國立交通大學

資訊科學系

碩 士 論 文

服務導向之分散式網頁表達架構的設計與實作

A Service-oriented Component-based Presentation Framework

研 究 生：張舜禹

指導教授：陳俊穎　教授

中 華 民 國　九 十 三　年 六 月

服務導向之分散式網頁表達架構的設計與實作

A Service-oriented Component-based Presentation Framework

研 究 生：張舜禹　　Student：Shun-Yu Chang

指導教授：陳俊穎　　Advisor：Jing-Ying Chen

國 立 交 通 大 學

資 訊 科 學 系

碩 士 論 文

A Thesis

Submitted to Institute of Computer and Information Science

College of Electrical Engineering and Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer and Information Science

June 2004

Hsinchu, Taiwan, Republic of China

中華民國九十三年六月

服務導向之分散式網頁表達架構的設計與實作

學生：張 舜 禹　　指導教授：陳 俊 穎 博士

國 立 交 通 大 學

資 訊 科 學 系

摘 要

在一般 N-tier 的網頁伺服器架構中，表現層介於使用者和後端的服務之間，將後端服務的介面轉換為使用者可使用的網頁介面。隨著系統的複雜度增加，後端服務的架構已漸朝向分散式且元件化的方向演進。但表現層通常還是以集中式的方式來開發，而缺乏有效的分工。本論文探討將表現層分散化的好處及可行性，並針對表現層設計一種分散式元件式架構。在這架構下，不同組織可獨立發展服務導向之表現層元件，而網頁設計者可因不同需要調整這些表現層元件並整合在同一個網頁伺服器下，達到分工的目的。

A Service-Oriented Component-Based Presentation Framework

Student：Shun-Yu Chang　　Advisors：Dr. Jing-Ying Chen

Department of Computer and Information Science
National Chiao Tung University

ABSTRACT

In a typical n-tier architecture that is widely used today for Web applications, the presentation tier lies between user and the back-end tier, transforming the interfaces of the back-end services into Web-based interfaces accessible by human. As the complexity of Web applications increases, back-end services have also evolved from centralized architecture to distributed and component-based one. Furthermore, these back-end services may be supported by different companies each specializing on particular business domain. On the other hand, the presentation tier is still developed in a centralized way. As a result, web application developers are responsible of constructing most of the presentation logics linking various back-end services, leading to improper division of labor and unnecessary software complexity. In this thesis, we propose a distributed, component-based presentation infrastructure on which different organizations can develop presentation services wrapping their own business services, respectively. These presentation services can be customized and combined flexibly by others to build their own Web applications. We argue that with this service-oriented, component-based approach, Web application development can be greatly simplified.

誌　　謝

二年的研究生涯即將結束，這段期間，在我的指導教授，陳俊穎老師的指導之下，讓我得以充實專業領域上的學識，在求學與做事上，他讓我學到了採用積極且謹慎的態度，對於我走在資訊科技這個領域裡，影響甚深。感謝陳俊穎老師在這二年的指導，在論文即將完成之際，在這裡特別對他表達感謝。除了陳俊穎老師以外，尚有我的論文口試委員袁賢銘教授、劉興民教授在本篇論文上給予的指導和提供寶貴的意見，使得能夠讓本篇論文更為完備。

感謝我的同窗摯友，陪伴我一起度過研究生涯的好伙伴，阿吉，訓宏，建宏，以及多位學弟們。有了你們常常從早到晚的陪伴和在學業上的指點、協助、和支持，才能讓我更順利，更有效率地把論文完成。

最後還需要感謝的是我的家人以及時常在我身旁鼓勵我的好友。感謝你們的付出和關心。若沒有你們默默的付出與全力的支持，我將沒有今天如此的成就，在此對於所有一起陪我走過這段歲月的家人和朋友致上最深的謝意。

張舜禹　謹誌　2004年6月

於交通大學研究生室

目　　錄

圖 目 錄

表 目 錄

第一章　緒論

全球資訊網 (World Wide Web) 演變至今，已成為最普遍的資訊取得媒介，經由簡易的超文件傳輸協定 (HTTP)、超文件標示語言 (HTML) ，及其中的超連結位址 (URL)，WWW 能夠連結遠端的多媒體格式的檔案，將全球的資訊連結在一起，達到資源的共享和整合。如今，瀏覽器已成為一致性被普遍使用的工具，而伺服端的系統、服務也仰賴著全球資訊網提供網頁介面給使用者隨時隨地地使用。

簡單而言，HTTP 為提供使用者來存取網頁的協定。此協定基本上是 memory-less，一次的請求，接著一次相對的回應，然後結束連線，無法紀錄狀態。如有下一次的請求，則需要重新建立連線。因為這樣的特性，它無法直接適用於動態的及分散式的計算，提供元件、服務之間的溝通協定。簡言之，它本身是為了靜態文件的取得而設計，而非動態的文件。

然而，靜態的網頁已不能符合現今許多使用者及系統的需求。許多系統都需要動態的網頁，以達到與使用者互動的效果。為了達成動態內容的產生，目前的方法為採用像 JSP 等技術[15]，能夠依據客戶端的請求，狀態，連線紀錄和其它運算邏輯來產生動態的內容。如常見的購物網站 (Shopping Cart)、醫療服務系統、圖書館借閱查詢服務，等等。這些大型系統大多是採用標準的 3-tier 架構，整個系統的網頁都集中在中間層 (Middle-tier)，而在這一層都是經由同一個組織，同一組人採取集中式的設計，開發，部署，和持續地更新[18]。

例如，像在企業和許多組織內部被廣泛使用的 Portal 技術，它提供了一個統一的入口網站，能使不同身份的使用者，存取原本許多獨立存在的系統，不同使用者所接觸到的資訊都依據他們的身份或是個人喜好而有所不同。但在表達層的設計開發中，大部份的元件仍沒有成為一個服務，仍需為不同的組織或 Portal Server 提供一份元件讓他們使用或擴充。這些元件如 UI 元件 (User Interface Components)、使用後端服務的應用程式 (Web Service Client Application)、網站樣版 (Site Template)、一系列的設計元件和顏色佈景 (Theme) 等等，可經由一中心網站，如微軟提供的 Web Component Directory，來將這些元件共享於開發者之間。

這種架構雖然具有良好彈性且益於管理，但隨著系統複雜性的增加，有著潛在的問題。首先，動態豐富的網頁內容來自於多樣的不同領域，例如 yahoo、msn 等入口網站[14]。如果要建立複雜的網站，可能需要不同領域的專業人員。不同內容不是自己企業內部來開發就是委託由專門領域的人來完成。若採用委託外部的方式，則希望被委託者能夠為其製作一份客製化的版本，使得整個網頁集合看起來具有一致性，和相同的風格。而開發完後，通常會將網頁檔案、相關原始碼集中地存放在同一個機器之中；即使是採用經由超文件傳輸協定和超文件標示語言的遠端連結的方式，如果為了要達到客製化，也需要為每個不同的客戶製作一份版本存放在遠端。

此外，網頁原始碼等資源的智慧財產權需要被保護時，網頁服務提供者可能不希望把網頁等原始碼複製一份給對方。而希望採用連結遠端網頁的方式。問題在於，簡單的 HTTP 及 HTML 沒有支援這樣動態客製化的情形，使得服務提供者需要為每個不同的客戶開發出一份不同樣式的網頁內容。

本篇論文要探討的是在表達層中如何在元件式的要求下達到客製化的目的，能夠達到動態的客製化能有效降低成本及系統複雜度，但為了達成這樣的目標，有一些議題需要被謹慎地思慮。像是分散式服務間的合作機制、網頁原型、效能考量等等。

針對此問題，本論文提出了一個架構，在此架構中，不同的網頁內容提供者以服務的形式提供網頁內容，稱作網頁表達服務 (Presentation Service)。不同網頁表達服務可經由此架構互相合作以產生客製化的網頁內容，而不需要讓網頁設計者去使用或知道它們後端內容產生的細節。

以下我們利用一個股票管理服務網頁介面的例子，來說明此分散式表達層架構的概念。假設有一家網路公司 C，他想要提供個人的證券投資組合管理服務，網頁看起來如圖 1 ，網頁介面包含了一個使用者的股票持有表，當使用者點選了某公司的連結，網頁會在左下角顯示此公司的概況和在右下角顯示它的股價走勢圖。

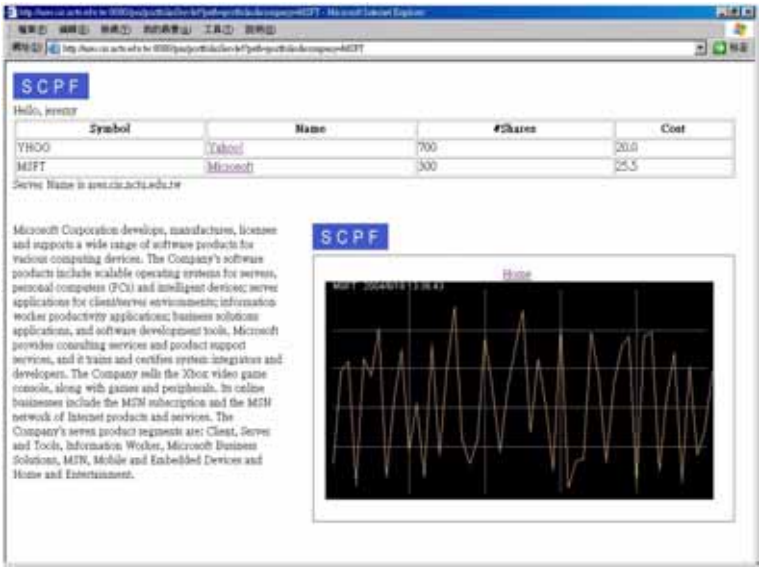

圖 1 Portfolio Example

C 公司，本身為它的客戶維護股票管理的資料庫和其相對應的網頁介面，也就是網頁上方統整持有股票的表格。至於網頁下方，公司概況和股票價格曲線圖，C 公司各自使用其他公司建立的網頁表達服務，假設分別為管理各上市公司資訊的 A 公司和善於製圖的 B 公司。換句話說，有關公司概要和股價走勢圖的網頁內容分別透過網路直接從 A 公司和 B 公司的網頁表達服務而得到。

為了擁有一樣的外觀 (Look and Feel)，C 公司希望能客製化這些使用外部表達服務提供的頁，以讓整個網站看起來有一致性，就如同 A 公司和 B 公司特別為 C 公司設計了這些網頁。

例如，當使用者點選右下角股價走線圖頁中的連結，新的頁面出現的位置以及連結的內容能夠被 C 公司所控制。如果有一個首頁的超連結，它應該連到的是 C 公司的首頁，而不是表達服務提供者的首頁。

傳統的方法中，除非提供的網頁是對所有使用者都一樣的，否則 A 公司和 B 公司必須為 C 公司做一份客製化網頁的設計。本論文提供了一個組織以及客製化網頁的架構，在此架構中，表達服務提供的頁被視為許多元件，而元件應該被客製化且可以互相結合，A 公司和 B 公司只提供它們的表達服務而不用煩惱誰將使用它們提供的網頁和如何對這些網頁做客製化。

值得注意的是，HTTP 和 HTML 已經提供一些形式的合作，例如影像連結和

框架 (Frame)，等等，可以讓來自不同來源的內容被結合在一個頁上。但是，在這個例子的情境中，動態的合作機制是需要的，因為表達服務是一個獨立自主的服務，而不是為每個使用它的主機提供一份客製化的網頁內容。

在分散式的網頁表達架構中，網頁設計者設計一張網頁的時候就好像把許多實用的小元件組裝到網頁上，在本論文中是以一個頁為被組裝的單元，整張網頁就看起來好像有許多塊元件被放在上面，而在建立一個網站的時候，可以把其他的網頁表達服務提供的網頁，引進來使用。就跟實作一套系統或是軟體一樣，把別的組織或社群寫好的堅固元件組裝進來。

第二章 相關背景

本論文所要提的分散式表達層架構，跟一些已存在的用於伺服端網路應用程式發展的表現層架構有相當密切的關係。許多網頁流動的模型和觀念都是相似的。本章先對現今表現層的技術，做一個介紹。接著討論一些現有技術中相似且重要的概念來與本論文所提的架構使用的方法做比較。

第一節 表達層架構概要

全球資訊網在今日已經變成不可少的技術。最重要的原因之一就是到處存在的網頁瀏覽器讓人們存取網際網路上的資訊與服務。在全球資訊網之前的年代，存取這些服務需要不同的應用程式，伴隨著的問題就是不同的安裝配置與管理。

從技術的觀點來看，另外一個相同重要促使全球資訊網成功的原因就是以超文件傳輸協定和超文字標記語言為基礎的架構的簡易性。超文字標記語言做為網頁瀏覽器和網路應用程式間的標準介面。不論伺服端的表現層和後端如何的發展，伺服端與使者間經由網頁的溝通仍是經由超文件傳輸協定。雖然目前伺服端的表達層與使用者溝通都藉由超文件傳輸協定，但伺服端後端的架構和溝通協定，則可自由的發展。

傳統的集中式表達架構中，伺服端的表現層為介於使用者和後端的服務之間，能將後端的服務轉換為較易使用的網頁介面。隨著系統的複雜度增加，後端服務的架構已漸朝向分散式且元件化的方向演進[21]。但表現層通常還是以集中式的方式來開發，而缺乏有效的分工。比如說，要在不同的網頁伺服器提供相同的網頁，就需要在每個伺服器上存放相同的一份網頁。要對相同的後端服務製作網頁介面，就需要在每個網站上放上相同且龐大的網頁。在開發上無法達到有效的分散化。這造成表達層的架構無法被模組化，而變得愈來愈龐大，加上它與後端服務的高耦合力，使得維護更加困難。

雖然使用者能藉由 HTTP 和 HTML 中的連結或框架標籤，來連到遠端網站中的網頁，但這方法所能達到的的合作關係是有限的。並且，在開發網站的時候，由於沒有辦法有效地利用別台網頁伺服器所提供的網頁，使得網頁在開發上沒有

辦法將表達層提升為一種服務，分享給他人使用。

N 層 (n-tier) 的架構大致可分成使用者端，中間層，和後端層。而中間層的功能就是作為網頁瀏覽器和後端層的橋梁。如圖 2所示，對使用者來說，中間層將後端服務的程式介面轉換為圖形化且具親和力的格式，對後端服務來說，中間層把使用者的互動轉換為它們可接受的格式。此外，中間層通常進而被分成表現層和商業邏輯層。表現層主要負責內容的繪製，而商業邏輯層負責了依照各個後端服務特殊的需要，來存取後端服務。商業邏輯層主要是做為那些後端服務的代理者，提供方便的介面讓表現層去使用。

