

國立政治大學資訊管理研究所

碩士學位論文

指導教授： 楊建民 博士

雲端運算 SOA 服務平台架構與運作機制之研究

A Study of Cloud Computing SOA Service
Platform Architecture and Operation Mechanism

研究生：蔡鈞華

中華民國九十八年七月

誌 謝

這本論文能夠順利付梓，首先要感謝黃瓊玉老師的熱心指導、季延平老師的諄諄善誘與黃明達老師快意凌厲的指正，讓這份論文更為嚴謹與完整；衷心感謝楊建民老師在學術研究上的指導與專業經驗的傳授，令學生收穫良多，楊老師樂於表達關心，分享自己的人生經驗，並適時解決學生的疑難與困惑，展現出智慧通達的學者典範，如武俠小說裡的師父般，扮演亦師亦父的角色，這兩年師生間的薰陶點滴，將會是未來面對人生未知挑戰的穩固基石。

來政大最大的收穫，是認識一群形形色色的好朋友們：感謝俊宏一路上的陪伴與支持，並與世傑在同研究領域上給予許多意見與回饋；春美長久以來的關心與照顧，散發穩定人心的溫情光輝；繼鴻不時扮演冷面笑匠，帶給大家許多歡樂與睿智的見解；婉瑋總是掛著滿臉笑容，有著令人羨慕的許多理想與堅持的鮮明形象。承翰、孝文、少華、宜儒、榕芝與世蓉這群熱心的學弟妹，還有月純學姐的照顧與鼓勵，讓整個實驗室的氣氛如大家庭般團結與互助。除了實驗室的大家，與至善同學的相互惕勵、支持，是研究所中培養出的難得友誼；感謝鈺婷這兩年來的包容與忍讓，因為妳讓我的人生更臻完整，何其有幸，在研究所過程能有你們陪伴，謝謝你們。

人生旅程中要感謝的人太多，從他人身上獲得的也難以細數，祝福所有曾經關心、幫助我的朋友們，希望你們也能感受到此刻的幸福與喜樂。在此要感謝一直以來支持我的家人，親愛的父母辛苦的工作，提供最大的空間實現我的夢想；妹妹天華總是運用她的巧手，幫我搞定生活中大小事情；弟弟一華充滿自信與想法，卻又貼心與善解人意；振清表哥的經驗分享與相互鼓勵，帶來的支持與幫助，謝謝你們無私的付出，這本論文是屬於你們的。

生命是由連串事件所累積而成，回首兩年研究所生涯，每次的喜怒哀樂，都加深點生命的厚度，這個歷程無法跳躍，也沒有捷徑，這路上最寶貴的收穫，是體會學然後知不足的真正含意，碩士並不是學習的終點，而是承受更多挑戰與責任的入場券。還記得法國里昂那個晚上，皎潔月光配上徐徐清風，身旁延伸的鐵軌引領回首，以往的無知與愚昧，都在人生旅程中逐漸滌清，成熟的人不問過去，冀求也無風雨也無晴的灑脫。

蔡鈞華 謹誌於
國立政治大學資訊管理研究所
中華民國九十八年七月二十四日

摘要

隨著雲端運算基礎架構的發展，如何結合企業資訊服務與外部雲端運算資源，是當前重要的研究課題。目前市面上一些主要的資服業者，如 Google、IBM 以及 HP 等，大力推動並提供強大雲端運算能力與完善的應用服務環境；然而目前雲端運算架構多需將整體應用與資料放置於外部平台，引發企業導入雲端運算時控制權移轉與資訊安全的疑慮。為解決雲端運算所衍生出內外部資源有效配置的問題，與企業導入上的資安疑慮，本研究結合雲端運算與服務導向架構，發展出雲端運算服務導向架構運作機制，包含服務串接、資源配置、資料控管與管理機制，並依照這些運算機制建構出以服務介面層、中心控管層、服務供應層與基礎架構層組成的『雲端運算服務導向架構平台』。

服務串接機制負責連接服務需求方、服務元件商與服務套餐業者；資源配置機制具備負載平衡、平行運算與容錯機制等功能；資料控管機制利用部分授權的方式讓資料以片段的方式存在，避免有心人士竊取的風險，對於資料運算的安全性，讓服務供應方只能取得經授權的必須資料，關鍵資訊僅掌握在資料擁有人手中，以確保資料安全性；管理機制實作了 AAA 協定，認證機制決定服務使用者的身份與服務範圍。

而服務平台中的應用介面層利用開放平台的優勢，降低服務介面開發的負擔；中心控管層連接服務供應方與服務需求方，掌控完整的服務流程，以及使用外部雲端運算與資料中心之資源；服務供應層結合各式應用服務與資料來源，藉由將運算與資料分離的方式，讓服務需求者保持對資料的控管權力，以解決企業導入雲端架構時對資安的疑慮；基礎架構層可連結雲端運算資源，未來可利用各雲端業者所提出的服務架構、運算能力以及儲存資源，並依照個別平台的優勢與特性，充分發揮雲端運算架構的效益。

最後本研究藉由發展金融發票融資之應用，以說明本服務平台之可行性與整合外部雲端運算服務的優越性，以及未來研究方向。

關鍵詞：雲端運算、服務導向架構、MVC 架構、Google 運算、金融發票融資

Abstract

Cloud Computing heralds an evolution of business, this creates a research question on how to integrate enterprise information service and external Cloud Computing resource. Information service big players like Google, IBM and HP, providing powerful Cloud Computing ability and application environment. However, when enterprises adopt Cloud Computing architectures proposed recently, and transfer applications and data to external platform arouses the problems of management authority shifting and information security.

We proposed a 4-layer service platform and related operation mechanism that incorporates Google Cloud Computing environment and Service Oriented Architecture in a sense to provide new paradigm of system development. Service Combination Mechanism connects service consumer, service provider and service menu. Resource Allocation Mechanism provides loading balance, parallel processing and fault-tolerant features. Data Control Mechanism only authorizes the necessary part of data to avoid the risk of information security. Management Mechanism implements AAA protocol.

Service Interface Layer utilizes the advantage of open platform and reduces the burden of interface development. Control Center Layer connect service providers and service requesters, while handling service flow and using the resource of external cloud computing and data center. Service Provider Layer reserves the management authority and ownership of the data resources. Infrastructure Layer assemble service framework, computing ability and storage resource that are provided by Cloud Computing providers. We believed that with the 4-layers service platform proposed in this paper, one can leverage the advantages and merits of Cloud Computing architecture.

Finally, by developing a invoice financing case based on proposed platform, we are able to justify the service platform's feasibility and its advantage with the use of cloud computing.

Keywords: Cloud Computing 、 Service Oriented Architecture 、 MVC framework 、 Google computing

目 錄

第一章 緒論	1
第一節 研究背景.....	1
第二節 研究動機.....	2
第三節 研究目的.....	3
第四節 研究架構.....	5
第二章 文獻探討	6
第一節 網路服務架構.....	6
一、 服務導向架構.....	6
二、 網路服務.....	9
第二節 運算服務架構.....	11
一、 網格運算.....	11
二、 公用運算.....	12
三、 軟體即服務.....	12
四、 雲端運算.....	15
第三章 研究設計	21
第一節 MVC 架構	21
第二節 開發工具.....	22
第三節 網路服務流程描述語言.....	23
第四節 雲端運算環境.....	24
一、 Google 應用環境.....	24
二、 Hadoop 運算架構.....	26
第四章 服務平台建置	28
第一節 雲端運算服務導向架構運作機制.....	29
一、 服務定位.....	29
二、 運作機制.....	29
三、 設計架構.....	36
第二節 雲端運算服務導向架構平台.....	38
一、 應用介面層.....	38
二、 中心控管層.....	38
三、 服務供應層.....	41
四、 基礎建設層.....	42
第五章 服務平台實作與展示	44

第一節 金融發票融資應用案例.....	44
第二節 服務平台分析與設計.....	47
一、服務需求方.....	47
二、服務元件商.....	48
三、服務套餐業者.....	55
第三節 服務平台展示.....	57
第六章 結論與未來展望	63
第一節 結論與建議.....	63
一、雲端運算服務導向架構運作機制.....	63
二、雲端運算服務導向架構平台.....	64
三、研究建議.....	65
第二節 未來研究方向.....	66
參考文獻	67
附錄 A：金融發票融資應用案例 BPEL 描述.....	70

圖目錄

圖 1：典型服務導向架構圖.....	8
圖 2：Azure 平台架構圖.....	18
圖 3：salesforce.com 平台架構圖.....	19
圖 4：MVC 架構圖.....	22
圖 5：Google Cloud Computing 服務架構示意圖.....	26
圖 6：Mapreduce 架構圖.....	26
圖 7：負載平衡記錄資料結構.....	32
圖 8：負載平衡監控程式片段.....	32
圖 9：Mapreduce 運行架構圖.....	34
圖 10：資料控管架構圖.....	35
圖 11：雲端運算服務導向架構圖.....	37
圖 12：雲端運算服務導向架構流程圖.....	39
圖 13：服務平台運作機制.....	40
圖 14：應用服務與資料分離示意圖.....	42
圖 15：金融發票融資案例服務流程概念圖.....	46
圖 16：『ServiceAccounting』服務元件資料結構.....	49
圖 17：銀行內部資料結構.....	51
圖 18：雲端資料庫示意圖.....	52
圖 19：雲端發票資料庫示意圖.....	52
圖 20：服務元件註冊清單畫面.....	53
圖 21：雲端服務平台使用者介面示意圖.....	54
圖 22：服務套餐 BPEL 描述片段.....	56
圖 23：服務套餐註冊清單畫面.....	56
圖 24：服務平台登入畫面.....	57
圖 25：服務平台主畫面.....	58
圖 26：服務平台已購買服務選單畫面.....	58
圖 27：『BankManager』服務主畫面.....	59
圖 28：『BankManager』服務新增資料選單.....	60
圖 29：『CheckCredit』服務執行結果.....	61
圖 30：『CheckCredit』服務運算資源利用表.....	61
圖 31：服務平台統計資訊畫面.....	62

表目錄

表 1：服務導向架構技術特性.....	9
表 2：BPEL 語言結構.....	24
表 3：服務元件註冊參數表.....	30
表 4：貸款信評計算公式表.....	49



第一章 緒論

第一節 研究背景

當企業面對營運環境變化的挑戰，為即時回應改變的需求，系統開發的趨勢考慮重複利用舊有與外部的服務，並以組合式應用服務(Composite Application)模式來滿足需求。這樣的思維已漸漸取代從頭開發全新的系統，事實上在 2003 年已經有許多商業應用服務採用外部資源來完成，也就是利用組合現有資源的方式來滿足系統開發的要求。因此，許多系統開發專案建立了可程式化介面(Programmatic Interfaces)，與舊有系統服務或外部資源進行整合，將異質平台(Heterogeneous)組合而成的應用服務導入到實際交易過程中。在實際的企業營運模式裡，組合並整合舊有與新開發的應用元件(Component)來完成服務，正滿足服務導向架構(Service Oriented Architecture)環境所要求的特性(Natis 2003)。根據研究報告指出，2008 年服務導向架構將成為普遍的系統開發模式，取代已沿用 40 餘年的整體系統開發(Monolithic Software Architecture)模式(McCoy and Natis 2003)。

服務導向架構是一種軟體架構，要求軟體元件間關係鬆散，並且提供開放的服務介面讓服務供需雙方溝通，因為當組織面臨問題時，可能需要分散在不同領域下的能力或資源，服務導向架構是將分散的能力或資源將其組織起來，並用來解決組織所面臨問題的一種準則。服務導向架構能夠整合業務流程，在技術上使用安全且標準的服務元件，而這些服務元件能夠因業務上的改變而再利用或重新組合(Pallos 2001; Bieberstein, Bose et al. 2005)。

雲端運算是以服務的形式提供運算與儲存資源，利用簡便的方式動態的將服務延伸到更多的運算與儲存資源上，以求得更好的運算效能與服務可靠度。(Morrison Aug 20, 2008.)指出相關平台業者Google宣布加入IBM、HP、Intel與Yahoo形成聯盟，並宣稱將會投入數以百萬計的資金在雲端計算的相關專案上，代表未來的資訊系統發展，將會與雲端運算更加密切的結合，點出這將會是企業資訊服務未來與服務領導平台整合的發展趨勢。

知名網路書城亞馬遜(Amazon.com)於 2002 年推出 AWS(Amazon Web Services)平台服務，容許遠端的網路服務(Web Service)得以在這個平台上運作。亞馬遜提供網路服務給其他的網站或是使用者端的應用程式，這些網頁服務並非直接提供給使用者，而是提供方便的開發介面，由軟體開發者來進行開發後，於 AWS 平台上發佈，供使用者來使用。在 2007 年 6 月，亞馬遜聲稱有超過 33 萬的軟體開發者註冊了 AWS 服務(Amazon.com 2007)。

隨著搜尋網站業務的成功，Google 也推出許多強大的網頁服務，包括如電

子郵件、線上地圖、辦公室應用軟體、社群服務與影片分享等免費服務。有些公司開始外包次要的服務交由 Google 提供，將營運重心專注於本身的核心競爭業務，換句話說，企業會讓原本應該是自己經營的業務轉由其它合作伙伴來替他完成(Gray 2007)。如 SaaS (Software as a Service)的領導廠商 SalesForce.com，宣佈參加 Google Apps 計畫，採用 Google 的 Gmail、Google Docs、Google Talks 與 Google Calendar 等應用服務，導入到現有的網頁版 CRM 應用服務中(Salesforce.com 2007)。

未來在微軟、昇陽等公司相繼投入雲端運算平台的開發工作後，可預見將發展出更多雲端運算資源，也就代表著更多機會的升起與產業的殞落。以運算資源為例，以往在進行科學運算時，需投資大量成本於超級電腦的建置與維護，若今天可以輕易地利用第三方的運算資源，以完成本身的工作，對於運算基礎建設的投資將可大幅降低，在大型運算市場萎縮的同時，卻興起另外一塊更蓬勃的市場，那就是充分利用現有資源，並能為企業解決問題的第三方加值平台，當這種專業分工的發展模式逐漸成熟，企業必須思考如何在這樣的發展氛圍中，重新找到自己的價值定位並進行價值鍊上的切分。

第二節 研究動機

綜觀網格運算(Grid Computing)、公用運算(Utility Computing)到軟體即服務(SaaS)的發展歷史，可以發現服務架構上的逐步演進過程。網格運算源自於能源網格的概念，發展出讓計算與儲存資源能形成互相流用的網格結構，這種以網格為出發點的運算思維，逐步構成全球性的網格網絡，改變了資料發布與連接應用服務的方式，也衍生出新的系統架構發展方式；當匯集了分佈於各處的運算資源，公用運算概念的提出，希望讓運算資源能夠如同水電等基礎建設一樣，隨取即用並以使用量計費，改變了企業的 IT 投資策略，並且促使資訊基礎架構朝向共用基礎架構的概念靠攏；軟體即服務這個概念也順勢而生，並且造成企業將服務流程外包給軟體即服務業者的風潮，透過網路來進行服務存取，而不需要負擔龐大的 IT 基礎建設投資成本，並可以達成多會員、可擴充性與服務組合等效益。

雲端運算(Cloud Computing)提供者，讓使用者利用它們的基本架構，客戶可以藉此建立特有的應用服務；Google 也利用雲端技術提供應用程式與服務給使用者，並提供方便的開發介面，讓開發者能夠輕易發展出各類型的應用；而 Salesforce.com，則將應用服務建立在各服務平台提供的基礎架構與應用服務之上。諸如此類的資訊分工模式正逐漸成熟，雲端運算架構具有網格運算、公用運算與軟體即服務的優點，搭配服務導向架構的特性，讓企業可依照策略目標，在預算不足的情況下取得資訊服務，並可有彈性的調整使用量與投資成本，減少閒置資源的浪費，對於一般難以負擔大量投資成本的中小企業而言，這樣的系統架構可說是一項理想的解決方案。

然而當企業採用雲端運算架構，並將服務與資料轉移到『雲』中時，卻也引發了許多資安上的疑慮。首先，各家雲端平台間並不存在通用的標準，這造成使用者轉換上的困難，降低服務轉移的彈性，同時也不符合服務導向架構的精神；即便雲端運算架構的高穩定性，但企業仍舊擔心服務中斷的情況，導致日常營運被迫中斷，造成許多營運風險上的疑慮；將服務與資料轉移到第三方平台上，對於一些敏感性資料的保護與隱私議題，是雲端運算業者必須致力於讓客戶能夠充分採用的另外一項關鍵因素。

雲端運算之發展雖日益完備，然使用者對於資料儲存與運算安全性上的質疑，致使目前之使用尚未能普及，若能加強客戶端對於資料與流程的控管權力，可消除導入雲端運算架構的隱憂。於是本研究試圖結合雲端運算與服務導向架構，發展出以雲端運算為基礎，並符合服務導向精神的服務平台，藉由這個服務平台，能夠整合服務需求方與服務供應方，結合現有雲端運算架構的發展狀況，提出一項新的服務營運模式。

第三節 研究目的

在雲端運算基礎設施逐漸完善的情況下，如何將企業應用服務與雲端平台發展趨勢相結合，避免重複投資的浪費，並將資源投注於企業核心業務上，應是企業未來應該思考的重點。但因為採用雲端運算架構時，衍生出企業對於資安與營運風險方面的疑慮，讓雲端運算的推廣受到許多質疑，也降低了企業採用的意願，這也是目前雲端架構平台業者，所致力解決的關鍵問題。

根據以上的研究動機，本研究目的是以雲端運算基礎發展具服務導向架構特色的服務平台，該服務平台具有網格運算、公用運算與軟體即服務運算架構上的優點。該平台可結合外部網路服務、充分利用現有的資源、降低系統開發的時間與成本、搭配本研究所提出之資料分離運算控管機制，可消弭企業對於導入雲端運算架構的隱憂，並有助於雲端服務市集形成。本研究亦將以金融貸款融資之應用，說明本服務平台符合服務導向架構之特色與雲端運算之優勢。

本研究採用開放原始碼的軟體架構技術開發此一服務平台，並拓展 MVC 模型，將服務平台分為應用介面、中心控管、服務供應與基礎建設共四層架構，採用 BPEL(Business Process Execution Language)作為服務流程描述，平台將依照服務流程描述的內容，進行實際的服務組裝與執行工作；在服務流程執行的過程中，可以依照運算與儲存資源的利用情況，動態的將服務工作分配到閒置的資源中，以達到最佳的資料配置；為了消弭企業對與導入資安上的疑慮，搭配資料控管機制將可有效管控雲端資料的隱私與安全性；最後本平台具備有以 AAA 協定為基礎的管理機制，可以針對服務運作與使用情況作完整的紀錄，供服務市集運作需要之用。故研究目的包括：

- (一) 發展雲端運算服務導向架構運作機制。
- (二) 提出雲端運算服務導向架構平台的四層式架構。
- (三) 透過金融貸款應用範例，說明本服務平台滿足雲端運算與服務導向架構之要求，並具實作上之可行性。



第四節 研究架構

一、緒論

說明本篇論文的研究背景、動機、研究目的與論文架構。

二、文獻探討

說明研究相關的文獻資料。包括網路服務架構、運算服務架構、服務流程描述與雲端運算環境。

三、研究設計

說明本服務平台建置方式與相關機制、流程的說明與設計，並提出雲端運算服務導向架構之服務平台。

四、系統建置

說明如何建構雲端運算服務導向架構之服務平台，並說明雛形系統運作之機制與流程。

五、系統實作與展示

說明並展示以企業貸款融資為例之雲端運算服務導向架構服務平台，並說明雛形系統實作細節與畫面展示。

六、結論與未來展望

根據研究過程與結果，歸納整理出本研究的結論，並提出未來研究方向與建議，以作為後續研究的參考。

第二章 文獻探討

第一節 網路服務架構

一、服務導向架構

(一) . 服務導向架構定義

服務導向架構(Service-Oriented Architecture, SOA)是一種軟體架構，服務實體透過配置及建構的過程，達成有效資源最佳再利用及鬆散耦合最大化。服務導向架構定義服務是最基礎的單元，而整個系統之概念模型是透過服務節點之間的連接而成。昇陽公司(Sun Microsystem)定義服務導向架構為以最佳實務為基礎的整合性軟體架構設計方法，充分利用服務導向架構，可促使企業有效的使用並整合企業的資訊科技資產，並增加企業面對環境的改變和競爭策略使用上的彈性。

服務導向架構的服務元件將服務的介面和服務的實際執行分開。亦即每個服務的實際功能(What)和服務內部執行內涵(How)的過程分開。服務元件使用者不需要瞭解服務實際執行方式或其內部作業流程的狀況便可使用該服務；當使用者需要服務時，介面描述能夠讓使用者瞭解該服務之功能，使用者只要從介面輸入該服務所需要的資訊就能透過服務滿足使用者需求。在整體架構上，服務導向架構採取「尋找、連結、執行」(find-bind-execute)的架構模式。服務導向架構可分為六大個體，分別為服務提供者(Service Provider)、服務消費者(Service Consumer)、服務註冊(Service Registry)、服務合約(Service Contract)、服務代理者(Service Proxy)及服務租約(Service Lease)。其架構如圖 1 所示。這樣的架構方式使得 SOA 的模組元件之間的相關程度達到最小，同時每個模組元件能再利用的程度也達到最大，以下就服務導向架構的組成個體進行說明：(McGovern, Tyagi et al. 2003)

1. 服務消費者

服務消費者是一個應用程式、服務或是其他類型的軟體系統。這個軟體在初始時經由服務登記機制來找出、並接合所需要的服務，依照服務合約的內容送出適當的需求訊息來執行該服務。

2. 服務提供者

服務提供者可放置於網路上的某一可存取的位置，並能接受由消費端傳出的需求訊息執行服務，服務的提供者可能是一個大型主機、資料庫系統或軟體元

件。服務提供者在服務註冊上發佈該服務的服務合約，讓消費者透過服務合約執行及使用服務。

3. 服務註冊

服務註冊就像是建立一個目錄，此目錄中紀錄著可用的服務，並儲存服務所提供的合約，供有興趣的服務消費者尋找與接合服務。

4. 服務合約

服務合約即是服務提供者和服務消費之間互動方式的說明，它明確的表示了該服務使用時需求的格式及服務可產出的結果，合約也明確記載了使用該服務先後的條件，此外合約同時也可能提供了服務品質的評等等級。

5. 服務代理者

服務代理者是由服務提供者提供，用來支援消費者使用服務。消費者能在服務代理者上透過 API 來執行服務的需求。服務代理者能在服務註冊上找到合約及合約所參照的服務提供者，並且將消費者的需求透過格式化送給提供者以執行服務。服務代理者可以透過快取的能力來強化使用服務的效能，當服務代理者記錄了遠端服務的快取之後，服務消費者可以不需要再透過服務註冊來取得合約，並且在快取中儲存的合約可以使消費者能透過合約直接接合和執行服務，這樣一來不但減少了網路的使用，也提升了服務執行的效率。服務代理者也能在本地端提供一些服務功能，如果該服務不需要服務資料或只需少量的服務資料，服務代理者可以在本地執行服務。服務的提供者也必須提供在不同環境下的服務代理者。雖然服務代理者並非是必要的實體，但是服務代理者對於消費者使用上的方面及 SOA 整體效能提升都有很大的助益。

6. 服務租約

服務租約是服務註冊授與服務消費者使用合約的期限。當消費者要求了一個合約之後，合約的租約便開始計算，直到租約到期，消費者必須重新要求合約。維持了服務的提供者和消費者之間接合的狀態，租約的使用是必須的。它定義了狀態維持的時間，使得提供者和消費者的接合被限制在一定的時間下。在租約的使用下，降低了服務提供者和服務消費者之間耦合的程度。

