國立交通大學

資訊學院　資訊學程

碩士論文

個人化儲存服務系統平台之設計與實作

Design and Implementation of a Unified Personal Storage Framework and Service

研究生：吳　捷

指導教授：曾建超　教授

中華民國九十九年七月

個人化儲存服務系統平台之設計與實作
Design and Implementation of a Unified Personal Storage Framework and Service

研 究 生：吳　捷　　Student：Chieh Wu

指導教授：曾建超　　Advisor：Chien-Chao Tseng

國 立 交 通 大 學
資訊學院　資訊學程
碩 士 論 文

A Thesis
Submitted to College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master of Science
in
Computer Science
June 2010

Hsinchu, Taiwan, Republic of China

中華民國九十九年七月

個人化儲存服務系統平台之設計與實作

學生： 吳　捷　　指導教授：曾建超 博士

國 立 交 通 大 學　　資 訊 學 院　　資 訊 學 程 碩 士 班

摘　　要

本論文設計並實作一套整合性個人檔案平台以整合個人在網路儲存媒體所擁有的儲存空間，使用者可透過此平台所提供之單一介面，存取不同儲存空間以及本地端硬碟的資料，而且存取不同儲存空間時，使用者不會察覺到儲存媒體或使用方式的不同，此外，也可藉由此平台與他人分享在不同儲存空間的檔案，分享時也不會有認證或是存取上的困擾。此平台讓使用者更能充分運用與發揮網路儲存空間所帶來的便利。

受惠於網路儲存媒體所提供免費且大量的儲存空間，使用者可擺脫時間與地點的束縛，透過網路存取上述儲存空間。善用這些儲存空間亦可有效地幫助使用者備份與分享檔案。然而不同網路儲存媒體間不僅存在有相容性問題，亦無法與使用者原有的儲存設備整合，在分享檔案時也因相容性問題使得分享功能倍受限制。因此在享受網路儲存服務所帶來的便利時，使用者也必須忍受使用上所產生的問題。

本論文針對上述問題提出整合性個人檔案平台，使用者可透過本平台利用單一操作方式存取各式儲存服務與儲存空間，除了延續使用者操作經驗，也達到管理眾多不種儲存媒體的目的。此外本平台亦提供使用者利用互相信賴之分享機制，透過較直覺的方式分享檔案給朋友而不需要額外的服務帳號或認證。最後本平台延伸瀏覽器的環境至使用者桌面使檔案存取不再限制於瀏覽器中，而能持續與使用者保持互動。

Design and Implementation of a Unified Personal Storage Framework and Service

Student：Chieh Wu　　Advisors：Dr. Chien-Chao Tseng

Degree Program of Computer Science
National Chiao Tung University

ABSTRACT

In this thesis, we present the design and implementation of a Unified Personal Storage Platform that can integrate the storage spaces a user acquired from various network storage services. Under this platform, users can access various network or local storages via a single access interface and will not be aware of the differences of underlying storage services. Furthermore, users can also share their resources in this platform without suffering from authentication or sharing policy issues. In a word, the "Unified File Platform" can help users to manage their own storage spaces across Internet and share information with others very conveniently without detail.

Because of the free and huge spaces provides by many emerging network storage service providers, users can access their photos, videos or other resources any time and any where. Furthermore, users can also use these storage spaces to backup or share files. However it is inconvenience for user to manage many storage spaces across various network storage service systems with different authentication or access mechanisms. Furthermore, different network storage service providers may adopt different sharing policies.

In order to resolve above issues, the UPS platform provides a unified GUI for users to manage and access different storage services across Internet. The unified GUI inherits the concept of common file access interfaces and all storages appear as files in UPS. Furthermore, UPS also provides a sharing mechanism with trusted relationship. With this mechanism, user can share or access file without tedious account creation and authentication procedures. Last but not least, UPS client can run on desktop so that it can notify users the changes in file sharing.

誌　　謝

本篇論文的完成首先要感謝指導教授 曾建超博士，感謝教授不論在研究方向或是研究方法上給予我的指導，特別是教授在教學上嚴謹的態度與完整的思考邏輯，更是讓我獲益不少，我也要感謝教授在這段期間能容忍我因工作所帶來的不便，讓我能順利完成研究與論文。

另外我也要感謝所有口試委員提供的寶貴建議，讓本篇論文更加完善，還有特別感謝實驗室的史永建學長與林家樑學長，謝謝他們在研究與報告上給予我的協助與建議，幫我解決不少問題與疑惑。

求學與研究的路是漫長且辛苦的，特別是還有工作的考量，感謝公司的同仁對我的幫助與諒解，也感謝我的父母以及太太姿如的鼓勵與支持，讓我在遭遇困難與挫折時，能夠打起精神繼續的走下去，感謝每一位幫助我完成論文學位的人，有了你們的陪伴，才能有我今日的榮耀。

目　　錄

表目錄

圖目錄

一、緒論

1.1 研究動機

隨著儲存的技術發展，儲存容量的成長速度越來越快，個人儲存裝置的容量已經不再是問題，在 Internet 上也有越來越多的線上服務開始提供大容量且免費的儲存服務，使用者可以將自己的照片，影片或是文件存放到這些空間中，並且開放與朋友分享，另一方面，自從雲端運算的概念被提出後，相對應的雲端儲存服務也如雨後春筍般的出現，使用者透過一些既有的介面存取雲端中的檔案，就如同存取自己硬碟裡的檔案一般，透過網路，使用者存取服務內的文件與檔案時可以不受時間與地點的限制，這些服務特別適用於檔案文件的備分[1]與分享，傳統上使用者必須花心力於儲存裝置的擴充與維護，但利用網路儲存服務，使用者只需專心於文件和檔案的管理即可，儲存硬體的整合與維護工作則是交給服務廠商。

但是在享受這些服務的同時，我們也見到許多使用這些儲存服務上的問題

1.使用者必須面對不同的操作環境
雖然目前很多的線上服務都有直覺性的操作介面，但各服務之間所能提供的功能與操作步驟都有一定程度的差別，有時使用者必須透過許多的網頁才能順利的存取到需要的檔案，這些網頁中有包含有許多的廣告，無形之中增加了存取的時間，再者使用者可能必須為了存取不同的服務而必須安裝不同的介面程式，在使用上來說也是一件麻煩的事情。

2.管理多個服務不易
使用者有時為了要得到更多的免費空間或是存放不同類型的檔案，常常申請許多相同或不同類型的儲存服務，當需要使用這些空間時，使用者必須記得這些散落在各地的檔案，有些服務也許因為閒置太久沒有存取，可能連帳號都已遺失。

3. 難與個人儲存裝置整合
線上儲存服務很難與本地儲存設備或是其他儲存服務整合，雖然目前雲端儲存都有提供標準的存取介面，但雲端服務大都擁有特別的使用端介面，與使用者既有的檔案管理介面不同，在使用的方法上也有出入，當使用者要在這些服務與儲存裝置之間移動檔案時，必須花費一番功夫才能達到目的。

4.會員制分享機制的限制
線上服務大多具有分享機制，使用者可以將檔案、圖片與朋友分享，但分享的對

象僅限於該服務的會員，不同的服務之間也無法做檔案的分享，試想在 Flickr 上分享照片給朋友時，朋友也必須有 Flickr 的帳號，若是沒有認證的分享機制，則又有安全上的考量。

5.瀏覽器存取的限制

線上服務大多透過瀏覽器進行存取，所有存取動作也都是透過瀏覽器，一但關閉瀏覽器，則使用者就無法取得服務的任何訊息，也無法得知及時的檔案狀態，再者，不同的瀏覽器也存在著不相容的問題，某些特別的服務功能可能只適用某個特定的瀏覽器，對使用的方便性來說，也是一大折扣。

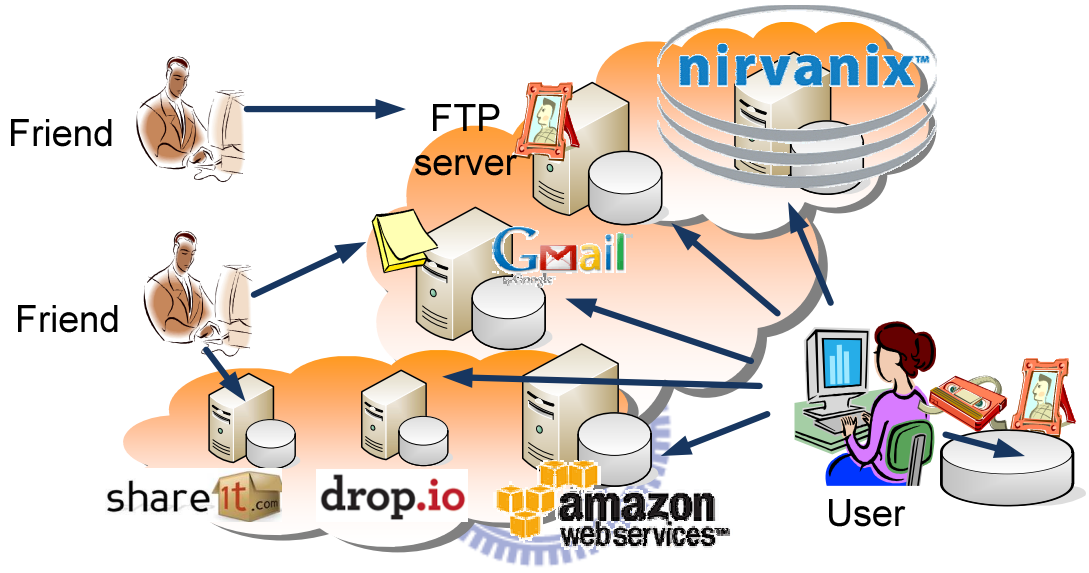

圖 1 使用者存取不同網路儲存服務

基於以上的問題與需求，我們提出一個新的個人檔案管理平台，能夠解決這些問題，並且符合使用者未來的需求。

1.2 研究目的

本研究的目的是在設計與發展一套個人化的檔案整合系統，透過本系統，使用者可整合目前的儲存服務與裝置，並且充分享受線上儲存服務所帶來的便利性，讓使用者不再有儲存空間上的限制，面對異質性的儲存媒體上也可以輕鬆的管理或是移動檔案，可減少日常備份或是分享檔案時不必要的轉換，以及在不同儲存服務之間難以管理的情況，系統必須要能提供以下三大功能:

1.能夠提供整合性檔案存取平台，使用者透過本系統可利用同一種操作方式存取各式各樣的儲存服務與儲存媒體，對使用者來說不但延續了使用操作的經驗，也達到管理眾多不種服務的目的。

2.能夠提供使用者利用互相信賴之分享機制，透過直覺的方式將檔案分享給所需要的朋友而不需要額外的服務帳號或認證。

3.延伸瀏覽器的環境到使用者的桌面，檔案的存取不再只限制在瀏覽器中，而是能持續與使用者保持互動。

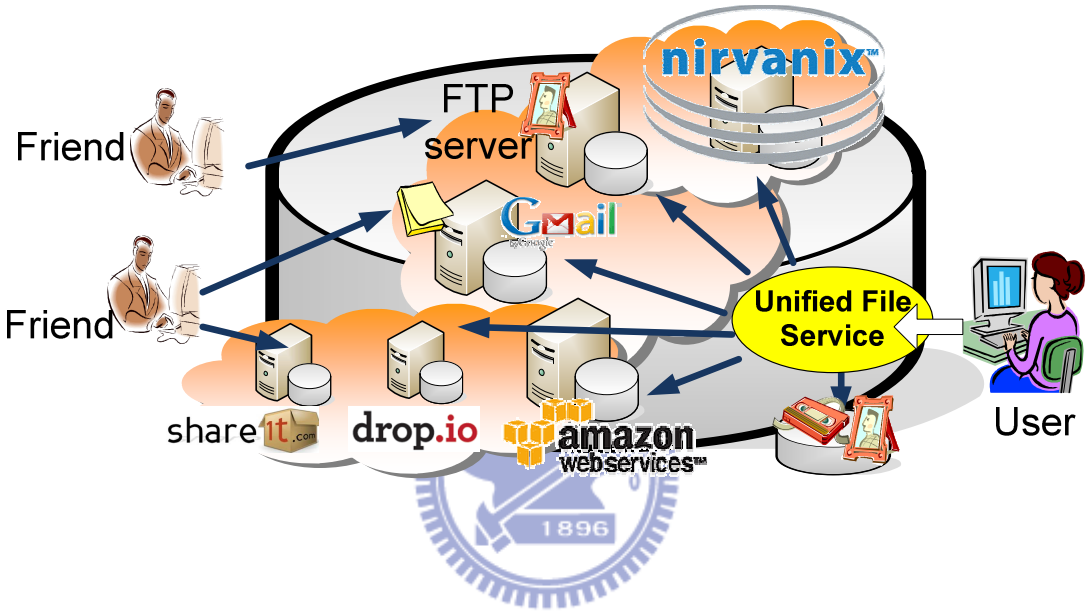

圖 2 使用者透過 Unified File Service 存取各式儲存服務

1.3 論文架構

本論文內共分七章，依序如下，第一章為序論，說明為何要發展此個人化檔案系統，其目的與目標為何? 第二章為相關背景研究，介紹目前有哪些標準或是類似的檔案整合系統，第三章系統功能與需求分析，描述現存系統的問題與缺點以及本系統如何解決這些問題，第四章為系統設計，詳述系統各內部元件之功能與機制，為了達到所需要的功能或是解決現存問題系統內部應該如何設計，第五章實作與成果展示，說明如何依據設計做出實際的系統以及系統細部實作，並且展示系統運作之畫面，第六章系統比較，列出本系統與現存系統之功能比較，最後第七章總結，說明本研究之結論與貢獻以及未來發展之可能方向。

二、相關背景研究

本章將就目前現有之檔案整合平台與相關技術作介紹。

2.1 Virtual File System 與 Apache VFS Library

虛擬化檔案系統(Virtual File system)，是架構在真實檔案系統上的抽象層(abstract layer)，不同的檔案系統透過抽象層將實體檔案系統隱藏起來，並且透過一個統一的介面進行存取，使用者在存取檔案時，就如同存取本地端的硬碟般，讓使用者不會查覺到所使用的檔案系統之不同，藉由 Virtual File System 的隔絕，不同的檔案系統可被同時使用在同一個 OS(Operation System)當中。

所謂的抽象層在 Virtual File System 中是指一個已經定義好的介面，最早實作此抽象層的系統是在 UNIX 類的 OS 中，此介面是介於系統核心與檔案系統之間的橋樑，系統核心存取檔案都須透過此一抽象層，近年來越來越多的系統都可以實作出 Virtual File System，它不但可以存在於系統核心層(Kernel Layer)也可以存在於應用層(Application Layer)。

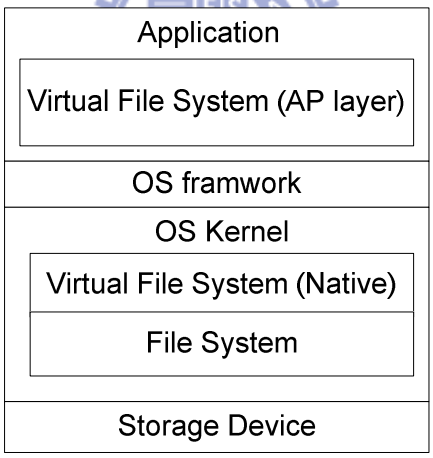

圖 3 Virtual File System 在 OS 中之架構

Apache Common Virtual File System[2]便是一個使用 Java 所開發在應用層的 Virtual File System Library，透過它所提供的 API，使用者可以開發存取不同檔案系統的應用程式，其中包含 Local Files、FTP、HTTP 以及 ZIP 和 GZIP 等壓縮檔，透過 VFS library 使用者不會察覺到使用不同的檔案系統，在存取檔案時所使用的方式都是一樣。

Linux 的 FUSE (Filesystem in Userspace)[15]則是一個存在於 Linux Kernel 的 Virtual File System 模組，透過 FUSE 使用者可以開發整合不同的 File System 並且掛載在 Linux 中讓使用者透過 Linux 檔案系統存取而不需重新編譯 Linux

kernel，目前已有多種檔案系統可透過 FUSE 掛載，以下舉幾個透過 FUSE 所完成的 FileSystem

- SSHFS: 透過 SSH 協定存取遠端檔案
- Httpfs: 透過 HTTP 協定存取遠端檔案
- GmailFS: 透過檔案系統方式存取 GMail

2.2 雲端儲存 (Cloud Storage)

雲端儲存是最近幾年從雲端計算(Cloud computing)所延伸出來的概念，雲端儲存不是一種儲存技術，而是一種服務，雲端儲存透過網路軟體技術將大量不同類型的儲存裝置整合起來，提供外界資料存放與管理之服務，使用者透過網路即可存取到自己所存放的資料，而不需考慮複雜的儲存設備與硬體建構的問題，在雲端儲存內部包含了許多複雜的技術，一般來說我們將雲端儲存的架構劃分為五層架構[3]。

