東　　海　　大　　學

資訊科學研究所碩士學位論文

公用物件請求仲介者架構上的工作流程系統

CORBA-Based Workflow System

研 究 生：林大卿

指導教授：羅文聰

中華民國八十九年六月

摘　要

本論文將提供一個發展以 CORBA (Common Object Request Broker Architecture)為基礎的工作流程系統 (Workflow System)建議架構，和目前現有的工作流程系統不同點是本系統遵照 WfMC (Workflow Management Coalition)及 CORBA Workflow Facility 規格，並配合 CORBA Event Service 之技術所建構，將會是一個具擴充性及以事件驅動為主的系統；此外，本系統的撰寫全數採用物件導向的方式，在系統本身的維護及擴充上具有較佳的能力。

目前的工作流程系統僅強調文件流程，本論文預計開發之系統具有觸發事件呼叫應用程式的能力，使得本系統除了能做文件的簽核之外，尚能提供不同應用程式執行自動化的能力。

有鑑於傳統工作流程系統的主控者是工作流程核心系統 (Workflow Engine)，所有和流程相關的作業都由核心系統主控，如果有大量的工作流程產生時，該核心系統可能就成為執行上的瓶頸，為了解決這個問題，本論文在客戶端建立了 Message Pre-Processor 的模組，期能透過分散作業的方式減少核心系統的負擔，進而增進執行效率。

誌　謝

非常感謝指導教授羅文聰老師在求學的這段時間內，在學業上對我的指導及觀念上對我的啟發，使我能順利完成學業。還要感謝文大資科博士班的蕭存喻學長給我的建議及協助。

接著要感謝學弟邱盈仁、馬若傑在實作上對我的建議及協助，並在日常事務上的幫忙。

更要感激我最愛的家人，在這段時間所給予的關愛與鼓勵，使我能專心地求學。對於所有關心我的朋友，也在此致上我最誠摯的謝意，並衷心地祝福你們。

目　錄

圖 示 目 錄

第一章 導論

1.1 研究動機

網路的發達，加快了訊息傳遞的速度，也改變了整個商業經營的模式，現今的企業，必需要快速回應外在環境的需求，這些需求包含了企業與顧客以及企業與其他企業之間的溝通，為了達到這種要求，我們需要利用電腦以及網路建構一個企業內及跨企業的數位神經網路，以使企業具有足夠的能力來面對競爭者的挑戰並提高整個企業的競爭力。

為了達到這個理想，除了具備有相關的電腦及網路基礎架構之外，還必需要有一些適用的軟體將企業內部的系統加以連接，並將企業中進行的各種程序加以自動化，因此有必要開發一個適用於企業內部各種系統的工作流程系統 (Workflow System)。

現今企業內部，多已具有各種相關的電腦軟體以應付平日企業內部的各種需求，一般的開發狀況大多是專為某個部門設計相關的專屬系統，這種系統的開發方式所完成的軟體系統會適用於各個部門內部使用，但是當有需要做部門間資料傳送溝通的話，就不見得適用，因各個軟體系統是獨立開發的，所使用的資料格式、開發語言、作業平台都可能有所不同，使得當各部門間要做訊息傳遞的時候，免不了要用到人工處理的方式，當有人介入時，可能會發生輸入錯誤的狀況，也會因此而使得處理的速度變慢，因此我們在開發整合系統時，採用 OMG (Object Management Group)所提出來的 CORBA (Common Object Request Broker Architecture)架構，以期透過這個架構的發展語言、作業平台獨立性的特點，使得開發時較不會遇到這方面的問題。

基於以上的考量，本論文將建議一個發展以 CORBA 為基礎設計的分散式工作流程系統 (Distributed Workflow System)的架構，簡稱 CBWS。

本論文所發展的系統和現有的各種工作流程系統的最大不同點有二：第一是具有可呼叫各個應用程式的能力，第二點則是結合 WfMC (Workflow Management Coalition)所提出之標準架構、CORBA Workflow Facility 所建議之架構、及 CORBA Event Service 的特性完成本系統。

由於現今的工作流程系統大都是做為組織內部文件的傳送，只是提供一個組織內的

文件流程自動化，本系統考量到組織內部的系統間也有執行先後順序的流程概念，故本系統將具有能呼叫應用程式 (Invoked Application)的功能，使本工作流程系統除了能使組織內部文件傳送自動化外，還能使應用程式執行流程自動化，這是本系統與其他現有系統不同的特點之一。

另外，目前大部份的工作流程系統都是按照各組織、企業的需求發展的，大都沒有依據一定的標準，這在系統需要擴充或是需要和其他的流程系統交換資料時會造成溝通上的問題，而本系統在建構時就已參考了 WfMC 及 CORBA Workflow Facility 所制訂的相關架構，並依其標準撰寫，故本系統會是一個物件導向及標準化的工作流程系統，這也是和目前大部份的工作流程系統不同的特點。

本系統的底層訊息傳遞的部份，採用 CORBA Event Service 實作，由於使用這個機制，可以降低伺服器端及客戶端的程式相關性，減低系統開發的複雜度。而且本機制可以使系統和系統間的互動改為事件驅動 (Event-driven)的方式，只有在相關事件發生的時候，系統才需要進行處理，一方面減少系統間無用的通訊，另一方面也會加快整體反應時間。

本系統發展完成後，將會具有一個可連結企業內各個不同系統的分散式工作流程系統，本系統將可提高企業的反應能力，進而增加企業的競爭力，讓導入的企業可以跟的上時代的腳步，成功的成為電子化企業。

1.2 相關研究

以下將介紹和本論文有關的國內外相關研究，相關的部份有：CORBA 架構、CORBA 所提供的 COS Event Service、工作流程的相關組織 (WfMC: Workflow Management Coalition)及其所建議的規格，最後介紹 CORBA 的 Workflow Facility。

1.2.1 CORBA 架構簡介

CORBA[7]是 OMG 所制定的，OMG 是一個由會員贊助而成立的非營利組織，其目的在推廣物件導向 (Object-Oriented)的觀念及使用並致力於加強軟體的可攜性 (portability)，再利用性 (reusability)，以及互通性 (interoperability)。該組織會員包括了廠商、學術單位及用戶。目前全球大約有七百家重要廠商為其會員。CORBA 為 OMG 在一九九一年十二月提出之物件導向分散式工作環境規格，一九九三年十二月及一九九

四年九月此規格多次被修訂；一九九五年七月之 CORBA 2.0 版，互通性及 C++ 對應(mapping)之標準，均於此版之標準中更明確地被製訂，目前更進一步，正在制定 CORBA 3.0 的規格，期能更適合現代分散式運算的環境。

為了提供分散式環境，OMG 制定了一個管理物件架構(OMA:Object Management Architecture)，也就是 CORBA 的主要架構，該架構如圖 1 所示：

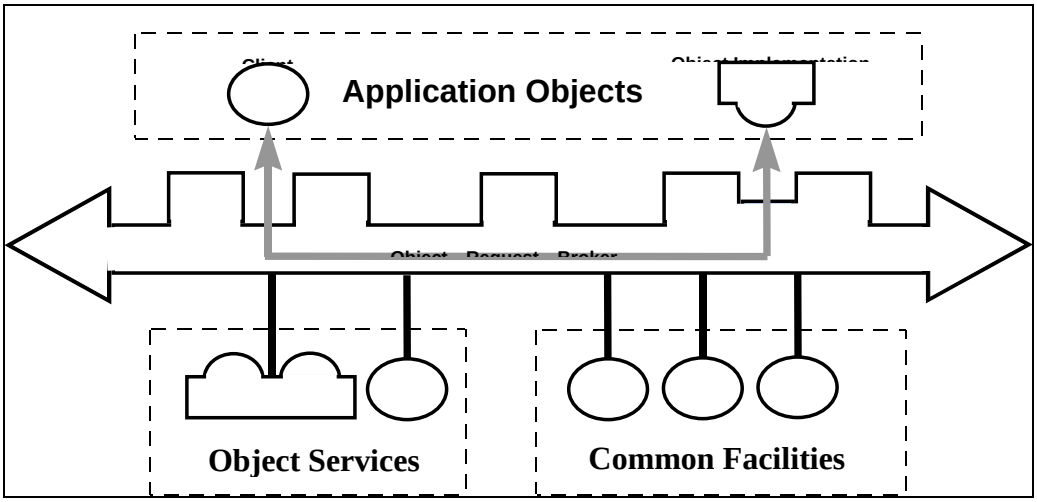

圖 1 物件管理架構(OMA)

物件管理架構，由四個主要的元件組成，說明如下：

(1). ORB：ORB 是一個中介軟體(middle-ware)，它可使客戶端的程式能夠透明地(transparent)呼叫位於伺服端的物件所提供的方法(method)，而不管這個物件是位於遠方的物件，或是同一部機器的物件。ORB 接收請求，並且負責找到一個實作這個請求的物件、傳給實作參數、呼叫指定的方法、和傳回結果等工作，客戶端可以不知道這個物件的位置、它所使用的程式語言、它的作業系統、或是任何和這個物件的界面 (interface) 無關的部份。這樣的作法，使得 ORB 提供在異質性 (heterogeneous) 分散式環境中應用程式的互通性，並且可以完美的連接多個物件系統。

(2). Object Services：除了 CORBA ORB 核心之規格外，CORBA Services 亦於一九九五年三月之 COSS (Common Object Service Specification) 規格中做了詳細之規訂。根據 OMG 的定義，服務為支援物件使用及實作之功能 (function)。為了在多部機器間提供較完美的物件環境，必須有一個機制來經由物件的名稱存取物件、在遠端產生或是修改物件、隨時維護物件的狀態和其他一般性的服務。目前提供的服務有 Naming 、Concurrency control、Event、Persistent Object、Life Cycle、Externalization、Relationship、Property、Query 和 Transaction 等服務。

(3). Common Facilities：OMG 定義一些應用程式間共用的界面和統一的語意，使得符合物件管理架構的應用程式易於建立。Common Facilities 包含 Vertical Market Facility 和 Horizontal Common Facility 兩個部份，Horizontal Common Facility 制定一般應用程式常用到的功能，如使用者界面設施、系統管理設施等，目前此部份的規格尚在制定發展中。

(4). Application Objects：在物件管理架構最上層為 application objects，其與下層物件的主要差別是 application objects 與實際應用有關且無法讓其它應用程式共用的部份，因此這個部份是應用程式設計者針對應用程式需要而自行發展的物件。

在 CORBA 環境下發展程式的步驟如圖 2 所示，說明如下：

(1). 介面描述：使用 CORBA 的介面描述語言描述 Client 及 Server 間的溝通介面，即 Server 物件所提供的成員函式(member function)及彼此傳遞之參數的資料結構。

(2). 程式實作：實作要放在 Server 端的物件，及修改原循序版本程式，使其具有分散執行的能力。

(3). 介面描述檔編譯：使用 CORBA 的 idl 編譯器編譯介面描述檔，可產生三個檔：處理 Client 呼叫 Server 端函式的程式 (Client Stub)，定義一些常數及函式原型的標頭檔 (Header File)，以及 Server 端用來處理 client 呼叫的函式 (Implementation Skeleton)。

(4). 產生 Client 及 Server 端的程式：分別將 Client Stub 與 Client 的程式連結，將 Implementation Skeleton 與 Server 端實作物件的程式連結，便可產生 Client 與 Server 端的應用程式。

(5). 散佈 Server 程式：最後將 Server 端的程式，散佈至每個可用的 Server 上，做些註冊動作即可。

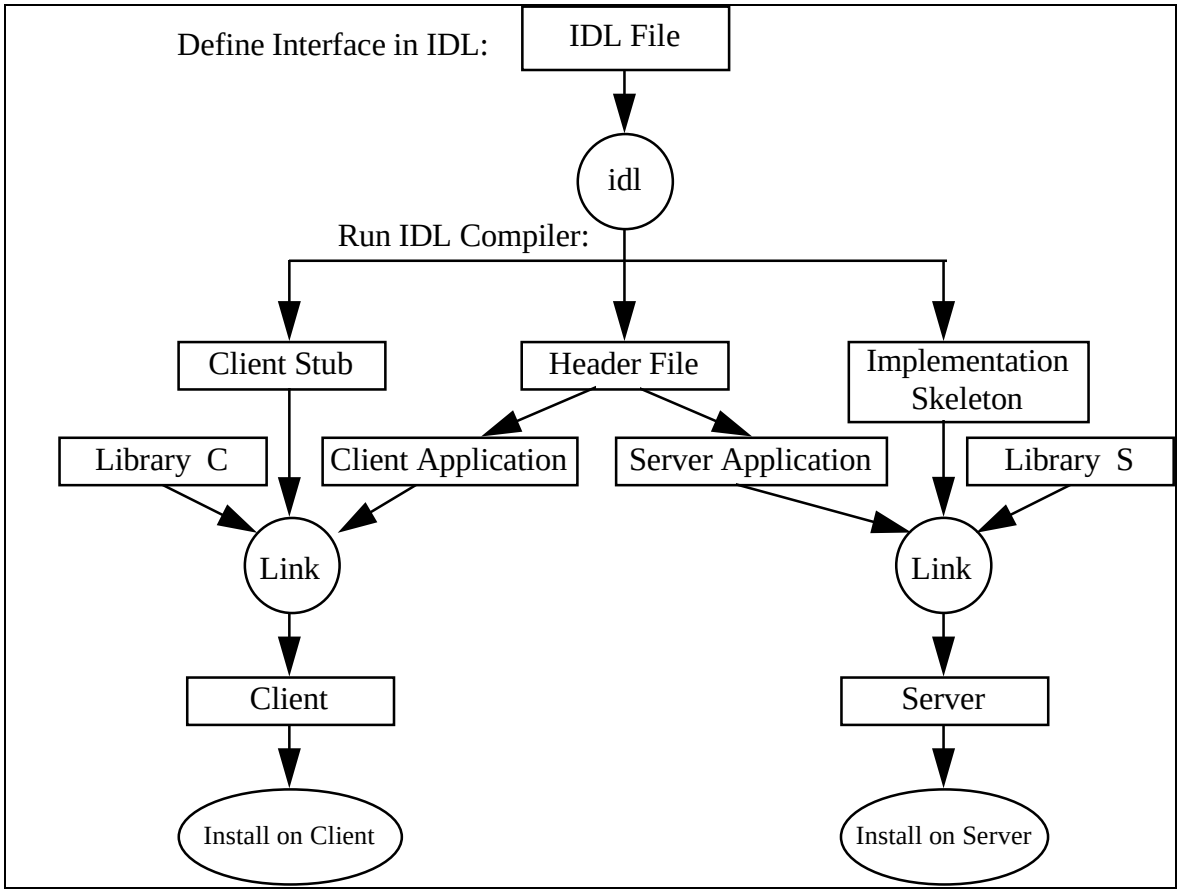

圖 2 CORBA 應用程式之設計流程

按照 CORBA 的架構發展程式具有以下的好處：

(1). CORBA 是專為跨平台處理所訂定的標準，故可在跨平台的環境中讓程式互相溝通。

(2). COBRA 使用 CDR（Common Data Representation）作為資料傳送的格式，可產生不同長度的訊息，在資料訊息格式的使用上較具彈性。

(3).由於 COBRA 具有物件導向(Object-Oriented)的觀念，對於訊息格式的擴充較為容易。

(4).資料表示方式較為高階。

(5). CORBA 提供許多共用服務，如 Naming Service、Persistent Object Service、Transaction Service、Other Services 等。

1.2.2 CORBA Event Service 簡介

Event Service 是 CORBA 所提供的共通服務之一[8]，它主要是針對物件與物件之間

的通訊所提供的服務。在 Event Service 中，有以下三種不同的角色：

(1). Suppliers：負責產生訊息資料，將訊息資料送給 Event Channels 處理，它信任 Event Channels 處理訊息的分配，所以 Suppliers 不用擔心訊息資料傳送時可能會發生的問題，以及該傳送給哪些接收者。

(2). Consumers：負責接收來自於 Event Channel 所傳過來的訊息。

(3). Event channels：介於 Suppliers 及 Consumers 之間，做為兩者間傳送資料的管道，它能成為多個 Suppliers 及多個 Consumers 之間的溝通管道，並提供非同步（asynchronous）的資料傳送功能。

在 Suppliers 和 Consumers 間透過 Event Channels 傳送資料的模式有兩種，分別是 Pull model 及 Push model 的方式，兩者的運作方式如(請參考圖 3 Push-Pull Model 表示圖)所示：