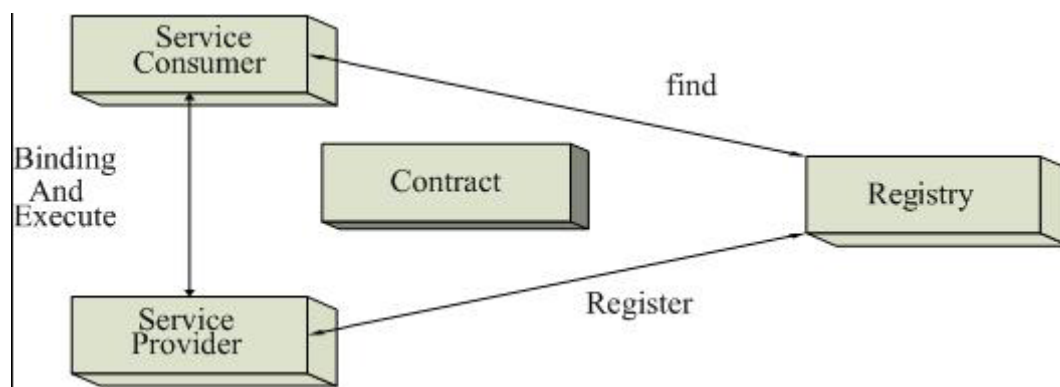


圖 1：典型服務導向架構圖

資料來源：(McGovern, Tyagi et al. 2003)

(二) 服務導向架構技術特性

服務導向架構是一種軟體架構，強調低耦合性(Loosely-Coupled)、可程式化介面開發與服務供需雙方間的介面呼叫(Interface Calls)。換句話說，不同的元件間將是以獨立作業的模式進行，在組成整體服務的過程中，無需知道其他元件的開發方式。服務導向架構是設計來充分利用現有的資源，滿足快速進入市場的需求，並減少系統開發與維護的成本與風險(Pallos 2001; Conlon, G.Hale et al. 2008)。

根據學者的分析指出，服務導向架構的優點包含減低系統整合的成本、改善企業回應的敏捷度與彈性、增進資產活用程度，更重要的是增進投資報酬率。從另外一個角度來看，改用服務導向架構架構讓合作伙伴間的系統整合變的更為容易，例如供應商與客戶間的供應鍊系統整合、將 IT 服務進行外包等應用，服務導向架構同時幫助企業實現公用運算(Utility Computing)的概念(Feuerlicht 2006)。關於服務導向架構的技術特性，請參考表 1。

表 1：服務導向架構技術特性

分散式架構	企業所需資源可能分散於不同領域，需透過服務間良善的溝通機制來連結資源。
關係鬆散的界面	在鬆散的環境下，開發人員可以輕易重複利用現有的服務元件。
開放的標準	開放標準是服務導向架構的核心特色，可降低異質平台整合上的困難。
定義明確的服務合約	服務合約決定每個服務的功能及呼叫使用的方法，提供如何使用服務的導引。
服務可發掘且動態取得	動態載入所需服務，而不需使用者介入，將可充分展現服務導向架構的平台整合特性。

資料來源：(Huang 2006)

隨著相關技術與基礎設施的成熟，服務導向架構概念也越來越被廣泛採用，成為替代舊有系統開發模式的新解決方案；由於服務導向架構可以滿足未來的應用程式許多嚴苛的需求，如：滿足大量互動的特性(Interaction Style)、快速開發、保持對需求快速改變的服務提供能力、服務品質、彈性與成本效益，這些特性都可以藉由服務導向架構開發模式來加以達成。因此，服務導向架構漸漸成為普遍的系統開發模式(Brereton, Budgen et al. 1999; McCoy and Natis 2003)。

二、網路服務

根據 W3C 的定義，網路服務(Web service)是一個軟體系統，用以支持網路間不同機器的互動操作。網路服務具有機器可處理的格式(WSDL)，其它異質系統藉由同樣規格的訊息與網路服務溝通(SOAP)，通常來說這些格式、訊息都是以 XML(Extensible Markup Language)等開放標準的格式所發佈。而 IBM 認為網路服務是一種介面，描述一個或是一串經由標準 XML 訊息存取的網路操作。此介面是使用標準、正式的 XML 格式來描述，稱為服務描述，提供接觸服務時所需的所有細節，包含訊息格式、傳輸通訊協定和位置。

W3C 定義網路服務由 SOAP、WSDL 與 UDDI 三種協定組成，以下分別介

紹各協定的定義與用途：

1. SOAP(Simple Object Access Protocol)

SOAP 是一種架構簡單的輕量級(Light Weight)資料傳輸協定，用於分散式網路環境下做資料訊息交換。SOAP 定義一個 XML 文件格式，該文件描述呼叫一段遠端程式碼的方法(Method)。應用程式可以藉由建立一個呼叫遠端服務的 XML 文件，並包含所有必要參數，透過網路將此文件傳遞給遠端服務，該服務接收到這份 XML 文件後，將呼叫文件中指定的方法，並回傳 XML 文件格式的執行結果。

2. WSDL(Web Services Description Language)

WSDL 即為網路服務的描述語言，是一個描述網路服務的 XML 詞彙表。在標準的描述格式下，WSDL 同時描述網路服務需要參數與執行內涵，可供使用者端呼叫網路服務時的參考，利用 WSDL 使用者端可呼叫其以前從未用過的網路服務，不需要事先指定或是做繫結的動作。

3. UDDI (Universal Description, Discovery, and Integration)

UDDI 為一服務註冊架構，服務提供者可於網路服務註冊中心定義 SOAP 介面，就如同一專門用於尋找網路服務的黃頁(Yellow pages)，當網路服務部署後，UDDI 規範定義如何將服務描述加入至註冊中心，UDDI 也可利用搜尋服務描述的方式，讓使用者找到適合的網路服務。(W3C 2008)

因為網路服務跨平台、分散性與有彈性等優點，許多學者將網路服務視為非常適合發展服務導向架構的解決方案(Group 2004; Conlon, G.Hale et al. 2008)。2001 年 3 月，由 IBM 與 Microsoft 發表的提議，將 Web Services 的技術分成三個堆疊，個別實作 SOA 中的「連結」、「刊登」及「搜尋」包括了：連結堆疊(the wire stack)、描述堆疊(the description stack)、探索堆疊(the discovery stack)，這代表利用網路服務進行服務導向架構開發在技術上的可行性。

第二節 運算服務架構

一、網格運算

網格運算於 1995 年間提出，其概念源自能源網格的概念，衍生發展成分享計算與儲存的資源，且網格間的介面並不需是標準化的，換言之，資源間將可任意相互流用。以網路服務為中心的網格運算(Grid Computing)，被視為是未來科技發展的潮流；網格運算可以分割工作由多個運算資源來完成，以達到良好的價格效能比(Cost Performance Ratio)。網格運算所使用的運算資源可能處在內部或是外部，並具備以下的優點：彈性資源配置與管理、安全性、提供不中斷的服務品質與增進資源的利用(Boden 2004; Zhang and Jeckle 2004)。

網格運算結合網路服務的發展趨勢，可分為兩個方向，第一是以資訊科技層級的角度出發，所有的服務與網格資源皆由網路服務的開放標準來定義；第二是更高層次的資源共享，如商業服務或商業流程等。配合服務導向架構與網格運算趨勢的發展的新架構思維，改變發佈資料與連接應用服務的方式。服務需求者與提供者間的關係，將透過一個新的基礎架構來連接，而各利益攸關者(stakeholder)專注於本身的核心業務，換言之，利他就是利己會是未來軟體服務產業的重要思維(Zhang and Jeckle 2004)。

網格運算的持續發展，逐漸形成全球網格網絡，這個網絡中可能包含資料庫、儲存空間與運算資源，這些強大的基礎架構，開始應用在許多領域，如科學實驗、生物運算等需要強大的運算資源的應用領域，這些跨國界的網格串連也逐漸衍生出一些新的機會與問題，例如使用身份的認證、利用資源的統計與衍生出許多資源中繼的仲介商，這些仲介商扮演中介軟體(Middleware)的角色，串連各地的網格資源，以提供服務開發者能更輕易的利用強大的基礎建設，能創造出更大的應用效益。但網格運算也有他的限制，例如缺乏明確的管理機置與互通的標準等問題，這是未來網格運算推廣上的最大限制。(Angelis, Gritzalis et al. 2004; Moorsel. 2005)

網格運算雖可視為平行運算的一種，但網格運算與超級運算(SuperComputing)的差別在於，網格運算是透過公開標準的網路協定進行連接，例如乙太網路與網際網路，而超級運算是投資集中式、多核心的大型電腦，這是最顯著的不同。利用分散式的運算資源來達成大型電腦的功能，其最大的好處在於可免去龐大的硬體投資，而改用較為便宜的微型電腦來組合，對於企業來說，可以降低大型電腦投資的初期成本，也有效降低運算資源閒置與未來性擴充的問題。在企業普遍希望降低 IT 投資成本的情況下，網格運算的概念會是未來投資運算基礎架構的解決方案。

二、公用運算

公用運算代表 IT 基礎建設在技術與財務兩方面的重新定義，隨著服務導向架構的發展完備，公用運算以服務的角度來重新定義所有的 IT 應用，這些服務可以透過如網路服務的 WSDL 與本體論的 OWL-S 等以 XML 為基礎的描述語言來進行服務描述，這些服務可以在今日的異質網路環境中被呼叫、使用。一旦 IT 應用都以服務的形式進行連接，這讓服務計價(Pricing)變的可能，就像水力、電力與天然氣等公用設施一樣，IT 服務也可以依照你使用的時間、數量來計費，並且如同上述的基礎建設一樣隨手可得。(Denne 2007)

公用運算概念的提出，主要是以企業的 IT 投資角度出發，因為如果 IT 資源能夠如同水、電般能夠輕易、依照需求取得，而非需要大量的初期投資成本，讓企業能夠降低成本投資風險，IBM 即提出以公用運算為基礎的系統開發架構，這個架構並不需要龐大的軟、硬體投資，相反的，他盡量利用共用的(shared)基礎設施來完成資訊應用的開發工作，如此可以有效的降低系統開發時間與成本，並且能夠使 IT 資源的利用更為經濟。(Eilam, Appleby et al. 2004)

公用運算所指的利用共用的基礎架構，跟傳統的系統外包模式有點不同，因為單純的資訊應用系統外包，是指將整個應用都由外部資服業者完成，然後在企業內部導入上線，跟企業本身原本擁有的資訊架構關連較低，但公用運算談的概念是，希望能夠善用企業內部已有的資訊架構，並在善用這個架構的前提下，盡量發揮 IT 投資的最大效益，以這個角度出發，企業本身在導入公用運算時，必須調整企業的 IT 投資策略，讓內部的資訊系統架構朝向符合公用運算架構的要求，其實這個概念跟服務導向架構所談的概念極為類似，以因為如此，許多學者也將服務導向架構視為推動公用運算發展的推手。(Ross and Westerman. 2004; Denne 2007)

當『用多少，算多少』(pay-as-you-go)的概念普遍為企業所接受後，勢必會改變原先資訊服務的產業結構，傳統需要龐大軟、硬體初期投資的模式將不再為企業所接受，而必須轉變為低成本、有彈性的資訊服務，資訊服務提供業者的營運方向也該隨之修正。但公用運算與網格運算一樣，都有其實現上的質疑與困難，例如：資訊安全、計價模式與服務持續能力等，這些都有待更多的研究投入，讓這個發展模式得以更為順暢的推廣。

三、軟體即服務

隨著全球化的腳步及越來越競爭的商業環境，企業無不尋找如何降低 IT 成本，同時能提出更具競爭力的成品和服務，這迫使企業必須更專注於本身的核心競爭力，並提昇對外包夥伴及企業夥伴的依賴關係，由於 IT 服務提供商具有規模經濟，他們比一般的企業能提供更好、更便宜且更可靠的應用軟體。

軟體即服務(SaaS)，即是一種透過網路來進行存取的軟體代管服務，其定義並非特定的軟體架構，也不存在固定的商業模式，更沒有提到特定的技術或協定，在這個定義下只說明了軟體程式碼的部署及存取及存放位置。利用軟體即服務推動的應用服務，使用者不需要知道應用程式運作的內涵，例如網路上的Webmail服務，後端可能有收發郵件的伺服器，但使用者僅透過瀏覽器介面與其溝通，而忽略後端的應用程式運作機制與資料儲存問題。換言之，一個資服業者擁有應用服務與資料，並透過公開的使用介面讓使用者存取這些資料，這就符合了軟體即服務的基本精神。軟體即服務分為二種類型，一種主要提供給各型的企業，它通常軟體規模較大，且必須客製化，使得一些商業處理更為便捷，如：財務系統、SCM、CRM等，它的收益來源是以客戶繳交的訂閱費為主。另一種則是提供給一般大眾，以會員繳交訂閱費作為收益來源，亦有免費提供給客戶使用，而以廣告收入為收益來源。

根據以上對軟體即服務的定義，並沒有很明確的說明如何創建一個成功的軟體即服務架構，因為定義中不存在明確的技術標準或是規範。例如：將企業內部的資訊系統新增一個Web的存取介面，這看來符合廣義的軟體即服務定義，這並沒有真正的受益於軟體即服務的架構優勢，因此我們不應該單純以資訊系統的軟體架構來看待軟體即服務，而是應該拉高到策略層級這個概念來看待。從一個軟體架構的角度來看，可以用三個主要特性來檢視一個軟體架構是否落實軟體即服務的概念，一個符合軟體即服務概念的軟體架構必須具有可擴充性、多會員效益、可自組性。可擴充性主要是在強化多工機制，更有效率的使用系統資源，讓系統資源不易被某些處理長期佔用。

多會員效益是與其他類似概念的最主要差異點，應用服務不需要針對單一公司做開發工作，而是可能利用這個架構來滿足更多不同使用者的需求。因此軟體架構就必須儘可能的讓使用者間能充份的利用系統資源，且不影響使用者的資料各自獨立。我們可以在一台伺服器中安裝一套軟體系統，透過網路的存取來服務多家不同的公司，為了在這個架構下能夠達成客製化的目標，利用軟體元件的自組性能讓各個使用者依本身業務的需求，透過簡單的參數設定，讓應用軟體的呈現或執行能符合各種不同的需求。(Chong 2006; Huang 2006)

由軟體即服務的定義與三個特性來分析目前的軟體架構，可以發現一個軟體架構可能滿足部分特性就能符合業務需求，我們可以用成熟度指標來看待軟體即服務，提出此軟體架構的廠商可依本身的業務需求，選擇性的滿足其中的幾項特性，而軟體即服務成熟度可分為四個等級：(Chong 2006; Huang 2006)

等級一：客製化

此等級和傳統ASP軟體模式相同，每一客戶擁有獨特客製化的軟體供業務使用，每個使用者在ASP端的伺服器使用各自的軟體，而不相互干擾。傳統的Client/Server架構的軟體最適合轉移到這個等級的軟體即服務架構，因為它只需

要少量的開發工作，且不需要重新從無到有來重新架構。在這個階段 ASP 廠商只是從統一的伺服器及方便的管理來節省成本，所以並無法突顯出軟體即服務概念的優勢。

等級二：自組性

在等級一中，各個會員使用各自的專屬軟體與專屬的程式碼，在等級二中則是使用統一的程式碼，再以詳細的選項讓會員設定，來改變軟體的使用介面及使用邏輯。以單一的程式碼為基礎，可以降低軟體即服務軟體服務的需求，因為只要修改一組程式碼就能讓所有的會員都全部更新。若有一個軟體架構已經針對某個顧客進程式客製化，要進行轉移到軟體即服務的等級二自然要付出較多的轉移成本。在這個階段，廠商可能要提供更充足的硬體及儲存空間，以應付未來更多執行個體(instance)的執行。

等級三：自組性及多會員效益

在這個等級中，廠商使用單一的程式執行個體來服務所有的顧客，運用詳細的選項來讓使用者自行設定符合本身業務需求的程式功能，透過授權機制及安全政策來決定各別使用者可以存取之資料，從使用者的使用上來看，並無法感覺出是在與其他的使用者共用這個程式執行個體。在這個等級中比等級二更重視系統資源的使用，廠商必須依顧客的多少來評估這個程式執行個體所會使用到的空間，例如網路連線的頻寬，伺服器的記憶體大小等。這個等級的缺點便是擴充性不足，它並沒有運用分割資料庫的方式來管理資料庫的效能，所有的使用者共用一個資料庫，當使用者越來越多時，只能將此資料庫轉移到功能更強大的伺服器才能解決問題。

等級四：可擴充性、自組性及多會員效益

在這個等級中，廠商運用服務協調平台來分配會員的程式執行個體，每個會員的資料各自獨立，並運用不同的選項來讓不同的會員設定所需的程式功能。在這個等級的軟體即服務架構可依服務規模任意擴充，因為它背後的伺服器或程式執行個體可依需求任意的增減，而不需要更動到整個軟體架構。

四、雲端運算

(一).雲端運算定義與特性

雲端運算可以從系統平台與應用服務兩個角度來解釋，從系統面來看，雲端運算系統可以依需求動態新增(Provision)、安裝、設定與移除伺服器的一種平台。雲端運算架構也能包含伺服器提供環境所需的其它計算資源，如：網路儲存群(SANs)、防火牆與其它網路設備等。從應用服務的角度來解釋，雲端應用服務代表利用大量的資料中心(Data Center)與強大的伺服器群來提供服務，並可直接透過網際網路連接，任何使用者只要有適當的連線能力與瀏覽器即可取得服務(Boss, Malladi et al. 2007)。

隨著大眾對於雲端運算的關注，相關的討論與產品如雨後春筍般的冒出，這些造成大家對於雲端運算這個名詞上的混淆，Gartner 研究機構在 2008 年 9 月提出了看法，其給雲端運算架構最初的定義為『雲端運算是一種提供給多使用者，利用網際網路科技提供『即服務』(as a service)，並可大量擴充的一種運算形式』，但其後發覺這並不足以完全涵蓋雲端運算的全貌，於是其後這份聲明將雲端運算分為兩類，第一類是以軟體即服務概念為出發的一種概念，這類別的雲端運算較為注重於如何將服務應用在雲端中，而非雲端本身的運算能力，例如思考如何將現有的 CRM 系統轉換到雲端運算架構上，並讓其能享受到雲端的好處；第二類是從虛擬化與自動化等技術角度出發，這個角度強調雲端運算技術改良所產生的嶄新應用與其服務能力，這也可視為是傳統資料中心(Data Center)或是企業外包第三方服務業者這種概念的延伸。(Gartner 2008)

從網格運算、公用運算到軟體即服務，再到今天所談的雲端運算，可以看到雲端運算並非橫空出世，而是一連串運算架構循序漸進發展的過程，也因為這個名詞所涵蓋的範圍太過廣泛，很難有統一的定義涵蓋所有的應用，讓雲端運算這個概念一直都有霧裡看花的感覺，但回歸到運算架構的發展歷程與其他學者、研究單位對雲端運算所下的定義，可以大略的為雲端運算歸納出幾個要素：

1. 平行、分散運算
2. 運算架構可彈性擴充
3. 低廉硬體投資
4. 以量計費
5. 輕量用戶端(Light Client)
6. 運算、儲存能力強大

7. 軟體、硬體即服務
8. 高度依賴穩定的網際網路
9. 儲存、服務外包

雲端運算架構可以讓企業投資在軟硬體上的成本更有效益，打破單一系統的最大限制，將運算群視為一個整體，具備自動系統管理、負載平衡與虛擬化技術等特點的一個整體，並可在幾分鐘內完成運算資源的維護，降低系統的複雜度，讓創新的應用更容易實現。無論是 Java、Linux、LAMP 等以堆疊(Stack)方式的應用服務，或是由 Google 所提出的 MapReduce 程式架構與 Google File System 都可以利用雲端架構來加以實現，換句話說，將現有的應用服務轉由雲端技術來實現是可行的。(Boss, Malladi et al. 2007)

雲端運算環境支援網格運算的特性，如快速提供運算資源供網格應用程式運行等。但網格運算著重需要大量運算的工作，由許多小運算單元共同進行，這通常需要數以千計的電腦支援才可能達成。而雲端運算同時也支援非網格運算的環境，如傳統的 Web 架構應用服務，雲端運算架構不僅僅是聚集大量的運算資源，同時也提供了管理的相關機制，包括：部署、重新部署、負載平衡、移除與監控等。(Boss, Malladi et al. 2007)

(二).雲端運算引發的問題

雲端運算的流行，引發許多學者對於其安全性的質疑，這些批評都可能成為推廣雲端運算架構上的阻力，以下整理目前各學者對於雲端運算在資安上的質疑：(Heiser and Nicolett 2008; Mansfield-Devine 2008; Stallman 2008)

1. 設計瑕疵(Design Flaw)

因為雲端服務都是外包給第三方業者進行服務，而且雲端運算強調無須知道資料、應用程式放置確切位置的特性，讓人擔心如果服務提供者所設計的服務有瑕疵，使用者將無從得知與預防，不像在本機開發應用程式，可以隨意的測試與修改，另外，像是 Google 與 Amazon 提供開放的 API 讓第三方業者能夠進行服務開發，也讓人擔心開發出來的程式安全性問題。有學者認為可以透過簽訂 SLA(Service Level Agreement)的方式來改善這個問題。

2. 網路資安問題

當使用者利用網際網路存取雲端服務，傳統網路釣魚、DoS(Denial of Service)攻擊與跨站腳本攻擊(XSS)等常見的網路資安問題，都可能造成使用者遭受攻

擊，並因此受到損失。尤其雲端服務常以網頁服務的形式出現，更容易成為駭客攻擊的目標，相較以往封閉式的環境而言，需承擔較高的風險。

3. 認證與授權

只要擁有登入帳號密碼，就能取得服務，換言之，如果這份帳號密碼遭受冒用，所造成的影響就如同本機電腦被入侵一般，將可以取得完整服務的權限，而且和過去相比，遠端服務的入侵難度相較封閉的本機系統要容易的多。

4. 服務持續性(long-term viability)

在服務都外包到雲端運算架構後，若發生服務中斷的情況，如網路斷線、無法提供服務等問題時，其損失與責任歸屬將會難以釐清，於是服務的可靠度是導入雲端架構時必須考量的重點之一，但與本地端服務相比，使用者所能掌控的程度較低，這也是造成導入雲端運算上的一個障礙。

5. 喪失控制權

對於自由軟體陣營而言，不使用專屬、封閉的軟體，而改用自由軟體的一大理由是希望保有對使用軟體的控制權力，而雲端運算要求將服務外包給服務提供者，這會有控制權喪失的疑慮，批評者認為當控制權移交到第三方業者的手中，使用者唯有受制於人的份。

6. 資料控管

在資料都外包給第三方業者的情況下，該如何確保資料安全的問題？在SaaS雲端服務中，往往有許多使用者一同共用服務資源，資料可能在錯誤的權限設定、分享或是服務程式產生錯誤的情況下外流，增加資料外洩的風險。

另外資料代管公司對於資料的處理也是令人擔心的一環，『特權使用者』(privileged user)的控管與資料管理規章問題，資料的存放位置、資料隔離與恢復與受調查時資料的曝光程度等，都讓使用者對導入雲端運算望而卻步，這也是雲端運算架構最為人所質疑的地方，在未來雲端運算架構的推廣上，這是必須正視且提出理想解決方案的重點。