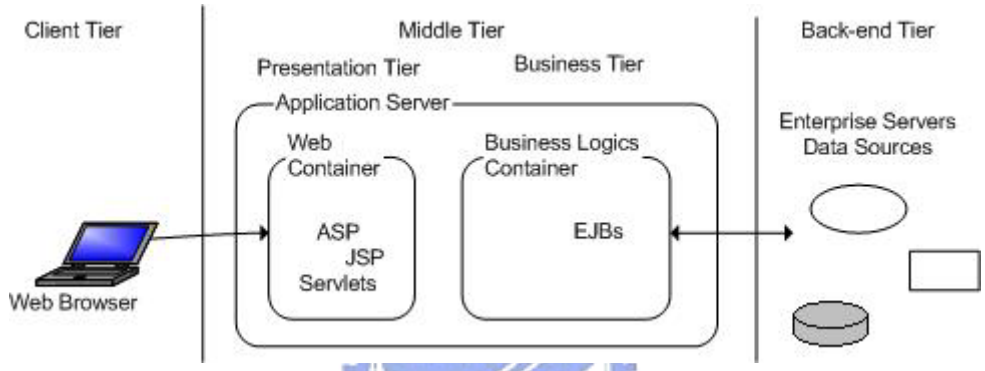

圖 2 3-Tier Architecture

N 層的架構達到層與層之間能夠獨立分開和開發，且每一層的責任劃分清楚，它有效地幫助管理並減輕複雜度。應用程式在伺服端被部署和維護，取代了應付使用者端應用程式在不同機器上安裝與管理的麻煩。一個大的系統，在每一層裡，還應該分成許許多多元件。每一層也應該朝向元件化式的發展，就像一般的應用程式的說計理念一樣。

事實上，許多後端的服務，經由某些中介軟體，已漸漸地成為分散式的架構。而表達層的技術也朝向相似的趨勢在發展。但由於它與後端服務間的高耦合，加上與 HTTP 緊密的結合，使得表達層的抽象化相對困難。

而從企業伺服端應用程式發展的方面來看，其角色可分為三類，分別為網頁編寫設計者 (Page designers)，元件發展者 (Component writers)，和應用程式開發者 (Application developers)。網頁設計者的工作為使用標記語言 (Markup Language) ，如 JSP tags，做一些圖片的設計，組合網頁的內容。元件發展者的

任務是建構可重複利用的使用者介面元件。應用程式開發者的工作為提供伺服端的運算功能，如商業邏輯元件 (model objects)，檢查身份是否正確的邏輯元件，事件處理的元件，等等。在採用 MVC 架構的情形裡，應用程式開發者的角色負責提供了 model 和 controller 兩個部份[5]。

Model2 [5]是 J2EE 藍圖所提出的概念，MVC (Model-View-Controller) 是一個物件設計樣本 (Design Pattern)。MVC1 把 Controller 元件和 View 元件結合在一起，而 MVC2 則是把它們分開，也就是 Model2 的概念。在 Model2 模式中，原本各 JSP 網頁裡 Control 的部份被移到一個中心的 Servlet，由此中心的 Servlet (the controller) 來負責網頁流動的邏輯，如此使得每個 JSP 網頁都只負責內容的描繪 (the view)。Controller 元件可以對 Request 做驗正，執行商業層次的運算，和決定回應的網頁。雖然 MVC 為目前被廣為接受的網頁系統發展模型，當表達層的開發者在設計網路應用程式時，還是無法很好的區分商業邏輯層和表達層。問題在於表達層的開發者必須想辦法來把多樣且不同的後端服務組合進來。而且網路應用程式終將接觸到後端服務的特定領域的細節。

除了前面所提 JSP，其他目前在伺服端的表現層上的技術有很多，如 ASP，PHP 等等。並有許多樣版語言，如 Tag Library，Template 和許多普遍的表達層元件式架構，如 Struts。雖然這些許多技術，已提供了表達層的模組化和良好的開發環境，但尚無良好的分散式架構的表達層環境。

為了將表現層的開發作進一步的分工，本論文想要藉由分散式的架構，分散一些表達層建構的責任給後端服務的提供者。這樣網頁可以被該領域的專家來開發或控制。此外，這些特定領域的表達層元件，可被當做全球資訊網上的一個獨立且模組化的服務，被不同應用軟體使用。

第二節 Application Server

一個 three-tier 應用程式包含了 GUI server，Application server，和一個資料交易的 server。Application Server 是一個伺服器程式，提供應用程式的商業邏輯，為 three-tier 應用程式的一部份。一個典型的 Application Server 架構分為了一個提供 HTTP 服務的 Web Server 層，一個資料庫層以儲存應用程式所需的資料，

和應用程式伺服器 (Application Server)，如下圖 3。常見的 Application Server 都會支援許多不同的 Web Server 和資料庫。在 Web Server 這一層，可以是一個 Application Server 對應到許多分散的 Web Server，而這些 Web Server 會把接收到的請求轉為送給 Application Server 處理，任何回應都將會再被傳回給 Web Server。

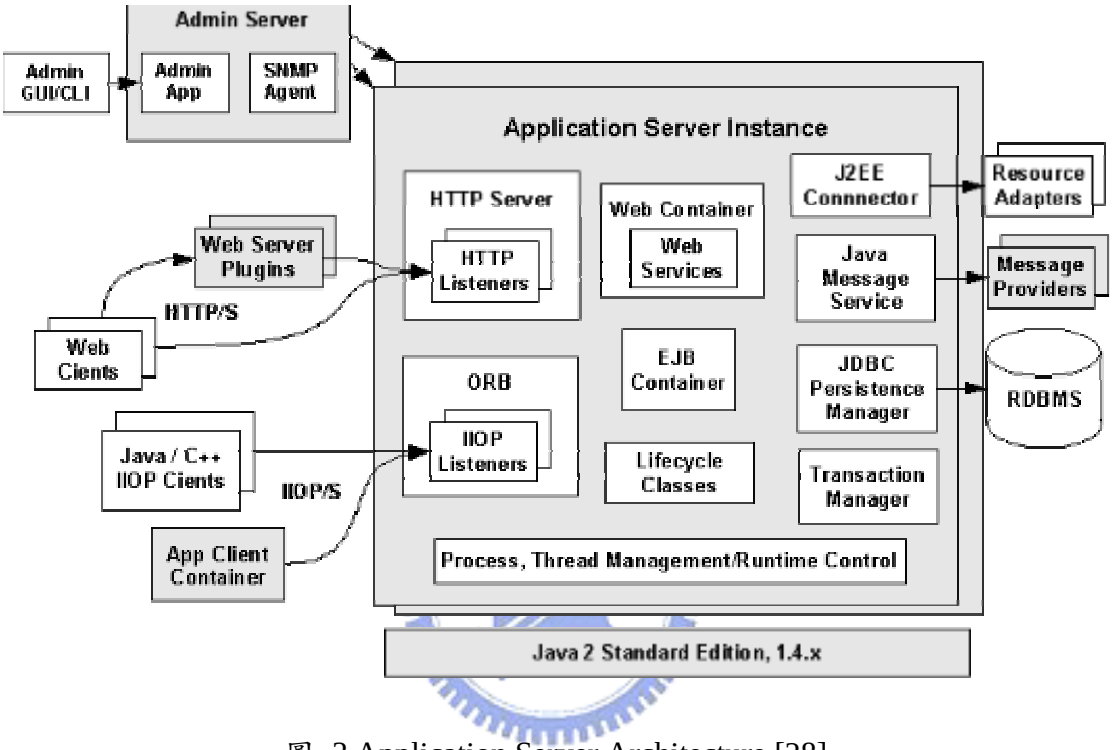

圖 3 Application Server Architecture [28]

現今常見的 Web 發展技援平台，如 JBoss，WebSphere，WebLogic，Oracle application server，等等，都為支援 J2EE 的 Application Server。JBoss 的架構在最底層利用 JMX (Java Management Extensions)做為管理各個元件的模型。Oracle application server 已整合了 portal 功能和 Grid computing。這些 Application Server 都強調了在後端環境的應用程式開發上已朝向分散式且元件化，並且幾乎都提供了 Security、Transaction、Mail、Messenging、Connector、Persistence、Clustering 等許多元件來增加應用程式的功能，以及良好的管理機制。

Portal 其實也算是在 Application Server 上的一個應用程式，而它主要的功能是做為一個 Portal 的平台來服務使用者。以 WebSphere Portal 來說，它包含的主要元件包括了安全和使用者認證的服務 (Security and member services)，網頁聚集的服務 (Page aggregation services)，和一組服務能夠讓 portlets 把豐富的內容和

應用程式帶給 Portal (Portlet container and services)；建立了一個提供連結、管理和所需的表達服務的環境。

IBM 的 WebSphere Portal 中，各別的 portlets 可以存取一個延申的架構，經由已提供的 Portlet API，可以存取 configuration objects，user session objects，和 persistent portlet data。經由延申的 Portlet 服務 API (Portlet Services API)，一群廣大範圍的服務可以被 portlet 存取與利用，像是 Collaboration services、Content access services、State Cache services、Credential vault services 和 J2EE services (JDBC, J2EE Connector Architecture, and JMS)，等許多服務。

Portal 架構中的子系統 Page Aggregation (Construction and Rendering) 負責建立一個 Portal 網頁，Portal 網頁建立的高階流程為，一開始 Portal Servlet 檢查請求，依據身份決定使用者將會看到什麼網頁和什麼 portlet，接著是呼叫版面系統決定整體的頁、列、欄、裝飾等，然後處理 portlet 的訊息並傳送事件 (event) 給其他 portlet，最後則是將被決定要寫出內容的 portlet 描繪出來。Portlet 也是屬於表達層的元件，一個 Portlet 負責內容 (Content) 的決定，而在如何表達方面，尚需要許多元件來負責。

微軟 (Microsoft) 的 Portal SharePoint Portal Server 提供一個稱為 Web Component Directory 的網站，讓網頁發展者能夠分享他們的自製元件。包含在 Web Component Directory 的元件有，UI 元件 (Web Parts)，使用後端服務的應用程式 (Web Service Client Application)，網站樣版 (Site Template)，一系列的設計元件和顏色佈景 (Themes)。

第三節 Template Framework

樣版架構 (Template Framework) 為表達層的一種架構類形，例子有 JSP，Velocity [8]等。樣版架構是強調以網頁中內容、樣本、格式為基礎的架構。讓使用者能夠以某種格式語言來方便編寫網頁，以達到網頁容易編寫和閱讀，並將複雜的程式邏輯與網頁區隔開來，提昇網頁的重覆使用性。

樣版架構最主要的目的是把表達邏輯的部份與商業邏輯分開，而達到網頁設計者 (Page Designer) 與元件開發者 (Component Developer) 的分工合作。常見

的樣版語言 (Template Language) 有 HTML，JSP，XML/XSLT，Velocity，FreeMarker，WebMacro，Tea，等等。某些樣版架構提供多種樣版語言，並可將樣本語言整合，比如 JSP 網頁中包含 HTML 語言。JSP 中的樣本語法 (Template Syntax) 有<% statements %>，<%= expression %>等。

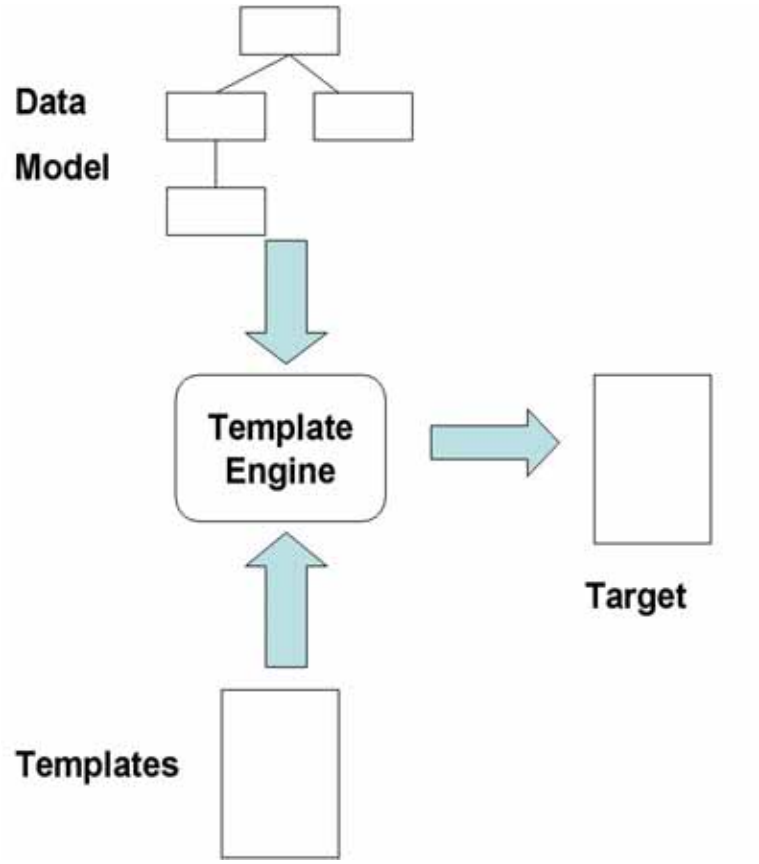

圖 4 Template-Based Transformation

如上圖 4，樣版架構中通常都會有一個 Template Engine。Template Engine 執行轉換的動作，它擁有組織為某種架構的資料，以及由網頁中的純文字建立的樣版 (Template)，接著一個樣版會黏結著一組變數 (Variables)，接著經由代換樣版中的參照為相對應的變數而產生輸出。經由置換不同的樣版，就會有不同的輸出。

第四節 Component Framework

元件式架構 (Component Framework) 為一表達層架構讓使用者以元件式的方式來建構網站。以元件為基礎的發展 (Component-Based Development) 是其基本的原則。特別地，在以頁為被組合的單位的頁的層級上，如何組裝外在的元件

的技術已有許多研究，例如截取機制 (Interception Mechanism)。強調以元件化為主的表達層架構有很多，在此以 Struts，JSF，和 Turbine 等系統做介紹。

Struts [4] 是一個具有網頁流動管理 MVC2 的架構，它藉由一個負責控制的 Servlet，提供網路應用程式生命周期的管理。Struts 推廣應用程式架構以 Model 2 的方法為基礎。Struts 提供他自己的 Controller 元件，且整合了其他的技術來提供 the Model 和 the View 元件。Struts 可與標準的資料存取技術結合，如 JDBC、EJB 來提供 the Model 元件。Struts 可與 JSP 技術合作，包括 JSTL 和 JSF，來做為 the View 元件。

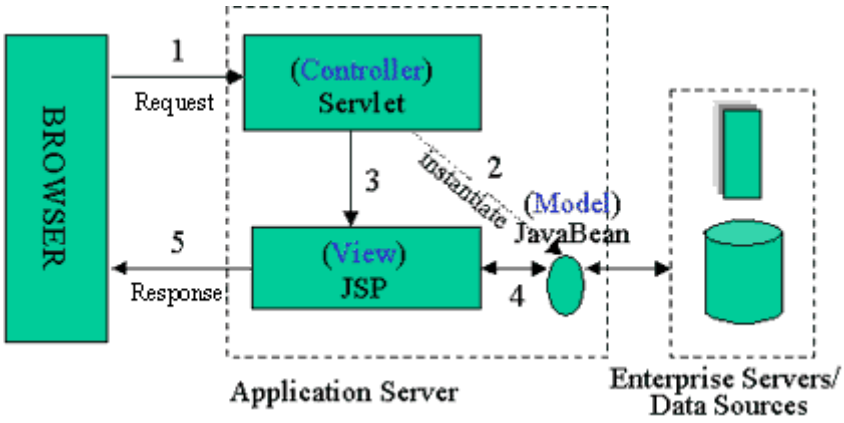

圖 5 Model2 Architecture [5]