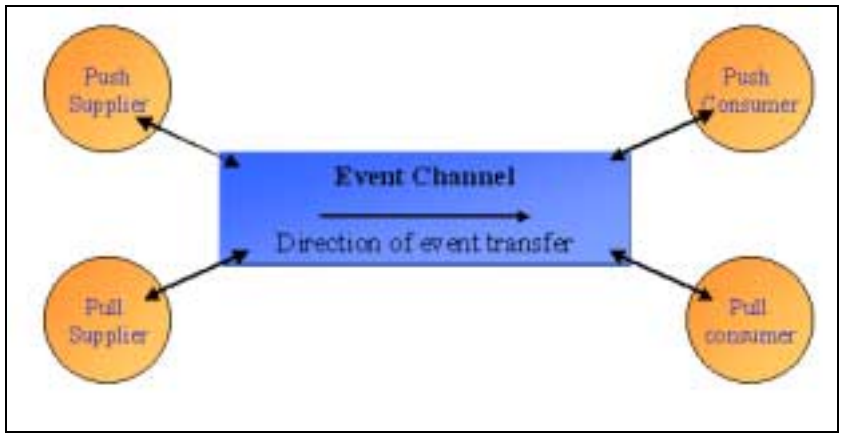

圖 3 Push-Pull Model 表示圖

(1). Pull model：由 Consumers 產生 requests，當 Suppliers 收到這個請求之後，就將 Consumers 要求的資料傳回。這種方式的主控權在 Consumers 端，即使 Suppliers 有新的資料可以提供，但只要 Consumers 沒有送出 request，Consumers 就沒有辦法拿到這些資料。

(2). Push model：Consumers 先註冊一些他有興趣的頻道，當 Suppliers 有資訊可以提供的時候，就透過 event channels 將這些資訊送交 Consumers。在這種方式下，只要 Consumers 先有做一個訂閱的動作，當 Suppliers 有新的資料可以提供時，就會自動送交給 Consumers。

在使用 Event Service 的 CORBA 環境中，我們能夠允許物件動態地對特定 Event

Channel 做註冊或是取消註冊的動作。在這種環境之下，ORB 所提供的 Supplier-Consumer (Push-style) 通訊模組，可以讓多個 Supplier 透過 Event Channel 以非同步的方式送訊息給多個 Consumer 物件。因此在這種 Supplier-Consumer (Push-style) 的通訊模組下，Supplier 可以動態的產生特定的訊息，將他丟給 Event Channel，而不用管後續的發送事宜，也不用管目前到底需要送給哪些 Consumer，Event Channel 會根據他所掌握的資訊將這些由 Supplier 來的 Event 自動送給有註冊這個 Event Channel 的 Consumers。，這種作法可以使得物件和物件間形成 loosely-coupled 的關係，我們可以因而免除大量繁複的溝通動作處理，並可免除掉傳統的 2-way method call 會有 one-to-one request blocking 的動作，發送者必需等到接收者回應才能繼續處理下去的問題。

1.2.3 WfMC (Workflow Management Coalition)建議規格

WfMC 是一個專門制定工作流程標準的組織，該組織提出一些在開發工作流程系統時希望大家能參考使用的規格書，以下介紹與本系統較有關的部份：

(1). 工作流程參考架構[2]

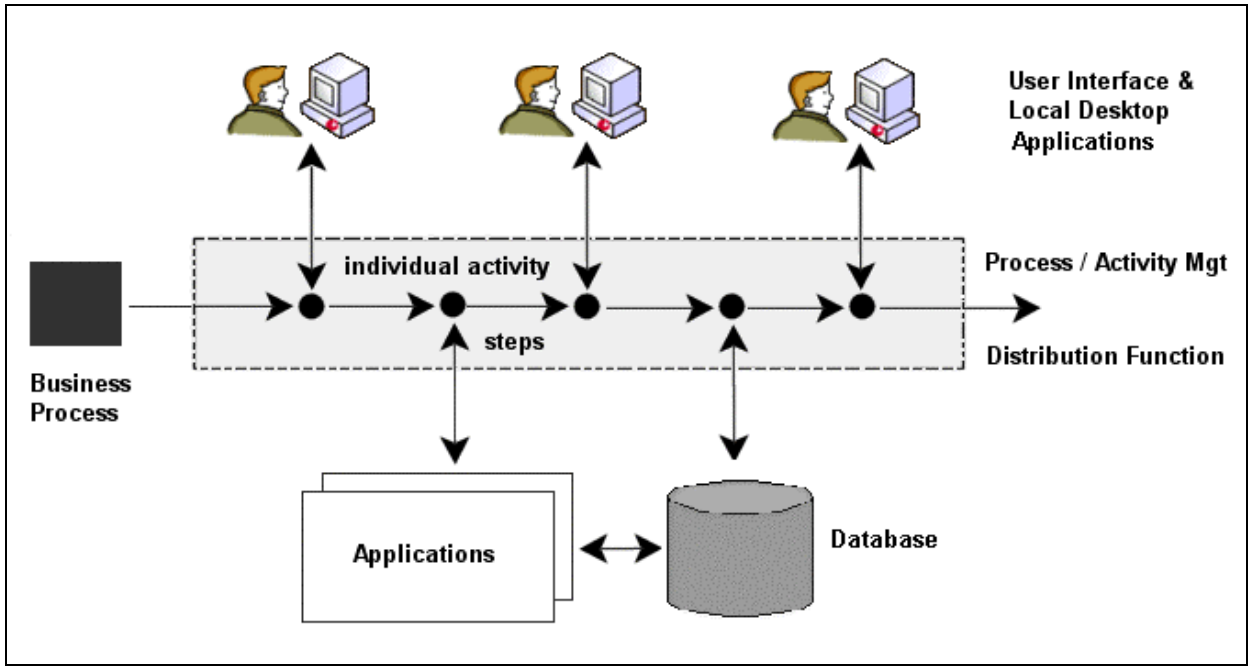

圖 4 流程簡介圖

圖 4 介紹的是 Process 與 Activity 的觀念；每個 Activity 是一個不可再分割的步驟，

這個步驟可能會和外部程式、資料庫或使用者進行互動，而一個 Process 則內含一個到數個的 Activities，Process 對應到企業內部實際的一連串有意義的行為，亦即是將企業實際運作的狀況及先後步驟對應成為電腦所能處理的程序。

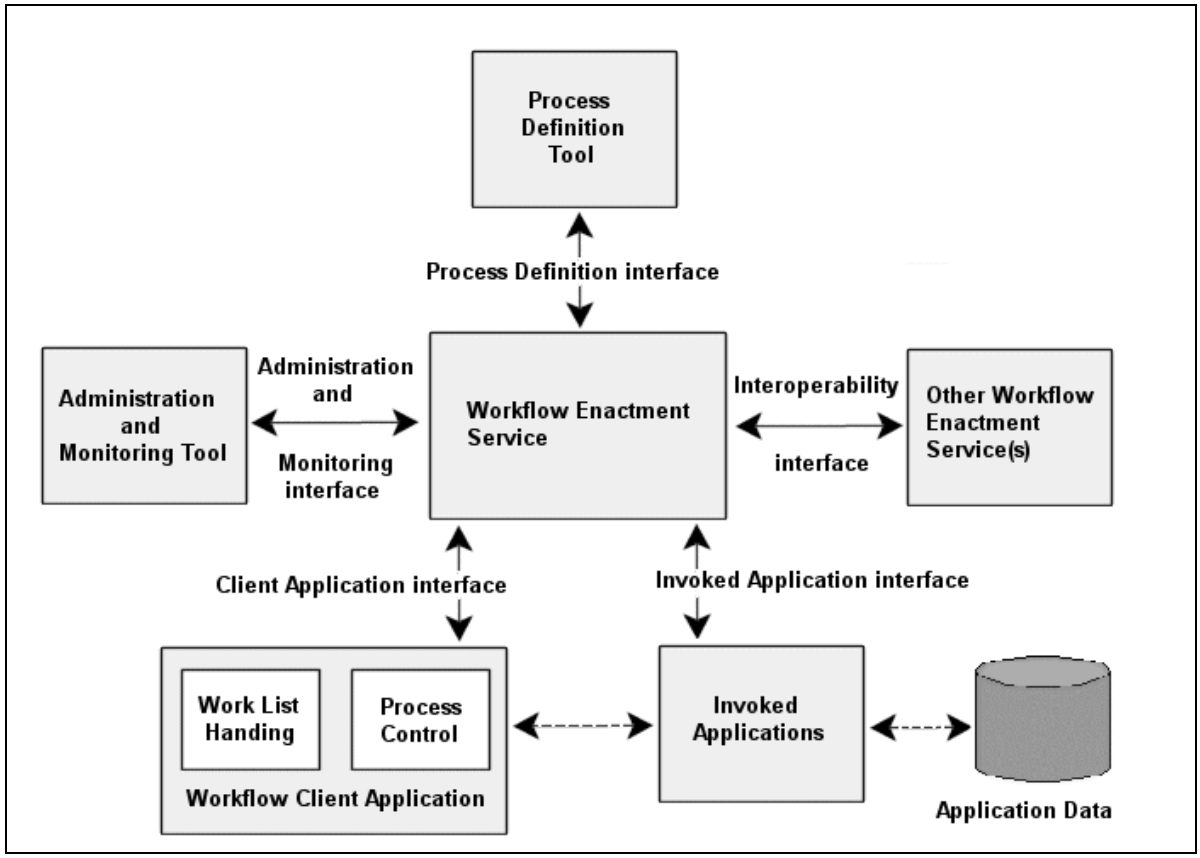

圖 5 WfMC 工作流程參考模型

圖 5 是 WfMC 提出在設計一個工作流程系統時應該提供的完整模組，模組共分為：

Workflow Enactment Service：為工作流程系統的核心，負責進行各模組間的溝通與控制。

Process Definition Tool：一個圖型介面的工具，用來將組織的工作流程儲存成電腦可處理的標準描述語言。

Administrator and Monitoring Tools：用來管理及監督各個工作流程的狀態。

Workflow Client Applications：可供使用者使用的工作流程客戶端程式。

Invoked Application：可供工作流程核心系統叫用以提供某些功能或服務的外部程式。

Other Workflow Enactment Service：其他不同供應商所提供的工作流程核心系統。

為了要讓工作流程核心系統能夠和各個模組溝通，因此 WfMC 將上述的五個模組訂出了五份建議規格書以供開發者依循，以便能確保各個不同開發者建構出來的工作流

程系統能彼此互相溝通。

(2). WPDL (Workflow Process Definition Language)

WPDL[2]是 WfMC 為了將實際工作流程轉化成電腦所能儲存處理的格式所訂出的標準語言，只要使用者在儲存流程相關資訊時以 WPDL 表示，便能讓各個不同開發者建構出來的工作流程系統取用，避免流程跨越不同的工作流程核心時，無法順利將資料傳送過去。

一個 WPDL 的內容約略如下所示：

```
MODEL
    < Model description >
    < Related participant list >
    < Related data list >
    < Workflow list >
END MODEL
```

宣告一個 MODEL 需要包含四部份的內容，模型的概要描述、流程相關的參與者、可能傳送的資料、以及實際流程的相關資訊，各部份的內容範例如下：

```
// Model description
WPDL VERSION  7.0 BETA
VENDER        Vendor:Product:Release
CREATED       1998-07-15
NAME          Business Example
DESCRIPTION   EXAMPLE
AUTHOR        WG/1B
STATUS        UNDER REVISION

// Related participant list
PARTICIPANT       'Tim_White'
    NAME          "Tim White"
    DESCRIPTION   "Mail Room Clerk"
    TYPE          HUMAN
END PARTICIPANT
```

// Related data list

DATA ‘DOCUMENT_TYPE’

TYPE STRING

NAME “DOCUMENT TYPE”

DEFAULT_STRING “SALES ORDER”

END PARTICIPANT

WORKFLOW

< Description >

< Activity List >

< Transition Information List >

END WORKFLOW

在描述 Workflow (也就是定義上所提的 Process)相關訊息的時候要加入較多的資訊，包含了這個 Workflow 的描述、該 Workflow 內含的所有步驟、以及資料的來源、目的及內容，一個 Workflow 的簡單範例如下：

// Description

CREATED 1998-07-15

NAME “In the Mail Room”

// Activity List

ACTIVITY ‘Finance_Process’

NAME “Finance Process”

IMPLEMENTATION WORKFLOW SYNCHR ‘At_the_finance_department’

PERFORMER “Finance Department”

END ACTIVITY

// Transition Information List

TRANSITION ‘t_1’

FROM ‘MailRoom’

TO ‘Sales_Older_Handling’

CONDITION ‘document_type’ = “Sales Older”

END TRANSITION

上述所列出的 WPDL 是目前的語法，而 WfMC 在 2000 年 5 月 8 日提出了以 XML 作為工作流程定義的標準規格[6]，將上述以 WPDL 描述的內容全部改以 WF-XML 這個語法表達之，其 DTD 內容定義如圖 6 所示，由於目前 XML 有成為文件標準交換格式的趨勢，故未來 WF-XML 必將成為新的標準。

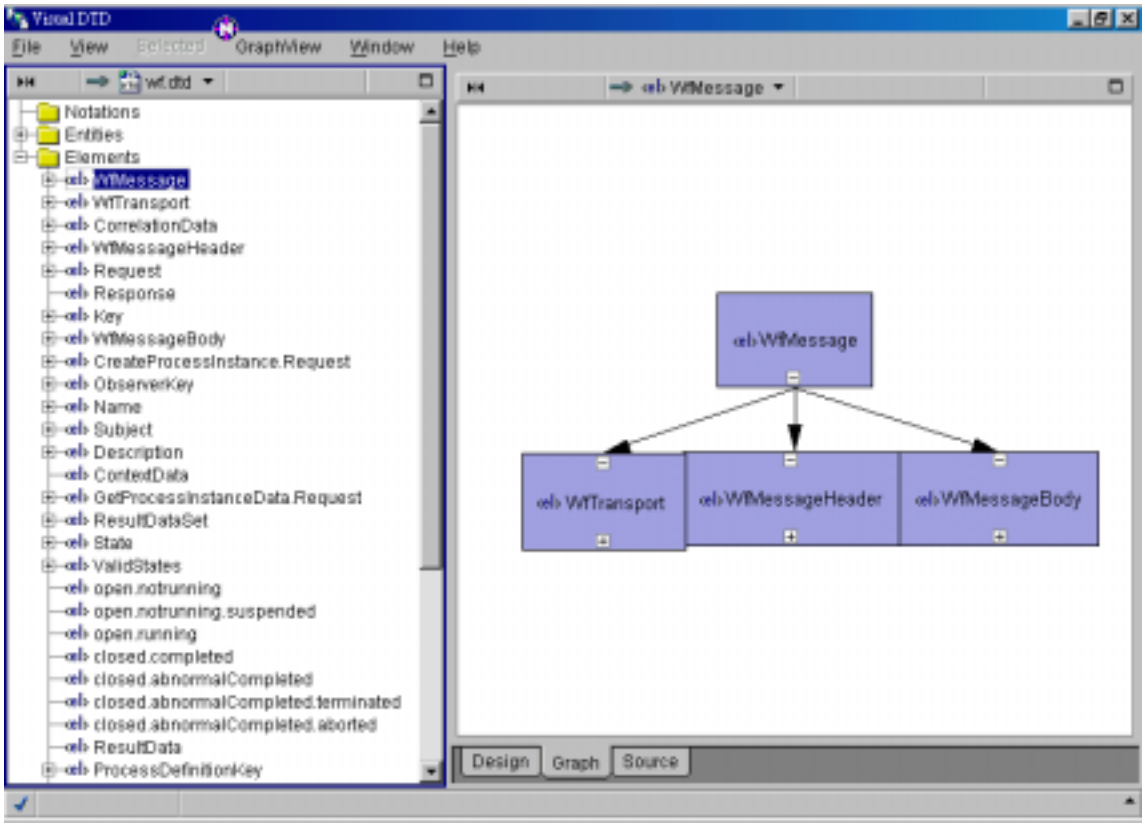

圖 6WF-XML 定義內容

各廠商對於 WfMC 所提出的各種標準介面的支援狀況，在 http://www.aiim.org/wfmc/standards/conformance.htm 有目前各廠商支援程度的列表。

1.2.4 CORBA Workflow Facility

OMG 制定了一些 Facility[9]，分別適用於特定的產業領域以及垂直整合應用上。

適用於特定產業領域的稱為 Horizontal Facility，包含了 CORBAmed、Telecommunication Domain、Financial Domain、Manufacturing Domain、E-Commerce Domain、以及 Life Science Research Domain 等。

適用於各種產業的稱為 Vertical Facility，包含了 Workflow、Meta Object、Business Object 等，這些都是在任何相關行業中都會使用到的功能。