(三).雲端運算現在與未來

根據 Gartner 研究機構的研究，指出雲端運算代表商業模式的革新，影響到電子商務的發展，而各種對於『雲端運算』這個名詞的矛盾與混淆，代表它改變 IT 市場現狀的潛力。Gartner 將雲端運算定義為一種運算的模式，包含可大量擴充 IT 提供相關的技術能力，利用網際網路的相關技術，包裝成一種服務來提供 (as a service) 給多個外在使用者 (Stamford 2008)。當雲端運算架構環境持續發展，運算服務變的無所不在，服務交易平台可以讓彼此交換運算資源，同時負責服務供需雙方的搓合與連接，並提供服務發佈、發掘、服務品質控制、計費機制與資訊安全保護等機制，而形成一『全球雲端服務交易市集』(Buyya, Yeo et al. 2008)。

綜觀目前各雲端業者的發展，可以分為兩類，第一是針對雲端運算服務平台解決方案，第二是利用雲端科技推出各類服務，以下介紹目前幾個致力於推廣的雲端業者：

1. 雲端運算服務平台

Microsoft 在 2008 年推出的 Azure 雲端平台，此平台是由微軟資料中心所承載的一個網際網路規模層級的雲端服務平台，包含作業系統和一套可個別或整合使用的開發工具。Azure 的彈性 (Dynamic) 及相互操作性 (Inter-operable) 平台可用來建置在雲端執行的應用程式或是運用雲端強大的運算能力來加強現有的應用程式。其運作方式如下所示：(Microsoft Corporation 2008)

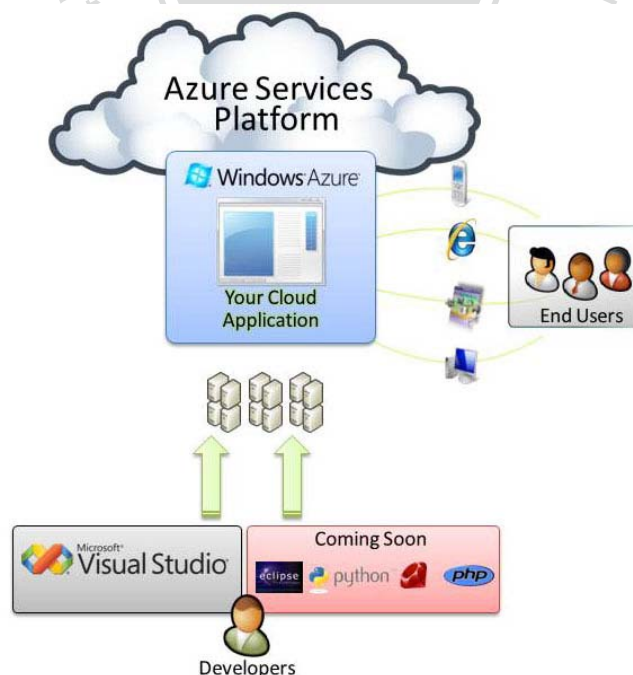


圖 2：Azure 平台架構圖

資料來源：(Microsoft Corporation 2008)

昇陽電腦(Sun Microsystems)推出開放式雲端運算架構-昇陽開放雲端運算平台(Sun Open Cloud Platform)。昇陽透過本身強大的軟體技術，如 Java、MySQL、OpenSolaris 和 Open Storage 等業界領先的軟體技術，投入目前逐漸成形的雲端運算市場，並瞄準其巨大的商機。

知名網路書城亞馬遜(Amazon.com)於 2002 年推出 AWS(Amazon Web Services)平台服務，這個平台主要是提供基於亞馬遜強大的基礎運算能力，所推出的網路服務，其他的網站或是使用者端的應用程式，可以透過方便、開放的開發介面，由軟體開發者來進行客製化開發後，於 AWS 平台上發佈，供眾多平台使用者來使用。(Amazon.com 2007)

Google 的『Google App Engine』，讓開發人員利用 Google 的基礎環境來運作其網頁應用程式；藉由 Google 強大的運算能力，忽略儲存、運算等基礎建設，專注於應用程式本身的開發。該應用程式將可利用 Google 的網頁空間代管(Hosting)、資料庫(Bigtable)、自動負載平衡、完整支援 Google APIs、支援本地端開發等特性，讓開發人員開發出的網頁應用程式，具有可供百萬人次使用的應用能力(Google.com 2007)。

2. 雲端科技服務

企業雲端運算公司 salesforce.com，是 SaaS 的領導廠商，主要業務是提供線上 CRM 服務，使用者只要透過瀏覽器，就可以使用過去必須大量投資、部屬的 CRM 系統，更可以依照使用量、功能來計費，讓這個模式廣受中、小型企業所接受。salesforce.com 是目前談到雲端服務中，所被認為的典範案例之一。以下是 salesforce.com 的運作架構圖：



圖 3：salesforce.com 平台架構圖

資料來源：Salesforce.com

趨勢科技(Trend)是另外一家積極投入雲端科技的防毒軟體公司，趨勢科技認為隨著網際網路的高速發展，病毒變種的時間越來越短，傳統採用病毒定義檔的儲存的空間也愈來愈大，在這個情況下，透過遠端的伺服器群(Server Farm)可以妥善的解決這個問題。目前趨勢科技已在美國東西岸各架設了伺服器數量在2000 部以上的資料中心，且計劃在日本及台灣再增設兩座，互為備援。趨勢科技身為第一家推出線上即時更新病毒碼的業者，採用雲端技術也可說是為推出SaaS 的新服務模式開啟序幕。

最後，綜觀網格運算、公用運算、軟體即服務到雲端運算，我們可以看出運算架構的發展沿革，歸納而言，雲端運算可以說集前三者之大成，包含網格運算的分散式運算能力，以達成效能與彈性資源配置與管理、安全性、提供不中斷的服務品質與增進資源的利用等優點；公用運算的概念讓企業重新調整 IT 的投資策略與基礎架構的配置，並且讓依照使用量來計價的服務模式變的可行；軟體即服務模式在某種程度上來說可以說是公用運算的具體實現，透過這種服務架構，可以達成擴充性、服務組合及多會員效益等優點。隨著相關資訊科技與基礎設施，如：網路服務與光纖網路普及的推動下，雲端運算的應用將會越來越普及，如同以往的發展經驗一樣，企業的資訊服務應用思維必須因應環境的變革而有所改變。



第三章 研究設計

第一節 MVC 架構

MVC(Model-View-Controller)用於表示一種軟體架構樣式。它把軟體系統分為三個基本部分：模型(Model)、檢視(View)和控制器(Controller)。模型交由資料庫專家進行資料管理與資料庫設計的工作；檢視負責提供圖形介面，是負責與使用者互動的窗口，一般而言由美工人員負責；控制器負責提供系統的應用功能，而這些應用功能將由程式開發人員所提供。MVC 的目的是實現一種動態的系統開發，使對系統的後續修改與擴展簡化，並盡量重複使用已開發的系統元件。透過這個開發樣式可以將系統複雜度簡化，使系統架構更加直覺，專案中各專長的開發人員也可以作適度的分工，讓系統開發的工作能夠更順利與有效率(Reenskaug 2003)。

根據 IBM 評估內部入口網站的評估報告指出，使用 MVC 架構存在許多優點，如提供網頁程式的基本框架，讓開發人員可以輕易的加入現有的應用資源；使用保有彈性的開發過程，減輕系統開發時的負擔；藉由控制層連結介面層與模型層的三層架構，讓網頁程式開發人員可以輕易的分工，將服務邏輯交由其它服務提供者負責，轉而專注於如介面與資料的處理等，可見 MVC 設計樣式的可用性受到相當肯定，並足以負擔商用營運環境的嚴苛挑戰(Yan, Leip et al. 2005)。

以網路服務為基礎的開發模式，相當適合採用 MVC 架構進行實作，由控制器負責流程控制的工作，組合不同的網路服務與模型所定義的資料格式，完成使用者的服務要求，最後再透過檢視將服務成果呈現給使用者。以設計樣式(Design Pattern)而言，MVC 架構的拓展可視為服務導向架構的理想實作模式，因其強調的動態系統開發模式，讓系統的修改與拓展盡量的簡化，並能將系統適度分工與強調元件重複利用等特性，都與服務導向架構所強調的低耦合性、專注介面開發、充分利用現有資源、減少系統開發時間、低成本、敏捷回應等優點相近。拓展後的 MVC 架構，也可充分利用雲端運算的特性，讓企業投資獲得更大效益、充分利用軟硬體資源、提供強大的服務能力與高品質服務水準(Reenskaug 2003; Davis, Demetriou et al. 2006)。

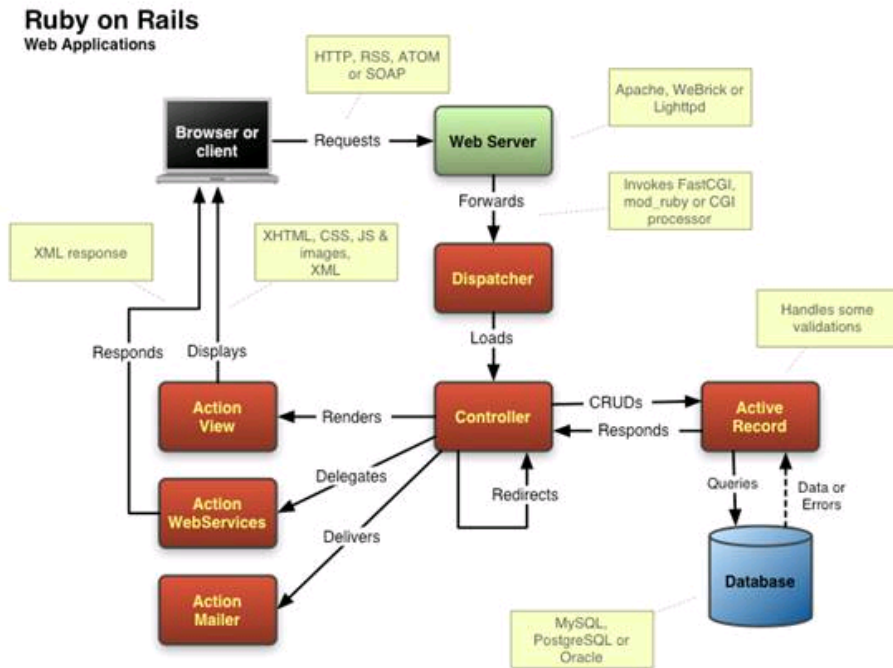


圖 4：MVC 架構圖
資料來源：rubyonrails.org

第二節 開發工具

MVC 架構將系統架構分為模型、檢視與控制器三個部分，本研究架構與傳統的 MVC 架構不同，將控制器定義為中心控管層，依照服務邏輯進行服務分配的功能；檢視定義為應用介面層，以 Google Sites 搭配 Gadget 的方式進行人機介面設計；而模型定義為服務供應層，提供資料中心的存取與應用服務的提供。

隨著開放原始碼軟體這幾年受到廣泛的接受，在企業環境中應用開發原始碼軟體進行開發的情況也日益普遍，組合商用軟體與開放社群軟體並用的開發模式，成為企業在選擇開發工具時的普遍考量；隨著越來越多的社群與開發資源的投入，開放原始碼的應用較以往更為成熟，也更能負荷商用營運環境的嚴苛挑戰，其中 Apache 專案的成功，囊括了大半網頁伺服器的市場，堪稱為最受歡迎的社群產品之一；而開放原始碼的開放特質，也讓開發人員能有更多的揮灑空間，而不至於受到個別平台的限制(Yan, Leip et al. 2005; Erdogmus 2008)。

動態語言(Dynamic Language)是程式執行期(Run-Time)保有彈性的一種程式語言，不同於在編譯期就決定程式執行方式的嚴謹作法，動態語言支援行為控制、執行期繫結與鼓勵快速開發並且回應需求等特色(Erdogmus 2008)。PHP 是一種內建多樣化函數的動態語言，這些開放程式碼的函數提供了各式功能，可供開發人員快速開發網頁程式，搭配 Apache 網頁伺服器，可以快速生成使用者瀏覽的網頁。PHP 支援在多種平台上執行，完全免費且開放原始碼，讓開發人員能

夠有更大的揮灑空間。

有鑑於此，本架構選擇以開放原始碼環境為主要的開發平台，開發環境選用 Linux，搭配 PHP 腳本語言(Scripting Language)與 Apache 網頁伺服器，運用網路服務技術來實作與服務提供商與資料中心的通訊機制，資料交換的格式採用 XML。皆以開放原始碼或標準為基礎進行開發。

第三節 網路服務流程描述語言

工作流程(Workflow)藉由固定的流程描述格式，讓工作流程管理系統(WFMS)得以達成商業流程自動化。服務導向架構的發展，衍生出跨系統與跨組織的分散式、異質且動態系統環境的服務需求，網路服務技術常用來發展服務導向架構系統，網路服務透過 WSDL 的描述，讓服務需求者得以透過這個以 XML 為基礎的服務描述與其互動。當一個企業中的各式應用程式都包裝成網路服務的形式，並分散在各處、分屬不同服務提供者，透過相關服務組合、串接的技術，這些服務將形成企業所需的工作流程。

BPEL (Business Process Execution Language)就是設計來描述一串服務組合的語言，這個商業流程描述語言注重可攜性(Portability)與可互通性(Interoperability)，可攜性代表一串定義出來的商業流程，可以同時被不同的企業所利用，也就是代表這些企業對於流程的描述方式有一個互通的標準，有人將 BPEL 稱之為『商業流程的 PDF 檔』；而可互通性則代表不同的服務執行引擎，都可以執行 BPEL 流程描述，因為 BPEL 是構築於網路服務的基礎之上，於是服務間的互通性可以被確保。除此之外，BPEL 還符合以下幾點以服務導向出發的優點，讓 BPEL 變成一項非常適合實作服務導向架構的流程描述語言。(Khalaf, Keller et al. 2006)

(一) 彈性整合(Flexible Integration)

彈性整合表示實際服務繫結可以在服務流程設計、部屬或執行這三個階段中發生，但服務實作細節交由 BPEL 執行來決定，而不在流程描述決定，BPEL 提供的『portTypes』描述，讓於網路服務溝通能皆透過同樣的網路服務介面達成。

(二) 支援即時多方狀態溝通 (Support for Simultaneous Stateful Conversations with Multiple Partners)

在 BPEL 的服務描述中，可能會在同一時間使用多方服務來源，為了讓服務來源間能夠同時執行，BPEL 提供『partnerLink』描述來提供相互溝通的機制。

(三) 生命週期管理(Life-Cycle Management)

當 BPEL 服務流程開始執行後，流程的創建與終止將由流程模型來決定，而非服務呼叫者，BPEL 的實體辨認機制 Correlation，讓 BPEL 執行引擎能夠透過『partnerLink』將訊息導向到正確的服務實體，以管控整個流程執行的生命週期。

(四) 可恢復性(Recoverability)

在一個較長的執行流程中，可恢復性是相當重要的，BPEL 支援流程執行回復(Rollback)的功能，以達成重新執行未完成工作與錯誤處理、更正機制。

表 2：BPEL 語言結構

Process	partnerLinks	定義服務流程與外在世界的關係，例如說：流程所有人、流程關係人與流程在整個服務中所扮演的角色
	Variables	宣告服務流程會用到的資料結構
	faultHandlers	宣告當服務流程發生錯誤時的處理方式
	Sequence	描述服務流程邏輯，如：次序、平行流程、條件、迴圈、收發參數等

資料來源：(Louridas 2008)

第四節 雲端運算環境

一、Google 應用環境

Google 推出了各式各樣憑藉於雲端運算架構下的網路服務，Google Apps 包含：Gmail、GTalk、Google Docs 與 Google Sites 等服務，其所適用的領域包含溝通、協同合作、企業服務解決方案等。Google 同時提供了方便的 API(Application programming interface)供第三方開發者實作，各網路服務提供者能夠輕鬆的憑藉在 Google 強大的運算、儲存資源上，減低系統開發上的難度，如單一使用者登入(Single Sign On, SSO)機制與 Email Migration API 等功能，使得 Google Apps 環境能夠與現有的企業內部服務整合，提升產品開發效率並推出市場，提供使用者更滿意的服務體驗，同時也讓企業本身保有更強的競爭能力。

Google 強調通訊的重要，「Google Apps」可提供各種服務來協助您更容易創建一個社群，輕鬆地為所有使用者部署應用程式，省下更多時間和成本以專注在核心目標上。透過 Google 的代管服務，使用者無須安裝或購買任何軟硬體，可以將導入的成本減到最低。這些服務包括 Gmail、Google 日曆、Google Talk、Google Docs、Google Sites 以及網頁建置服務等等，Google 將這些服務整合在一個網域內，讓以往開發成本很高的使用者管理、協同與溝通機制等，都由 Google 的基礎設施負責提供，讓社群網站架設的成本大幅降低，也促使更多第三方開發人員加入這個平台，共創更大的平台價值(Google.com 2007)。

Google Sites 是讓使用者快速、即時傳遞訊息的良好工具，使用者可以在 Sites 中討論、添加附檔、結合上述提及的各種 Google 服務，甚至與其他的資訊來源作整合，是故使用者可以輕易的加入影片、照片與線上文件等內容。使用者如編輯文件般建立自己的 Sites，並且保有控制分享的權力，可將 Sites 在個人、團隊或組織中分享，甚至直接發佈到網路上，發佈的內容將可輕易的被 Google 強大的搜尋引擎所找到。

Google Sites 富有彈性的整合特性，讓以往需要花費高成本與冗長開發時間的企業入口網站，能夠在彈指之間做出雛形，並輕易的整合各式資源，讓企業營運所需的各項服務，能夠透過第三方的服務提供者，方便的達成服務需求。本架構採用 Google Sites 作為前端的呈現介面，並且導入其協同合作的強大功能，將 Sites 視為本研究架構的介面(Interface) (Google.com 2007)。

Google 小工具(Gadget)是以 iFrames 提供的簡單的 HTML 和 JavaScript 小型應用程式，可嵌入網頁和其他應用程式中。根據 Google 官方網頁的解釋，小工具包含易於建構、多平台運行與結合 Google 現有服務、流量的優勢。因為小工具採用 JavaScript 函式庫，讓開發工作的負擔能夠減輕，有助於快速、低成本的建構出可用的服務；小工具也具有可在多個平台上運行的能力，例如 iGoogle 入口網站、Sites 與一般網頁等，都可以輕易的嵌入已完成的小工具；小工具可以有效的利用 Google 所提供的服務與帶來的流量，讓開發者能夠專注於應用服務的實現，將開發的效益最佳化(Google.com 2008)。

2008 年 4 月，Google 推出『Google App Engine』，讓開發人員可以利用 Google 的基礎環境來運作其網頁應用程式；在雲端架構環境之下，可忽略運算、儲存資源的限制，專注於應用程式本身的開發。該應用程式將可利用 Google 的網頁空間代管(Hosting)、資料庫(Bigtable)、自動負載平衡、完整支援 Google APIs、支援本地端開發等特性，讓開發人員開發出的網頁應用程式，具有可供百萬人次使用的應用能力(Google.com 2007)。



圖 5：Google Cloud Computing 服務架構示意圖

資料來源：本研究整理

二、Hadoop 運算架構

Hadoop 是一項由 Apache 基金會所發起的軟體專案，Hadoop 是一個軟體平台，讓開發人員可以輕易的在這個架構上運算大量的資料，並具備有可擴充性(Scalable)、經濟(Economical)、有效率(Efficient)與可信賴(Reliable)等優點，這個軟體平台基於 MapReduce 演算法與 HDFS(Hadoop Distributed File System)檔案系統之上，HDFS 建立可信賴的儲存叢集，而 MapReduce 演算法將應用程式分割成小塊，並利用這個龐大的資料叢集進行運算。Hadoop 已經證實可以支援多達 2000 台電腦所構築成的叢集，目前也有多個學術單位與 Hadoop 進行合作，利用這個平台架構應用在不同的領域上。(Apache 2008)

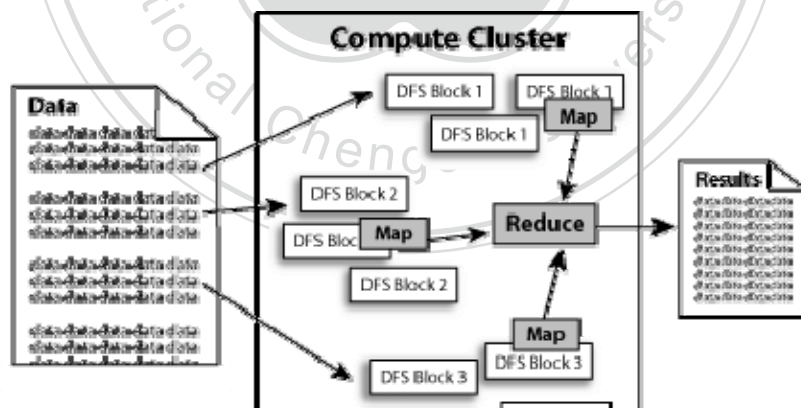


圖 6：Mapreduce 架構圖

資料來源：(Apache 2008)

MapReduce 是一種軟體設計模型，用來開發處理或產生大量的資料組合的應用程式。在開發應用程式時，開發人員用 map function 處理一對對由 key/value 組成的集合，而 reduce function 將所有處理完的集合併起來，整合完的資料就是應用程式執行的結果。利用這個模型所設計的軟體，會自然呈現平行運算的形

式，可以被運算叢集自動分散運算，分散運算的機制可以透過執行期系統(run-time)來決定，例如：資料輸入、任務執行排程、錯誤控制與必須的叢集節點間的溝通等，換言之，開發人員可以不需要考量任何分散式程式開發工作，讓平行運算變的更容易開發。

在這個架構之下，因為分散式架構的特性，MapReduce 可以用來處理幾兆位元組(TeraByte)的資料，並且可以輕易的將運算任務交由運算叢集來運算，這個運算叢集將是可以任意擴增與縮減的，Google 所推出的雲端服務中，許多是透過這個軟體架構進行開發，可以輕易的達成雲端運算的要求與優點，我們可以說 MapReduce 是一種符合雲端運算架構要求的演算法，只要軟體架構能夠以這種模式進行開發，將可以降低平行運算軟體開發上的困難。(Dean and Ghemawat 2004)



第四章 服務平台建置

本研究在雲端運算基礎之上發展滿足服務導向架構特色的服務平台，以解決雲端運算所衍生出內外部資源有效配置的問題，與企業導入時的資安疑慮。本研究結合雲端運算與服務導向架構，發展出一個四層式服務平台，包含服務介面層、中心控管層、服務供應層與基礎架構層。

應用介面層專注於人機介面的呈現，以 Gadget 負責畫面呈現，將版面與程式邏輯分開，充分利用開放介面的優勢，降低服務介面開發的負擔，讓開發者能專注於關鍵問題的解決。

中心控管層係基於服務導向架構之設計，能連接服務供應方與服務需求方，將服務需求轉送到服務供應層，並依照服務邏輯定出服務路由，可使用外部雲端運算與資料儲存之資源，並達成服務流程控管的需求；在完成服務的執行流程後，將服務供應層執行完畢的服務結果回傳給應用介面層。中心控管層具備服務導向架構的優點，並能善加利用雲端運算的優勢。

服務供應層結合各式應用服務與資料來源，透過內建函式庫與透過網路服務與第三方伙伴溝通，包含資料中心與服務提供者。服務供應層藉由將運算與資料分離的方式，讓服務需求者保持對資料的控管權力，以解決企業導入雲端架構時對於資安的疑慮；在匯集越多服務資源後，能產生更大的平台加值效益。

基礎架構層可充分利用各雲端業者所提出的服務架構、運算以及儲存資源。本研究係採用 Google 應用服務環境，透過 Gadget 連接本平台所整合的應用服務，並部署於 Google 的雲端運算環境之中。基礎架構層未來可導入更多雲端平台業者，依照個別平台的優勢與特性，充分發揮雲端運算架構的效益。