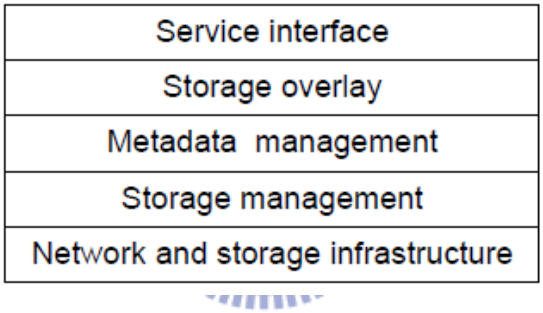

圖 4 雲端儲存服務之基本架構

1. Network and storage infrastructure: 提供基本的網路與儲存的硬體架構，使用者主要透過網路存取雲端裡的資料，所以在雲端儲存服務當中，最底層的基本設施便是網路與儲存設備。
2. Storage management: 提供實體資料的儲存方式，其中定義了儲存方法與結構，分散式儲存的管理與資料群組的劃分等。
3. Metadata management: 透過整合不同資料群組的 metadata，讓處於不同群組內的資料互相交流，進而讓不同群組協同工作。
4. Storage overlay: 提供單一的邏輯儲存單元，虛擬化技術與資料存取都在此層實作，從使用者角度來看只會看到簡單的介面與單一的儲存裝置。
5. Service Interface: 提供一致性的介面給使用者進行資料存取，並且管制 Client 端的存取方式與權限。

Amazon 的 S3 (Simple Storage Service) 是典型且相當有名的雲端儲存服務，S3 提供使用者利用 web service 介面存取無限制的容量空間，S3 將每個儲

存物件的容量定為 5 Gbyte，所有的儲存物件在系統中都被存放在 Buckets 中，每個 Bucket 都有一個特有的名稱與使用者所設定的密碼，Bucket 與儲存物件可被建立或是透過 HTTP 與 SOAP (Simple Object Access Protocol) 介面存取，每個存取的請求都必須先經過使用者所設定在 Bucket 內的密碼或是物件中的 access list 的驗證，Amazon 聲稱直到 2010 年三月，儲存於 S3 服務的資料物件已高達 102 百萬個。

2.3 eXtensible Access Method (XAM)

2.3.1 XAM 之目的

XAM 是指 (eXtensible Access Method) ，它是由 Storage Networking Industry Association (SINA) 所提出的新一代儲存裝置介面的標準，SINA 有鑒於在現存之儲存設備的管理與整合因為受到各家業者特定的儲存介面而無法互相溝通，因此才提出了此一標準。

一般所謂的資料檔案(fixed content file)的生命週期相當長，這些檔案大部分只做於參考使用，不會經常被更新，但是當我們在儲存這些檔案時，所使用的儲存技術卻是不斷的更新，當管理者把資料從舊系統移植到新系統時，許多不相容的問題也就浮現，XMA 的概念即是把資料(data)看成是一個物件(Object)，其中包含資料本身以及自我描述(self-describing)的資料，所謂自我描述的資料即記錄資料本身特性和屬性的 metadata，藉由這樣的方法，儲存系統不需要去識別(recognize)這些資料，而是去讀取這些 metadata 進而了解資料的特性，XAM 定義了這些 metadata 的格式與查詢這些描述資料的介面，以及如何藉由這些介面去取得所需要的資料，SINA 期望透過 XAM 的制定能夠做到以下三點:

1. 應用程式可以跨不同儲存媒體使用。
2. 資料物件(Data Object)可以在不同的儲存裝置中移動。
3. 資料物件(Data Object)可以在不同的應用程式之間移動。

2.3.2 XAM 架構

在 XAM 中，定義了兩個標準的 Interface，分別是 XAM API 與 VIM(Vendor Interface Module)**[14]**

1. XAM API: 定義 XAM 架構與應用程式端的介面，應用程式透過此介面可存取到 Storage 資料，應用程式必須連結此 XAM library 方可使用此 API。
2. VIM API: 定義儲存系統與 XAM 架構之間的介面，讓 XAM 可透過此介面存取儲存裝置，此部分 Interface 的實作需要由設備廠商提供。

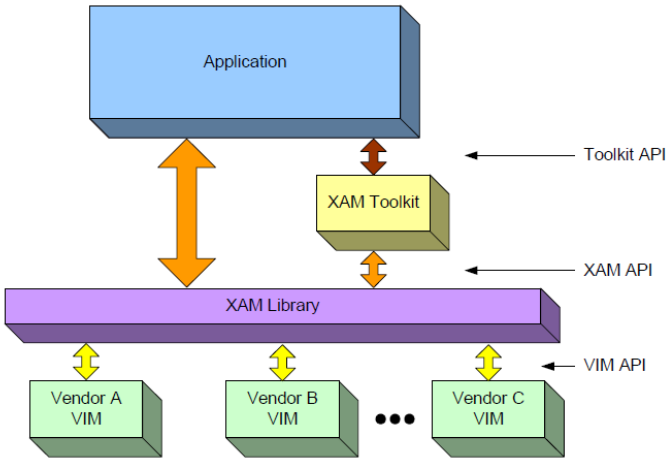

圖 5 XAM 架構圖

應用程式透過 XAM API 存取儲存裝置，而 XAM library 會選擇適當之 VIM 與裝置溝通，應用程式同時也可透過 Toolkit API 使用 XAM library，Toolkit 是包含 XAMQuery，XUID 等工具類的 API，使用這類 toolkit 的 API 可更方便存取 XAM library。在 XAM 中包含兩個重要的元件，分別是 XSet 與 XSystem。

- XSet: 在 XAM 中，XSet 可以代表資料的邏輯單元，任何資料進入 XAM 都被包裝成 XSet，當應用程式將資料透過 XAM 送往儲存裝置時，XAM 會建立 XSet 並且將資料填入後送至儲存裝置，若過程順利成功，儲存裝置會回應一個 XUID 給應用程式，應用程式透過此 XUID 便可搜尋到這筆資料，同時也可以透過交換此 XUID 而達到與其他應用程式交換資料的目的。

- XSystem: 可以看成是裝滿 XSet 的容器，XSystem 提供了存取與管理 XSet 的能力，XSystem 同時也提供了 XSet 的建立、搜尋、刪除與儲存的方法，應用程式透過 XSystem 可以存取到 XSet 並且存取到所需要的資料，而 XSystem 本身也具有基本的 XSet 管理能力，像是存取 XSet 的安全機制、虛擬化資料、容錯能力與效能的調整等。

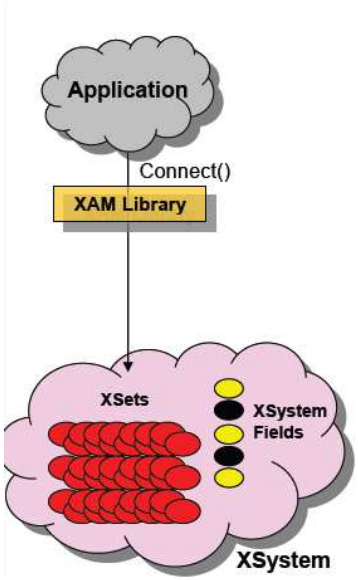

圖 6 XSystem 與 XSet 之關係

外部的應用程式可使用 XAM Library 存取 XSystem，XAM library 目前提供 C 與 Java 的程式 binding 介面讓應用程式透過 XAM library 存取 XSystem，另外 XAM Library 也定義了安全機制，它使用 SASL (Simple Authentication and Security Layer) authentication 機制確保與應用程式之間的安全。

2.4 Storage Resource Broker (SRB)

Storage Resource Broker 是一種分散式資料系統，由聖地牙哥國家超級計算中心 (SDSC）所開發， SRB 是一套結合資料庫系統與檔案系統觀念的分散式檔案管理與資源儲存系統，它是一個 Client/Server 架構的 middleware 系統，在 Client 端，它提供了使用者一個存取多種異質存儲系統的統一介面，涵蓋了異質儲存系統的特性。它支援廣域網路環境下多種資料源的存取，並且提供了資料複製、複本資料的存取、檔案的彙集、分散式檔案的邏輯集合等功能。

2.4.1 SRB 架構

在 SRB 的架構中有三項重要的元件，SRB Client, SRB Server 與 MCAT (Meta data Catalog)**[4]**，其功能分述如下:

1. SRB Client: 透過 SRB 所提供的 API 進行資料的存取，SRB 有提供可整合之 API 與 user application 程式。
2. SRB Server: 負責接收 Client 端的請求，並將結果回傳，另一方面 SRB Server 也必須負責與 MCAT 溝通以及與底層不同的儲存系統進行存取。
3. MCAT: MCAT 本身是一個存放 metadata 的資料庫，其中包含了 data set、

使用者資料以及各種不同儲存資源的 metadata，這些 metadata 定義了存取權限的控制、資料儲存的結構 (Collection) 等，透過 MCAT 所提供資料的 logical name 或是屬性，使用者所存取的不再是資料檔案的實體位置，因此可以將儲存系統與應用程式分離。

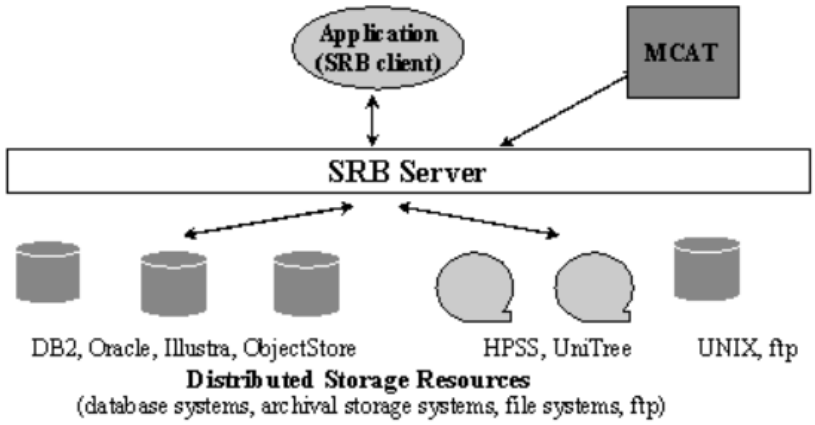

圖 7 SRB 的基本架構

2.4.2 SRB 處理流程

在 SRB 處理 Client 端請求時，主要的兩個基本元件為 SRB Master 與 SRB Server 也稱為 SRB Agent。

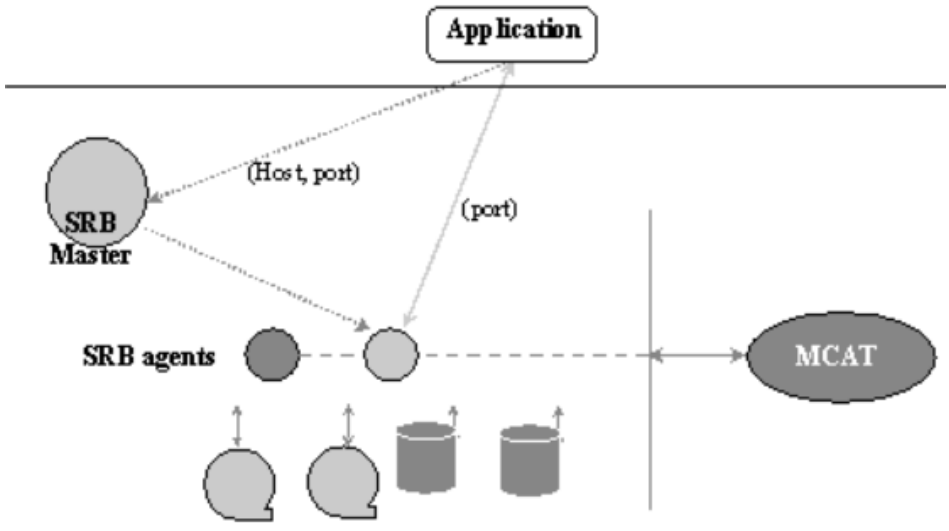

圖 8 SRB 處理流程

SRB Master 在 SRB 系統中扮演 Server 的角色，它持續接收 Application 所發送的請求，一但 Application 與 SRB Master 連線並通過認證後，SRB Master 會產生一個新的 Server Process 去處理 Application 後續的請求與結果回應，在此同

時 SRB Master 也可以接收其他 Application 的請求，而這個新產生的 Server Process 在 SRB 中被稱為 SRB Agent，Application 透過網路利用一組定義好的 API 與 SRB Agent 溝通，因此 Application 與 SRB Agent 或 SRB Master 可存在於不同的主機當中。

在 SRB Agent 的架構當中，每個 Client 的請求都經過 Dispatcher 發送到位於 SRB Agent 內的兩個 Handler，分別是 High level handler 與 Low level handler

- High level handler: 負責處理需要經過 MCAT 的請求，若資料經由 MACT 取得，則 MCAT 會自動對資料進行管理追蹤，直到資料從儲存系統中刪除，當使用者進行資料的建立、檔案存取、搜尋或是檔案權限管控時，這些動作都必須存取 MCAT 內的 metadata，而這些動作也必須透過 High Level Handler 進行。

- Low level handler: 處理不需經過 MCAT 的 Application 請求，有時這些請求也來自於 High Level handler，Low level handler 負責處理與儲存系統溝通的 I/O 能力，目前在 SRB 系統中 Low level handler 只有定義對檔案系統與資料庫存取的能力。

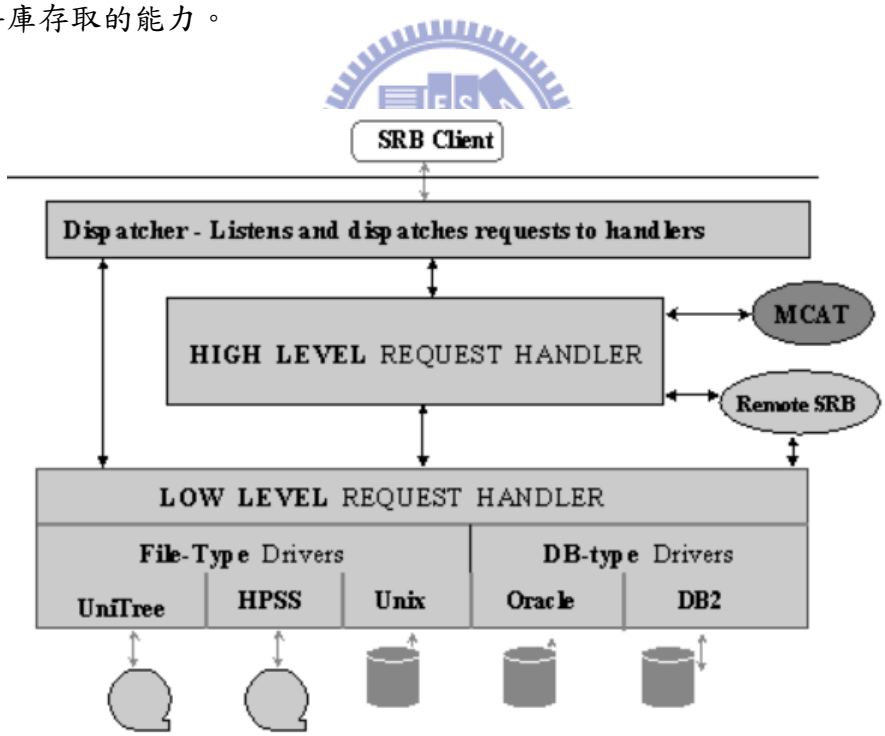

圖 9 SRB Agent 內架構圖

何時需要 High level handler 或 Low level handler 是取決於 Application 的請求，例如: 使者想在系統中儲存一個檔案，當請求進入 SRB Agent 後，SRB Agent 先使用 High level handler 到 MCAT 中找尋使用者權限或是可使用之儲存資源，當 MCAT 回傳明確的儲存位置後，系統再使用 Low level hander 作建立資料的

動作，並且同時在 MCAT 中建立該檔案的 metadata 進行追蹤，而 SRB Agent 會將該檔案的 handler 回傳給 Application，Application 便可利用此 handler 進行後續的讀寫檔案，若此資料是存放於本地主機上的儲存系統，則 SRB Agent 會直接將後續 Application 讀寫請求交由 Low level handler 作 I/O 上的存取。

2.4.3 SRB Federation 之檔案交換機制

SRB Server 也可以組合起來成為一個 Federation，一同處理 Application 的請求，並且在這些 Server 中交換彼此檔案資源，以 Application 發出開啟資料之請求為例，整過資料交換過程如下:

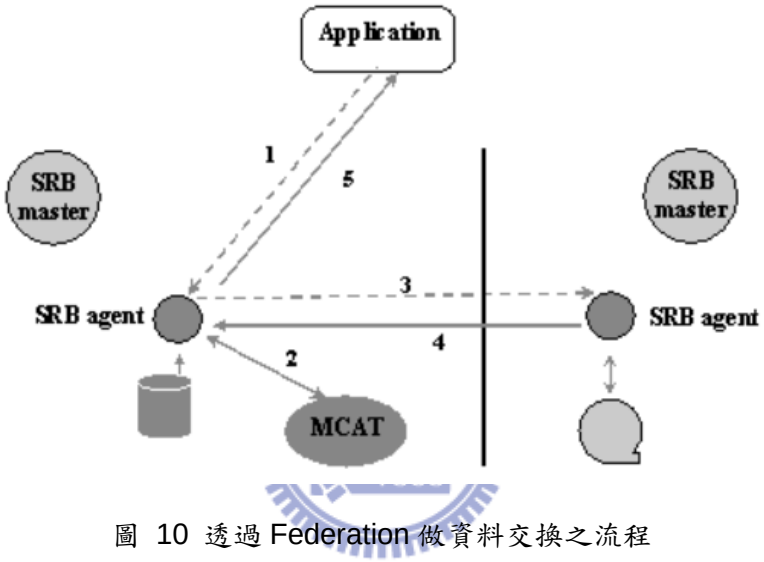

圖 10 透過 Federation 做資料交換之流程

1. Application 在與 Server A 上的 SRB Master 完成認證與連線後，便發送開啟資料的請求給所屬的 SRB Agent。
2. SRB Agent 會傳送 User ID 與資料名稱給 MCAT 要求確認 Application 是否有足夠的權限存取相關資料，經過確認後，MCAT 會回傳該資料所在儲存裝置上的實體位置。
3. 若 Server A 上的 SRB Agent 發現所要求的資料位於 Server B 上，Server A 上的 Agent 會發出開啟資料的請求給 Server B 上的 SRB Agent。
4. Server B 上的 Agent 會根據儲存系統的型態，發出存取檔案請求給 Low level handler 以便取得 File handle 以及資料的相關資訊回傳給 Server A。
5. Server A 上的 Agent 將所得到的 File handle 以及相關資訊回傳給 Application，Application 再跟據這些資訊與 handler 進行資料讀取。

三、系統功能與需求分析

本章介紹系統設計的概念與系統架構，在決定系統的功能之前，必須先瞭解現行檔案整合系統平台的問題與缺點，分別由以下三個部分來說明。

1.在使用者介面方面:

大多數的網路儲存服務的使用者介面都是以瀏覽器為主，雖然使用者端不需額外安裝任何軟體只需用瀏覽器即可，但有許多功能也因此被限制在瀏覽器中，脫離了瀏覽器便無法使用。

2.在整合網路儲存服務方面:

FUSE 與 Apache VFS 雖然有能力整合不同檔案系統，但本身不是一個完整的檔案平台，兩者沒有檔案分享機制或是概念，這兩個虛擬檔案系統主要解決在不同檔案系統之間的檔案存取問題，而非提供一個完整可用的個人化檔案管理平台。

新興的雲端儲存服務大多只能存取服務供應商所提供的儲存空間，空間大小的使用則是依照使用者付費的多寡而決定，對於現有其他免費的儲存服務則無法相容，在這類服務當中雖然有提供檔案分享與權限管理的概念，不過分享的方式為被動的告知，而非主動通知。

XAM 是未來儲存媒體整合的一個標準，新標準的提出有助於對未來儲存服務的整合，並且有越來越多的標準將線上存儲也納入管理整合的範疇，但這必須要線上儲存服務與平台開發者相互的配合才能實踐，對現今線上儲存服務的使用者來說，此構想並非立即可行，而新系統對舊儲存服務的相容性也是一大問題，現存的服務必須透過開發相容的介面才能與標準溝通，無形之中也增加了新標準推行的困難度。

SRB 具有很好的異質性儲存系統的整合能力，不過整合的儲存對象是以檔案系統或是資料庫為主，而不是線上的儲存服務，並且在使用 SRB 之前必須完成相當龐大的網路元件的設置[5]，以個人的使用角度來看，其實並不適合。

3.在分享檔案方面:

不同的網路服務有不同的安全或是分享的機制，也因此造成使用者與朋友之間分享檔案的困難，有些服務分享的對象也必須具有該服務帳號才可以存取檔案，所以當使用者做檔案分享時，也只能分享給服務中的會員，無形之中限制了分享的便利性，另一方面，多數的服務在分享檔案給非會員時大多是使用 URL 連結的方式，將 URL 寄給朋友，朋友再利用此連結存取到使用者所分享的檔案，此方法雖然可以解決會員機制的問題，但是 URL 連結的安全性也比較低，任何人取得此連結都可以存取得到檔案，即使存取 URL 連結時可再增加一個密碼的

安全認證，但對存取者來說，每次存取都必須要記得密碼，無形之中是增加了存取分享檔案的困難度。

在分享的機制上來說，多數的服務對服務訊息都是被動的通知，例如: 使用者分享了一張照片給朋友，朋友必須去瀏覽使用者的檔案或是使用者的訊息才知道有新的照片被分享出來，如此被動的分享機制對社群訊息的即時性來說是大打折扣。

基於以上所提到問題，我們以使用者個人的檔案服務平台最為目標，在安裝容易與不耗用龐大資源為前提下提出檔案平台(Unified File Platform)的設計，UFP 必須要具備以下特性與功能:

1. 在使用者介面方面:
   平台必須提供一個定義完整且容易整合的使用者程式介面，讓使用者製作客製化的程式時，不受會侷限於到某個特定介面程式，如此也可提高使用者程式端的彈性，使用者透過此介面可存取到後端各種不同的儲存服務，而使用者也不會查覺到服務之間的差異。

2. 整合網路儲存服務方面:
   系統要盡可能的將現行與未來服務整合到此平台，在整合服務時，必須考慮以服務端變動最小的情形下進行整合，平台同時也必須提供基本檔案管理與操作的功能，例如檔案的刪除，更改檔名，複製，但不包含對檔案即時編輯之能力，對檔案的編輯應於使用者桌面環境中透過適當的程式進行，而不是在 UFP 之中。

3. 在分享檔案方面:
   透過平台新的分享機制達到一致性的分享功能，此分享機制必須要能不受各種不同服務端的認證與分享方式的影響，使用者在分享檔案時也不需要申請新的帳號或是輸入額外的密碼才可存取，另一方面此分享機制也必須提供一個即時的分享訊息通知，讓分享訊息能達到即時通知的能力。

四、系統設計

4.1 系統架構

根據以上需求分析，我們計畫將透過以下方法來達到系統所需具備的功能與特性。

- 使用 file virtualization 的機制將服務間不同的差異性隱藏起來，讓系統可以與現存網路儲存服務做整合。
- 使用不同的 service adapters 與不同服務做溝通，讓儲存服務端在整合時的變動降到做低，甚至不需服務端的改變便可以整合進 UFP 中。
- 設計一個 active sharing with transparent authentication 的方法，在使用者存取不同儲存服務時，不會受到各服務的認證機制所阻擋，同時能夠提供一個主動式的分享通知功能，讓朋友可以即時接收到使用者所分享檔案的訊息。
- 在使用者介面部分，可使用 XML/HTTP 為介面設計檔案相關操作的 API，前端的使用者程式只要能利用 HTTP 與處理 XML，就可與 UFP 介接。

由上述所使用到的方法，我們設計出 UFP 系統的基本架構，如下圖

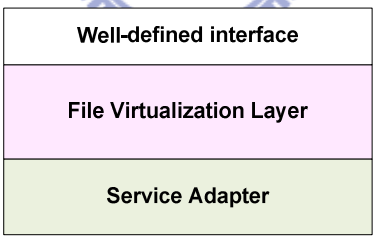

圖 11 UFP 基本系統架構圖

Well-defined interface 是面對使用者程式的介面，讓使用者透過此介面存取檔案，File Virtualization Layer 主要做 file virtualization 的部分，同時它也是進行檔案分享與分享事件發出的主要部分，而 Service Adapter 則是主要網路服務的溝通者，負責與不同的服務溝通。

由圖 11 的架構中，我們更進一步的設計出在 UFP 每一層中的元件，如下圖

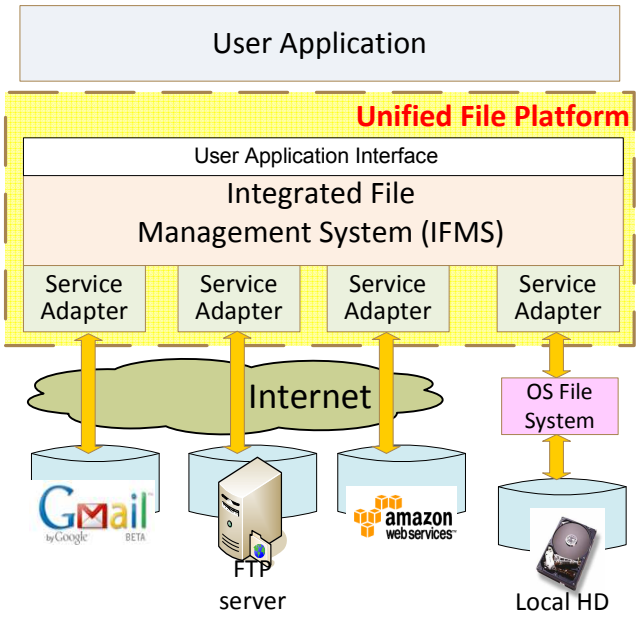

圖 12 UFP 架構元件圖

系統是由兩大部分組成，分別是 Service Adapter, Integrated File management System (IFMS)，以下針對這兩個元件說明彼此扮演角色與溝通的流程。

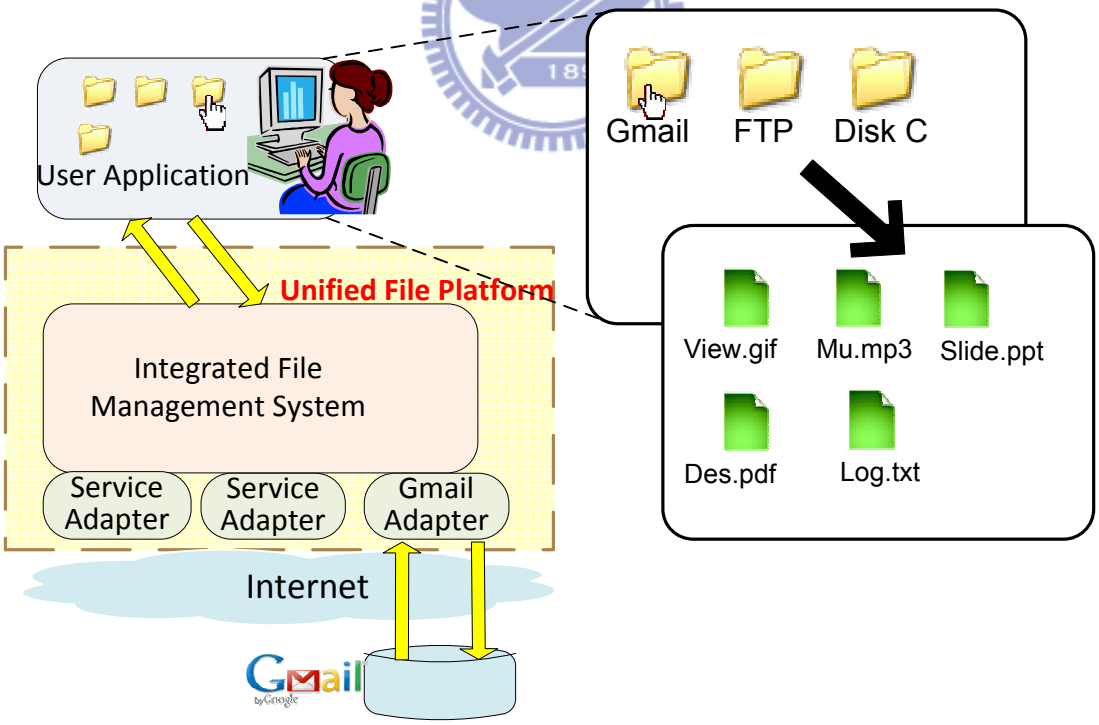

圖 13 UFP 基本處理流程

User Application 負責與使用者溝通，提供使用者檔案操作的介面，它透過 IFMS 的 user application interface 與 UFP 做前端使用者介面的溝通，在 UFP 的 User

Application 中，所有的資源都是以檔案的概念呈現，一種服務使用一種資料夾來表示，資料夾內都是該服務內的資源，當使用者選取該資料夾時，其訊息流程如下:

1. User Application 透過 user application interface 發送 List File 命令給 Integrated File Management System (IFMS)。

2. IFMS 參考相關內部資料處理邏輯並且選擇一個適當的 Service Adapter 當成是與服務端溝通的橋樑，IFMS 透過一個定義好的 common interface 將命令送給 Service Adapter。

3. Service Adapter 將 List File 命令轉換成服務端的指令後透過網路傳送給服務端，讓服務端執行使用者所期望的 List File 操作。

當服務端處理完 List File 命令後便會將 resource 內容回傳，使用者在 User Application 的介面上就會出現該資料夾內的檔案。

4.2 各元件設計

本章介紹 UFP 內兩個主要元件 Integrated File System Management 與 Service Adapter 的細部設計。

4.2.1 Integrated File Management System (IFMS)

IFMS 是 UFP 中主要處理 file virtualization 的元件，它對於前端有 user application interface 與 User Application 做溝通，再根據內部邏輯與處理，透過 Service Adapter 與儲存服務溝通。

為了要保留 User Application 介面的彈性，IFMS 將使用 XML over HTTP 為介面，所有的使用者操作命令都將透過 HTTP GET 參數格式傳送給 IFMS，並且由 IFMS 做出對應的檔案操作，最後使用 XML 格式的文件將結果回傳給前端 User Application，使用 XML 的好處是資料的分析與轉換相當容易**[12]**，在 IFMS 中設計一個 HTML 的轉換元件，便可將 XML 轉換成 HTML 回應給 User Application，如此就可以使用瀏覽器進行檔案操作，更進一步，使用者若是需要開發客製化 User Application，也可透過此 HTTP 介面與 IFMS 傳送命令與結果。

使用 HTTP 作為傳送的傳輸協定主要是 HTTP 已經廣泛的被一般系統或是開發環境所支援，對於必須要客製化前端的 User Application 而言，是較為容易整合的傳輸協定，另一方面因為 IFMS 透過網路與前端溝通，User Application 可以在遠端的裝置中執行，透過 HTTP 從遠端進行檔案操作，UFP 系統與 User

Application 不需要在同一台主機上，如此又增加 User Application 使用上的彈性。

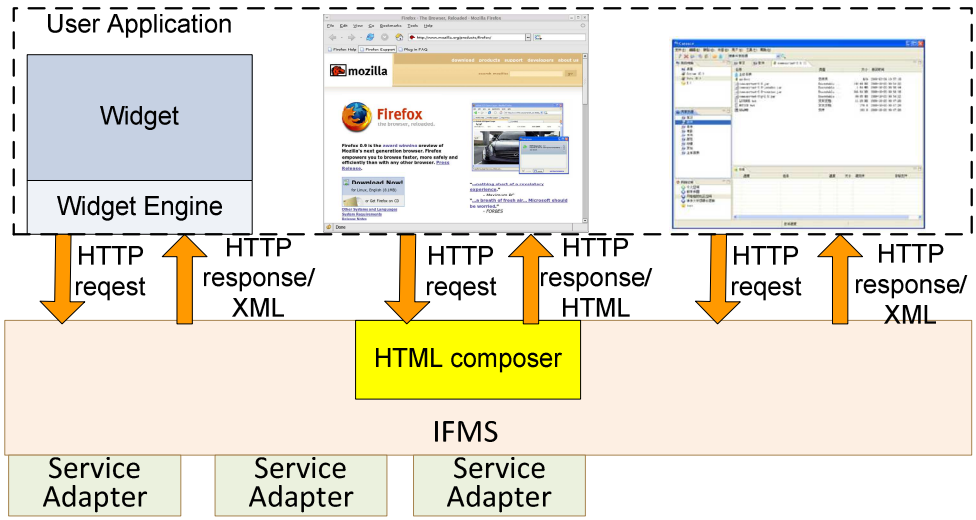

圖 14 User Application 介面

對 IFMS 來說，它必須負責處理所有 User Application 傳送來的請求，經過處理後，再選取適當的 Service Adapter 傳送到服務端進行檔案操作。

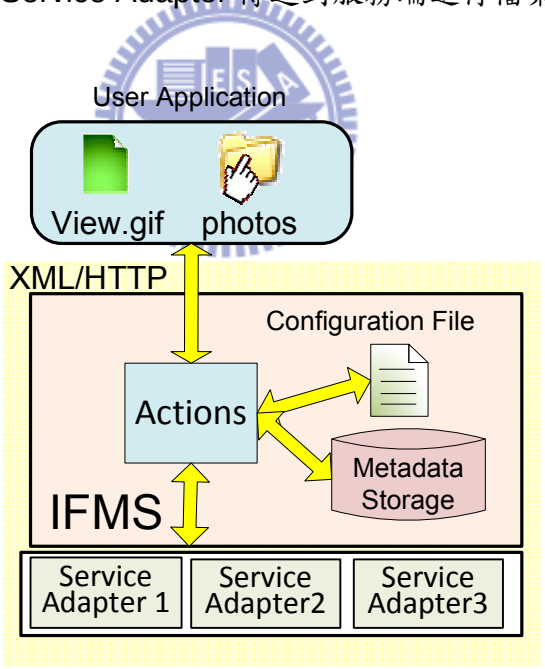

圖 15 IFMS 之內部處理流程

由於在 IFMS 中要處理許多 User Application 的檔案操作需求，所以必須要有相對應的元件來處理這些請求，因此我們在 IFMS 中設計了 Action 元件來處理不同的命令，每個 Action 負責處理單一的命令，從 User Application 傳送來的請求必須指定所要操作的檔案命令，命令進入 IFMS 後交給合適的 Action 進行處理，在每個 Action 中都有各自檔案操作的邏輯，依照邏輯的處理結果再交由