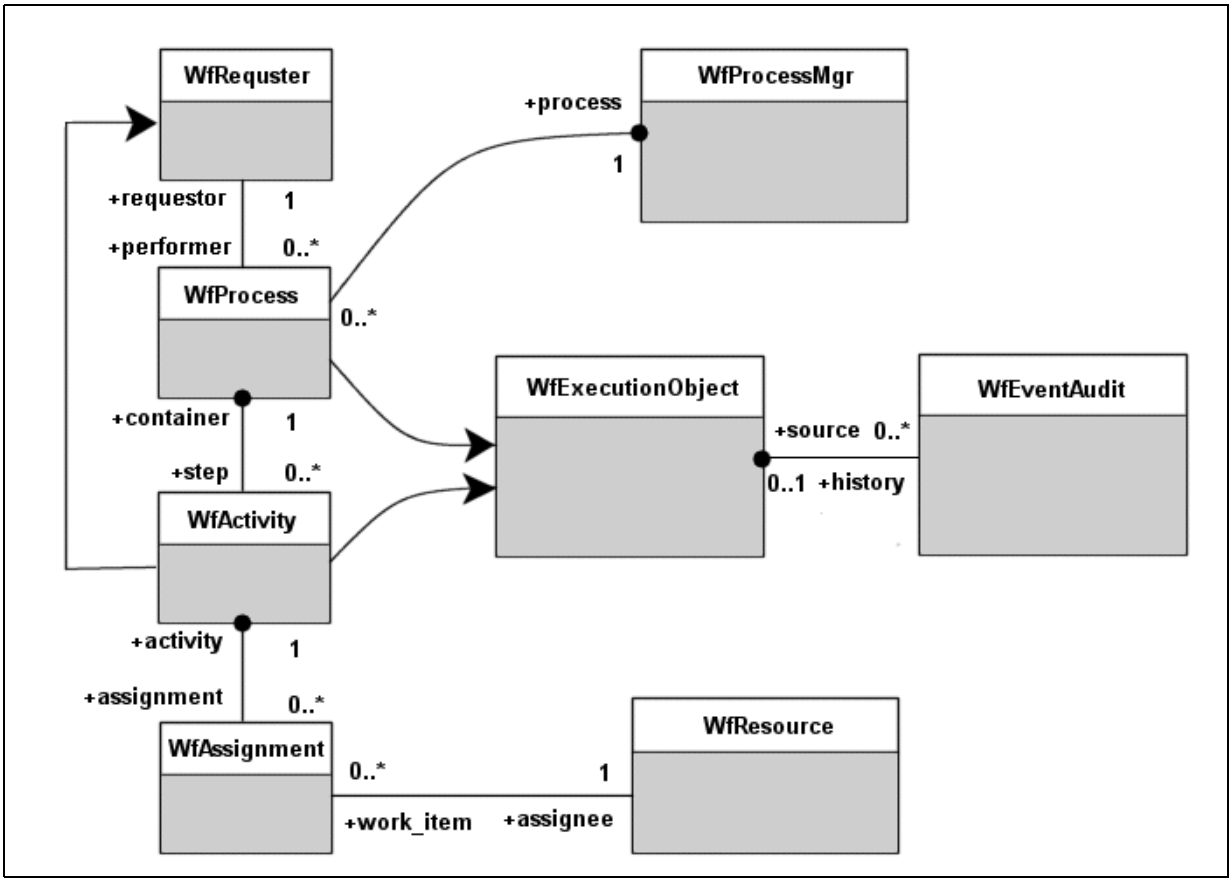

圖 7 CORBA Workflow Facility 架構圖

圖 7 是 CORBA Workflow Facility[10]的架構圖，只要各行業中有需要使用到工作流程的功能，都可以參考這個模型來開發工作流程系統，在這個參考模型中，定義了以下的 Class 及其關係：

WfRequester：要開始一個流程，必須先產生一個 WfRequester，以掌握這個物件的狀態，作為流程是否結束的依據。

WfProcessMgr：每個 Process 是由 WfProcessMgr 來管理，用來產生出各 WfProcess 物件。

WfProcess：可以用來啟動 Process，並啟動 Process 中的各步驟的 WfActivity。

WfActivity:是每個流程中不可分割的一部份,亦即是每個流程中的步驟,當 Activity 有需要使用到相關的 WfResource 時，會向 WfAssignment 取得相關的資源。

WfAssignment：每個 WfResource 必須要透過 WfAssignment 來做管理。

WfResource：每個物件對應到一個實際的資源，可察知資源是否使用中或是在可使用的狀態。

1.3 論文章節架構

本論文的第一章介紹提出本系統的動機及本系統相關的研究項目及相關組織所提出之標準架構；第二章介紹本系統的整體架構，並說明系統中各個模組的功能及實作所需用到的相關技術，除此之外，並說明各個模組間的互動關係；第三章說明系統實作時的界面設計及系統的實作畫面；第四章則提出整個系統的結論並說明本系統尚待改良之功能。

第二章 系統架構

2.1 CBWS 系統概述

有鑑於一般的工作流程系統大都是在伺服器端以一個工作流程引擎來控制所有和流程有關的事項，而且大都只是以處理文件的簽核為主，少有呼叫現有程式來達成各種必要的功能，這種方式的缺點在於伺服器端的工作流程引擎將會是一個非常主要的核心，如果工作量非常龐大的話，就會造成整個系統的瓶頸；另外一點是在無法呼叫並取得現有程式之間的資料，如果有資料要運算，可能還是必須透過人工的處理，會造成執行效率的降低、資料輸入錯誤的潛在危險性，也無法做到 Event-driven，使企業缺乏即時回應的能力。

為了克服以上的缺點，本論文建議一個分散式的工作流程系統架構，提出的系統架構如圖 8 所示，計有九個主要的部份，分別為 Certified Event Service、Organization Hierarchy、Workflow Process Definition Tool、Process Repository、Event Audit、Message Dispatcher、Message Parser、Event Filter、以及 Wrapper，在客戶端，將 Message Dispatcher、Message Parser、Event Filter 及 Wrapper 合稱為訊息前置處理器 (Message Pre-Processor)，下一節將會詳細的介紹各部份的功能以及使用的相關技術。

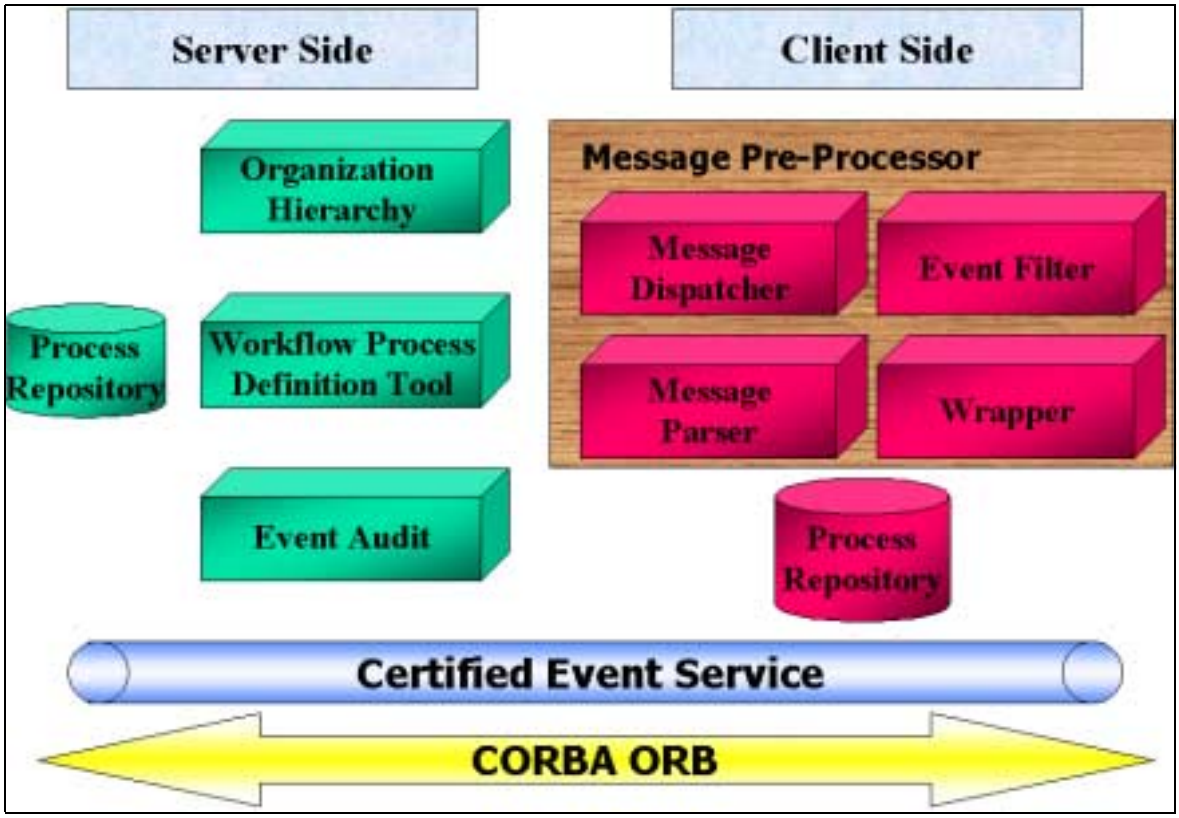

圖 8 分散式工作流程系統架構圖

2.1.1 Certified Event Service 的功能簡介

(1). 功能簡介

Certified Event Service 針對 CORBA COSS 中所訂定的 Event Service 加以擴充，原本所定義的 COS Event Service 並不能確保 Consumer 一定能收到 Supplier 送出的資料，而本模組要加入驗證 (Certified)的功能，保證 Consumer 一定能接收到 Supplier 所產生的訊息資料，不會發生訊息資料遺失的情況。為了達成這個目的，必須制定一個特別的機制，使得由 Supplier 所產生的訊息資料都是可驗證的。在這個機制中必須提供一個用來存放需要被驗證訊息資料的資料庫，一個負責管理整個 Event Channel 訂閱相關事項與維護整個驗證機制的管理者，以及一個 Consumer List 用來提供給管理者一份使用者訂閱狀況的參考資訊。而在 Certified Event Service 中，對於遞送的訊息資料格式，考慮以 XML 來定義訊息資料的格式，因為 XML 已經是一個描述資料的標準，許多的發展工具都已支援 XML，再加上 XML 本身對於描述資料具有易讀的特性，所以目前打算以 XML 做為考量。整個 Certified Event Service 的架構可以概分為六個部分(請參考圖 9)。

a. Supplier 主要負責發送訊息資料，當 Supplier 要發送訊息資料時會將訊息資料送往特定的 Event Channel。

b. Consumer 主要是負責接收訊息資料，當 Event Channel 裏有 Consumer 所需的資料，Consumer 就可由 Event Channel 取得這些資料。

c. Event Channel 主要負責接收來自於 Supplier 所送來的訊息資料，接到訊息資料後則將這些訊息資料送給 Consumer，因此所有的訊息資料傳送到各個 Consumer 的動作皆由 Event Channel 來處理。

d. Event Manager 是用來管理由 Supplier 所發送到 Event Channel 的訊息資料，以及記錄有訂閱的 Consumer。Event Manager 會將這些由 Supplier 暫留在 Event Channel 的訊息資料轉存入一個專門存放 Certified Event 的資料庫中，並且將參加訂閱的 Consumer 紀錄到 Consumer List 中。每個 Consumer 啟動時，都要先跟 Event Manager 註冊，而 Event Manager 會去檢查 Certified Event Database，把該傳給該 Consumer 的訊息資料送給它。

e. Certified Event Database 主要用來存放可驗證訊息資料的資料庫，由 Event Manager 所管理，當有訊息資料需要傳送時會置於 Certified Event Database 中，直到所有訂閱的 Consumer 都取走為止。

f. Consumer List 是一個存放訂閱者資訊的資料庫，由 Event Manager 所管理。

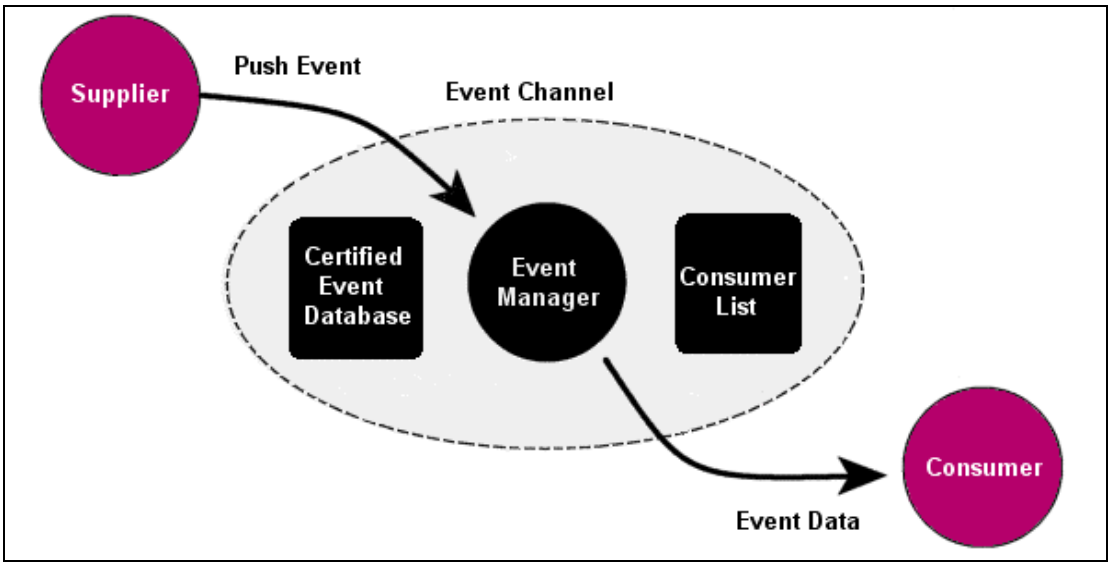

圖 9 Certified Event Service 架構圖

(2). 相關使用技術 COS Event Service

a. COS Event Service

根據 CORBA Event Service 所定義的標準規範，定義了兩種角色的物件，分別為 Supplier 及 Consumer，Supplier 產生(Produce) event data，Consumer 處理(Process) event data。CORBA Event Service 對於 Supplier 與 Consumer 之間訊息的遞送動作分為兩種模式：

Push 模式由 Supplier 主動推送訊息，Consumer 被動接收訊息。

Pull 模式由 Consumer 主動要求訊息，Supplier 被動拉送訊息。

Supplier 與 Consumer 之間訊息的傳送皆透過 Event Channel，而 Event Channel 提供 Supplier 與 Consumer 之間非同步方式溝通的管道。當一個新的 Event Channel 被建立後，並沒有任何的 Supplier 與 Consumer 與它相連接，Supplier 必須先對特定的 Event Channel 做註冊的動作而成為訊息發送者。Consumer 也必須先對特定的 Event Channel 做註冊的動作而成為訊息接收者。它們之間再以不同的模式對 Event Channel 執行通訊的動作。主要可以分為三種不同形式的通訊動作：

Push-Style：Supplier 將訊息資料推送到 Event Channel，再由 Event Channel 將訊息資料推送給 Consumer。(請參考圖 10)

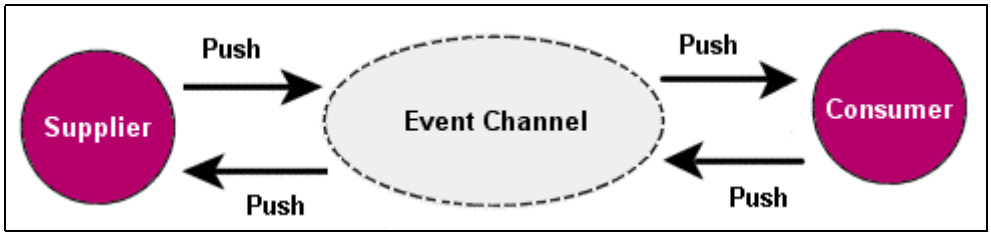

圖 10 Push-Style 表示圖

Pull-Style：Consumer 向 Event Channel 要求資料，再由 Event Channel 從 Supplier 取得資料傳回給 Consumer。(請參考圖 11)

