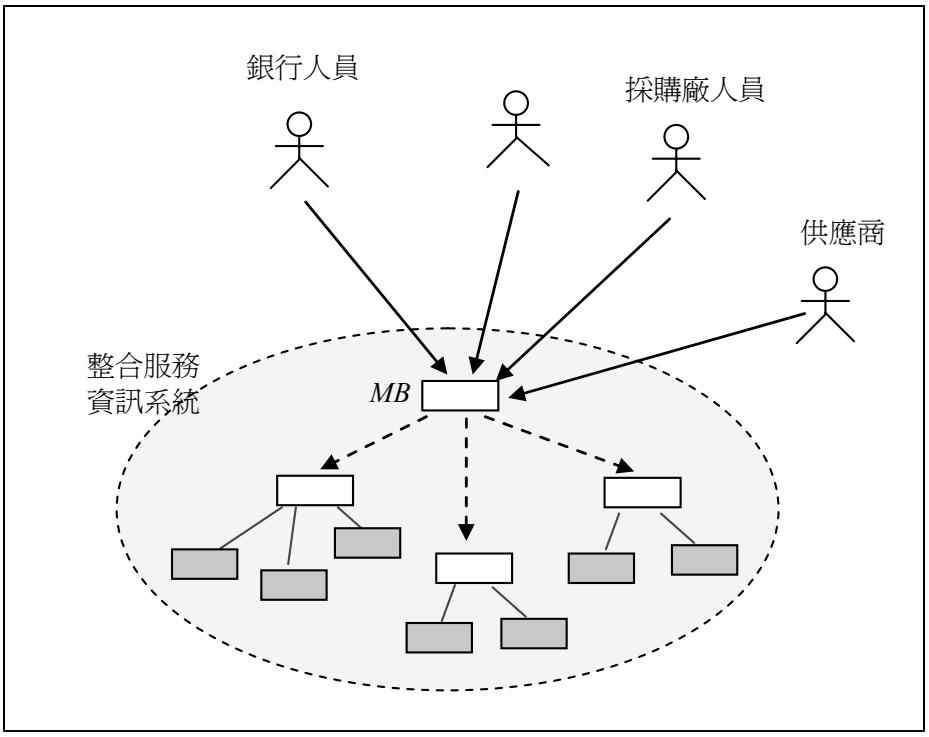


圖 59　MB-based 整合設計之三

MB-based 整合架構之形式是千變萬化的，想一想，電腦網路整合了全球各地的硬體主機板，而且形式無限變化。藉由硬體主機板的對比，的確能促進對軟體 MB 的體會，激發更多形式之整合架構設計。

系統整合不只追求一時的連結，更要追求持久的和諧與成長。爲了達成這個目標，系統的新陳代謝必須順暢，也就是，系統內之物件必須能隨時迅速汰舊換新、分合自如。此汰舊換新之效果通稱爲 PnP(Plug and Play)，即系統內之物件易於抽換與易於修改，簡而言之，就是：“Easy to change”。就如 Rogerson 在其 “Inside COM” 書裡提到：

"Interfaces protect the system from being crippled by change."
（介面讓系統不會因某部份改變而造成癱瘓）

爲了實現 PnP，我們必須進行介面設計，就如同汽車的輪胎能隨時抽換(PnP)，其依賴標準的輪盤(即介面)設計。也是因爲能隨時抽換輪胎(即分合自如)，所以汽車能持久保持整體的和諧。PnP(即分合自如)是目的，而介面是手段。介面設計之優劣關係到 PnP 之效果。唯有優越的 PnP 效果才能確保持久的整體和諧之序，然後序中容納複雜與變化，締造氣象萬千、無盡繁榮。

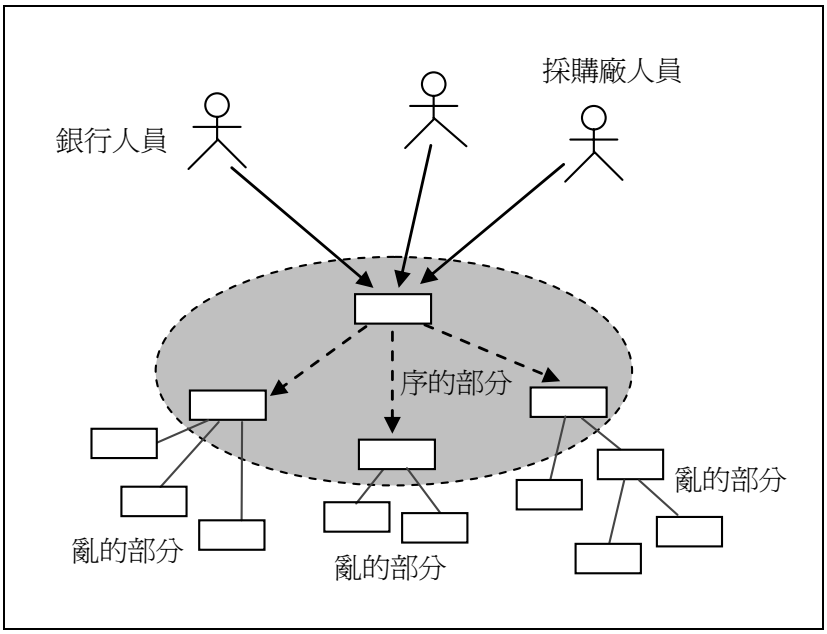


圖 60　序支持亂、變化萬千

從上述之設計實例中，相信您已經看到：

1. 以物件、類別來「分門別類」，分析出「亂中之序」 ---- 共同之介

面(行為)，建立系統整合之基礎。

2. 以 Façade 樣式 + Web Service 來規劃「整體之序(介面)」，支持「序中之亂」，讓系統彈性成長、無限繁榮。
3. 基於不斷自我重複之序，讓 VB.Net 的程式碼也非常簡潔而井然有序。

本文摘自　高煥堂 所著的
"VB.Net: 大型系統整合 DIY"　一書
也請您參考
"Java: 系統整合大作戰"一書

參考資料

[**張97**] 張榮發, 張榮發回憶錄, 遠流出版, 1997.

[**劉2002**] 劉墉, 創造超越的人生, 2002.

[**Bank2002**] David Bank, *比爾蓋茲教你透視微軟*, 盛鴻時 譯, 時報出版, 2002, ISBN: 957-13-3690-4.

[**Boast2000**] William Boast, *擅變*, 蔡依玲 譯, 方智出版, 2000.

[**Brig2000**] John Briggs & David Peat, 亂中求序, (姜靜繪 譯), 先決出版, 2000.

[**Gam95**] Erich Gamma, etc., *Design Patterns:Elements of Reusable Object-Oriented Softawre, Addison-Wesley, 1995.*

[**Roger97**] Rogerson D., *Inside COM*, Microsoft Press, 1997.

[**San98**] Irene Sanders, 致勝思維, (張美智 譯), 金錢文化, 1998.

------ END ------