國立交通大學

資訊管理研究所

碩士論文

以 Web Services 為基礎且兼具圖形化設計環境與工作流程控管能力的 P2P 運算能力分享環境

A Web Services Based P2P Computing-Power Sharing Architecture with Visual Development Environment and Workflow Control Mechanism

研究生：楊博宇

指導教授：蔡銘箴博士

中華民國九十四年八月

以 Web Services 為基礎且兼具圖形化設計環境與工作流程控管能力的 P2P 運算能力分享環境

A Web Services Based P2P Computing-Power Sharing Architecture with Visual Development Environment and Workflow Control Mechanism

研 究 生： 楊 博 宇　　Student: Po-Yu Yang

指導教授： 蔡 銘 箴　　Advisor: Min-Jen Tsai

A Thesis
Submitted to Institute of Information Management
College of Management
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of
Master of Business Administration
in
Information Management
June 2005
Hsinchu, Taiwan, the Republic of China

中華民國九十四年八月

以 Web Services 為基礎且兼具圖形化設計環境與工作流程控管能力的 P2P 運算能力分享環境

學生：楊博宇　　指導老師：蔡銘箴

國立交通大學

資訊管理研究所

摘要

隨著資訊系統的運算量以及資料處理量的日益增加，現有的資訊系統架構開始不堪負荷。一些企業便開始思考如何在不額外增加軟硬體設備之下維持系統高工作量運作。因此，分享網路資源的 Peer-to-peer (P2P)的概念因而誕生，透過 P2P 把網路上成千上萬台電腦的閒置資源串連起來，來因應資訊系統的運算需求。

本研究針對現有的 P2P 運算分享架構，發現在安全、動機、彈性、相容性以及工作流程控管上的問題。並建置出 Computing Power Services(CPS)架構，利用 Web Services 和 Business Process Execution Language (BPEL)來克服彈性以及工作流程控管能力上的限制；同時縮小應用範圍到 peer 和 peer 之間存在著信賴關係的網路(本文稱信賴網路)之中運行，如此將不用考量安全和動機上的問題。

該架構提供了一個以 Web Services 為基礎，輕量級的 P2P 運算能力分享環境。。同時，藉由 BPEL 的導入，使得整個架構兼具了圖形化的開發與管理環境以及工作流程控管的能力。適用於可批次化的需高運算能力的執行程式，並實行於組織的信賴網路之中，CPS 架構為組織提供了運用組織內現有電腦運算能力的可行方案。

關鍵字：Peer-to-peer、Web Services、BPEL、運算能力分享、分散式運算、分散式系統

A Web Services Based P2P Computing-Power Sharing Architecture with Visual Development Environment and Workflow Control Mechanism

Student：Po-Yu Yang　　Advisor：Min-Jen Tsai

Institute of Information Management
National Chiao Tung University

Abstract

As demands of data processing and computing are increasing, current information system architectures become insufficient. Some organizations try to keep their systems work without cost of purchasing new hardware and software. The peer-to-peer (P2P) model which shares the resource over the network is a solution proposed in this paper.

In addition, this paper discusses some problems about security, motivation, flexibility, compatibility and workflow management for the traditional P2P power sharing models. Thus, we proposed a new computing architecture: Computing Power Services (CPS) that aims to solve the problems discovered. It utilizes Web Services and Business Process Execution Language (BPEL) to overcome the shortcomings about flexibility and workflow management. CPS is limited to a trust network where peer and peer trust each other. Thus, the concerns about security and motivation are negated.

CPS is a lightweight Web-Services-based P2P power sharing environment, and suitable for executing computing works which are able to run in batch in a trusty network. The architecture relies on BPEL which provides a visualized development environment and workflow control management.

Key words：Peer-to-peer, Web Services, BPEL, computing power sharing, distributing computing, distributing systems

致謝

碩士是學習生涯中一個培養獨立研究的階段，而論文完稿正是整個過程的開花結果。整個過程之中，除了「知」有所得外，「道」也得到一定程度指教。這些成就，全賴各位師長的教誨。首先，要感謝蔡銘箴老師，在碩士的兩年裡，能在課業上給予指導；另外，學長王振生老師則是我資訊技術上的導師，在論文系統的建置上幫了我很大的忙；還有同窗好友承龍、冠輝以及學弟錡樂、聖紘、紹榮和國鼎，有了他們的幫助，論文方能在順利、愉悅之中進行。其他，也感謝幫我做實驗的大學同學們，讓實驗得以完成。論文的完成，不完全是我的功勞，也賴於上述這些貴仁的相助，再一次由衷地感謝他們。

目錄

圖目錄

表目錄

1. 簡介

1.1. 研究動機與目的

早期的電腦系統，大型主機(mainframe)擔負著運算一切事務的角色，也往往一個資訊系統的瓶頸和主機的運算能力有著密不可分的關係。隨著資料處理量的日益增加，企業必須投入更多的金錢在昇級主機之中，以維持主機以及系統運作的效率。為了解決這樣的問題，主從式(client/server)架構誕生了。這樣的一個架構把部分工作量從單一主機分擔到 client 之中。也藉由工作量的分擔，企業可以避免在主機上的軟硬體支出，同時亦能維持住系統運作的效率。

主從式架構在廣受各方接納使用之後獲得了成功。只是，隨著競爭日益激烈，愈來愈多的資料與運算量讓許多企業再一次面臨著效率與金錢的取捨。能否在不增加額外的設備投資情況下，利用現有的資源來達成維持系統效率的目的？於是開始有人趨思考如何利用企業中閒置的電腦與剩餘儲存空間。就這樣新的 Peer-to-peer 架構產生了，透過網路，Peer-to-peer 架構把散佈各地的閒置資源給連接起來。最大的好處在於對於每一個參與者都能分享到其他的資源而達成經濟地提升效率之目的，目前也有一些 P2P 的架構在成功的運作。

既然 P2P 是講求分享與互惠的一種運作模式，當然一些電腦中常常閒置的資源自然容易被提起。諸如：檔案系統、網路頻寬、運算能力等等，也都有一些實作的架構運行著。但在分享運算能力的領域之中，現存架構中存在著一些問題，例如：無法自訂運算任務、工作流程控管問題等等。

因此，本研究針對這些問題，提出了一個以 Web-services 為基礎，並融合了 BPEL 作為實驗流程制定語言的輕量級 P2P 環境。適用於中小型組織的信任網路之中，具備可圖形化方式設計運算任務的彈性與工作流程控管的能力；同時繼承了 Xml Web Services 開放性標準以及非緊密結合，可以用來分派可批次化的高運

算能力依賴的工作，給信任網路之中閒置的資源代為運算。對於施行的組織可以達到經濟地提升運作效率之目的。

1.2. 論文架構

本論文架構如圖 1-1所示，共有六個章節。第一章是簡介，分別說明本研究的研究動機與目的以及論文的撰寫架構。第二章是文獻探討，針對 P2P架構、Web Services、Web Services Composition和非同步Web services回報的現狀與研究做文獻分析與探討的工作。第三章將對於目前幾個基於分散式運算能力分享的P2P架構作問題分析，並提出解決之道。第四章提出解決問題的架構，也是本研究的核心--Computing Power Services架構，簡稱為CPS。該章節會分別從流程運作面與系統面切入，對CPS做詳盡介紹。第五章則是系統建置與成果，先對實際的CPS系統建置作說明，包括了環境需求與實際畫面可供參考。之後針對CPS架構提出的一些特點，做實際的實驗來予以驗證。最後則在第六章提出結論與未來可行之研究方向。

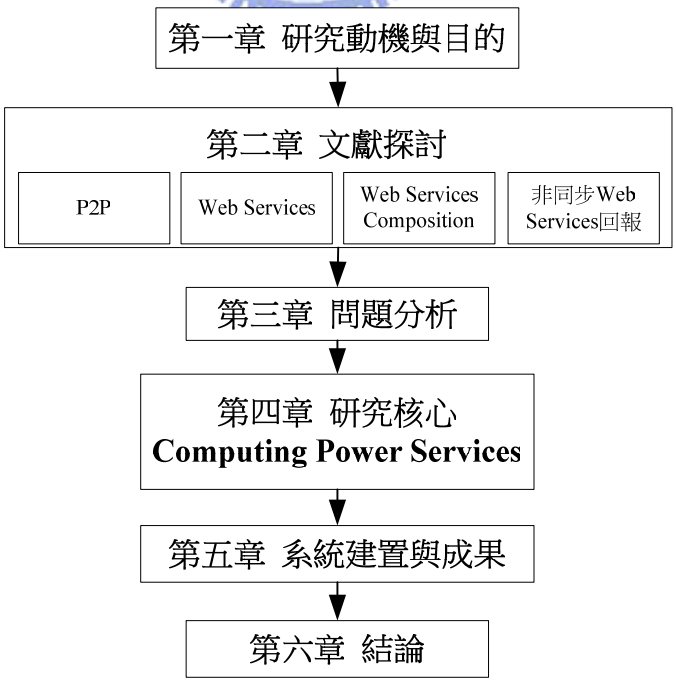


圖 1-1　論文流程圖

2. 文獻探討

2.1. Peer-to-Peer model

Peer-to-peer 這個架構，簡稱為 P2P，根據 Dejan S. Milojicic 等人在文章「Peer-to-Peer computing」 [1]裡把它定義為，利用分散資源的一個系統或應用程式，採用非集中的方式運行一些特定工作。利用的資源可以是電腦運算能力、儲存空間、檔案分享或網路頻寬。而 Peer 這個字所代表的意含，是指每個平等一致的點。因此在 Peer-to-peer 的架構裡，每一個點(Peer)之間即使是存在著資源互享的關係，但彼此之間的溝通是直接而對等的，不存在任何的主從關係或是依賴某特定者的依賴性。

在 Alfred W. Loo 的「The Future of Peer-to-peer Computing」[2]提到：一般來說，Peer-to-peer 的系統會有三個特色：

- 在架構之中的每部電腦，既是 server 的角色，也可以是 client 的角色。
- 參與者可以藉由網路，直接地共同享用彼此的資源。
- 參與者建立在互助互惠的環境之下。

使用 Peer-to peer 這樣的架構，可以把現有的資源結合起來，形成更具能力規模的資源；對於採行的組織來說， Peer-to-peer 系統的利用仍可整合組織現有的資源，來滿足資源需求漸增的環境，達到降低設備升級成本的支出。無論是多老舊的電腦、有限的儲存空間、不大的網路頻寬與散布網路各處的檔案，資源分享以及積少成多聚集成大規模的能力，可以說是 P2P 架構的中心思想之一。在 Alfred W. Loo 的「The Future of Peer-to-peer Computing」[2]中把現行架構分為檔案分享和分散式運算能力分享。

2.1.1. 檔案分享架構

檔案分享的 P2P 架構是目前相當熱門的一個領域，無論是在檔案內容本身還是儲存空間的分享，都是相當盛行的一種架構。建立在資源共享，互助互惠上的多媒體檔案分享，使用者可以把自己電腦的某些區域設為檔案交換區，分享檔案給其他使用者。這種方式除了檔案內容透過電腦網路頻寬共享之外，多個點同時存在著副本也增加了檔案可靠性，和傳統方式相比較明顯得較為優良。

這一類的架構之中，Napster 是一個重要的代表。Napster 是一個音樂共享環境，使用者可以透過連結 Napster 中央電腦維護的一份資源清單中，來搜尋想要的 mp3 在哪個其他使用者的硬碟之中並可以下載之。

在整個架構中，Napster 的中央電腦扮演著一個重要的角色。他除了維護之前曾提及的檔案資源和會員對應的清單之外，同時還會監控會員的上線狀態，並在會員登入時，根據會員所分享的檔案，來更新中央電腦中的資源清單。一但，有會員送出尋找某音樂檔案的需求時，中央電腦就會根據清單尋找需求方所能連接並擁有該特定檔案的電腦，再由需求方自行和擁有者接洽檔案交換事宜。

Napster 和其他相似的架構如：E-Donkey、eMule、Bittorrent 等等，儘管有些架構並不需要中央電腦處理清單，但是建立在資源共享、互助互利上，他們都獲得了很大的成功並日益風行。就拿 Napster 為例，超過 36,000,000 人的加入，同時也帶動了其他類似架構的出現與成長。儘管，新的檔案分享 P2P 架構不在受限於 mp3 這種檔案類型，但是他們最大的問題還是在法律上面。Napster 就曾因為被控侵權敗訴而被迫關閉，儘管已經又重新運作，但是改為收費制度之後，盛況似乎大不如前。[2]

2.1.2. 運算分享架構

另一種架構是建立在架構網路內運算能力分享的基礎上。此種架構試圖把網路上的電腦運算能力給集合起來，以應付高度的運算需求。因此，以往需要幾個禮拜甚至幾個月的運算工作，在不需多餘的設備升級之下，可以縮減所需的執行時間。在 1999 年一月時，這種分散式運算系統就成功在一天之內破解了 RSA 的密碼加密。[1]

分散式運算架構主要被應用在大規模的科學計算實驗之中，其中計算癌症藥物的 UD 便是一個例子。UD 的出現在於 2001 年 4 月一個代號為 LigandFit 的克服癌症計劃由 Intel、牛津大學、國家癌症研究基金會(NFCR)和 United Device 共同發布。它是一個透過網路結合網路上電腦運算能力的一個 P2P 架構。這個架構當時主要的目的是進行一項癌症研究計畫，嘗試去做一些基因上的計算與研究，找出能治療癌症的藥物。

這一個架構的運作相當簡單，首先參與者必須先下載一個程式，並安裝它。這個參與者程式會在參與者電腦空閒時，藉由一個公正第三方尋找實驗專案發行者，並向它下載相關程式與資料進行運算。一但運算完畢，再回傳結果給專案發行者，並再向第三方詢問新的實驗發行者，這樣一直循環下去。LigandFit計畫在這個簡稱為UD的P2P運算環境獲得很大的成功。目前投入UD的電腦總共有 3,063,461 台機器，提供了超過了 401,875 年的運算時間，平均一天貢獻超過 247 年的運算時間(以上數據來源於http://www.grid.org/stats/)。

有鑑於該專案的成功，新的蛋白質基因研究(代號為 Rosetta)，由 ISB(Institute for Systems Biology)、華盛頓大學和 IBM 主導的研究計畫也在 2004 年加入 UD 之中進行實驗。該專案嘗試分析人類基因中的各種蛋白質的特性，看看是否能在疾病預防和治療上有所貢獻。這一類類型的架構，比較有名的還有偵測外星人的 Seti@Home、計算相對論之中脈衝星的 Einstein@Home，也和 UD 一樣有許多的

參與者支持著。

這一類的架構有著一些限制。首先，目標工作必須能被切開成數個小部份，以方便分配給每個參與者運算；另外，由於分散式運算架構大都有一個中央伺服器來控制參與電腦的資源分配；而每個參與者(Peer)由於只負責運算工作與回報實驗，彼此之間並不會有直接的通訊關係存在。因此，有些觀點甚至認為分享運算架構並不是很純粹的 P2P 架構。

2.2. XML Web Services

2.2.1. Web Services 簡介

Web Services(網路服務)，可以說是一種分散式系統的方式之一，可以把它當作就是存在於網際網路上面的一種應用程式。不同於本機端開發的應用程式，每個功能模組都可以自行定義及開發，透過Web Services的機制，任何人可以在網際網路上面尋找自己想要的應用程式模組，將其納入所要開發的應用程式中，也可以將自行開發完成的應用程式模組，經過註冊後提供給網際網路上使用者使用。圖 2-1所示為Web Services的基本架構概念圖，從此圖中可以更清楚的了解Web Services的運作機制是由三個單元組成，分別是服務提供者（Service Provider）、服務需求者（Service Requester）及服務註冊機構（Service Registry），而三者之間的關係分別存在有發行（Publish）、尋找（Find）及聯結（Bind）。

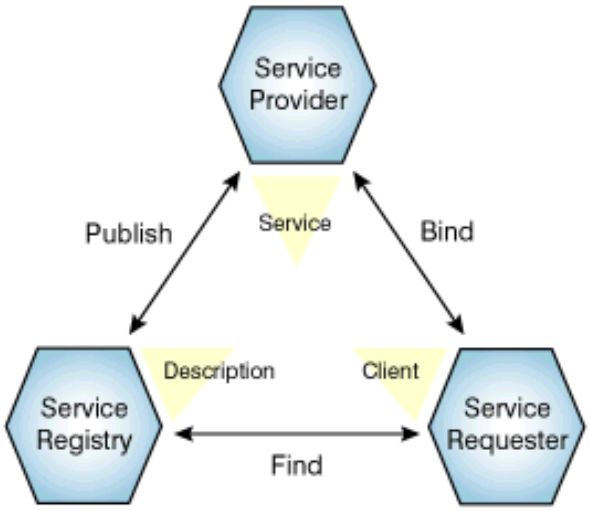


圖 2-1　Web Services 基本觀念圖[4]

根據喻瀚寬等人的歸納，Web Services 運作行為依順序有下列幾個步驟 [3]：

- 描述：Web Services提供的功能以及連結方式，需要一種標準的描述語言，來讓Web Services需求者知道。WSDL(Web Services Description Language)正是W3C定義出來的標準描述語言。一個以XML為基礎的語言，定義了Web Services訊息交換的各式與類型、連接位址與方式等等。WSDL為Web Services提供了服務內容的描述，也是一個辨別Web Services身分的一個參考。

- 發佈：在廣大的網路之中，Web Services提供的服務想要被需求者找到，就得透過服務發佈的過程。UDDI（Universal Description, Discovery and Integration）的功能正是如此，如同電話簿黃頁一般，給予每個Web Services發佈找尋自己的相關資訊。

- 尋找：Web Services 如何要在網路中找到提供服務的 Web Services 就需要透過尋找的機制。透過 Query 向 UDDI 詢問有登錄的服務是否合用，來尋找適用者。

- 組裝：根據取得之 Web Service 描述資訊(WSDL)，建構一個連結到該 Web

Service 的代理程式。一般程式語言會產生一個 Proxy Class 來代與 Web Services 溝通。Proxy Class 會控制該有的輸入輸出格式，以避免不一致的問題產生。

- 呼叫：透過 Internet 加上 SOAP(Simple Object Access Protocol)傳遞適當訊息參數，呼叫該 Web Service。一般常見的模式有文件模式或 RPC 模式兩種。
- 整合：利用上述機制，整合其他 Web Services 以建構出自己的應用程式或是新的 Web Service。

根據上述的六個網路服務的運作行為，圖 2-2呈現了更清楚的運作行為與相關協定的關係。UDDI（Universal Description, Discovery and Integration）、WSDL(Web Services Description Language)與SOAP(Simple Object Access Protocol)代表的是整個Web Services架構的核心，分別涵蓋了服務發佈與發現、服務內容描述以及資料傳送協定。目前也廣被各方所接受，成為主要的標準。

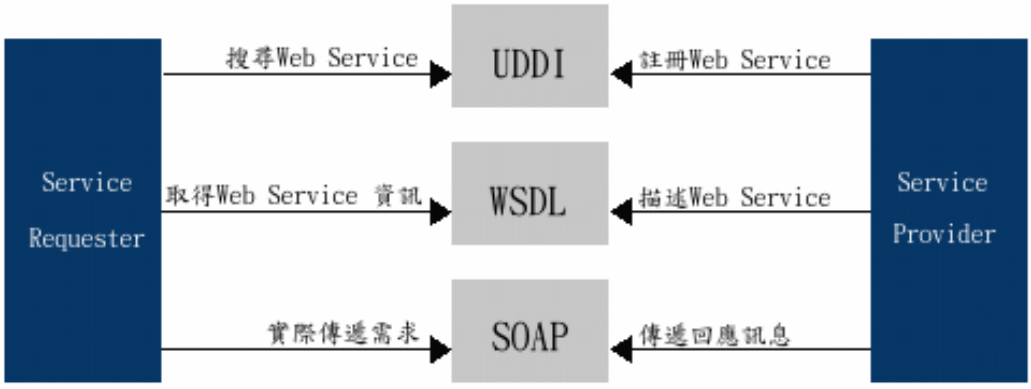


圖 2-2　網路服務簡單輪廓圖 [3]

2.2.2. Web Services 的特性

從上述的網路服務基本架構，可以看出其所不同於其他分散式架構的優勢，

本研究針對相關技術文件整理後，提出兩點來說明：

A. 非緊密連結 (低耦合性)

Web Services 的第一個特性是非緊密結合，亦稱之為低耦合性。過去，分散應用程式邏輯需要一個分散物件模組，如 DCOM、CORBA、RMI 等。在進行企業內系統整合或更新時，利用這樣的架構，開發人員除了可以保有舊有的應用程式之外，並能提供遠端系統的服務的可能。然而上述這些架構最大問題是，當它們延伸至網際網路上時，通常需要相當的開發成本及更進一步的溝通技術；由於他們需要緊密的與其他應用程式連結，也就是說在 client 與 server 之間通常需要特定的溝通管道，而且大多是非開放式架構的標準，這通常需要雙方都付出同樣的代價來實現分散式運算的架構。這樣的架構往往代表著連結是非常脆弱的，假使當連結的一方突然改變時，另外一方則會中斷。例如，當主機的應用程式介面改變時，使用者端就會被迫中斷連線了。

需要緊密相連的架構並沒有什麼本質上的錯誤，一直以來許多應用程式都是採用此模組開發。然而，這種模組並非長久之計。當公司之間相互購併或當提供資訊服務的廠商結束營業的時候，公司就很難保證能維持單一完整的架構，也因為無法預知它們採用什麼作業系統、物件模組或程式語言，所以很難保證其所需要的服務在遠方的另一端是否仍能保有的完整架構。