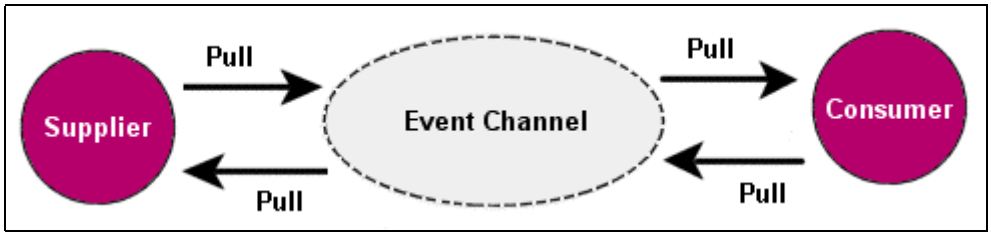

圖 11 Pull-Style 表示圖

Mixed-Style：Supplier 與 Event Channel 之間及 Consumer 與 Event Channel 之間分別使用 Push-Style 或 Pull-Style。(請參考圖 12)

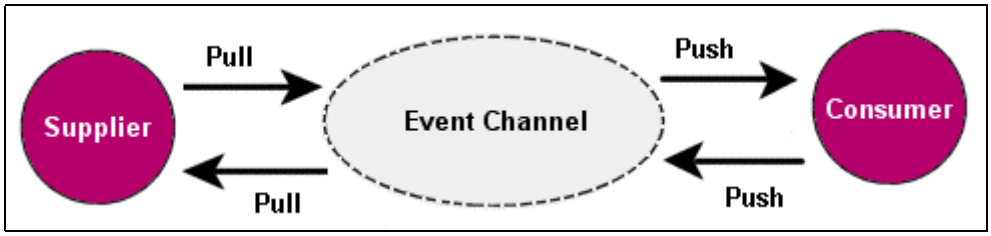

圖 12 Mixed-Style 表示圖

b. 模組互動關係圖

圖 13 說明 Certified Event Service 中各模組間的關係。

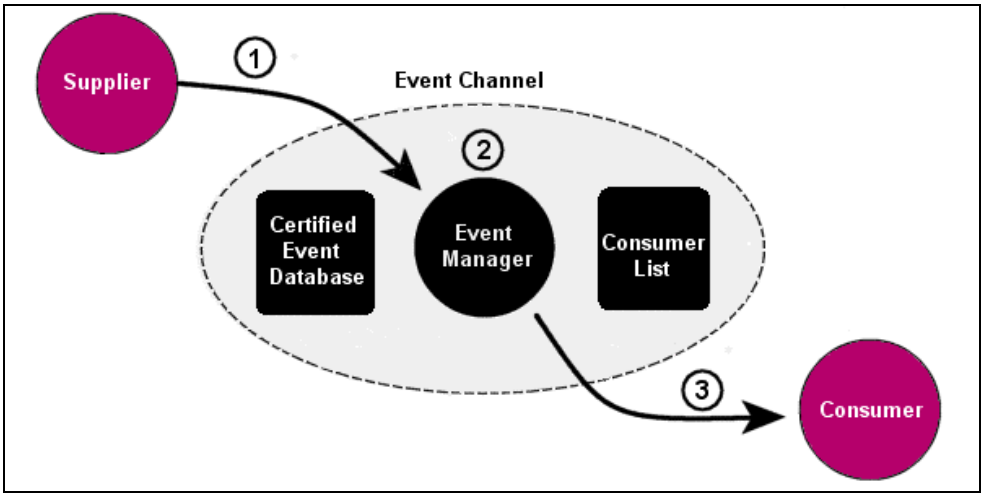

圖 13 模組互動關係表示圖

❶ Supplier 發送訊息資料到 Event Channel。

❷如果由 Supplier 所提供的訊息是需要認證功能的話， Event Manager 就會查看 Consumer List 中的資訊並將 Supplier 所發送的訊息資料存放到 Certified Event Database。

❸ Consumer 接收到 Event Channel 的訊息資料。在這個步驟中，Consumer 在收到 Event Channel 的資料後必須要做一個回應(Ack)的動作，Event Manager 會根據 Consumer 是否有回應來判斷 Consumer 是否已收到訊息，如果正常收到回應，就將 Certified Event Database 中的資料清除掉；如果 Consumer 沒有回應，則再重傳一次，如果連續三次都沒有收到 Consumer 的回應，則待傳的資料會存放在 Certified Event Database 裏，直到 Consumer 下次啟動，並向 Event Manager 註冊時才把訊息傳給 Consumer。

2.1.2 Organization Hierarchy 的功能簡介

(1). 功能簡介

在一個組織、企業中，都會具有一個描述組織階層的組織圖，其型式如同圖 14 所示，而每個人在這個組織圖中的責任與分工也有所不同。

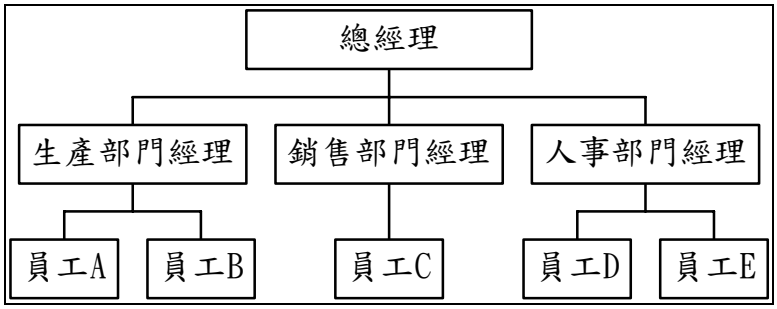

圖 14 組織圖範例

需要組織架構圖的目的有二，其一是使每個 Client Agent (客戶端代理人)能對應到實際組織上的人員，並將圖中每個人的責任權力及相關關連加以釐清，其二是在於所有相關的資料都是存放在此，應用程式執行時才存取本資料庫，並不是寫死在應用程式中，這使得當架構變動時，應用程式並不需要改變，使應用程式的開發維護更方便。

在本模組中，我們必須利用電腦所能表示、處理的方式來建構出組織架構圖，建出這個圖之後，除了能對應到每個客戶端代理人之外，還可以將每個人的權限、個人資訊也一併存於此資料庫中，為了要滿足這些需求，我們認為採用 JNDI (Java Naming & Directory Services)技術將可符合我們的需求，該項技術除了可以使用樹狀結構的方式將組織圖建立出來，還可以將相關的訊息一併存入，並提供一組建立以及查訊的 API (Application Programming Interface)供我們使用，另外的特點是其後端可以使用 LDAP、DNS、DB 等儲存資料的方式，提供一個更彈性化的方式供我們使用。

(2). 相關使用技術

在使用的技術上，採用 JNDI (Java Naming & Directory Services)來建立組織階層架構圖。

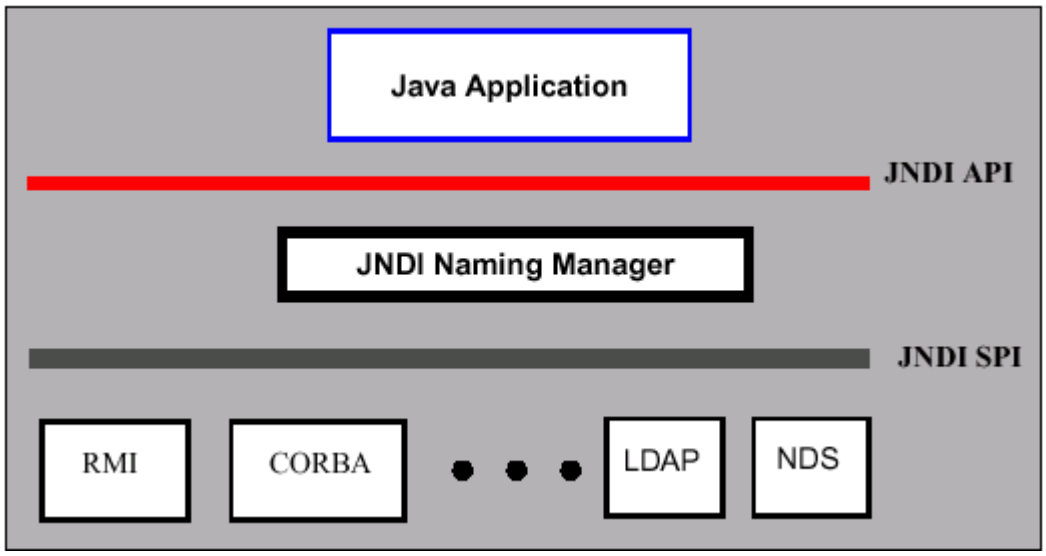

圖 15　JNDI 架構圖 (from SUN JNDI Spec.)

JNDI 提供一般化的介面(參考圖 15)，以取得所需要的物件或是服務的資訊。而資訊的來源可以是 Directory Service、naming service (DNS)、或是資料庫等等。

透過 JNDI 這種技術，可以提供系統一個較標準、有彈性的方式來建立組織的階層架構圖，這是採用此項技術的原因。

2.1.3 Workflow Process Definition Tool 的功能簡介

(1). 功能簡介

流程定義工具(Process Definition Tool)的功能是做流程的 Modeling 和描述並轉化成電腦內能儲存處理的資訊。本模組的前端將提供一個圖形使用者介面，該圖形介面可用來定義組織的工作流程，並按照 WfMC 提供的流程定義語言儲存起來，以提供給工作流程核心引擎處理。

流程定義工具必須提供一個可以將每個 Activity 的參與人員、所需資源、參考到的流程遞送規則及引發下一個遞送的條件，記錄在以標準化的流程定義語言所撰寫的 Process Repository 裡。

流程定義語言是 WfMC 為了要能將實際工作流程抽象化給電腦處理而訂出的標準

語言，只要使用者在儲存流程的相關資訊時能儲存成這個語言，便能讓各個不同開發者建構出來的工作流程系統取用，避免跨越不同的工作流程核心系統的流程發生時，無法順利為 Workflow Engine 解譯的問題。

(2). 相關使用技術

圖 16 便是 WfMC 建議使用的流程定義模組，包含下面幾個部份：

a. 工作流程型態定義 (Workflow Type Definition)

在定義工作流程的基本型態時，必須要提供以下幾個相關的資訊，包括：

❶工作流程名稱。

❷版本。

❸流程起始與中止條件。

❹安全性、稽查、和其它控制資料。

b. 步驟 (Activity)

要描述流程中的一個步驟，必須要提供的資訊有：

❶步驟名稱。

❷步驟型態(子流程或自動化流程)。

❸先前和後來步驟條件。

❹其它排程限制。

c. 轉換條件 (Transition Condition)

流程中可能會有流程的執行條件，用以決定流程的下一步要執行哪一個步驟，這些資訊也必須提供。

d. 工作流程相關資料 (Workflow Related Data)

在執行工作流程的過程中，必須要使用的一些資料，要使用前，必須要提供的相關資訊有：

❶資料名稱和路徑。

❷資料型態。

e. 角色 (Roles)

參與工作流程的相關角色，必須先標明角色的名稱和其所屬的部門資料。

f. 被呼叫的應用程式 (Invoked Application)

執行工作流程中，會呼叫到的應用程式，必須要提供的資料有以下各項：

❶一般型態和名稱。

❷執行參數。

❸找出資料所在位置。

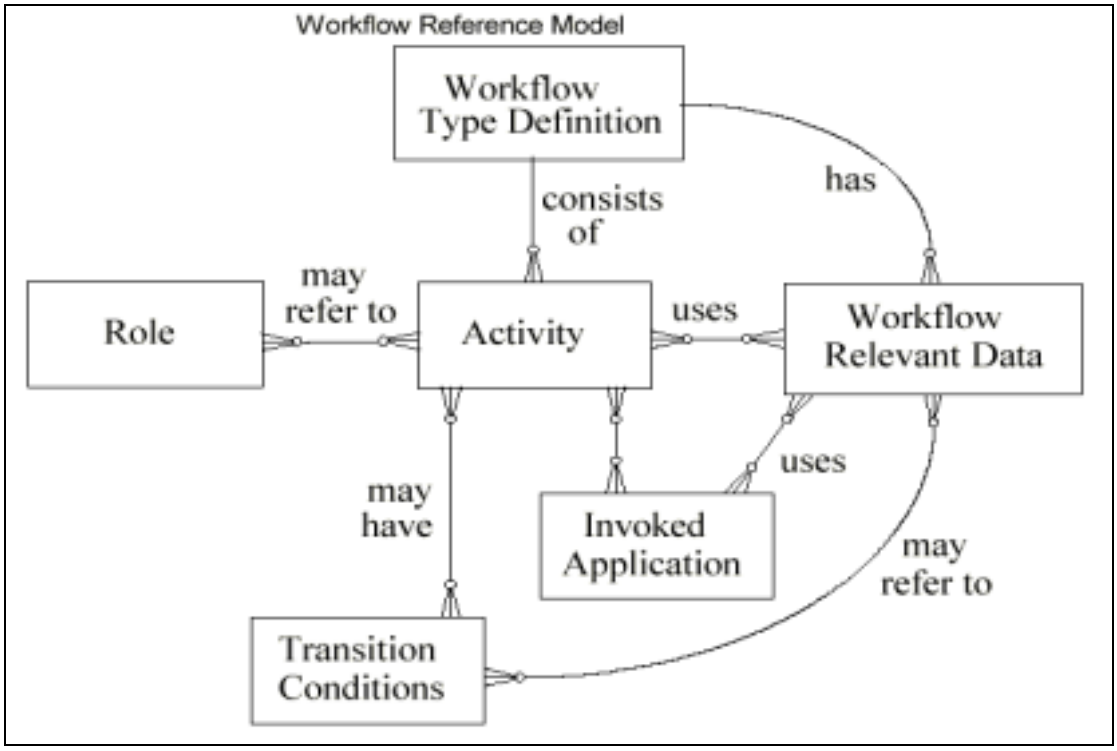

圖 16 Basic Process Definition Meta-Model

Process Definition Tool 將是一個圖形使用者介面(Graphic User Interface : GUI)工具，使用 Drag-Drop 來完成流程的定義。該相關定義資料會依 WfMC 標準的流程定義存在 Process Repository 之中。

在進行系統模組發展時，該 GUI Tool 的外觀可能如圖 17(取自 IBM MQSeries Workflow 網頁)所示，在該工具中會定義各種對應圖 16 模組的物件，使用者可以新增角色、執行條件、以及執行過程中所會用到的相關資料等，再利用拖、拉的方式將各個物件的順序及關係建立出來。

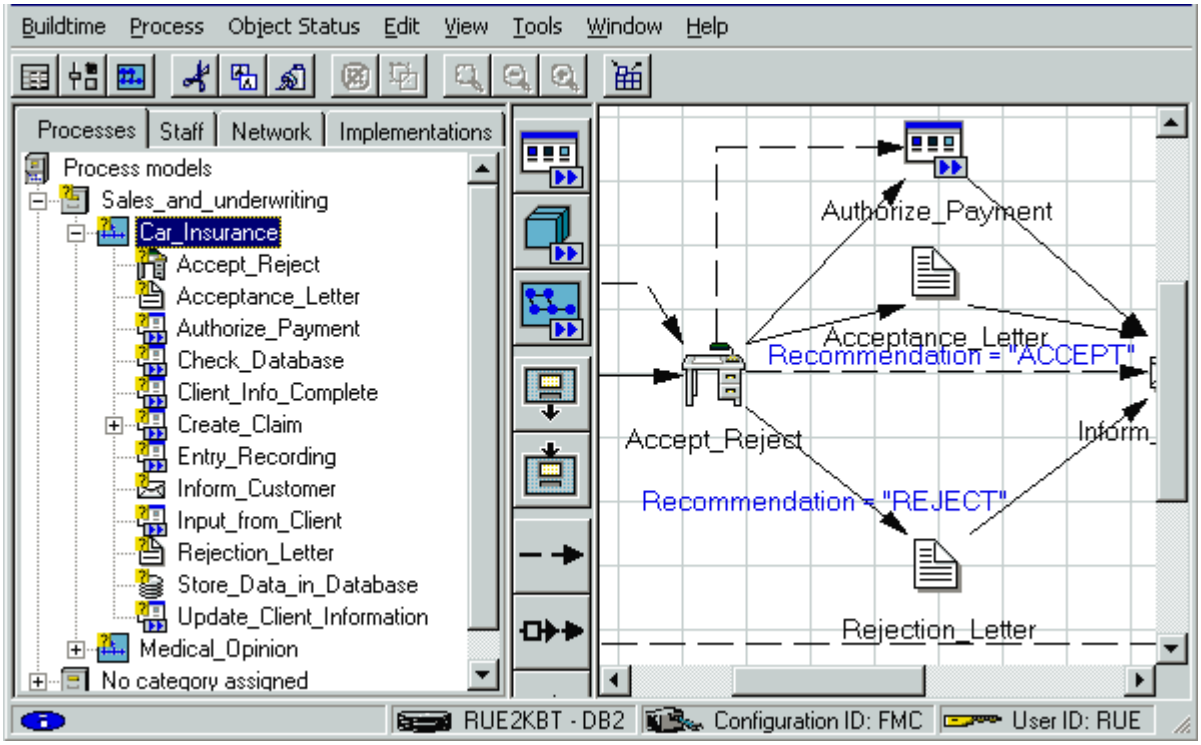

圖 17 Process Definition Tool 外觀樣式

2.1.4 Process Repository 的功能簡介

(1). 功能簡介

透過 Process Definition Tool 定義好相關流程資訊後，會將資訊以標準的描述語言儲存於 Process Repository 中，並分別存放在伺服器端及客戶端中，伺服器端中存所有的流程相關資訊，而客戶端則只存有客戶端本身相關的流程資訊，一旦流程本身的狀態有所改變時，要將更動過後的資訊傳給伺服器端的 Process Repository 中儲存，並更新受影響的相關客戶端的 Process Repository。