Struts 使用一個配置檔 (Configuration File) 來初始化它擁有的資源。這些資源包括 ActionForms 元件來收集來自使用者的輸入，ActionMappings 元件來把請求傳給伺服端的 Actions，和 ActionForwards 來選擇輸出的網頁。Struts 是一個用 JSP 為中心的 MVC 架構，普遍被視為 Model 2 架構的代表實作。Struts 直接專注在為 JSP 網路應用程式提供一個網頁流動控制的架構。它提供了一個控制的 Servlet，JSP custom tag libraries，和不同的功能來支援 XML 和國際化 (Internationalization)。Struts 沒有強調元件的組合和重複使用。

JSF (Java Server Faces [1]) 是一個新興的技術，是一個用來替網路應用程式建構使用者介面的架構。Java Server Faces 技術包括了：

- 一組 API，用來做為 UI 元件，和管理它們的狀態，事件的處理，輸入的驗正，定義網頁的流動，提供國際化和存取性

- 一組客訂標記函式庫 (Custom Tag Library) 用來在 JSP 網頁中描述 Java Server Faces 使用者介面。

JSF 跟 Struts 一樣，經由一個中心控制的 Servlet 提供網路應用程式生命週期管理。JSF 像 Swing，提供豐富的元件模型和完整的事件處理和元件描繪。JSF 比 Struts 的 HTML tag library 提供了更豐富的功能，並提供與 Struts，JSTL 等良好的整合機制。

Turbine[9]，與 Struts 相似，雖然它提供更豐富的功能集合，且有意不跟 JSP 綁在一起。Turbine 的重心在於建立一群可重複使用的元件，目的在於減輕伺服端的建構的工作量。功能上來說，Turbine 提供了超過 200 個類別在套件裡。這個計劃的範圍似乎有點過於寬廣，而有些工具並不是很相關。例如，Turbine 提供 MVC 流動控制，把 Object Relational Mapping 工具整合在一起，工作排程者，地方化服務，快取服務，和許多其他的功能。此外，Turbine 提供了大量的與 Template Engine 的整合，如 Cocoon，JSP，ECS，WebMacro，Freemarker，和 Velocity。

第五節 Web Services

Web Service 主要的概念為藉由統一的資料格式為介面，達到系統服務的實作能夠跨語言，跨平台。並藉由服務之間透過網路互相溝通，達到系統可動態的整合。一個 Service 被視作為一個模組化的元件，可獨立的運作，並為其他元件所利用。

隨著 WWW 的普及與快速成長，WWW 愈來愈被使用做為應用程式間的溝通。而 Web Service[14]就是一種使用其他 Web 站台資源的服務，或是提供本 Web 站台的資源給其他 Web 站台所使用的服務。

傳統的 Web 應用程式主要是使用本機電腦上的資源，透過 Web 伺服器分享 HTML 文件。而 Web Service 應用程式提供一組通用服務，遠端函式呼叫，讓用戶端和伺服端能夠依據相同的訊息格式和規格來傳遞資訊。

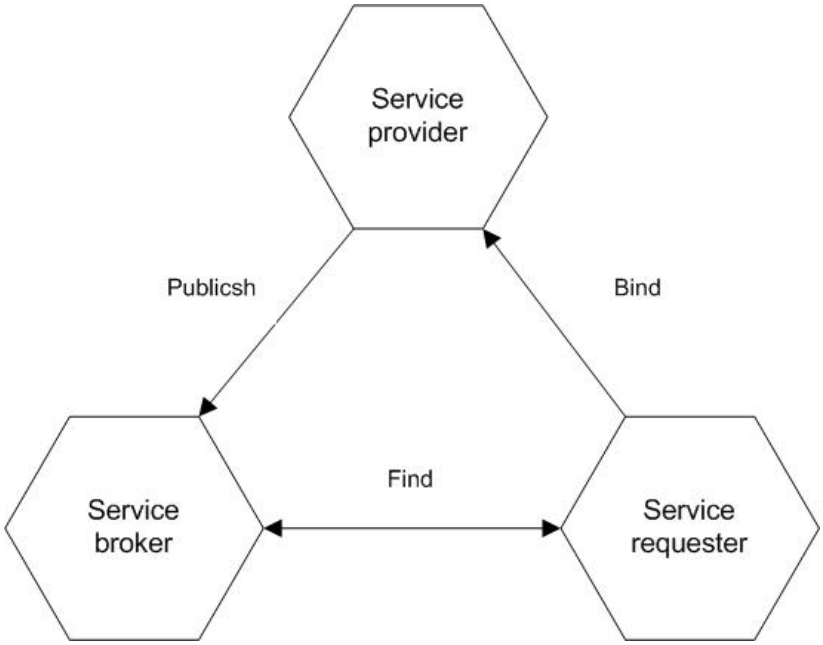

圖 6 Web Service Architecture

SOAP (Simple Object Access Protocol) 為與 Web Service 溝通的通訊協定。它是一種結合 XML 格式的訊息和 HTTP 協定的通訊協定。WSDL (Web Service Description Language) 用來定義和描述 Web Service，和 UDDI (Universal Description Discovery and Integration) 提供註冊的機制讓伺服端和用戶端能夠互相找到對方。

服務導向的特性使得網路上的元件能夠有效地被重覆利用，在開發應用程式時，可利用服務所提供的介面作為開發程式的介面。服務導向的概念已成為趨勢，在表達層的開發中，也應採用服務導向的架構，有效地利用網路上已存在的元件。本論文所提的表達架構是分散式的，其中每個元件是以服務為導向，讓表達服務成為一個獨立自主的服務恰好接近 Web Service 的精神。

第三章 動機、目的、方法

第一節 動機

在 3-tier 的架構中，因為表達層中表達資訊的集中開發，部署，管理，仍採取集中式的開發，而呈現出資料日趨龐大，且頁與頁之間的相依性複雜。又通常涉及多領域的大型網站可能會委託外部專門的人來開發特殊領域的網頁，甚至是委託內部特殊部門的人，因此造成此類大型網站開發及維護的困難度。

此外，目前表達層集中式的架構，難以將表達內容模組化的切散在各地，每個網站提供的外觀元件，都必須自己保有一份屬於自己風格的版本，並且不能夠被廣泛地被使用，我們想要將表達層也分散化來解決這樣的困難，分散式的表達層概念如下圖 7。

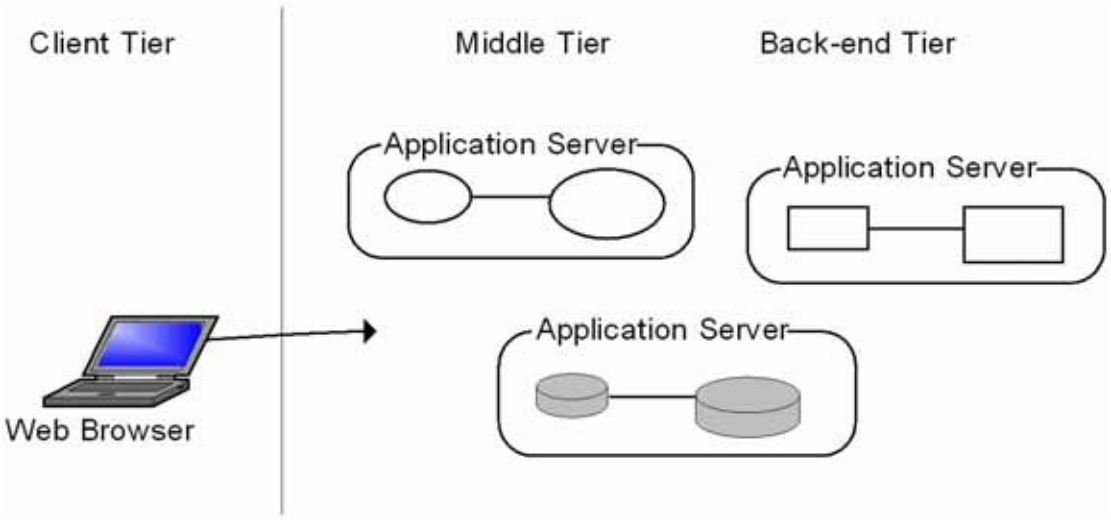

圖 7 A Decentralized Presentation Framework

第二節 目的

我們希望藉由分散式的架構，能夠將表現層建置的責任分配給後端的服務開發者，達到有效的分工。也因此降低網頁開發者的負擔。網頁開發者可以利用或整合網路上被提供出來的表達層元件。在本篇論文中，網頁表達服務就是被視為一個表達層元件，如圖 8。一個表達層元件，也就是相對應於一個特殊的後端元件的使用者介面。它為一份提供給不同的人或組織使用的網頁集合，但又為每個

使用者達到不同的表達風格，每一頁能夠分別地被調整。

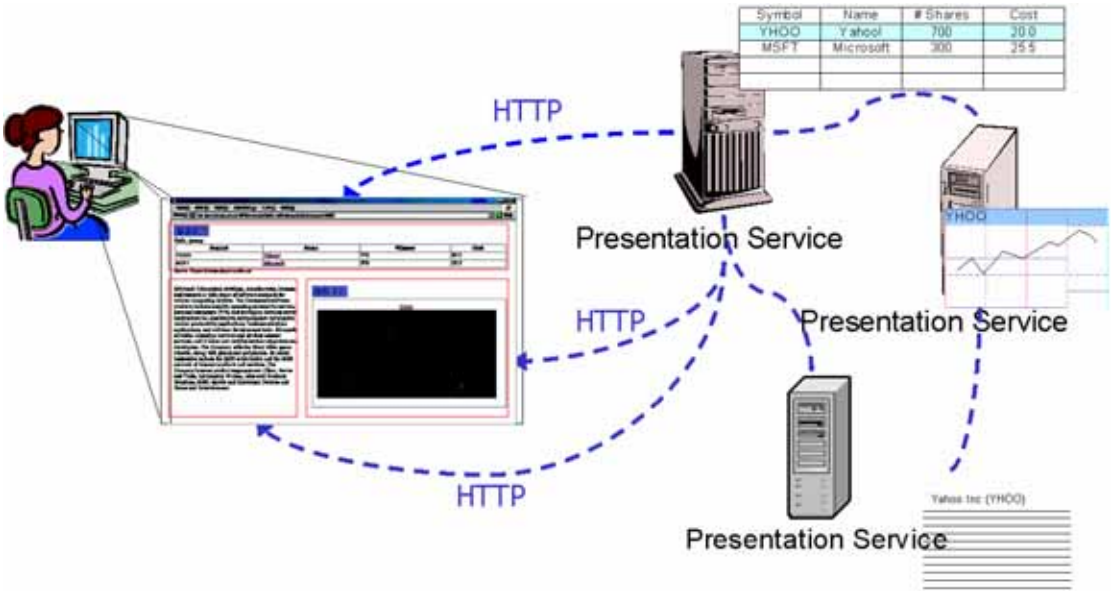

圖 8 Service-Oriented, Component-Based Presentation Concept

如圖 8所示，網站與網站之間藉由網頁表達服務能夠互相地溝通，以求達到動態的完成合理的內容。而每個網站與網頁提供的內容能夠更模組化，藉由架構中的表達服務，提供動態客製化的服務，並提昇頁的重覆利用性，減少表達層中表達程式碼的重複。表達層得以更容易被開發，也更容易被維護。

此外，我們希望在以表達服務為模組化並提供網頁內容的概念下，創造出新的表達層的市場。藉由提供前端的網頁表達服務，讓其他公司經由某權限可得到表達層的網頁表達服務，也就等於得到相對的後端服務。屬於運算核心的後端服務能夠被包裝、隱藏。將原本屬於提供運算服務的市場，移向以提供介面服務為主的市場。藉由提供表達服務，而提供整個系統服務，而市場也非僅侷限於後端的服務。

此概念的另一個好處是保護智慧財產權。藉由提供網頁表達服務，讓除了表現內容的頁以外，其餘都可被隱藏起來，大量程式碼的智慧財產權更能被保護。這些目的也正好符合 Web Service 元件化的精神，使得表達資訊更容易被維護且能夠達到後端服務的資訊包裝、隱藏。

第三節 方法

為了達到上述所說的以服務為導向的架構，許多技術上的問題必須克服。為了達到跨語言和跨平台，需要在表達服務之間設計需要的通訊協定，使得不同的表達服務能有不同的實作語言。以下討論一些必要的要素，使得表達服務能夠彈性地合作，並且能夠簡單地被實作。

首先，此架構本身是以通訊協定為基礎的 (Protocol-Based)，亦即，每個表達服務是以 XML 來互相溝通，達到平台、語言的獨立。如同網頁伺服器經由 HTTP 與客戶端溝通，並不限定其網頁伺服器必須使用何種語言。

此外，為了讓頁能夠被使用者輕易地調整，並且提昇網站與頁的模組化，頁必須以被視為一個元件的方式來設計。本論文提供了一個頁原型 (Page Model)，使得網頁開發者能夠利用遠端不同來源的網頁組裝網站。此頁原型，以頁為組裝的單位，各別以不同 XML 格式，來表示它的內容。而一個頁的 XML 內容即代表著它將輸出何內容。本論文以 XML 做為頁的樣版語言 (Template Language)。

在此頁原型中，並將頁分為多種類型，不同類型的頁都有著其不同的功能與目的。而頁的類型可再被擴充，以達到與現有技術的整合。不同種類的頁都專注於網頁的表述，而其他的事就都交給後端服務來完成。經由傳遞頁的 XML 內容，達到組裝遠端的網頁內容，並利用所設計的頁脈絡 (Page Context)，藉由它的傳遞，達到頁與頁能夠合理的連結在一起，以及內容的完整。

本論文架構的實作是建置在 Java Servlet 技術之上，由一個中心的 Servlet，來負責接收所有請求，並要求被請求的頁寫出它的內容做為回應，並且頁脈絡 (Context) 的傳遞也藉由網頁伺服器的實作來完成。而 Java 實作只是初步方便。經由 XML 格式做為溝通介面，表達服務可以是以 CGI，PHP 等語言來實作，而不只是 JAVA。因此本論文的重點在於，探討在我們的需求下表達服務所需的介面，以及頁原型的完整可用性，能夠允許不同語言實作表達服務介面。

第四章 服務導向之分散式網頁表達架構

為了達到動態客製化的需求，本論文提出一分散式的網頁表達架構。在架構中經由表達服務與網站的結合，將與遠端的動態合作交由表達服務來完成。架構中，藉由表達服務來管理網頁並負責動態的互相溝通，以動態的完成正確的內容。每個表達服務都是一個整體的模組，其包含的頁，皆視為一組相關的頁。