相反地，網路服務(Web Services) 的架構則顯得較有彈性，這意味著隨時可以更改任何一端的連結，但應用程式仍可以繼續運作。就技術上而言，網路服務透過以訊息為導向（Message-oriented）的技術，使用 HTTP 或 SMTP 的通訊協定達到高效能的同步或非同步的呼叫。

另一方面，訊息導向的技術建構出網路訊息交換架構。傳輸訊息者只需要假設被發出的訊息內容能被接收者了解就可以了，完全不用擔心訊息接收時會發生什麼狀況，或者是訊息傳輸中途會有什麼突發狀況。因此，建立獨立的網路訊息

交換架構的優點是顯而易見的，例如，訊息接收者可以在任何時間變動而完全不會影響到信息發送者，只要它能了解相同的訊息內容。接收的一方隨時可以進行更新而不需要終止現有的應用程式。傳輸者甚至不需要任何的特殊的軟體來與接收者溝通，只要他們維持在相同的訊息結構裡，接收的一方就能夠回應。

B. 開放式標準

一個架構的能否被廣泛的使用端賴於其架構所建構底層標準是否具有開放性的原則，也就是是否具有彈性跟延伸性。XML 的資料格式是使網路服務能跨越網域及不同異質平台架構的關鍵，網路服務需要 XML 的原因有三項：基本線上傳輸格式(basic wire format)、服務描述(service description)、及服務搜尋(service discovery)。而這三個不同的需求也分別被 SOAP、UDDI 和 WSDL 這三個以 XML 為基礎的標準所滿足。

因此，建築在 XML 之上的 Web Services 將可以繼承 XML 跨平台的特性。公開的開放式標準與其強大的功能與展望，吸引了許多大廠的目光。包括：Microsoft、Sun、Borland 等等，也紛紛把其公司的產品加入對 Web Services 的支援。Java、.Net、C++、Python、Perl、Delphi 等等，愈來愈多語言也漸漸支持 Web Services 的標準，一個跨語言、跨平台的 SOA 分散式的解決方案，蓬勃地發展著。

2.3. Web Services Composition

2.3.1. Web Services Composition 簡介

Web Services 是一種分散式系統的解決方案，透過 WSDL、UDDI、SOAP 等等的標準，來提供一個 Service Oriented Architecture(SOA)的環境。但是儘管使用者可以透過 UDDI 查詢所需的服務在何處；透過 WSDL 得知服務的性質與輸入、輸出；再利用 SAOP 來與 Web Services 溝通。但是，在一些 Web Services 應用的範疇裡，無論是 EAI(Enterprise Application Integration)、或是電子商務等等，基本的 Web Services 應用是不足的。許多的流程是需要一連串的 Web Service 去達成的，而且往往需要依照當時的情況，動態的去決定之後的流程。這就是 Web Services Composition 出現的目的。複合的 Web Services 可以達成單一 Web Services 所不能達到的許多功能；流程化、SOA 的觀念，也使的組合之後的 Web Services 在概念上能變得更加抽象，和人的思維更加得接近，也更加容易應用。

從 Biplav Srivastava 和 Jana Koehler 的“Web Service Composition - Current Solutions and Open Problems”[5]一文中可以歸納出：目前，Web Services Composition 的發展主要著重在兩大路線。一種是從學術的角度切入，加入了語意網路(Semantic)以及 A.I.人工智慧的概念，為 Web Services Composition 提供了更多的可能與可證明性，其中又以 OWL-S 最為著名；另一種是從業界的角度切入，導入了流程(flow)的概念，視 Web Services Composition 為一個資訊流的控管，為其中加入了流程控管的元素，該領域目前以 BPEL(Business Process Execution Language)為代表。

其中，BPEL 由於易建置、相容於 WSDL，加上有大廠推動，目前在 Web Services Composition 領域裡最為人所接受。

2.3.2. BPEL(Business Process Execution Language)

Business Process Execution Language for Web Service 簡稱為 BPEL4WS 或 BPEL 是 Web Service Composition 目前最普及的一種標準及解決方案。1.1 版在 2003 年五月，由 Microsoft、IBM、Siebel System、BEA 和 SAP 共同訂立。

BPEL 建立在 WSDL 之上，和 WSDL 一樣是 XML-Based 的一種語言。該標準，是 Microsoft 和 IBM 分別把自家的 XLANG 和 WSFL 整合之後的成品。結合了 XLANG 和 WSFL 的特色，BPEL 一樣具有可圖型化和 XML 巢狀化表示的特性，工作流程控管也能處理平行、序列、同步、非同步等等的流程，另外也有極佳的例外處理機制。

BPEL 主要支援兩種程序型態：可執行程序(executable process)和抽象程序(abstract process)。可執行程序主要指的是在流程的溝通互動之中，參與者的行為。而抽象程序，又稱之為商業協定(Business Protocol)，則定義了每個角色之間的訊息交換規定。商業協定不能執行，也不能傳達程序流程的內部細節。BPEL 還包括了兩種行為：基本行為(basic activity)和架構行為(structured activity)。前者控制一些和程序本身之外的一些外部物件溝通的命令，例如：invoke、reply 等等；後者則管理整個程序的流程和程序中引用的 Web Services。[6]

BPEL 同時也具有一套控制交易(transaction)和例外處理的強大機制。建築在 WS-Coordination 和 WS-Transaction 這些標準之上，可以和舊標準相容。也由於 BPEL 和大多數的舊標準相容；概念上也採用比較直觀、易瞭解的工作流程的方式表示；再加上該標準由大廠協同制定(Microsoft、IBM、BEA 等等)，相較於其他 Web Services Composition 的標準，BPEL 有著比較高的接受度。資訊大廠，如：IBM、Oracle 等，也都開發出相關工具與以支援。

2.4. 非同步 Web Services 呼叫

Web Services 的呼叫在某些情況之下，服務的回覆，並不會在一開始的服務需求送出之後，立即的傳送回來。這種模式稱為非同步 Web Services 呼叫，在現實的情況中，是很常見的。例如：需要多個部門查核的請購單，少則幾小時，多到一兩天的流程；或者是網路傳輸及電腦運算造成的可能延遲。這些情況，正統的同步回報會使得電腦停在該流程點的延遲，對於電腦的資源或使用者的使用經驗，都是一個負面印象。

因此，真實的環境裡需要非同步的Web Services呼叫的存在。但是在現行標準之下，非同步Web Services呼叫並無正式的規範。Holt Adams在“Asynchronous operations and Web services: A primer on asynchronous transactions”[7]一文中把典型的非同步web Services呼叫過程，整理為下列四步：

- Client 產生並傳送需求訊息
- Server(服務提供者)收到需求訊息
- 服務提供者回傳運作結果
- Client 收到回傳的運作結果

針對這樣地延遲資訊交換，一些大廠便對支援 Web Services 的語言，加入了一個「關聯(correlation)」的觀念，來控制呼叫 Web Services 的 client 和 server 的程序(process)或執行緒(thread)能在呼叫送出一陣時間後，依舊把回覆傳送給另一方。

其中，如何去確認服務呼叫的雙方是非同步 Web Services 呼叫的一個癥結點所在。對此，Holt Adams 在文中[7]列出了非同步呼叫建置時，所要注意的一些要項：

- 定義清楚訊息交換的關聯者與機制。
- 確定回傳地址(Reply-to address)真的是回覆訊息期望回報的地方；而目的位址真的是服務提供者之所在。
- 服務提供者回覆訊息的產生，可視為因需求所引起的一個行為。
- 非同步回覆由 client 接收。
- 因該需求而產生的回覆關聯建立在一樣的 client 和服務提供者上。

目前，較為常見的標準有 WS-address[8]和 BPEL 中定義的 Correlation Set[6]。許多 application server 利用它們來提供非同步 Web Services 呼叫的機制，但在相容性上依舊存在著一些問題。

3. 問題分析

3.1. 分散式運算分享架構運行的可能問題

分散式運算分享架構，透過網路可以把電腦的運算能力聚集起來。應用在一些運算能力依賴的工作上，例如：大規模的科學運算。如此，可以有效的提升運算效率，提高了運算複雜度高的程式運作的可能。但是對於一些的較小規模的組織或企業想要推行類似的事，卻似乎存在著一些問題，使得在施行上難以成功。茲分別討論如下：

3.1.1. 安全性

在運算分享的架構之下，參與者的電腦會從實驗擁有者端下載檔案與資料並執行之。以 UD 來看的話，這建立在參與者對於該組織的信賴。倘若遇到不懷好意的實驗擁有者，利用這種模式針對作業系統的弱點加以攻擊。無論是從參與者電腦增刪檔案；抑或擷取參與者私人資料；還是掌控參與者電腦控制實行跳板攻擊等等，都會對參與者的個人隱私有很大的衝擊，把別人貢獻給自己閒置運算時間的熱心轉化成對他人的傷害。這些問題，都是在安全性上面，影響其他潛在參與者加入的阻力。當然如果該組織的公信力有如牛津大學、INTEL 一般，就可給予參與者在安全性上一定的信心。

3.1.2. 動機

在運算分享的 P2P 架構中，參與者並沒有獲取任何實體的回報，往往只是參與者認為這對世界有所貢獻，願意貢獻一己之力。在很多企業之中，在下午五點到隔天早上九點之間，有著許許多多的電腦是閒置的，那為何不拿他們來做些運算？有些學者對此提出了企業的「排他性」，認為自己的財產沒必要做一些對

自己沒幫助的事情。同樣的，對一般使用者來說，倘若運算分享的對象帶有一些商業目的，這確實會使他們卻步。以 UD 來說，即使克服癌症計畫的贊助者中不乏像 INTEL 這樣的企業，但是研究成果並不屬於任何藥商，立場中立也穩定了參與者加入的信心。

3.1.3. 彈性

在 UD 環境中，使用者和實驗之間的溝通是要透過下載安裝一特定程式進行。但是，一但有新類型的實驗加入，該程式就必須被升級以配合之。但是，沒有一個有效的辦法來讓大家同時更新，程式彼此之間的耦合性造就了這個問題。另一方面，參與者也沒有辦法提出實驗來參與運算。一些網格(grid)運算有提供這類的服務，但是大多使用命令模式的方式而不是一個較簡單甚至圖形化的方式產生，也是另一個問題點所在。

3.1.4. 相容性

UD 的參與者程式目前只能適用在 NT 系列的作業系統上，對於其他的作業系統，如：Mac、Unix、Linux 就無法加入。儘管 NT 系列的作業系統相較於其他作業系統佔了大多數，但是還是不能輕忽使用其他作業的潛在參與者的能力。不同於 UD，另一個分享運算的架構 seti@home 就有提供其他平台的使用者程式。它的方法是對於不同的作業系統提供相對應的程式，但是這麼做也使他日後若面臨升級時所需付出的成本相對增加。

3.1.5. 工作流程控管

大多數的 Grid 或 P2P 分散式分享運算架構大多著重在效能改善上，工作負擔的合理分配、如何穩定地來容許數以萬計的電腦加入運算，便成了許多基於分散式運算能力分享的中介軟體(middleware)所著重的部份。但是，一旦發生所要運算的工作存在著相依關係，或者是必須要有平行運算的狀態發生時，許多架構

並不支援或者說並不是原生的支援這些工作流程控管機制。

3.2.可行的解決方案

針對上述提出的問題，一個分享運算的架構之中，在滿足一切需求與必要條件之餘，是否有一些技術或方法可以用來改善這些問題。

3.2.1. 非同步 Web Services

非同步 Web Services 的導入，兼顧開放性與可延展性，可以當作 P2P 架構中解決彈性與相容性的解決方案。Web Services 是一種分散式 SOA(Services Oriented Architecture)的一種解決方案。採用 Web Services 建置的系統，將可以繼承 Web Services 低耦合和開放式標準的特性。因此，對於系統的彈性來說，就算遭遇改版或更動程式邏輯，只要確定彼此交換的訊息能夠被系統解析，系統依然可以正常運作。

此外，由於 Web Services 的開放式標準以及眾多廠商的支援，打破了異質系統之間的限制。採用 Web Services 建置分散式運算分享系統將可以提高整個架構版本更新的彈性。

但是，由於 Web Services 以訊息為主的架構，Services 之間的互動建立在訊息交換之上。因此，一但 Services 之間的溝通發生，一般同步的 Web Services 會一直等待對方回報為止。但是，分散式運算環境，工作運算時間長短不定，快則一兩分種，慢有一兩天的可能。過長的時間會對 Web Services 造成延遲例外、而 Service 一直停在某處等待回應也不是很合理。

因此，採用非同步 Web Services 是一個解決方式。除了延遲例外發生的避免之外，等工作運算完畢再回覆的方式也顯得比較合理。非同步 Web Services 的應用是採用 Web Services 建置分散式運算分享架構的一個關鍵。

3.2.2. BPEL

工作流程控管以及彈性問題，可以藉由 BPEL 的導入而解決。BPEL 是目前在處理 Web Services Composition 最廣被接納的一種標準。綜合了可圖形化、XML 巢狀式結構、工作流程控管與強大的交易處理以及例外處理的特性，並完全和 Web Services 現行標準相容。目前許多廠商也紛紛推出圖形化 BPEL 的設計與管理介面，也有助於該標準的發展。

若是從分散式運算分享的運作實驗流程來看，BPEL 的導入可以提供實驗者一個工作流程控管以及圖形化、視覺化的開發環境。一些工具也可以幫助實驗設計者透過圖形化的開發環境，規劃出許多複雜的實驗來透過分享運算架構進行協同運算。儘管流程中有平行處理或相依關係，BPEL 的流程控管機制也可以予以處理。

因此，採用 BPEL 將能改善分散式運算架構實驗流程設計上的彈性；另外，也賦予了它工作流程管理的能力。

3.2.3. 信任網路內施行

現行的一些分散式運算分享架構在處理安全性和動機上往往需要建立在信賴和名聲上。因此，主要推動或施行的組織大多是知名大廠或名聲顯赫的學術單位。但對於中小企業、或是一些營利組織來說，他們該如何來處理一些高運算能力依賴的工作呢？

因此，不妨反過來想。把組織內電腦的閒暇時間，拿來處理組織內想要運算的工作。從動機上考量，由於組織內的電腦本來就是歸組織所有，因此儘管運算的工作具營利目的，在動機上也站得住腳。同樣的問題拿到安全性考量上，既然運算工作都是在自家電腦上運行，當然沒有理由去運行惡意程式。縮小整個分享運算架構的網路環境，建立一個輕量級的運算分享架構，讓它在一個實體上已建

立信賴關係的網路中執行，可以克服安全性和動機的問題。

4. Computing Power Services 運作流程及系統架構

綜合上述的改進方案，本研究提出了 Computing Power Services 架構(CPS)，一個以 Web Services 基礎的 P2P 運算分享架構，並利用 BPEL 的圖形化設計工具，為實驗設計者提供了一個具圖型化以及工作流程控管能力的實驗設計平台。並針對中小企業與較小規模的組織，提供一個輕量級的分散式運算分享環境，供其在組織內的信賴網路中運行，以期充分運用組織中閒置的電腦運算能力，來運算高重複性、可批次化切開的高度運算依賴的工作。

在本章的論述中，第一、第二小節，會從流程的觀點來介紹 CPS 架構的運作。透過其運作流程的介紹，來說明 CPS 如何從 BPEL 的流程與運算能力分享的中介層溝通；以及需要那些角色的參與，進而完成網路協同運算的目的。第三小節開始，則是從技術及系統架構的角度，來探討 CPS 的系統架構以及建置環境需求。

4.1. Web Services 方式的運算分享模式

一個分散式運算分享架構，最基本的功能就是把工作分派出去。因此，CPS 的基本運作架構中，在實驗擁有者的實驗發佈後，它將會需要去找尋其他想要分享其閒置運算能力的機器，來幫忙提升實驗運算的能力。比較直觀的想法是以運算能力需求者為出發點，自發地去尋找閒置的電腦，並把工作佇列裡的工作分派給這些閒置電腦。但是由於 CPS 是一個以 Web Services 為基礎所建置的環境；Web Services 是屬於以訊息為導向的架構，因此若想要由需求者主動送訊息出去，將變成每個使用者都需要安裝一個 Web Service 在電腦上。

換言之，若想要使用這種每個參與者都是一個 Web Service 的方式建置，每一個加入運算的電腦上都需要安裝一個 application server 作為 Web Services 運行的環境。如此，對於運算參與者來說，過於複雜龐大的環境需求會遏止參加者的

意願，也會增加參加電腦的負擔與風險。因此，在合理化考量之後，參與者端應該只是一個簡單的應用程式，才是較為可行的方法。

針對於此，CPS架構的基本運作模式將有所改變，如圖 4-1所示。首先，實驗擁有者在實驗設計完畢之後，就會在運算能力需求者端生成一串工作佇列。而透過Web Services型態的介面，實驗擁有者將在網路中等待願幫他分擔運算的電腦跟他接洽。此時，運算端的運算端程式將會去找尋並接洽有需求的電腦。待接洽完畢，需求端就會把任務佇列裡的工作分給運算端，等到運算端運算完畢，再回傳實驗結果給需求端。就這樣一直循環下去，直到實驗擁有者設計的整個實驗運算完畢。

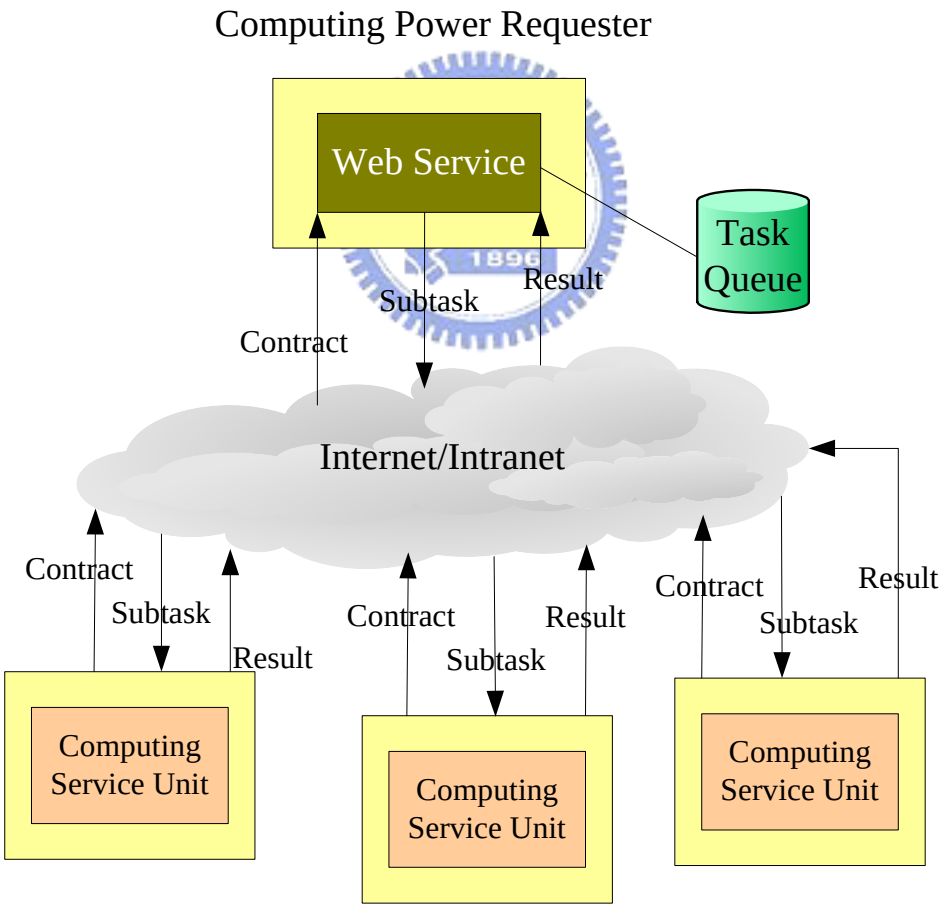


圖 4-1　CPS 基本架構圖

4.2. Computing Power Services 運作方式

4.2.1. 參與角色及其功能

確立了架構的運算能力分享方式之後，尚有一個問題待解決：在網路上有著這麼多台的電腦，運算能力提供者要如何去找到有需要的人呢？因此，協調者的角色出現了，他幫忙維護一組有需求電腦的清單，一但運算提供者閒置時，便可以透過它找尋到有需求的電腦。

CPS 架構要能正常運行，將需要下述三種角色：

- 協調者(Coordinator)

  協調者這個角色在整個架構裡如同一個仲介和公正第三方的角色。它主要的功能是負責維護一份需求者所在與需求的清單。而該份清單是根據需求者在實驗流程規劃完畢之後，會到協調者這裡登記需求而生成。

  而運算者找工作也是透過協調者。在運算者閒暇之時，它會到協調者端詢問是否有適合的工作。協調者則會根據清單，循序的(round robin)根據需求把需求者的連結點分派給運算者。之後，運算者會自行和需求者溝通，協調者只負責撮合的工作。

  另外，協調者也從事紀錄哪些運算者完成了哪些需求者的工作、還有帳號管理之類的工作。正如同，基本 Web Services 架構之中的 UDDI server，一個單純的公正第三方以及仲介的角色，並不直接涉及運算分享的指派或運算。

- 需求者(Computing Power Requester)

  需求者在整個架構中可以說是一個起點，它設計了整個實驗流程，

也因而有了運算能力的需求。在運作上，首先需求者會根據自己的意向設計整個實驗流程。爾後需求者會跟協調者接洽，公告它的需求。之後，它要針對找上門來的運算者指派工作，來協助自己完成實驗。