有了 Process Repository 提供隨時儲存客戶端狀態的改變及隨時更新伺服端的 Process Repository，在流程傳送過程中，Event Filter 就可根據本身 Process Repository 所記載的資訊，過濾 Certified Event Service 的資訊，只存取和自身相關的訊息資料。

(2). 相關使用技術

相關工作流程流訊，由 Process Definition Tool 定義後，會被記錄在 Process Repository 中，收集來的資訊，採用標準的描述語言儲存於系統中，該描述語言可能是 WfMC 目

前所用的規範(Workflow Process Definition Language)，或是目前逐漸被廣泛採用的 XML 格式描述語言。

以一個訂單流程的處理為例，用 Workflow Process Definition Language 定義的方式如下：

```
// ========================MODEL
MODEL                    'Inline_Block_Example'
WPDL_VERSION             "7.0    Beta"
VENDOR                   "…"
CREATED                  1998-07-15
NAME                     "Inline Block Example"
DESCRIPTION              ""
// ========================Workflow
WORKFLOW                 'Order management'
NAME                     "Order management"
DESCRIPTION              ""
// ----------------------------------------ACTIVITY
ACTIVITY                 'Order_management.31'
NAME                     'Order management.31'
DESCRIPTION
IMPLEMENTATION           NO
PERFORMER                'Internal_marketing'
START_MODE               MANUAL
FINISH_MODE              MANUAL
WAITING_TIME             0
DURATION                 3600
COST                     "0.00"
DOCUMENTATION            ""
INLINE_BLOCK_BEGIN       'V_Order_management.31'
END_INLINE_BLOCK_BEGIN
EXTENDED_ATTRIBUTE       'X_POS'     FLOAT     44.391972
EXTENDED_ATTRIBUTE       'Y_POS'     FLOAT     62.630359
EXTENDED_ATTRIBUTE       'QUANTITY' .INTEGER   0
END_ACTIVITY
```

```
// ---------------------------------------- --TRANSITION Information
TRANSITION                    'Transition 1'
NAME                          ""
FROM                          'Order_manegement.31'
TO                            'check_credit_standing.8'
DESCRIPTION                   ""
EXTENDED_ATTRIBUTE            "TRANS_TYPE' STRING "INFO_TRANS"
EXTENDED_ATTRIBUTE            "INFORMATION' STRING "order_data"
EXTENDED_ATTRIBUTE            "VARIABLECOSTS' FLOAT 0.00
EXTENDED_ATTRIBUTE            "FIXEDCOSTS'    FLOAT 0.00
EXTENDED_ATTRIBUTE            "TRANSPORTTIME INTEGER 600
EXTENDED_ATTRIBUTE            "ITEMCOUNT'     INTEGER 1
EXTENDED_ATTRIBUTE            "DESTINATIONCOUNT' INTEGER 1
END_TRANSITION

END_WORKFLOW

END_MODEL
```

接著要考量的就是要如何儲存 WPDL，可能使用的儲存媒介有：

❶使用關聯式資料庫。

❷使用物件導向資料庫。

❸使用 XML 儲存。

在本系統中，儲存媒介主要考量的要點是具有跨平台的能力，因為這個儲存媒介要放在伺服器端與各個不同的客戶端，因此必須要能方便的安裝在各個平台之上；除此之外，必須要能快速的由資料庫中取得我們所需的資料，故亦為考量的重點。

2.1.5 Event Audit 的功能簡介

(1). 功能簡介

由於本架構是一個分散式的工作流程系統，並不是一個中央集權式的工作流程系統，因此無法像中央集權式的工作流程系統能夠由工作流程引擎中知道目前各個工作流程執行的狀況，因此提出一個 Event Audit 的模組以達成監督整個工作流程執行狀況。

工作流程是由客戶端所引發的，當每個工作流程開始之後，將訊息由一個 Activity 送到下一個 Activity 之前，要先到 Event Audit 做一個登記的動作，Event Audit 透過這些資料，就能掌握目前所有工作流程執行的狀態。

為了能有效的掌控各個工作流程的狀況，必須要有一個機制能顯示執行中的各個物件的狀態，將利用 CORBA Workflow Facility 所訂出來的各物件之間的關係來表示。

原本的 CORBA Workflow Facility 是適用在發展一個 Workflow Engine，但是在此用來發展成一個監督工作流程的機制也很適用，CORBA Workflow Facility 的相關技術會在以下陳述。

(2). 相關使用技術

CORBA Workflow Facility 的模型如圖 18 所示，定義了 WfRequster、WfProcessMgr、WfProcess、WfActivity、WfAssignment、WfResource 這些物件以及物件之間的關係。

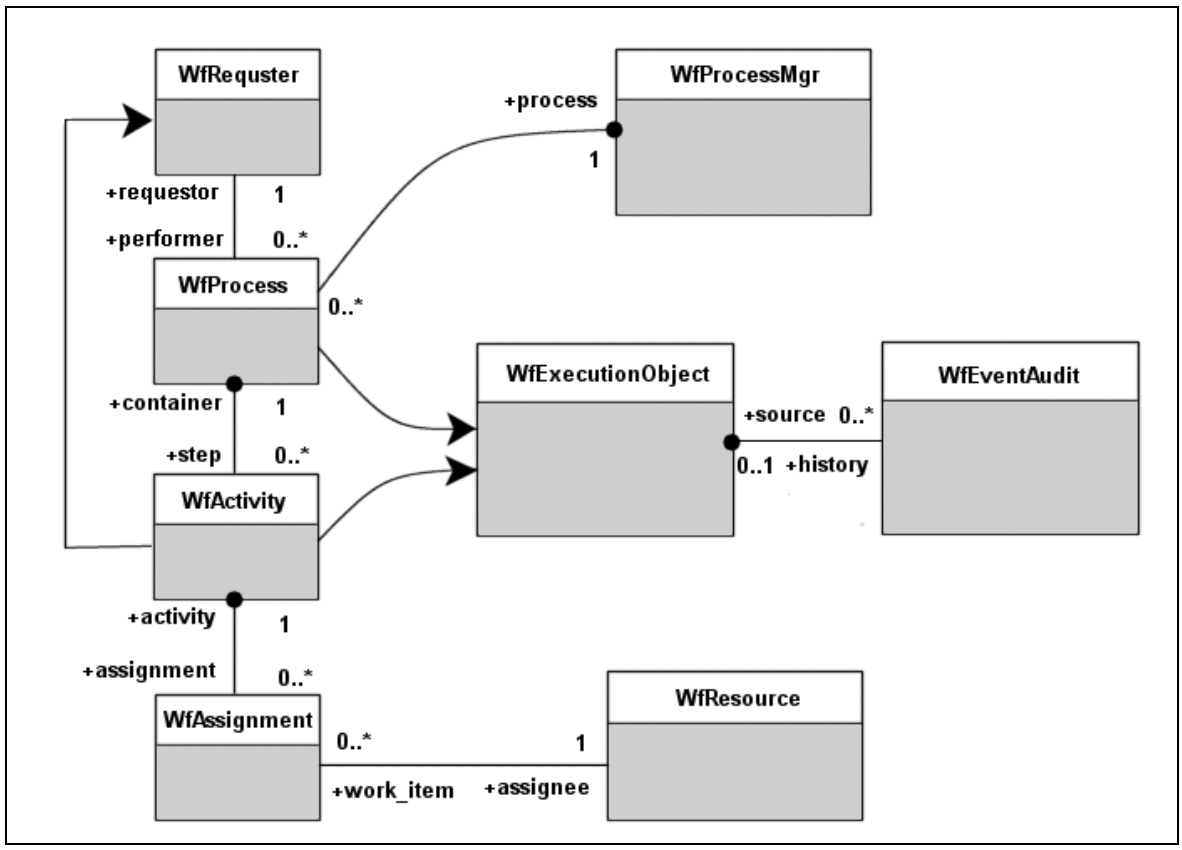

圖 18 CORBA Workflow Facility 架構圖

當客戶端引發一個新的工作流程時，就會依序產生 WfRequster、WfProcessMgr、WfProcess，再根據 Process Repository 中的記載，產生相關的 WfActivity 及取得所需的 WfResource。

當客戶端做完一個步驟之後，將相關訊息傳送到 Event Audit，由 Event Audit 更新相關的 WfAcitvity，如果一個 WfProcess 內的所有 WfActivity 都完成之後，則視為該 WfProcess 已全部完成。

本監督程式模組在實作時，將會提供一個圖形介面，在圖形介面裏將各物件的狀態表達出來，就可以表達出各個相關的工作流程目前進行的進度如何，可以讓監督者快速的了解各流程的現況。

2.1.6 Message Dispatcher 的功能簡介

本模組提供的功能是根據所設定好的 rules，將訊息作分配和繞送(如圖 19)。功能應用主要包含前端的 Message Dispatcher Graphical User Interface (GUI)和後端 Dispatching Rules Engine 兩個部分 。Message Dispatcher GUI 主要用來定 Dispatching Rules，Rules Editor 則提供一個定義 functions 的表單，來定義規則，同時 Message Dispatcher 必須提供 Message Routing 的功能，來設定訊息傳遞的流程。Dispatching Rules Engine 則在 run-time 執行那些已在 design-time 制訂好的規則。

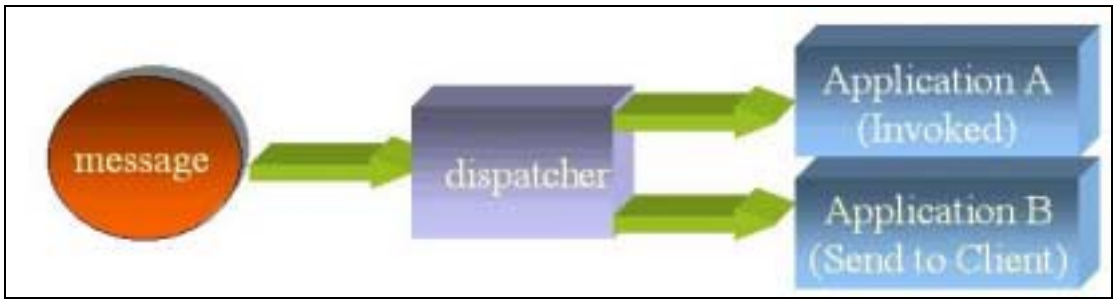

圖 19 訊息分配模組架構圖

2.1.7 Message Parser 的功能簡介

Message Parser 的目的，是將資料轉成適合於不同平台或不同系統的資料型態，轉換後的訊息可根據組合過的原始資訊和資訊內容作訊息的傳遞，其概念如圖 20。我們將提供一個功能，做資料的轉換，為了符合擴充性的需求，可能採用 XML 的架構，將各

種不同的資料型態轉成 XML 的格式，此 XML 格式的 message 中將包含 header 和 content，在 header 中則會紀錄發送地點、接收地點和 Process_ID 等等資訊，而 content 則是經由 mapping 的動作將不同的 data type 轉成 XML 的格式。

不同的資料轉成 XML 後，由 Wrapper 將這一些訊息傳到 Certified Event Service，各個部門接收到訊息並加以處理後，可以利用 Message Parser，將 XML 轉成符合該系統使用的 data type，然後交給各系統處理。

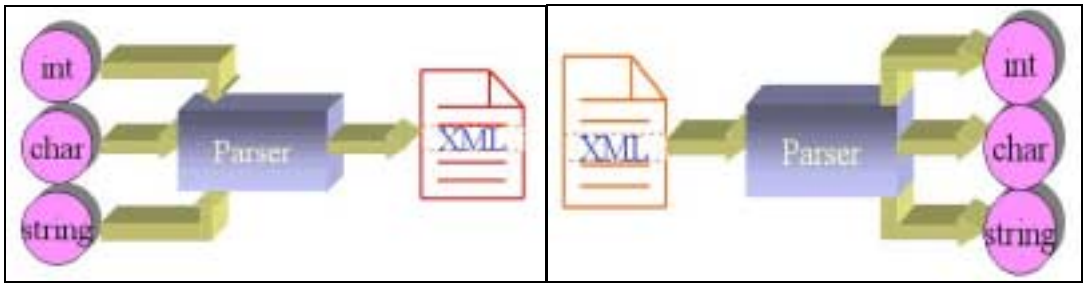

圖 20 Message Parser 模組架構圖

Message Parser 的特徵：

(1). 輸入的欄位對應成輸出的欄位。

(2). 欄位轉換的語法。

(3). 轉換的從新使用和儲存。

2.1.8 Event Filter 的功能簡介

本模組只有在接收訊息時才會用到，主要的功能在做訊息過濾的動作，由於 Certified Event Service 用廣播的方式將訊息傳送到各個客戶端，客戶端必須要知道傳送來的訊息是否要接收並加以處理，客戶端的 Event Filter 會根據訊息表頭所記載的內容和客戶端本身的 Process Repository 所存有的資訊來判斷傳進來的訊息是否和客戶端相關，若是則接收，交由 Message Parser 及 Message Dispatcher 處理；若和客戶端無關，則直接捨棄掉這份訊息。

Event Filter 判斷的方式是依照資料內含的 Process ID、發送地點、接收地點和本身的 Process Repository 來篩選該筆資料是否需要接收，資料的可能格式如下所示：

Process ID	From	To	Content

2.1.9 Wrapper 的功能簡介

Wrapper 這個模組只有在客戶端有要傳出訊息時會用到，Wrapper 會依據客戶端 Process Repository 儲存的資料，將資料標示上 Process_ID、發送地點、接收地點後，將相關訊息通知伺服器端的 Event Audit，Event Audit 確認收到後，將附上 Process_ID、發送地點、接收地點的訊息放到 Certified Event Service 上，Certified Event Service 的機制會確保訊息會被下一個要接手流程工作的客戶端收到，下一個客戶端的 Event Filter 就可以依據 Wrapper 加在資料之前的相關訊息判斷是否要接收處理這筆資料。

2.2 模組互動關係

2.2.1 在伺服器端定義工作流程

在伺服器端定義工作流程的執行程序如圖 21，執行過程的描述如下：

(1). 使用者可以在伺服器端透過 Process Definition Tool 提供的使用者界面來定義企業的工作流程。

(2). 利用 GUI 界面定義時，必須指定作業必須由某個外部程式或使用者處理，在此時必須參考到依循企業組織圖建立出來的 Organization Hierarchy 系統模組。

(3). 利用圖形界面將作業流程建構出來之後，再將這些資訊利用標準描述語言儲存於系統中，該描述語言可能是 WfMC 目前所用的規範，或是目前制定中的 XML 格式的描述語言。

(4). 在伺服器端定義好之後，要將和每個客戶端相關的工作流程定義資訊傳送到客戶端的 Process Repository 中儲存，這些資訊必須包含該客戶端在特定工作流程中的地位，及其前一個與後一個 Client Agent 的資訊，以備以後客戶端要接收資料時用來篩選判斷之用。