第一節 架構設計 (Service-Oriented Presentation Infrastructure)

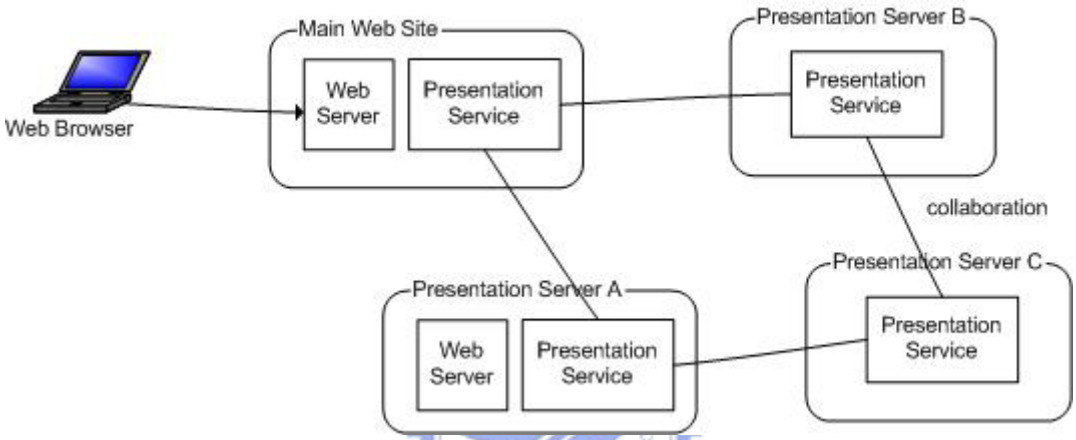

圖 9 Service-Oriented Presentation Infrastructure

圖 9的架構，為此論文中所提出的分散式網頁表達架構。在這樣的架構環境中，表達層是分散式的。每個單位或組織都可在網路上提供他們的表達服務(Presentation Service)。而每個表達服務都被視為是一個相對應於某個網路服務的使用者介面 (View)。它們彼此間可以互相溝通，合作。

每個表達服務都是獨立自主的服務，而不是為每個使用它的主機提供一份客製化的網頁內容。當網頁設計者要把分散在各個表達服務中的某些網頁，組裝在一個表達服務中的一個頁時，各別網頁仍然需要具有合理的彈性在。比如說，當某一頁是從遠端被拿進來組裝在某一頁中的時候，此從遠端來的頁的連結仍然會被控制，此頁的連結連出去的結果還是會顯示在同一個框架中，而看起來仍然還是具有相同的外貌。甚至，網頁設計者也可以修改其連結的位址，例如，有一個做為導覽的用的網頁，裡面包含首頁 (Home)，前一張，下一張等等的導航連結，如果是從遠端把它拿過來用的話，首頁的連結可被修改成自己設定的首頁，而不

再是服務提供者的首頁。網頁中包含對彼此的連結和其連結屬性，可以說是網頁間的相依性，要把此相依性從網頁中去除掉，才可把一個頁當做是一個可調整的元件般使用。

網頁表達服務間的溝通需要使用標準的網路協定，而非一些分散式的計算平台，藉以達到平台及語言的完整與獨立性。就像 WWW，此架構能允許持續的改善，且防止包含不必要的技術。

對於每個網頁表達服務來說，其本身也可以是一個網頁伺服器 (Web Server)。當有需要提供網頁服務給網頁瀏覽器時，一個 HTTP Handler 就只是把使用者的請求丟給網頁表達服務中相對應的頁。網頁表達服務中的每一頁都能產生 HTML 文件。

表達服務中的頁可以產生延申標記語言 (XML) 格式的內容，此 XML 格式也相對於它會產生什麼樣的 HTML 內容。但此 XML 格式的內容是為了給其他頁來使用，進而準備並產生它們的內容，由 XML 內容就可知道相對應的輸出內容。換句話說，也就是一個頁 經由彼此交換 XML 訊息可以結合其他頁的網頁內容。本論文中稱此內容的產生方式為嵌入 (embedding)。 另一種方式，則是不結合來自遠端的 XML 內容，一個頁可能選擇在 HTML 內容中產生連結，指示瀏覽器直接去遠端的表達服務抓取 HTML 內容，此種方式稱為連結 (linking)。注意在這種情形下，此網頁表達服務需要伴隨著一個 HTTP 伺服器。

此架構的另一個需求是效能，它是其能否被企業所使用的考量之一。大量的內容，像是動態產生的圖檔，應該直接提供給使用者端的瀏覽器，而非複製一份傳給主表達服務。雖然後面的方式在某些情形下也有它的好處，可以被主表達服務來做調整或修飾。

第二節 網頁表達服務

以較高的層次來看，一個網頁表達服務，包含一群有著網頁流動 (Page Flow) 的邏輯互相連結著的頁。如下面的圖 10，HTTP 伺服元件 (HTTP Handler) 只是檢查從瀏覽器送來的請求，查詢在網頁表達服務中的頁，然後要求它組織好內容，並送出適當的 HTML 內容回應給瀏覽器。

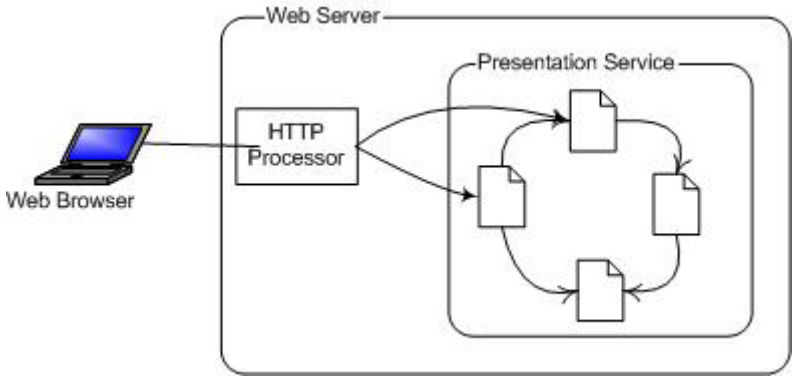

圖 10 Presentation Service

網頁表達服務中的每一頁，皆交由一 Page Space 物件來管理。當 HTTP 請求處理元件接收到一客戶端的請求時，必須透過 Page Space 物件來得取一個頁。每一頁都藉由其獨一無二的路徑來辨別。當 HTTP 請求處理元件要求一個頁寫出它的內容時，一個頁必須藉由一 Context 物件來完成完整的內容，而此 Context 物件由 HTTP 請求處理元件來完成。Context 包含了頁所需要得知的動態資訊，提供查詢介面。

每一個網頁表達服務都是一個可獨立運作的服務。本身就包含了一組互相有關聯的頁。頁與頁之間可以藉由獨一無二的路徑而互相連結。並將每一頁都視為一個元件，一個被別頁組合的單位。當要把不同頁組合在一個頁的框架中時，頁中的連結依然能夠被控制。

第三節 合成關係

組合其他遠端的表達服務管理的頁是經由匯入 (Import) 的方式。匯入的動作只是匯入它們的連結，並產生當地的代理頁 (Local Proxy)。在我們的設計中，它們也是頁的一種，稱作遠端頁 (Remote Page)。匯入後表達服務會有一份匯入遠端表達服務的紀錄。如圖 11所示，匯入之後，可以被 HTTP 伺服元件來存取。如同是本地的頁，也可被其他頁連結或組合來使用。

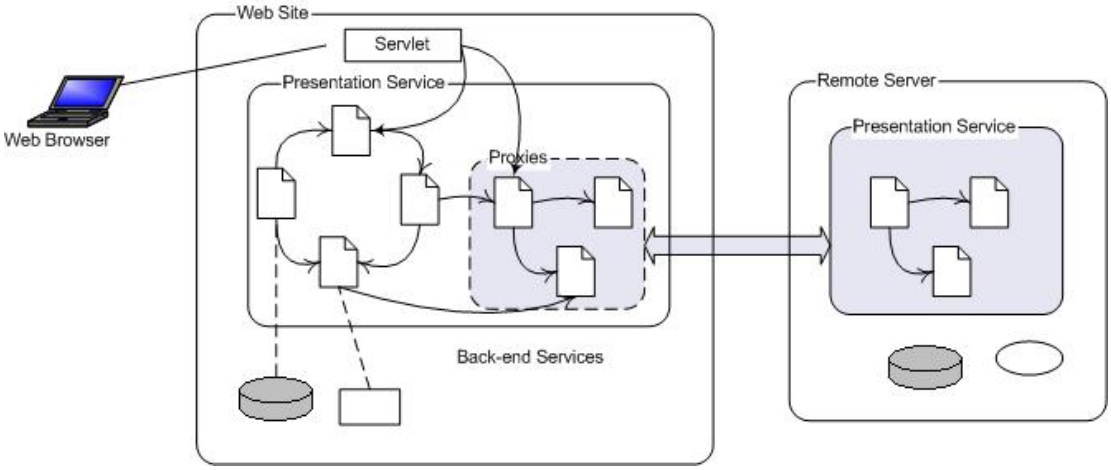

圖 11 Import Concept

在開發需要使用其他遠端的網頁表達服務的網站時，開發者需要另外登入要使用其網頁的網頁表達服務，以了解其每一頁本身所具備的一些特性和屬性，才知道匯入後該如何使用它的網頁。另一種方式是藉由直接匯入就能得到足夠的資訊，以知道每一頁如何被使用。

第四節 嵌入與連結 (Embedding/Linking)

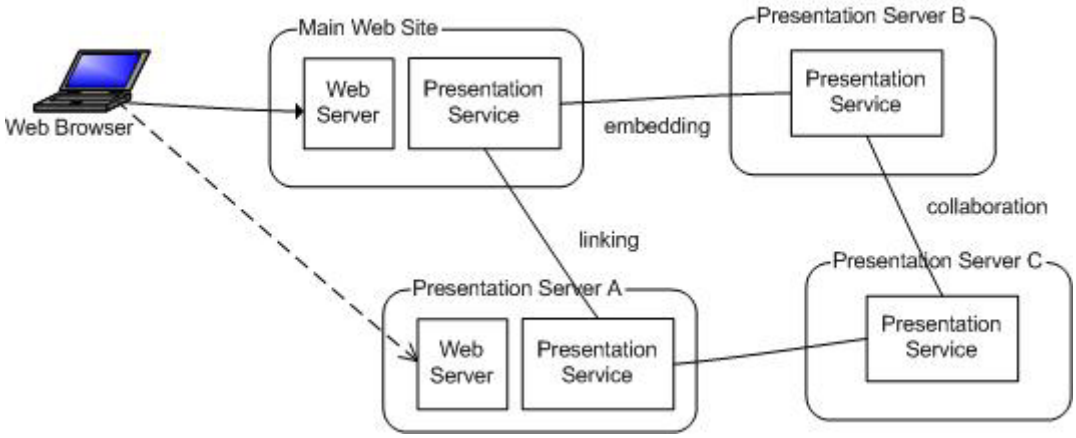

圖 12 Embedding and Linking mechanism

在圖 12中，整個系統被分成四個網頁表達服務。主網頁伺服器經由網頁表達服務對於遠端的公司概況頁使用嵌入的方式，而對股價走線圖的頁則是使用連結的方式。採用連結的方式，告知瀏覽器直接連結到遠端得取圖片也許是兩個表達服務之間溝通後的結果，而不是直接告知瀏覽器一個固定靜態的連結。

嵌入與連結是本論文所提之分散式架構中，互相溝通以完成正確內容的機

制。在匯入遠端網頁，並建置完整個網頁架構後，當使用者送來一個請求，要求遠端頁寫出它的內容時，它實際上會跟遠端的網頁表達服務溝通，此時有兩種情形，它可以採用嵌入 (Embedding) 的方式，要遠端的頁寫回內容回來；或是採用連結 (Linking) 的方式，告訴瀏覽器直接去遠端存取。不管使用何種方式，都需要把該需要準備好的資訊，或稱之為頁脈絡 (Page Context) 以某種方式傳送給被請求的頁，以讓它能夠正確地將內容寫出，並且達到完整合理的網頁流動 (Page Flow)。

為了讓遠端頁知道該採用何種方式得取頁的內容，當表達服務之間在做匯入的時候，被匯入的表達服務需要表明它的身份，表明它是否有提供 HTTP 伺服器的功能和其主機位址、通訊埠，等相關訊息。如果是嵌入的方式，就一定要跟遠端的表達服務做溝通，以得到此頁的 XML 內容，進而產生相對的輸出。而連結方式的話則有多種情形，分為三種。如果遠端的頁是靜態 (Static) 類型的話，則可直接寫出固定的連結，告知瀏覽器去此頁實體所在的地方取得內容。第二種情形是要通知遠端的表達服務，接著不用等回應結果，即寫出告知瀏覽器直接去遠端取得正確的內容。第三種是通知遠端的表達服務，接著要再等得到遠端的回應結果後，才完成並寫出正確的連結。

連結與嵌入兩種方式各有使用的考量與時機。在效能上，連結的方式會比嵌入的方式更有效率，尤其是頁的內容龐大的時候；但有時使用嵌入的方式，主表達服務可以再對傳送過來的檔案或頁的內容做適當的處理。網頁開發者可以設定匯入的頁被請求送出內容時，該使用何種方式以得取此頁的內容。

在我們的實作中，表達服務內部的頁，以階層的方式被組織，而可分別經由獨一的路徑來辨別。一點要注意的是，必須把一個表達服務裡的所有頁的集合當做是一個完整的模組，所以匯入總是匯入一整個表達服務作為一個整體。藉此表明每一個表達服務應該表達的是一個完整且定義明確的頁集合。在列表 1 中，顯示了例子中提供投資管理服務介面的網頁表達服務的最上層配置。

```
<!-- at companyinfo.com-->
<pageSet>
  <page path="portfolio"> ... </page>
  <page path="symbolTable"> ... </page>
  <import host="companyinfo.com"
      port=”5678” destPath=”com”>
  <import host="quotes.com"
      port="1234"  destPath="qs"/>
</pageset>
<!-- at companyinfo.com-->
<pageSet>
 <page path="profile"> ... </page>
</pageSet>

<!-- at quotes.com-->
<pageSet>
  <page path="chart"> ... </page>
</pageSet>
```

表 1 Top-Level Presentation Service Configuration

第五節 頁原型 (Page Model)

如下圖 13的類別圖顯示。在此圖中，PageServlet 相當於 HTTP 請求處理元件 (HTTP Handler)，它包含了一個 PageSpace 物件的連結，此 PageSpace 物件持有並管理在表達服務中的所有頁，附錄二顯示了一實作範例中的表達服務的介面與配置設定。當一個頁要寫出 HTML 文件或產生 XML 的內容時，它需要一個 PageContext 物件，經由它可以得到所需要的資訊，例如輸出流。圖 13並顯示一個頁可以寫出多樣不同的內容格式。