實驗設計、實驗工作流程控管和工作數量指派是需求者在整個架構中主要的工作。

- 運算者(Computing Unit)

運算者在整個架構中顧名思義就是進行運算的工作。透過運算端程式，運算端會在電腦閒暇時去詢問協調者是否有適合的需求者。待協調者回復需求者的位址(Web Services 的 access point)，運算端會透過需求者端的 Web Services 跟需求者接洽。接洽之後，運算者會從需求者處得到一些工作，在下載一些需求檔案之後開始運行，待完成後回報給需求者。之後再一次詢問協調者有無合適需求者，就這樣不斷循環地的運算下去。

4.2.2. CPS 運作流程的程序單元--TaskUnit

CPS 架構運作的起點是從需求者設計實驗開始。透過 Eclipse 或一些工具設計出一段 BPEL 流程，來表示該實驗的所有程序及流程。使用 BPEL 的目的在於，試圖利用 BPEL 語法標準以及 BPEL 的解譯引擎，來增加 P2P 架構的工作設計彈性以及工作流程控管的能力。

BPEL 流程設計完成，透過 Application Server 發布之後，需求者可以執行這一個實驗程序。而一但實驗程序開始執行之後，BPEL 的既定標準將根據需求者設計的流程，逐一運行這些程序。同時，工作流程控制的特色也使的該流程可以處理無論是平行處理還是相依程序之類的問題。

BPEL制定的目的主要是要針對Web Services Composition的問題，定義一個

運行架構，因此想要應用BPEL在CPS架構上，相對應的Web Services是一個關鍵。因此，本研究針對這個問題，設計了一個名為TaskUnit的BPEL流程，其運作可參照圖 4-2。採用TaskUnit這個機制的目的主要有下述幾項：

- 作為 BPEL 流程和 P2P 運算分享中介層之間互動的一個媒介

  BPEL制定的實驗流程，只能在BPEL引擎的控制上，具有流程控管的能力而已，本身並不具有P2P的運算分享能力。因此，要透過TaskUnit的利用。在它的流程之中，會呼叫Exclient這個需求者端的Web Service，該Web Service會根據傳來訊息，在需求者端生成一串的工作佇列之後，並向協調者登錄自己有運算能力的需求。透過引用TaskUnit的方式，在整個實驗的BPEL流程之中，TaskUnit將代表一個需要其他電腦幫忙運算的流程點，建立一個與P2P運算分享中介層溝通的機制。詳細流程可參考 4.2.3 的運算流程或見圖 4-3的系統架構介紹。

- 運算工作結束時，非同步回報機制控制

圖 4-2中，TaskControl相關的兩個程序，TaskControlManager(initiate Task)和TaskControlManager(onTaskResult)，，是負責處理非同步Web Services回報的工作。由於分散式運算這種模式本身就難以估計工作的運作時間，因此非同步回報的確是一個比較合理的方式。但由於目前眾多支援Web Services的語言在處理非同步Web Services上仍有所不足和限制，因此，CPS採用Oracle Process Manager的TaskManager API去建置一個流程非同步機制。當該流程的工作佇列全部完成時，會通知TaskManager API該流程已完成，此時BPEL流程上的資訊流就會結束該程序，繼續往下一個程序執行。

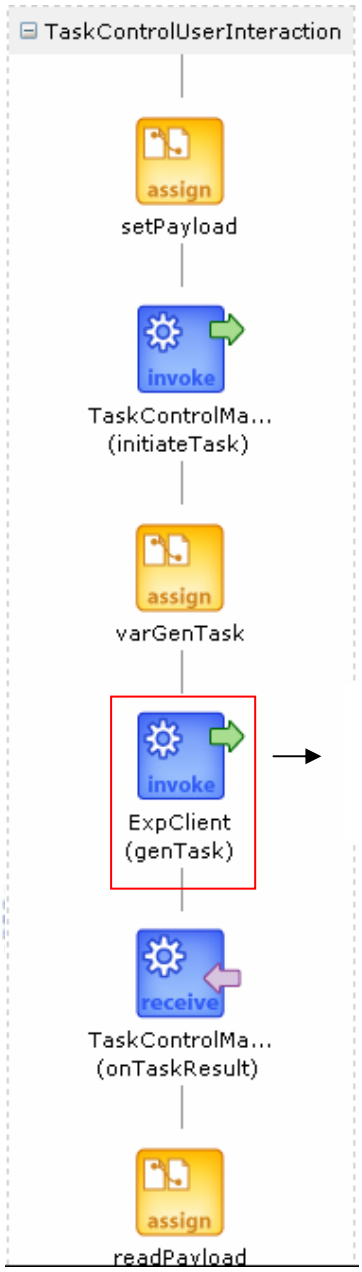


圖 4-2　TaskUnit 程序架構圖

4.2.3. P2P 運算分享中介層的分散式運算流程

P2P 分享運算中介層(P2P Power Sharing Middleware)是整個 CPS 架構進行分散式協同運算的一個關鍵。透過 TaskUnit 對於需求者端 ExpClient 這個 Web Service 的呼叫，而與整個 BPEL 程序連結。

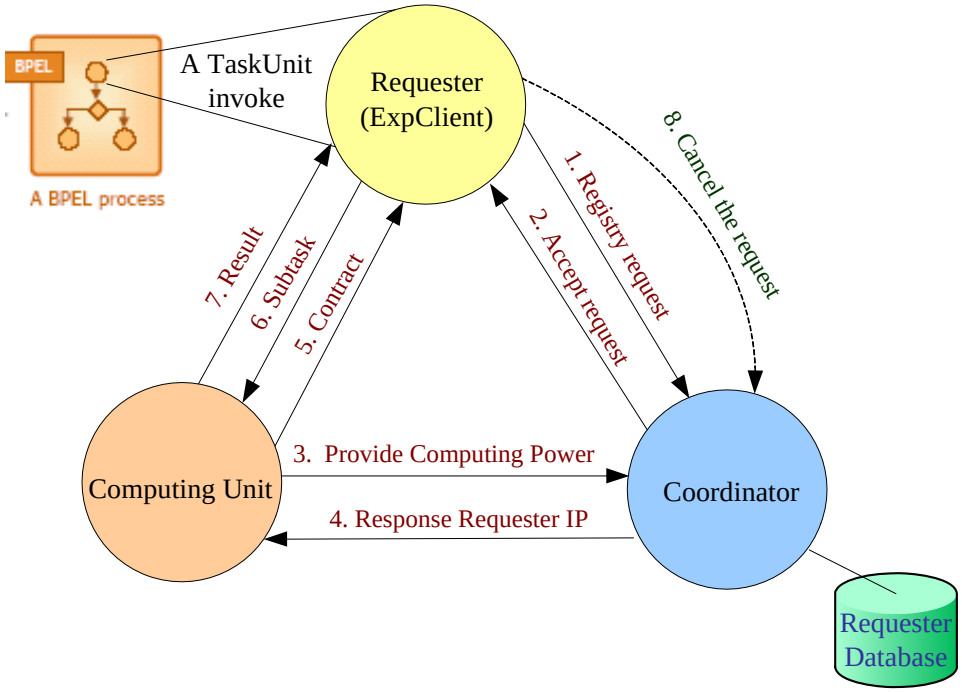


圖 4-3 CPS 運算分享示意圖

當需求者啟動這個實驗流程時，資訊流將照著BPEL流程圖的規劃流動著。因此，當流程執行到某個TaskUnit的時候，如圖 4-3所示。需求者和流程圖連接的Web Service(TaskUnit所Invoke的ExpClient)會收到產生工作佇列的需求。爾後，需求者端便會根據需求產生一串工作佇列，並向協調者登錄它的運算需求。該工作佇列，是以XML檔案格式的方式，紀錄該流程點中，需求者期望分派出去子工作(Subtask)。每個子工作(Subtask)的內容包括該工作的工作命令，對應參數及環境等等。圖 4-4為某一執行一個叫TokiRun的檔案的工作佇列的內容。

```
<TaskSet>
  <SubTask>
    <no>1</no>
    <execfile>TokiRun.exe</execfile>
    <para>1</para>
    <status>2005/5/31 下午 09:28:03</status>

<source>run_1.exe;run_1.bat;ft_1.txt;psu.key;tree.txt;WMD
T.exe;Corr.exe;lena.idwt.raw;lena.jp2.raw</source>
    <result>$f:corr1_1.txt;$f:corr2_1.txt</result>
  </SubTask>
  <SubTask>
    <no>2</no>
    <execfile>TokiRun.exe</execfile>
    <para>2</para>
    <status>Not Assigned</status>

<source>run_2.exe;run_2.bat;ft_2.txt;psu.key;tree.txt;WMD
T.exe;Corr.exe;lena.idwt.raw;lena.jp2.raw</source>
    <result>$f:corr1_2.txt;$f:corr2_2.txt</result>
  </SubTask>
  <SubTask>
    <no>3</no>
    <execfile>TokiRun.exe</execfile>
    <para>3</para>
    <status>Not Assigned</status>
```

圖 4-4 工作佇列內容實例

待協調者登錄完畢後，協調者會回覆接受訊息給需求者。此時，需求者端的 BPEL 實驗程序之中，資訊流是停在該 TaskUnit 裡的，非同步回報機制會等待工作完成的回報；待收到回覆之後，資訊流才再流動到下一個程序之中。

運算者的運算端程式則會在運算者電腦閒暇時，向協調者詢問有沒有適合的工作需求。協調者則會根據需求清單及運算者電腦的特性，採用 round robin 的機制回報清單中某一需求者的位址(Web Services 的 Access point)。

運算者得到需求者的位址(Web Services的Access point)之後，會連結到此處的Web Service，跟需求者要工作。這個動作在CPS架構裡稱之為簽訂契約(contract)。該份契約記錄了運算者和需求者所協議參與的工作項目以及需求者分派給運算者的工作批次(SubTask)的運算工作內容。圖 4-5即是CPS架構實際運作時所簽訂的契約內容。

```
<contract>
  <requester>
    <uniqno>050531212738TokiRun.exe%i</uniqno>
    <accesspoint>http://140.113.72.174/EXPClient/EXPClient.asmx</accesspoint>
  </requester>
  <mysubtask>
    <no>8</no>
    <execfile>TokiRun.exe</execfile>
    <para>8</para>
    <excption />

<source>run_8.exe;run_8.bat;ft_8.txt;psu.key;tree.txt;WMDT.exe;Corr.exe;lena.idwt.
raw;lena.jp2.raw</source>
    <result>$f:corr1_8.txt;$f:corr2_8.txt</result>
    <status>no</status>
  </mysubtask>
</contract>
```

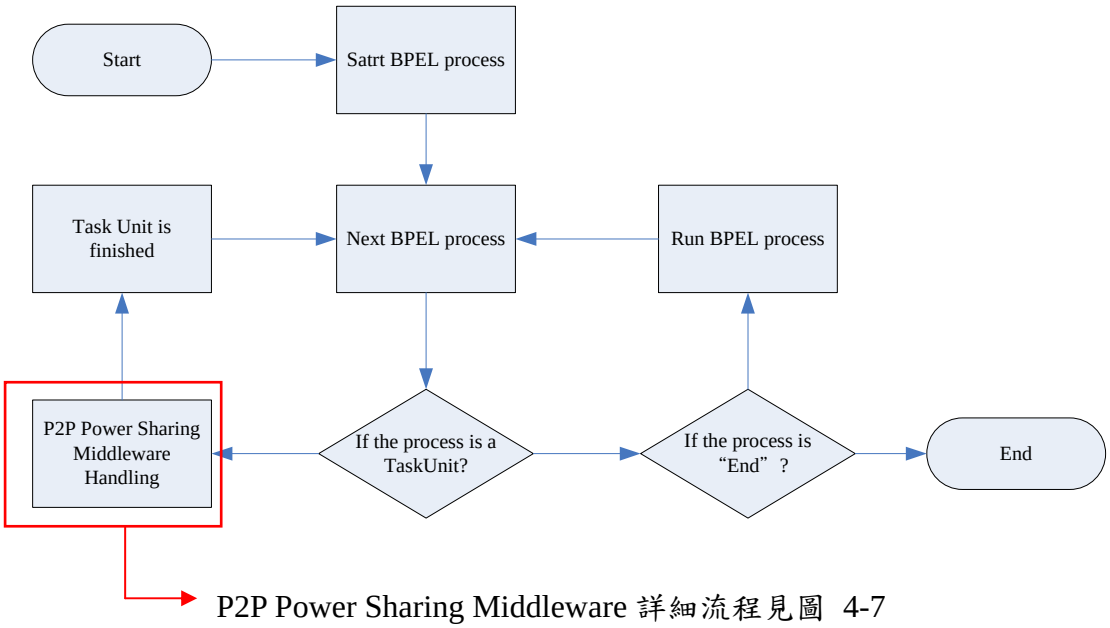
圖 4-5 契約(contract)內容實例

運算者簽訂契約之後，在 CPS 架構中也表示了運算者允諾了需求者幫它運算契約之中的工作。因此，運算端程式將根據契約內容，下載所需的檔案。之後，在 OS 之中以最低的優先權執行所約定的工作。待工作全都完成之後，運算端會根據所約定的內容及格式，看是要回傳某檔案或某值，以回報需求者端該契約工作運算的結果。

結果回報完成之後，運算端則會再到協調者端詢問合適的需求者位址，循環地再一次地進行另一次的運算。而需求者，在運算者回報實驗結果時，也會確認工作是否已全部完成。若已全部完成，需求者端的 Web Services 會跟協調者取消有運算需求的狀態。同時，也觸動 TaskUnit 裡的非同步回報機制，來完成這一個 TaskUnit 流程。

BPEL 流程在完成該 TaskUnit 之後，資訊流會繼續往下一個流程跑去；若是流到了另一個 TaskUnit，則同樣的分散運算程序會再一次進行下去，直到整個實

驗流程結束為止。



P2P Power Sharing Middleware 詳細流程見圖 4-7

圖 4-6 CPS 系統運作全流程圖

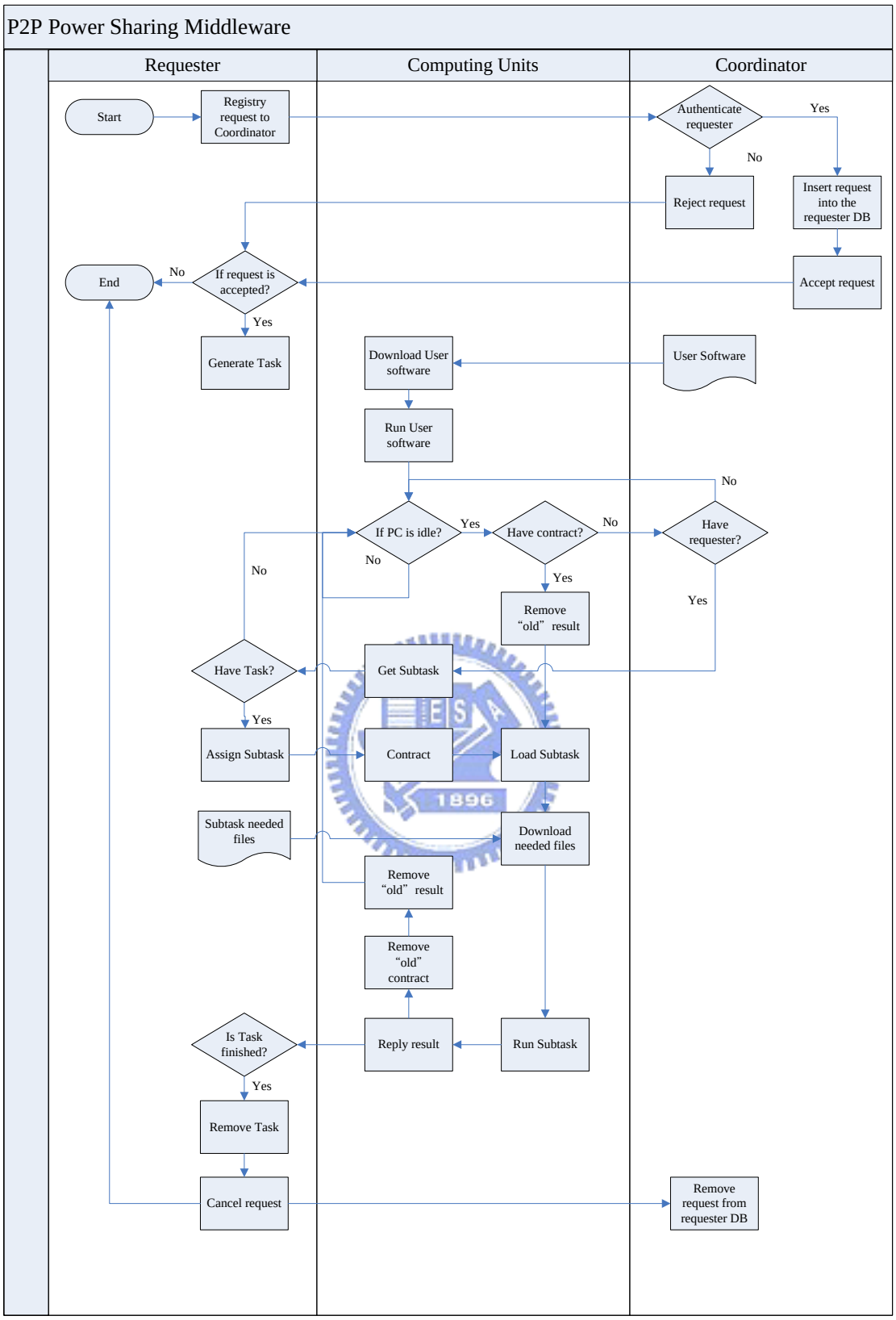


圖 4-7　P2P Power Sharing Middleware 運作全流程圖

4.3. CPS 系統架構

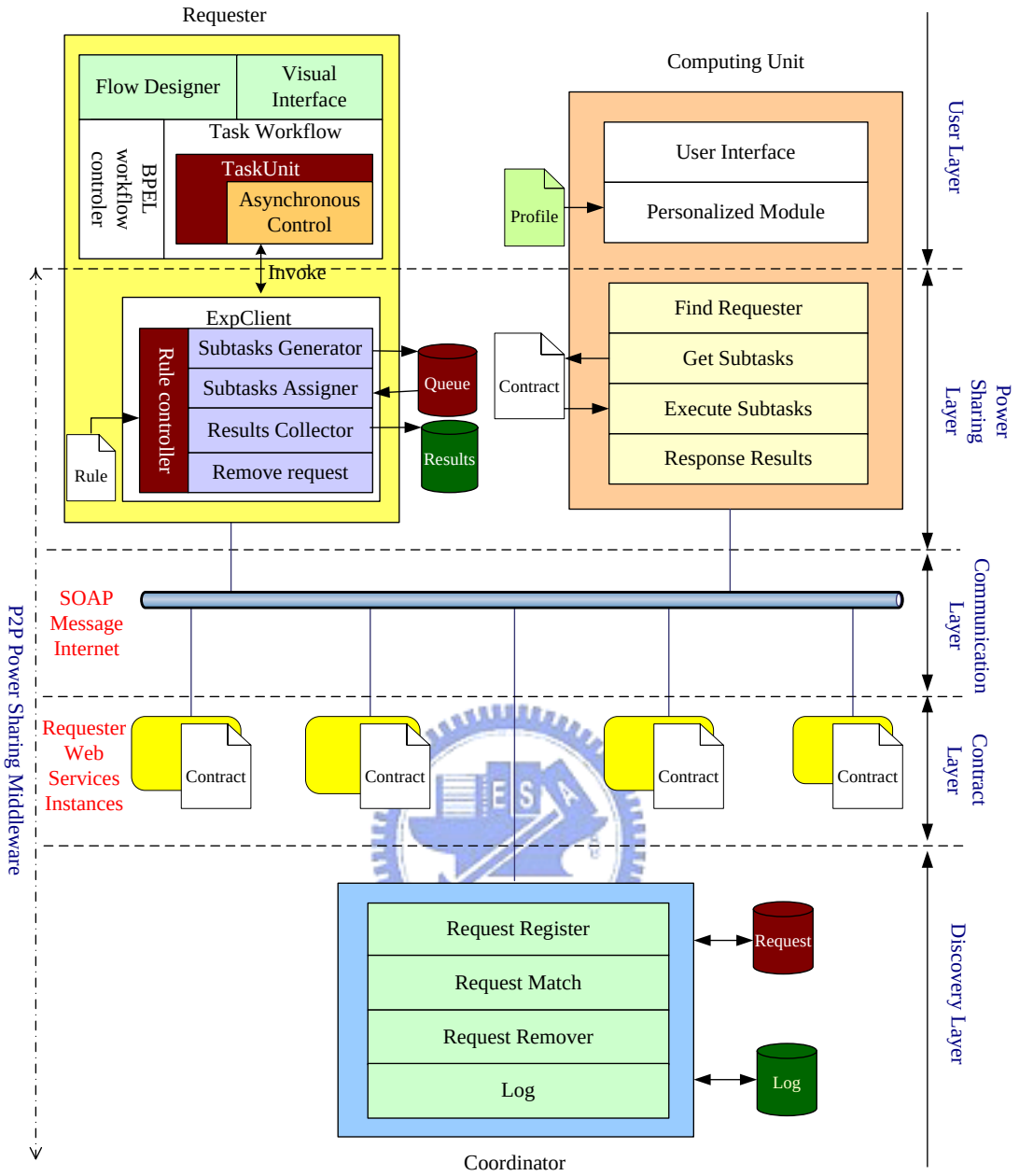


圖 4-8　CPS 系統架構圖

CPS系統架構圖如圖 4-8所示。以整個系統架構來看，可以依功能把它分為五層。分別為使用者層(User Layer)、運算能力分享層(Power Sharing Layer)、通訊層(Communication Layer)、契約簽訂層(Contract Layer)以及發現層(Discovery Layer)。其中，後面四層，運算能力分享層、通訊層、契約簽訂層以及發現層，