Service Adapter，因此在 Action 中，我們利用 Action Command 來區分處理不同檔案操作的 Action 元件。

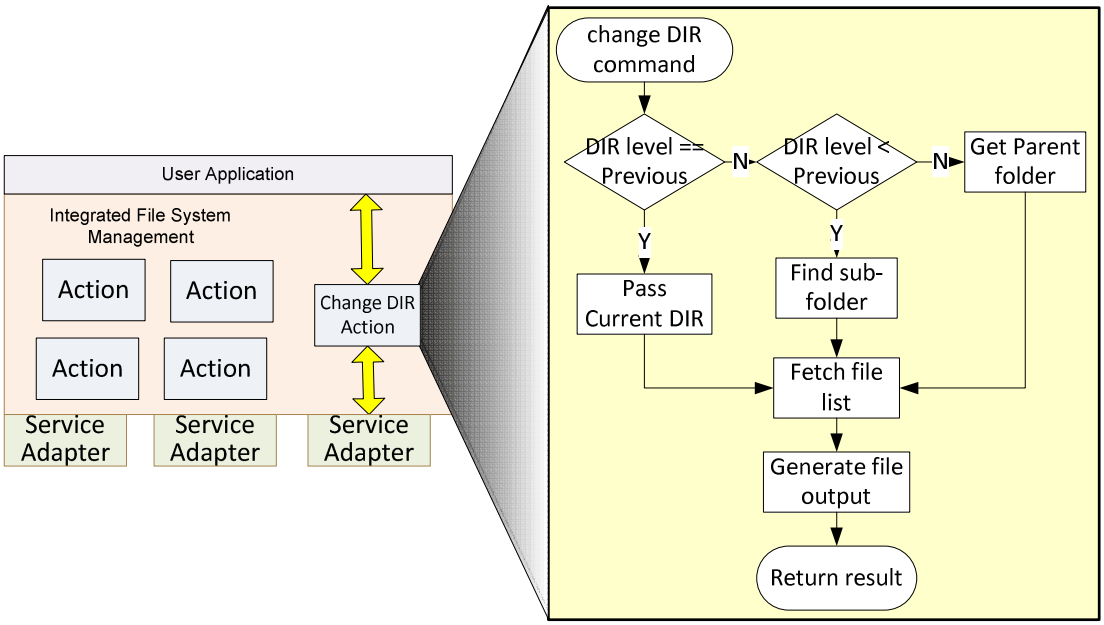

圖 16 Action 元件的內部邏輯

同一個 Action Command 所處理檔案操作的流程是相同的，但 Action 依照內部邏輯處理過程當中必須存放 Action Data 以區分流程中不同的狀態，例如：在切換目錄時，必須要先從 Action Data 中取得目前所在的目錄，如此才知道往上一層目錄切換時必須傳送何種指令給服務，這些 Action Data 必須依照不同狀態儲存在 Action 中，所以同一個 Action 可能有數個 Action Data 存在其中，另一方面，Action 在處理過程當中，必須要參考 Metadata 才能完成虛擬檔案的操作，這時 Action 就必須與 Metadata storage 相互溝通，Metadata storage 在 IFMS 中負責存放 metadata 的資料中心。

在 UFP 中，所有的 resource 都以檔案呈現，但是在網路上，不是所有的資源都是以檔案的形式呈現給使用者，很多的網路儲存服務的資源事實上是不能執行檔案操作命令，因此將網路上的實體資源與系統中虛擬檔案做切割變得十分重要**[6]**，而在 UFP 內部就是透過虛擬檔案(Virtual File)的方式將資源虛擬成 UFP 中的檔案，Virtual File 與真實資源的對應關係都儲存在 Metadata 當中，每一個 Metadata 在 UFP 中就代表一個虛擬檔案，當使用者所操作的對象為虛擬檔案時，Action 會對儲存在 Metadata storage 中的 Metadata 進行操作，如果使用者要對虛擬資料進行存取時，Action 必須先從 Metadata storage 中取出該虛擬檔案的真實資訊，再針對真實資源進行存取，因此不是所有的檔案都需要建立 Metadata，只有無法進行檔案操作的資源或分享檔案才需利用 Metadata 做成虛擬檔案，所以 Metadata storage 事實上可以在 UFP 中提供 logic view 與 physical resource 對應的關係。

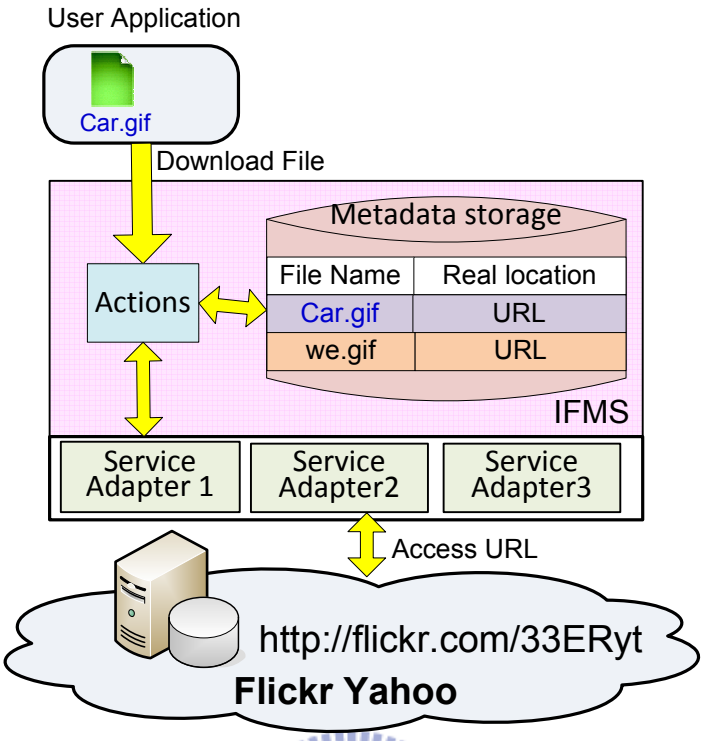

圖 17 UFP 虛擬檔案與 Metadata

當 Action 要對實際服務資源操作時，必須透過 Service Adapter 將命令轉換成服務的命令，但不同的服務需要不同的 Service Adapter，Action 要如何知道需要取用哪一種 Service Adapter? 從 User Application 的設計中，我們知道每一個 UI 上的資料夾就是一種服務，不同的服務有不同的資料夾，當使用者點選某個資料夾時，Action 就知道使用者需要使用哪一種資源，如此 Action 就可以尋找相對應的 Service Adapter 進行操作即可，由此可知，在 UFP 中必須要有元件紀錄服務與 Service Adapter 的對應關係，透過這個對應的關係，Action 可以正確的選取 Service Adapter，在 IFMS 中我們使用 Configuration File 來紀錄此對應關係，Configuration File 是以檔案的形式存在，因為 Service Adapter 與服務的關係不會經常變動，比較適合存放於檔案中，另外在選取 Service Adapter 時，也必須給予 Service Adapter 該服務相關認證資訊，因此我們設計也將此資訊紀錄在 Configuration File 中。

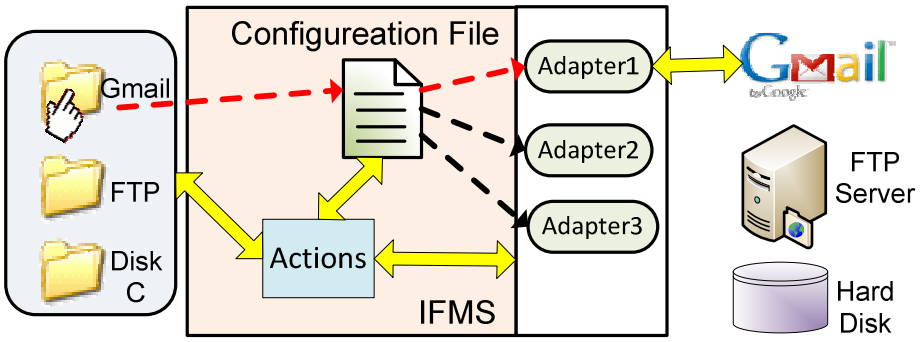

圖 18 服務與 Service Adapter 對應關係

4.2.2 Service Adapter

Service Adapter 是與服務溝通得主要橋樑，它同時具備以下兩種功能:

1. Protocol Converter

   Service Adapter 必須要將 Action 所傳送的命令轉換成服務所能接受的 Protocol[11]，例如: 當使用者要檢視在 Gmail 資料夾中存在哪些檔案時，Action 就會選用 Gmail Adapter 並且要求列出檔案，Gmail Adapter 就必須將命令轉換成 IMAP protocol，透過 IMAP 對 Gmail 進行操作，同樣的，如果使用者點選了 FTP 資料夾，代表使用者要列出 FTP Server 中的檔案，所以 Action 必須尋找 FTP Adapter，同樣要求列出檔案，此時 FTP Adapter 便會使用 FTP protocol 對 FTP server 進行操作，因此相同的命令到 Service Adapter 當中都會被轉換成不同的 Protocol 或是命令。

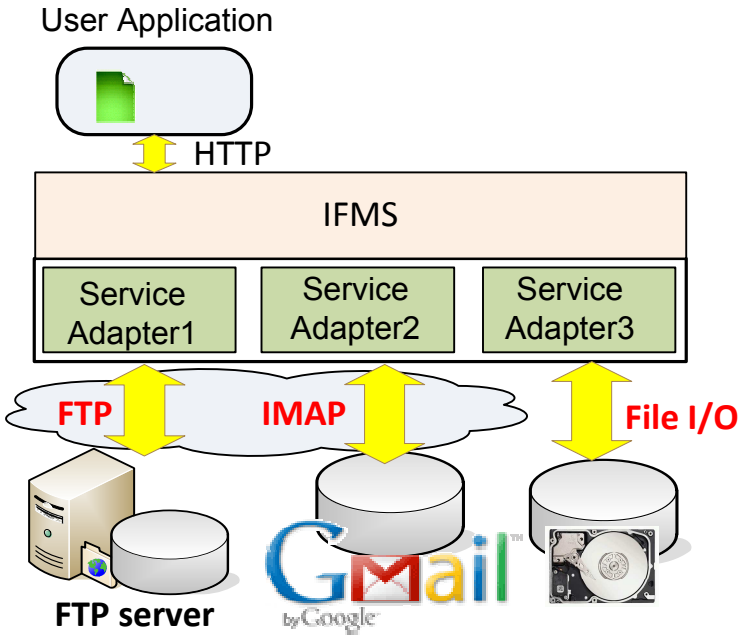

圖 19 透過 Service Adapter 轉換網路儲存服務的 protocol

2. Service Transformer

Service Adapter 同時也是 Service Transform，在 Gmail 的例子當中，我們可以將 Gmail 的郵件服務轉換成可以儲存檔案的資源，當使用者要上載檔案到 Gmail 資料夾時，UFP 會將檔案變成信件的附加檔，在 Gmail 中雖然是以郵件的形式呈現，不過在 UFP 中則是以檔案的形式存在，當使用者要下載檔案時，便是將 Gamil 信件取回，同樣的概念也可以利用在其他的服務，原來的信件服務透過 UFP 轉變成儲存的 resource，因此 Service Adapter 也是一種 Service Transform。

為了要做到不同服務必須對應不同的實作方式，但是對上層 IFMS 又必須提供統一的介面，讓 protocol 或是服務的轉換在 Service Adapter 中完成，Service Adapter 必須提供一個 Common Interface，讓 Action 對 Service Adapter 的命令是一致，此 Common interface 所定義的動作事實上是一些檔案操作的命令，各動作的實作則是在 Service Adapter 中根據不同的服務所完成，透過這一層的 Common interface 的包裝，Service Adapter 將各服務異質性隱藏起來，使得上層 IFMS 甚至 User Application 都不會感覺到在使用不同服務內的檔案。

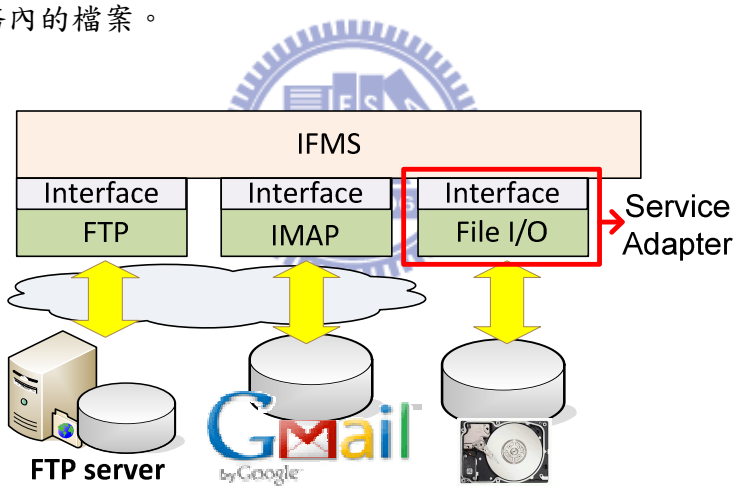

圖 20 Service Adapter 與 Common Interface

由上面的分析可得知不同的服務需要有不同的 Service Adapter，因此要實作 Service Adapter 之前必須要先知道有哪些服務需要整合進 UFP，我們將目前的儲存服務分類成以下四大類:

1. Local Storage

與 UFP 透過硬體所直接連結的儲存裝置，例如:硬碟，USB 拇指碟或是其他的連接裝置，要存取這些儲存服務必須透過作業系統(Operation System) 的介面作存取。

2. Online Storage service with API or standard protocol

在網路上的儲存服務，具有標準 protocol 或是 API 介面可供外界存取，使用者可透過這些介面，開發客製化存取端程式，這類儲存服務如:Cloud storage 或是 FTP server,使用者可透過 Cloud storage 所提供介面或是 API 對 Cloud Storage 中的檔案進行存取，而我們也可以利用 FTP protocol 對 FTP Server 內的檔案進行操作。

3. Online storage service with HTTP and HTML page

在網路上的儲存服務,但是只提供 HTTP protocol 的存取與 HTML 網頁的操作，這類服務主要是設計給瀏覽器使用的儲存服務,使用者可透過瀏覽器進行檔案的操作與存取，服務本身並沒有提供客製化之介面與 API，代表性的服務如: DropIO, Share1T 等免費的線上儲存空間。

4. Standalone storage device with network access

相較於以上第二與三類服務,這類裝置在網路上扮演的是 Client 端的角色,它通常是需求的發出者,而不是服務的提供者,它本身具有網路能力與儲存裝置但卻沒有提供外界任何的檔案存取介面，這類裝置服務列如: 手機，使用者的筆記型電腦。

針對這四類需要整合的儲存服務，我們必須實作不同的 Service Adapter，透過標準介面或是作業系統所提供的檔案系統 API，我們可以很簡單的實作出第一，二類服務，針對第三與第四類服務，我們必須設計特別的 Adapter 或是額外的元件才能讓這兩類的服務整合進 UFP 中，以下就針對這兩類服務的 Adapter 設計作說明。

- 第三類服務的 Adapter

由於此類服務是設計給瀏覽器所使用，因此我們設計在 Service Adapter 當中實作一個 HTTP Client，讓此 HTTP Client 模擬瀏覽器存取檔案過程，透過 HTTP Post 與 Get 對檔案進行上載與下傳，若需要對檔案進行刪除，更改檔名等動作，就必須對 Metadata 進行操作，而非 resource 本身，此類 Adapter 在使用上還是有所限制，若是有認證圖片、Flash 或 script 所組成的上下傳的服務，就無法使用。

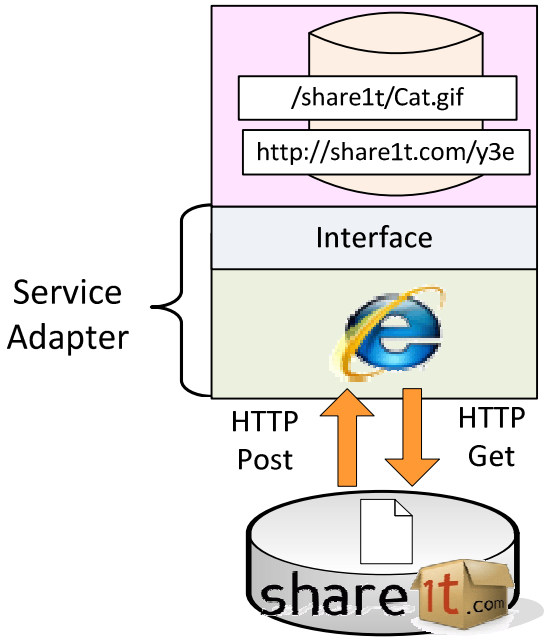

圖 21 第三類服務之 Service Adapter

- 第四類服務之 Adapter:

此類服務在遠端的裝置上沒有檔案存取服務，因此在設計上我們必須透過一個在遠端的裝置上的 Agent 代替 UFP 存取資源，我們稱此 Agent 為 IFMS Agent，它接受 UFP 之命令存取檔案，IFMS Agent 是由 UFP 所發佈或是從 UFP 下載到裝置中，它必須具有以下兩大特性:

1. IFMS Agent 必須是一個 light weight 的程式，它可透過網路發佈或是下載到遠端的裝置，它不需要繁複的安裝過程或是耗費大量資源就可執行。
2. IFMS Agent 必須具有存取遠端裝置檔案系統的能力，如此在接收 Service Adapter 的命令後，才可以做出相對應的檔案操作。

IFMS Agent 在遠端裝置上透過網路與 UFP 平台建立溝通的環境，讓原本沒有提供檔案存取服務的裝置也具有檔案服務的能力，IFMS Agent 與 UFP 溝通也是透過特定之 Service Adapter 進行命令與結果的傳送，讓 UFP 可透過 Service Adapter 與裝置整合，使用此種方法之好處是透過 IFMS Agent 可將第四類的網路儲存資源很容易的整合進 UFP 中，例如:使用者要拿取手機中的相片時，必須使用傳輸線或是藍芽裝置與手機連線，再透過連線軟體才可取出，其中傳輸線的準備、藍芽裝置的配對與連線軟體的安裝，對使用者來說是需要經過一番功夫才能達成，完成這些項目才可存取到手機中的相片，若是使用 IFMS Agent，使用者可用手機連上特定 UFP 所開放的 URL 就可下載 IFMS Agent，下載後直接安裝並且連上 UFP 系統，使用者即可從 User Application 端發現手機的目錄夾，

並且從資料夾中存取到相片。

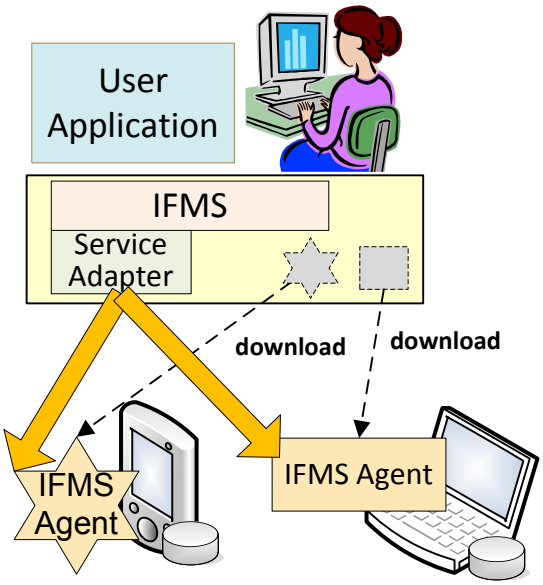

圖 22 第四類服務與 IFMS Agent

4.3 UFP 檔案分享機制

在 UFP 的分享機制中，我們設計在各 UFP 之間可以彼此分享檔案，在分享檔案之前，彼此的 UFP 必須先建立信賴的連線，有了此信賴的關係，才有接下來的分享機制，在整個機制中，我們必須先定義兩種角色，分別是 UFP Client 與 UFP Server。

1. UFP Client: 發出檔案請求的 UFP 系統
2. UFP Server: 接收並處理檔案請求的 UFP 系統

當使用者公開發佈一個分享的檔案時，系統會將此紀錄存入 Metadata storage 當中，當 UFP Client 想瀏覽 UFP Server 所分享出來的檔案時，在 UFP Server 中的 Action 會到 Metadata storage 中取出使用者所分享出來的檔案。

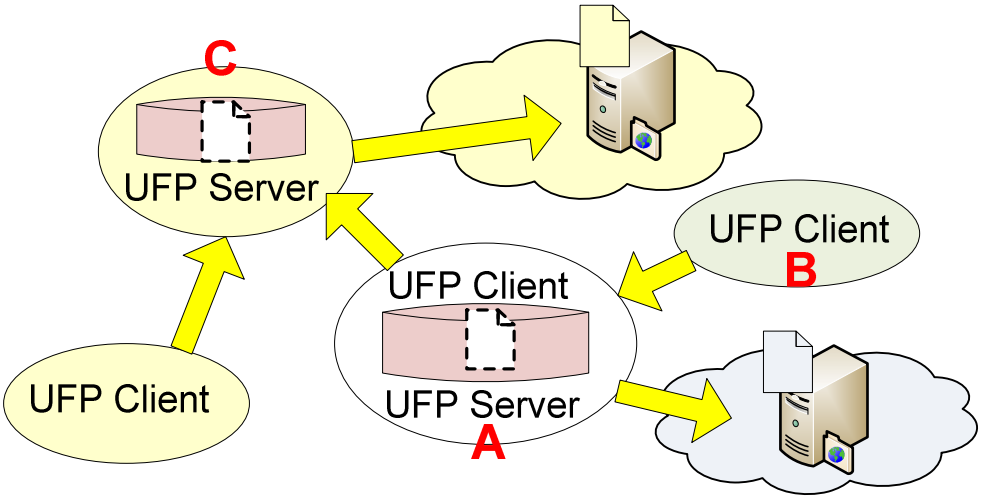

圖 23 UFP 的分享角色與機制

圖 23 中的 UFP A 同時扮演 UFP Server 與 UFP Client 的角色，當 UFP Client B 要存取 A 中的分享檔案時，UFP A 就扮演 UFP Server 的角色，當 UFP A 的使用者要存取 UFP C 的檔案時，它就是扮演 UFP Client 的角色，UFP 會將分享檔案的資訊儲存在 Metadata 中，但實際的 resource 還是存在網路上的服務，基於安全的考量，UFP Client 對所分享出來的檔案只具有讀取的權限，UFP Client 不能對分享檔案進行修改或是刪除。

因此到目前為止，Metadata storage 中會儲存兩樣虛擬檔案的資訊，一是 virtual files，一是 shared files，雖然兩種資訊都存在 Metadata storage 中，但使用目的不同，彼此不會互相使用各自資源。

4.3.1 認證問題

由於真實的 Resource 還存在於網路上的服務中，所以當 UFP Client 要存取 UFP Server 所分享的檔案時，會遇到服務對 UFP Client 認證的問題，在我們的設計中，UFP Client 存取分享檔案時，是透過信任的連線到 UFP Server 中，利用 UFP Server 將網路上所存放在服務端的 Resource 取回，再傳送給 UFP Client，在 UFP Server 端，使用者已經將各服務認證方式紀錄在 Configuration File 當中，所以當 UFP Client 的使用者點選分享檔案時，其存取步驟如下所述:

1. UFP Client 之 User Application 透過 XML over HTTP 發送命令給 IFMS。
2. UFP Client 中之 IFMS 發配命令給 Action。
3. Action 選擇 Service Adapter 進行服務存取。
4. Service Adapter 透過網路，對 UFP Server 之 IFMS 發送命令，IFMS 檢查是否為 UFP 所信任 Client 端。
5. UFP Server 接受後，再交由 Server 端 Action 處理。

6. Server 端 Action 存取 Metadata Storage 中的分享檔案資料，找出 Resource 的真實所在位置。
7. Action 再根據真實所在位置到設定檔中找尋所需要服務之 Service Adapter。
8. 在取得 Service Adapter 的同時， Action 一並從 Configuration File 中取出該服務所需認證資訊。
8. 透過適當的 Service Adapter 到服務中存取檔案，Adapter 存取檔案時也會通過該服務的認證
9. 取得檔案後，由 UFP Server 傳遞給 UFP Client
10. 再由 UFP Client 之 IFMS 傳送給 User Application，並且呈現在使用者面前。

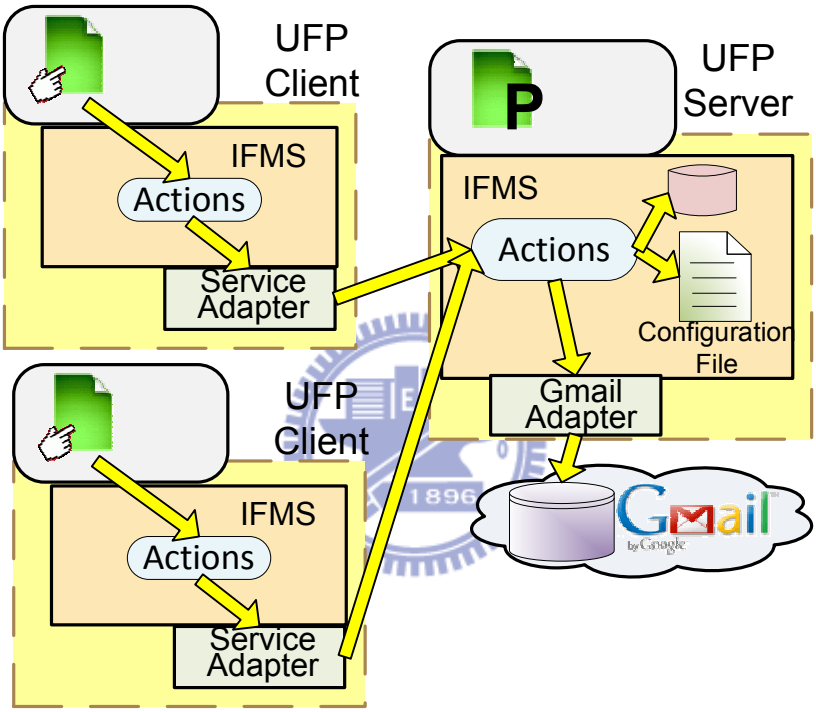

圖 24 分享檔案之存取流程

透過這種存取或是分享機制好處是:

1. 整個存取或是信賴過程當中，在 UFP Client 端只有一次認證需求，那就是在建立信連線時的認證，建立信任之後，要取得 UFP Server 端所分享的檔案時，皆不需再作額外的認證請求。

2. UFP Client 端存取 UFP Server 端所分享服務內的檔案時，不需再申請該服務的帳號，另一方面，存取的同時，UFP Client 也不需察覺到這些檔案是在哪些網路儲存服務中，存取分享檔案就如同存取本地端檔案一般。

4.3.2 即時的檔案分享資訊

當你的朋友分享一個檔案時，你可以及時得知，並且進行存取，而不是定時的瀏覽朋友的資訊或是資料夾才會知道他分享了哪些檔案。

要將分享資訊由被動的得知改成主動告知，在分享機制當中必須要有 PUSH 的機制存在，事件由 UFP Server 中的 Action 觸發，當使用者分享檔案時，負責處理的 Action 將此事件透過網路發佈出來，UFP Client 之 IFMS 就會接收到此資訊，再由接收處理之 Action 將此訊息 Push 給 User Application。

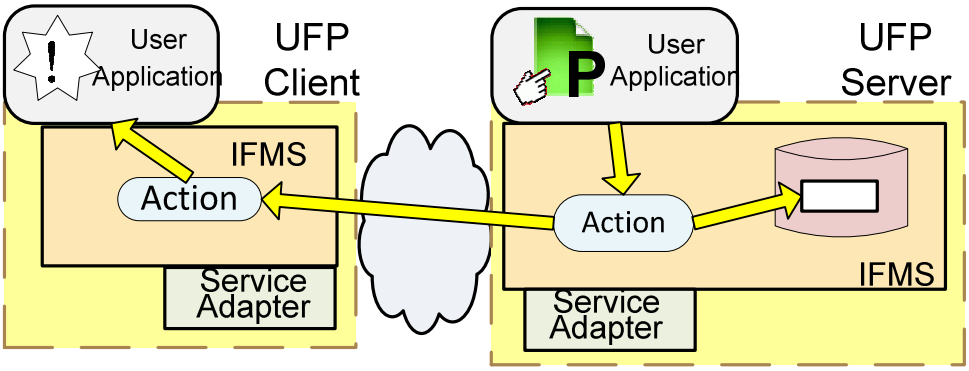

圖 25 主動分享訊息之發佈

五、實作與成果展示

5.1 系統實作

系統主要區分為兩大元件，IFMS 與 Service Adapter，另外在 User Application 部分，我們使用 Yahoo Widget[7]來實作此 User Application，會使用 Widget 成為使用者端的 User Application，其原因可列為以下四點:

1. Widget 的 UI 介面具有高度的客制化能力，從視窗到按鍵都可以自行訂製，透過客制化的介面開發，能夠讓使用者介面具備完整的檔案操作，另一方面 Widget UI 具有很多特殊的 UI 效果，可提供滑鼠拖拉操作並且與桌面程式結合，讓使用者可以更直覺的操作檔案。

2. Widget 可停留在使用者桌面，不需要經常開啟或是關閉程式，Widget 很適合顯示及時的資訊，因為它能馬上顯示在使用者的桌面，讓使用者及時接收這些訊息。

3. Widget 是一種輕量級的程式，Widget 通常只為特定功能或是目標所開發的程式，它不需要消耗大量的系統資源，也不需要安裝或是建置太多的元件就能完成某些工作，很適合當成個人檔案平台的使用者介面程式。

4. Widget 的執行是透過 widget engine，而通常 widget engine 有提供大量且多樣化的 API 可供開發者開發不同的 Widget 程式， 透過這些 API 可以完整建構使用者對檔案的操作以及其他不同的功能。

UFP 系統本身則架構在 Tomcat Web application 之上，整個系統都使用 Java 開發完成。

5.1.1 Widget 與 IFMS 之介面

Widget 與 IFMS 是透過 XML over HTTP 作命令請求與結果回傳，Widget 透過傳送 URL 參數的方式指定 UI 的動作，請求進入 IFMS 之後，透過 Dispatcher 將請求交給對應的 Action 進行處理，回應結果可分成三大類:

1. 檔案列表回應: 通常使用在取得服務列表或是檔案列表時所回傳的結果，以 fs tag 為一筆檔案，其內容屬性為

Attribute Name	Description
name	檔案名稱
fstat	檔案種類 1 = 分享的資料夾 2 = 分享的檔案 3 = 未分享的資料夾 4 = 未分享的檔案

表 1 XML 資料回傳內容

以下為其範例

```
<res>
 <fs-list>
  <fs name='0990503.rar' fstat='4'/>
  <fs name='vnc-4_1_2-x86_win32_viewer.exe' fstat='4'/>
  <fs name='y24022602.rm' fstat='4'/>
  <fs name='table.xls' fstat='4'/>
 </fs-list>
</res>
```

2. 狀態回應: IFMS 回應所執行的命令結果，其中包含錯誤訊息或是 OK

以下為其範例

```
<res>format is not correct!</res>
```

3.檔案上下傳: 檔案上下傳回應的結果非 XML 文件，而是依照 HTTP file upload[8]與 download 的協定進行。

5.1.2 Actions

Action 為負責處理 User Application 請求之邏輯單元，因此我們在實作上將每種 Action 都設定為一種 Action Class，每個 Action Class 都繼承 MyAction Interface 使得 Action 都具有 execute function，各 Action Class 的邏輯就實作在於此 function 當中，IFMS 的 dispatch function 在產生 Action 物件後就直接呼叫每個 Action 之 execute function，而 Action Data 則剛好為每個 Action 物件的 data member，不同 Action 物件可以擁有不同的 data member，但所使用的 Method 則是相同，Action 物件回傳結果為 XML 文件，Action 物件必須將處理結果組成 XML 並回傳給 User Application，MyAction 之介面如下

```
public interface MyAction
{
    public String execute();
```

}

為了完成基本的檔案操作，我們在 IFMS 中實作了以下 Action.