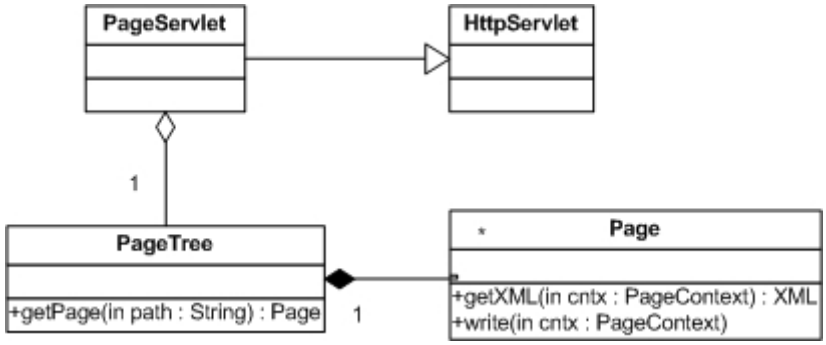

圖 13 Presentation Service Class Diagram

4.5.1 頁

一個頁 (Page) 的用途是傳送它的內容，一個頁可以傳送 HTML 內容給瀏覽器，或是產生 XML 格式的內容。頁 (Page) 可被分為許多種類，而每一種類的 Page 都有不同的行為模式和應用上的目的。HTML 是最普遍也最基本的網頁格式，因此需要提供使用者可以編寫 HTML 格式的網頁。另外，在大部份的情況下，會需要呼叫後端的服務，以產生動態結果的動態網頁。有的網頁還需要負責提供版面的彈性配置，可以整合許多頁在同一個頁。另外，有某些型態的頁是要負責網頁流動的控制邏輯，比如說登入成功和登入失敗會導致於流向不同的頁。為了達到以上具基本且完整功能的頁原型，本論文中提供不同類型的頁，針對以上的功能來說，就有 HTML 頁 (HTML Page)、產生動態結果的 XSLT 頁 (XSLT Page)、從遠端表達服務匯入而產生的遠端頁 (Remote Page)、控制版面配製的複合頁 (Composite Page)，和依邏輯來決定內容的支配頁 (Control Page)，等等。

頁的類型是可以再擴充的，甚至可以是一個 JSP，PHP 等類型的頁。如下圖 14的類別圖所示。

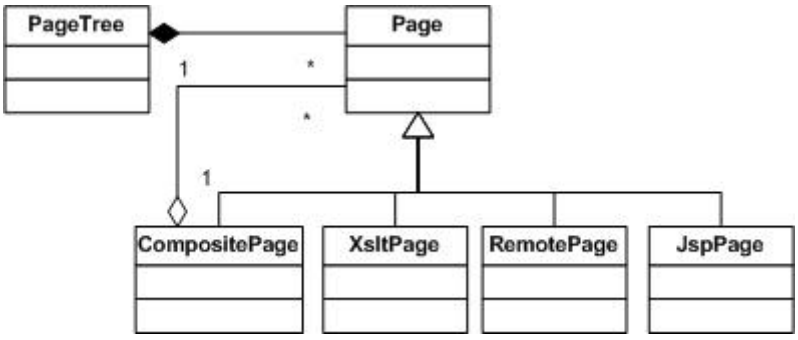

圖 14 Types of Pages

表達服務間的通訊協定，是使用延申標記語言 (XML)。在表達服務與後端服務間的訊息溝通也是採用標準的 XML 介面。這有利於把 XML 的資料轉變成另一種格式。雖然必須要努力達到跨平台，但在實作上得避免重新去發明一個完整的內容表達語言和程式語言。

在此頁原型中，一個頁主要傳送 HTML 內容，也需要能夠產生 XML 來表示它的內容，由此 XML 就可得知最後要產生的 HTML 內容，因此一個頁實際儲存的內容就是此 XML。一個頁的 XML 格式中會有許多宣告，包括超連結 (Next Pages) 的宣告，也就是網頁中會有那些連結；和它需要一些事先被準備好的參數，或稱為頁脈絡 (Page Context) 的宣告，和需要某些其他頁做為此頁的參數，參數頁 (Parameter Pages) 的宣告，等等，如表二。

```
<page path="symbolTable">
    <cntx name="user"/>
    <next name="portfolioLink"/>
    <body>
       . . .
    <link path="portfolio"
          name=”portfolioLink”>
      <cntx name="symbol" value="YHOO"/>
      YHOO
    </link>
     . . .
    </body>
</page>
<page path="chart">
    <cntx name="symb"/>
    <cntx name="range" default="1week"/>
    <next name="chart"/>
    <body> . . . </body>
</page>
```

表 2 Page Declaration Format

4.5.2 頁脈絡 (Page Context)

頁脈絡 (Page Context) 主要是用來做為頁與頁流動之間動態客製化資訊的儲存。它主要儲存一些需要的資訊，或說一個頁所需的參數值，來讓一個頁去取得它以產生正確的內容，比如說一個頁的使用者參數，因此它是當一個頁寫出內容時得以去查閱以得知參數值的集合。但真實上的頁脈絡的傳遞溝通機制還是依靠網站伺服器的實作來達到。

頁脈絡為一個物件類別。它提供一些介面如下。

- getAttr(): Object
- getParameterPage(String name):String
- getNextPages():Set
- getService(String name): Object
- getWriter(): Writer
- combineCntx()
- toXml(): Xml

頁脈絡可以結合其他頁脈絡，並可轉為 XML 格式來傳遞。一個頁藉由頁脈絡可以得知許多不同類型的參數值，包括頁中會使用的服務，參數頁，次頁的連結，以及其他所需的參數值，在頁寫出內容時都將用到資訊。而跟使用者和系統有關的參數值，查詢時依照不同層次的優先順序來查詢，由低而高和查詢順序分別為頁，使用者請求、連線 (Session)、和表達服務 (Application) 四種層次，概念上如同 JSP 中所定義，如下圖 15。

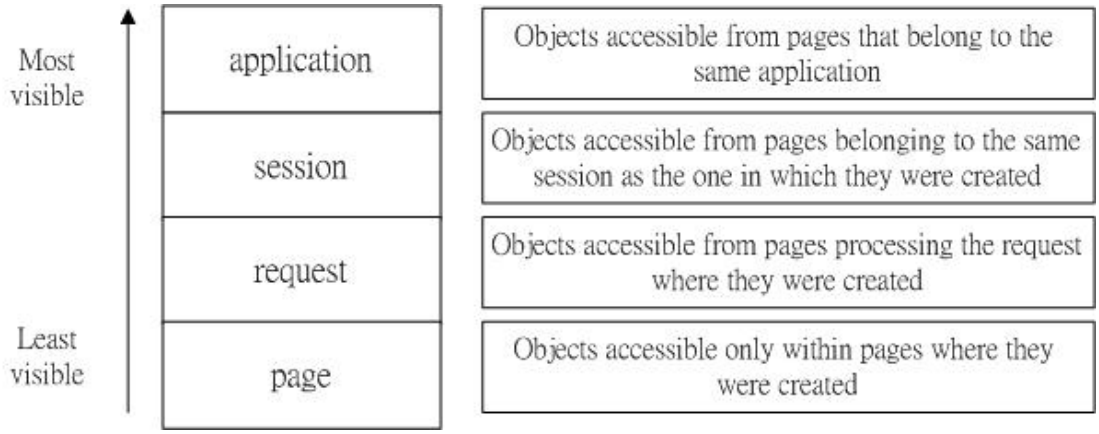

圖 15 Page Context Levels

第六節 頁的種類

頁的種類可被分類為下列幾種類型，並分別依序對其功用與其頁中 XML 格式做介紹：

4.6.1 HTML 頁 (HTML Pages)

HTML Pages 是為了用來讓網頁撰寫者可以就如同以往來編輯 HTML 格式的網頁，使其在編輯網頁的過程中，就好像以往在編 HTML 網頁一樣，而不需學習以及編寫 XML。HTML Pages 的 XML 表示，如下有個範例：

```
<page file="index.html" type="html">
      <cntx name="user">
      <next name="link1">
</page>
```

表 3 HTML Page Format

而實際 HTML Pages 在寫出 HTML 內容時，會去找對應到的 HTML 檔案，在這例子中是 index.html 檔案，然後就只是把它的內容寫出去。不過在編此 HTML 檔案時，還是可以有些許的不一樣，它可以有某種奇異符號來代表在文件中某些位置它需要某些資訊的輸入，比如說：

```
<p>
   Welcome ${user}
</p>
<p>
   ${ <link name="link1" path="portfolio">
        Portfolio
      </link>
   }
</p>
```

表 4 HTML File Format

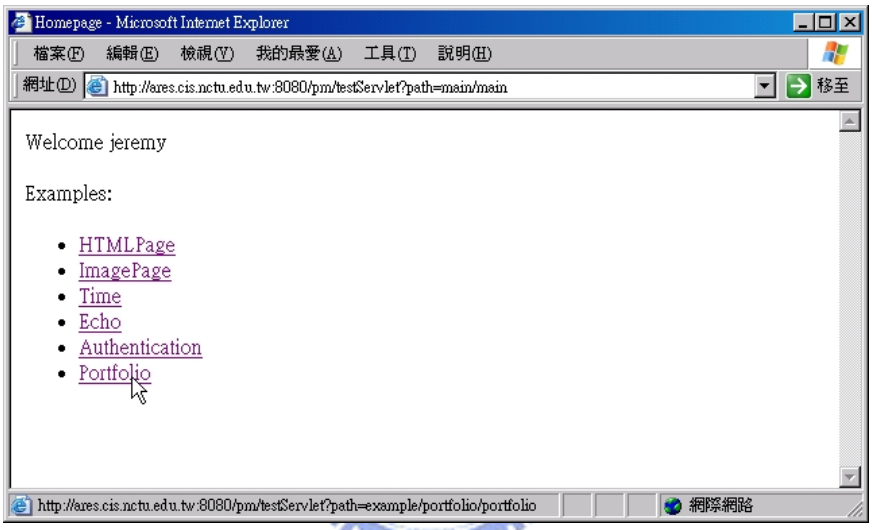

圖 16 HTML Page Sample

HTML 文件還是可以插入自己定義的一些標籤格式，上面的例子就是${}括號裡面的東西會另外被轉換為其他 HTML 格式的資訊。轉譯過程中，看到${user}會去查詢頁脈絡，以正確的 user 值取代之，而看到括號裡是 XML 格式的話，如${<link></link>}，就會依自定的標籤格式來做轉譯。圖 16為 HTML 頁實作的一個例子。

4.6.2 XSLT 頁 (XSLT Pages)

XSLT Pages 為動態的頁，它提供動態的 XML 內容，之後將可被 XSL 頁 (XSL Pages) 中的轉譯規則轉換為自定的 XML 標籤格式，最後再經過轉譯成為最後要輸出的 HTML。XSLT Pages 有內嵌 (In-Line) 模式和一般模式。內嵌模式就是把 XSL Pages 的轉譯規則直接嵌入在 XSLT Page 中。這樣的話在此頁就會轉譯為最最後的 HTML 格式輸出。而如果是一般模式，它則輸出 XML 格式，並要求瀏覽

器再去請求某 XSL Page 來讓瀏覽器負責轉譯的工作。一般模式的話，藉由 XSL Page 中提供的轉譯規則，就必須能夠將 XML 格式直接轉換為 HTML，不能夠先轉換為自定的格式，少了些許範圍的彈性，這是與內嵌模式不同的地方。

XSLT Pages 可呼叫網路上的服務，得到服務傳回的 XML 結果，再對這些特定的 XML 格式做轉譯。表 5介紹一個 XSLT Page 呼叫某個服務的例子，此例為內嵌模式：

```
<page type="inlinexslt">
   <cntx name="user"/>
   <next name="link1"/>
   <service host="localhost" path="symtable"
            name="symbol" port="1121"/>
   <body>
      <img src="logo"/>
      <br/>
      Hello, {user}
      <call service="symbol">
         <getSymbolTable user="{user}"/>
      </call>
      Server Name is {pm.ServerName}
   </body>

   <xslt>
      <xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            version="1.0">
         <xsl:output method="html"/>
         <xsl:template match="/">
            <html>
               <head>
                  <title>Symbol Table</title>
               </head>
               <xsl:apply-templates/>
            </html>
         </xsl:template>
         <xsl:template match="body">
            <body>
               <xsl:apply-templates/>
            </body>
         </xsl:template>
```

```
        <xsl:template match="symboltable">
           <table width="100%" border="1">
              <tr>
                 <th>Symbol</th>
                 <th>Name</th>
                 <th>#Shares</th>
                 <th>Cost</th>
              </tr>
              <xsl:for-each select="symbol">
                 <tr>
                   <td>
                    <link path="profile" name="link1">
                       <cntx value="{@symbol}"
                             name="company"/>
                     <xsl:value-of select="@symbol"/>
                    </link>
                   </td>
                   <td>
                     <xsl:value-of select="@name"/>
                   </td>
                   <td>
                     <xsl:value-of select="@shares"/>
                   </td>
                   <td>
                     <xsl:value-of select="@cost"/>
                   </td>
                 </tr>
              </xsl:for-each>
           </table>
        </xsl:template>
        <xsl:template match="*">
           <xsl:copy-of select="."/>
        </xsl:template>
       </xsl:stylesheet>
   </xslt>
</page>
```

表 5 InlineXslt Page Format

上面的例子是股票持有表的例子，結果如圖 17。其中它呼叫一個股票持有表的服務，得到某種外在的 XML 格式，再用下面的規則對它做轉譯。如果非內嵌模式的話，頁的 XML 內容則為如下表 6，其中 body 標籤中有個 file 屬性，指的是所使用的 XSL Page 的路徑。

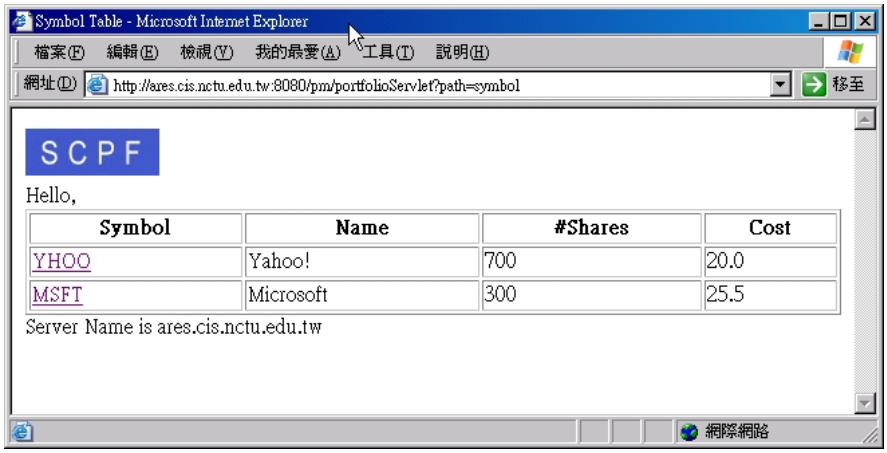

圖 17 XSLT Page Sample

```
<page type="xslt">
  <cntx name="user"/>
  <next name="link1"/>
  <service host="localhost" path="symtable"
           name="symbol" port="1121"/>
  <body file="xsl/symbol">
     <img src="logo"/>
     <br/>
     Hello, {user}
     <call service="symbol">
        <getSymbolTable user="{user}"/>
     </call>
     Server Name is {pm.ServerName}
  </body>
</page>
```