構成了整個架構中的P2P分享運算中介層(P2P Power Sharing Middleware)，是為分散式運算分享運作的核心。對於各層的功能，茲分述如下：

4.3.1. 使用者層(User Layer)

使用者層是使用者存取整個系統及架構的介面。下面將對系統於該層提供的各個功能，依使用者角色分類後分述如下：

- 需求者端(Requester)

  需求者端的使用者層是由 BPEL 實驗流程和使用者介面所構成。使用介面主要有 BPEL 的流程設計和 BPEL 流程的執行工具。因此，一個視覺化的 BPEL 設計工具與一個具網頁管理模式的執行環境構成需求者端的介面。

  而設計出來的 BPEL 流程，需要透過一個 BPEL 流程控制引擎來執行。流程的基本單元 TaskUnit 是該層與運算能力分享層溝通的橋樑。其中包含著非同步控制單元，控制運算分享工作的開始與結束。

- 運算者端(Computing Unit)

  運算者端的使用者層主要就是運算端程式的使用者介面。另外個人化模組，也是運算者可以直接存取到的部份。個人化資訊會保存在「profile」這個文件之中。

4.3.2. 運算能力分享層(Power Sharing Layer)

運算能力分享層是 CPS 架構中運作 P2P 協同運算的主體，也是需求者端和運算端運行分散運算的關鍵。其功能依角色不同分述如下：

- 需求者端

運算能力分享層中，需求者部分主要是 ExpClient 這個 Web Services。透過上一層中的 TaskUnit 對於該 Web Services 的呼叫，來產生 P2P 協同運算之效果。

該層的主體 ExpClient 主要有四個功能：(1)工作產生器(Subtask Generator)根據 TaskUnit 傳來的需求產生工作佇列；(2)工作分派器(Subtask Assigner)把工作佇列的工作分派出去；(3)結果回收器(Result Collector)負責會收分派出去工作的運算結果；和(4)取消需求模組(Remove request)負責監控該工作佇列是否已完全執行完畢，若是則取消協調者上的需求登記。

另外，獨立存在著一個規則控制器，根據文件「rule」之中的規則控制其他功能的運作規則。

- 運算者端

運算者端在運算能力分享層主要的工作在於要求運算工作來本機上執行。透過下列四項功能，來完成需求者端協同運算之目的：(1)需求尋找模組，會向協調者詢問需求者的所在。(2)工作索取模組，則會向需求者要求工作，並簽定契約。(3)工作執行模組之後則根據契約內容下載需求當案並進行運算。(4)最後結果回報模組則在工作運算完畢後，回報運算結果給需求者。

4.3.3. 通訊層(Communication Layer)

通訊層主要的功能是負責 CPS 架構之中各個 component 間的溝通。因此，採用 Web Services 建置的 CPS 架構自然繼承了 Web Services 的通訊架構。SOAP 的資料交換網路構成了 CPS 的通訊層，形成了整個 P2P 分享運算架構中介層之中主要的通訊方式。

4.3.4. 契約簽訂層(Contract Layer)

契約簽定層提供運算端 component 和需求者 Web Services 互動並簽訂契約的功能。繼承了 Web Services 的特性，以 XML 為基礎的資料交換機制增加了 CPS 架構平台的能力。每一個運算端在和需求者端簽訂契約時會連結到獨一的 Web Services instance，並和它協調出唯一的契約並簽定之。這個機制構成了整個架構之中工作分派的契約概念，是控制運算分享秩序的關鍵。

4.3.5. 發現層(Discovery Layer)

發現層的主體就是協調者本身，而協調者本身需求登錄、需求配對、需求取消和日誌記錄的工作就是 CPS 架構中發現層所提供的功能。其中，需求登錄會把需求者在工作產生之後的運算需求登錄到需求紀錄資料庫裡；爾後，運算者來詢問有無需求者可服務時，會把資料庫中適合的需求者位址 Round robin 地分派出去；需求取消則處理需求者在工作完成後取消資料庫中需求登錄的請求；至於日誌則負責記錄需求者完成了哪些契約。

4.4. 例外處理機制

CPS 是一個 P2P 分散式架構，因此在運作時，每一個參與者的動態是難以掌控的。因此，本節將就 CPS 在實行分散式協同運算時，一些可能發生例外狀況以及系統對應的處理方式，做個介紹。

一般來說，CPS 架構運作中的例外可能有以下兩種：(1)使用者關閉運算端程式；(2)運算端電腦過於繁忙或其他因素無法履行契約。針對這些問題，CPS 採用了(1)久未回報工作重新分派以及(2)運算端程式的 Roll-Back 機制來予以克服。

4.4.1. 需求者端的正常工作分派以及久未回報工作重新分派機制

需求者端實際分派工作的流程如圖 4-9所示。正常的派工機制為：首先，工作分派程式會先載入需求者所制定的一些工作分派原則(assigning rule)；然後先從工作佇列之中尋找未分派的工作。再根據尋找的結果以及工作分派規則產生工作清單(Subtask Lists)；最後把工作清單分派給運算者完成工作分派。

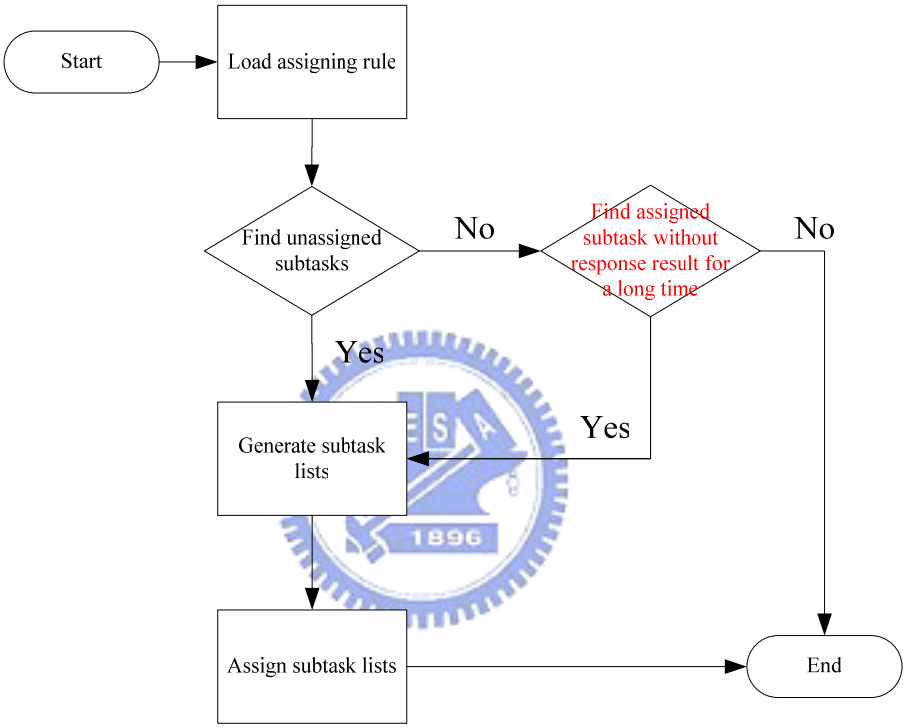


圖 4-9　需求者端工作分派流程圖

當有工作分派出去後，卻遲遲沒有回報結果時，CPS需求者端的派工便會運「工作重新分派」的機制來處理，流程見圖 4-9。一但需求者端分派工作時找不到工作佇列中未分派的工作，它便會尋找已派出但「過久」未回覆結果的工作。至於多久未回覆結果者可成為「過久」，則根據需求者的工作分派原則。之後如同正常派工一般，產生工作清單並分派之。

「工作重新分派」機制，可以應付運算者無法正常履行契約的問題，但重複工作分派的可能也會造成網路內資源的浪費。需求者在制定工作分派原則時，須

謹慎考慮。

4.4.2. 運算端程式的 Roll-Back 機制

當使用者關閉運算端程式之後，回報運算到一半的工作，是不合理的操作；若是不回報之重新要求新的工作運算，則是沒有履行至之前簽訂的契約。因此，運算端程式需要一個機制，來管理運算端程式被關閉之後，如何履行舊契約的問題。

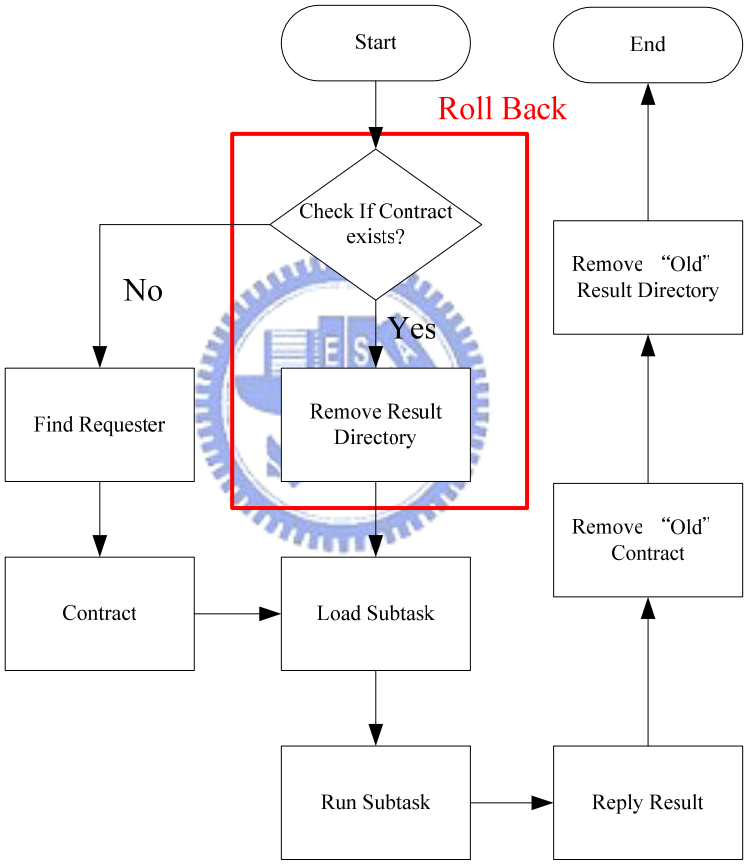


圖 4-10 運算端程式(Computing Unit)運作及 Roll-Back 流程圖

正常的運算端程式的運作會如同圖 4-10所示。未簽訂契約的運算端程式，會向協調者詢問合適的需求者位置，再跟需求者要工作做，訂簽訂契約。接下來去執行契約，運算工作，完成後回報工作結果，終結此份契約的效力。

一但運算端程式被關閉之後，之前的運算中斷，自然運算的結果也不能採

用。但是已簽訂的契約還是要履行，因此，程式在重新啟動時，會去確認是否有未履行的契約存在。若存在的話，表示之前有運算中斷的情況發生，因此，之前的運算結果要刪除，再重新運算一次，直到成功履行契約為止。這個機制稱為「Roll-Back」，採用重新運算這種較為保守的方式，來處理運算端程式被關閉之後，未履行契約的問題。

5. 系統建置與成果

5.1. 系統建置

CPS的建置環境如表格 5-1所示：

表格 5-1　CPS 架構環境需求

參與角色	環境需求
需求者端 (Requester)	Microsoft Windows 2000、XP、 2003 Oracle BPEL Process Manager 2.1.2 Server Oracle BPEL Process Manager Designer 2.2 + Eclipse 3.0 Microsoft Internet Information Services 5.1 Microsoft .Net framework v1.1 J2SE SDK 1.4.2.08 ExpClient v1.0 (需求者端 Web Services)
運算者端(win32 版) (Computing Unit)	Microsoft Windows 2000、XP、 2003 Microsoft .Net framework v1.1 Computing Unit v4.0 win32 版(運算者端程式)
協調者(Coordinator)	Microsoft Windows 2000、XP、 2003 Microsoft .Net framework v1.1 Microsoft Internet Information Services 5.1 Microsoft SQL Server 2000 Personal Coordinator v1.0 Web Services(協調者端 Web Services)

以下三小節，將針對各參與角色的系統建置作分別敘述。

5.1.1. 需求者端

需求者端主要有兩個部份，一個是BPEL流程相關的環境，另一個是需求者端Web Services。前者主要是採用Oracle BPEL Process Manager配合J2SDK做為BPEL的執行引擎。同時，導入BPEL PM Designer加上Eclipse為圖形化BPEL流程設計工具(見圖 5-1)，賦予需求者一個視覺化且能簡單操作的BPEL設計及執行管理環境(見圖 5-2)。在需求者的使用者層上主要是利用現成工具達成。

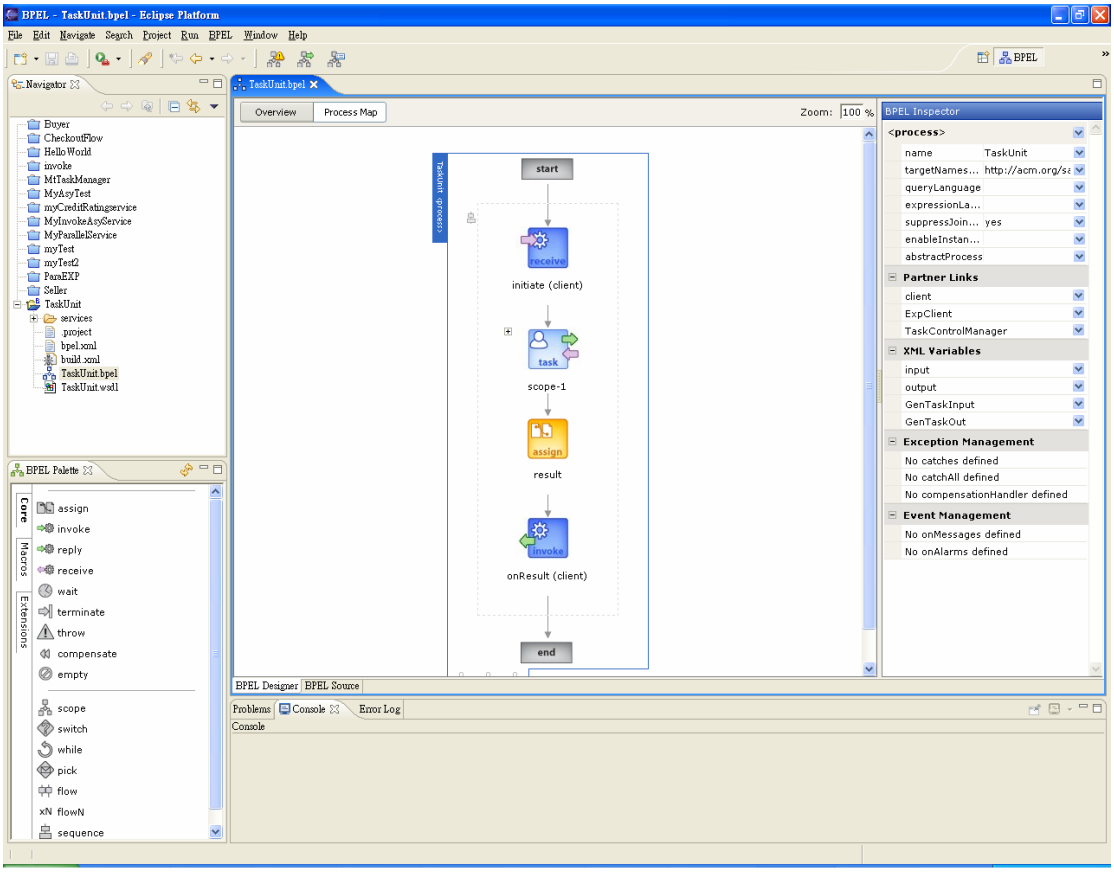


圖 5-1 Eclipse + BPEL PM Designer 開發工具

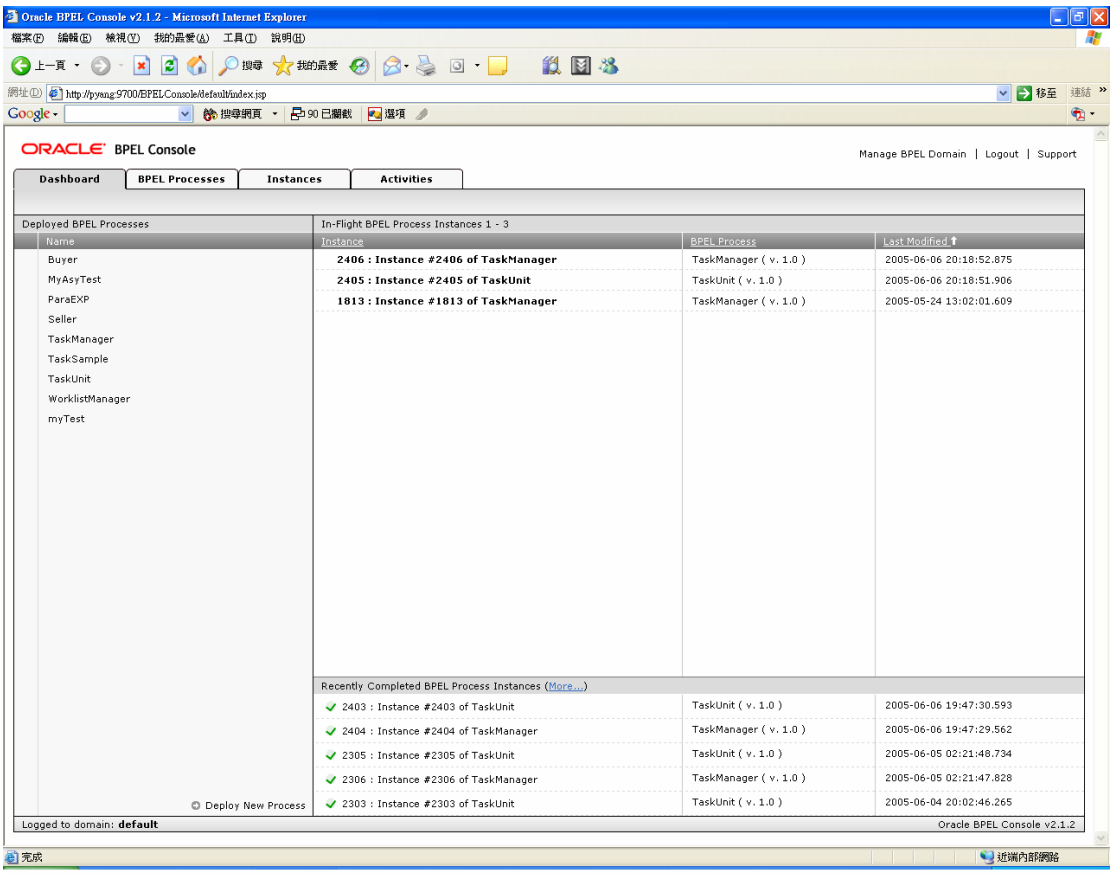


圖 5-2　BPEL Process Manager 執行與管理介面

需求者端 Web Services，主要在需求者端的運算能力分享層上，ExpClient Web Services 採用 VB.Net 撰寫而成，配合.Net Framework v1.1 及 IIS 建置而成需求者端 Web Services，提供 CPS 架構協同運算的能力。

5.1.2. 運算者端

運算者端的運作主要由運算端程式(Computing Unit)來負責。該程式具有平台依賴性，不同平台應該有不同的版本可供使用。目前的建制只開發出Win32 版本的Computing Unit。Computing Unit Win32 版，一個使用VB.Net建置而成的視窗程式。在介面上，包含了隱藏於工具列(Hiding)、休眠/繼續工作(Snooze/Resume)切換按鈕以及關閉程式按鈕。同時，並有狀態列回報目前運作的情況及參與工作名稱還有運作工作編號。隱藏與工具列之後，也能透過點滑鼠右鍵的工作列清單

管理Computing Unit這支程式。以上可參考圖 5-3及圖 5-4。

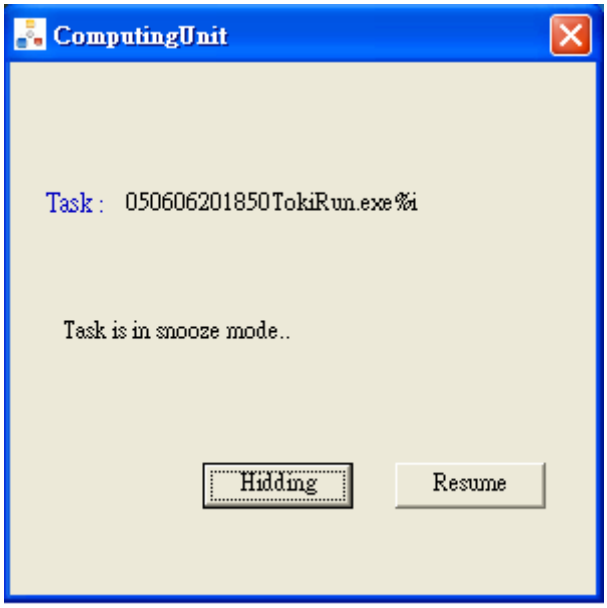


圖 5-3 Computing Unit 正常模式介面

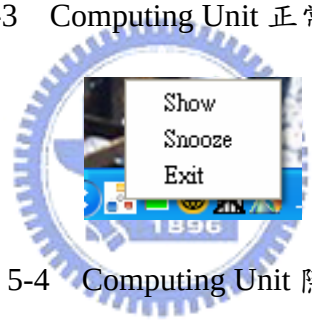


圖 5-4 Computing Unit 隱藏模式

運算端程式在執行契約之中約定的任務時，把要運算的工作優先權設為最低之後，再交由 OS 去做排程。OS 會根據目前運作的全部程序的優先權以及資源的需求，去對 CPU 時間做妥善的分配。由於，工作執行是以低優先權運作的，理論上在 CPU 時間上是搶不贏其他運作程序的，因此，應當不會影響運算端電腦的日常作業。