Action Name	Description
InitAction	列出起始可使用之服務，從 Configuration 檔中讀取出來
AddSrv	增加一個 Trusted UFP 到可用服務中，加入後便可瀏覽該 UFP 所公開分享的檔案
CdAction	變換目錄
SearchAction	搜尋本地端服務內之檔案
DelAction	刪除檔案
UploadAction	上載檔案
DownloadAction	下載檔案
PirvateAction	將公開分享檔案設定為不公開之狀態
PublicAction	公開分享檔案
RDLAction	從 Trusted UFP 中取回分享的檔案
RemoteSearchAction	搜尋 Trusted UFP 中的分享檔案
RemoteListAction	瀏覽 Trusted UFP 中的分享檔案
NotifyAction	接收 Trusted UFP 所傳送之 event 並 PUSH 給 Widget

表 2 UFP 中 Action 動作

5.1.3 Common Interface

Common Interface 是存在於 Service Adapter 與 IFMS 之間的介面，在實作上我們將此定義為 Java 的 Interface 類別，由於檔案的架構通常是以階層式呈現，因此我們依照此架構將 Interface 分為 FileSystem 與 jFileObject 兩種 Interface。

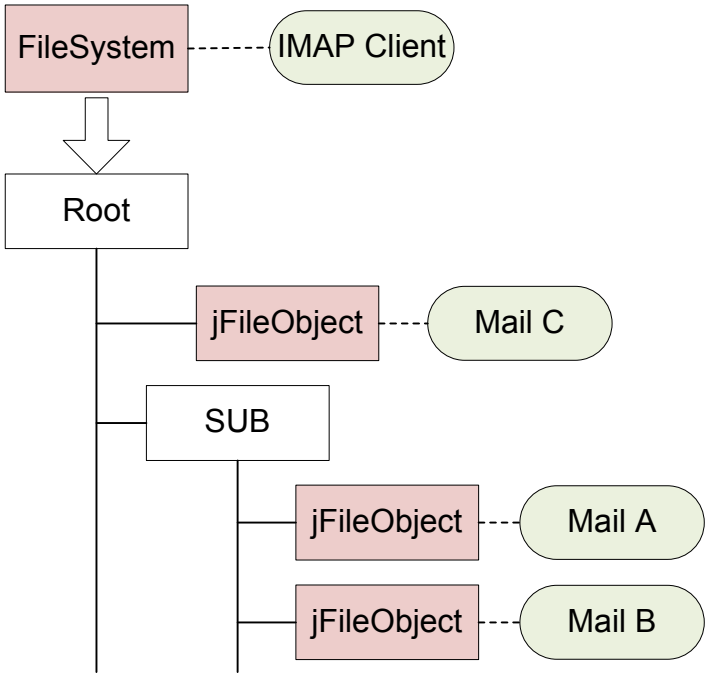

圖 26 Common Interface 中的階層架構

FileSystem 代表整個檔案系統的介面，負責取得整個檔案目錄架構，包含開啟關閉整個檔案系統，以及取得該目錄內的檔案物件，其擁有 function 如下:

Function Name	Return value	Input parameter	Description
close	N/A	N/A	關閉檔案系統，實作上則是必須與服務結束連線
getFileList	jFileObejct []	String path	取得該目錄內的檔案物件，並須輸入需要取得的路徑
getRootFile	jFileObject	N/A	取得第一階層目錄的檔案物件
putFile	N/A	String filename	上傳檔案，若不能取得 inputStream 則使用此 Method
putFile	N/A	InputStream fin, String name	上傳檔案，若可取得 inputstream 則使用此 Method
findFile	List of jFileObject	String filename	尋找檔案系統中之特定檔案

表 3 FileSystem 介面中的 function

jFileObject 則是代表每個檔案的物件，在經由 FileSystem 介面取得目錄內的檔案物件後，IFMS 可透過物件上的 Method 進行物件的操作，其中包含之 function 如下:

Function Name	Return value	Input parameter	Description
delete	N/A	N/A	刪除檔案
rename	N/A	String filename	更改檔名
getFileName	String filename	N/A	取得檔案名稱
listFile	jFileObject []	N/A	若是該檔案物件為目錄時,取得該目錄內的檔案物件
getInputStream	InputStream	N/A	取得 InputStream，準備對檔案進行下載
getOutputStream	OutputStream	N/A	取得 OutputStream，準備對檔案進行上載
isFolder	Boolean	N/A	檢驗該檔案物件是否為目錄
getParent	jFileObject	N/A	取的該檔案物件之上層目錄

表 4 jFileObject 介面中的 function

5.1.4 Service Adapter

根據設計，Service Adapter 必須實作 common interface 用以完成不同服務溝通的方法，而不同的服務必須要有不同的 Service Adapter 做聯繫，在系統中我們分別實作四種 Service Adapter，分別代表四種不同服務類型。

1. Local Storage

我們使用 Apache common virtual file system library 實作此類 Service Adapter，透過 Apache VFS library，我們可以存取到不只是 local 儲存裝置，還有 FTP 內的檔案也可經由 VFS 存取得到。

2. Online Storage service with API or standard protocol

我們選定 Gmail 為此類服務之整合目標，Gmial 本身為信件服務，但有提供標準的 IMAP 介面，要與 Gmial 溝通必須要使用 IMAP Client 為實作方法，每次上傳檔案，即是傳送一封信件並且夾帶所要上傳的檔案，信件的主旨即是該檔放在 Gmial 資料夾內的位置與檔名,若要下載檔案或是刪除檔案,在此 Gmail Adapter 中的動作也是下載信件附檔與刪除信件。

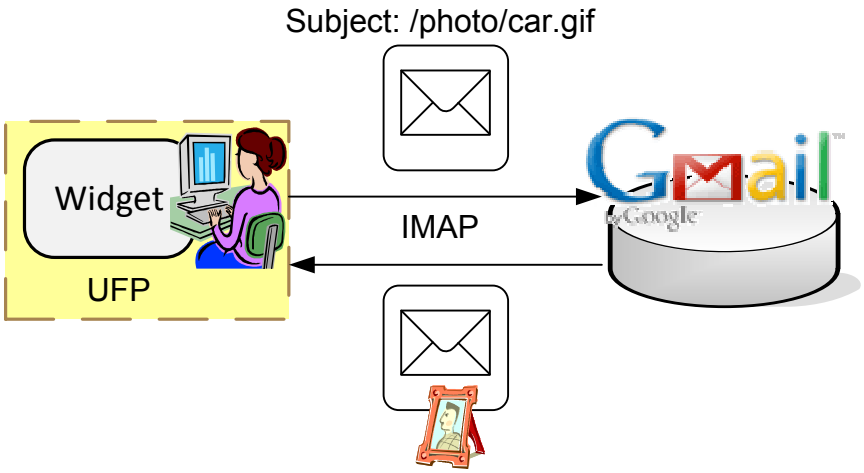

圖 27 Gmail Adapter 之實作

其 java mail library [13]與 common interface 之主要對應動作如下:

Common Interface	Java main library
FileSystem	Gmail 連線並且尋找 INBOX 信件資料夾
Close	關閉 Gmail 信件資料夾
getFileList	在信件資料夾中搜尋以輸入路徑為主題的信件
putFile	將所指定之上傳檔案讀入 buffer 並且將其附加在信件中寄出到 Gmail。
getFileName	回傳該信件主旨
listFile	搜尋指定主題之信件
getInputStream	取得信件檔案附件之 File stream
delete	Purge 信件

表 5 Java mail 與 common interface 之對應關係

3. Online storage service with HTTP and HTML page

在此類服務當中，我們選定 Anyhub(http://anyhub.net)為此類服務的整合目標，Anyhub 具有簡潔的畫面，沒有 flash 與圖案認證，相對於 Service Adapter 的實作來說會比較方便，與此類服務溝通主要發生在檔案上傳與下載，其他的檔案操作皆是在 Metadata Storage 中完成，所以要實作此類服務之 Service Adapter，首先必須先了解服務的溝通 message flow。

透過 Ethereal 等工具，觀察服務與瀏覽器之間的溝通可發現，要上傳檔案時必須先利用 HTTP File Upload 到 Http://anyhub.net/upload 之 URL，上傳成功之後，系統會使用 HTML 回傳檔案所在之 URL，此時 UFP 便可 parsing 回傳之 HTML 文件，並從其中找出檔案所在 URL 之位置，組合成 metadata 並存入

Metadata Storage 中。

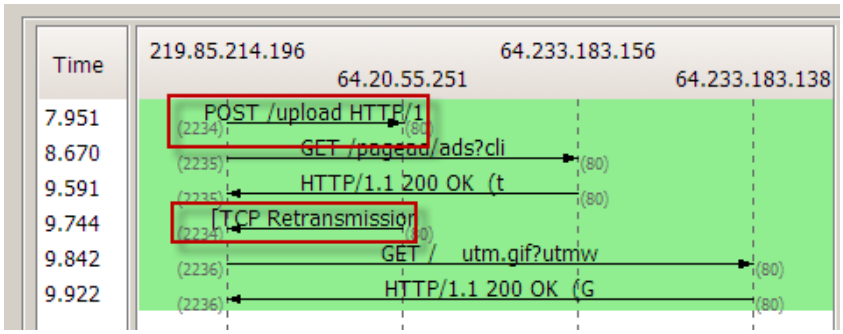

圖 28 服務之檔案上傳流程

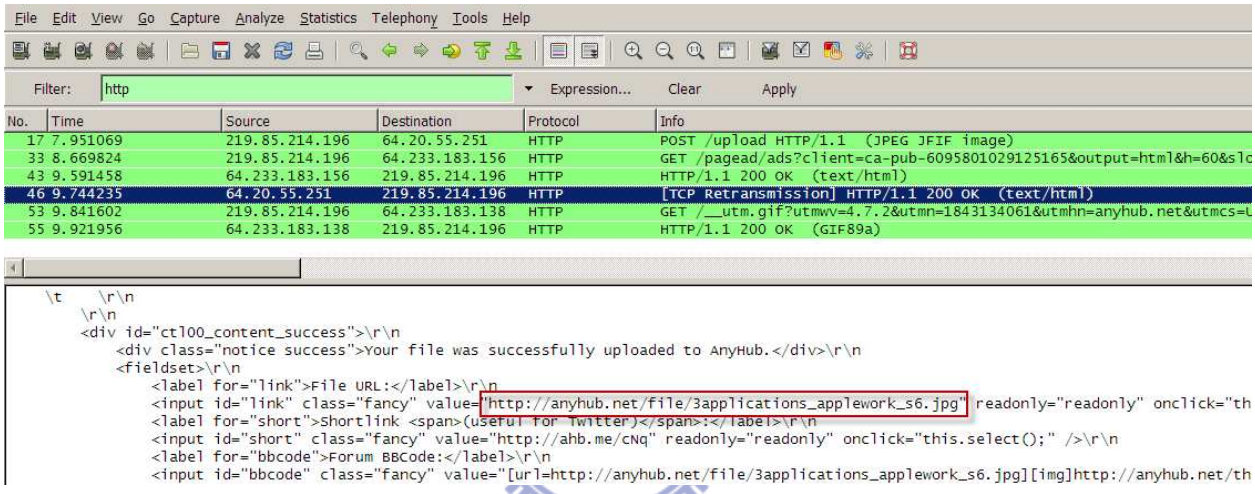

圖 29 從 HTML 中找出下載資源之 URL

當需要下載檔案時，並須先從 Metadata Storage 中取出該檔之 URL，並且利用 HTTP GET 請求檔案，此時系統便會依照 HTTP File download protocol 將檔案回傳。

圖 30 檔案下載

根據這些觀察結果，我們可以依照此 message flow 來設計在 Adapter 中 HTTP Client 的行為。

4. Standalone storage device with network access

我們選定 standalone PC 為此類整合目標，在 IFMS Agent 的實作方面，我們選擇 Java Applet[9]實作 IFMS Agent，透過遠端 PC 的瀏覽器存取 UFP 中特定的 URL 下載 IFMS Agent applet 並安裝在瀏覽器中，在 Applet 當中必需要有 listen socket，透過此 Applet 在遠端 PC 接收 Service Adapter 所傳送的檔案命令，我們就可以將這台 PC 整合進 UFP 中。

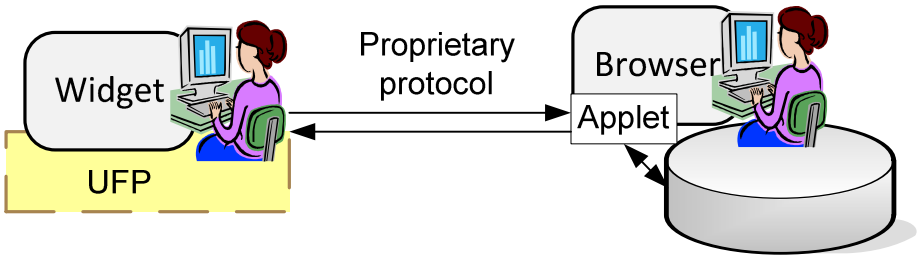

圖 31 Mobile Agent in Applet

使用 Applet 的好處是，Applet 可以經由網路發送或是下載，並且在不同的平台中只要有 JVM 就可以執行，雖然 Applet 本身在 Sandbox 中並沒有操作檔案與網路連線能力，但透過 Signed Applet 的安全控制，Applet 本身就可變成遠端的小程式，完全操控遠端的 PC，此時 Applet 就如同 Agent 般代替 UFP 在遠端 PC 操作檔案。

5.1.5 Configuration File 設定檔

在 UFP 系統中，我們必須有 Service Adapter 與資料夾的對應關係，在實作上，我們將此檔案設計為一 XML 格式的檔案，系統必須設定相對應的服務與 Service Adapter 於此檔案中，另一方面，也必須將此服務會使用到的認證資訊紀錄於此，檔案格式如下:

```
<fsconfig>
    <fslist>
        <fs name="Folder name in Widget">service
    adapter://security token@service host</fs>
    </fslist>
</fsconfig>
```

在 security token 中，若為 user name/password，則使用 username:password 格式紀錄。

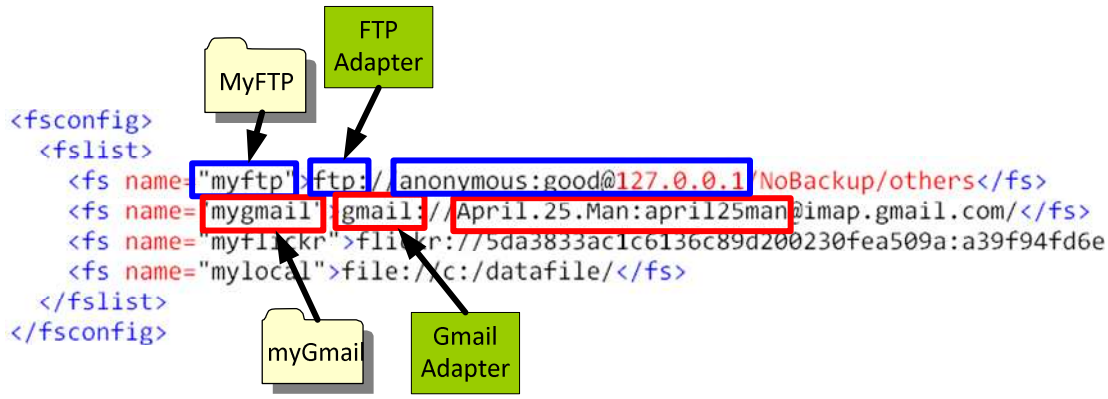

圖 32 UFP 之設定檔

5.1.6 Widget Notify 實作

在實作 Notify 的機制時，UFP Client 之 Notify Action 在接收到 UFP Server 所發布事件後，會透過 HTTP 通知 User Application，但很多 Widget engine 為了提供 light weight 的 Widget 程式，Widget engine 本身並不提供 listening socket 的能力，這使得在實作 PUSH 機制[10]時，我們必須要使用 Long Pooling 的方式，其運作方式如下:

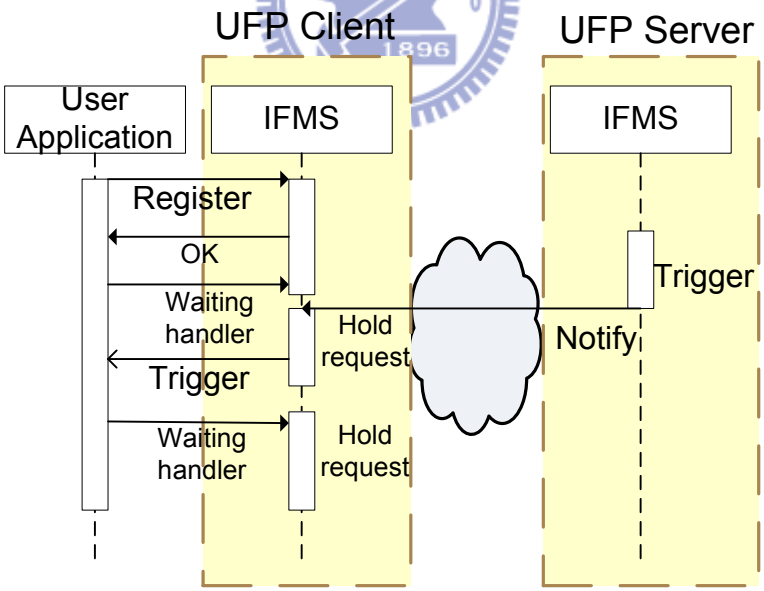

圖 33 UFP 之 PUSH 流程

1. 在 UFP 系統啟始時，每個 UFP 之 Widget 就必須使用非同步的訊息之 Ajax 訊息先對 IFMS 發出註冊 event 的請求。
2. IFMS 回應註冊成功，此時 User Application 再發出 waiting handler 的請求，IFMS 此時不會回應此請求，由於 Ajax 非同步的特性，UFP Client

端不須等待回應可繼續處理使用者介面操作。

3. 當遠端 UFP Server 發佈分享檔案的訊息時，UFP Client 的 Notify Action 接收到訊息後，會回應一開始的 Ajax 請求。
4. 請求從 IFMS 回應後，會觸動 Widget 中非同步訊息的 callback function，讓 event 顯示在 Widget UI 上，使用者就可以立刻接收到訊息。
5. Widget 立刻再發出另一個 waiting handler 之 Ajax 請求，等待另一個事件通知。

User Application 端的 Widget 程式與 IFMS 之間只保持一個 long polling 的連線，所有的 event 皆經由此管道進行對 User Application 的通知，所以不會對 IFMS 系統造成太大的系統負擔。

5.1.7 Message flow

在完成各元件之實作後，各元件之間相互溝通 message flow 如下:

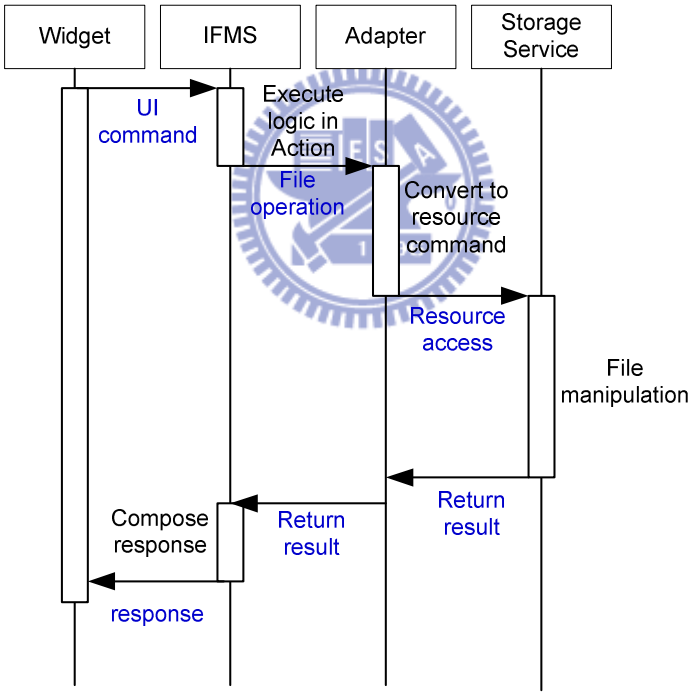

圖 34 檔案基本操作之流程

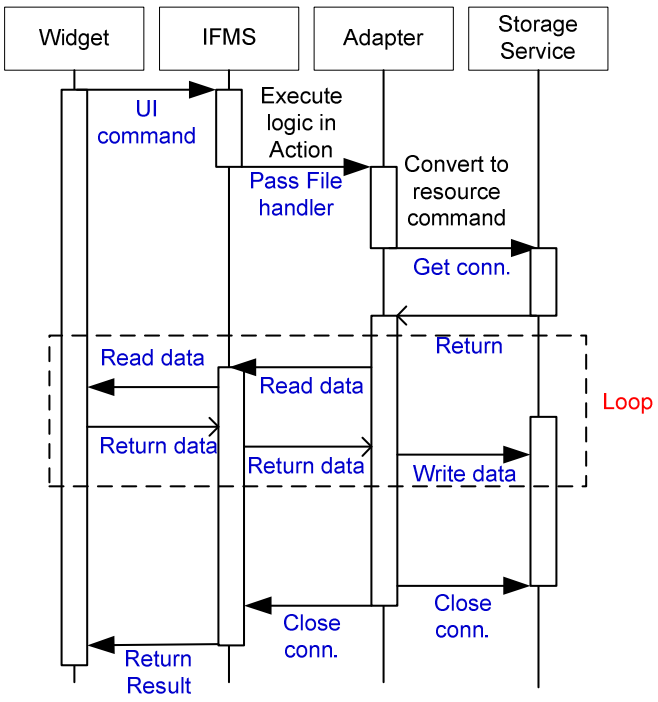

圖 35 檔案下載之基本流程

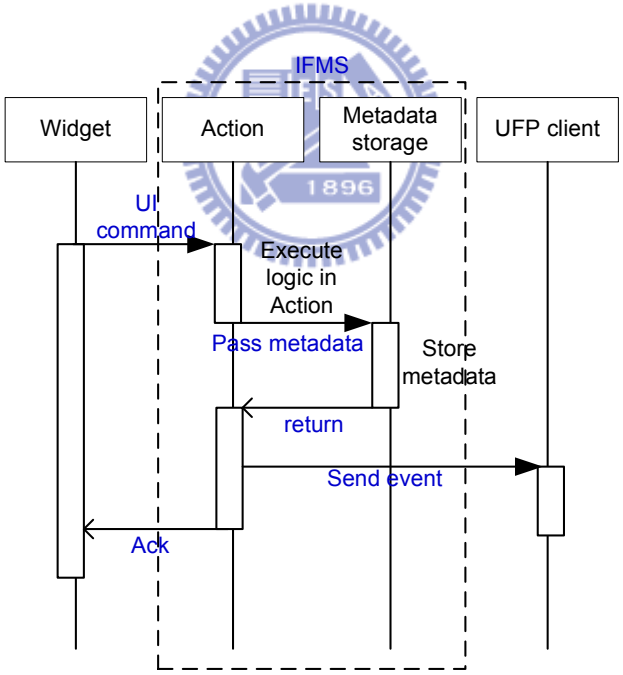

圖 36 分享檔案之流程

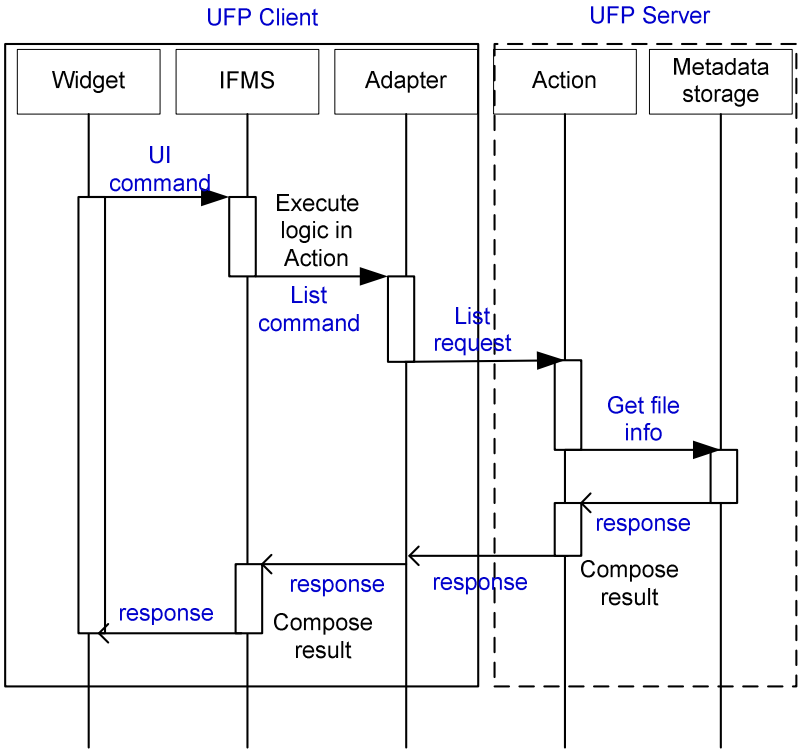

圖 37 瀏覽分享檔案之流程

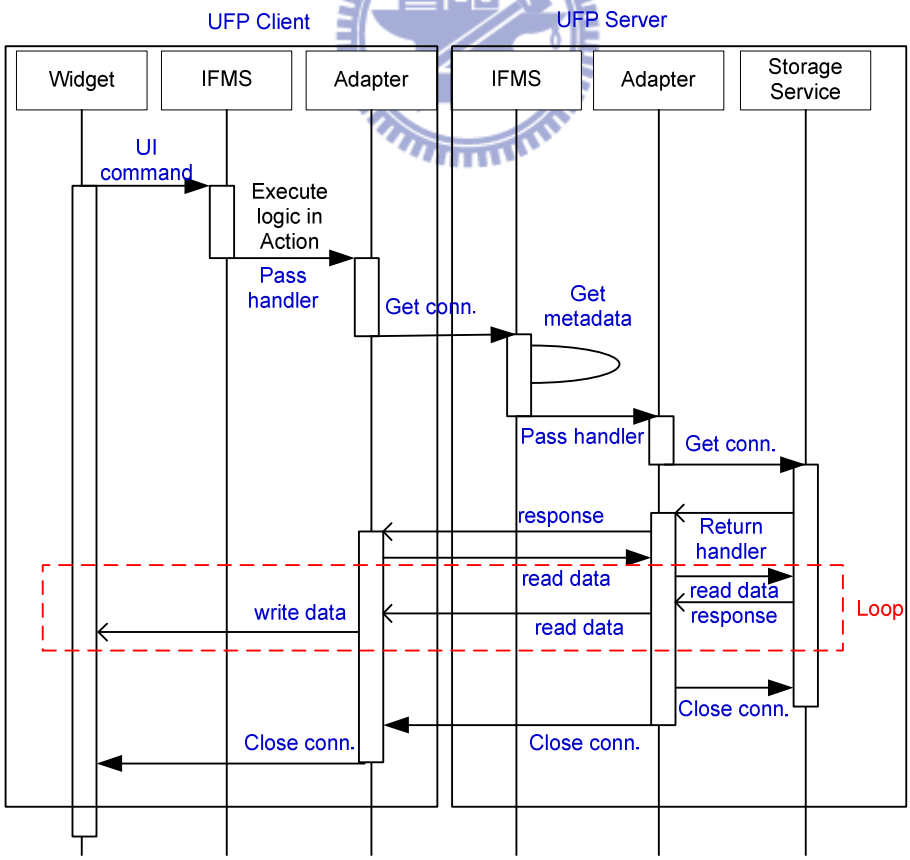

圖 38 下載分享檔案之流程

5.2 成果展示

在本章中，將展示一些系統操作之過程與畫面。

1. 系統開啟，並且將 Widget 安裝完成後之畫面如下圖

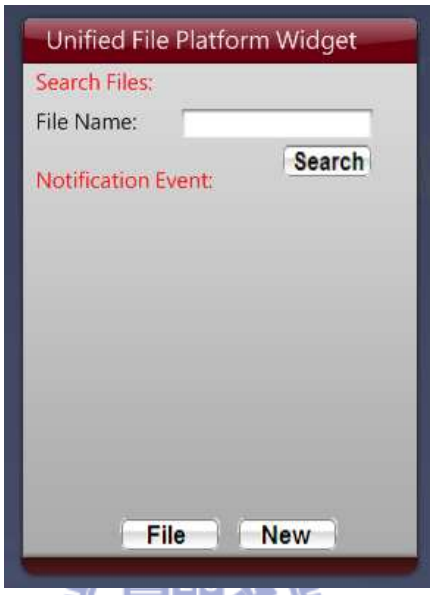

圖 39 基本 Widget 畫面

2. 點選"File"便可開啟檔案資料畫面，使用者可根據需求進入不同的資料夾，每個資料夾就是一種服務，由此圖中可知，本系統目前有 FTP、anyhub、Flicker、Gmail 與本地端磁碟五種服務。

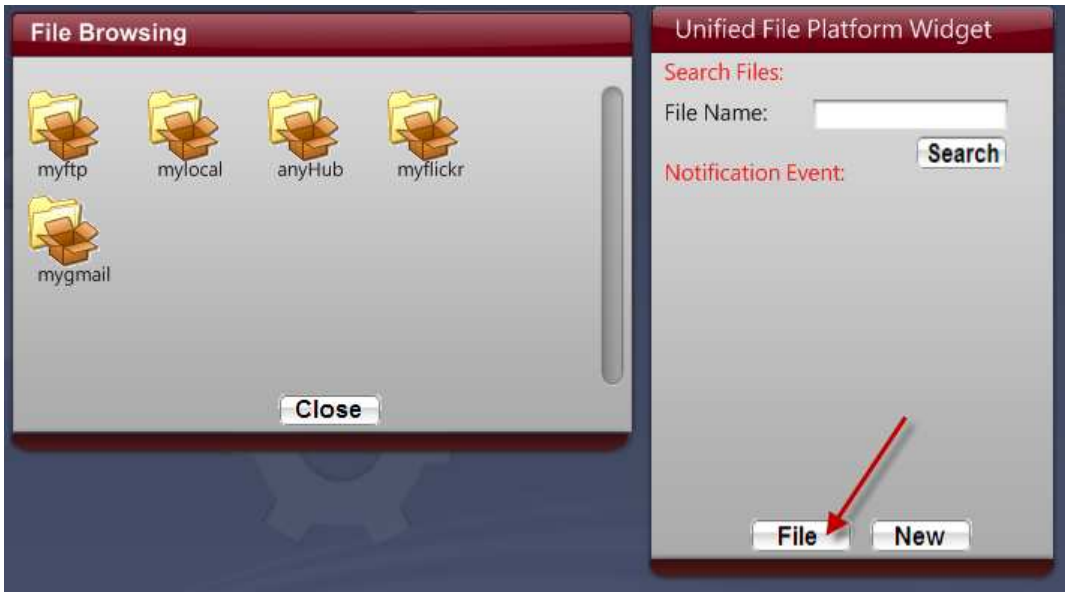

圖 40 開啟服務內容之畫面

3.點選進資料夾後，可見檔案或是子目錄，在檔案上按左鍵，可選擇瀏覽檔案、刪除檔案或是設定分享檔案。

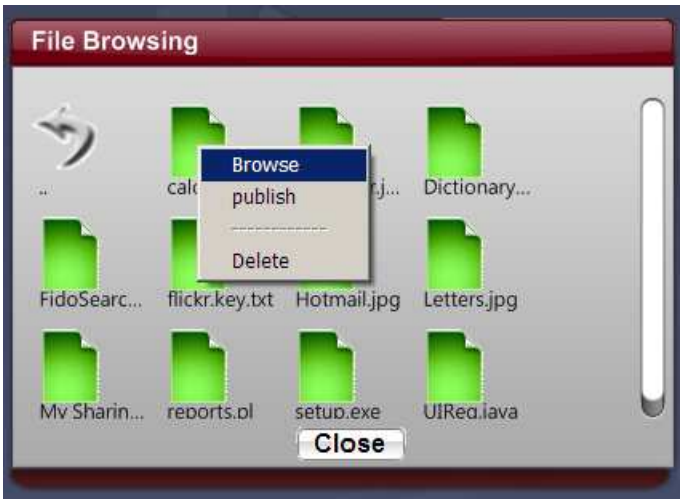

圖 41 設定分享檔案或開啟檔案

4.選擇公開分享檔案後，所分享的檔案會以"P"字母代表該檔案已被分享，可被朋友存取。

圖 42 分享之檔案被標記"P"

5. 在主畫面選取"New"便可新增 Trusted UFP server，將朋友的 UFP 加入，便可互相分享檔案，輸入朋友系統代號、密碼以及 IP address，加入後在檔案瀏覽時會發現新增一個朋友的資料夾。

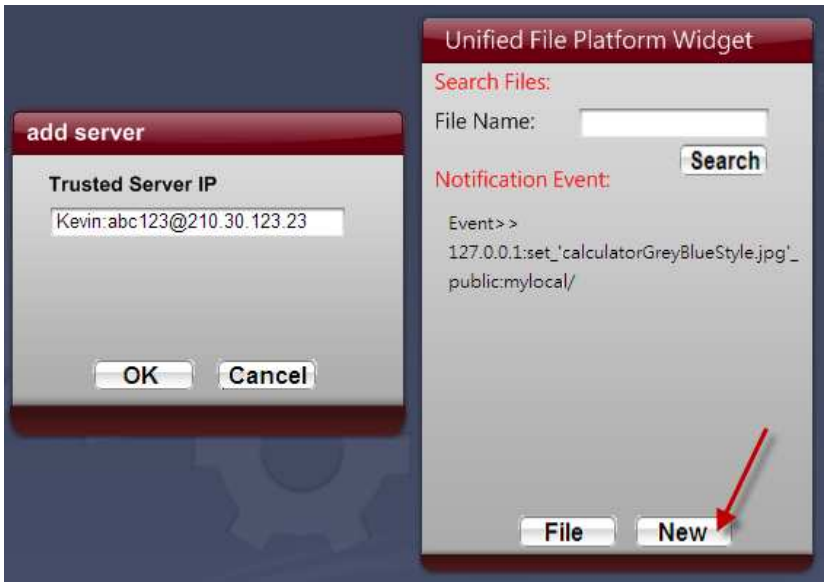

圖 43 建立遠端 UFP 之 Trust Server

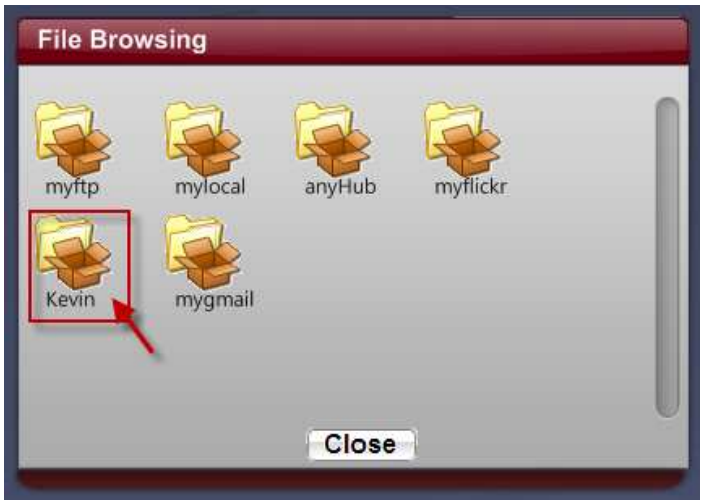

圖 44 新增朋友 UFP 後之資料夾

6.透過信賴關係，點選朋友的資料夾便會看到朋友所分享出來的檔案，如下圖，圖 45 為 UFP Server 分享檔案，圖 46 為 UFP Client 看到所分享出來的檔案。