最後本研究藉由金融貸款應用之應用，以說明本服務平台之可行性，以及整合外部雲端運算服務的優越性。未來研究將探討如何將本服務平台發展成雲端運算架構交易市集。

第一節 雲端運算服務導向架構運作機制

一、服務定位

SaaS 的成熟度指標可看出企業在 IT 基礎建設上的完備程度，在加快開發速度、降低成本與風險等因素考量下，企業會考慮將服務流程外包，傳統上是向個別公司洽談『服務合約』，以界定雙方合作的權利與義務，這屬於 SaaS 成熟度第一級。隨著雲端環境的逐漸成熟，可預見未來將會出現更多服務元件商，也就是專注於服務價值鍊中一個環節的廠商，這樣的元件商可能輕易的利用雲端平台業者提供完善的軟硬體基礎建設，而創造出原本大型資服業者才能完成的應用服務。在這樣的發展趨勢下，傳統外包方式的服務合約顯得不適用，因為以往的外包模式是將一段流程統包給資服業者來承做，而非針對單個服務元件進行外包，於是在可見的未來，亦將會出現專門整合各方資源，以提供企業完整服務流程選擇的『服務套餐業者』。

服務套餐業者專注於整合各方資源與服務元件，並負責服務流程的定義與服務水準管控，也就是說在未來的資訊服務市場，應會出現兩種角色，分別是專門提供服務元件的服務元件商與提供服務套餐的服務套餐業者，在本研究中通稱為服務供應方；這兩者的緊密結合，將發展出一門新興的商業營運模式。但這兩者間仍舊缺乏統一標準來連接，本服務平台試圖扮演服務需求方與服務供應方間的溝通角色，讓各類服務供應方在本服務平台中進行註冊，透過本服務平台的運作機制，達到更優良的服務水準，換個角度來看，本平台所扮演的角色可說是創建一個多會員對多服務提供者的 SaaS 平台。

二、運作機制

（一）服務串接

在服務導向架構的系統環境中，服務資源可能分存於不同地點，為整合這些分散的資源，必須有一個中介平台來完成『服務串接』的工作。網格運算中介商透過服務供需雙方簽訂服務合約，以決定服務執行細節，但在缺乏互通標準與管理機制的的情況下，服務合約必須依照不同的需求持續修改，這不符合未來雲端運算環境的要求；而公用運算主張必須善用現有的服務資源，讓 IT 服務能夠以使用量計價的方式使用，於是衍生出 IT 服務間的組合機制這項必須解決的問題；在軟體即服務中，讓服務供應商可以利用同一套軟體架構服務不同的客戶，並達到降低成本等目標，但軟體即服務本身就應該是一項值得利用的服務資源，應被視為是能夠串接的服務。

為了將散落四處的服務資源連接起來，我們需要一張地圖幫助我們找到該走

的路，BPEL 正是扮演這樣的角色，我們可以將其視為一張藏寶圖，為了找到寶藏（完成服務），這張地圖將指引該如何在眾多道路（服務元件）中，找到最適當的道路，對於知道寶藏在哪的冒險家（服務套餐業者），將可以透過提供各種藏寶圖滿足冒險家（服務使用者）的需要。此外，透過 BPEL 描述語言進行服務流程的描述，具有提供的彈性整合、支援及時多方狀態溝通、服務生命週期管理與可恢復性等優點，以上皆是本平台的服務串接機制所要達成的目標。

本研究試圖發展服務串接的運作機制，以滿足服務資源整合的需要，依照本研究的設計邏輯，服務流程串接的參與者有三，分別是服務使用者、服務元件商與服務套餐業者。服務使用者可將本身私有的服務資源註冊到服務平台，例如企業內部的資料庫與已開發、上線的各式網路服務，這些既有的服務資源將可以被視為服務元件的一環，可在服務執行過程中被充分利用，這滿足雲端使用者需要掌握資料所有權的要求，同時將使用者本身既有的服務能量能持續在本平台發揮，不至於因為轉換服務平台而必須完全放棄先前的投資；服務元件商可以將發展出的服務元件註冊到本服務平台成為候選服務，憑藉服務導向架構資源分散的特性，元件商可利用雲端平台完善的軟硬體基礎建設，創造出大型應用服務服務，並可專注於服務價值鍊中的一環，與本服務平台達成互利共生的產業結構；而服務套餐業者掌握企業服務流程的抽象層次，利用 BPEL 商業流程描述語言，依照服務平台所提供的服務元件與使用者服務資源，進行服務能量的串接、組合，並創造出屬於各服務平台使用者所需的『服務套餐』。關於服務元件註冊所需的參數，請參見下表：

表 3：服務元件註冊參數表

Component	Name	服務元件名稱
	Type	服務型態，可分為資料庫(DB)、應用程式(AP)與服務介面(Interface)。
	Function	服務連接的方式，可分為採用本地程式庫(Lib)、網路服務(WS)或是服務介面(Interface)。
	Input	服務元件的輸入變數
	Output	服務元件的輸出變數
	Note	服務說明

資料來源：本研究整理

以註冊銀行內部私有資料庫為例，銀行使用者將內部的私有資料庫以網路服務的方式包裝成可使用的服務介面，並且在本服務平台中定義如何進行服務連結、輸出入變數與簡要服務說明，藉由這些資訊的註冊，服務套餐業者可以將此筆註冊資訊視為可利用的服務資源，並結合到服務套餐之中，換言之，客製化的服務套餐設計是可行的，透過本服務平台的溝通介面，服務雙方間無須另定服務合約，就可以達成服務外包與傳統資訊服務業者功能，最為人所擔心的資訊安全，也藉由服務存取憑證(Token)的管理下，讓使用者得以管理服務提供方對於資訊的存取權限，在不犧牲資料安全的情況下，享受服務外包的廉價與優勢。

(二) 資源配置

雲端運算講求資源的有效配置，如何盡量善用可利用資源，並藉此提升服務執行效率與可靠性，是雲端運算架構的重點，當我們可以透過服務串接機制掌握服務流程的運作，可以利用這個特性，進行最有效的資源配置，理想的資源配置可以分為三個部分：負載平衡、平行運算與容錯機制。

服務執行的過程中，本服務平台可以依照服務元件的負載情況，動態的決定運算與儲存資源的配置，這個情況能夠提升服務執行的效率，讓資源的利用更為經濟。除此之外，本服務平台可以依照執行服務的特性，決定出更有彈性的執行策略，例如選擇不同的運算平台或是將服務分由多個服務元件進行運算等，發揮平行運算的優勢。在資源配置的過程中，當任務執行失敗時，在存在多個服務元件的狀況下，本服務平台除了可以達成平行運算的優勢，尚可將該失敗的任務轉交其他服務元件接手，以確保服務的可靠度。

以下分別說明三種資源配置運作模式：

1. 負載平衡

當服務元件接收到任務(Task)，並開始執行時，服務監控機制將會紀錄服務元件的執行情況，包含執行時間與執行期狀況。當服務平台要分派任務之前，會先對監控狀況進行查詢，瞭解各服務元件的負載情況後，再挑選負載最低的服務元件進行分配。舉例來說，假如目前服務平台預計指派十項任務到三個服務元件，第一個任務的指派因為三個服務元件都是閒置狀態，將會先分配到第一個服務元件，第二個分配到第二個服務元件，依此類推，但由於各任務的執行時間可能不同，當分配完一輪後，有可能發生某個服務元件被需時較久的任務所佔據的情況，此時為避免無意義的佇列等待，服務元件將會依照上述邏輯優先選擇閒置的服務元件進行指派，讓資源達到最有效利用。以下是針對監控狀況查詢的資料結構與程式碼片段：

			ID	AP	SEC	ACTIVE
☐			1	1	2	0
☐			2	1	3	0
☐			3	1	3	0
☐			4	1	2	0
☐			5	1	3	0
☐			6	2	2	0
☐			7	2	2	0
☐			8	3	1	0
☐			9	3	3	0
☐			10	1	1	0
☐			11	1	1	0
☐			12	1	2	0
☐			13	2	1	0
☐			14	1	3	0
☐			15	1	2	0

圖 7：負載平衡記錄資料結構
資料來源：本研究整理

```

public function JobsDispatch(){
    $result = mysql_query(
        "SELECT COUNT(ID) FROM SERVICECENTER.MONITOR WHERE `AP`='1' AND `ACTIVE`='1'");
    if(is_resource($result)) $row = mysql_fetch_row($result);
    list($ap1) = $row;
    $result = mysql_query(
        "SELECT COUNT(ID) FROM SERVICECENTER.MONITOR WHERE `AP`='2' AND `ACTIVE`='1'");
    if(is_resource($result)) $row = mysql_fetch_row($result);
    list($ap2) = $row;
    $result = mysql_query(
        "SELECT COUNT(ID) FROM SERVICECENTER.MONITOR WHERE `AP`='3' AND `ACTIVE`='1'");
    if(is_resource($result)) $row = mysql_fetch_row($result);
    list($ap3) = $row;

    $min = min($ap1,$ap2,$ap3);

    switch($min){
        case $ap1: return 1; break;
        case $ap2: return 2; break;
        case $ap3: return 3; break;
        default: return 1;
    }
}

```

圖 8：負載平衡監控程式片段
資料來源：本研究整理

2. 平行運算

平行運算的高效能、高負載與高可靠度都是雲端運算架構吸引人的優勢，在傳統的程式開發上，為了達到平行運算的目的，需要深入到演算法層次針對程式邏輯進行調整，但修改的規則依照服務內涵的不同，並無法一體適用，造成即便有高效能的多工硬體，但都無法發揮其優勢的窘境。Google 所提出的 MapReduce 開發架構，提供了一個原生的平行運算架構，只要符合這個程式開發架構的程式，都可以直接進行平行處理，可充分發揮雲端運算架構的優勢。本服務平台實作了 MapReduce 程式開發架構，將服務平台中的眾多任務分散，藉由多個服務元件分別處理這些任務，也就是 MapReduce 架構中的 Map 階段，緊接著將陸續完成的任務分別存到中繼(Intermediate)的資料存放點，待任務全部執行完畢後，藉由一些 Worker 服務元件，將所有的中繼資料彙整成一個整體，即是 Reduce 階段，最後輸出成為有意義的執行結果，詳細的執行過程可以參考圖 9。

3. 容錯機制

在服務導向架構與 IT 服務外包中，最為人最擔心的問題就是服務中斷的疑慮，雲端運算架構為了提昇服務的可靠性，藉由將資料、服務元件皆複製多份或是部署多份的方式來提昇服務可靠性，以 Google 利用的雲端架構而言，一份資料至少會存放在四個不同的伺服器群，能將資料遺失的風險降到最低，並且隨時可以進行異地備援；本服務平台也利用部署多份服務元件的方式來確保服務的可靠性，舉例而言，一項金融運算服務可能同時有多家服務元件商提供，也就是說在服務平台上至少存在二個以上的相同服務，當服務平台試圖分配任務但失敗時，會自動將任務轉發給下一個相同的服務元件，讓服務得以持續、順暢的執行下去，詳細的執行過程可以參考圖 9。

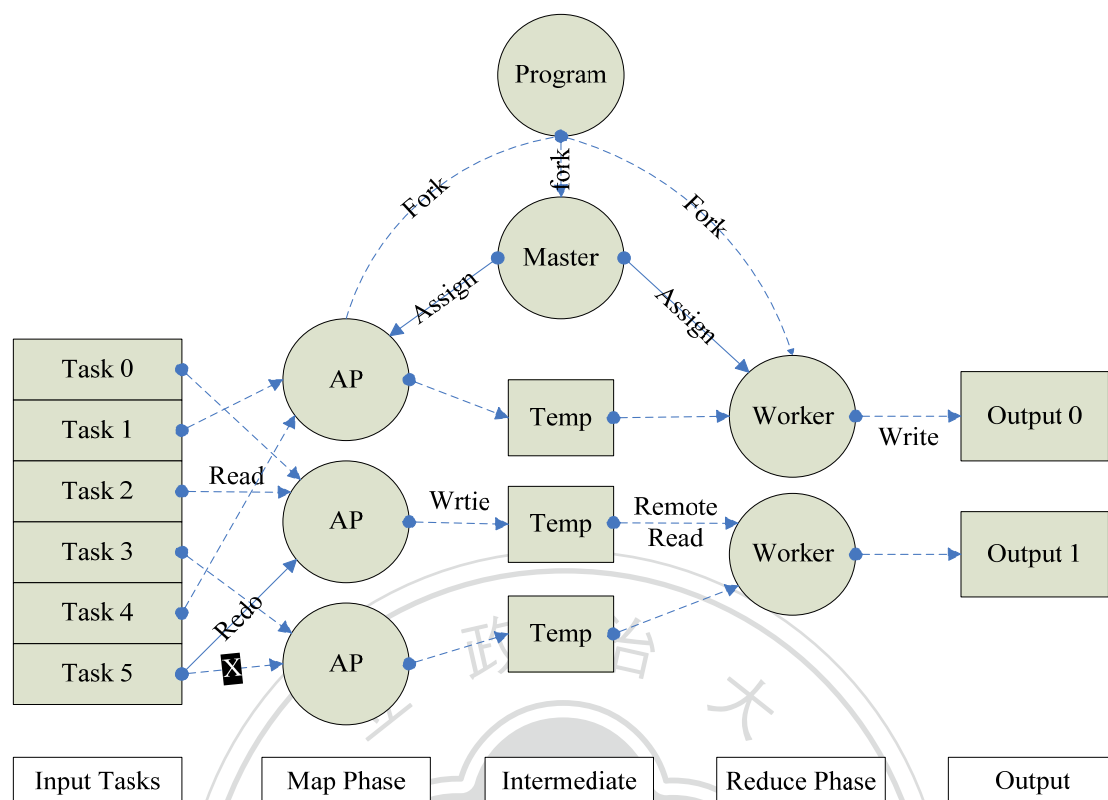


圖 9：Mapreduce 運行架構圖

資料來源：本研究整理

(三) 資料控管

在企業將資料儲存或運算放到雲端架構中時，因為雲端運算分散的特性，往往產生如何管理這份資料的疑慮，為了為避免企業導入雲端運算架構的疑慮，本研究也提出了雲端架構下資料控管的機制，資料控管的概念有兩個，分別針對資料儲存與資料運算的角度出發。

在運算服務的過程中，本服務平台透過授權的方式，讓服務元件無法擁有完整的資料存取權力，而僅能存取該節點所需要且被授權的資料，並利用 MapReduce 的分散運算概念，使該經過授權的資料經過拆解，供不同的服務元件進行運算，亦即單個服務元件所運算出來的結果是沒有意義的，而必須經過服務平台的組合後，才會產生有意義的結果。於是一些較為敏感的任务將可以放心的交由外部元件商完成，降低隱私等相關資安議題的疑慮。

在資料儲存上，本服務平台將資料分由不同的資料中心進行儲存，跟運算服務的分散概念相同，交由不同資料中心儲存的資料是經過分散甚至加密過的資訊，而握有彙整資料完整格式的鑰匙，將可掌握在企業手中，換言之，資料可能分存在雲端中的任何位置，但沒有經過有結構的組織，任何一個單點所看到的資訊都是片面且無意義的，確保資料存放在雲端架構中的安全性與隱私顧慮。

以銀行客戶資料為例，說明如何在保有資料隱私的情況下，將資料外包到雲端儲存空間中。假定銀行將客戶資料、收入、職業與擔保發票資訊等分別儲存到私有內部資料庫與兩個外部雲端資料庫，在銀行內部資料庫中，僅僅儲存姓名與身份證字號兩項基本資訊，而收入、職業與擔保發票資料這兩項資料將分別外包到不同的雲端資料庫中，在內部資料庫中，使用者可以明顯的知道顧客的資料細節，而在雲端資料庫中，僅可以看到 Token、收入與職業等資訊，其中 Token 是藉由銀行內部資料庫中的身份證字號，經過 MD5 加密後所得的亂碼。當單獨的審視這個資料表，我們除了知道職業與收入外，並無法得知這筆資料到底真正屬於誰，能夠將兩者配對起來的，唯有持有鑰匙，也就是藉由身份證字號產生正確的 Token 後，方知該筆資料真正的歸屬與完整資訊。在雲端發票資料庫也是一樣的情況，若不知道正確的 Token，亦只能看到片面的發票登錄資訊，而無法產生真正有意義的資訊。

在這樣的情況下，唯有將三處資料庫都集合在一起後，才能看出客戶資料的全貌，但唯一具有這樣能力的，只有掌握庫客戶真實身份資料的銀行本身，也就是說，資料中最有價值的部分，也就是至為關鍵的部分，依舊掌握在使用者的手中，即便存放在雲端中的資料遭到有心人士的窺視與竊取，也無法還原完整資訊，如此可確保企業免除資料外洩與喪失控制權的疑慮。

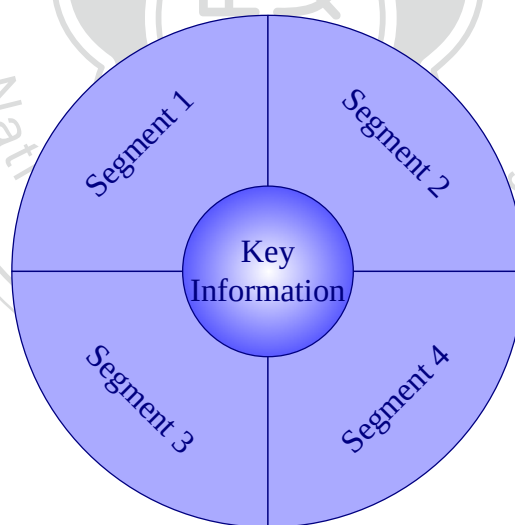


圖 10：資料控管架構圖

資料來源：本研究整理

（四）管理機制

本服務平台採用 AAA 協定來進行服務管理，這個機制被普遍的利用在許多網頁服務中，AAA 機制分別是：認證 (Authentication)、授權(Authorization)與統

計(Accounting)。

認證機制決定使用服務的權力，本服務平台採用常見的帳號密碼驗證登入，在經過身份資料的認證後，本服務平台可知道使用者使用服務的範圍，因為不同的使用者存在相異的權限。以本平台的應用為例，認證機制可分為兩個使用角色，分別是服務需求方與服務供應方，服務需求方需要取得對於服務套餐的使用權力，惟有經過認證才具有對服務的使用權力；而服務供應方則需要註冊服務的權力，必須經過認證才具有註冊服務成為候選服務的權力。

授權機制決定使用服務的範圍，常見的應用是依照會員等級的差異授與不同的服務水準，經過認證的使用者，本服務平台將會授與其可使用服務的範圍。以本平台應用為例，授權機制可分為兩個使用角色，服務需求方需要取得對於服務套餐的使用權限，唯有經過授權才具有對服務的使用權限；而服務供應方則需取得對運算資料的授權，以取得完成服務所必須的資料輸入。

統計機制紀錄使用者的活動情況，例如使用次數、時間等資訊，本服務平台將記錄服務供需雙方的使用情況，以作為服務計價的依據。本服務平台的統計機制將會記錄服務的使用次數、運算資源與儲存資源的使用情況，這些資訊可以作為服務運作的歷程記錄，並可依照使用量的多少，來決定收費的依據，達成公用運算的目標。

舉例而言，當服務使用者登入服務平台時，必須先輸入帳號密碼，這時認證機制將會決定使用者能夠使用的服務，並且藉由授權機制將呼叫服務的方式列出，隨時供使用者使用，當使用者每次登錄、執行、訂購服務時，這些動作都會藉由統計機制完整的紀錄下來，並製成統計報表供使用者參考，這些也是服務平台向使用者收費的重要依據。

三、設計架構

本研究之雲端運算服務導向運作機制包括服務提供方、服務需求方與第三方平台，是原生的分散式架構，並保持服務雙方間呈現鬆散的關係；雲端運算服務導向架構平台係根據服務導向架構之分散式架構、關係鬆散的介面、開放的標準、定義明確的服務合約與服務可發掘並動態取得等技術特性所發展。本服務平台將透過開放標準的網路服務來達成溝通的功能，並藉由中心控管層來定義服務合約，決定服務的執行內容與流程；服務提供方與資料中心可以動態加入或移出本平台，一切服務執行的依據將由中心控管層的服務合約來決定，達成服務可發掘與動態取得的特性。

為在雲端運算基礎架構上，滿足服務導向架構的特性，本研究以拓展 MVC 架構建立一個新的系統設計架構，這個架構可分為應用介面層、中心控管層、服務供應層與基礎建設層。應用介面層是服務介面，由 Google 提供的 Gadget 負責

扮演服務需求與供應雙方的橋樑；中心控管層決定服務路由，依照服務需求，組合不同的服務供應方與資料；服務供應層結合服務供應方與資料中心，以完成各種服務需求的實際應用功能；基礎建設層採用 Google 提供的應用服務環境，這個環境構築在雲端運算架構之上，服務提供者與消費者共處於雲端之中，必須藉由中心控管層進行服務整合。服務提供者於平台註冊後，即可加入服務市集，成為中心控管層的候選服務；使用者只需透過瀏覽器，即可使用各種應用服務，簡化服務消費流程。本研究架構圖請見圖 11：

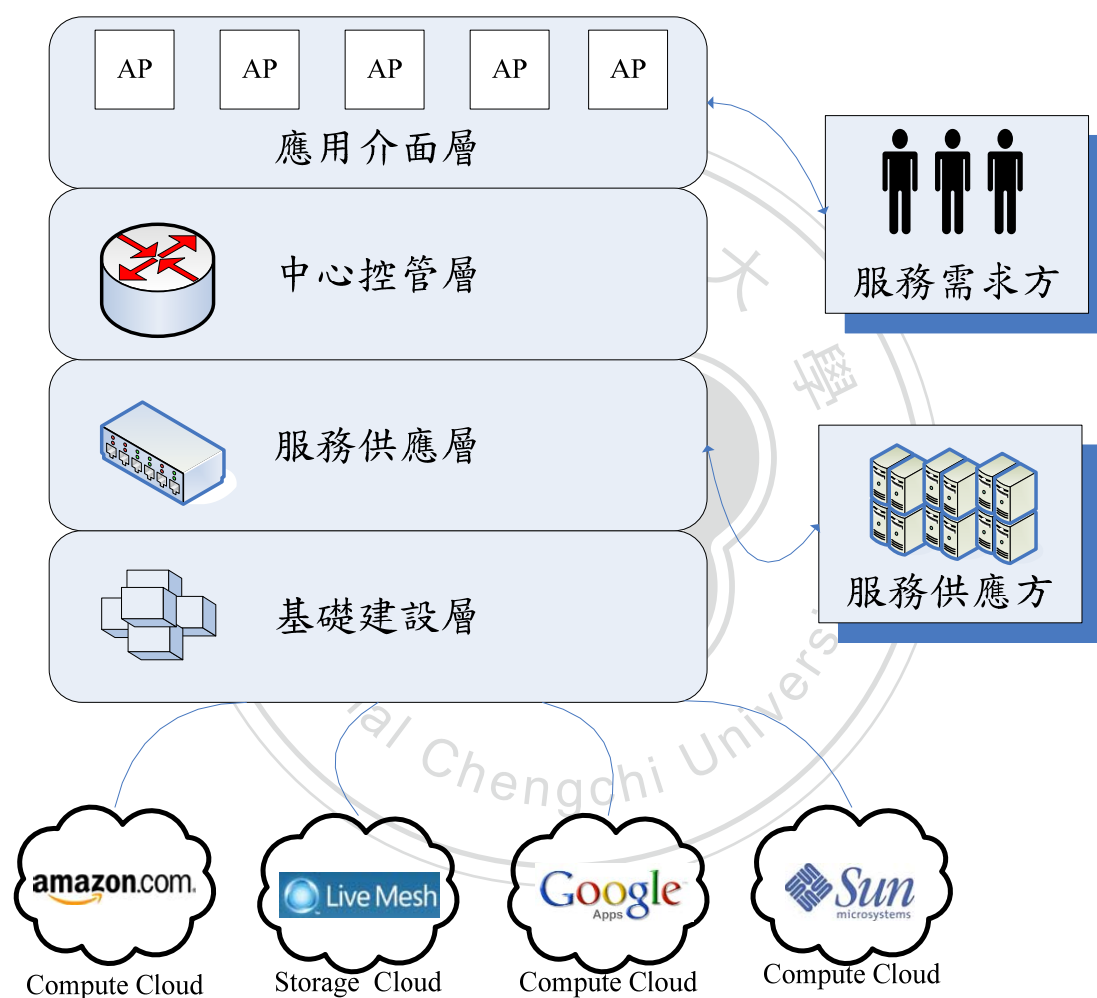


圖 11：雲端運算服務導向架構圖

資料來源：本研究整理

第二節 雲端運算服務導向架構平台

一、應用介面層

應用介面層負責人機介面的呈現，以往的網頁應用程式開發工作，程式人員需同時負擔應用程式開發與介面美工設計的工作，讓降低專案開發的效率與品質。導入 MVC 架構後，雖然利用樣版(Template)的概念，將畫面與邏輯分開，但介面設計的可攜程度仍然很低，版面必須針對個別專案調整的情況也很常見，導致專案開發人員花費許多力氣在介面開發上，而無法專注於關鍵問題的解決。