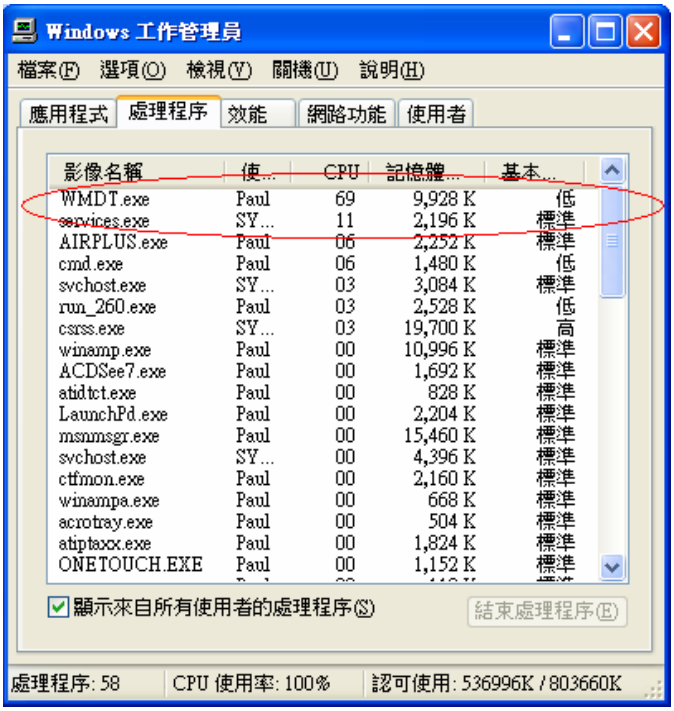


圖 5-5 工作執行交由 OS 做排程

圖 5-5所示為工作管理員中，工作以低優先權執行的情況。OS會優先把CPU時間配給其他優先權較高的程序(高>標準>低)，剩下CPU時間的才配給低優先權的工作。

5.1.3. 協調者端

協調者端的主體是 Coordinator Web Services，一個採用 VB.Net 建置而成的 Web services。配合.Net Framework 與 IIS，協調者以 Web Services 為介面，協調著整個 CPS 架構的協同運作。其中，需求清單和日誌部分採用 Microsoft SQL Server 2000 Personal 為資料庫，以求存取與搜尋上的便利性。

5.2. 成果分析

對於 CPS 架構是否能確實聚集網路內的閒置電腦運算能力，為可批次化的繁重運算之工作，提供一個具有工作流程控管能力的輕量級分散式運算能力分享環境。本小節將採用下述的實驗模擬，來證明這些論述。

- 使用多部電腦來運算一個繁重工作，來證明分散式協同運算的效果。
- 實際運作平行工作流程和相依工作流程，來驗證本架構工作流程控管的能力。
- 分析運算端程式在協助工作運算時對於運算端電腦日常運作的影響，根據實際數據的表現來證明本架構不影響運算端電腦日常作業的運作。

5.2.1. 分散式協同運算

要驗證 CPS 在分散式協同運算的效能，需要一個高運算能力依賴的實驗來予以執行。因此，本研究參考曾立信在 2003 年提出的「多重小波轉換濾波器浮水印嵌入」[9]，作為實驗對象。

其流程為：首先使用原嵌入浮水印時所使用的各組小波轉換濾波器組合以及同樣的小波樹結構來對影像進行小波轉換分解，之後再將浮水印所嵌入之子頻帶的資料取出，並以公式 5.1 與原始浮水印資料進行相關係數(correlation)之計算。其中 W 為欲檢測之原始浮水印資料，W’為由影像偵測出之浮水印資料。若計算出來的相關係數大於一門檻值(threshold)—以該文中所提出之標準為 0.4—可依判定此圖是否嵌入有用以檢測之原始浮水印資料 W。詳情可見或直接對照參考文獻[9]。此處將不多作介紹。

$$Correlation = \frac{\sum_i (W_i - \overline{W}) \times (W'_i - \overline{W'})}{\sqrt{\sum_i (W_i - \overline{W})^2} \times \sqrt{\sum_i (W'_i - \overline{W'})^2}} \tag{5.1}$$

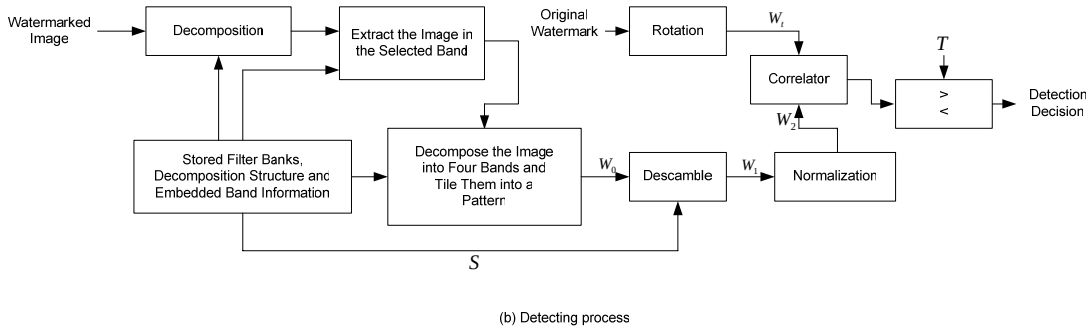


圖 5-6　多重小波轉換濾波器浮水印嵌入流程示意圖 [9]

實際實驗部分，採用了 76177 組 filters 來解浮水印，此步驟由於對於電腦的運算能力相當依賴，加上可以把 76177 組 filters 拆成小批次進行。因此是一個可批次化的高運算依賴工作，很適合於 CPS 系統作協同運算處理。

在實驗設計上，本實驗把 76177 組 filters 切成 100 個一組(subtask)，共 762 組子工作待運算。工作內容上，分別使用不同組的 filters，對加入浮水印的 raw 檔以及 Jpeg2000 的圖檔解出浮水印，並計算出對應的相關係數值。

參與運算電腦皆為相近效能的電腦(Intel Pentium 4 3.2GHz加上 512MB DDRII記憶體或等同效能電腦)，數量上由 2 台漸增至 6 台。同時，參需求者端和協調者端皆不參與運算的情況，且與運算者電腦是持續有使用者在正常使用的環境之下，可以得到圖 5-7之結果。

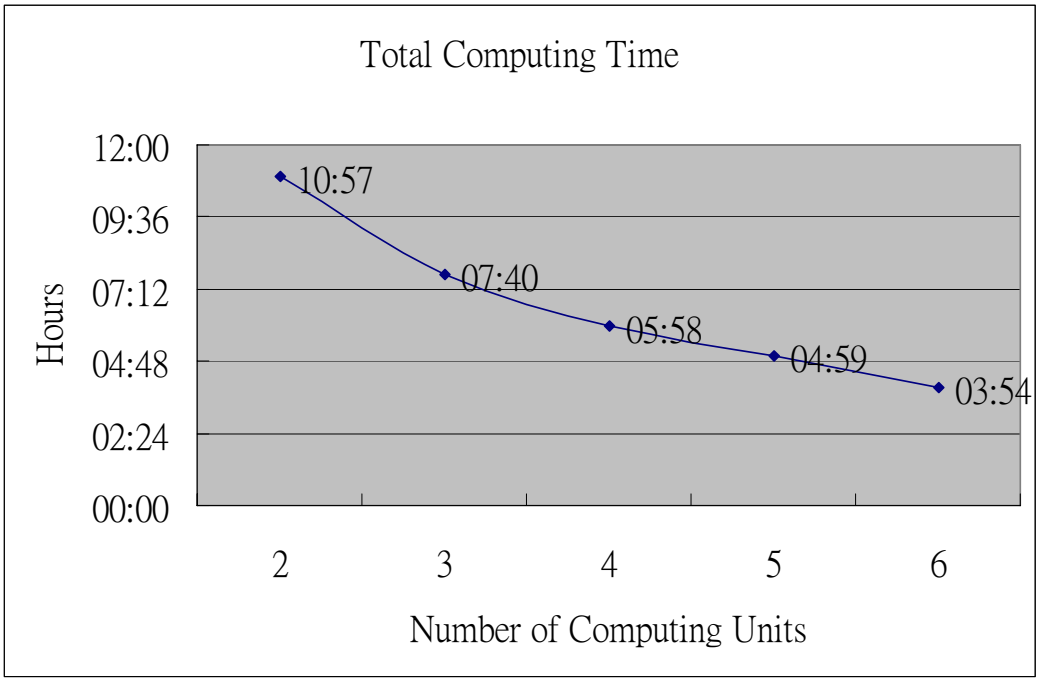


圖 5-7　參與電腦數量遞增之需求總時間遞減曲線

根據以往的經驗，實際在運作解浮水印的工作中，分別使用 76177 組的 filters 對加入浮水印的 raw 檔以及 Jpeg2000 的圖檔解出浮水印，並計算出對應的相關係數值，在 Intel Pentium 4 2.4GHz 的電腦需要約 23 個小時。預估若以 Intel Pentium 4 3.2GHz，也需要 20 個小時左右。

但在 CPS 架構中協同運算的幫助之下，在愈多電腦的參與之下，總工作時間可以得到有效減少；甚至在 6 台電腦加入運算時，只需要 4 小時不到的時間。CPS 架構的運作確實可以藉由協同運算的幫助，來整合並利用網路內的閒置電腦。

若把 filters 的數量增加到 1,628,250 個，在 500 個切為一組，並交由 15 台快慢不一的電腦，在正常使用的環境之下，參與實驗運算。這樣一個預估單機運作需要 18 天才能完成的工作，在 CPS 環境下用了 2 天 14 小時 45 分鐘 13 秒完成這個實驗的運算。

5.2.2. 工作流程控管

本小節將著重在工作流程控管能力的驗證上。由於，CPS 架構中採用 BPEL 作為實驗設計語言，並藉助 BPEL 標準本身配合 BPEL 引擎的工作流程控管能力來運作。因此，先設計一個兼具平行與相依的 BPEL 工作流程，在透過該流程的執行來予以驗證 CPS 架構是否具有工作流程控管的能力。

圖 5-8為CPS架構工作流程能力驗證BPEL實驗流程圖。本實驗依舊使用之前使用「多重小波轉換濾波器浮水印」的方法為實驗對象(詳情參考 5.2.1.或參閱參考文獻[9])，利用了兩個平行的TaskUnit分別去執行取出加過浮水印的raw和Jpeg2000 圖檔，並算出相關係數值。結果回報，會由圖 5-8中的Client(onResult)負責執行，將在兩個平行的流程都運算完畢之後發生，並結束整個流程。

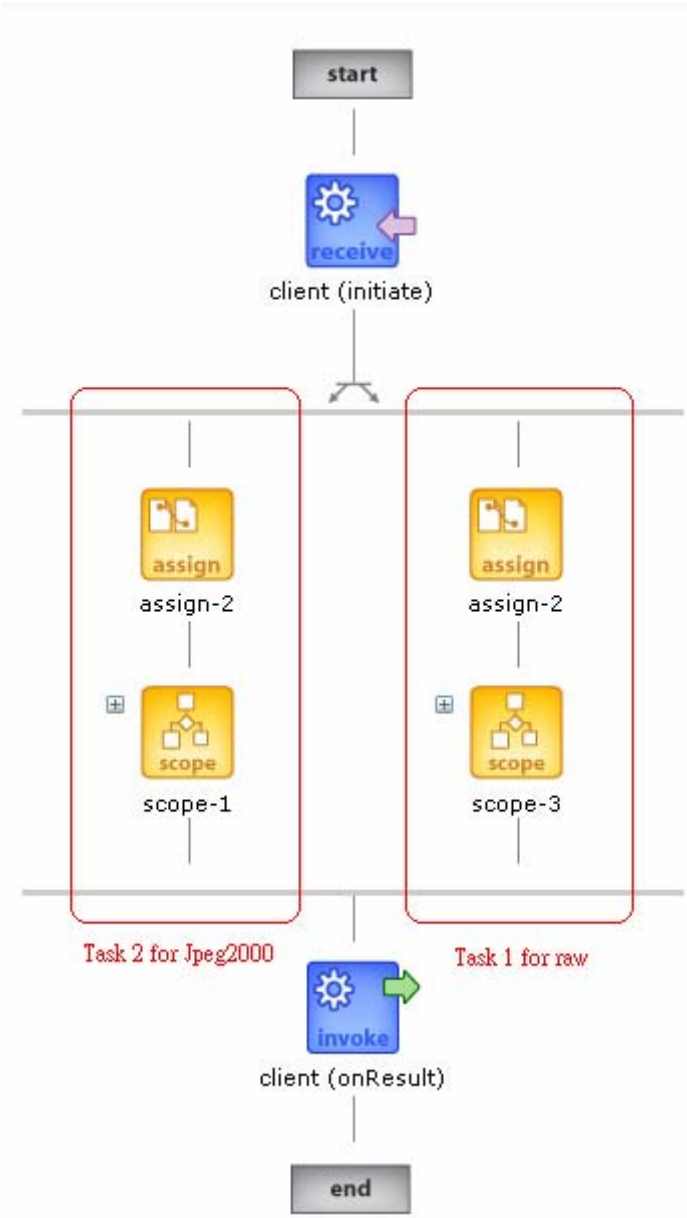
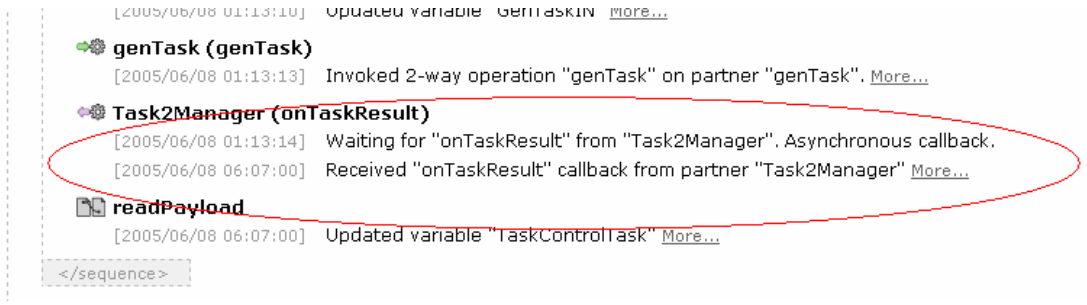


圖 5-8　工作流程控管能力驗證實驗 BPEL 流程圖

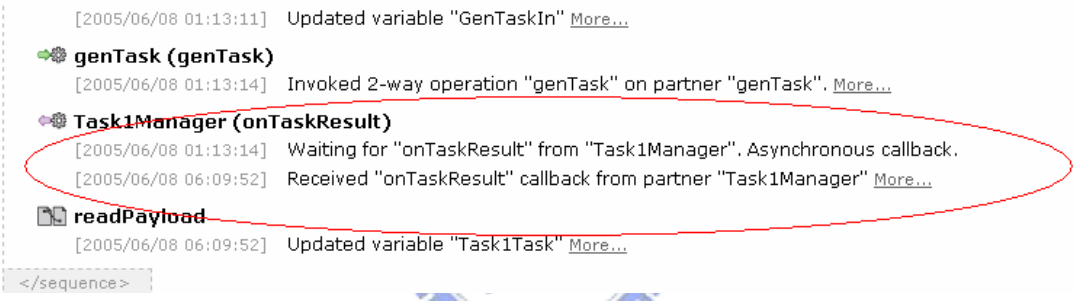
在預期實驗結果方面，由於兩個TaskUnit(Task 1 與Task 2)的執行是平行的，因此期望的開始運作時間應該是一樣的，而結束時間則視情況而定，不一定會一樣。而最後結果回報因為和之前的兩個平行程序有相依關係，所以應該待之前的兩個平行流程都完成之後才引發，故引發時間應該較慢或等於兩平行流程的完成時間。實際運作結果如表格 5-2所示：

表格 5-2　工作流程控管能力驗證實驗結果

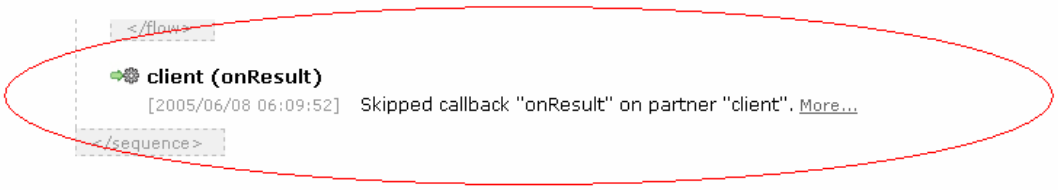
Task 1：



Task 2：



Client(onResult)(結果回報)：



由運作結果可以看到，兩個平行程序(Task 1 和 Task 2)的開始時間是一模一樣的，並各在不同的時間點結束運算；而結果回報的引發，則在較慢的一方完成之後發生。結果符合之前的期望，故能證明 CPS 架構在 BPEL 的導入之後賦予分散式協同運算環境一個工作流程控管的能力。

5.2.3. 運算端程式對運算端電腦日常運作的影響

運算端協助運算是以不影響運算端電腦日常運作為原則，本小節就針對該項目來做驗證。實驗方法上，採用一個叫 Super PI 的軟體，並紀錄其運算時間。該軟體由日本人所開發，是一個計算圓周率小數點後幾位的軟體，藉由其運算時間的長短，可以當作一台電腦效能衡量的標準。也由於 Super PI 相當依賴 CPU 運算能力，運作的同時將佔用電腦絕大多數的運算資源。因此，本實驗利用 Super PI 來模擬運算端電腦在運行繁重日常工作的環境；另外採用 Super PI 運算圓周率小數點後 1M 個位數(Super PI 1M)所需的時間作為衡量標準。

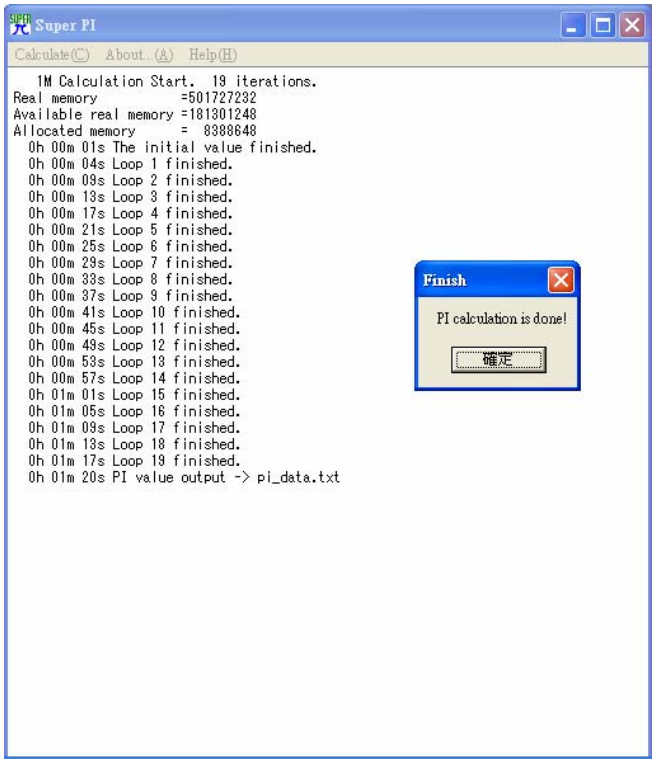
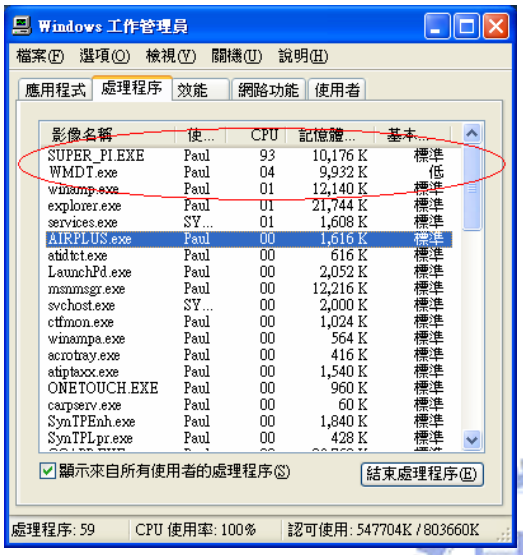
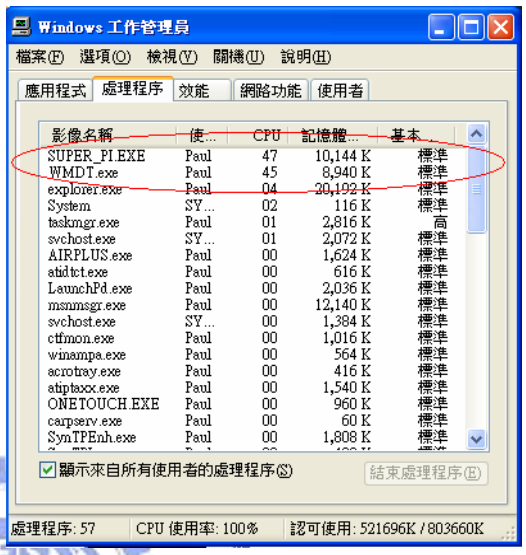