圖 45 UFP Server 分享檔案

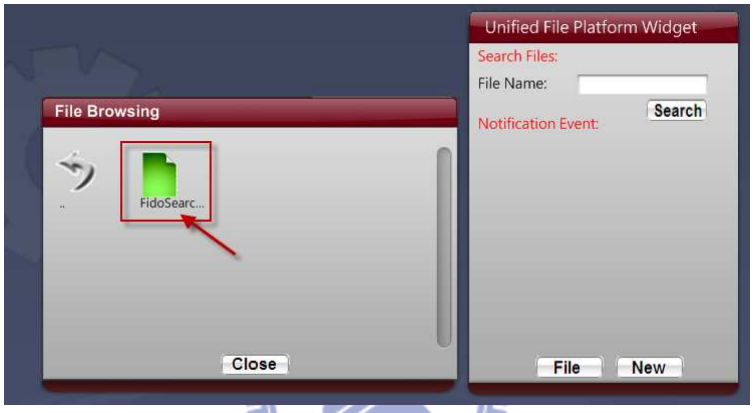

圖 46 UFP Client 存取分享檔案

7. 當朋友分享檔案時，UFP 會即時顯示資訊在 Widget 上。

圖 47 分享事件通知

8. 遠端 Device 可透過 IFSM Agent 的下載與安裝快速的整合進 UFP 當中，圖

48 中的 MobileDisk 即遠端 IFMS Agent 所在的儲存裝置。

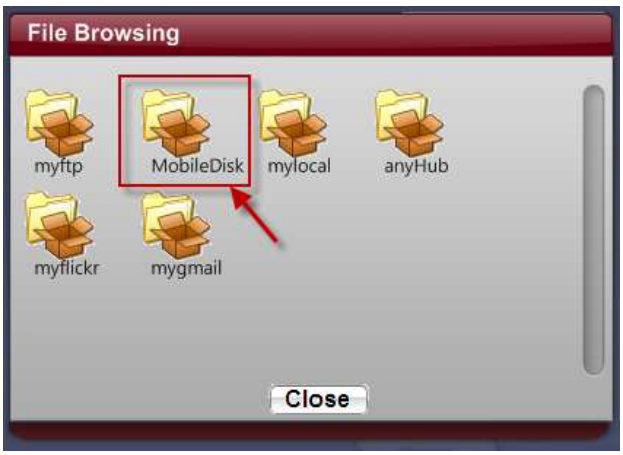

圖 48 透過 IFMS Agent 整合儲存服務

六、系統比較

6.1 功能比較

在本章中我們將 UFP 與現行整合性檔案系統在 User Application Interface、Web storage service 整合與分享功能三方面來比較。

	Unified File Platform (UFP)	***Storage Resource Broker(SRB)***	***Cloud Storage***	***XAM***
User Application Interface	- Customized user application - XML/HTTP	- Customized user application - Web	- Proprietary client - Customized user application	- Customized user application - Binding interface
Storage service integration	- As many as possible - Need adapter module	- Storage system - More effort in building system	- Only vendor's storage - Not free	- Storage media - Need vendor support VIM
Sharing Function	- Between different service - No account needing - Real time event notify	-Different storage system -Passive message notification	- Share with non-member - Passive message notification	- No data sharing mechanism

表 6 UFP 功能比照表

- 在 User Application Interface 方面，幾乎所有的系統有提供 API 介面，可供使用者自行開發 User Application，也可以整合進其他檔案系統，但少部分系統，如:XAM，必須透過 binding 的方式與系統函數連結，無法透過網路存取，所以 User Application 也無法遠端存取系統，UFP 提供 XML/HTTP 介面讓使用者客製化 User Application，也可讓使用者由遠端控制 UFP 平台。

- 在 Storage service 整合方面，雲端服務多以自己所提供的儲存空間為主，對其他儲存服務則無整合能力，SRB 與 XAM 的整合目標則是以儲存設備或是儲存系統為主而不是儲存服務，另外多數檔案整合系統的服務對象是以企業用戶或是多人線上服務為主，而非個人用戶，在資源耗用與系統建構上不如 UFP 簡便，UFP 盡可能的將網路儲存服務整合到 UFP 中，線上儲存服務型態多樣，透過 Service Adapter 將服務整進 UFP 中，但每一種服務就必須要開發相對應的 Adapter 才行。

- 在分享功能方面，多數的系統包括 UFP 都必須是會員或是擁有該系統帳號才

能夠進行分享，但只有 UFP 能做到 transparent 分享能力，使用者不需知道分享的檔案是位於分享者的哪個服務內的檔案，也不需額外的認證或是帳號申請，即可存取到所分享出來的檔案，另一方面，也只有 UFP 能做到即時的分享資訊通知。

6.2 效能比較

操作檔案時系統反應時間直接影響使用者對系統的使用意願，在本章中我們將同樣是屬於整合 Gmail 網路服務的 GmailFS**[16]**與 UFP 做檔案移動的時間效能比較，GmailFS 是架構於 FUSE 上的 userspace filesystem，透過不同大小檔案的移進與移出系統做出兩者操作上的效能差異比較。

系統平台 ：Linux RedHat Enterprise 5, Intel Core 2 Duo P8700 CPU, 1 GB memory,3M/768 VDSL.，測試檔案分別為 1MBytes ~ 5MBytes 之檔案，測試結果如下圖；

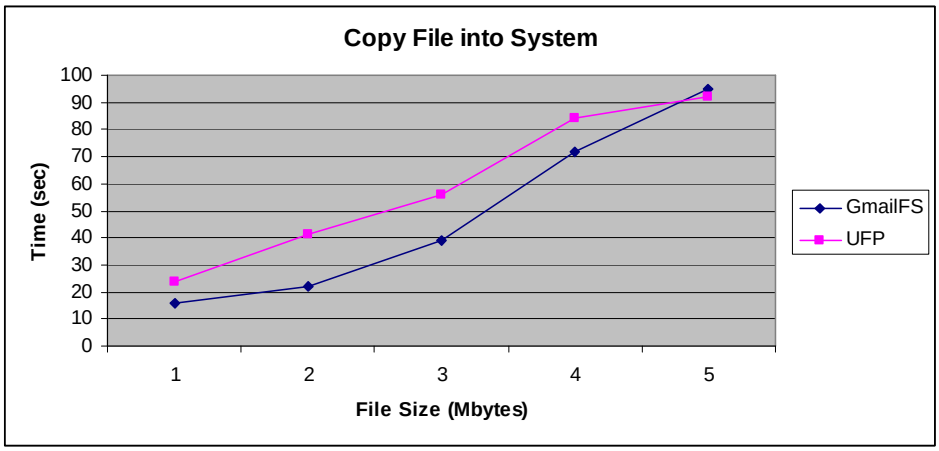

圖 49 檔案上載至 Gmail 服務

圖 49 為複製檔案進入系統，對 Gmail 服務來說為上載檔案到服務當中，由圖中可知，UFP 在檔案上傳至服務所需時間較 FUSE 為多，但兩者差異不大。

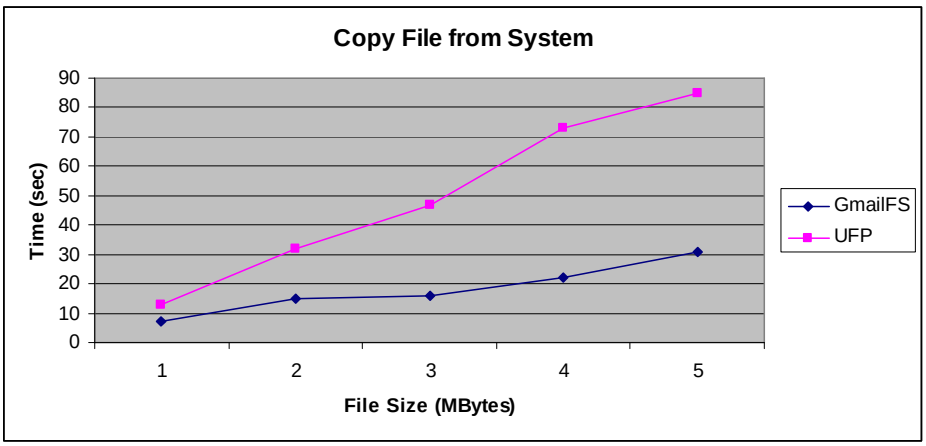

圖 50 檔案從 Gmail 服務下傳

圖 50 為檔案下傳部分，由圖可知 UFP 在檔案下傳部分所需時間遠多於 FUSE，兩者差異最多有近 50 秒的差異。

由以上兩種數據可發現，一般來說 UFP 所需要時間要比 FUSE 時間多，這跟 FUSE 是存在於 Kernel Layer 的 Virtual File System 有直接的關係，再者，在 UFP 的設計上前端 User Application 程式(Widget 介面)與 IFMS 之間還有一層網路介面(XML/HTTP)與 FUSE 的直接使用 Linux File System 介面有所不同，接著我們以 UFP 與 FileZilla 做檔案移動的比較，FileZilla 是具有圖形化的 FTP Client 程式，我們在區域網路內架設 FTP Server 並透過 UFP 與 FileZilla 存取 FTP Server 中的檔案。

系統平台 : Windows XP , Intel Core 2 Duo P8700 CPU, 4 GB memory, 100Mbps Ethernet，測試檔案分別為 25MBytes ~ 100MBytes 之檔案，測試結果如下圖；

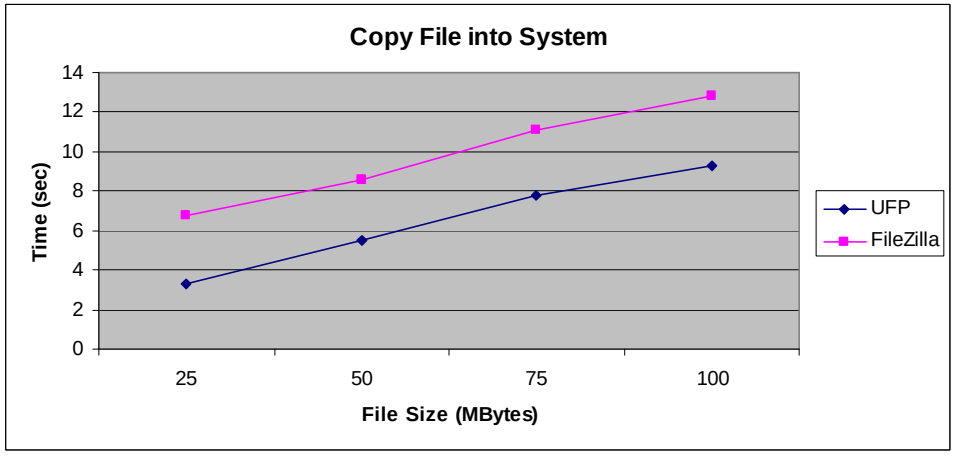

圖 51 檔案上載至 FTP Server 服務

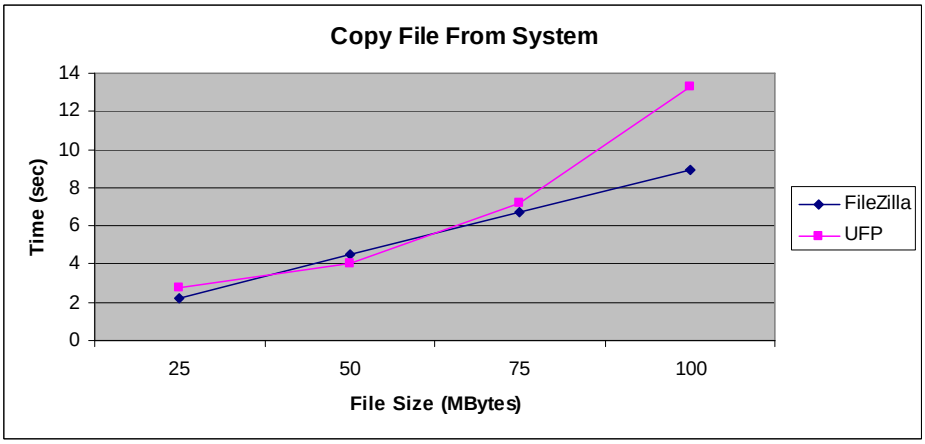

圖 52 檔案從 FTP Server 服務下傳

由圖 51、52 可得知，UFP 與 FTP Client 在存取檔案時，實際使用的時間差不多，在 FTP 測試中，測試環境內的網路速度提升相當多，這也讓整體系統時間提升不少，由此可知，透過 UFP 存取檔案事實上與使用者利用一般工具程式所花的時間差不多，存取服務的快慢主要還是決定於網路速度的快慢，系統所產生的延遲遠不及透過 Internet 存取服務所造成的延遲。

七、結論與探討

在本論文中我們設計與實作了個人檔案整合系統(UFP)，透過此系統使用者可達到

1. 將目前網路上的儲存服務與個人的儲存裝置作整合，透過網路儲存服務，使用者可不受環境與時間的限制存取檔案，同時也不必擔心儲存空間不足的問題。

2. 在整合的環境中，提供一致性的操作與存取檔案的方法，在操作檔案方面，使用者存取這些檔案如同存取本端檔案般容易，使用者也不需知道該服務或是檔案是何種類型並且位於何處。

3. 在整合的環境中提供一致性的檔案分享機制，使用者存取分享檔案時，不需要知道該檔位於哪一種網路儲存服務，也不需要擁有該服務帳號便可存取到使用者所公開出來的檔案，並且提供了及時分享資訊的功能，讓朋友可於第一時間得知你所分享的檔案。

在實際使用上面，使用者可將此系統放至於 Home Gateway 上，使用者透過前端 User Application 的遠端控制，即可存取存放於系統內之檔案，UFP 是一個立即可用的個人檔案管理平台，它不需要新的標準的推出，也不需要網路服務端開發新的介面，透過模組化的設計與其彈性化的整合能力，盡可能的利用到網路上的儲存空間，讓使用者能更有效的分享或是備份個人檔案。

7.1 未來發展

在實作與設計本系統時，已考慮透過物件與模組化的設計讓系統具備足夠的彈性，用以面對未來越來越多的網路儲存服務，對於現在或是未來新的儲存服務，仍然是有很大的改進空間，以下列出三點:

1. 整合新的儲存服務
   透過開發新的 Service Adapter 可以整合未來新的儲存服務或是更多的儲存裝置，目前在實作線上免費服務的整合時，依舊有些服務無法被整合到系統中，也許可透過其他的元件完成目前無法整合的儲存服務，另一方面，有些免費的網路服務在使用上有許多限制，例如單一檔案不可超過 1GByte 或是檔案超過 1 個月就自動消失等，這些限制在系統中並未幫使用者作任何處理，如果能幫使用者作進一步處理，讓使用者不需察覺這些限制，相信在使

用上會更為便捷。

2. 提供更多檔案服務之功能
   透過開發新的 Action，可以提供更多的檔案服務，例如: 檔案同步，在不同網路儲存服務內的檔案可以透過系統隨時同步更新，或是提供檔案轉換的功能，相同的檔案在不同 User Application 中會有不同的呈現或是顯示方式，這些都是除了基本的檔案操作之外的功能。

3. 利用 Cache 機制加快檔案存取
   透過網路存取檔案通常會因為網路速度造成存取檔案速度變慢，另一個因素是在分享檔案時，透過 UFP Server 代為存取分享檔案再將檔案發送到 UFP Client 端也會造成存取檔案的延遲，這方面有待透過 Cache 機制加速檔案的存取，使用 Cache 機制就不得不考慮檔案同步的問題，使用者存取的 Cache 是否為最新的檔案狀態? 這都是未來可在 UFP 系統中所加強與研究的部分。

參考文獻

[1] Avishay Traeger, Nikolai Joukov, Josef Sipek, and Erez Zadok, “Using Free Web Storage for Data Backup”, StorageSS’06,ACM, October 30, 2006.

[2] Commons VFS, http://commons.apache.org/vfs

[3] Wenying Zeng, Yuelong Zhao, Wei Song, “Research on Cloud Storage Architecture and Key Technologies“, ICIS 2009, ACM, November 24-26, 2009.

[4] Chaitanya Baru, Reagan Moore, Arcot Rajasekar, Michael Wan, “The SDSC Storage Resource Broker”, In Proceeding of CASCON’98 Conference, Nov. 1998.

[5] Jieren Ni, Xiaomeng Huang, Wenhao Xu, Guangwen Yang,”FRB: File Resource Broker for Integrating Heterogeneous File Resources”, Eighth International Conference on Grid and Cooperative Computing, 2009

[6] HAN Hua, GUO Chaoyang, DAI Yafei, YUE Bin, LI Xiaoming,” A Scheme to Construct Global File System”, Web Information Systems Engineering, 2001.

[7] KONFABULATOR4.5, http://widgets.yahoo.com/tools/Yahoo

[8] RFC 1867,“Form-based File Upload in HTML”, November 1995.

[9] Steven Versteeg, “Languages for Mobile Agents”, 25 August, 1997.

[10] Engin Bozdag, Ali Mesbah and Arie van Deursen, “A Comparison of Push and Pull Techniques for AJAX”, Software Engineering Research Group, 2007.

[11] Pengzhi Xu, Xiaomeng Huang, Yongwei Wu, Likun Liu, Weimin Zheng,”Campus Cloud for Data Storage and Sharing”, Eighth International Conference on Grid and Cooperative Computing, 2009.

[12] Flavius Frasincar, Geert-Jan Houben, “XML-Based AutomaticWeb Presentation Generation”, 2001

[13] JavaMail API 1.4.3,http://java.sun.com/products/javamail/

[14] SNIA, Information Management Extensible Access Method (XAM) Part 1: Architecture 1.0, July 9, 2008.

[15] Filesystem in Userspace Project, http://fuse.sourceforge.net/

[16] GMail Filesystem over FUSE, http://sr71.net/projects/gmailfs/