表 6 XSLT Page Format

4.6.3 XSL 頁 (XSL Pages)

XSL Pages 為提供轉譯規則的頁。格式就跟標準的 XSLT (XSL Transformation) 格式一樣。也就是用 XSLT 語法來做為轉譯規則的語言。它跟 HTML Pages 一樣，都對應到一個檔案，而只是將此檔案的內容依正確的格式寫出，頁的 XML 內容格式範例如下表 7：

```
<page type="xsl" file="simpleTime.xsl"/>
```

表 7 XSL Page Format

4.6.4 影像頁 (Image Pages)

Image Pages 為專門提供某個影像的頁，這裡將一個圖檔，也以頁的觀念來對待之。當使用者要求此型態的頁，就跟平常要求一張圖檔的結果是一樣的，影像頁會依其所對應圖檔的格式將圖檔內容寫回給瀏覽器，看圖檔的格式、副檔名為何，也許是 image/gif 或是 image/jpeg。Image Pages 也跟 HTML Pages 和 XSL Pages 的內容類似，只是對應到另一種格式的檔案，在這裡為圖檔。頁的 XML 內容格式範例如下表 8：

```
<page file="night.jpg" type="image"/>
```

表 8 Image Page Format

4.6.5 複合頁 (Composite Pages)

Composite Pages 的目的為提供彈性的版面配置，就像 HTML 已提供的 frame、frameset 標籤。它用來把其他頁組合起來，而組合的單元為一個頁。特別要注意的是，每一頁不能因為有可能被他頁組合而有與他頁的相依性存在，例如連結標籤的 target 屬性。在沒有了頁與頁之間的相依性之後，如此頁在概念上才可被視作為元件，以被任何人來使用。下表 9為一複合頁的例子：

```
<page type="composite">
  <cntx name="user" default="Guest"/>
  <cntx name="company"/>
  <vsplit framespacing="0" rows="30%,70%"
          frameborder="0" border="0">
    <page name="top" path="symtblRW">
      <cntx name="link1" value="_top"/>
      <cntx name="link2" value="bottomLeft"/>
      <cntx name="user" value="{user}"/>
    </page>
    <hsplit cols="40%, 60%">
      <page name="bottomLeft" path="profileXslt"/>
      <page name="bottomRight" path="chart"/>
    </hsplit>
  </vsplit>
</page>
```

表 9 Composite Page Format

此 Composite Page 的範例分成三塊框架。每個框架代表某個路徑下的頁。利用自定的格式，vsplit、hsplit、page 來作為配置網頁用的標籤，這些也是此類型的頁的專用標籤，或說是樣版語言 (Template Language)。vsplit 標籤為垂直方向分割，而 hsplit 為水平方向分割。

4.6.6 支配頁 (Control Pages)

Control Page 用來作為決定顯示何者內容的頁。決定的方式可能依照某程式邏輯，或是依照網頁當時的狀態。它跟其他類型的頁的不同在於它不是一個本身具有內容的頁，它不是著重於描繪內容出來，它是決定要用何者內容或是要使用那一頁的內容做為它的輸出。下表 10為一複合頁的例子：

```
<page type="control">
  <cntx name="user"/>
  <cntx name="password"/>
  <service host="localhost" name="auth" path="auth"
              port="1119"/>
     <body>
       <caseset>
          <call service="auth">
             <check pass="{passwd}" user="{user}"/>
          </call>
          <case>
              <true/>
              <link path="main/main">
                  <cntx name="@user"/>
              </link>
          </case>
          <case>
              <false/>
              <link path="main/index">
                  <cntx name="@user"/>
              </link>
          </case>
       </caseset>
     </body>
</page>
```

表 10 Control Page Format

有些頁需要依據動態的狀態來決定內容，此時就可使用 Control Pages。可以經由標準的查詢介面詢問頁脈絡，也可直接經由呼叫後端服務，一方面減低控制元件 (Controller) 的負擔，一方面分散式且複雜的的網頁流動邏輯也許不容易管理。另一種方式是，在網頁表達服務啟動時，可讓一個相隨的後端服務也跟著被啟動，此後端服務可做身份確認或流程控制的服務。

4.6.7 遠端頁 (Remote Pages)

RemotePage 可以說是構成這個分散式架構的頁原型裡頭中最重要的元件之一，它需要與遠端的表達服務溝通，才能完成要表達的內容。Remote Pages 是經由匯入遠端的表達服務而產生的頁，它沒有實體的頁檔案，只是存在於記憶體中

的物件。它代表的是某個遠端的表達服務中的某個頁。Remote Page 物件記錄的是遠端表達服務主機的位址，表達服務通訊埠，頁的路徑、類型，等等。當某個 Remote Page 被要求寫出回應的時候，它會去跟遠端的表達服務溝通。而它完成內容的方式，有兩種，一種是嵌入，另一種是連結。嵌入是當遠端的表達服務沒有伴隨著一個 HTTP 伺服器的時候，或是為了能夠得取網頁中所需的相關檔案，藉以做進一步的修改時，而採用的方式。而連結是較有效率的方式，寫出的內容皆為告知瀏覽器直接導向到遠端的此頁所真正存在於的位址 (URL)，但要注意的是，必須將需要準備好的頁脈絡轉為參數接在連結位址的後面，連結的情形有三種模式：

1. 直接寫出固定的連結：

   當網頁為靜態的類型，像是 Image Page 時，不因為參數或使用者狀態而對內容有所影響的時候，就可直接通知瀏覽器導向到此頁實際上存於在遠端的位址。

2. 通知遠端的表達服務後，接著立即寫出導向連結：

   傳遞已有的頁脈絡給遠端，叫遠端把某一頁的內容準備好，然後告知使用者去得取。此種情形為為了表達服務之間有合約在，必須透過約定的表達服務的觸發，不能讓使用者直接能夠透過瀏覽器得取某頁的內容。

3. 通知遠端的表達服務並需要等到它回應之後，才將連結完成並寫出：

   需要跟遠端的表達服務溝通，並得到它的回應，是因為為了要得知遠端的頁需要有那些被準備好的頁脈絡，以轉為參數接在連結位址之後，也就是將原本的頁脈絡轉為頁的請求參數形式。動態類型的頁需要採用這種方式。

而嵌入的方式，是要遠端的頁為把頁的 XML 內容傳遞過來，轉為某型態的頁，接著再要求它經由傳入的頁脈絡寫出內容，傳給客戶端。

也因為藉由 Remote Page，不論是採用嵌入或是連結的方式，都能夠將頁脈絡的資訊完整的傳給頁，因此得以達到動態的客製化內容。關於頁中連結 (Link) 的修改需要，可藉由截取頁 (Interception Page) 來完成。

4.6.8 樣版頁 (Template Pages)

Template Pages 被當作一個頁型態 (Type) 來使用比較恰當，像是程式中的巨集。它並不是能直接被客戶端所要求的頁。它是會被 Interception Page 頁所呼叫要使用的頁。如下表 11是一個 Template Page 的範例，它需要一個名稱為 inPage 的參數頁。Interception Page 將在下節描述。

```
<page type="template">
   <cntx name="user"/>
   <cntx name="range"/>
   <cntx name="symbol"/>
   <paramPage name="inPage"/>
   <vsplit>
      <page path="example/portfolio/symtblRW"/>
      <page param="inPage">
         <cntx name="range" value="{range}"/>
         <cntx name="symbol" value="{symbol}"/>
      </page>
   </vsplit>
</page>
```

表 11 Template Page Format

這裡，Template Page 寫出的內容跟 Composite Page 很像，以使用者介面的角度來說，它一樣也是把頁當做組合的單元。

4.6.9 截取頁 (Interception Pages)

Interception Page 為用來呼叫 Template Page 或是其他類型的頁的頁，無法由自己產生內容。它負責把需要填入的參數頁和相關資訊準備好，然後去使用某一個 Template Page，或是使用某一頁，並修改此頁的連結內容、圖示、文字，等等。Interception 頁是為了能夠得到對連結 (Link) 完全的控制，使得可以彈性的改變連結內容，藉以達成客製化的目的。通常前端與後端服務之間難以消除的耦合，就在於後端服務動態的產生連結資訊。Interception 頁可用來對於一個遠端頁產生的內容再做適當的修改，以得到正確的內容，就像遠端頁中 Charting Service 產生圖檔的連結資訊之後，也許需要被適當的修改。使用 Template Page 的好處

就是許多相同版面的設計不用重複寫好多次，而讓 Template Page 變成像是一種樣板。

當此 Interception Page 使用某一個 Template Page，它會呼叫 Template Page 去把內容寫出來給它，它再把內容寫回給呼叫它的人。 有時 Interception Page 的使用上，不一定是一個頁，而可能是其他動態頁，如 XSLT 頁，動態依規則轉換而成 Interception 頁的格式，進而再被做一次轉譯的處理。

```
<page type="interception" use="test/template">
      <cntx name="user"/>
      <cntx name="currentSymbol"/>
      <paramPage name="inPage" value="main/main">
            <map from="portfolio" to="profile"/>
      </paramPage>
</page>
```

表 12 Interception Page Format

如上表 12是一個 Interception Page 的例子，有一個名稱為 inPage 的參數頁，此參數頁的路徑為 main/main。對於每一參數頁或使用的頁，可以設定其中的連結被對應的修改。這些資訊都將放在頁脈絡裡，交給 test/template 這個路徑的 Template Page 來把內容完成。這裡範例中使用的 Template 頁，就是在上一節中介紹樣版頁時使用的例子。

如下表 13，則不是使用一個樣版頁，而是使用其他一般的頁。它使用另一頁做為它的內容，但多了一些不同的是，它可以去修改一些內容。不可避免的，要使用遠端頁，通常會需要修改連結的路徑。圖 18顯示出截取頁修改路徑的例子。

```
<page type=”interception” use=”charter/chartRW”>
   <ctnx name=”user”/>
   <cntx name=”currentSymbol”/>
   <map from=”charter/index” to=”main”/>
</page>
```

表 13 Interception Page Format

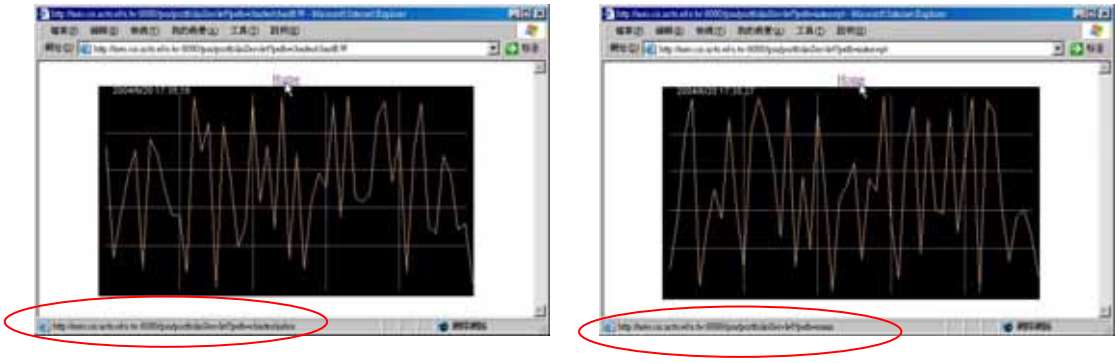

圖 18 Interception Page Replace Link Sample

4.6.10 改寫頁 (Rewrite Pages)

Rewrite Pages 跟 XSLT Pages 所要達到的目的是一樣的，都是為了能夠將 XML 內容做重新轉述為被接受的格式。只是這裡所用的轉譯規則不是採用標準的 XSLT 語法。而是自己提供的規則導向引擎 (Rule Engine) 來做重寫 XML 內容的動作。要做 XML 格式重寫的動作，必須提供 Rule Engine 二個輸入，一個是規則集合 (Rule Set)，一個是要被重寫的 Xml。下面是一個範例：<body>標籤為一個 XML 架構。它會依照下面的規則被做轉寫。規則集合裡可包含任意多寡的規則。配對規則的順序可以是依照規則寫的順序。而配對到某一個規則的時候，有可能再將片段的 XML 去做重新的規則配對。當配對到的時候，就會傳回被改寫的 XML。如下表 14，為一項重寫規則的定義。

```
<rule>
      <if>
         <tablerow>
            <symbol cost="@cost" name="@name"
               shares="@shares" symbol="@sym"/>
         </tablerow>
      </if>
      <tr>
         <td> <rw:txt name="@name"/> </td>
         <td>
            <link name="link1"
               path="example/portfolio/profile">
              <cntx name="company" value="@sym"/>
              <rw:txt name="@sym"/>
            </link>
         </td>
         <td> <rw:txt name="@shares"/> </td>
         <td> <rw:txt name="@cost"/> </td>
      </tr>
</rule>
```

表 14 Rewrite Rule Format

此範例也是為股票持有表的實作例子，這個範例使用的轉譯規則較為複雜，完整的網頁內容請詳見附錄一。它負責將一個可包含任意子標籤個數的 XML，做需要的轉換，此例為將一個包含任意多寡<symbol>子標籤的 XML 轉換為用 HTML 裡的表格 (Table) 標籤格式來表示，裡頭包含<link>標籤是自定的標籤，需再做一次自定格式內容的轉譯。

第五章 實作範例與效能探討

此章中，將以股票管理服務這個例子的實作來說明的確能夠達到動態的客製化，以及如何利用前章所述的不同類型的頁來完成。接著以此例中各種不同類型的頁，來與一般或常見的架構中的頁來做效能上的比較，以確定如此的架構設計的可行性。