Gadget 是一種小型應用程式，可嵌入網頁和其他桌面應用程式中。使用 Gadget 的方式嵌入網頁應用程式，可以輕易的將程式部署到各式網頁平台中，無論是 iGoogle、Google Sites 或一般網頁，都比過去減輕許多部署上的困難。這種往平台靠攏的設計思維，隨著各網路平台紛紛開放各式 APIs 後，開始讓許多開發者致力於發展單一功能的網頁應用，無須獨立創建嶄新的平台，而社群也能夠因為開發者能夠輕易的加入開發團隊，讓整個社群更為有活力，並且逐步演變成平台間的領導廠商。

服務需求方需取得 Google 環境的登入帳號，可以使用 iGoogle 的個人入口網站，或是如 Google Apps 的應用社群來註冊使用服務，當使用者需要使用本系統的服務時，只需要將服務需求，透過增加小工具的方式，新增到應用程式介面上，即可開始使用本服務平台所提供的應用服務。使用者無須安裝任何軟硬體，也無須作任何的設定工作，只需要具備有瀏覽器的電腦，連上網際網路即可使用服務，簡單、直覺且方便行動使用。個別服務會切分成單獨的 Gadget，使用者在使用上不容易混淆，可以用看待個別服務的使用觀念來著手進行，降低使用者混淆的程度，提供更友善的使用體驗。

本研究將 Gadget 視為客戶端與中心控管層溝通的介面，程式開發人員無須分心於版面設計與美工，只要專注於服務元件的撰寫，即可讓更多使用者輕易使用開發出來的服務。以應用服務角度的雲端運算定義而言，透過中心控管層決定服務邏輯後，可整合多方服務提供者與資料中心，結合大量服務群與資料群來完成服務的能力，發揮雲端運算架構的平台整合能力，讓服務消費者的消費流程更為簡易。

二、中心控管層

中心控管層負責連接服務供應方與服務需求方，當客戶端在應用介面層上使用服務時，客戶端將透過 Gadget 發送需求(Request)到中心控管層的 WebServer，此時 WebServer 將把需求轉送到 Dispatcher 元件，依照服務邏輯的定義，決定此

服務應執行的流程與所需相關資料；當決定執行流程與所需資料後，Controller 元件將在服務供應層中透過 WebService、Library 與 SourceRequest 元件組合資源後，Service Provider 與 Data Center 會依照 Controller 要求的流程完成客戶端要求的服務，最後將服務結果透過 View 元件在客戶端呈現。所有候選服務資源皆為外在第三方伙伴所提供，各種服務與資料來源可以動態的在服務供應層新增或移除，中心控管層負責組合累積的服務資源以完成客戶端的需求。

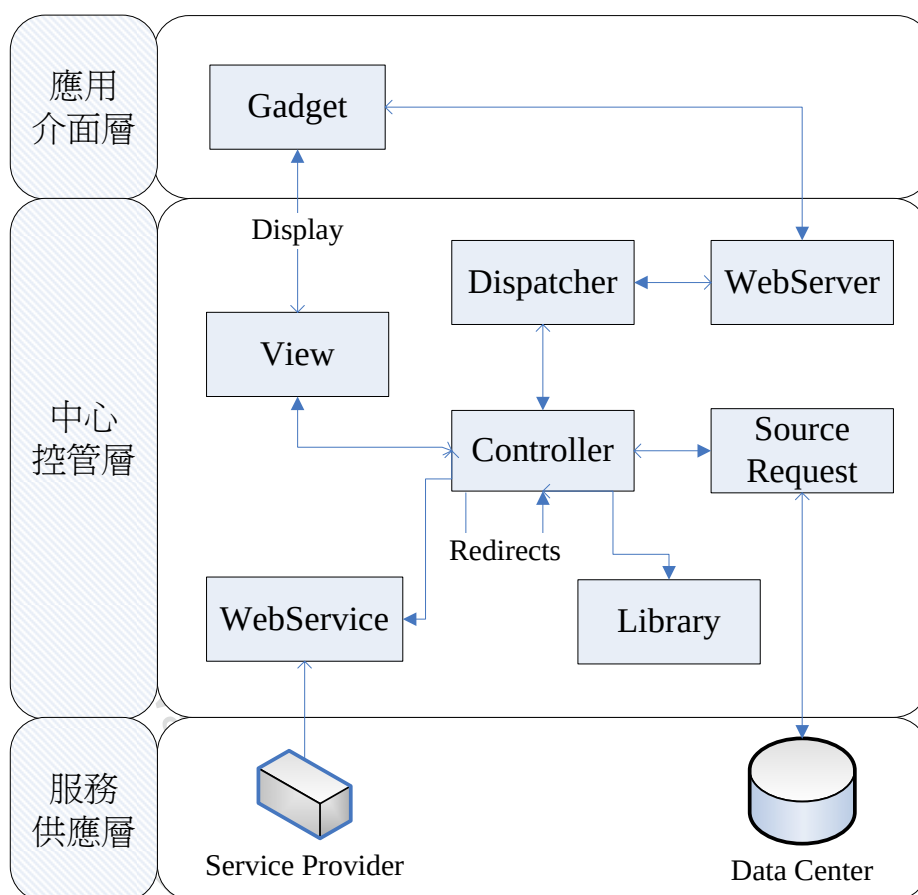


圖 12：雲端運算服務導向架構流程圖

資料來源：本研究整理

中心控管層負責接收服務需求，定義出適當的流程後，決定結合相對應服務的方式，並且取得所需的應用程式與資料，在完成服務後，回傳給使用者。服務平台的系統架構，可參考圖 13：

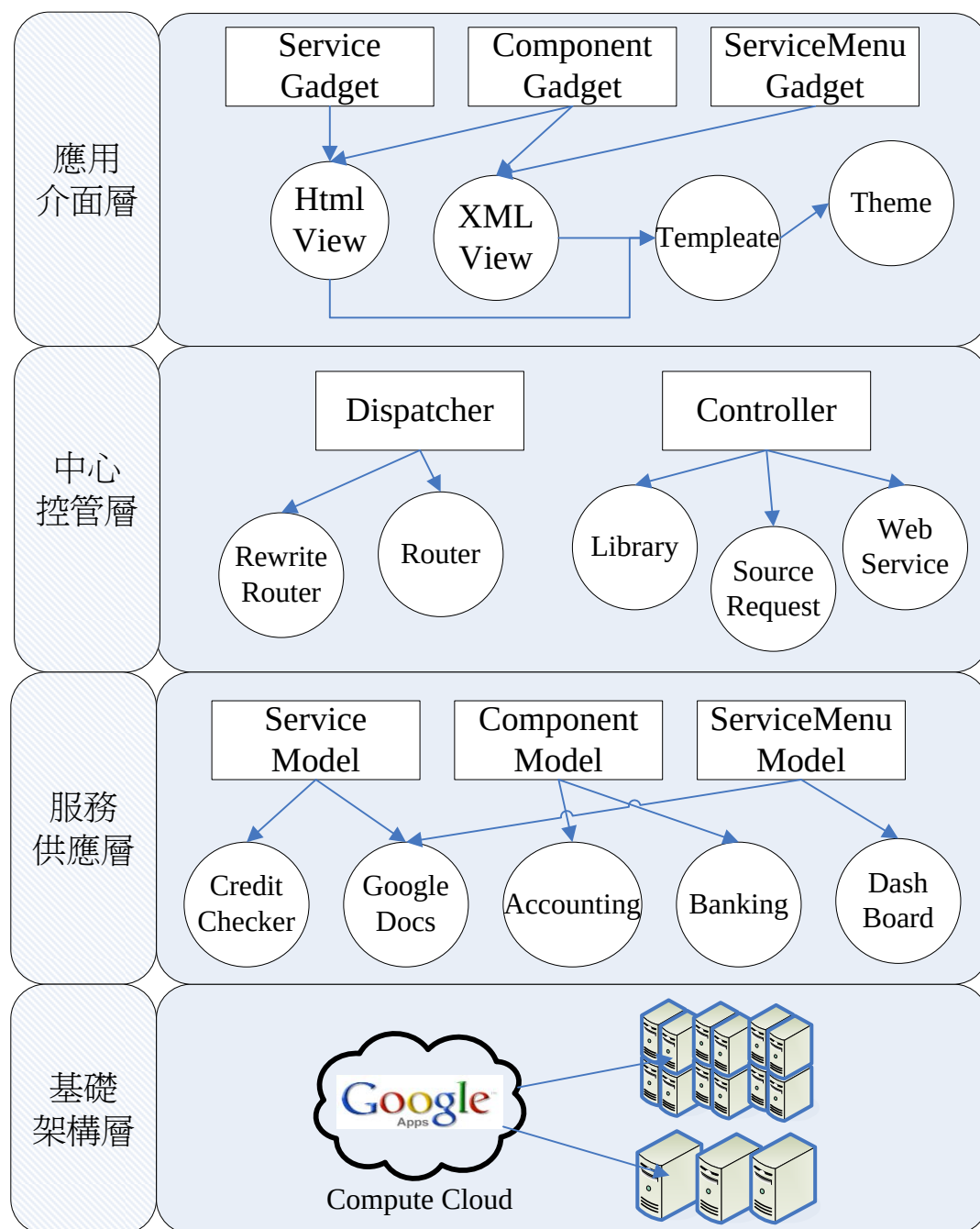


圖 13：服務平台運作機制

資料來源：本研究整理

本服務平台利用 PHP 動態語言的 Late binding 與 Autoload 功能，設計出服務路由機制，客戶端加入服務 Gadget 後，本服務平台透過 XMLView 元件回傳 XML 格式的服務資訊，讓客戶端可以將小工具加入如 Google Sites 等 Web 服務介面。客戶端使用服務時，Gadget 透過 HTTP Get 發出服務需求，在服務平台接收到服務參數後，透過 Dispatcher 元件的服務路由機制決定對應執行流程，這個動作由 Router 與 RewriteRouter 兩個元件進行，將原本的服務需求重新導向到新

的服務流程裡。服務流程的定義在 Controller 元件中進行服務串接，串接的服務存放於 Model 中，Model 的服務實作可以透過 Webservice、Library 與 SourceRequest 等元件與實際服務提供者進行連接，並定義出服務執行內涵，也就是進行實際的執行流程。最後的執行結果會引導(Render)到 HTMLView 元件，產生人機介面所需的 HTML 碼，結合顯示用的 Template 元件後呈現給使用者，換句話說，在程式邏輯的開發過程中，程式邏輯與介面呈現將交由不同階段完成，讓應用程式的開發能夠更有效率。

中心控管層連接服務供應方與服務需求方，將服務需求轉送到服務供應層，並依照服務邏輯定義出服務路由，滿足在服務導向架構對於軟體架構的要求。讓服務供需雙方位於分散式架構中，可發掘並且動態的取得服務，採用開放的標準並且以 Dispatcher 元件定義明確的服務合約，以達成服務流程控管；完成服務的執行流程後，將服務供應層執行完畢的服務結果回傳給應用介面層，善加利用雲端架構優勢以提升服務效率。換言之，中心控管層具備服務導向架構的優點，並能善加利用雲端運算的優勢。

三、服務供應層

服務供應層可以輕易結合各式服務與資料來源。在服務部分，包含本地端所內建的函式庫(Library)與透過網路服務與第三方伙伴溝通的各式應用服務；資料來源包含提供存取介面的專業資料管理公司與客戶端的本地端資料庫。本研究之服務平台所提供的應用服務乃是透過不同的應用程式與資料所交互組成的，當匯集更多服務資源後，本服務平台能夠產生更大的加值效益，並隨著雲端運算架構環境持續發展，運算服務變的無所不在，服務供應層可逐步朝向『全球雲端服務交易市集』的特性發展，讓服務供應商間交換運算資源，以充分發揮雲端運算的平台特性。

應用服務提供者與資料中心所組成的服務供應層，皆透過網路服務的方式進行溝通。服務提供者可以是任何平台業者所提供，因為網路服務跨平台的特性，本系統架構可以輕易的與不同類型的網路服務結合；至於資料中心則扮演管理資料的角色，服務提供者並不直接擁有資料，而是必須在取得客戶端授權後，向資料中心索取適當的資料，這可以確保資料的隱私能夠得到最大程度的保障；資料中心也可與客戶端的私有資料庫結合，透過中台的授權機制，仍可在保有資料隱私的情況下，享有雲端運算與本架構所帶來的優點。

服務提供方的連接方式可分為二種，第一是透過網路服務的方式，與服務提供者與資料中心進行聯繫；第二是透過本地端開發的 APIs 函式庫與服務進行聯繫。以透過網路服務來取得服務而言，服務提供者只需要在本平台註冊後，並同時提供網路服務介面，即可成為本平台的候選服務，在使用者發出服務需求後，將可以直接透過本平台進行介接，完成服務需求。於資料中心的連接也透過網路

服務進行連接，但概念上有些不同，本平台將資料中心定位為類似目錄服務，當發出要求資料的需求後，資料中心將會發出存取資料相關的連線資訊，例如連接位址、參數等，服務平台將會透過這些資料，進行與資料庫的連線工作，取得資料以完成服務，也就是說，資料中心代表資料儲存的導引者，這個導引的對象，可以是網路上任何一個資料庫，包含專業資料庫代管公司，或是公司內部的資料庫系統等，都可以透過資料中心作一個轉接的動作，而能夠為本服務平台所用。

如圖 7 所示，客戶端將需求交由中心控管層進行處理，第三方服務提供商所提供的應用服務與不同的資料來源，將在中心控管層結合，並且最終完成服務。服務提供商與中心控管層本身並不直接擁有資料，而是透過客戶端的授權，取得服務所需的資料，再透過服務提供者的運算資源完成服務。保有使用者對於服務流程與資料控管的權力，並避免企業在導入雲端架構時，所衍生出的資訊安全與隱私等疑慮。



圖 14：應用服務與資料分離示意圖
資料來源：本研究整理

四、基礎建設層

基礎架構層採用 Google 應用服務環境，透過 Gadget 連接本平台所整合的應用服務，並部署於 Google 的應用環境之中。本應用環境同時提供了 Gmail、Google 日曆、Google Talk、Google Docs、Google Sites 以及網頁建置服務等應用程式，讓服務間的溝通、單一窗口登入與協同合作等功能轉由 Google 來提供；搭配 Google App Engine 更可以充分利用 Google 運算資源。

在過去的網頁程式開發，如使用者的帳號管理、登入認證與溝通機制等，都需要花費一定的開發成本才能提供，但是這往往跟應用程式本身的功能沒有太大的關係，也就容易造成開發資源的浪費，Google Apps 同時完成了會員登入管理

與相關的溝通機制，讓開發人員能夠省去繁複的認證機制，將應用程式的開發單純到功能本身，減輕應用程式開發的困難度，並縮短開發時間。網頁程式常常使用到的 Email 與即時通訊等機制，Google Apps 也提供了相對應的資源，開發人員不需要再分心開發這些基礎服務，而能專心於程式功能本身的開發工作，讓開發工作的進行更有效率。

IBM、HP、Intel、Yahoo、Sun 與 Microsoft 等公司也相繼投入雲端運算相關基礎設施的研發工作，透過本服務平台，可以輕易的利用各陣營所提出的基礎架構與運算資源，並在基礎建設層中進行連接；亦即，本研究以 Google 的基礎應用環境為例，示範與雲端平台業者共生的整合營運模式，在相關雲端基礎設施益發茁壯的情況下，本研究之服務平台可扮演雲端運算服務市集平台的角色，結合不同服務供應方與雲端基礎建設供應方，提供強大的服務能力，並且充分發揮雲端運算與服務導向架構的優勢。



第五章 服務平台實作與展示

第一節 金融發票融資應用案例

台灣的產業以電子與製造業為主體，為了增強產業的資訊化程度，經濟部以電子化採購(e-Procurement)為核心，推動 IBM 等國際大廠與國內主要電子廠商建立從設計至採購階段之電子化供應鏈，稱之為 A 計畫；另外由國內重要的電子資訊大廠與其上游零組件供應商，建立從採購至製造階段之電子化供應鏈體系，稱之為 B 計畫。在『資訊業電子化 AB 計畫』順利推行之後，加強了企業間的資訊化程度與國際的資訊連通能力，成功協助國內資訊業者建立電子化供應鏈體系。有了 AB 計畫的成功經驗，經濟部提出了電子化金流計畫 C(Cash)計畫，本計畫的重點在於將既有之電子化供應鏈接軌與金融體系緊密結合，包含銀行業、製造中心廠與供應商等多方；所建構出的金融網絡，以電子化供應鏈上之商流資訊為基礎，希望達成全球收付款、多行帳戶整合、線上融資等目標。其中以交易資訊進行線上融資，其融資項目包括訂單前融資、訂單融資、驗收單融資、發票融資、應收帳款融資，換言之，供應商可彈性的向銀行業者申請融資，以及時反應快速變遷的商業環境，讓以中小企業為主的台灣產業鍊能夠突破資本的限制，更能活絡產業的整體營運。

此外，由於金融環境瞬息萬變，實施多年之巴賽爾協定內容已無法充分反應金融機構內部資本效益及面臨風險的全貌，巴賽爾委員會於 1999 年公佈新資本適足率資本架構(A new capital adequacy framework)，並於 2001 年公佈新版巴賽爾協定草案，包括最低資本適足、監理審查及市場制約等三大支柱，中華民國金管會亦預計於 2007 年底推動金融機構全面滿足新版巴賽爾協定。以放款為例，放款為金融業者核心業務之一，目前各家金融業者皆有完整的貸款作業流程，但為滿足新版巴塞爾協定的風險性資產計提需求，必須對銀行內部的傳統作業流程作一整體性的分析與調整，並修改相關的資訊系統，對銀行的負擔將會相當沈重。況且銀行內部採用的系統架構多且複雜，考量成本與時間效益，如何能夠以最低的系統轉建成本，快速且有彈性的滿足金融業所面臨的需求，成為各家銀行努力思考的重要課題。

經營環境的改變帶動企業的變革，而扮演支援角色的資訊系統，必須隨時依照不同的需求而更新，在經營環境日趨嚴苛的今天，如何降低成本，提升己身的競爭能力，往往是左右企業能否持續發展的重要因素，如果銀行的系統開發思維仍舊依照傳統的開發模式，以重新評估、開發系統的方式來面對環境的挑戰，需耗費高成本與較長的反應時間，此外，隨著資訊硬體設備日新月異的更新，企業必須編制相關人員維護新購入與舊有的資訊設備，花費大量的前置成本，但我們

可以發現大部分的時間這些資訊設備都處於閒置的狀態，讓資產利用率低落，造成投資上的浪費，基於以上理由，銀行內部的系統開發思維不應再採用過去缺乏彈性的開發思維。

為減少系統開發上的負擔，資服業者開始倡導服務導向架構的開發思維，在保持舊有系統的情況下，降低系統建置與開發的成本與時間，達成快速且有彈性的開發目標，並將銀行內部可能使用到的相關服務整合成網路服務，銀行作業流程可藉由此平台所提供的流程架構，在傳統資訊系統作業流程不做大幅更改的情況下，仍舊能滿足新巴塞爾規範。在流程面滿足服務導向架構後，服務效能與可靠度上，藉由雲端運算架構可以幫助讓整體的服務能量更為提升，並且讓軟體架構具有網格運算、公用運算與軟體即服務的優點，具有強大的服務能力與穩定性，並可依照業務需要的使用量來計費，提升資產利用效率，讓企業能夠快速、低成本的回應環境需求。

但目前所提出的雲端服務，都要求使用者將資料、程式通通放到雲端中，引發企業對於資料安全與服務品質的質疑，尤其在高度講求資訊安全的金融業，不太容易接受將敏感性資料外包到第三方業者，這也是雲端服務推行上所遇到的最大挑戰。本研究試圖提出一項解決方案，讓使用者能夠享受雲端運算架構所帶來的好處，並同時保有對於敏感性資料的控管權力。本研究所發展出的四層式服務平台，以中心控管層整合服務需求方、服務元件商與服務套餐業者等三方，藉由發展出的資料控管機制，讓放在雲端的資料必須經過擁有人授權後，才能被存取使用，亦只有資料擁有人能夠解讀資料，讓資料安全性能夠確保，免除使用者在導入雲端運算架構時的隱憂；此外，為組合各式服務元件以提供服務，中心控管層具有服務串接的功能，能夠整合不同來源的服務元件，並能依照各元件的負載情形動態的分配服務資源，達成資源的最有效利用；最後中心控管層所提供的管理機制，讓這個服務平台能夠成為服務供需方與套餐業者的交易平台，發展出完整且可行服務交易模式。

為驗證本研究所提出之服務平台確實具有應用上的可行性，本研究從銀行的角度出發，以滿足新巴塞爾協定要求的風險性資產計提的發票融資作為範例，藉此說明服務平台的設計機制與其運作情況。在本範例中，銀行可以透過本服務平台進行貸款客戶的管理，並且能夠計算出申貸客戶的貸款額度與違約機率(probability of default, PD)，銀行內部的資料庫僅儲存關鍵的客戶隱私資訊，其餘如收入資料、發票資訊等都外包給雲端資料庫業者進行管理，所用到的服務元件、顯示介面與平台等也都完全外包給第三方服務業者，除了滿足服務導向架構的要求外，也能享受到雲端運算架構的優勢。下圖是本研究範例的服務流程概念圖：