圖 5-9 Super PI 運算圓周率小數點後 1M 個位數的情況

實驗方法為：再一台單獨執行Super PI 1M需要 1 分 20 秒的電腦上，分別採用CPS架構的運算端程式和以及直接執行的方式來執行「多重小波轉換濾波器浮水印」的實驗。並由Super PI 1M所需的時間來判定兩種環境之下對於使用者操作的影響。不同的環境之下各做 5 遍，分別取平均值和單獨執行Super PI 1M所

需的時間做比較，差距越多，代表該方式越能影響使用者的日常作業。

表格 5-3　對運算者端電腦的影響

使用 CPS 運算端程式	直接執行
	
需時：01m22s	需時：02m05s
附註：單獨執行 Super PI 1M 需時 01m20s	

實驗結果如表格 5-3所示。使用CPS運作時，由於工作優先權是低的。所以OS會把大多數的時間分配給優先權式標準的Super PI去使用；相對的，直接執行的情況之下，優先權皆為標準的兩個工作會去搶奪CPU時間。從效果來看也很顯著：在單純只執行Super PI 1M需時 01m20s；而加入了採用CPS運算端執行「多重小波轉換濾波器浮水印」的實驗後時間略升為01m22s；至於直接執行「多重小波轉換濾波器浮水印」的實驗時間會暴增到02m05s。若以Super PI的運作當成運算者電腦的日常作業來看，CPS運算端的運作並不會對其造成巨大的影響。

5.3. 高檔案傳輸量環境探討

透過之前的印證，證明了 CPS 在處理小批次的需高運算能力的執行程式的分散式運算環境中，是一個能有效縮短執行時間，同時具有工作流程控管能力的解決方案。但是，倘若運行的實驗產生較大量的檔案傳輸時，CPS 是否適用？以下，將透過實驗驗證。

本實驗，透過傳送一個 130MB大小的檔案，主要著重在於較大量的檔案傳輸流量的環境上，運行的工作並不耗費大量的運算能力。單一批次流程圖可以參考圖 5-10。在共做 20 次的環境之下，參與運算電腦由一台漸增至 6 台，並監控工作的總完成時間及需求者端的網路流量狀況。每組實驗各做 3 次，取平均值做為判斷依據。

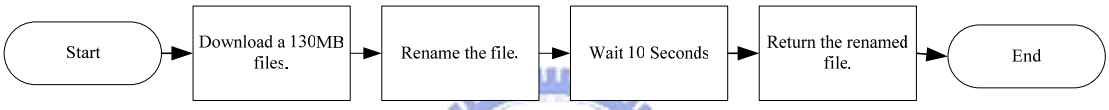


圖 5-10 高檔案傳輸實驗流程圖

在進行該實驗之前，先傳輸較小的檔案，分別為 1MB、2MB、4MB作為對照。在這個對照組實驗中，流程大致與圖 5-10相似，只是改變傳送檔案大小與數量，每個批次控制檔案的傳輸為 20MB，亦即 1MB傳 20 個、2MB傳 10 個、4MB傳 5 個，總共進行 30 個批次。參與電腦依舊由 1 台漸增至 6 台，取工作的總完成時間及需求者端的網路流量狀況為判斷依據。

對照組的實驗結果見圖 5-11。隨著電腦的增加，可以看到所需時間逐漸減少。儘管，相同環境下三組實驗的所需時間略有不同，但差異不大，可以歸為網路環境所以造成的合理誤差。

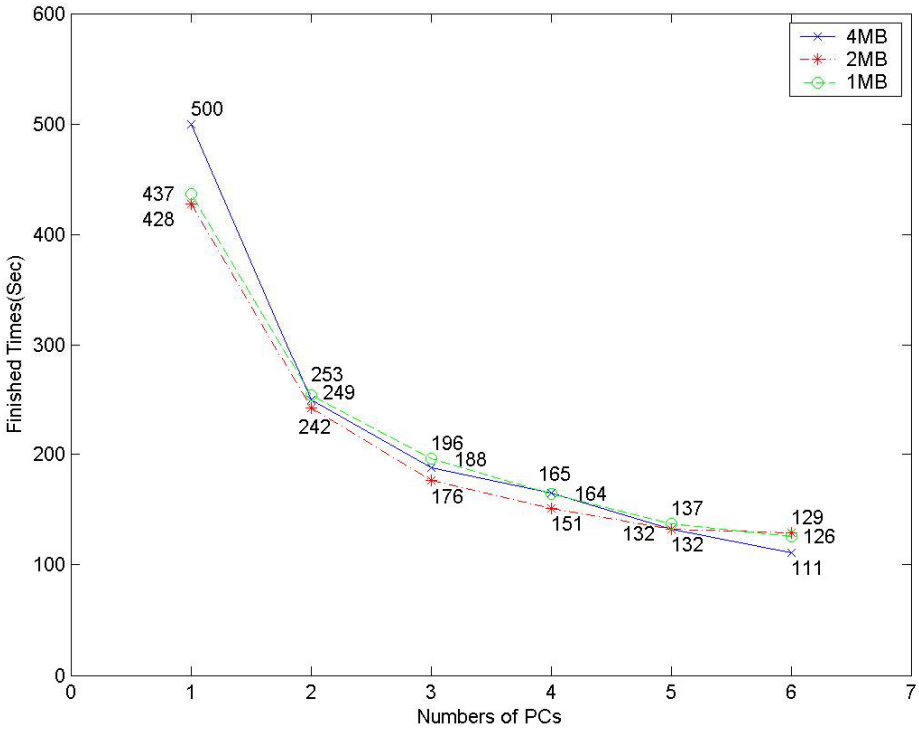


圖 5-11 較小檔案傳輸流量實驗完成總時間圖(對照組)

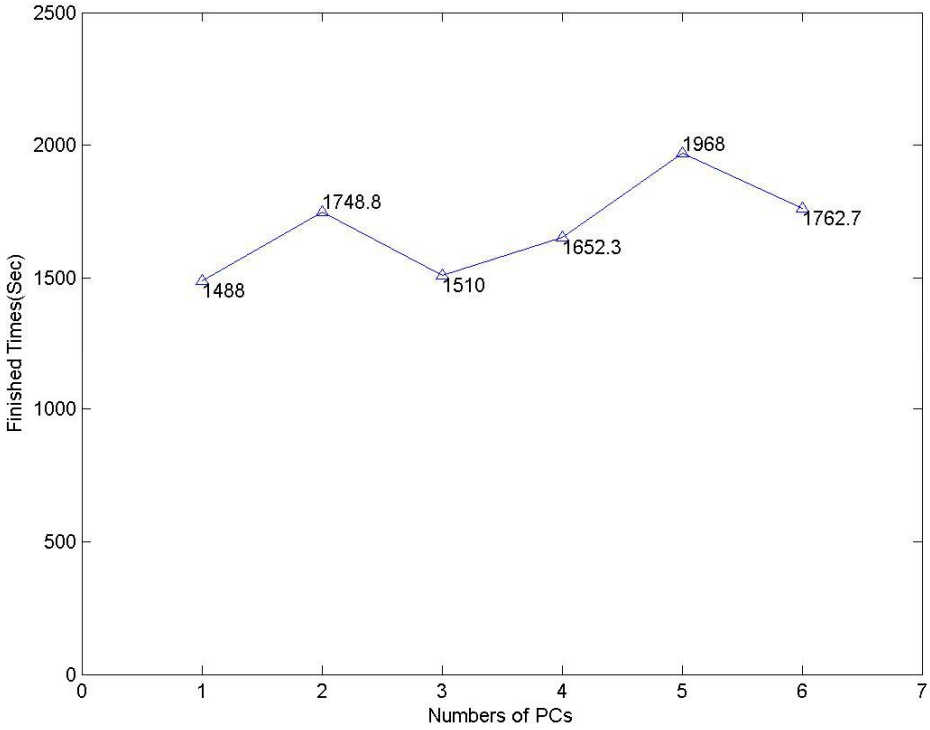
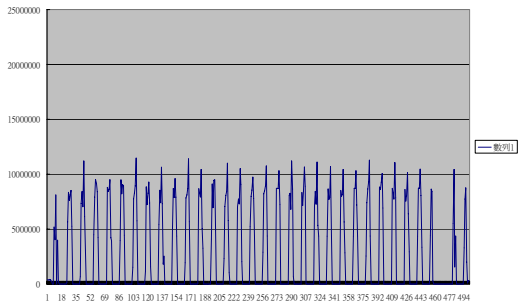
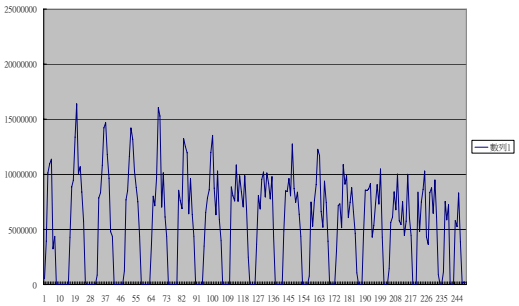
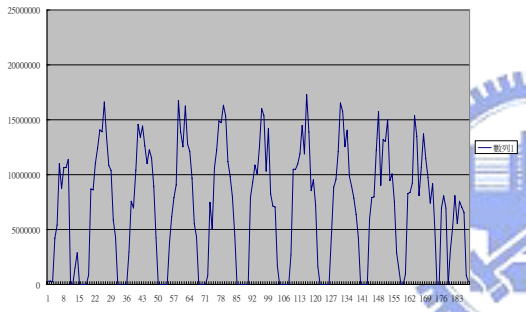
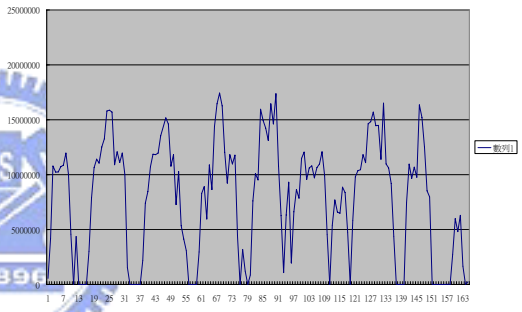
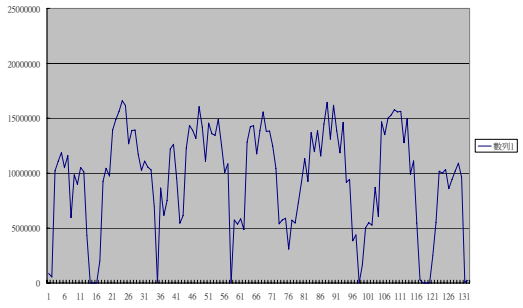
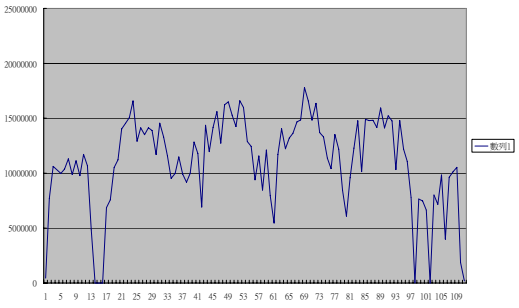
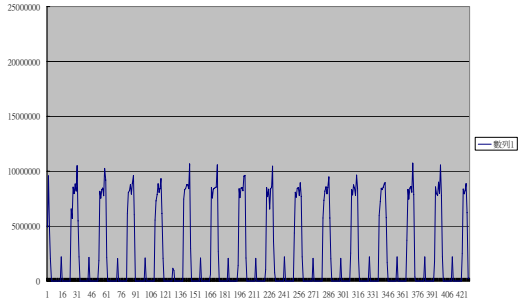
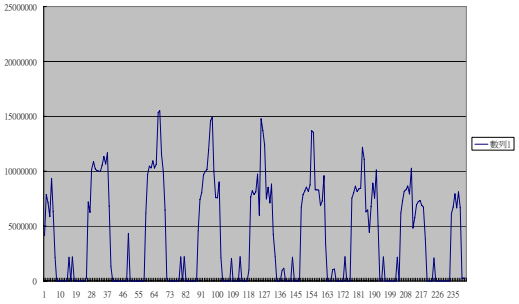
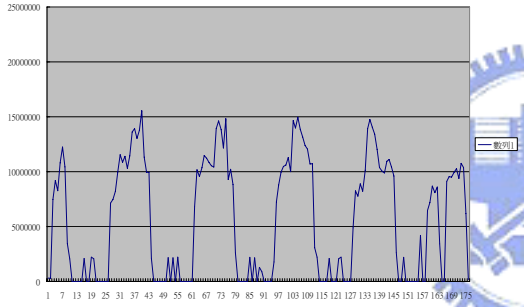
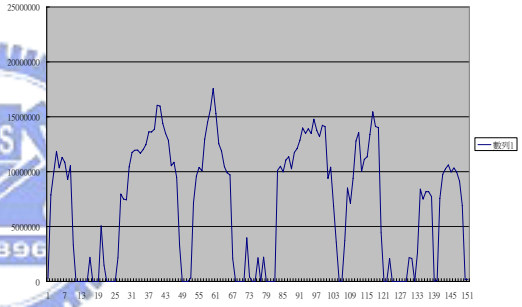
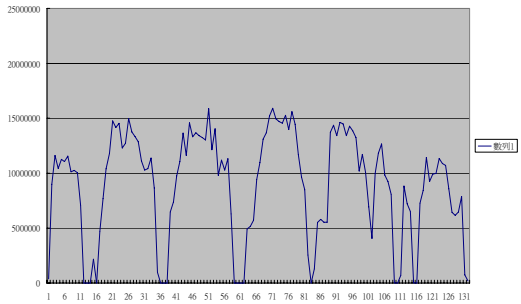
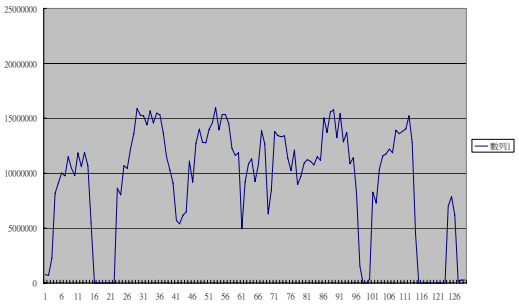


圖 5-12 高檔案傳輸流量實驗完成總時間圖

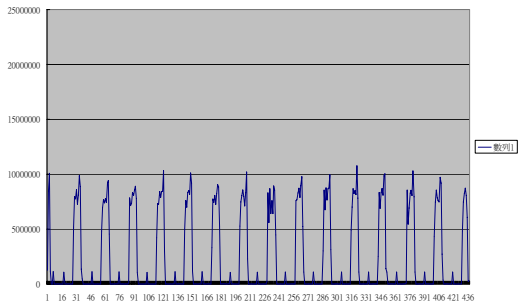
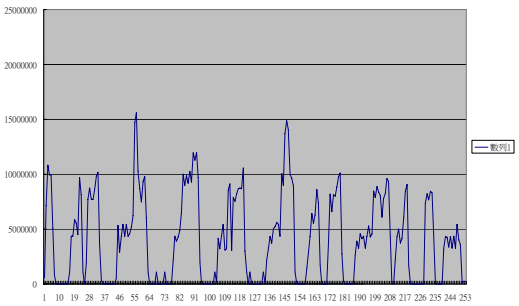
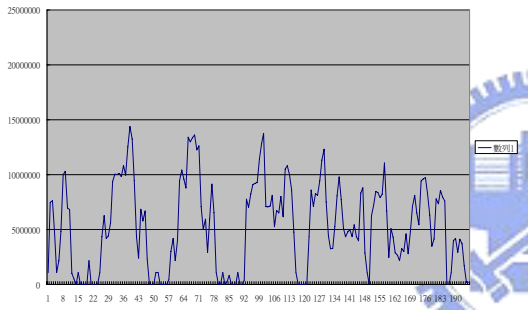
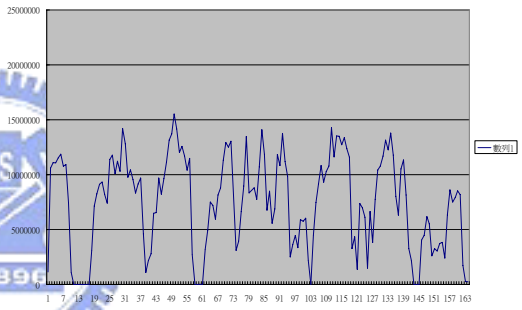
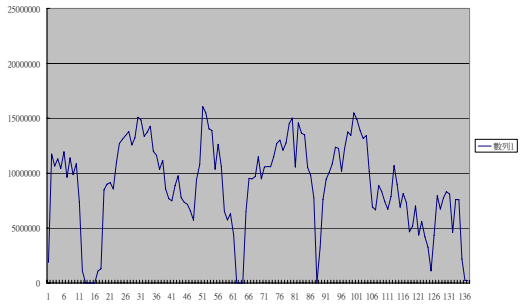
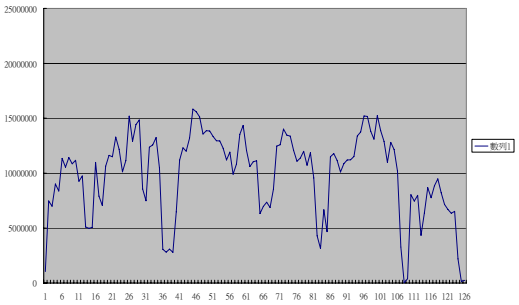
表格 5-4　較小檔案傳輸流量實驗需求端流量表(4MB)(對照組)

1 台電腦參與運算	2 台電腦參與運算
	
3 台電腦參與運算	4 台電腦參與運算
	
5 台電腦參與運算	6 台電腦參與運算
	
註：Y 軸最大值為 25000000 bits/sec	

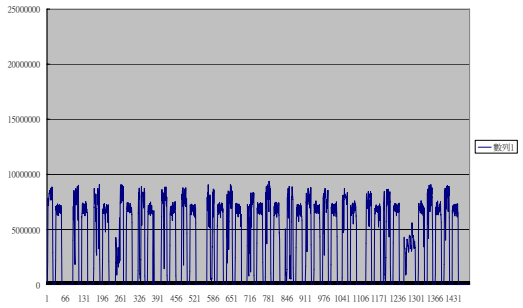
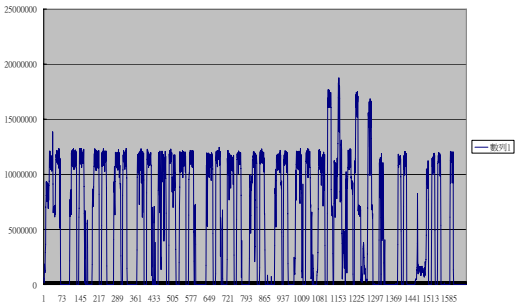
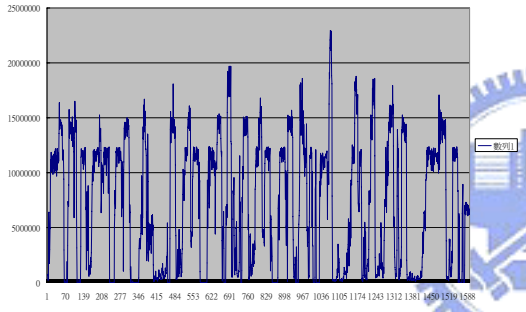
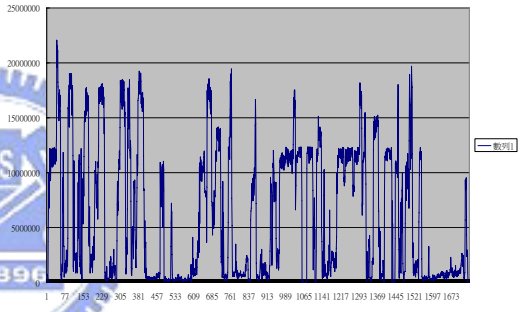
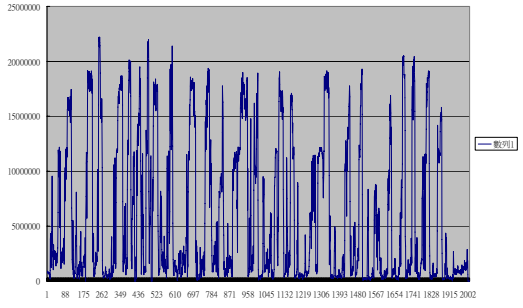
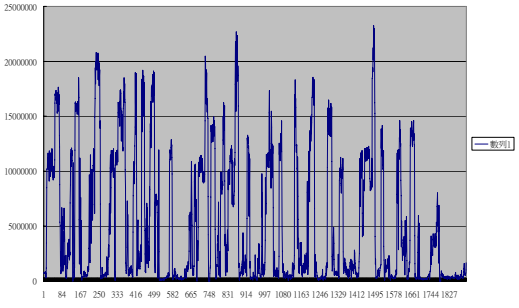
表格 5-5　較小檔案傳輸流量實驗需求端流量表(2MB)(對照組)

1 台電腦參與運算	2 台電腦參與運算
	
3 台電腦參與運算	4 台電腦參與運算
	
5 台電腦參與運算	6 台電腦參與運算
	
註：Y 軸最大值為 25000000 bits/sec	

表格 5-6　較小檔案傳輸流量實驗需求端流量表(1MB)(對照組)