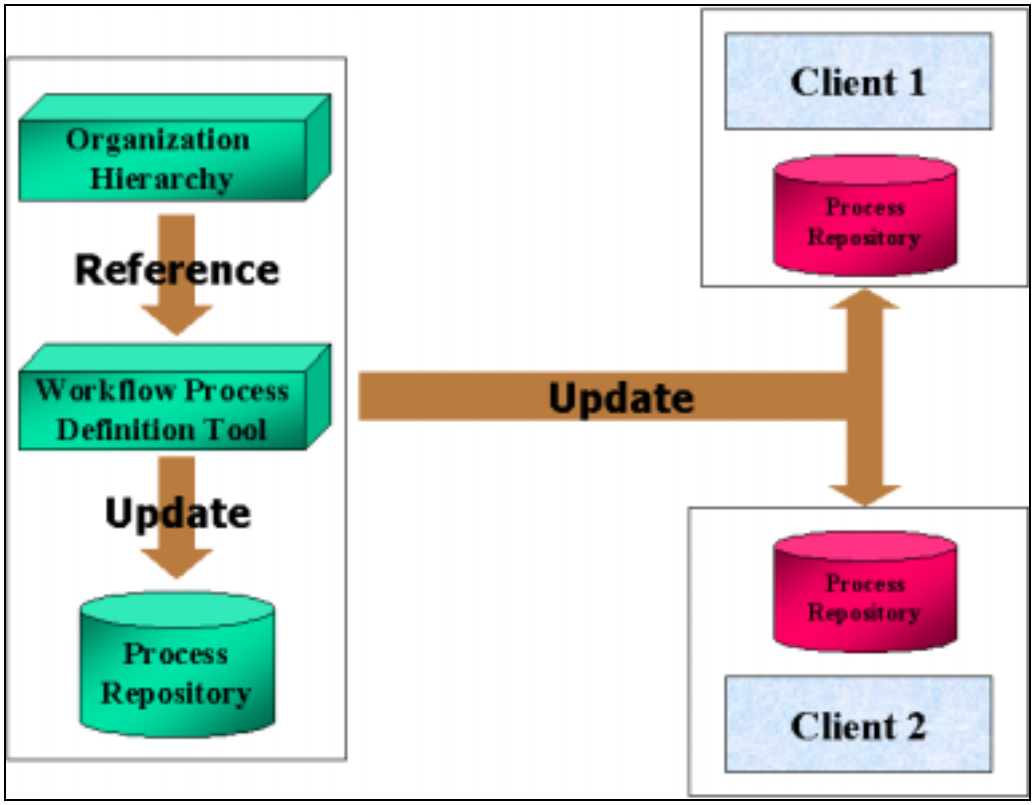

圖 21 在伺服器端定義工作流程

2.2.2 在客戶端定義工作流程

本系統提供一個功能，讓使用者在客戶端也可以定義工作流程，在客戶端定義工作流程與在伺服器端定義工作流程的最主要差異是在於客戶端定義的時候只能改變和客戶端本身相關的部份工作流程定義，而非像在伺服器端進行定義時定義的範圍不受限制。

在客戶端進行定義的流程如圖 22 所示，步驟如下：

(1). 客戶端開啟 GUI 界面的 Process Definition Tool，改變和本身相關的工作流程。

(2). 客戶端可能必須參考伺服器端的 Organization Hierarchy 資訊，但會受到存取權限的限制，只能觀察到和自身流程相關的訊息。

(3). 更新修改完之後，會將資訊以標準的描述語言儲存於客戶端 Process Repository 中，藉以在過濾訊息時使用。

(4). 要將更動過後的資訊傳給伺服器端的 Process Repository 儲存，並更新受影響的相關客戶端的 Process Repository。

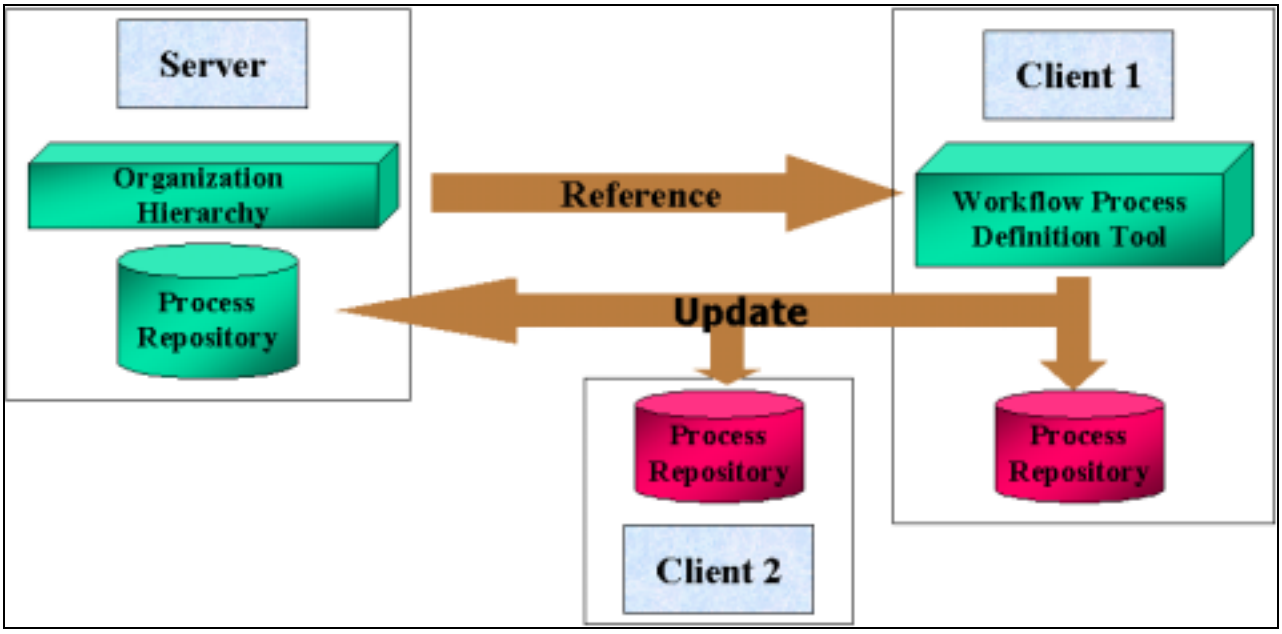

圖 22 在客戶端定義工作流程

2.2.3 執行工作流程

當實際在執行工作流程時各模組的關係和執行順序如圖 23 所示，以下就各個步驟加以說明：

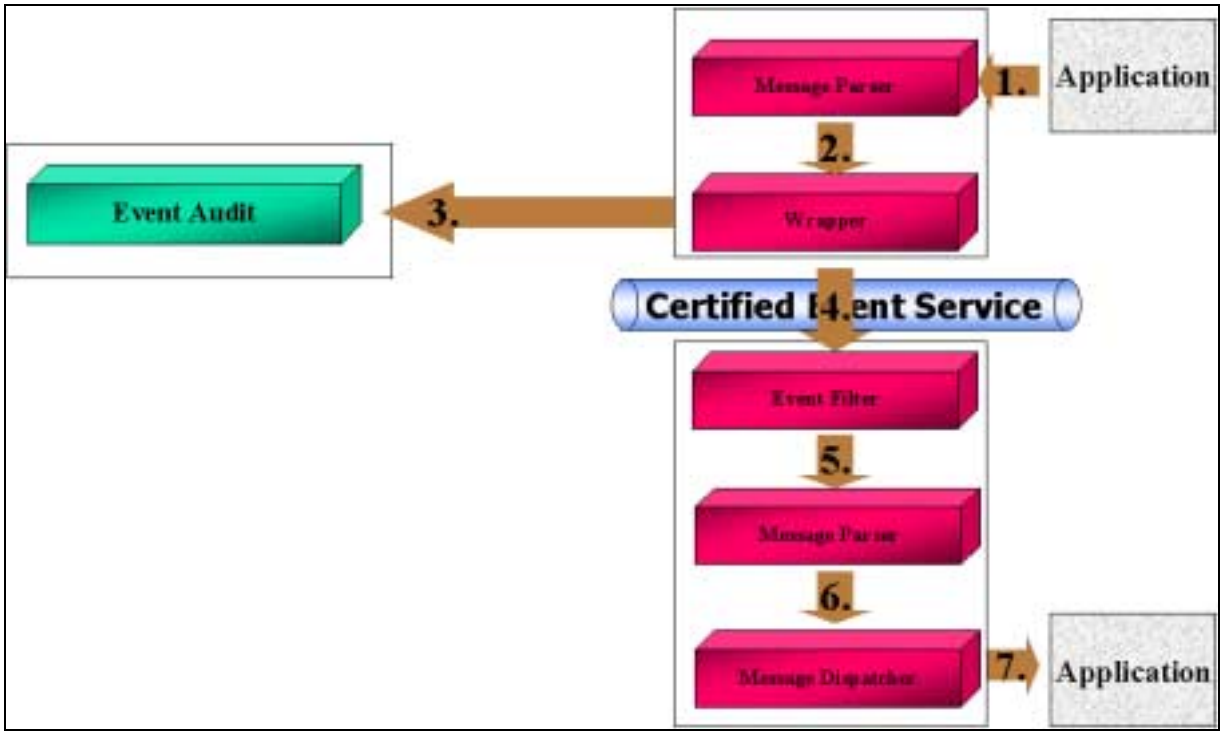

圖 23 工作流程執行過程

(1). 應用程式或使用者開始啟動一個工作流程，欲傳送的資料先透過 Message

Parser 將之轉換為 Certified Event Service 中傳送的共通資料型態，以便透過 Certified Event Service 傳送相關資料。

(2). Wrapper 依據客戶端 Process Repository 儲存的資料，將資料標示上 Process_ID、發送地點、接收地點後，將相關訊息通知伺服器端的 Event Audit。

(3). Event Audit 會將這些資料做為監督流程執行狀況之用，由於每次客戶端要做任何改變之前都會通知 Event Audit，所以 Event Audit 會知道目前的執行狀態為何。

(4). Wrapper 將附上 Process_ID、發送地點、接收地點的訊息放到 Certified Event Service 上，下一個要接手流程工作的客戶端就會接收到相關的資料。

(5). 相關的客戶端會利用 Event Filter 依據本身 Process Repository 所記載的資料篩選 Certified Event Service 傳送過來的訊息，藉由訊息中的 Process_ID、發送地點、接收地點這些資料過濾訊息。

(6). 客戶端的 Message Parser 收到這些資料以後，會將訊息轉為後端 Application 所能處理的資料型態。

(7). 客戶端的 Message Dispatcher 收到這些轉換過型態的資料後，會將訊息轉送給相關的 Application，要求 Application 做相關的處理，

第三章 系統架構實作

3.1 實作環境

本論文中所提到的系統預計採用 JAVA 程式語言配合 OMG 的 CORBA 架構開發雛型系統，對相關環境的描述於各小節所述。

3.1.1 JAVA 程式語言

JAVA 程式語言為昇陽公司 (Sun Micro System)所設計，具有跨平台的特性，經過多番的修正與改進，目前正式的版本為 1.2 版，並在進行 1.3 版的制定與開發。

JAVA 語言有以下幾個特性：

- 物件導向
- 強固性
- 安全性
- 平台獨立性
- 可攜性
- 動態連結

這幾個特性都可以幫忙系統建構。除了語言本身之外，昇陽公司也附帶提出了一些與 JAVA 相關的技術以協助程式開發人員進行開發，並為了適合在不同的開發環境，除了標準版本之外還另外提出了 JAVA 2 Enterprise Edition (J2EE)與 JAVA 2 Micro Edition (J2ME)分別適用於企業級和嵌入式系統的開發。

本論文所採用的部份包含了 JAVA 程式語言、JAVA Bean 物件模型、JDBC 資料庫存取、JNDI 目錄存取、及物件序列化等技術。

3.1.2 JAVA 與 CORBA 的關係

使用 JAVA 搭配 CORBA 有幾個好處：

- 寫出符合分散式物件標準的跨平台程式，減少系統在不同平台上執行時的困擾。

- CORBA 與 JAVA 能互相整合，JAVA 由 1.2 版開始提供了 JAVA IDL 以便與 CORBA 物件溝通，而利用 JAVA 原有的 RMI (Remote Method Invocation)規板開發的程式若需要和 CORBA 物件溝通的話，亦可以透過 RMI over IIOP 協定達成，使 JAVA 與 CORBA 的合作能力更佳。
- 由於 JAVA Applet 可在 Internet Explorer 及 Netscape Communicator 瀏覽器上執行，如果將系統的使用者介面開發成 Applet 再透過 IIOP 與系統溝通，在程式的分配安裝上可節省很多時間。

3.1.3 Borland 公司的 CORBA 解決方案

Borland 公司提供了各種 CORBA 的解決方案，以提供程式開發者使用，以下是其提供的產品項目：

- Visibroker for JAVA/C++
- Visibroker Gatekeeper
- Visibroker SSL Pack
- Visibroker Integrated Transaction Service
- Application Server
- Borland JBuilder Development Tool

在系統開發時，採用 Borland JBuilder 3 Enterprise 及其中內含的 Visibroker 3.4 做為系統的開發環境。

3.2 模組設計

在以下的兩小節中將介紹本系統界面定義及實作，在界面定義方面以 UML 的方式表達，而系統的執行狀況將於第二小節中介紹。

3.2.1 CBWS 之界面設計

Certified Event Service 的設計界面如圖 24 所示，繼承自原 CORBA Event Service 並加入 Event Manager 的功能，使得實作出來的模組能具有確保訊息可被送達的功能。

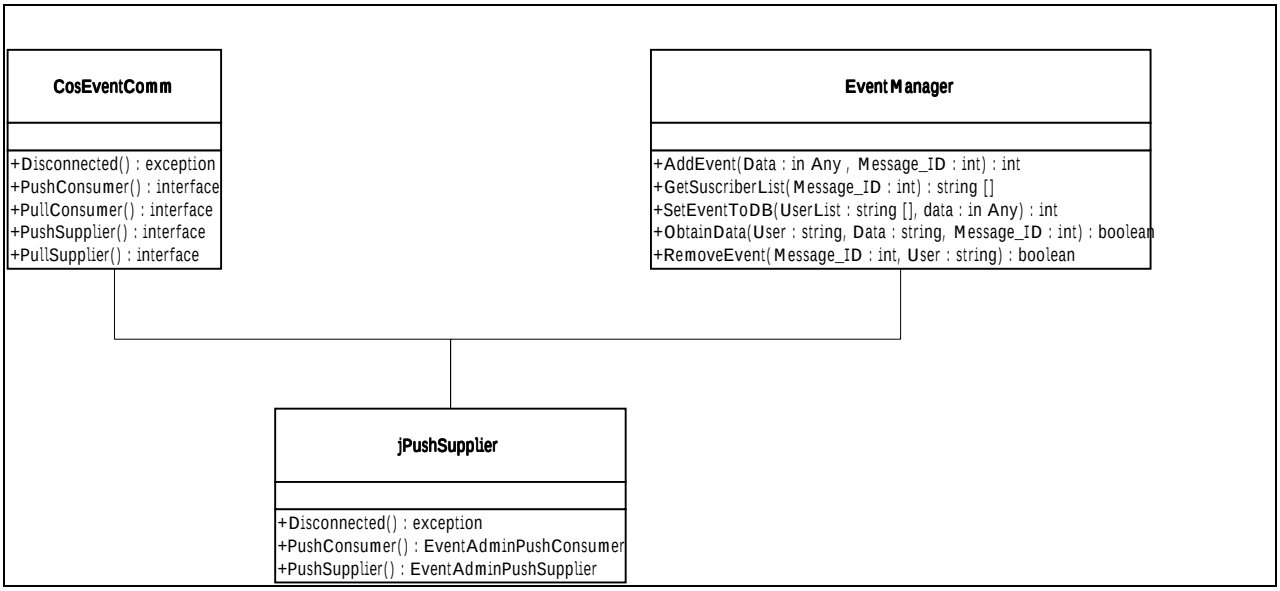

圖 24 Certified Event Service 界面設計

Organization Hirarchy 的界面設計如圖 25，本模組的後端存取 LDAP Server，在存取時使用的是 Sun J2EE 的 JNDI，故在模組的設計上，先繼承 javax.naming.directory 後再逐步設計出模組。