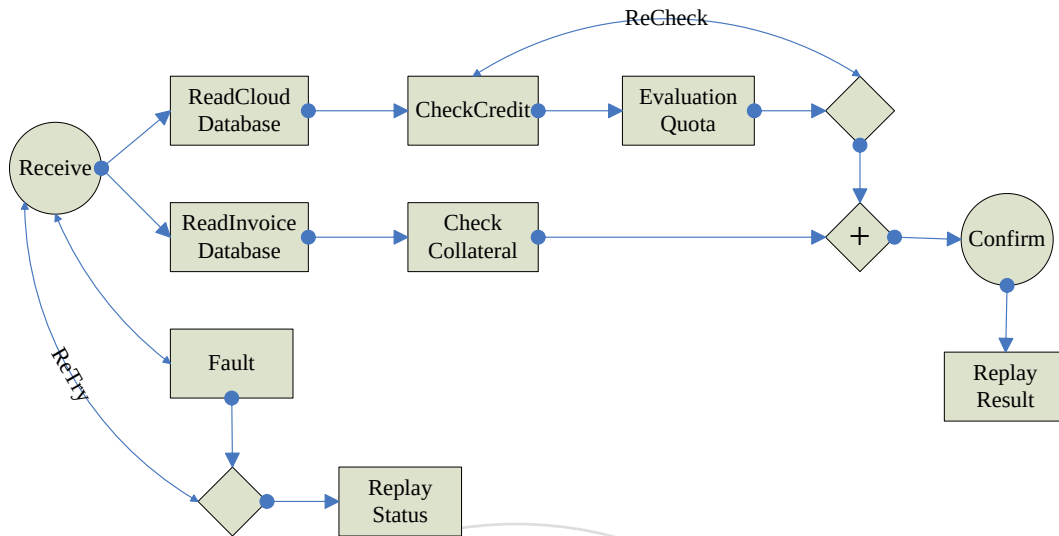


圖 15：金融發票融資案例服務流程概念圖

資料來源：本研究整理

以上案例流程代表一個完整的貸款信評過程，當使用者觸發信評流程後，首先會以平行流程的方式同時進行兩個雲端資料庫的存取工作，分別是『客戶雲端資料庫』與『發票雲端資料庫』這是為了取得由外部資料庫業者所代管的雲端資料，緊接著進行『信評確認』與『擔保品確認』服務，其中『信評確認』服務尚包含一個『評估貸款額度』服務，並且以迴圈(Loop)的方式重複進行服務，以確保其正確性，兩個平行流程最終會整合並且進行最後確認(Confirm)的工作，最後將執行結果回報給使用者，值得一提的是，若發生了服務執行錯誤的情況，如服務或資料庫失效等狀況，將會進入到錯誤(Fault)處理流程，並且自動嘗試相關定義的備援服務，盡可能讓服務能夠順暢的執行完畢。

第二節 服務平台分析與設計

依照本研究提出服務架構，一個服務所參與的角色有三個，在本應用案例中，所參與的服務需求方、服務元件商與服務套餐業者等三方，皆有各自需扮演的角色與提供之功能，本章節以系統分析的角度出發，說明其功能與設計概念，分述如下：

一、服務需求方

服務需求方無須具備強大的運算能力，也無須大量的初期軟、硬體投資，只需要透過穩定的網際網路連線能力，即可使用本服務平台上的各項服務，並且以量、次計費，享有網格運算、公用運算、軟體即服務與雲端運算架構的優點。在利用本服務平台取代原先需要自行開發、管理的種種商業流程後，服務需求方可以調整 IT 投資策略與思維，以更經濟的方式來利用 IT 資源，將軟硬體維護等支援性活動外包，並專心於屬於本身的核心競爭優勢。

本應用案例設定的服務需求方為金融業者，因為金融業往往需要龐大的資訊服務資源，來處理眾多分行與大量的顧客資訊，非常適合導入雲端運算架構；不過金融業也是對於資料安全高度要求的行業之一，雲端運算架構的概念雖然立意良善且有諸多好處，但關鍵在於資料外包所引發的資訊安全與服務品質疑慮，若無法妥善的解決這項問題，雲端運算架構很難被企業廣泛接受。

為解決在資料安全外包上的疑慮，可透過本服務平台所設計的中心控管層之資料控管機制，將服務需求方不便外包的關鍵資訊留存在企業內部的資料庫，並在確保資料安全的情況下，外包其餘資料到雲端資料庫之中。在雲端中的資料看起來將會是分散且沒有意義的，服務元件商與雲端資料庫業者只能看到部分資訊，唯有掌握內部關鍵資訊的使用者本身，才能將這些離散的資料組合成有意義且完整的資訊，利用這樣的方式確保關鍵資料的掌握與資料安全，期能消弭企業在導入雲端運算架構的疑慮。

當企業決定將應用服務與資料外包到雲端時，必須考量如何確保資訊服務的掌控能力，以避免雲端架構所帶來相關的資安疑慮，首先要考慮如何進行資料的切分，以本金融發票融資案例為例，銀行所掌握的是顧客的基本資料、收入、職業資訊與擔保品資訊，為了確保資料的掌控能力，本案例將顧客的基本資料（姓名與身份證字號）放在銀行端內部，而將收入職業與擔保品資訊分別外包到不同雲端資料庫中，於是單就個別雲端資料庫而言，只能看到一連串不完整的資訊，

但這些不完整的資訊，已經足夠服務元件商在進行運算服務時所需，當企業彙整運算結果與雲端資料庫後，才可以看出完整的服務執行結果，如此既能享受到雲端運算架構的低成本與強大服務能力，又能同時確保對於資料的掌控權。

二、服務元件商

服務元件商是由眾多服務元件開發業者所共同組成，依據提供服務的類型，服務元件可分為應用元件（AP）、雲端資料庫（DB）與使用者介面（Interface）三類，以下分別就各類型的服務元件進行說明：

（一）應用元件（AP）

以提供運算能力與應用程式為主的服務元件歸類為應用元件。在服務導向架構中，認為應該以網路服務為主的鬆散耦合結構來組合服務，以提升整體開發的效率與資源利用率，是故各式服務應以元件的型態出現，透過網路服務或是程式庫的方式進行連接，讓商業流程中的每一步驟能交由不同的服務提供者完成；而雲端架構也強調服務需求端不需太多的運算能力，將運算所需要的資源轉由雲端中的運算群為其完成。

雲端運算架構所強調的重點就是透過多個運算群，來達成強大的資訊服務能力，本服務平台利用 MapReduce 演算法的概念，動態配置服務需求（Service Request）至不同應用元件中，這樣做的用意在於達成負載平衡、平行運算與容錯機制等三大功能，讓服務資源能夠被妥善的利用，並且透過平行運算提升服務效率與能力，最後提升服務的可靠度，充分發揮雲端運算架構的優勢之餘，讓本服務平台所搓合的服務供需雙方都可以互蒙其利。

在本應用案例中，註冊的應用元件一共有四個，分別是『ServiceAccounting』、『CreditChecker_A』、『CreditChecker_B』與『CreditChecker_C』，分述如下：

1. 『ServiceAccounting』服務元件是針對服務平台中使用者行為進行記錄，例如登入次數、服務使用次數與時間等資訊，使用者可透過這項服務來進行 IT 投資估計，而這項資訊也是服務平台對使用者進行收費的主要依據。其設計概念是參考流水帳的記錄方式，例如每次使用者登入後即新增一筆紀錄，包含登入時間、登入者等；在使用服務時，也會針對使用服務的次數與時間等資訊進行記錄，累積的紀錄可以透過『ServiceAccounting』進行瀏覽。下圖是本服務元件的資料結構圖：

←T→	ID	TYPE	U_ID	C_ID	DATE
☐  	1	Login	1	1	2009-05-09
☐  	2	Login	1	1	2009-05-09
☐  	3	Login	1	1	2009-05-09
☐  	4	Login	1	1	2009-05-09
☐  	5	BankManager	1	1	2009-05-09
☐  	6	BankManagerTime	1	1	2009-05-09
☐  	7	Login	1	1	2009-05-09
☐  	8	CheckCredit	1	1	2009-05-09
☐  	9	Login	1	1	2009-05-09
☐  	10	Login	1	1	2009-05-09
☐  	11	Login	1	1	2009-05-09
☐  	12	CheckCredit	1	1	2009-05-09
☐  	13	CheckCreditTime	1	30	2009-05-09
☐  	14	CheckCreditTime	1	30	2009-05-09
☐  	15	Login	1	1	2009-05-09
☐  	16	Login	1	1	2009-05-09
☐  	17	Login	1	1	2009-05-09
☐  	18	Login	1	1	2009-05-09

圖 16：『ServiceAccounting』服務元件資料結構

資料來源：本研究整理

2. 『CreditChecker_ABC』等三項服務元件，是為模擬服務市集的實際運作情況，設定 ABC 三家專門為銀行提供貸款信評的公司，這三家公司提供的功能完全相同；在本應用案例中，會把銀行的評估需求分依照負載情況分給這三個服務元件進行處理，若發生服務元件失效等突發情況，也可隨時由其他兩個服務元件接手處理，提升服務的效能與可靠度。本元件的設計邏輯是接收貸款人的職業、收入、擔保發票名稱、發票金額、買受人與發票日期等資訊，其後藉由模擬的內部風險評等模型計算出可貸金額與違約機率，其計算公式與規則如下：

表 4：貸款信評計算公式表

可貸金額 =	(收入 * 職業調節因子) + (擔保發票金額 * 買受人調節因子)
違約機率 =	職業調節因子 * 買受人調節因子

資料來源：本研究整理

（二）雲端資料庫（DB）

以提供儲存空間與資料庫的服務歸類為雲端資料庫。在企業資訊系統中，資料的儲存、處理與備援等工作都是相當重要但又瑣碎的工作，對於企業而言，這些維護的成本並無法直接創造收益，因此往往被視為設備維護等成本支出。在現今競爭的商業環境中，如何降低成本以維持本身的競爭力，將會是未來 IT 投資所要考量的重點，於是 IT 服務外包蔚為風潮，也是企業尋求降低成本的途徑，這些需求衍生出許多專業的資料、設備代管廠商，就雲端業者而言，強大的運算、儲存基礎架構與低廉的成本，是企業尋找外包的理想解決方案。

在本應用案例中，DB 類的服務元件可分為三種，分別是『CloudBankInc_DB』、『Google Invoice Cloud Database』與『Google Cloud Database』三種，分述如下：

1. 『CloudBankInc_DB』是進行銀行內部資料庫存取的服務元件，本案例的使用者設定為銀行，並以由銀行承做的發票貸款融資業務為應用範例，在情境設定中，銀行本身已經具有存有顧客資料的私有資料庫，這個資料庫掌握了顧客的公司名稱與統一編號，藉由這些關鍵的基本資訊，得以進一步掌握顧客的確切身份，這也是銀行不願意外包的關鍵資訊。於是本案例假定銀行本身的私有資料庫依舊存在於內部，但利用網路服務的方式，在服務平台上註冊為服務元件，可與其他服務元件商與服務套餐業者所提出的服務相互應用，至於其他服務所需的必要資料，則採取外包的方式，交給第三方雲端資料庫業者管理，並藉由銀行所授權的 Token 來進行存取，以確保資料的安全與其掌控能力。

銀行所持有的私人資料庫結構可參考下圖：

←T→			ID	NAME	NID
☐			6	yespasta	A1234567
☐			5	highman	N12345678
☐			7	smallwhite	J12345678
☐			8	may	T12345678
☐			9	oneway	E12345677
☐			10	Lee	E12342414
☐			11	Wang	Q1234567
☐			12	Tsai	i7652345678
☐			13	Tseng	t552345677
☐			14	ArTu	q652345678
☐			15	Yang	c12345678
☐			16	Chu-yi	e52523423
☐			17	highman junior	c652525224

圖 17：銀行內部資料結構

資料來源：本研究整理

2. 『Google Invoice Cloud Database』與『Google Cloud Database』是利用 Google 所提供的雲端服務 - Google Docs 而成的雲端資料庫，本研究利用 Google 所提供的雲端服務，來模擬未來雲端資料庫業者代管資料後的運作情況，簡單來說，我們可以將這項雲端資料庫視為傳統的關連式資料庫使用，也就是進行一般常見的資料庫操作，例如新增、查詢等，在這上面的資料都憑藉於 Google 的雲端運算基礎架構，享有高效能、高可靠度的服務優勢，以目前的情況而言，使用者可以用相當低廉的成本取得資料儲存空間。本案例將顧客的收入與工作類型放在『Google Cloud Database』中，而擔保發票資料則放在『Google Invoice Cloud Database』中，雲端資料庫的運作情況可見下圖：

Cloud DataCenter #1 : Sheet1			
Timestamp	TOKEN	INCOME	JOB
4/29/2009 17:02:31	a283573407a945b4b	50000	Research
4/29/2009 17:06:18	7367cc4cee061a47E	1000000	Argiculture
4/29/2009 17:08:46	ab0f5410731093129f	9000000	Service
4/29/2009 17:11:04	dd2354c5f62b84a9e0	5000000	Dealer
4/30/2009 11:19:12	8b933c3f5ab7b0471f	800000	Public
4/30/2009 13:12:06	0b405965f905e1efb2	5123432	Public
4/30/2009 13:12:35	86e0eed77dd654094	1000000	Dealer
4/30/2009 13:13:07	4f18c3a510f388c3f8f	952343	Dealer
4/30/2009 13:13:52	2eac2b02113f1c32d1	1234000	Service
4/30/2009 13:14:24	d2366929dd14c19cb	9123432	Argiculture
4/30/2009 13:15:15	ec4ce68a1b80b891f5	9999999	Research
5/2/2009 17:26:25	ca752a94cf6a0ba850	800000	Argiculture
5/8/2009 17:25:12	eb58133678168b1f37	666000	Dealer

圖 18：雲端資料庫示意圖

資料來源：本研究整理

Cloud DataCenter #2 : Sheet1					
Timestamp	TOKEN	InvoiceAmount	InvoiceType	InvoiceAssign	InvoiceDate
4/29/2009 17:11:06	dd2354c5f62b84a9e0	9999999	sell	HiBuilding	2/25/2009
4/30/2009 13:13:53	2eac2b02113f1c32d1	5645645	buy	McDonalxx	4/23/2009
4/30/2009 11:19:14	8b933c3f5ab7b0471f	777777	sell	MyArgiculture	3/10/2009
4/30/2009 13:13:08	4f18c3a510f388c3f8f	564564	sell	NIKKE	4/15/2009
4/29/2009 17:02:32	a283573407a945b4b	30000	buy	Orange	1/2/2009
4/30/2009 13:12:35	86e0eed77dd654094	43656785	buy	Orange	3/31/2009
4/30/2009 13:15:15	ec4ce68a1b80b891f5	33454565	sell	Orange	5/23/2009
4/30/2009 13:14:25	d2366929dd14c19cb	5245425	buy	TaiwanSemi	5/14/2009
5/2/2009 17:26:26	ca752a94cf6a0ba850	867445	sell	TaiwanSemi	5/28/2009
4/29/2009 17:06:19	7367cc4cee061a47E	5000000	sell	XMall	1/29/2009
4/29/2009 17:08:47	ab0f5410731093129f	800000	buy	XMall	2/20/2009
4/30/2009 13:12:07	0b405965f905e1efb2	634635	sell	Yahxx	3/21/2009
5/8/2009 17:25:12	eb58133678168b1f37	200000	sell	McDonalxx	5/30/2009

圖 19：雲端發票資料庫示意圖

資料來源：本研究整理

以上服務元件資訊都會被服務平台所記錄，這些已註冊的資訊，在未來服務套餐業者要進行服務串接時，以上服務都會成為候選服務，也就是說得以透過本服務平台來與服務需求方與服務套餐業者兩者間進行互動，下圖是服務元件管理介面的示意圖：

Component Manager

Add New Component

Edit	Del	Name	Type	Function	Input	Output	Note
▶	X	ServiceAccounting	AP	WS: ServiceAccounting	AccountingData	ExecutionResult	This is a ap for accounting all the action in this service center.
▶	X	CreditChecker_C	AP	WS: CheckCredit_C	Userdata	EstimateResult	This ap is a cloud-based credit checking application.
▶	X	CreditChecker_B	AP	WS: CheckCredit_B	Userdata	EstimateResult	This ap is a cloud-based credit checking application.
▶	X	CreditChecker_A	AP	WS: CheckCredit_A	Userdata	EstimateResult	This ap is a cloud-based credit checking application.
▶	X	Google Invoice Cloud Database	DB	WS: GoogleInvoiceGDB	Token	InvoiceDataSet	This is the function make Google Docs as a cloud invoice database.
▶	X	Google Cloud Database	DB	WS: GoogleGDB	Token	DataSet	This is the function make Google Docs as a cloud database.
▶	X	CloudBankInc_DB	DB	WS: CloudBankInc	Token	DataSet	This is a private database of CloudBankInc.
▶	X	BankingDashBoard	Interface	Interface: BankingDashBoard	ServiceRequest	ServiceInterface	This is a dashboard interface for cloud-based application.

圖 20：服務元件註冊清單畫面

資料來源：本研究整理

(三) 使用者介面 (Interface)

著重使用者介面與呈現方式者稱之。在軟體開發的過程中，使用者介面往往是最不受注意，但卻也是最重要的環節，因為使用者介面直接與人進行互動，其所設計的好壞所造成的影響是直接且關鍵的，尤其是在講求服務科學與服務行銷的現在，使用者介面的設計與安排更是一門專業的學問，程式開發人員往往難以兼顧漂亮的美工與方便的操作介面，於是衍生出相關的美工、介面設計等專業需求；在系統發展架構上也藉由如 MVC 等架構來讓美工人員能夠直接參與專案的開發工作，本類的服務元件就是在解決有關畫面設計與操作介面的設計，讓這項重要卻又跟服務流程本身較不相關的功能得以外包給服務元件商來負責。

本服務元件是『BankingDashBoard』，採用 Google 的小工具(Gadget)與 iGoogle 個人化入口網站，當作雲端服務所需的人機介面。小工具包含易於建構、多平台運行與結合現有服務的優勢，且採用 JavaScript 函式庫，讓開發工作的負擔能夠減輕，有助於快速、低成本的建構出可用的服務；小工具也具有可在多個平台上運行的能力，例如 iGoogle 入口網站、Google Sites 與一般網頁等，都可以輕易的嵌入已完成的小工具；而 iGoogle 個人化入口網站則可以收集來自不同來源的資訊，集中整合到一個頁面中，方便使用者編排屬於自己的入口網站。原先需要花費許多開發成本卻又跟專案目標無關的使用者互動介面，可以透過第三方平台來輕易達成。下圖是採用小工具與 iGoogle 個人化入口網站後的呈現結果：

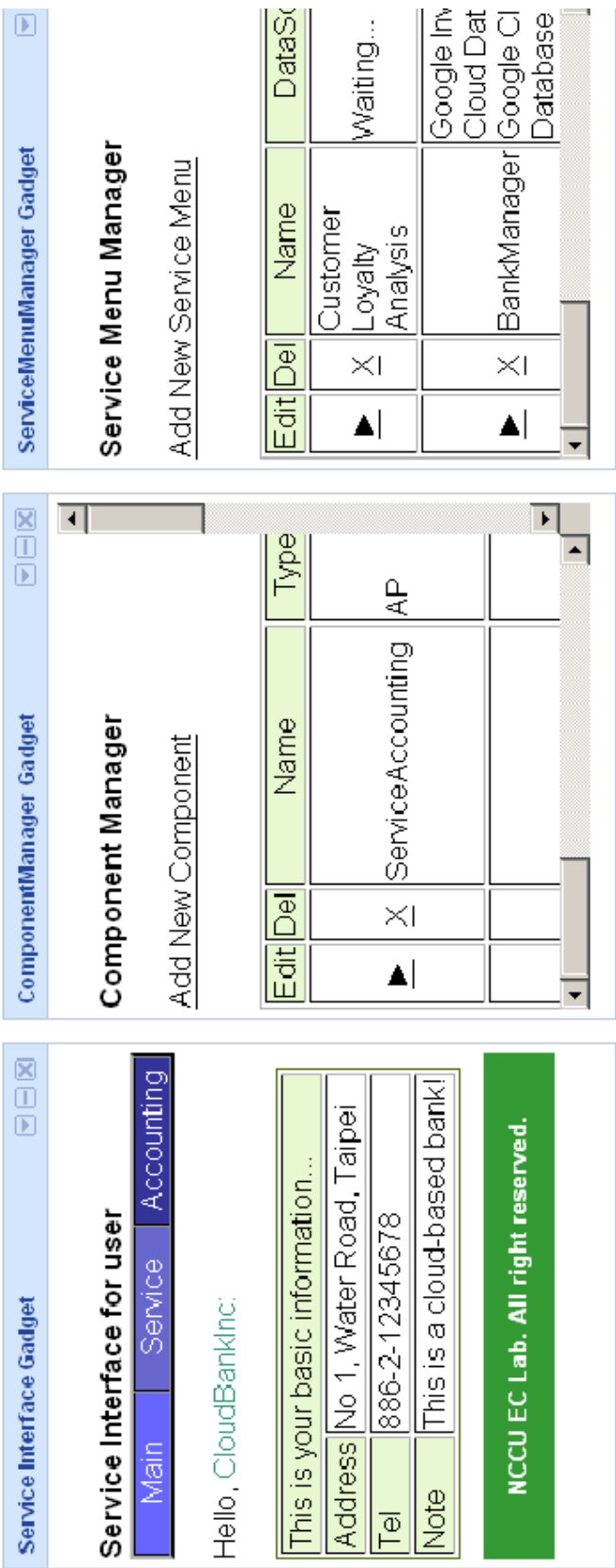


圖 21：雲端服務平台使用者介面示意圖

資料來源：本研究整理

三、服務套餐業者

當雲端服務平台累積了許多服務元件，代表這些服務元件應該能夠利用服務串接的方式，組合成各式服務供服務需求方所用，這時衍生出一個瞭解專業知識 (Domain Knowledge)，又知道如何利用雲端資源來完成服務的服務套餐業者。其負責工作就是專門與企業溝通服務需求，並依照相對的服務需求組合服務元件與定義服務流程。以價值鍊上的劃分而言，服務元件商專注於單一、專業的服務內涵，以本應用範例為例，信用評等計算元件由不同的服務供應商來提供，其負責的內涵只是一個方程式的計算工作，但是一個完整的信評流程，包含收件、身份確認、信用評等公示計算與擔保品鑑價等流程，這些工作經過拆解並交由不同服務供應商負責，服務元件商專注於本身的優勢，僅切出價值鍊中的一層，而服務套餐業者則負責整合這些元件，各司其職以完成完整的服務，讓雲端服務平台能夠讓更多的業者投入，並且構成一種良性的競爭狀態，服務需求方將可以在這樣的良性競爭下達成最佳的服務效率。

本研究之服務平台，透過建立符合 AAA 協定的雲端平台管理機制，紀錄服務供需雙方與服務套餐業者之間的交易細節，並依照這些統計記錄進行拆帳與成本評估，這樣的營運模式，可以讓服務需求方能夠以更為低廉的價格，取得強大且符合需求的服務，而服務元件商可以專注於每一層的價值鍊切分，將整體服務流程中的一部份做到最好，換句話說，這也就是服務元件商所必須專注的『核心競爭力』，最後透過服務套餐業者的分析與整合，讓整個服務消費得以順利進行，我們可以觀察到以上三個角色分別站在不同的角度利用本雲端服務平台，每個角度都創造出新的可能與新的價值，而這些新的可能與價值導出了未來資訊服務產業的改變契機，如何瞭解本身價值所在，應該是未來資服供需雙方所必須思考的課題。

服務套餐業者瞭解如何完成企業所需的相關服務，同時知道該如何結合所需的資源，為多方創造價值，依照以上需求，服務套餐業者至少需要具備瞭解商用流程管理(Business Process Management)與 IT 資源整合的能力。在本服務平台中，為 IT 資源整合提供了方便的基礎架構，讓服務套餐業者得以輕鬆展現商用流程管理的特點，當經過分析本案例後，案例的流程圖可參考圖 15：金融發票融資案例服務流程概念圖，從概念圖中可以知道如何完成一項信評作業，而為了充分發揮雲端服務平台與服務導向架構的優勢，服務套餐業者必須要依照流程的特性，將可分散處理與利用外部資源的特性納入考量，讓整體流程不單單只是一項單純的商用流程，也包含了以資訊技術為出發點的演算法最佳化。服務套餐業者所構思出的服務套餐，將會透過 BPEL 的形式輸入到服務平台中，並且註冊成為候選服務套餐，供服務需求方使用，下圖是 BPEL 描述語言的片段，依照本案例所建構出的完整 BPEL 描述，請參考附錄 A。

```

<?xml version="1.0" encoding="UTF-8" ?>
- <process name="ncname" targetNamespace="uri" queryLanguage="anyURI" expressionLanguage="anyURI"
  suppressJoinFailure="yes|no" enableInstanceCompensation="yes|no" abstractProcess="yes|no"
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
+ <partnerLinks>
+ <variables>
+ <message name="Input">
+ <message name="CreditCheckerRequest">
+ <message name="CreditCheckerResponse">
+ <message name="doGetCloudDataRequest">
+ <message name="doGetCloudDataResponse">
+ <message name="doGetInvoiceCloudDataRequest">
+ <message name="doGetInvoiceCloudDataResponse">
+ <message name="doEvaluateQuotaRequest">
+ <message name="doEvaluateQuotaResponse">
+ <message name="doCheckCollateralRequest">
+ <message name="doCheckCollateralResponse">
+ <flow name="Flow">
+ <scope>
+ </process>

```

圖 22：服務套餐 BPEL 描述片段

資料來源：本研究整理

當完成服務套餐的設計後，可透過服務平台的服務套餐管理頁面進行註冊，這些被註冊完畢的服務套餐，就可以直接被使用者所選用，並且透過本服務平台的統計機制開始記錄使用情況，此外，有些服務套餐可能已有使用者需求，但是尚未完成 BPEL 描述的，也可以進行註冊，只是會顯示出尚不可使用的狀態，直到完成 BPEL 描述檔後，才可投入實際的營運，註冊完的清單介面資訊如下圖：

Service Menu Manager

Add New Service Menu

Edit	Del	Name	DataSource	AP	Interface	Price	Status
		Customer Loyalty Analysis	Waiting...	Waiting...	Waiting...	10	Waiting...
		BankManager	Google Invoice Cloud Database, Google Cloud Database, CloudBankInc_DB	ServiceAccounting	BankingDashBoard	1	Waiting...
		CheckCredit	Google Invoice Cloud Database, Google Cloud Database, CloudBankInc_DB	ServiceAccounting, CreditChecker_C, CreditChecker_	BankingDashBoard	5	Available...