1 台電腦參與運算	2 台電腦參與運算
	
3 台電腦參與運算	4 台電腦參與運算
	
5 台電腦參與運算	6 台電腦參與運算
	
註：Y 軸最大值為 25000000 bits/sec	

表格 5-7　高檔案傳輸流量實驗需求端流量表

1 台電腦參與運算	2 台電腦參與運算
	
3 台電腦參與運算	4 台電腦參與運算
	
5 台電腦參與運算	6 台電腦參與運算
	
註：Y 軸最大值為 25000000 bits/sec	

高檔案傳輸流量實驗結果請見圖 5-12以及表格 5-7。首先，從完成時間的趨勢圖來看，高檔案傳輸流量的環境之下，CPS的運作無法透過更多台電腦的參與來縮減總完成時間。在流量分析上，透過流量表的比較，也可以看到隨著參與電腦的漸增，需求者端的網路流量也逐漸上升；和小檔案傳輸的對照組相比較，網路流量更加顯得繁忙，甚至趨近於需求者電腦的網路頻寬上限。因此，圖 5-12的不規律結果，可以歸咎為網路存取過於雍塞所造成的。

為了近一步證明圖 5-12的結果是因為網路過於繁忙，試著延長運算端的等待時間，由原先的十秒增加到兩百秒，來減輕實驗對於需求者端網路存取的頻繁程度。實驗的結果見圖 5-13，可以看到隨著參與電腦的增加，所需時間會逐漸減少，這和之前運作小檔案傳輸的對照組實驗所得的結果相似。因此，網路存取的繁忙程度是影響完成時間的重要因素。

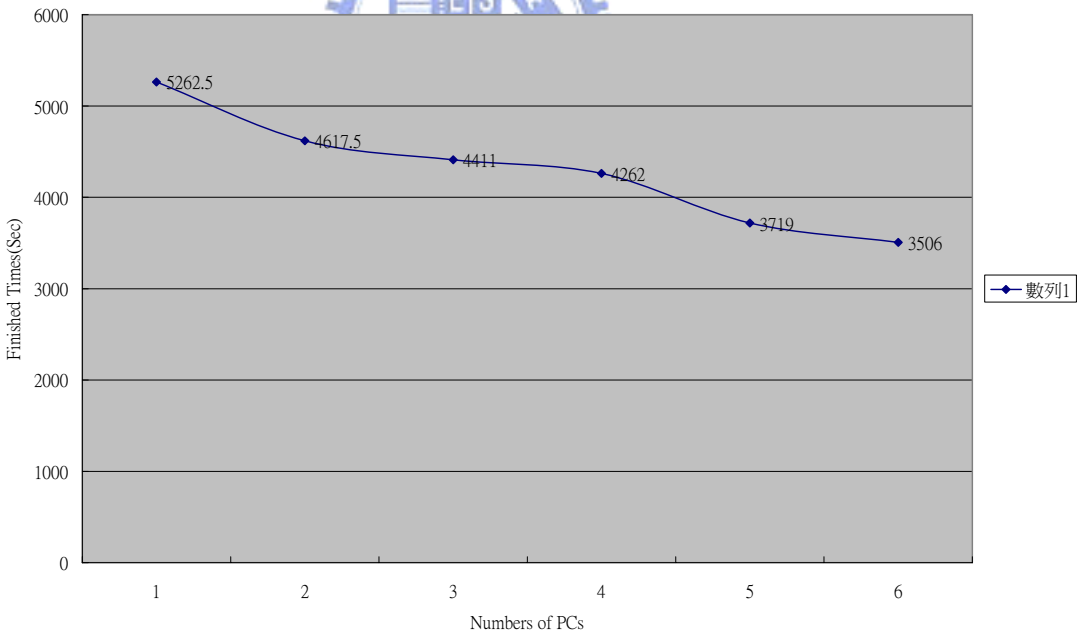


圖 5-13 延長等待時間為 200 秒後的完成時間圖

實驗過程之中，也發現了許多以往不曾發生的例外情況。例如，檔案無法完全正常回傳或者運算端程式無法回傳檔案而停止運作的情況。因此，可以研判在

高檔案傳輸流量的環境之下，運作時間將不再取決於參與電腦的多寡，反而更加依賴網路頻寬的大小以及需求者端電腦的效能，明顯和當初 CPS 建立的精神不符。

由這些實驗結果可以知道，CPS 架構並不是適合處理高檔案傳輸流量的環境。過大的網路流量以及過於頻繁的存取，對於需求者來說，會造成其負擔的增加，會造成工作運作上的瓶頸，甚至頻繁地產生例外情況使的工作效率降低。在實際應用 CPS 架構時，應該避免這種頻繁地傳輸大檔案的環境。

6. 結論與未來展望

6.1. 結論

本研究提出了 CPS 架構，利用 Web Services 的非緊密結合以及開放式標準，突破了現行 P2P 分散式運算環境彈性的不足；導入 BPEL 則賦予了 P2P 架構實驗設計上的彈性以及工作流程控管的能力；在信賴網路中實行，則提升了中小企業在施行分散式運算環境上，安全性和動機上的立場和保證。

CPS 架構提供了一個以 Web Services 為基礎的輕量級 P2P 運算能力分享環境。適用於傳輸量小的可批次化的需高運算能力的執行程式，在組織的信賴網路之中實行，為中小型組織提供了一個運用組織內閒置電腦運算能力的可行方案。同時兼具了圖形化的開發與管理環境以及工作流程控管的能力，不影響運算端的日常作業。簡單使用且能 24 小時運作的 P2P 協同運算環境，可以有效利用老舊電腦的運算能力，減少組織為因應運算需求增高而添購設備的成本支出。

6.2. 未來展望

6.2.1. 工作進度管理

目前運算端程式的 Roll-Back 機制採用重作的方式，主要的目的在於保守地確定契約被達成而已，卻是一個浪費運算資源的舉動。因此，要如何有效率地做工作進度管理。例如當程式被關閉之後，能記錄目前的進度狀態；當重新執行運算端程式時，便能夠較有效率的 Roll-Back，甚至能夠接續之前的進度繼續運算，資源才不會被浪費。工作進度管理兼具資源合理使用以及效率提升的好處，是今後值得研究的方向之一。

6.2.2. 流程最佳化

具有工作流程控管能力之後，如何使流程最佳化，是一個容易被提起的部份。動態派工機制應該是一個可行的方式：以一個流程點來看，參考圖 6-1。有ABCD代表四個運算端，圖中每一個時間區塊代表該運算端執行一個工作的所需時間。圖中狀況表示在需求者還有兩個工作時，A、D分別向需求者要求新的工作。在循序式(round robin)派工機制的環境下，需求者會各分給A、D一個工作，其狀態如圖中（a）所示；倘若透過動態派工機制，根據之前的A、D運算效能來估計如何派工，工作完成時間會最佳。結果就會如圖中(b)所示，把兩個工作都派給D運算。由圖 6-1可以看出，使用動態派工，工作完成時間將會優於循序式派工的情況。利用動態派工，可以讓單流程點的工作完成時間最佳化。

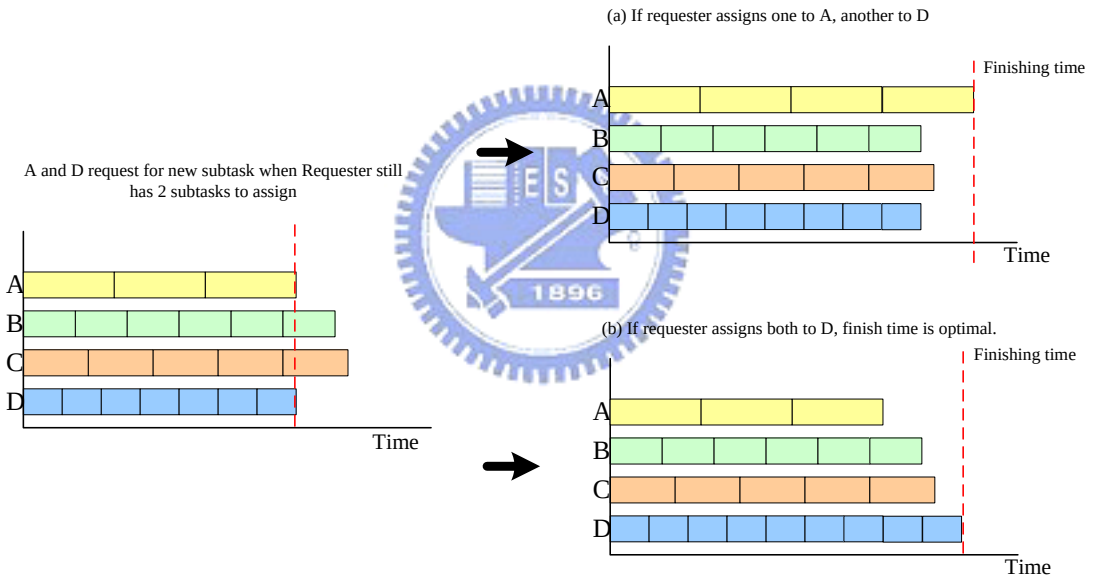


圖 6-1　單流程點最佳化派工示意圖

同時，CPS 既然可以運作平行、相依程序構築而成的實驗流程，如何最有效率地的執行完整個複雜流程實驗，是值得探討的一個議題。相信透過監控參與運算電腦的運算能力來做動態的派工，應該是一個可行的方式，這是日後值得加強的部份。

6.2.3. 整合網路資源的虛擬機器

另外，若能換過來把整個網路當成一個虛擬機器，利用 BPEL 作為外界與虛擬機器、甚至虛擬機器內部元件之間溝通的一種媒介，建構出一整個虛擬機器運作的邏輯。透過這種方式，整個虛擬機器所包含的資源將能被統籌管理使用；屆時，無論是程序或者是執行緒，也能透過虛擬機器核心的運作分配給網路中閒置資源運作。這種方式將突破本研究目前只能執行可批次化的需高運算能力的情況，也是未來值得研究的方向之一。

參考文獻

[1] Dejan S. Milojicic “Peer-to-Peer Computing” HP Laboratories Palo Alto March 2002

[2] Alfred W. Loo “The Future of Peer-to-peer Computing” Communications of The ACM September 2003 Vol.46,No.9 P.57-.61

[3] 喻瀚寬、蔡澤銘 ”解析 Web Service 的技術內容與意涵----打破平台疆界 啟動 internet AP 整合新視界”，軟體產業通訊，第三十七期，2001/06

[4] Dan Gisolfsi, Web Services Architect , Solutions Architect, IBM jStart Emerging Technologies 01 Apr 2001
http://www-106.ibm.com/developerworks/webservices/library/ws-arc1/

[5] Biplav Srivastava, Jana Koehler “Web Service Composition - Current Solutions and Open Problems”

[6] Tony Andrew, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein et al. “Business Process Execution Language for Web Services”
http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

[7] Holt Adams “Asynchronous operations and Web services: A primer on asynchronous transactions”
http://www-106.ibm.com/developerworks/library/ws-asynch1.html 2002

[8] Oracle Lab Segments “Oracle BPEL Process Manager Training”
http://otn.oracle.com/bpel August 2004

[9] 曾立信 ”小波封包轉換的浮水印對數位影像所有權之確認” 國立交通大學 碩士論文 2003

[10] Nikola Milanovic and Miroslaw Malek“Current Solutions for Web Service Composition” IEEE INTERNET COMPUTING NOVEMBER • DECEMBER 2004 P.51-59

[11] Matt Powell “Asynchronous Web Service Calls over HTTP with the .NET Framework” http://msdn.microsoft.com/library/en-us/dnservice/html/service09032002.asp?frame=true September 9, 2002

[12] Matt Powell “Server-side Asynchronous Web Methods” http://msdn.microsoft.com/library/en-us/dnservice/html/service10012002.asp?frame=true October 2, 2002

[13] “Asp.Net 程式設計徹底研究” 董大偉著 文魁資訊股份有限公司出版

[14] “Building XML Web Services for the Microsoft .NET Platform” Short Microsoft Press

[15] “Visual Basic .Net 強力調校” 彭明柳譯 博碩文化出版

[16] “使用 VB.Net 與 C#.Net 開發 XML Web Services 與伺服器元件” 啟真工作室 Microsoft Press

[17] Chris Peltz “Web services orchestration and choreography IEEE Computer Volume:36, Issue:10 Oct. 2003

[18] Boualem Benatallah and Quan Z. Sheng “The Self-Serv Environment for Web Services Composition” IEEE INTERNET COMPUTING JANUARY • FEBRUARY 2003

[19] HEATHER KREGER “FULFILLING THE WEB SERVICES PROMISE” COMMUNICATIONS OF THE ACM June 2003/Vol. 46, No. 6

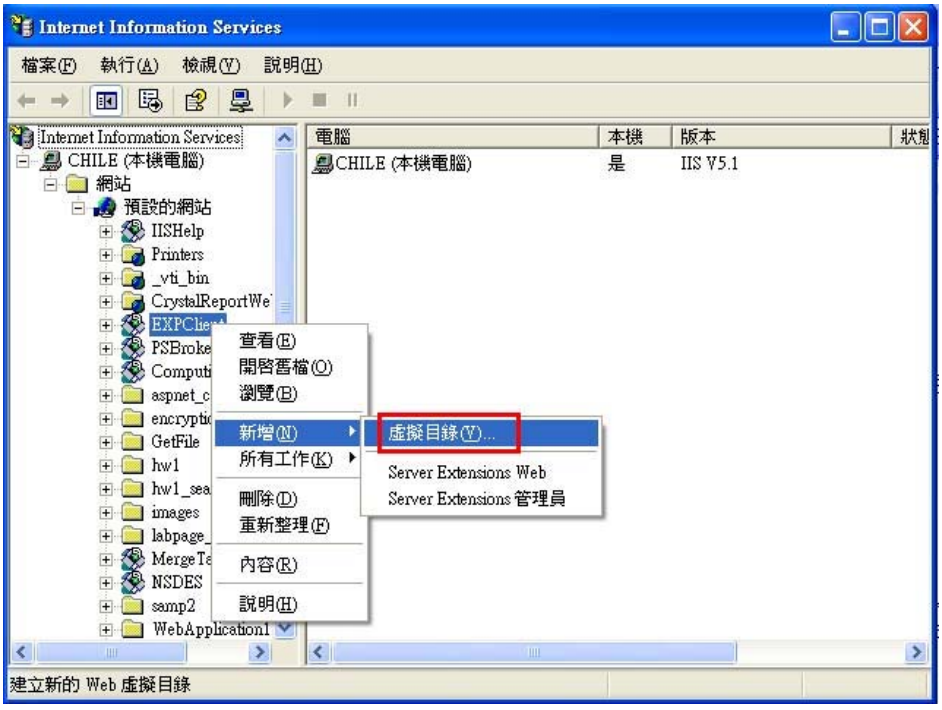
附錄(一)　Computing Power Services 系統使用手冊

軟體安裝

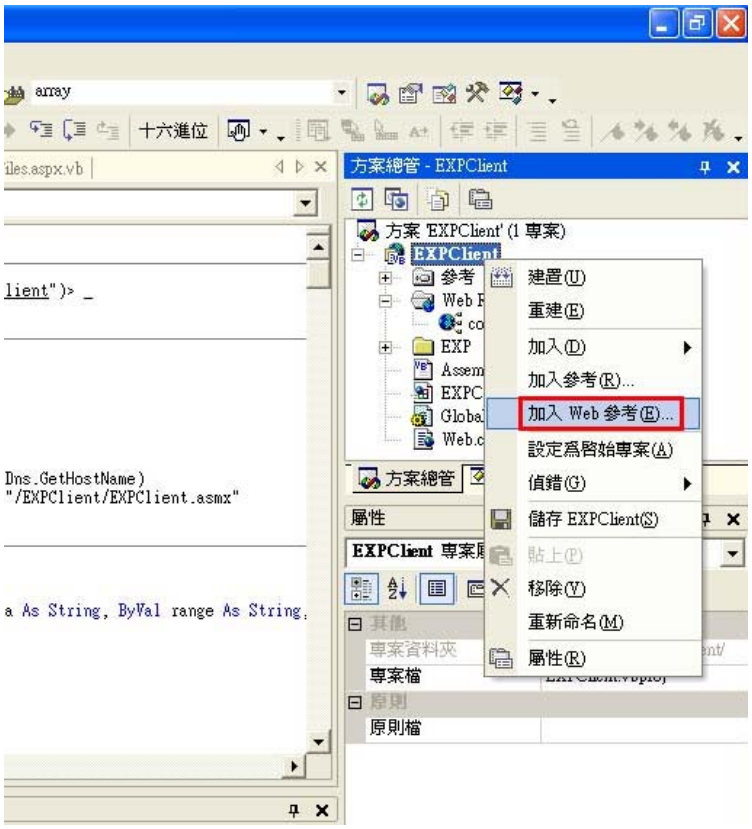
- Microsoft Visual Studio .Net 2003
- Microsoft .Net Framework
- Microsoft SQL Server 2000
- Oracle BPEL Process Manager 2.0
- Eclipse 3.0（若使用 3.1 板，在安裝 BPEL Process Manager 時會發生問題）
- J2SDK1.4.2 以上版本
- ActivePerl-5.6.1.638（若使用 5.8.7 版，會發生不支援的問題，此軟體為針對「多重小波轉換濾波器浮水印嵌入」實驗，生成實驗檔所使用）

系統建置

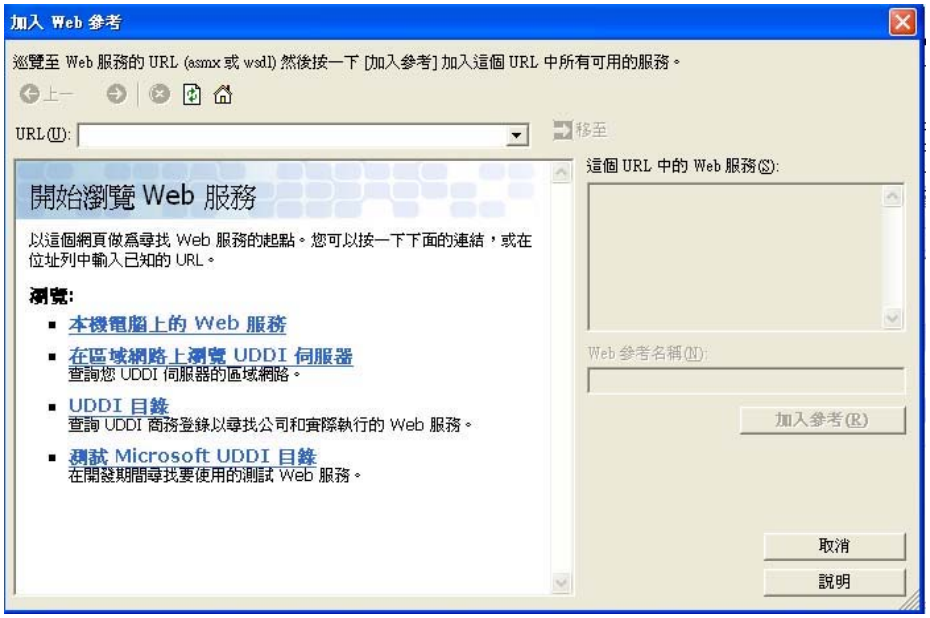
1. Microsoft .Net 系統主程式
   - 需求者端（Requester）－ EXPClient
     - 在 C:\Inetpub\wwwroot\ 下建立資料夾 EXPClient。
     - 控制台→系統管理工具→Internet Information Services→新增虛擬目錄。開啟服務。



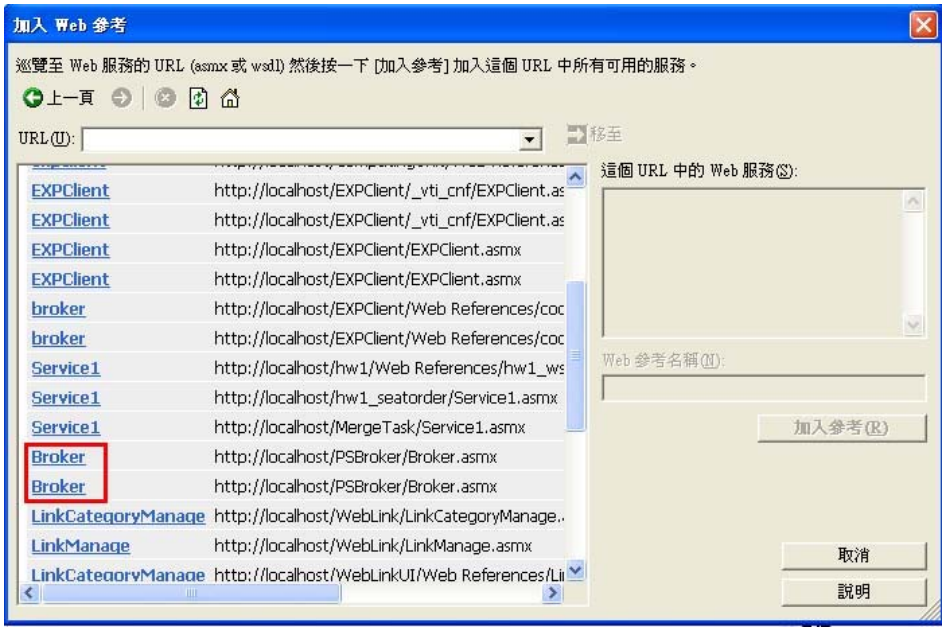
- 進入 Microsoft Visual Studio .Net 2003 開啟 EXPClient 專案，並重新編譯。
- Web Reference_coordinator依照協調者端Coordinator的安裝位址進行更新，例如：http://140.113.72.173/psbroker/broker.asmx
  - EXPClient→滑鼠右鍵→加入 Web 參考