第一節 實作範例

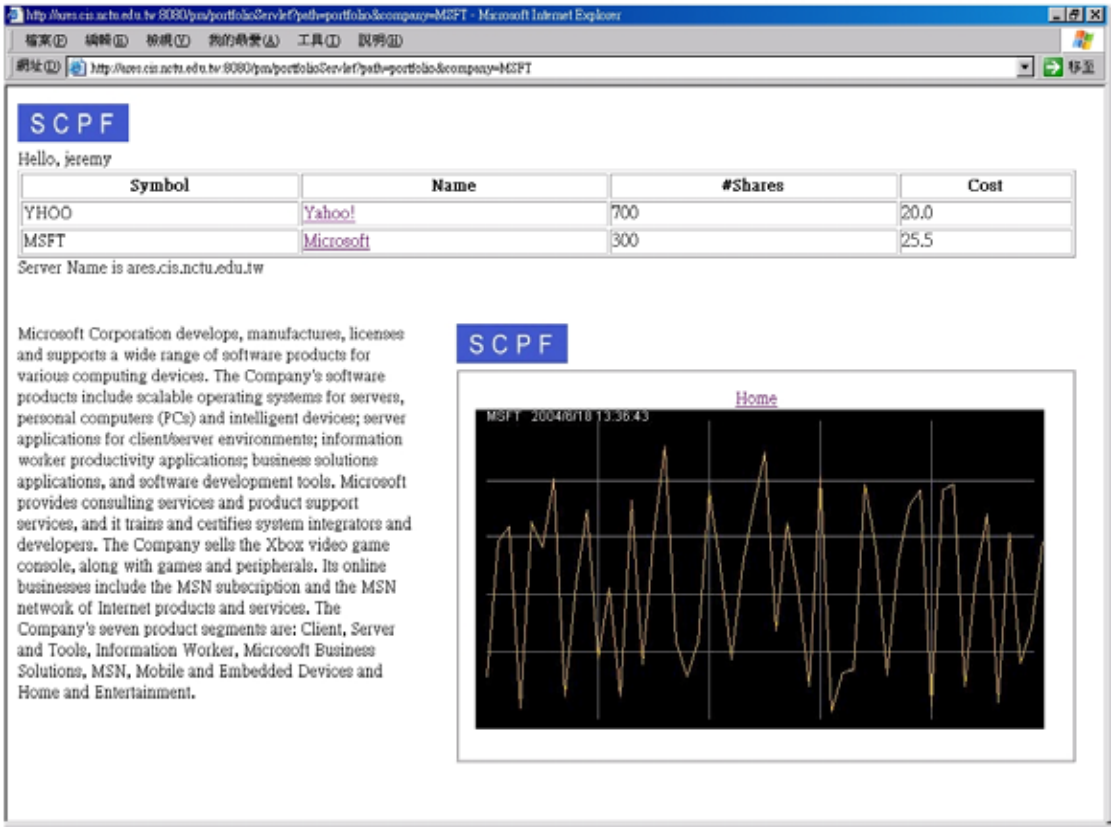

圖 19 Portfolio Example

圖 19中，股票管理服務的介面包含了四個頁，這裡用了複合頁，來將另外三個獨立的頁組合起來。上方和左下角都用了 XSLT 頁，來呼叫客戶持有股票，和公司概況資訊的後端服務，右下角的股票走勢圖是一個經由匯入而產生的遠端頁，此遠端頁即代表著遠端動態產生股價走線圖的服務，它實際上在所屬的遠端的表達服務中也是個 XSLT 頁。頁的實體 XML 內容可參照第四章中的頁的種類一節。

為了呈現出這個範例，我們也實作了這個例子所需要的後端服務。包括一家某公司提供的客戶持有股票清單服務和公司概況資訊服務。和另一家提供的價格曲線圖服務。

另外我們也實作了網頁開發圖形使用者介面 (Page Builder)。在介面上，為了能夠立即觀測到網頁表達服務中每一頁所呈現的內容，一個表達服務除了擁有它的通訊埠，還需要提供 HTTP 通訊埠。我們的介面提供動態且所見即所得(WYSIWYG)，並沒有把網頁部署和開發的時間及過程分開，增加開發效率。

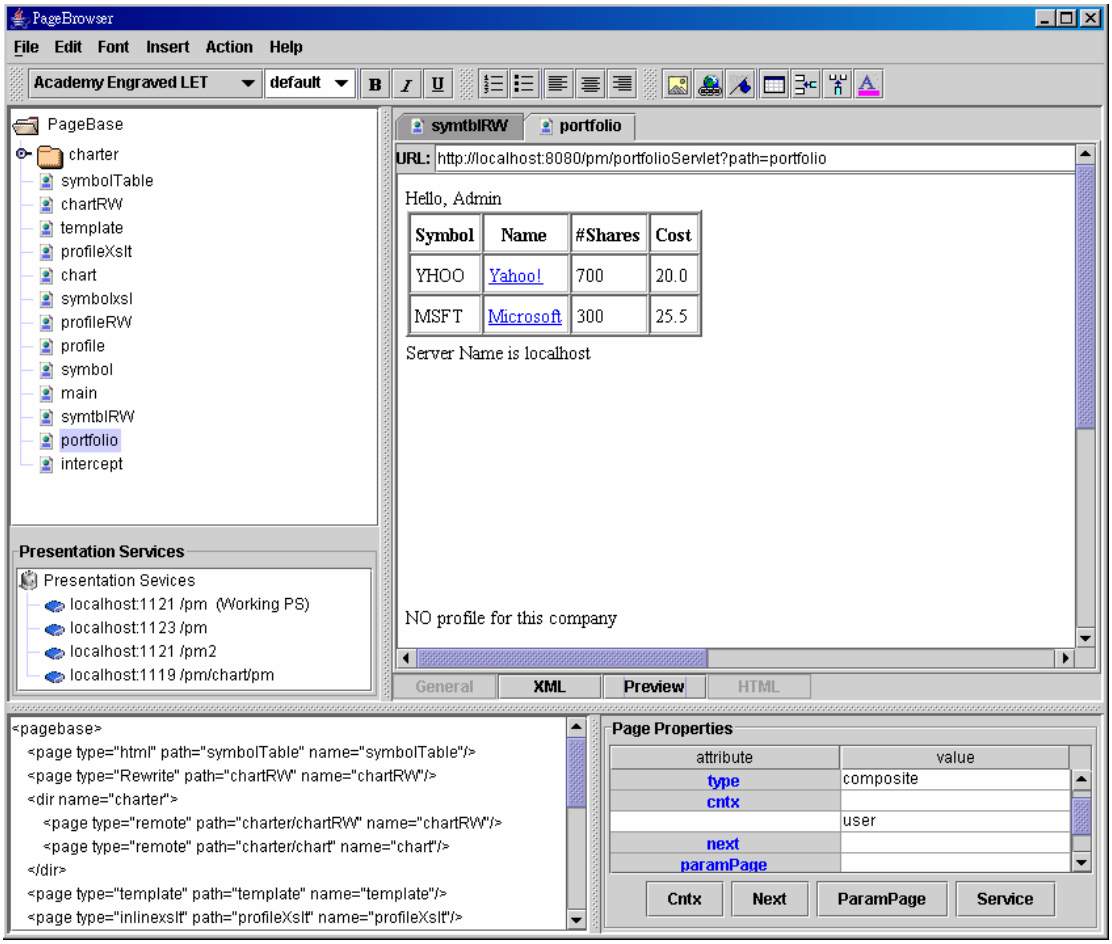

圖 20 Page Builder

第二節 效能探討

在這裡要考量的是遠端頁的內容取得，可以是嵌入或是連結的方式。兩種方式都有各有各的考量和使用時機。而效能的考量，除了嵌入和連結上的比較之外，我們尚考慮表達服務之間的溝通管道。本論文中的實作在表達服務之間使

用 XML 協定，使得能夠做轉換和最佳化，允許在頁的內容產生的時候能夠做不同的最佳化。我們也允許不同的溝通管道，可以是直接的物件指標傳遞 (RMI) 或是使用 SOAP。同樣的頁、內容可以被部署在不同的系統，不同的溝通管道上，而不影響頁的內容。

在效能上，本論文的設計考量了溝通管道技術 (SOAP, socket, RMI)，嵌入或連結的合作方式和語言。在系統的設計上，使用了 TCP/IP 協定，建立 socket，來做為表達服務間的溝通管道。使用標準的通訊協定和我們設計的 XML 訊息格式做為溝通介面，因此表達服務得以跨語言，跨平台。

效能必需在可以接受的範圍內。為了與一般的網頁製作做比較，這裡用 JSP 的方法來實作與例子相同的網頁，動態網頁中使用後端服務的部份也是用相同的遠端呼叫、連結方式，來跟後端服務溝通。這些 JSP 網頁與此例子被放在同一台網頁伺服器的容器上。針對不同性質的頁，這裡做了連續重覆連線所花的時間比較，系統環境設置為：

- CPU：Intel Pentium4 2.0G
- Memory：512 Mb
- Servlet application server：Tomcat 5.0.19
- OS：Windows XP

結果如下表 15與表 16，單位為秒：

次數	STATIC HTML	HTMLPage	Image	ImagePage
100	0.953	1.703	2.641	5.062
1000	7.219	16.312	27.687	49.094

表 15 靜態頁的效能比較

次數	JSP page	RewritePage	InlineXSLT Page	XSLT Page + XSL Page
100	3.672	12.079	7.328	3.656
1000	27.406	105.656	72.922	42.0

表 16 動態頁 (Sym-Table) 的實作效能比較

由 HTMLPage 和 ImagePage 所做的比較可以看出，不做任何處理，只是完整的把靜態固定的內容送出去，就差不多是一倍時間的差別。這時間的付出是花在多了透過 Servlet 來做內容的輸出。在動態網頁的比較方面，以 XSLT 頁來說，跟 JSP 比，時間量為它的二倍，而改寫頁則為四倍。這一點是因為多了轉譯他端 XML 結果為自定 XML 格式並在伺服端把內容轉譯為 HTML 的處理。

遠端頁使用 Linking 和 Embedding 的方式，需要考量其效能上的差異。如下拿動態產生圖檔的頁的例子，做了遠端頁使用 Linking 和 Embedding 兩種方式和非遠端頁的效能比較。

次數	Local	RemotePage (Linking)	RemotePage (Embedding)
100	21.343	22.25	44.077
1000	206.797	226.656	407.231

表 17 遠端頁 (Chart) 運作方式之效能比較

由表 17，可以看出說，Embedding 比 Linking 所花的時間多了一倍。以這個例子來說，傳送圖檔與儲存的時間占了相當大的時間量，加上修改網頁中圖檔的 URL 以及超連結的路徑所花的時間，因此造成了如此時間上的代價。

由上述的比較可看出，使用這個架構，基本上會大約比一般的集中式的網頁伺服器多耗費一倍的工作量。雖然效能上幾近差了一倍之多，但在實作上，還有改善的空間，例如能夠讓網頁伺服器直接處理我們自定 XML 格式的頁，而非透過 Servlet 作為中央控制元件 (Controller)，如此相信可大幅地提昇效能。而現今已有的表達層的網路應用程式架構，大多也都建置在 Java 技術中的 Servlet 之上。

為了要達到提供網頁的服務，有效率且模組化的開發，在某些情形下以如此的效能做為犧牲或許是可以被接受的，尤其是在運算能力愈趨強大的今天。

而 XSLT Page，Rewrite Page 依照 rule 來將動態內容轉為適當的格式來描述。當動態取得的內容愈多愈複雜時，轉譯規則 (Rule) 就愈難寫。因為一頁只用一個轉譯規則集合 (rule set) 來對整頁做轉述。複雜的 Rule 是寫作網頁上的缺點。

對於網路應用程式架構 (Web Application Frameworks) 來說，通常還有許多問題需要考量以完成一個完整的架構。例如，安全、全球化 (Internationalization)、表格驗證、錯誤處理、支援執行 Web Service，支援某些資料存取技術 (像 JDBC、EJB、iBATIS)，等等。但有些已經不在本篇論文所提的架構所針對的重點。但這些功能和整合的提供，似乎是常被與其他的網路應用程式架構來做比較。

第六章 相關研究

本章討論一些相關論文提出的架構和其概念。並討論一些現在已常用且標準的技術。分為如何撰寫內容，元件間如何互相溝通以完成內容，如 JSP，Struts；以及跟 Page Model 有關的內容的表示方法，如包括 HTML，XML + XSLT，CSS，FO (Format Object)。

WCCM (Web Content Component Model) [16]，為一個內容導向的 web 架構原型，其主要概念為，一個 web 應用程式以元件的立場來看，在設計上被視為許多 web 內容元件 (Content Component)，和一些其他提供某內容上的服務的服務元件的集合。

WCCM 強調元件化，它把內容視為一個元件，在概念上，將內容和元件結合在一起。並欲藉此原型，增進 web 應用程式設計與維護的效率。WCCM 並描述此其架構中，內容元件與服務元件的關係。它將內容元件只專注於內容的處理，表達，而將處理內容上的特殊功能都交由服務元件來輔助。甚至將內容的交換，交換資料格式上的轉換，都交由其他元件來處理。

WCCM 將軟體工程的精神，應用在 WEB 應用程式上，它的概念在於強調元件化帶來系統開發上的好處。將內容視為一個元件，並規範其與其之間的關係，以這樣的觀點來看，能夠提昇設計與維護上的效率，也降低重新設計與維護上的成本並降低其工作量。

在本論文中所提出的頁原型，裡頭的設計概念也是將頁視為一個負責內容組織和表達的元件，並將其他特殊的功能都可交由其他元件來完成，比如，搜尋，統計，網頁流向管理，帳號認證管理，等等。而提供這些服務的，可能是某種類型的頁，或是其他伴隨的服務。而元件與元件之間的溝通，也是經由 XML。

Bull's Eye [17]提出將 UI 元件設計分為五種層級的概念，以不致於設計 UI 時太過一般或是只符合於某特殊的實例下使用。它強調出 UI 元件的使用性的問題。這五個層級分別為元件 (Component)，頁樣版 (Page Template)，頁流 (Page Flow)，互動樣式 (Interaction Model)，與包羅萬象的原則 (Overarching Principle)。藉由在每一層級設計它的設計方針的概念，達到 UI 元件能夠跨系統，

跨網站的被使用。

以本論文中的頁模型為基礎，設計一個頁時，尚需要考量此頁是否能夠被大多數的情形下來使用，包括了國際化，瀏覽器相容性，和存取性等問題。Bull's Eye [17]提出的第二層級頁樣板的概念，與本論文中提出的頁模型中的樣版頁，是相同的概念，能夠在一個頁上結合多種元件。而頁流是第三個層級，將多個頁樣版結合在一起，依據使用者的狀態與操作，而決定接著流向不同的頁。定義頁流向的概念，在本論文的設計中，也是將它從頁元件裡頭獨立出來，交由專門的支配頁或是另外一個附屬的服務來負責。

Saimi's PlayFwas Framework [15]讓初級的開發者能夠經由一套程序和介面定義文件來有效率地開發 web 應用程式，降低開發的時間，並改善維護和擴充的問題。在其中它的架構是以一個 Java Servlet，一個 JavaBean，一個 JSP 來完成一個頁的內容。它們的角色分別為區隔請求並檢查輸入資料，收集並維護產生的資料，和版面的設計和編排。藉由基本的 Servlet 和 JavaBean 已提供大部份共同基本的實作，網站設計者在開發 Servlet 和 JavaBean 的時候只需繼承並覆寫掉一些方法。

本論文提出的架構與 PlayFwas 的架構所針對的問題不同，但在架構的設計上也對其做了個參考。本論文的架構並沒有為了一個頁 (Screen)，而產生了一個 Servlet 和 JavaBean。但相對的，將這些工作轉移到由表達服務中共通的元件來完成。整個網站的開發上就只有頁和服務兩種元件，並可交由不同的人分別完成。頁的工作只負責表達，而資料的取得，以及請求的檢查與管理，和其他內容上的特殊處理，都交由某元件或特定服務來完成。我們藉由將頁元件化，和分散式的架構提昇頁的使用性，而改善維護和擴充的問題。