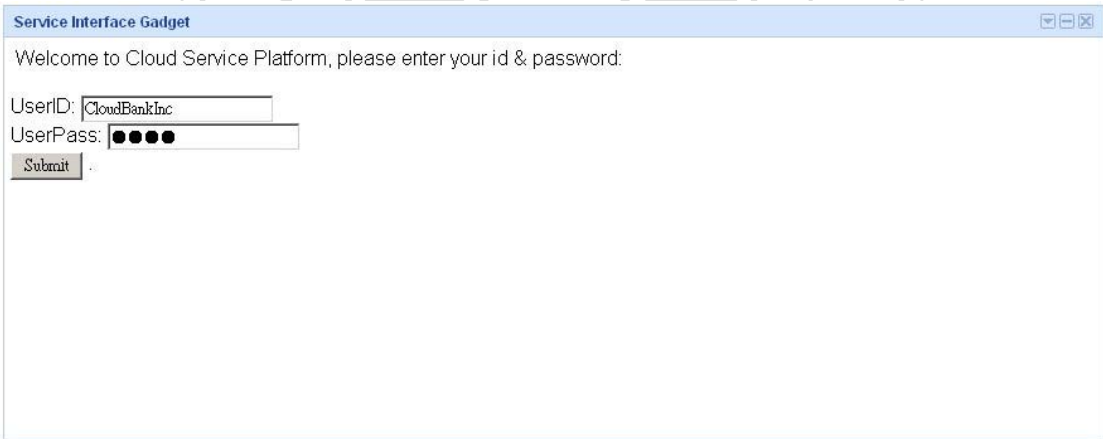
圖 23：服務套餐註冊清單畫面

資料來源：本研究整理

第三節 服務平台展示

本案例的服務需求方為金融業，在此虛擬了一間名為『CloudBank』的銀行作為使用者，而在服務平台上，已經註冊有服務平台統計元件與三個信用評等計算元件，另外註冊有兩個雲端資料庫與銀行內部的私有資料庫，最後註冊有一個使用者介面元件，而服務套餐部分則是一個銀行管理服務套餐與符合新巴賽爾協定之信用評等服務套餐等兩項。

為說明本服務平台運作的情況，以下將由實際的系統操作畫面來說明本應用案例，首先在服務介面輸入帳號密碼，登入雲端服務平台。服務平台依照輸入的帳號密碼的不同，會決定使用者的身份、可使用服務與相關統計資訊等，值得注意的是，現在所看到的使用者介面是透過 Google 所提供小工具所完成的，這些小工具可以嵌在 Google Sites 或是 iGoogle 上面，利用 Google 的協同合作平台達成網頁入口的概念，簡單、功能強大且成本低廉。



Service Interface Gadget

Welcome to Cloud Service Platform, please enter your id & password:

UserID: CloudBankInc

UserPass: ●●●●

Submit

圖 24：服務平台登入畫面

資料來源：本研究整理

在這個主畫面中，我們可以看到上方一共有三個標籤，分別是 Main、Service 與 Accounting，這三個標籤分別代表使用者基本資訊、已購買服務與服務統計資訊，完成身份認證後，使用者可以看到自己的基本資訊，本範例中登入的身份是『CloudBankInc』，使用者可以透過這個畫面進行各式服務的執行、訂閱與統計資訊查詢等功能，可說是使用者進入本服務平台的入口。

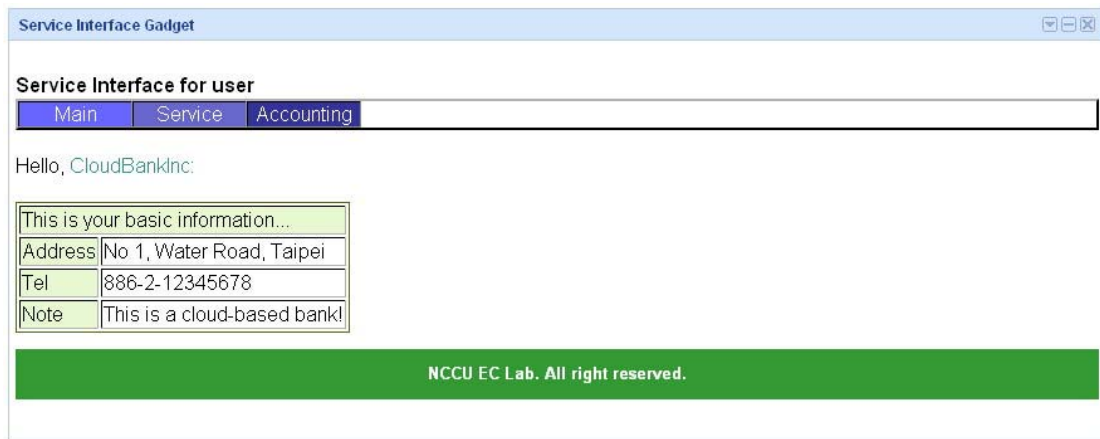


圖 25：服務平台主畫面

資料來源：本研究整理

接著選擇上方的 Service 標籤進入服務選單，進入服務選單後，可以看到已購買的服務套餐清單，在下圖中可以看到已購買的有『CheckCredit』與『BankManager』兩項服務套餐，每次使用的費用分別是五元與一元。此時使用者可以開始使用任一項已購買的服務，值得注意的是，當使用者按下『Run』的按鈕，也就代表授權給服務元件商存取必須的資料。

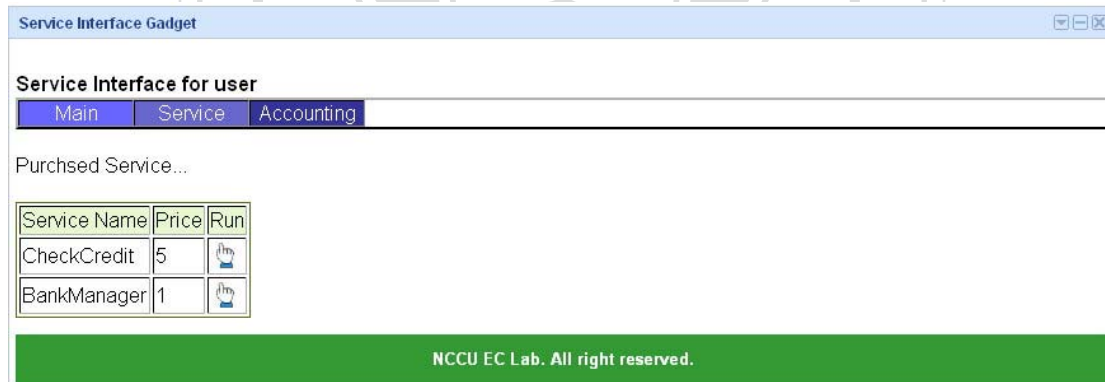
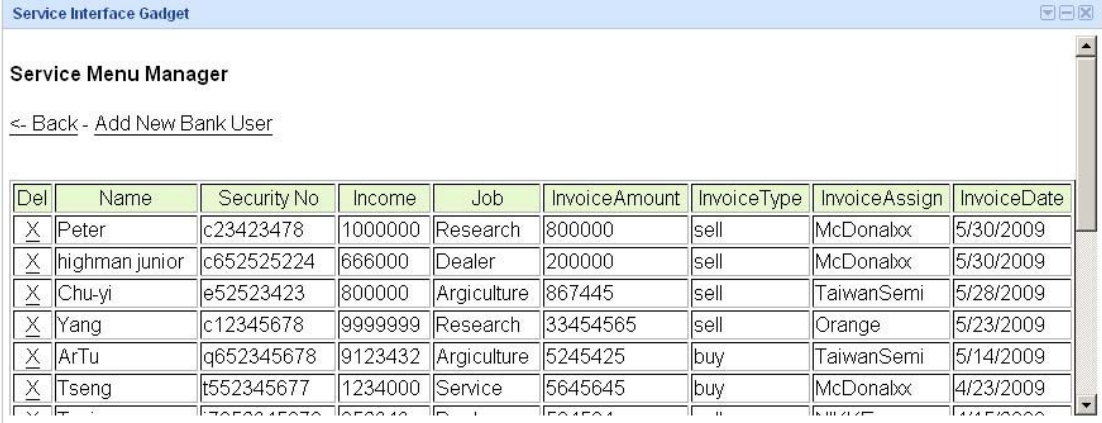


圖 26：服務平台已購買服務選單畫面

資料來源：本研究整理

『CheckCredit』服務套餐是由服務套餐業者，組合服務平台中的各式服務資源而成的信評服務套餐，這個套餐符合新巴賽爾協定的要求，並且依照個別銀行對於信評的要求所製作而成。值得注意的是，使用者並不直接面對銀行監理協定的內涵，而是將這個工作交由服務套餐業者去完成，也就是說，當未來有新的監管機制或是信評需要其他修正，可以直接透過服務套餐業者完成修改，使用者本身並不需要作太大的改變，充分發揮軟體即服務的精神。

因為『CheckCredit』服務套餐會自動算出資料庫中所有顧客的可貸金額與違約機率，故在使用『CheckCredit』服務套餐前，必須先藉由『BankManager』服務套餐新增與管理顧客的資料。下圖是銀行管理顧客資料的介面，我們先選擇『Add New Bank User』選項。



Service Interface Gadget

Service Menu Manager

<- Back - Add New Bank User

Del	Name	Security No	Income	Job	InvoiceAmount	InvoiceType	InvoiceAssign	InvoiceDate
X	Peter	c23423478	1000000	Research	800000	sell	McDonabx	5/30/2009
X	highman junior	c652525224	666000	Dealer	200000	sell	McDonabx	5/30/2009
X	Chu-yi	e52523423	800000	Argiculture	867445	sell	TaiwanSemi	5/28/2009
X	Yang	c12345678	9999999	Research	33454565	sell	Orange	5/23/2009
X	ArTu	q652345678	9123432	Argiculture	5245425	buy	TaiwanSemi	5/14/2009
X	Tseng	t552345677	1234000	Service	5645645	buy	McDonabx	4/23/2009

圖 27：『BankManager』服務主畫面

資料來源：本研究整理

『BankManager』服務套餐讓銀行可以管理由兩個雲端資料庫與銀行本身私有資料庫等三者所組成的顧客資料庫，這些資料分散於三個地方，透過此服務套餐進行整合與管理的工作。當使用者進行新增顧客的動作時，網頁表單中的資料將分別存放到上述的三個地方，唯有掌握銀行內部的私有資料庫，才可以組合出完整的顧客資料，

以下是新增顧客資料的畫面，在輸入完必填的欄位後，按下 Submit 按鈕，這時公司名稱與統一編號將會存入銀行內部的資料庫，而收入、工作類型與發票資訊等將分別存入雲端資料庫中。換言之，這項服務所涵蓋的服務元件有三個，第一是銀行內部的資料庫，第二是客戶資訊雲端資料庫與最後的發票資訊雲端資料庫，都透過這個新增頁面完成。

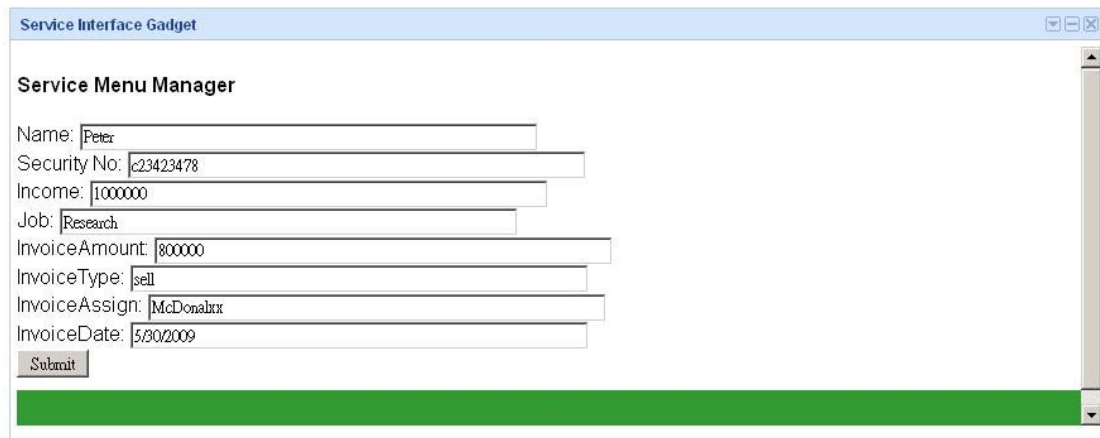


圖 28：『BankManager』服務新增資料選單

資料來源：本研究整理

在新增完所有的顧客資料後，可開始授權執行『CheckCredit』服務，當使用者開始執行本服務後，其會將所有的客戶資料整理成一個佇列(Queue)，也就是依照 MapReduce 設計架構的『Input Task』階段，將本佇列視為等待執行的任務集合；接著進入『Map Phase』，在這個階段服務平台將會依照三個 AP 的負載情況決定該將如何指派任務，以本案例而言，第一個任務將交由應用元件 1 執行，而第二、第三分別由應用元件 2 與應用元件 3 執行，而第四個任務將視三個應用元件中是否已有閒置的，若有則指派之，若無則指派給應用元件 1，以此類推；每一個任務執行完的結果將會放置在『Intermediate』暫存區域，等待所有的任務完成後，一併進入『Reduce Phase』階段；『Reduce Phase』將所有的任務執行結果，依照 Token 對應執行結果的方式排列，這個 Reduce 的動作也就是將各 AP 的執行結果合而為一；最後到『Output Phase』，將執行的結果回傳給使用者。

以下是『CheckCredit』服務執行的結果，可以看到執行結果的表格結合了三方資料庫的來源，包含銀行內部資料庫的公司名稱與統一編號，客戶雲端資料庫的收入、工作類型與雲端發票資料庫的發票資訊等，CheckCredit 服務套餐會藉由這些資料計算出信貸額度與違約機率，並且將結果回報給使用者。這次的服務執行涉及多方，包含三個 CheckCredit 服務元件、內部資料庫與兩個外部雲端資料庫與最後的介面呈現等。

Name	Scurity No	Token	Income	Jobs	InvoiceAmount	InvoiceType	InvoiceAssign	InvoiceDate	Credit	Default Prob.
oneway	E12345677	8b933c3f5ab7b0471c9f9d6c2a7faeab	800000	Public	777777	sell	MyArgiculture	3/10/2009	102888.5	0.4
yespasta	A1234567	7367cc4cee061a476290d18978830414	1000000	Argiculture	5000000	sell	XMall	1/29/2009	3500000	0.3
Yang	c12345678	ec4ce68a1b80b8915e2405ac71318fab	9999999	Research	33454565	sell	Orange	5/23/2009	10176364.45	0.06
Peter	c23423478	3f0f79b2d49f43519d9fd059b08781d5	1000000	Research	800000	sell	McDonabx	5/30/2009	380000	0.03
Tsai	t7652345678	4f18c3a510f388c3f85ab0dd72e03888	952343	Dealer	564564	sell	NIKKE	4/15/2009	751324.7	0.105
may	T12345678	dd2354c5f62b84a9e00eb12ef287587f	5000000	Dealer	9999999	sell	HiBuilding	2/25/2009	11499999.2	0.56
highman junior	c652525224	eb58133678168b1f371b6d6cd1f2c275	666000	Dealer	200000	sell	McDonabx	5/30/2009	486200	0.07
highman	N12345678	a283573407a945b4b0941c47b7c29bd7	50000	Research	30000	buy	Orange	1/2/2009	39900	0.06
Wang	Q1234567	86e0eed77dd6540948ecaa857da55735	1000000	Dealer	43656785	buy	Orange	3/31/2009	4192542.8	0.056
smallwhite	J12345678	ab0f54107310931299e394fc46f7635a	9000000	Service	800000	buy	XMall	2/20/2009	5880000	0.36
ArTu	q652345678	d2366929dd14c19cb3aaa378ecb8c351	9123432	Argiculture	5245425	buy	TaiwanSemi	5/14/2009	4614170.25	0.005
Chu-yi	e52523423	ca752a94cf6a0ba850ff76634ce3317	800000	Argiculture	867445	sell	TaiwanSemi	5/28/2009	408674.45	0.005
Lee	E12342414	0b405965f905e1efb2e92d69a3a5d762	5123432	Public	634635	sell	Yahxx	3/21/2009	1651263.9	0.054
Tseng	t552345677	2eac2b02113f1c32de851f85933cb7ec	1234000	Service	5645645	buy	McDonabx	4/23/2009	1304964.5	0.06

圖 29：『CheckCredit』服務執行結果

資料來源：本研究整理

下圖是跟隨『CheckCredit』服務而得到的運算資源利用情況，可以看到本次服務交由三個 AP 進行運算，總任務為 14 個，個別服務元件的運算時間如下圖所示，使用者可藉此知道服務執行的概況，依照之前的描述，我們可以知道待執行的任務將會依照應用元件的負載情況來指派，可以從下圖中得知，各應用元件被指派的情況，本案例執行結果是應用元件 1 負擔最重，而應用元件 2、應用元件 3 次之。

Computing Resource:

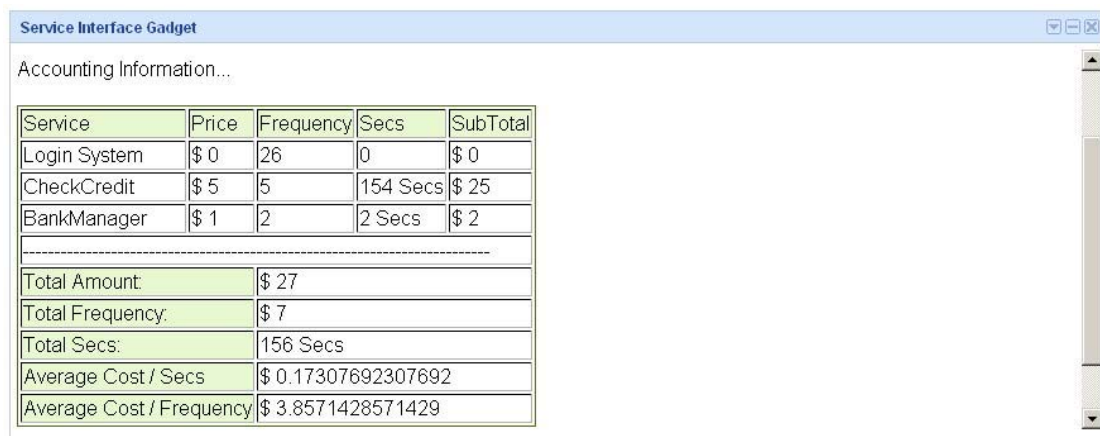
Item	Value
Task	14
Ap1	22 Sec
Ap2	5 Sec
Ap3	4 Sec

圖 30：『CheckCredit』服務運算資源利用表

資料來源：本研究整理

在執行完服務後，使用者回到主選單選擇『Accounting』標籤，下圖是 Accounting 標籤所提供的統計表格，在這張表格中顯示了使用的服務類別、價格、次數、時間與小計等資訊，最後也提供了總服務時間、累積服務金額與總服務時間等資訊，而平均每秒費用與平均每次費用，可以提供給使用者評估運算資源的運算成本，作為評估 IT 投資的比較數據。在有更多服務元件可選擇的情況下，可藉如競價等方式來用更低成本取得運算資源。

以上資訊充分的呈現出雲端資源的利用情況，讓每一次服務的使用與負載都能夠清楚的被計算，若依照這樣的模式持續發展，未來的服務需求者，將可依照本身的特性與需求，擁有不同程度的服務選擇，例如可能有的服務套餐符合最新版本的協定，或是有的服務套餐專門組合價格最為低廉的服務元件，讓整體成本最低，又或者有可能營造出消費者品牌形象，讓服務套餐也可能出現所謂的明星商品，以上種種模式，都是未來服務平台逐漸發展後，有可能看到的景象。



Service	Price	Frequency	Secs	SubTotal
Login System	\$ 0	26	0	\$ 0
CheckCredit	\$ 5	5	154 Secs	\$ 25
BankManager	\$ 1	2	2 Secs	\$ 2

Total Amount:	\$ 27			
Total Frequency:	\$ 7			
Total Secs:	156 Secs			
Average Cost / Secs	\$ 0.17307692307692			
Average Cost / Frequency	\$ 3.8571428571429			

圖 31：服務平台統計資訊畫面

資料來源：本研究整理

第六章 結論與未來展望

本章根據研究過程與結果，歸納整理出本研究的結論，並提出對後續未來研究方向的建議，以作為後續研究的參考。分別說明如下：

第一節 結論與建議

為解決雲端運算所衍生出內外部資源有效配置的問題，與企業導入上的資安疑慮，本研究結合雲端運算與服務導向架構，發展出雲端運算服務導向架構運作機制，包含服務串接、資源配置、資料控管與管理機制，並依照這些運算機制建構出以服務介面層、中心控管層、服務供應層與基礎架構層組成的『雲端運算服務導向架構平台』。

一、雲端運算服務導向架構運作機制

（一）服務串接機制

將服務使用者、服務元件商與服務套餐業者三方連接在一起，讓已開發、私有的各式服務資源得以註冊到本服務平台，這些服務透過 BPEL 描述語言整合成服務套餐，創造出有利於三方的商業運作機制。