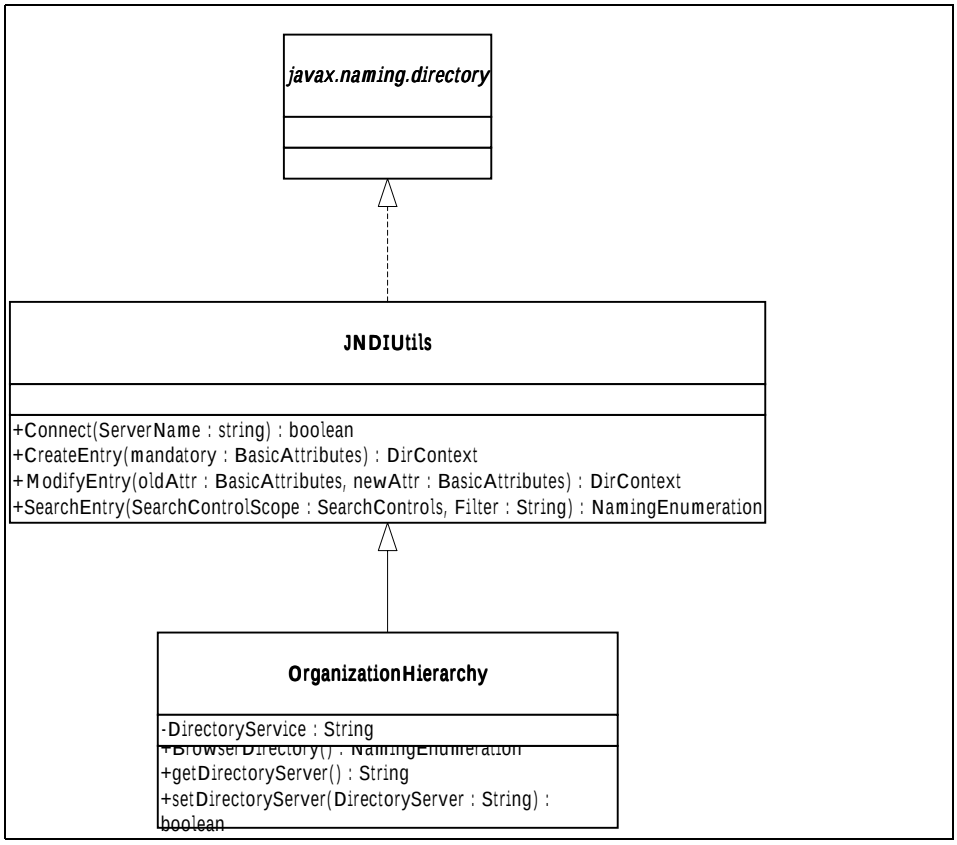

圖 25 Organization Hierarchy 界面設計

Process Repository 的界面設計如圖 26 所示，由於必需處理 XML，採用了 ObjectSpace

所提供的 Dynamic XML 程式模組，因此本模組繼承自該程式模組；另外， 為了要處理 Workflow DTD 內所定義的各個標籤，必需使用到由 ObjectSpace XGEN 所生的各個界面；故本模組繼承自這些程式模組，並擴充適當的功能之後完成本模組。

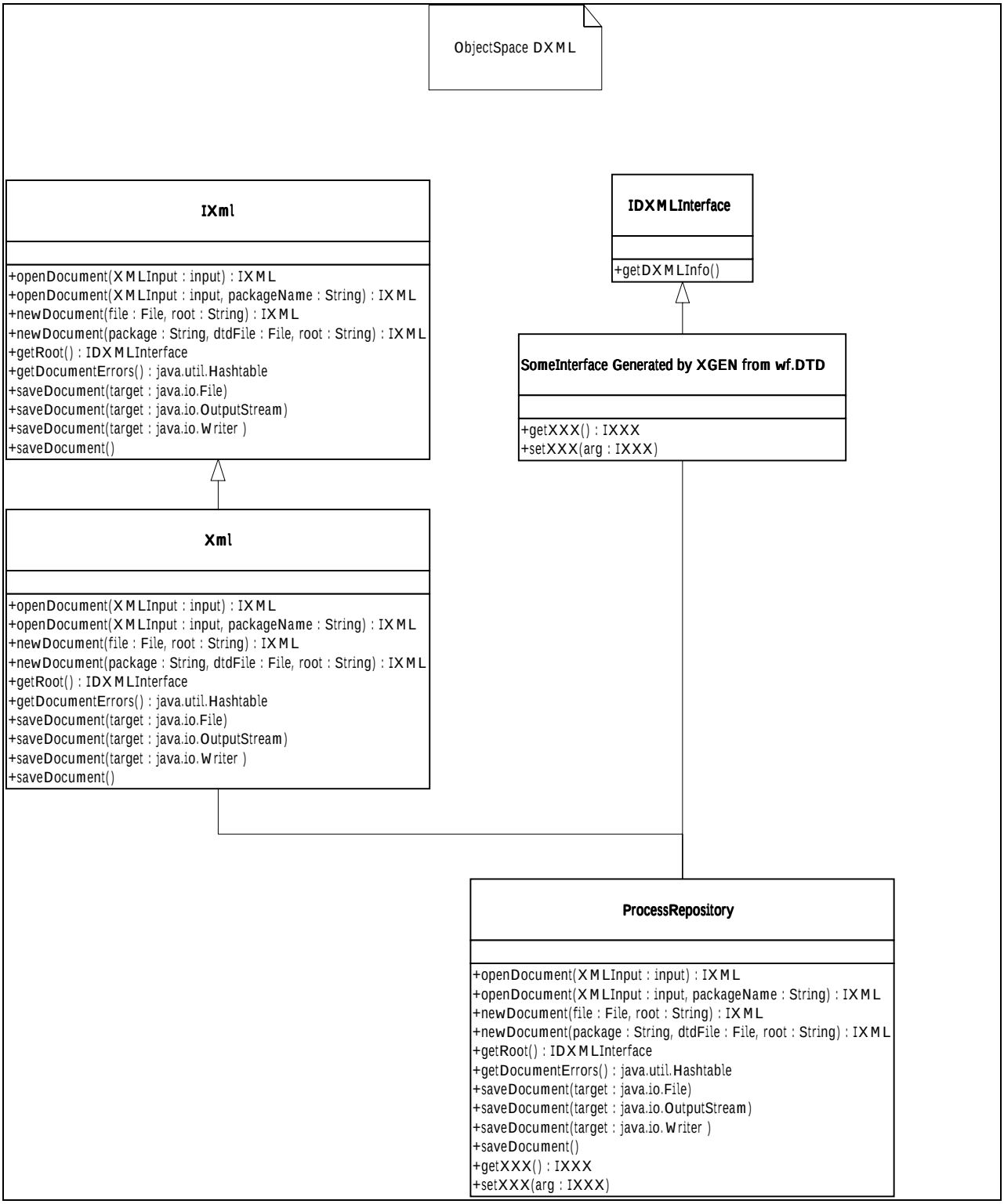

圖 26 Process Repository 界面設計

Message Dispatcher 的界面設計如圖 27，其功用在指定欲呼叫的應用程式及欲傳送的資料或檔案，完成後將屬於客戶端的 Pre-Processor 的一部份。。

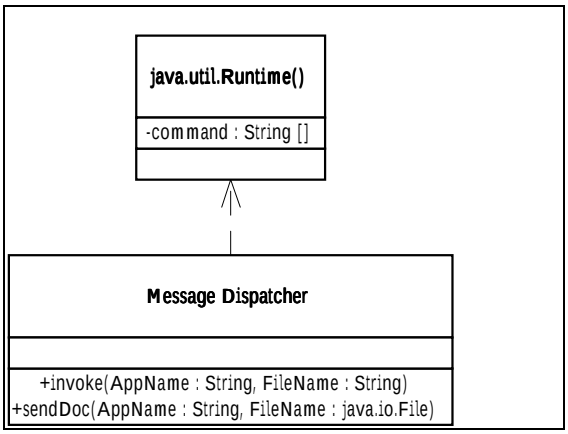

圖 27 Message Dispatcher 界面設計

Message Parser 的界面設計如圖 27 所示，由於必需做型態的轉換，而所有欲傳送的資料在傳送前都要先轉換成 XML，故必需擴充先前所開發的 Process Repository 模組，以期透過該模組處理 XML 文件，而型態轉換的功能則為本模組所加入。

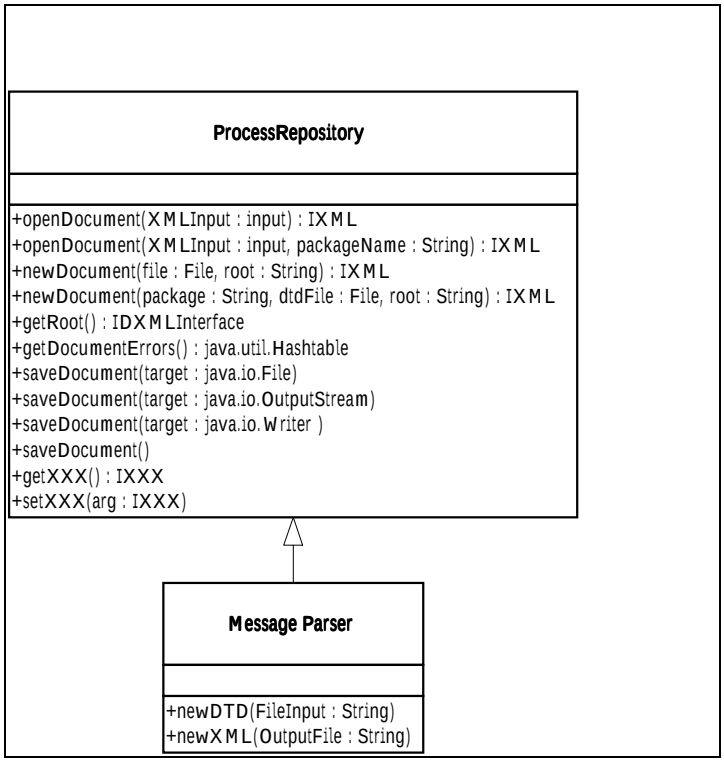

圖 28 Message Parser 界面設計

Event Filter 的界面設計如圖 29，本模組必須具備對 XML 文件的內容進行搜尋比對的功能，這部份的功能透過 IBM XML Bean Suite 中的 XML Search Bean 來達成，故本模組繼承該程式模組，並就搜尋所需之功能加以擴充而完成本模組。

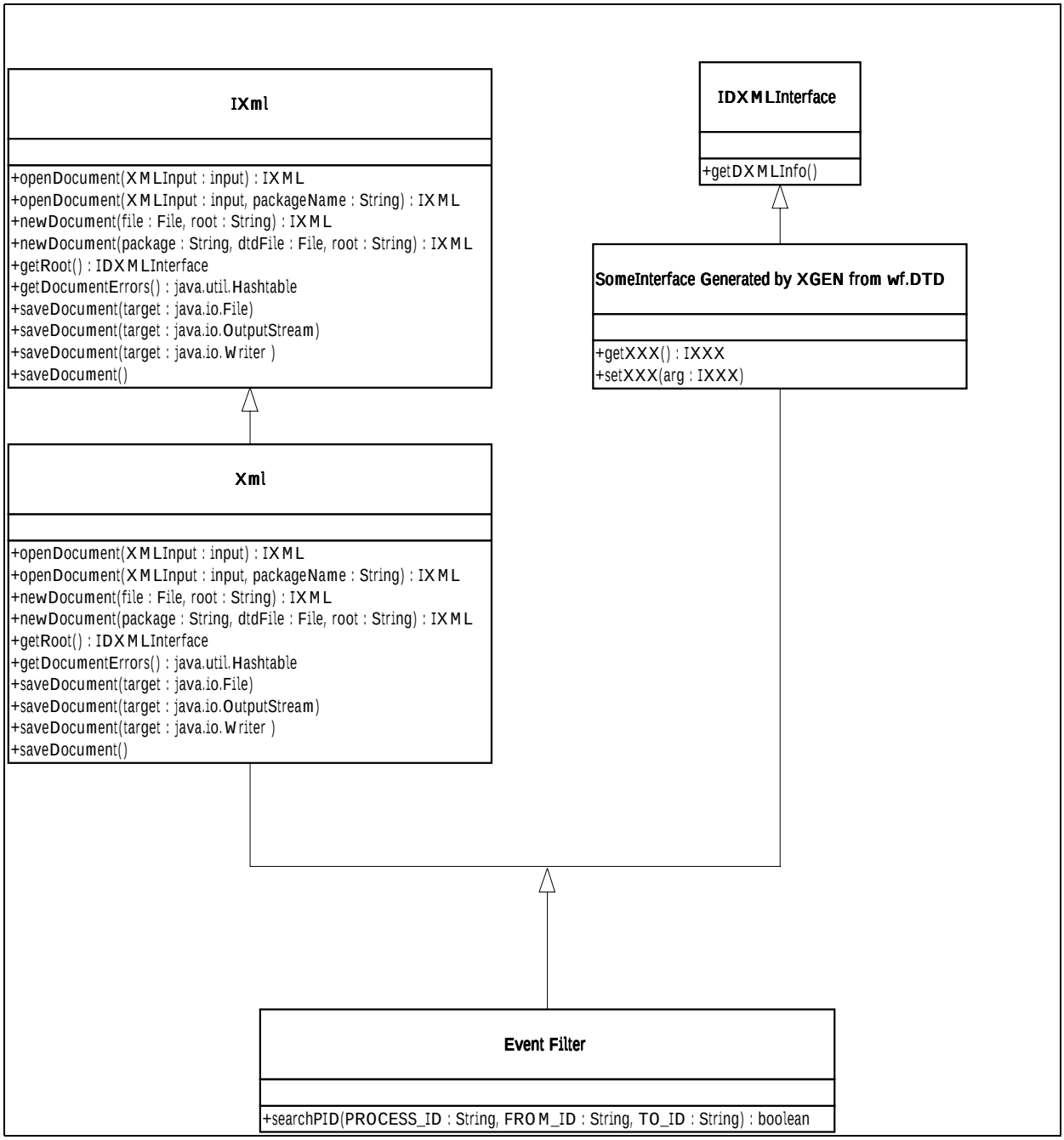

圖 29 Event Filter 界面設計

3.2.2 系統設計

本系統執行的流程概圖如圖 30 所示，在每個客戶端都配置一個 Message Pre-Processor，該模組負責將客戶端的資料轉換成 XML 的型態，再將這些文件放置到 Certified Event Service 上供所需之客戶端透過 Message Pre-Processor 存取及處理。

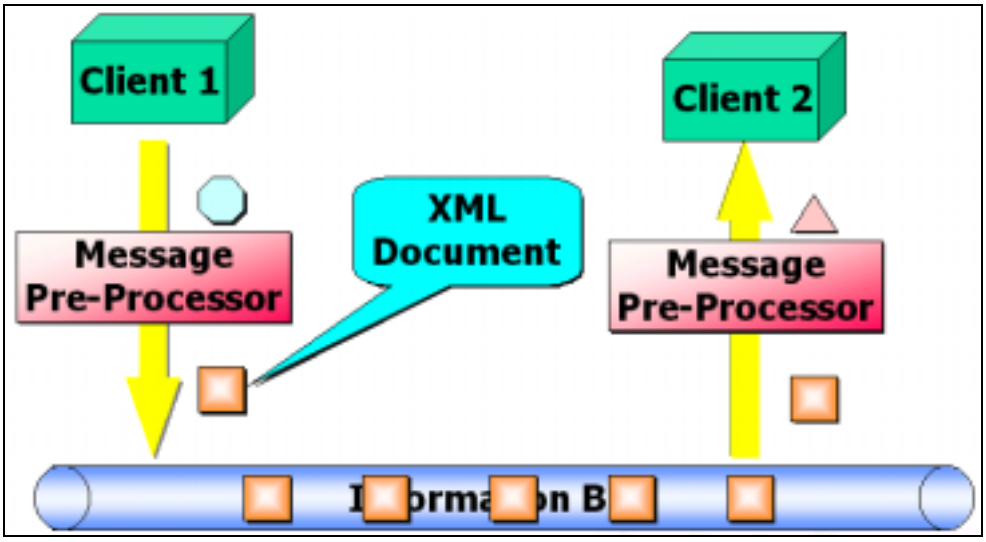

圖 30 系統執行流程

Certified Event Service 的執行流程如圖 31，擴展 CORBA Event Service 的功能，將 CORBA Event Service 上的資料做一個備份儲存在資料庫中，再將之傳送出去，等到該訊息被正確的接收之後，才由資料庫中清除。

在本部份的程式設計上，透過良好的程式包裝，將 CORBA 及 Event Service 的相關細節儘量簡化，讓應用程式在開發時，只要建立一個連接物件，就可自動連上 CORBA ORB 及 Event Service，運作其上的應用程式不一定要先使用 IDL 定義物件界面也可透過該連接物件取得本 Certified Event Service 的相關服務。