網際網路入口網站 (Portal)，如 Yahoo!，Lycos，等提供大量資訊的網站，都常會讓使用者客製化他們所需的資訊，版面，以及外觀。MyPersonalizer [18]為一個用來建造入口網站的架構。它採用了 MVC 架構和分層結構模式 (Layers architectural patterns)。入口網站的架構強調 model 的部份，後端的個人化服務依照使用者的個人喜好來提供資訊，將個人化的資料對應到關聯式資料庫 (relational database) 裡的資料，並且每個個人化服務能夠有效地讀取它們所需的

個人化資料。個人化的儲存物件應該被用一個一般的方法來表示且依據關聯式資料模型 (Relational Data Model) 來對應。

在本論文中的分散式表達架構，並不專注在使用者個人化資訊的管理，只強調出這些個人化資訊能夠轉為網頁中所需的 Context 或是連結後面的參數，來完成調整及客製化遠端網頁的需求。通常此類型的大型網站所得到的個人化資料相當的龐大，因此個人化的服務都是交由後端服務和資料庫來完成。至於前端網頁介面 (View) 的部份，只紀錄著使用者身份，和一些按鈕等使用者介面的狀態。

CSS (Cascading Style Sheets) 最早是搭配 HTML 語言的樣式語言，但 CSS 也可作為輸出 XML 文件的樣式語言。CSS 能夠針對指定的標籤定義全新的樣式，但 CSS 只是單純的定義樣式，XSL (eXtensible Style Language) 提供更強大的功能。

XSL 在功能上分為兩個部份，分別是轉換 XML 文件，和格式化 XML 文件。其中轉換的部份是將 XML 文件架構轉換為另一個 XML 架構的文件，此轉換部份的規格稱為 XSLT (XSL Transformation)。XSLT 並不是在顯示 XML 元素內容，只是將 XML 元素轉換成其他文件的格式。而格式化 XML 的部份為將 XML 文件轉換為另外一種格式的文件，例如 PDF 檔。先藉由 XSL Stylesheet 將 XML 文件轉換為內部仍為一個樹狀結構的文件 (XSL FO)，而此樹狀結構的節點是格式物件 (Formatting Objects)，例如，段，章。此樹狀結構即對應到此文件即將被輸出或表達在螢幕上的架構。這個樹接著被格式化成輸出，藉由使用 Stylesheet 中定義的格式物件裡的格式屬性，描繪此 XML 文件的元素，。

第一個步驟是由 XSLT Processor 轉換輸入的 XML 文件為 XSL FO 實體。第二個步驟是由一個支援 XSL FO 的描繪者 (Renderer) 將 XSL FO 描繪出來。

因此藉由 XSLT 轉換語言，如本論文中頁模型中的 XSL 頁，提供轉換規則，利於達到多樣的輸出表達結果。

第七章 結論

總結本論文的系統架構，此系統使網頁容器 (Container) 更具模組化，並使其成為一個提供網頁表達的網路服務。也因為經由網頁表達服務的提供，能夠達到將重要的後端服務被隱藏，對於企業內部的整合或是與外部的合作都將能提供另一個選擇的方案。此外，經由這樣的分散式架構，網頁開發的工程可以被分散，且因為部份的網頁開發責任被交予後端服務的特定領域的開發者，可以達到更適當的分工，提昇網頁的開發效率。而這些分散的網頁表達服務，就跟後端服務一樣，經由以 XML 為基礎的溝通協定為介面來互相合作，最後把完成的網頁內容傳送給網頁瀏覽器。

除了達到良好的分工之外，文中並提出了為了達到本論文的目標而需要應付的議題。藉由提供這樣的網頁表達服務，可造就出表達層的商業市場，而可將後端的服務包裝隱藏起來。

未來仍會在效能的考量上著手，在服務間的溝通上求取最佳化，並實作更多的應用實例。並且將本論文的網頁表達架構能跟其他技術，如 JSP，Struts 等達成整合，採用現有不同的樣版語言和熟悉的 MVC 開發架構，以利用已存在且被普遍使用的技術與資源。

參 考 文 獻

[1] JSF. Java Server Faces. Sun corp. http://java.sun.com/j2ee/javaserverfaces/

[2] The Apache Software Foundation http://www.apache.org

[3] W3C (World Wide Web Consortium) http://www.w3c.org

[4] Struts, The Apache Software Foundation http://jakarta.apache.org/struts/

[5] MVC (Model-View-Controller). J2EE Patterns. Java BluePrints. Sun corp.

http://java.sun.com/blueprints/patterns/MVC-detailed.html

[6] Chuck Cavaness. *Programming Jakarta Struts* O'Reilly November 1, 2002

[7] Cocoon, The Apache Cocoon Project. The Apache Software Foundation
http://cocoon.apache.org/

[8] Jakarta Velocity, The Apache Jakarta Project. The Apache Software Foundation
http://jakarta.apache.org/velocity/index.html

[9] Jakarta Turbine, The Apache Jakarta Project. The Apache Software Foundation
http://jakarta.apache.org/turbine/index.html

[10] Jakarta Tomcat, The Apache Jakarta Project. The Apache Software Foundation

http://jakarta.apache.org/tomcat/

[11] JetSpeed, Apache Portals Project. The Apache Software Foundation
http://www.xml.com/lpt/a/2000/05/15/jetspeed/index.html

[12] Portal. IBM WebSphere Application Server
http://www-306.ibm.com/software/webservers/

[13] Style Sheet, World Wide Web Consortium http://www.w3.org/Style/

[14] Web Services, World Wide Web Consortium http://www.w3.org/2002/ws/

[15] Saimi, A.; Syomura, T.; Suganuma, H.; Ishida, I.; “Presentation Layer Framework of web application systems with Server-side Java technology”, Computer Software and Applications Conference, IEEE 2000, pp 473 - 478

[16] Qingshan, L.; “Study of a Content Oriented Web architecture Model”, Computer Networks and Mobile Computing, International Conference, IEEE 2001. pp 3 – 7.

[17] Betsy Beier, Misha W. Vaughan “The Bull’s-Eye: a framework for Web Application User Interface Design Guidelines”, Proceedings of the conference on Human factors in computing systems. ACM press 2003. pp.489 – 496.

[18] Fernando Bellas. *A Flexible Framework for Engineering “My” Portals*, International World Wide Web Conference. Proceedings of the 13th conference on World Wide Web. ACM press 2004. pp.234 - 243

[19] Alan W. Brown. *Large-scale, component-based development* .Prentice Hall PTR May 30, 2000

[20] Ezra Ebner. “The Five-Module Framework for Internet Application Development”, ACM Computing Surveys (CSUR) Volume 32 , Issue 1es. Article No. 40.

[21] Liu Chao, He Keqing, Liu Jie, Ying Shi. “Some Domain Patterns in Web Application Framework”, (27th Annual International Computer Software and Applications Conference). IEEE 2003. p674-678.

[22] Gustavo Rossi, Daniel Schwabe, Robson. “Designing Personalized Web Applications” WWW10th, ACM 2001. pp 275 – 284.

[23] Udi Manber, Ash Patel, and John Robinson. “Experience with Personalization on YAHOO!”. Communications of the ACM. Volume 43 , Issue 8, 2000. pp35 - 39

[24] The JavaServer Pages Specification. Sun corp. http://java.sun.com/products/jsp

[25] The Java Servlets Specification. Sun corp. http://java.sun.com/products/servlet

[26] Tidwell, J. *UI Patterns and Techniques: About Patterns*. May, 2002. Available at: http://time-tripper.com/uipatterns/about-patterns.html

[27] Douglas K. van Duyne, James A. Landay, Jason I. Hong, *The Design of Sites*, Addison Wesely, 1st edition (July 22, 2002)

[28] Sun ONE Application Server 7 Server Architecture Overview. Sun corp. http://docs.sun.com/

附　錄　一

實作股票持有表網頁範例的改寫頁內容

```
<page type="rewrite">
   <cntx name="user" default="dummyUser"/>
   <next name="link1"/>
   <service host="localhost" name="symbol"
      path="symboltable" port ="1119"/>
   <body>
      Hello, {user}
      <call service="symbol">
        <getSymbolTable user="{user}"/>
      </call>
   </body>
   <rules>
      <rule>
         <if>
           <body>
             <rw:txt name="txt"/>
             <rw:var name="symtbl"/>
           </body>
         </if>
          <body>
            <rw:txt name="txt"/>
            <rw:rw>
             <rw:var name="symtbl"/>
             </rw:rw>
          </body>
      </rule>
```

```
<rule>
  <if>
       <symboltable>
          <rw:var name="symbol"/>
          <rw:list name="symbols"/>
       </symboltable>
  </if>
  <table border="1">
     <tr>
        <th>Symbol</th>
        <th>Name</th>
        <th>#Shares</th>
        <th>Cost</th>
     </tr>
     <rw:rw>
        <tablerow>
          <rw:var name="symbol"/>
        </tablerow>
     </rw:rw>
     <rw:rw>
        <tablerows>
           <rw:list name="symbols"/>
        </tablerows>
     </rw:rw>
  </table>
</rule>
```

```
   <rule>
      <if>
         <tablerow>
            <symbol cost="@cost" name="@name"
               shares="@shares" symbol="@sym"/>
         </tablerow>
      </if>
      <tr>
         <td> <rw:txt name="@name"/> </td>
         <td>
            <link name="link1"
                path="example/portfolio/profile">
              <cntx name="company" value="@sym"/>
              <rw:txt name="@sym"/>
            </link>
         </td>
         <td> <rw:txt name="@shares"/> </td>
         <td> <rw:txt name="@cost"/> </td>
      </tr>
  </rule>

  <rule>
     <if>
        <tablerows>
           <rw:var name="symbol"/>
           <rw:list name="symbols"/>
        </tablerows>
     </if>
     <rw:rw>
        <tablerow>
           <rw:var name="symbol"/>
        </tablerow>
     </rw:rw>
     <rw:rw>
        <tablerows>
           <rw:list name="symbols"/>
        </tablerows>
     </rw:rw>
  </rule>

   <rule>
     <if>
        <tablerows/>
     </if>
   </rule>

  </rules>
</page>
```

附　錄　二

Servlet 和表達服務的配置與介面範例

Configuration

Servlet:

```
<web-app>
  <display-name> War name</display-name>
  <description> War description </description>
  <mappings>
      <mapping name="portfolioServlet" url="/portfolioServlet"/>
      <mapping name="chartServlet" url="/chartServlet"/>
      <mapping name="testServlet" url="/testServlet"/>
  </mappings>
  <servlet>
      <servlet-name>portfolioServlet </servlet-name>
      <description>PM Service PageServlet </description>
      <servlet-class> pm.PageServlet </servlet-class>
      <init-param name="host" value="ares.cis.nctu.edu.tw" />
      <init-param name="port" value="8080"/>
      <init-param name="uri" value="pm/portfolioServlet"/>
      <init-param name="hostCfg" value="webapps/pm/config/pm/portfolio.conf"/>
      <init-param name="pmPath" value="pm"/>
  </servlet>
  <servlet>
      <servlet-name>chartServlet</servlet-name>
      <description> PM Service PageServlet </description>
      <servlet-class> pm.PageServlet </servlet-class>
      <init-param name="host" value="ares.cis.nctu.edu.tw"/>
      <init-param name="port" value="8080"/>
      <init-param name="uri" value="pm/chartServlet"/>
      <init-param name="hostCfg" value="webapps/pm/config/pm/chart.conf"/>
      <init-param name="pmPath" value="pm"/>
  </servlet>
  <servlet>
      <servlet-name>testServlet</servlet-name>
      <description> PM Service PageServlet </description>
      <servlet-class> pm.PageServlet</servlet-class>
      <init-param name="host" value="ares.cis.nctu.edu.tw"/>
      <init-param name="port" value="8080"/>
      <init-param name="uri" value="pm/chartServlet"/>
      <init-param name="hostCfg" value="webapps/pm/config/pm/portfolio.conf"/>
      <init-param name="pmPath" value="pm2"/>
  </servlet>
</web-app>
```

PageSpaceModule:

<config>
<pagebase path="D://home/work/pm/portfolio/pagebase"/>
<db root="D://home/tomcat/Tomcat 5.0/webapps/ROOT/portfolio/db"
rootURL="http://ares.cis.nctu.edu.tw:8080/portfolio/db"/>
</config>

Interface

between Presentation Services:

- get Page Xml Content (Embedding)
  - query:

    <getPageContent path="a/b/c"/>

  - return:

    <page>
    .....(Page Xml Content).....
    </page>

- get Required Page Context (Linking)
  - query:

    <getRequiredCntx path="a/b"/>

  - return:

    <requiredCntx>
    <cntx name="abc"/>
    <cntx name="cde"/>
    .....
    </requiredCntx>

- import remote Presentation Service
  - query:

    <importRemote/>

  - return:

    <pagebase httpServURL="http:// + hservHost + : + hservPort + / + hservURI">
    <page type="html" path="abc" name="abc"/>
    <page type="image" path="abc" name="abc"/>
    <dir name="a">
    <page type="html" path="a/b" name="b"/>
    <page type="html" path="a/c" name="c"/>

```
    </dir>
  </pagebase>
```

for GUI:

- get accompanied File (HTML, XSL files)
  - query:

    ```
    <getFile path="a/b"/>
    ```
  - return:

    ```
    <htmlContent>  .....  </htmlContent>
    ```

    or

    ```
    <Xsl/> (XSL File Xml)
    ```

- get Page Sets
  - query:

    ```
    <getPageSpace/>
    ```
  - return:

    ```
    <pagebase>
      <page type="html" path="abc" name="abc"/>
      <page type="image" path="abc" name="abc"/>
      <dir name="a">
        <page type="html" path="a/b" name="b"/>
        <page type="html" path="a/c" name="c"/>
      </dir>
    </pagebase>
    ```

- new a Page
  - query:

    ```
    <newPage path="a/b" type="composite"/> or <newPage path="a/b"
            type="html" file="abc.htm"/>
    ```
  - return:

    ```
    <newPageSuccess path="a/b"/> or <PageExists/>
    ```

- save a Page
  - query:

    ```
    <savePage path="a/b">
      <page> ..(Page Xml Content)...  </page>
    </savePage>
    ```
  - return:

    ```
    <savePageOK/> or <savePageError/>
    ```

- save the file
  - query:

    <saveFile path="a/b">

    <![CDATA[(File Content)]]>

    </saveFile>
  - return:

    <saveFileOK/> or <saveFileError/>

- new a Dir
  - query:

    <newDir path="c/d"/>
  - return:

    <newDirOK/> or <DirExists/>

- delete a Page
  - query:

    <deletePage path="c/d"/>
  - return:

    <delete state="true"/> or <delete state="false"/>

- import
  - query:

    <import host="localhost" port="1121" path="pm" destPath="a/b/c"/>
  - return:

    <importOK/>