（二）資源配置機制

具備有負載平衡、平行運算與容錯機制，得以發揮雲端運算強大的運算架構，本研究所開發的服務監控機制能有效分配運算資源，並且藉由容錯機制提升服務的可靠度。

（三）資料控管機制

解決企業對於資料外包的許多疑慮，在資料儲存外包的部分，利用部分授權的方式讓資料以片段的方式存在，避免有心人士竊取的風險；對於資料運算的安全性，讓服務供應方只能取得經授權的必須資料，關鍵資訊僅掌握在資料擁有人手中，以確保資料安全性。

（四）管理機制

實作了 AAA 協定，分別是：認證 (Authentication)、授權(Authorization)與統計(Accounting)。認證機制決定服務使用者的身份與服務範圍；授權機制讓使用者得以妥善的授權服務需求方存取資料的權限；而服務使用者在服務平台中的所有行為將會被有效紀錄，作為收費與提供 IT 投資評估的有效資訊。

二、雲端運算服務導向架構平台

憑藉以上運作機制，本研究建構出『雲端運算服務導向架構平台』，此平台由四層架構所構築而成，分述如下：

(一) 應用介面層

專注於人機介面的呈現，以 Gadget 負責畫面呈現，將版面與程式邏輯分開，充分利用開放介面的優勢，降低服務介面開發的負擔，讓開發者能專注於關鍵問題的解決。

(二) 中心控管層

係基於服務導向架構之設計，能連接服務供應方與服務需求方，將服務需求轉送到服務供應層，並依照服務邏輯定出服務路由，可使用外部雲端運算與資料儲存之資源，並達成服務流程控管的需求；在完成服務的執行流程後，將服務供應層執行完畢的服務結果回傳給應用介面層。

(三) 中心控管層

具備服務導向架構的優點，並能善加利用雲端運算的優勢。服務供應層結合各式應用服務與資料來源，透過內建函式庫與透過網路服務與第三方伙伴溝通，包含資料中心與服務提供者。服務供應層藉由將運算與資料分離的方式，讓服務需求者保持對資料的控管權力，以解決企業導入雲端架構時對於資安的疑慮；在匯集越多服務資源後，能產生更大的平台加值效益。

(四) 基礎架構層

可充分利用各雲端業者所提出的服務架構、運算以及儲存資源。本研究係採用 Google 應用服務環境，透過 Gadget 連接本平台所整合的應用服務，並部署於 Google 的雲端運算環境之中。基礎架構層未來可導入更多雲端平台業者，依照個別平台的優勢與特性，充分發揮雲端運算架構的效益。

三、研究建議

基於以上的研究成果，我們歸納出幾點建議：

應用本研究所提出的雲端運算服務導向架構平台，讓服務需求方得以如水、電般取得己身所需的服務資源，資訊服務本身已經如同基礎設施一般普遍，這代表服務需求方必須思考如何專注核心競爭力，將次要的 IT 服務與資料管理在可靠、可信賴的基礎上，外包給第三方業者。雲端運算架構具有網格運算、公用運算與軟體即服務的優點，搭配服務導向架構的特性，讓企業可依照策略目標，在預算不足的情況下取得資訊服務，並可有彈性的調整使用量與投資成本，減少閒置資源的浪費，在強調降低成本以提升競爭力的思維下，這樣的系統架構將會是一項理想的解決方案，也是未來企業營運必須思考改變的方向。

隨著雲端服務平台的興起，可預見將發展出更多雲端運算資源，以往需要軟硬統包的集中開發模式將不再適用，取而代之的，會是更多專攻企業價值鍊中一環的資服業者，這些業者將可依照本身的強項與對於市場的嗅覺，開發出各式服務並切入市場。透過本研究所提出的服務平台，能發揮出服務聚集效益，最終形成服務市集，並改變以往傳統的服務供需模式，這是在大型雲端業者逐漸構築起基礎設施的競爭障礙後，中小型服務供應者能夠發揮且生存的營運模式。

在基礎設施與各式服務都備齊的情況下，瞭解企業商用流程的服務套餐業者，也能輕鬆的以聚合各方資源的方式，為自己創造出價值。在資訊服務逐漸朝向專業分工的模式後，資訊服務依舊能有提升的機會，例如服務科學、行銷、美學與文化產業等，都可能是未來資訊服務再往上提升的關鍵與能量。透過本服務平台，可以讓掌握如何做(How to)的規劃人員得以利用強大的基礎設施與完善的各式服務，做出更多加值應用，無須擔心技術實行上的細節，而能將資訊服務與其他產業作更緊密的結合，讓 IT 科技真正以需求的角度出發，創造出更吸引人的服務能量與應用。

第二節 未來研究方向

一、網路資安問題

因為雲端服務大多透過網頁的方式進行，一些常見的網路攻擊，如 DoS 攻擊與跨站腳本攻擊等，都可能對雲端使用者造成損失與傷害，再加上雲端運算構築出的龐大基礎架構，讓安全性問題更複雜且不容易掌握，於是如何在雲端服務中加入更多的防範機制，例如更完善的身份認證機制、惡意攻擊防範與偵測與程式安全滲透測試等，都是未來可以加強的研究議題。

二、服務水準維持

雲端服務多屬外包服務，關於外包服務的設計瑕疵與喪失控制權的疑慮，都是企業在決定時的重要考量依據，有學者建議透過簽訂 SLA(Service Level Agreement)的方式來管控外包服務品質，另外，本研究認為透過適當的程式碼審視(Code Review)的方式，以公正第三方的角度，來對服務品質做出類似『信用評等』的評鑑機制，可能亦是一種解決方案，在未來的研究之中，可以針對以上議題加以討論。

三、商業營運細節

在雲端運算逐漸朝向以市場導向(Market-Oriented)，對於雲端服務的監控、計價、拆帳與合約規範等相關營運內容，都有必要再提出更詳盡的營運細節，在這個部分值得相關財務、會計與統計研究人員投入研究，在研發出更完善的營運細節與數據後，可幫助企業擬定出更詳盡的 IT 投資策略，讓雲端運算的服務概念能夠更臻成熟與完善。

四、雲端架構導入分析

以管理面的角度而言，應用何種標準決定是否導入雲端運算？為充分發揮雲端運算架構之優勢，本研究認為應對各式應用進行分析，例如針對服務的機密性與安全性要求等方面，評估雲端運算架構的適用性，這些分析結果將可做為未來企業在導入雲端運算架構時的參考依據。

參考文獻

1. Amazon.com. (2007). "Amazon.com Announces Third Quarter Sales up 41% Year over Year - Raises Financial Guidance - Expects Record Holiday Season." from <http://phx.corporate-ir.net/phoenix.zhtml?c=176060&p=irol-newsArticle&ID=1066358&highlight=Amazon%20Web%20Services>.
2. Angelis, G., S. Gritzalis, et al. (2004). "Mechanisms for controlling access in the global grid environment." *Internet Research* **14**(5): 347.
3. Apache. (2008). "Hadoop Official Site." from <http://hadoop.apache.org/core/>.
4. Bieberstein, N., S. Bose, et al. (2005). "Service-Oriented Architecture Compass: Business Value, Planning, and Enterprise Roadmap." *IBM Press Publishing*.
5. Boden, T. (2004). "The grid enterprise — structuring the agile business of the future." *BT Technology Journal* **22**(1): 107-117.
6. Boss, G., P. Malladi, et al. (2007). "Cloud Computing." from http://download.boulder.ibm.com/ibmdl/pub/software/dw/wes/hipods/Cloud_computing_wp_final_8Oct.pdf.
7. Brereton, P., D. Budgen, et al. (1999). "The Future Of Software." *Communications of the ACM* **42**(12): 78-85.
8. Buyya, R., C. S. Yeo, et al. (2008). "Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities." *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications*.
9. Chong, F. (2006). "Saas Application Architecture Case Study:Remend." from <http://blogs.msdn.com/fred%5Fchong/>.
10. Conlon, S. J., J. G.Hale, et al. (2008). "Information Extraction Agents For Service-Oriented Architecture using Web Service Systems: A Framework." *Journal of Computer Information Systems* **48**(3).
11. Davis, N., G. Demetriou, et al. (2006). "Web Service Architectures for Text Mining: An Exploration of the Issues via an E-Science Demonstrator." *International Journal of Web Services Research* **3**(4): 95-114.
12. Dean, J. and S. Ghemawat (2004). "MapReduce: Simplified Data Processing on Large Clusters." *ACM OSDI*.
13. Denne, M. (2007). "Pricing Utility Computing Services." *International Journal of Web Services Research* **4**(2).
14. Eilam, T., K. Appleby, et al. (2004). "Using a utility computing framework to develop utility systems." *IBM Systems Journal* **43**(1): 97.
15. Erdogmus, H. (2008). "So Many Languages, So Little Time." *IEEE Software* **25**(1): 4-7.

16. Feuerlicht, G. (2006). "Enterprise SOA: What are the Benefits and Challenges?" Systems Integration.
17. Gartner, I. (2008). "Gartner Says Contrasting Views on Cloud Computing Are Creating Confusion." from <http://gartner.com/it/page.jsp?id=766215>.
18. Google.com. (2007). "Google Apps." from <http://www.google.com/apps/intl/en/business/index.html>.
19. Google.com. (2008). "What are Gadgets?" from <http://code.google.com/apis/gadgets/>.
20. Gray, P. (2007). "SOA - Moving From Doing it Yourself to Getting Others to Do it for You." Information Systems Management **24**(1).
21. Group, W. C. W. (2004). "Web Services Architecture." from <http://www.w3.org/TR/ws-arch/#whatis>.
22. Heiser, J. and M. Nicolett. (2008). "Assessing the Security Risks of Cloud Computing ", from <http://www.gartner.com/DisplayDocument?id=685308>.
23. Huang, M. F. (2006). A Study of Multi-Business Service Oriented Architecture. Managerment Inforamtion System. Taipei, National Chengchi University. **Master**.
24. Khalaf, R., A. Keller, et al. (2006). "Business processes for Web Services: Principles and applications." IBM Systems Journal **45**(2): 425.
25. Louridas, P. (2008). "Orchestrating Web Services with BPEL." IEEE Software **25**(2): 85.
26. Mansfield-Devine, S. (2008). "Danger in the clouds " Network Security **2008**(12): 9-11.
27. McCoy, D. W. and Y. V. Natis. (2003). "Service-Oriented Architecture: Mainstream Straight Ahead." Retrieved August, 2008, from <http://www.gartner.com/pages/story.php.id.3586.s.8.jsp>.
28. McGovern, J., S. Tyagi, et al. (2003). "Java Web Services Architecture." 35-63.
29. MicrosoftCorporation. (2008). "What is the Azure Services Platform?" from <http://www.microsoft.com/azure/whatisazure.mspx#Whatis>.
30. Moorsel., A. P. A. v. (2005). "Grid, Management and Self-Management." The Computer Journal **48**(3): 325.
31. Morrison, S. (Aug 20, 2008.). Cloud Computing' Makes Gains. Wall Street Journal.
32. Natis, Y. V. (2003). "Service-Oriented Architecture Scenario." Retrieved August, 2008, from http://www.gartner.com/DisplayDocument?doc_cd=114358.
33. Pallos, M. (2001). "Service-Oriented Architecture: A Primer." eAI Journal.
34. Reenskaug, T. (2003). "The Model-View-Controller (MVC)Its Past and Present ", from http://folk.uio.no/trygver/2003/javazone-jaoo/MVC_pattern.pdf.
35. Ross, J. W. and G. Westerman. (2004). "Preparing for utility computing: The role

- of IT architecture and relationship management." IBM Systems Journal **43**(1): 5.
36. Salesforce.com. (2007). "Salesforce.com and Google Form Strategic Global Alliance." from
<http://www.salesforce.com/company/news-press/press-releases/2007/06/070605.jsp>.
37. Stallman. (2008). "Stallman: Cloud computing is 'stupidity'." The Guardian
38. Stamford. (2008). "Gartner Says Cloud Computing Will Be as Influential as E-business." Retrieved August, 2008, from
<http://www.gartner.com/it/page.jsp?id=707508>.
39. W3C. (2008). "Web Services Activity." from <http://www.w3.org/2002/ws/>.
40. Yan, N., D. Leip, et al. (2005). "The use of open-source software in the IBM corporate portal." IBM Systems Journal **44**(2): 419-425.
41. Zhang, L.-J. and M. Jeckle (2004). "Convergence of Web Services and Grid Computing." International Journal of Web Services Research **1**(3).



附錄 A：金融發票融資應用案例 BPEL 描述

```
<?xml version="1.0" encoding="UTF-8" ?>
- <process name="ncname" targetNamespace="uri" queryLanguage="anyURI"
  expressionLanguage="anyURI" suppressJoinFailure="yes|no"
  enableInstanceCompensation="yes|no" abstractProcess="yes|no"
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
- <partnerLinks>
  <partnerLink myRole="CreditCkeckerProcessRole" name="ProcessStarter"
    partnerLink-Type="wsdl0:CreditCkeckerProcess" />
  <partnerLink name="GetCloudDataService"
    partnerLinkType="wsdl1:GetCloudDataPartnerPLT"
    partnerRole="GetCloudDataRole" />
  <partnerLink name="GetInvoiceCloudDataService"
    partnerLinkType="wsdl2:GetInvoiceCloudDataPartnerPLT"
    partnerRole="GetInvoiceCloudDataRole" />
  <partnerLink name="CheckeCreditService"
    partnerLinkType="wsdl3:CheckeCreditPartnerPLT"
    partnerRole="CheckeCreditRole" />
  <partnerLink name="EvaluateQuotaService"
    partnerLinkType="wsdl4:EvaluateQuotaPartnerPLT"
    partnerRole="EvaluateQuotaRole" />
  <partnerLink name="CheckCollateralService"
    partnerLinkType="wsdl5:CheckCollateralPartnerPLT"
    partnerRole="CheckCollateralRole" />
</partnerLinks>
- <variables>
  <variable messageType="wsdl0:Input" name="input" />
  <variable messageType="wsdl0:CreditCkeckerRequest"
    name="CreditCkeckerRequest" />
  <variable messageType="wsdl0:CreditCkeckerResponse"
    name="CreditCkeckerResponse" />
  <variable messageType="wsdl1:doGetCloudDataRequest"
    name="GetCloudDataRequest" />
  <variable messageType="wsdl1:doGetCloudDataResponse"
    name="GetCloudDataResponse" />
  <variable messageType="wsdl1:Exception" name="GetCloudDataFault" />
  <variable messageType="wsdl2:doGetInvoiceCloudDataRequest"
    name="GetInvoiceCloudDataRequest" />
  <variable messageType="wsdl2:doGetInvoiceCloudDataResponse"
    name="GetInvoiceCloudDataResponse" />
  <variable messageType="wsdl2:Exception"
    name="GetInvoiceCloudDataFault" />
  <variable messageType="wsdl3:doCheckeCreditRequest"
    name="GetCheckeCreditRequest" />
  <variable messageType="wsdl3:doCheckeCreditResponse"
    name="GetCheckeCreditResponse" />
  <variable messageType="wsdl3:Exception" name="CheckeCreditFault" />
  <variable messageType="wsdl4:doEvaluateQuotaRequest"
    name="EvaluateQuotaRequest" />
  <variable messageType="wsdl4:doEvaluateQuotaResponse"
    name="EvaluateQuotaResponse" />
  <variable messageType="wsdl4:Exception" name="EvaluateQuotaFault" />
  <variable messageType="wsdl5:doCheckCollateralRequest"
    name="CheckCollateralRequest" />
  <variable messageType="wsdl5:doCheckCollateralResponse"
    name="CheckCollateralResponse" />
  <variable messageType="wsdl5:Exception" name="CheckCollateralFault" />
</variables>
- <message name="Input">
```

```

    <part name="Username" type="xsd:string" />
    <part name="Token" type="xsd:string" />
  </message>
- <message name="CreditCkeckerRequest">
  <part name="Username" type="xsd:string" />
  <part name="Token" type="xsd:string" />
  <part name="CloudDataLink" type="xsd:string" />
  <part name="InvoiceCloudDataLink" type="xsd:string" />
  <part name="EInvoiceDataLink" type="xsd:string" />
</message>
- <message name="CreditCkeckerResponse">
  <part name="Username" type="xsd:string" />
  <part name="Token" type="xsd:string" />
  <part name="Income" type="xsd:string" />
  <part name="Job" type="xsd:string" />
  <part name="InvoiceAmount" type="xsd:string" />
  <part name="InvoiceType" type="xsd:string" />
  <part name="InvoiceAssign" type="xsd:string" />
  <part name="InvoiceDate" type="xsd:string" />
  <part name="InvoiceStatus" type="xsd:string" />
  <part name="LoanQuota" type="xsd:string" />
  <part name="DefaultProbability" type="xsd:string" />
</message>
- <message name="doGetCloudDataRequest">
  <part name="Token" type="xsd:string" />
</message>
- <message name="doGetCloudDataResponse">
  <part name="Token" type="xsd:string" />
  <part name="Income" type="xsd:string" />
  <part name="Job" type="xsd:string" />
</message>
- <message name="doGetInvoiceCloudDataRequest">
  <part name="Token" type="xsd:string" />
</message>
- <message name="doGetInvoiceCloudDataResponse">
  <part name="Token" type="xsd:string" />
  <part name="InvoiceAmount" type="xsd:string" />
  <part name="InvoiceType" type="xsd:string" />
  <part name="InvoiceAssign" type="xsd:string" />
  <part name="InvoiceDate" type="xsd:string" />
</message>
- <message name="doEvaluateQuotaRequest">
  <part name="Income" type="xsd:string" />
  <part name="Job" type="xsd:string" />
  <part name="InvoiceAmount" type="xsd:string" />
  <part name="InvoiceType" type="xsd:string" />
  <part name="InvoiceAssign" type="xsd:string" />
  <part name="InvoiceDate" type="xsd:string" />
</message>
- <message name="doEvaluateQuotaResponse">
  <part name="LoanQuota" type="xsd:string" />
  <part name="DefaultProbability" type="xsd:string" />
</message>
- <message name="doCheckCollateralRequest">
  <part name="Username" type="xsd:string" />
  <part name="InvoiceAmount" type="xsd:string" />

```

```

    <part name="InvoiceType" type="xsd:string" />
    <part name="InvoiceAssign" type="xsd:string" />
    <part name="InvoiceDate" type="xsd:string" />
  </message>
- <message name="doCheckCollateralResponse">
  <part name="InvoiceStatus" type="xsd:string" />
</message>
- <flow name="Flow">
  - <links>
    <link name="link1" />
    <link name="link2" />
    <link name="link3" />
    <link name="link4" />
    <link name="link5" />
    <link name="link6" />
    <link name="link7" />
    <link name="link8" />
    <link name="link9" />
    <link name="link10" />
    <link name="link11" />
    <link name="link12" />
  </links>
  - <receive createInstance="yes" operation="get" name="Receive"
    displayName="Receive" portType="wsdl0:CreditCkeckerPort"
    variable="input">
    <source linkName="link1" />
  </receive>
  - <assign name="DataMap1" displayName="DataMap1">
    <target linkName="link1" />
    <source linkName="link2" />
  - <copy>
    <from part="Token" variable="Input" />
    <to part="Token" variable="CreditCkeckerRequest" />
  </copy>
</assign>
  - <invoke inputVariable="CreditCkeckerRequest" name="CreditCkecker"
    operation="CreditCkecker" outputVariable="CreditCkeckerResponse"
    partnerLink="CreditCkeckerService"
    portType="wsdl0:CreditCkeckerServiceImpl" displayName="Credit
    Ckecker">
    <target linkName="link2" />
    <source linkName="link3" />
    <source linkName="link6" />
  </invoke>
  - <assign name="DataMap2" displayName="DataMap2">
    <target linkName="link3" />
    <source linkName="link4" />
  - <copy>
    <from part="Token" variable="CreditCkeckerRequest" />
    <to part="Token" variable="doGetCloudDataRequest" />
  </copy>
</assign>
  - <invoke inputVariable="doGetCloudDataRequest" name="GetCloudData"
    operation="GetCloudData" outputVariable="doGetCloudDataResponse"
    partnerLink="GetCloudDataService"
    portType="wsdl1:GetCloudDataServiceImpl" displayName="Get Cloud

```

```

Data">
  <target linkName="link4" />
  <source linkName="link5" />
</invoke>
- <assign name="DataMap3" displayName="DataMap3">
  <target linkName="link3" />
  <source linkName="link4" />
- <copy>
  <from part="Token" variable="CreditCheckerRequest" />
  <to part="Token" variable="doGetInvoiceCloudDataRequest" />
</copy>
</assign>
- <invoke inputVariable="doGetInvoiceCloudDataRequest"
  name="GetInvoiceCloudData" operation="GetInvoiceCloudData"
  outputVariable="doGetInvoiceCloudDataResponse"
  partnerLink="GetInvoiceCloudDataService"
  portType="wsdl2:GetInvoiceCloudDataServiceImpl" displayName="Get
  Invoice Cloud Data">
  <target linkName="link7" />
  <source linkName="link8" />
</invoke>
- <while condition="DefinedByPHPCode" name="while">
- <condition>
- <phpCode>
- <![CDATA[
  <!-- PHP Code Inserted here -->
]]>
</phpCode>
</condition>
<target linkName="link10" />
<source linkName="link11" />
- <flow>
- <links>
  <link name="link13" />
  <link name="link14" />
</links>
- <invoke inputVariable="CheckeCreditRequest" name="CheckeCredit"
  operation="CheckeCredit"
  outputVariable="CheckeCreditResponse"
  partnerLink="CheckeCreditService"
  portType="wsdl1:CheckeCreditServiceImpl"
  displayName="Checke Credit">
  <source linkName="link13" />
</invoke>
- <invoke inputVariable="EvaluateQuotaRequest"
  name="EvaluateQuota" operation="EvaluateQuota"
  outputVariable="EvaluateQuotaResponse"
  partnerLink="EvaluateQuotaService"
  portType="wsdl1:EvaluateQuotaServiceImpl"
  displayName="Evaluate Quota">
  <source linkName="link14" />
</invoke>
</flow>
</while>
- <invoke name="Confirmation" operation="null" partnerLink="null"
  portType="null" displayName="Confirmation">

```

```

- <script>
  - <phpCode>
    - <![CDATA[
      <!-- PHP Code Inserted here -->
    ]]>
  </phpCode>
</script>
<target linkName="link2" />
<target linkName="link4" />
<target linkName="link7" />
<source linkName="link15" />
</invoke>
- <reply name="Reply" operation="get" displayName="Reply"
  partnerLink="CreditCkeckerProcess"
  portType="wsdl0:CreditCkeckerProcess"
  variable="CreditCkeckerResponse">
  <target linkName="Link15" />
</reply>
</flow>
- <scope>
  - <faultHandlers>
    - <catch faultName="wsdl4:Exception" faultVariable="creditCardFault">
      - <flow>
        - <links>
          <link name="Link16" />
        </links>
        - <invoke name="HandleFault" operation="null"
          displayName="handleFault" partnerLink="null"
          portType="null">
          <source linkName="Link16" />
        </invoke>
        - <reply name="Reply" operation="get" displayName="Reply"
          partnerLink="ProcessStarter" portType="wsdl0:travelPort"
          variable="CreditCkeckerResponse">
          <target linkName="Link16" />
        </reply>
      </flow>
    </catch>
  </faultHandlers>
  - <flow>
    - <invoke name="Exception" operation="null" partnerLink="null"
      portType="null" displayName="Confirmation">
      <target linkName="link2" />
      <target linkName="link4" />
      <target linkName="link7" />
      <source linkName="link16" />
    </invoke>
  </flow>
</scope>
</process>

```