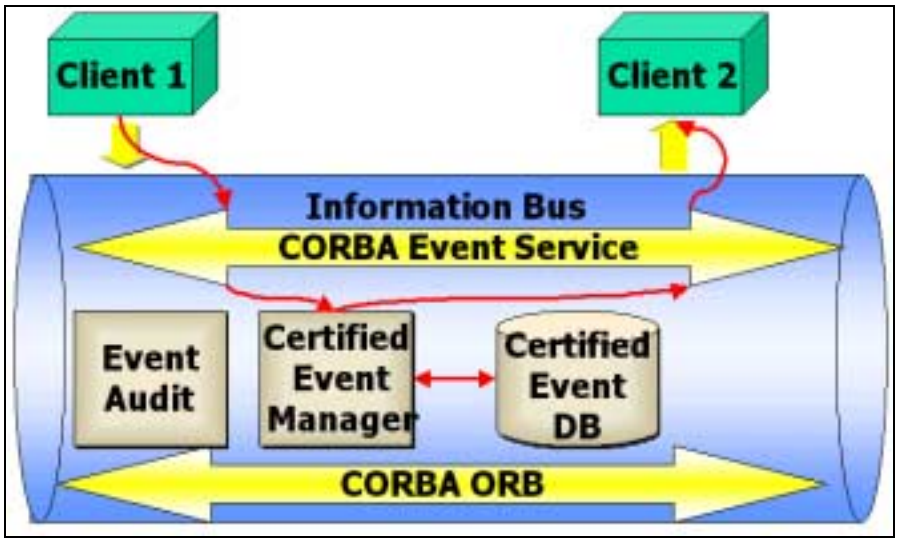

圖 31 Certified Service 中的執行流程

Message Pre-Processor 的執行流程如圖 32 所示，當資料由傳送端要傳送出去之時，先將資料透過各種格式的轉換工具，將資料轉換為 XML (或任何對方可以處理的資料格式)，再連同接收者的相關訊息輸入 Wrapper，Wrapper 會將訊息包裝成可供辨識的 XML

文件，最後再提供給 Event Filter 將訊息推入底層的 Certified Event Service 中。

由於 Certified Event Service 所使用的是廣播的方式傳送資料，接收端必須要透過 Event Filter 將辨別在流動的訊息是否為該客戶端所需要的，之後再由 Wrapper 取出經 XML 包裝的資料內容，該資料內容尚需透過適當的格式的轉換工具將資料內容轉為接收端應用程式所能處理的資料格式，最後再由 Message Dispatcher 來呼叫應用程式處理之。

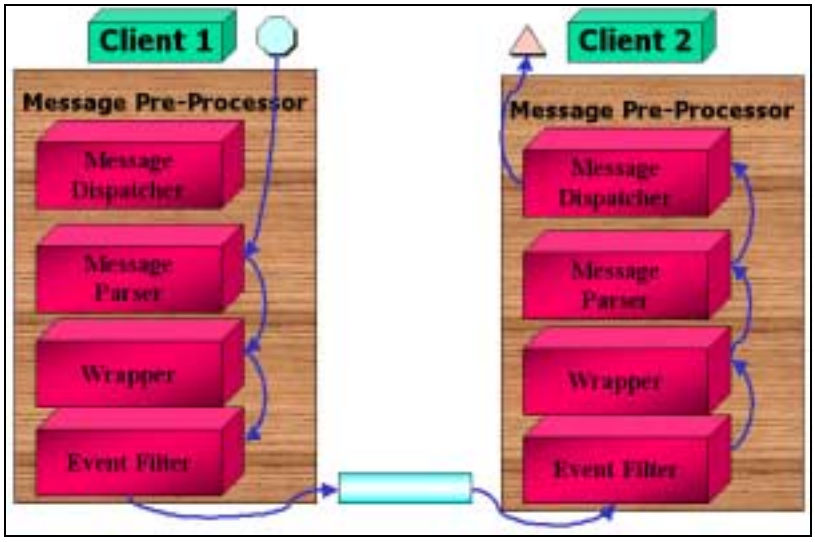

圖 32 Message Pre-Process 執行流程

Message Parser 的執行畫面如圖 33，本例是將 JDBC 資料庫中的資料轉換成 XML 的檔案，以備傳送處理之用。

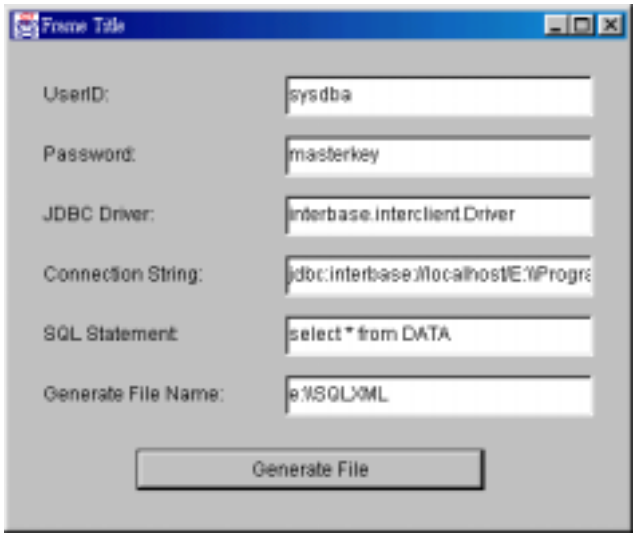

圖 33 Message Parser 執行畫面

Wrapper 的執行畫面如圖 34，本畫面在提供 XML 封裝時所需之訊息之用，在 EVENT_DATA 處可以選擇欲傳送的資料檔，按下 Wrapper 的按妞之後，本程式模組會

將畫面上的訊息及欲傳送的資料檔全部壓成一個 XML String 以供文件傳送之用。

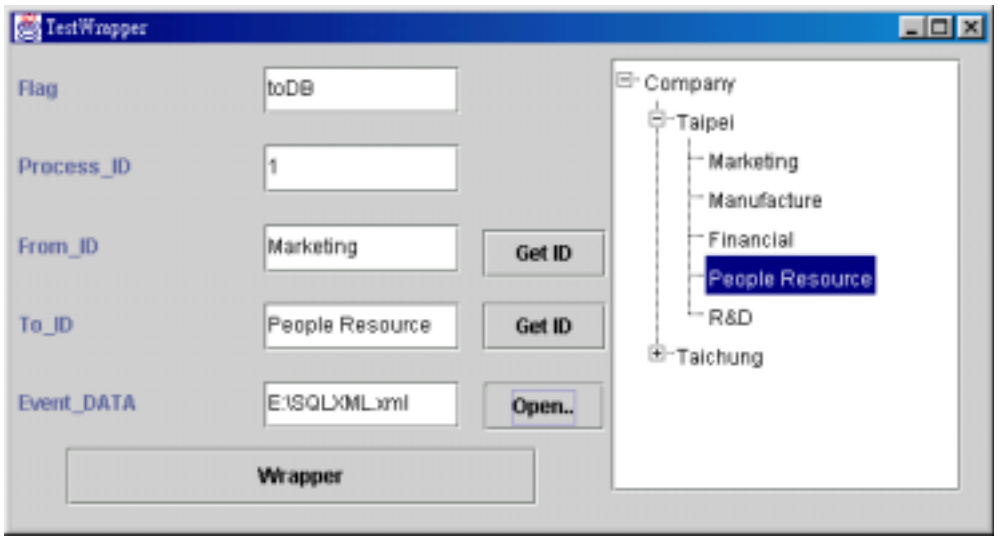

圖 34 Wrapper 執行畫面

經由 Wrapper 包裝後的 XML 文件內容如圖 35 所示，該程式將 XML 文件建成一個 XML Tree 以供使用者瀏覽之用，在 EVENT_DATA 的欄位處即為欲傳送的資料內容。

該程式並呼叫 Event Filter，將該 XML 文件推播到 Certified Event Service 中，其界面如圖 36。

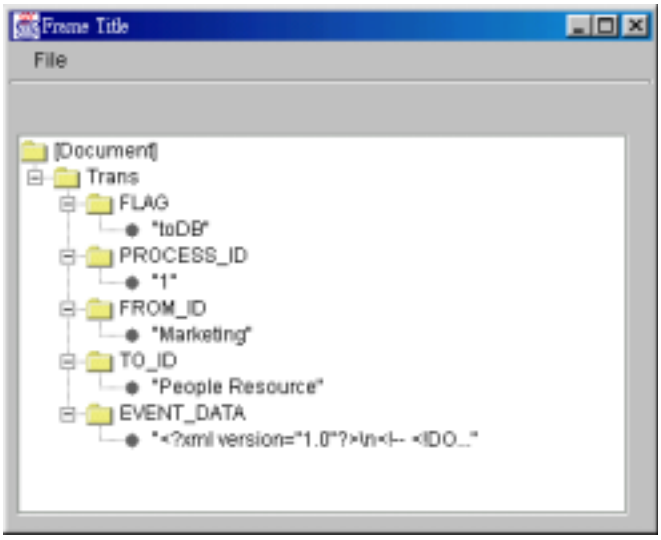

圖 35 XML Viewer 執行畫面

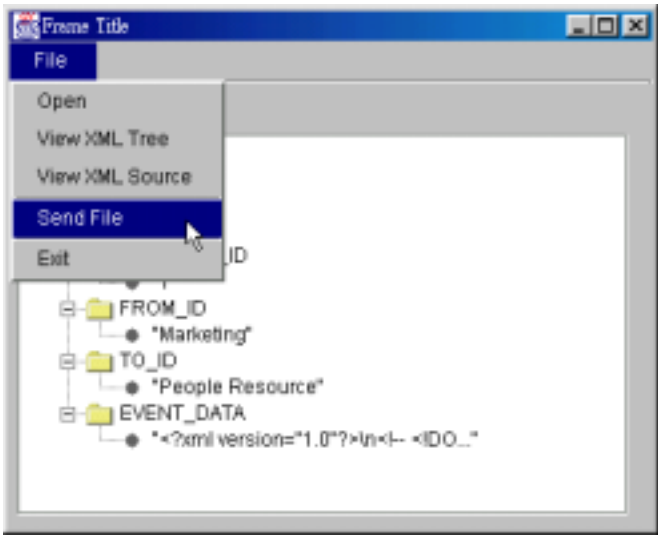

圖 36 進行推播程序

當訊息進入 Certified Event Service 後，會檢察 XML 中的 FLAG 這個欄位，如其內容為 toDB 的話，則將該訊息儲存一份在 Certified Event DB 中，監督程式的界面如圖 37 所示，若 FLAG 欄位內含值並非 toDB，則 Certified Event Service 不做任何的動作。

在資料庫內容的儲存上，除儲存 PROCESS_ID、FROM_ID、TO_ID 之外，還取得現在的 TimeStamp，藉以將傳送的訊息部份儲存在資料庫中，若客戶端沒收到推播出去的資料時可由 Certified Event DB 中取得。

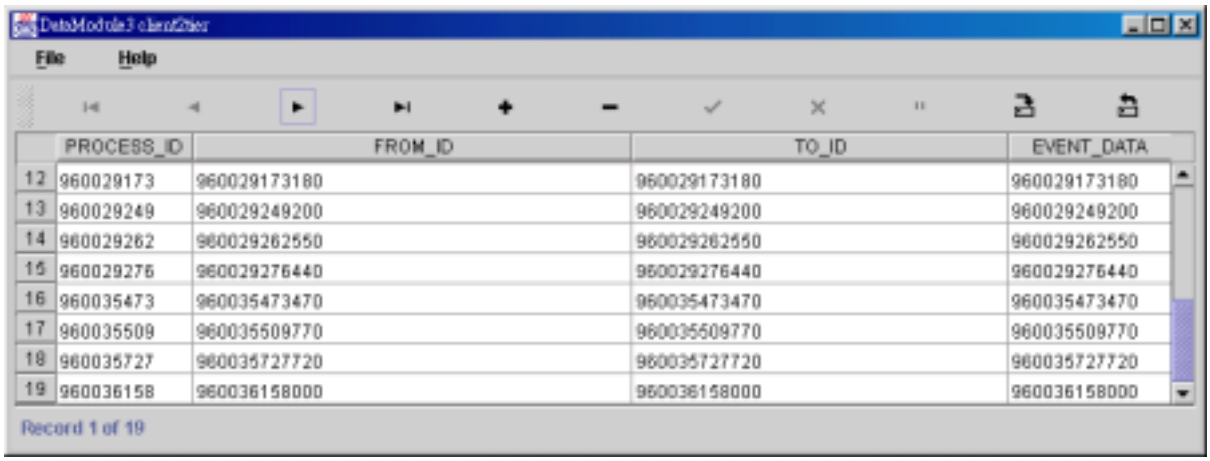

圖 37 Certified Event Service 的監控程式

第四章 結論及未來工作

本論文提出的架構，其貢獻在於：(1). 在系統底層將 CORBA 架構搭配上 CORBA Event Service，並透過良好的物件封裝使其成為一個類似 MOM (Message Oriented Middleware)的溝通管道，並具有可確保訊息正確送達的機制。(2). 在相關程式方面，則建構出針對系統訊息的監督工具。(3). 在訊息處理上，提供將傳送的訊息包裝成 XML 的處理工具。以上所建構出的各個工具，除可在本系統中使用之外，尚可在其他相關領域上使用之。

在本系統的未來發展方面：

(1). 實作完成系統的各個部份：在實作上，目前還有些部份需要改善，包含有工作流程系統的定義工具、系統監控程式、支援 LDAP 協定的組織架構、及發展更多種類的資料格式轉換工具等都需進一步的改善。

(2). WF-XML：由 WfMC 所定義的工作流程描述語言 XML 標籤定義，在今年 (2000 年)的 5 月 8 日發表了最新的版本 (1.0 Final)，但目前系統尚未依此版本修改，故系統須將描述工作流程的描述語言依 WfMC 對 WF-XML 的定義重新實作，以備和其他不同的系統進行資料交換之用。

(3). 執行時期自動定義工作流程：目前發展的工作流程系統，都還是適合靜態流程所用，未來的改善方向，則是朝向在執行時期自動產生最適工作流程方面改善，以使資源的配置達成一個最適的分配。

(4). 加入階層式傳輸架構：目前本系統所提出之架構，在訊息傳遞時需透過 Certified Event Service，將需要整合階層式 Event Service 的架構，以減低 Certified Event Service 的負荷，並減少不必要的網路傳輸資料量。

(5). 延伸工作流程至企業外：考量將原本適用在企業內部的工作流程系統延伸至企業外，以加快企業對企業間的溝通速度。

參考書目

[1] Padmanabh Dabke, “Enterprise Integration via CORBA-Based Information Agents,” IEEE Internet Computing, pp.45-57, Sep. 1999.

[2] Workflow Management Coalition, *Interface 1: Process Definition Interchange Process Model*, Nov. 1998.

[3] Workflow Management Coalition, *Interface 2&3: Workflow Process Application Programming Interface Specification*, July 1998.

[4] Workflow Management Coalition, *Interface 4: Workflow Standard – Interoperability Abstract Specification*, Oct. 1996.

[5] Workflow Management Coalition, *Interface 5: Audit Data Specification*, Sep. 1998.

[6] Workflow Management Coalition, *Workflow Management Coalition Workflow Standard - Interoperability Wf-XML Binding,* May 2000.

[7] Object Management Group, *CORBA: Common Object Request Broker Architecture and Specification Revision 2.0,* July 1995.

[8] Object Management Group, *CORBA Services: Common Object Services Specification*, March 1995.

[9] Object Management Group, *CORBA Facilites: Common Facilities Architecture*, January 1995.

[10]Object Management Group, *Workflow Management Facility*, Submission by jFlow Consortium, Aug. 1997.

[11]Object Management Group, *The Common Object Request Broker (CORBA): Architecture and Specification, v 1.2*, Dec. 1993.

[12]Michael McCaffery and Bill Scott, *The Official VisiBroker for Java Handbook*, SAMS, May 1999.

[13]Doug Pedrick, Jonathan Weedon, Jon Goldberg, and Erik Bleifield, *Programming with VisiBroker,* Addison-Wesley, 1998.

[14]Robert Ofrali and Dan Harkey, *The Essential Distributed Objects Survival Guide* , Wiley, 1996.

[15]Robert Ofali, Dan Harkey, and Jeri Edwards, *The Essential Client/Server Survival Guide Second Edition*, Wiley, 1997.

[16]Robert Ofrali and Dan Harkey, *Client/Server Programming with JAVA and CORBA*, Wiley, 1997.

[17]Robert Ofrali and Dan Harkey, *Instant CORBA* , Wiley, 1997.

[18]Andreas Vogel and Keith Duddy, *Java Programming with CORBA*, Wiley, 1997.

[19]Thomas J. Mowbray and Raphael C. Malveau, *CORBA Design Patterns*, Wiley, 1997.
[20]Jon Siegel , *CORBA Fundamentals and Programming*, Wiley, 1996
[21]Geoff Lewis, Steven Barber, Ellen Siegel, and Geoffrey Lewis, *Programming with Java IDL*, Wiley,1997.
[22]Sean Baker,*CORBA Distributed Objects Using Orbix*, Addison-Wesley, 1997
[23]Web site of Object Management Group, http://www.omg.org/
[24]Web Site of Workflow Management Coalition, http://www.wfmc.org/
[25]Web Site of INPRISE Corporation, http://www.inprise.com/visibroker