逢 甲 大 學

資訊工程學系專題報告

以SIP為基礎結合Web界面的分散式多功能服務系統

學　　生：曾秉鈞 (四丁)
鄭龍凱 (四丁)
蕭煒翰 (四丁)
林聖凱 (四丁)

指導教授：李維聰

中華民國九十二年十月

目　　錄

第一章　前　言

隨著寬頻網路的逐漸普及，一些以往因為頻寬限制所做不到的多媒體傳輸在頻寬上已經不成問題，許多與此有關的應用軟體也越來越多〔如：MSN Messanger、NetMeeting、Yahoo Messanger...等〕。雖然這些應用軟體的功能都很強大，但是仍然有一些更理想的功能可以達到，本專題就是為此而產生，希望能夠完成一套功能更完整的視訊會議軟體。

第二章　研究動機及目的

隨著寬頻網路的逐漸普及，一些以往因為頻寬限制所做不到的多媒體傳輸在頻寬上已經不成問題，許多與此有關的應用軟體也越來越多〔如：NetMeeting、msn...等〕。雖然這些應用軟體的功能都很強大，但是仍然有一些更理想的功能可以達到，本計畫就是為此而產生，希望能夠完成一套功能更完整的視訊會議軟體。

本計畫所希望達到的目的除了一般視訊會議所需的多媒體即時傳輸、電子白板之外，還包括了影音留言服務、本地端多媒體串流紀錄服務、線上會議投票服務、檔案傳遞服務、網路電話、多人混音，使得整個會議能夠有更完整、更方便的功能，除此之外更希望能夠穿越防火牆以及 NAT 之間的互相連結。

- Ø 多媒體即時傳輸：使用者們可以即時看到對方的影像、聲音，更能拉進彼此之間的距離。
- Ø 電子白板：提供繪圖界面，並將本地所繪圖的資訊傳至多個遠端使用者。
- Ø 影音留言服務：當使用者不在的時候，其他人可以透過此服務留下影音訊息給使用者，而當使用者上線時除了可以透過本軟體收下訊息外、亦可以透過網頁觀看訊息，以達到只要有網路的地方就不怕會漏掉任何一筆重要的訊息。並且，使用者也可以透過此服務達到類似答錄機的功能。
- Ø 本地端多媒體串流服務：由於是視訊會議，所以讓使用者可以隨時錄下會議中的影像及聲音，以供日後參考。
- Ø 線上會議投票服務：在會議中，不免會需要投票，所以讓與會者可進行即時的線上投票並且馬上可以得到投票的結果。
- Ø 檔案傳遞服務：以檔案分割的方式，將分享的檔案是格式化的方式，切成不同的片段，加快了檔案傳遞的速度，以達到只要有其檔案片段的來源就可以同時向不同來源下載其所需的檔案內容。
- Ø 網路電話：藉由網際網路無遠弗界的特性，以專有的通話協定建立、結束並管理通話期間的影音串流，因為網際網路的廣域性，使用者無須透過越洋電話與其他人聯絡，因而減低通話的成本。

第三章 系統原理及分析

3.1 SIP 協定概觀與實作架構

3.1.1 SIP 協定概觀

Session Initiation Protocol〔SIP〕是一套定義於應用層的連線控制協定，其主要功能在於多個參與者之間會議的建立、修改以及終止，會議可包含網路視訊會議、網路電話連線以及多媒體資源的散佈，會議裡參與者可透過群播、單點傳播甚至是合併兩者的機制與其他參與者溝通。

SIP 通訊協定的訊息格式十分類似 HTTP〔Hypertext Transport Protocol〕，這個通訊協定的訊息是以 ISO10646 UTF-8 的純文字格式編碼，以該方式做為傳遞的訊息格式，其優點在於無作業系統平台上的限制，因而實作者無須考慮到發展平台之間的相異處，使用 Java、Tcl 和 Perl 等程式語言皆可輕易實作出來，而 SIP 通訊協定最大的優勢是在它的彈性及延伸性，加上 SIP 的主要功能在建立視訊會議，相較於視訊會議期間的大量媒體串流，建立會議之前所需要的額外 SIP 控制訊息便不影響連線的品質。

SIP 訊息可分為請求〔Request〕和回應〔Response〕訊息，它是使用 RFC822 所定義的通用訊息格式〔Generic-Message Format〕，兩種訊息皆包括起始行〔start line〕、一個以上的標頭欄位〔headers〕、一行空白行〔僅包括 CRLF〕以及選擇性(可有可無)的訊息內容〔message body〕，下圖是通用訊息格式的 ABNF。