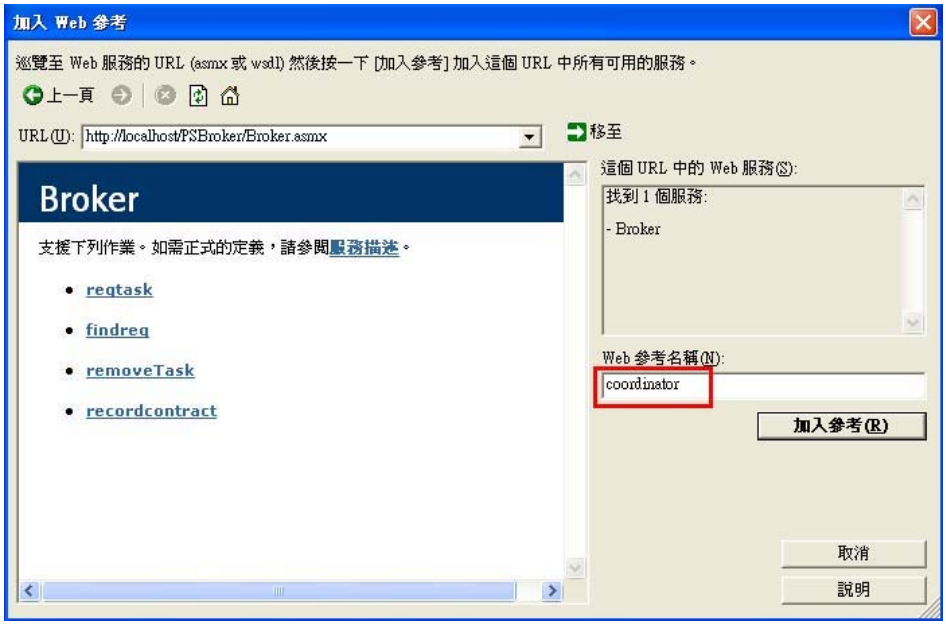
  - 依照協調者端 PSBroker 的安裝位址進行更新，選擇「本機上的電腦服務」(PSBroker 和 EXPClient 架在同一主機上）或直接輸入 URL 位址。



➢ 選擇協調者端 PSBroker 位址

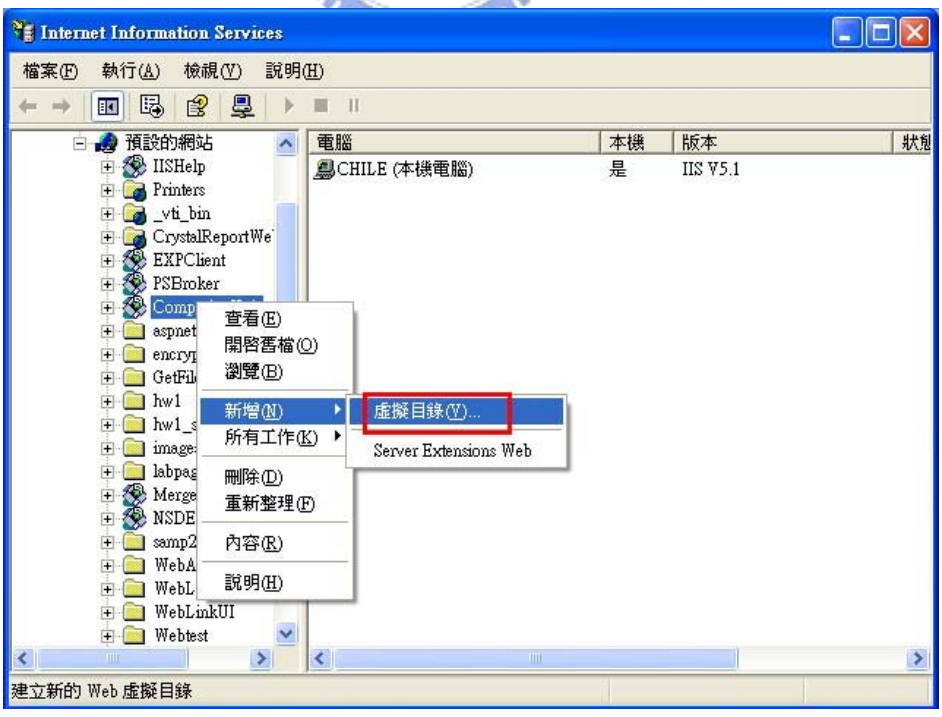


➢ Web 參考名稱命名



- 可依據使用者需要更改程式碼中的帳號密碼。

- 運算者端（Computing Unit）— ComputingUnit
  - 在 C:\Inetpub\wwwroot\ 下建立資料夾 ComputingUnit。
  - 控制台→系統管理工具→Internet Information Services→新增虛擬目錄。開啟服務。



- 進入 Microsoft Visual Studio .Net 2003 開啟 ComputingUnit 專案，並

重新編譯。

- Web Reference_Broker依照協調者端Coordinator的安裝位址進行更新，例如：http://140.113.72.173/PSBroker/Broker.asmx

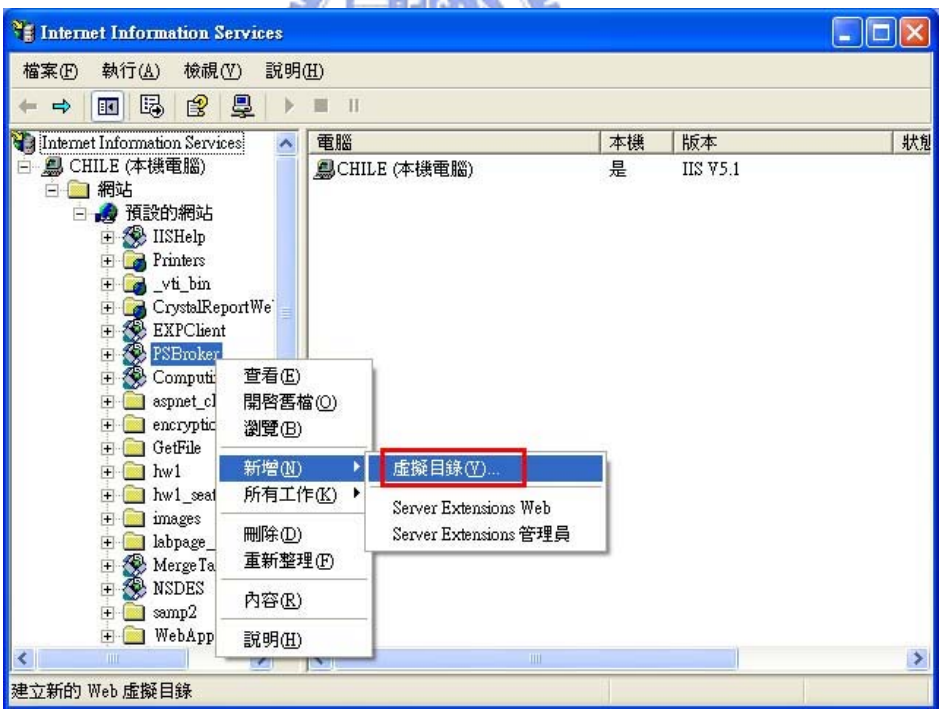
（方法同上 EXPClient）

- Web Reference_EXPClient依照需求者端Requester的安裝位址進行更新，例如：http://140.113.72.173/expclient/expclient.asmx

（方法同上 EXPClient）

- 可依據使用者需要更改程式碼中的參數。

- 協調者端（Coordinator）－ PSBroker
  - 在 C:\Inetpub\wwwroot\ 下建立資料夾 PSBroker。
  - 控制台→系統管理工具→Internet Information Services→新增虛擬目錄。開啟服務。



- 進入 Microsoft Visual Studio .Net 2003 開啟 PSBroker 專案，並重新編譯。
- 可依據使用者需要更改程式碼中連接資料庫的位址、帳號、密碼。
- 當編譯完成後，便可將 C:\Inetpub\wwwroot\ComputingUnit\bin 下的 ComputingUnit.exe 發送給即將進行運算的電腦。

2. Oracle BPEL Process Manager 程式

- TaskUnit 非同步回報 － testtasj.jsp
  - 啟動「UserTasks」BPEL Process 範例。
    - 開啟 command prompt（執行 cmd）
    - 轉換路徑到 C:\orabpel\samples\tutorials\110.UserTasks

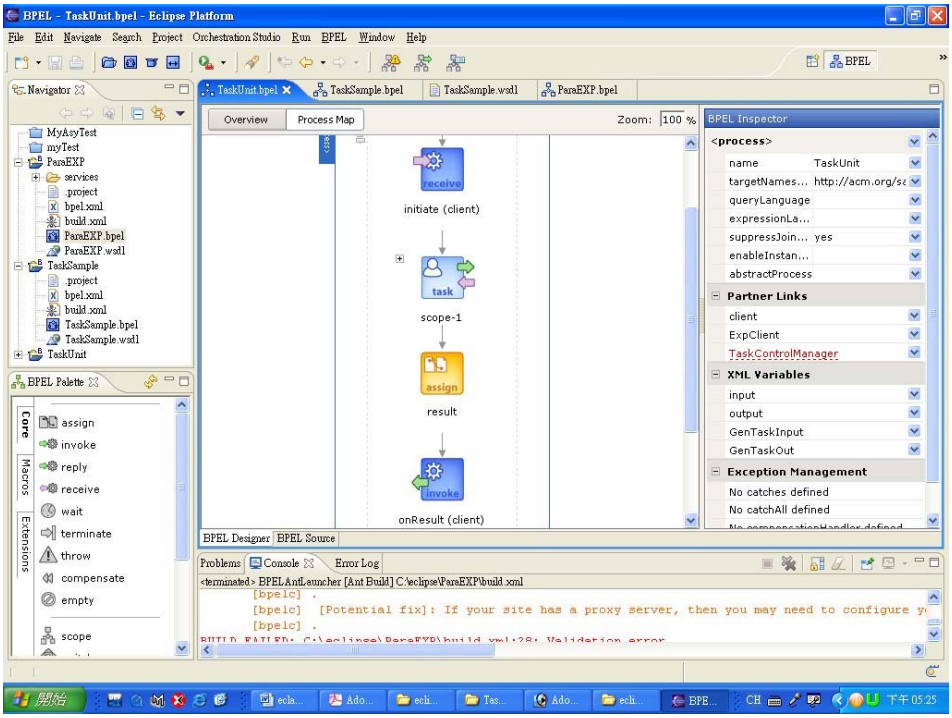
      > cd C:\orabpel\samples\tutorials\110.UserTasks

    - 執行 obant 指令

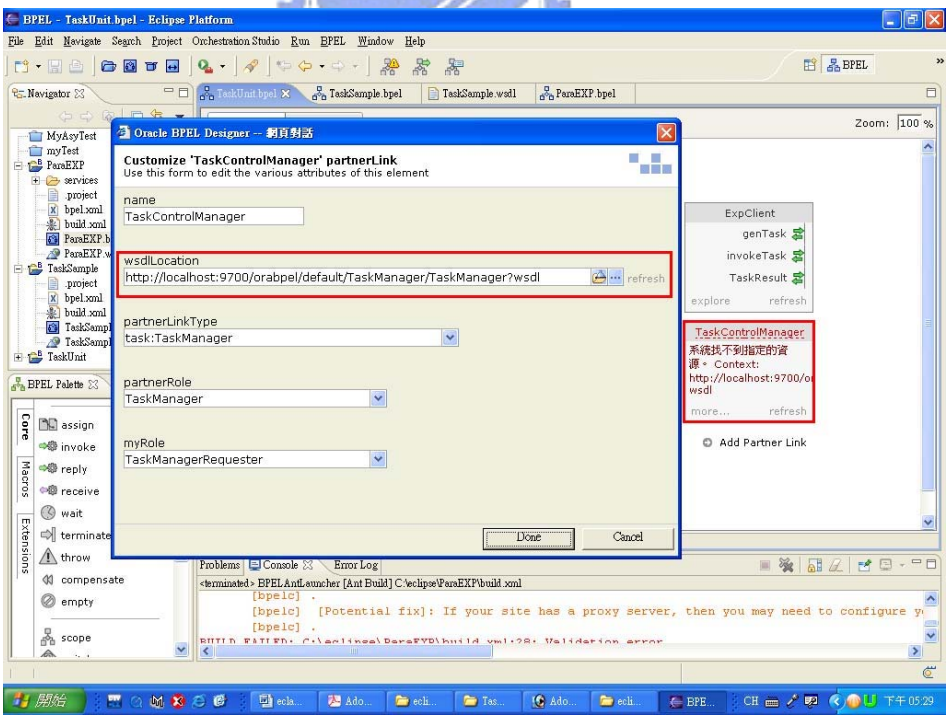
      > obant

  - 進入 C:\orabpel\system\appserver\oc4j\j2ee\home\applications 會看到 TaskSampleUI 資料夾。
  - 將 testtasj.jsp 置於 C:\orabpel\system\appserver\oc4j\j2ee\home\applications\TaskSampleUI\TaskSampleUI 中。

- BPEL Process 實驗程式 － TaskUnit（Sequence Process）
  - 將「加入浮水印的 Raw 檔」和「Jpeg2000 圖檔解出浮水印」以 Sequence Process 方式處理。

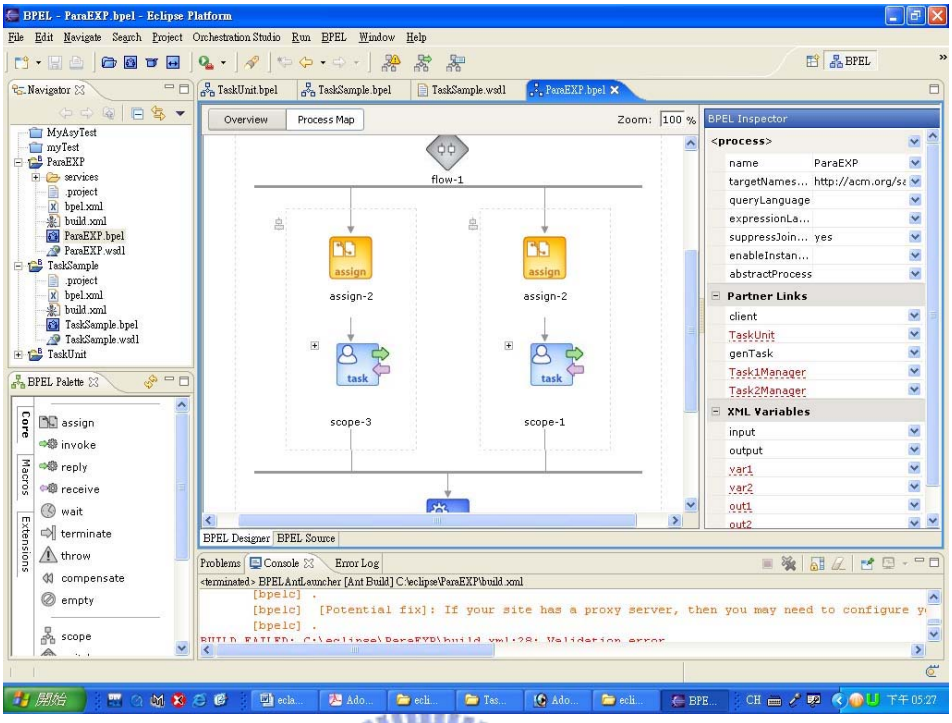


- 開啟 Eclipse→File→Import→Existing Project into Workspace→ Browse…→選擇 TaskUnit 位置
- 重新設定 Partner Link，更新 Link 位址。

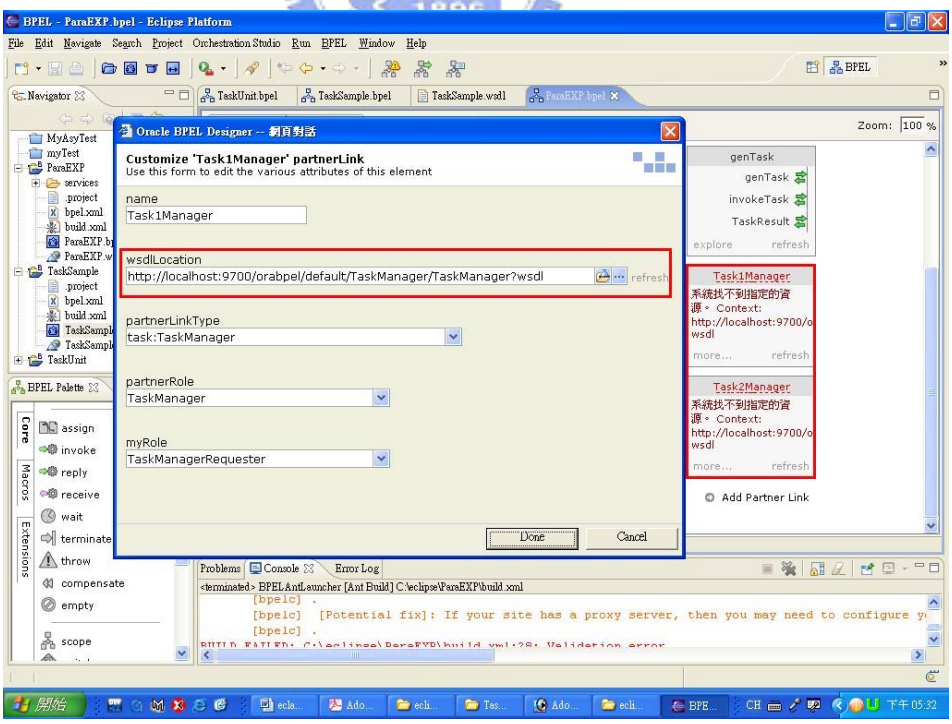


- BPEL Process 實驗程式 － ParaEXP（Parallel Process）
  - 將「加入浮水印的 Raw 檔」和「Jpeg2000 圖檔解出浮水印」以 Parallel

Process 方式處理。



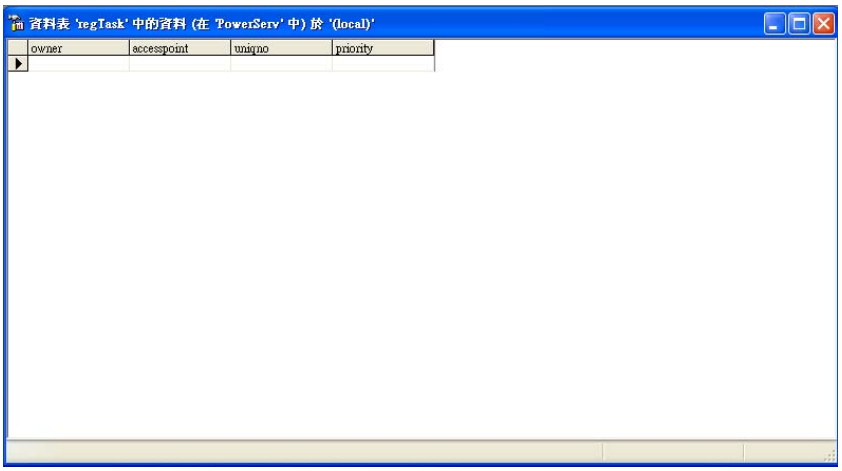
- 開啟 Eclipse→File→Import→Existing Project into Workspace→ Browse…→選擇 ParaEXP 位置
- 重新設定 Partner Link，更新 Link 位址。



3. Microsoft SQL Server 資料庫 PowerServ

- 資料表 － regTask
  - 註冊任務

（owner , accesspoint , uniqno , priority）

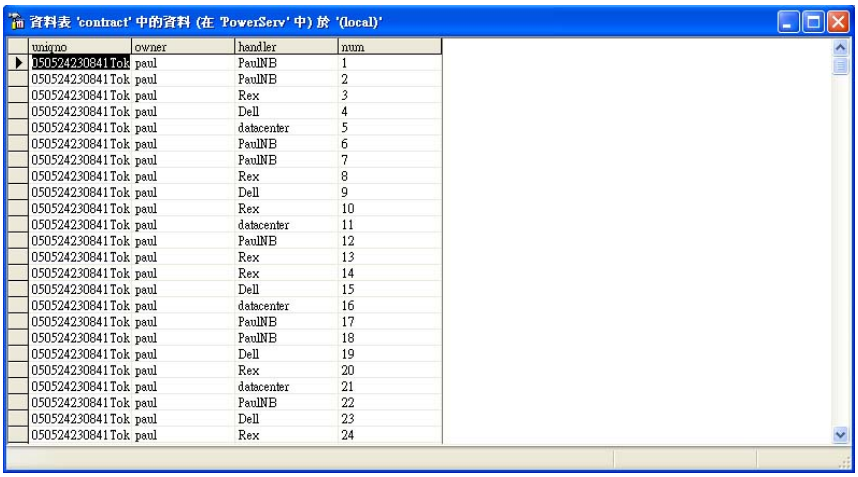


- 資料表 － Member
  - 需求者端 Requester 的始用者註冊

（ID , password）

- 資料表 － contract
  - 儲存完成任務的資訊

（uniqno , owner , handler , num）



- 資料表 － con_decript
  - 方便資料查詢而存在的資料表，和實驗進行無直接的相關。

（uniqno , description）

附錄(二)　著作

[1] Chen-Sheng Wang, Po-Yu Yang, Min-Jen Tsai　“A WEB-SERVICES-BASED P2P COMPUTING-POWER SHARING ARCHITECTURE”, ICEB 2005