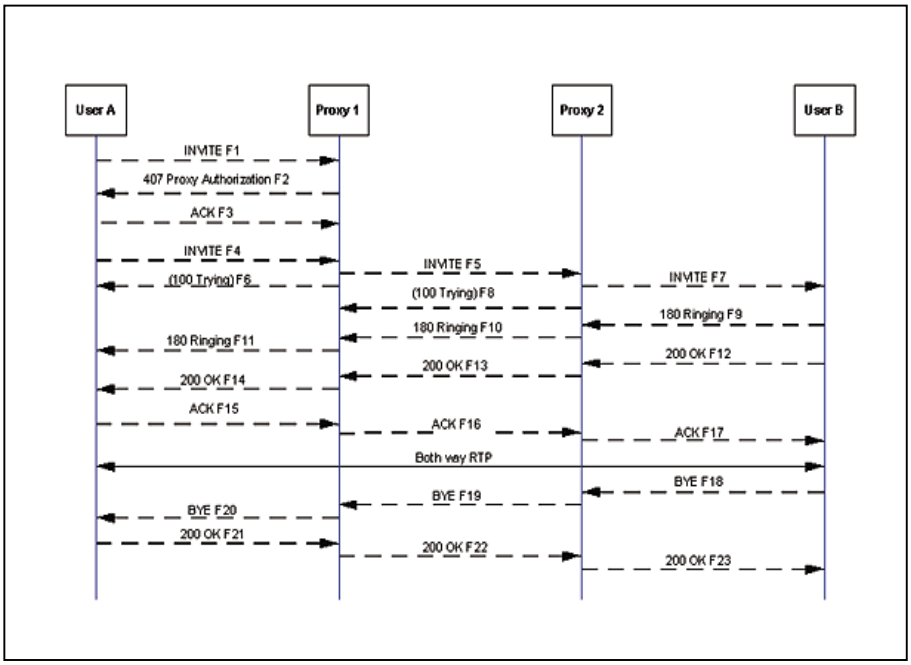
```
generic-message = start-line
                  *message-header
                  CRLF
                  [ message-body ]

start-line      = request-Line | status-Line

message-header = 〔 general-header
                  | request-header
                  | response-header
                  | entity-header 〕
```

a. 請求訊息〔Request〕

請求訊息的格式如下圖：

```
Request    = Request-Line
             *〔 general-header
              | request-header
              | entity-header 〕
             CRLF
             [ message-body ]
```

請求訊息的方法可分為：ACK、BYE、CANCEL、INVITE、OPTIONS、REGISTER。當發話端欲初始一個通話連線，可傳送一個 INVITE 的 SIP 請求訊息；終止一個通話連線則可傳送一個 BYE 的 SIP 請求訊息：下圖即為兩個端點之間的簡易呼叫流程範例：



n 呼叫流程說明

如上圖所示，發話端 A 欲透過兩個代理伺服器(Proxy1 和 Proxy2)向使用者 B 要求建立通話連線，Proxy1 的進行訊息轉傳前需要使用者進行認證手續，但發話端 A 起初並未將認證資訊(F1)存放在訊息之中，因此代理伺服器回傳一個認證需求的回應訊息通知發話端進行下次的認證步驟(F2)。當發話端 A 再次傳送含有認證資訊的邀請訊息後(F4)，使用者 B 接受該通話請求並開始進行通話，而後，當使用者 B 傳送 BYE 請求訊息之後，通話連線便結束。

- F1 - INVITE A -> Proxy 1
  通話連線的開始是透過某個端點傳送 INVITE 請求訊息，其訊息內容如下圖所示，在這個訊息傳遞流程的範例中，要求訊息內容包括了會議描述請求(Session Description Request)。

```
INVITE sip:UserB@ss1.wcom.com SIP/2.0
Via: SIP/2.0/UDP here.com:5060
From: BigGuy
To: LittleGuy
Call-ID: 12345600@here.com
CSeq: 1 INVITE
Contact: BigGuy
Content-Type: application/sdp
Content-Length: 147

v=0
o=UserA 2890844526 2890844526 IN IP4 here.com
s=Session SDP
c=IN IP4 100.101.102.103
t=0 0
m=audio 49170 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

- F2 - 407 Proxy Authorization Required Proxy 1 -> User A
  SIP 協定是以請求—回應模式進行訊息的傳遞，在這個範例中，Proxy1 要求發話端 A 需先進行認證步驟。

```
SIP/2.0 407 Proxy Authorization Required
Via: SIP/2.0/UDP here.com:5060
From: BigGuy
To: LittleGuy
Call-ID: 12345600@here.com
CSeq: 1 INVITE
Proxy-Authenticate: Digest realm="MCI WorldCom
SIP",
domain="wcom.com",
nonce="wf84f1ceczx41ae6cbe5aea9c8e88d359",
opaque="", stale="FALSE", algorithm="MD5"
```

- F17 ACK Proxy2 -> UserB
  當我們察看呼叫流程的下方，可發現實際的聲音串流傳送是以Realtime Transport Protocol(RTP)處理。

```
ACK sip: UserB@there.com SIP/2.0
Via: SIP/2.0/UDP ss2.wcom.com:5060
Via: SIP/2.0/UDP ss1.wcom.com:5060
Via: SIP/2.0/UDP here.com:5060
From: BigGuy
To: LittleGuy ;tag=314159
Call-ID: 12345601@here.com
CSeq: 1 ACK
Content-Length: 0
```

- F18 BYE UserB -> Proxy2
  通話連線是透過傳送 BYE 請求給接收端終止。

```
BYE sip: UserA@ss2.wcom.com SIP/2.0
Via: SIP/2.0/UDP there.com:5060
Route: ,
From: LittleGuy ;tag=314159
To: BigGuy
Call-ID: 12345601@here.com
CSeq: 1 BYE
Content-Length: 0
```

b. 回應訊息〔Response〕

```
Response  =  Status-Line
             *〔 general-header
              | response-header
              | entity-header 〕
             CRLF
             [ message-body ]
```

其狀態碼依受話端的情況可分為：

- **1xx: Informational**
  表示伺服器或代理伺服器正在處理某些動作而且尚未有任何確切的回應。
- **2xx: Success**
  請求成功而且必須終止該搜尋。
- **3xx: Redirection**
  提供使用者的新位址或其他能滿足該請求的服務。
- **4xx: Client Error**
  該類型狀態是由特定伺服器傳回的確切失敗回應。
- **5xx: Server Error**
  該類型狀態是因為伺服器本身發生錯誤而傳回的失敗回應。
- **6xx: Global Failure**
  表示伺服器含有某些使用者確切資訊

3.1.2 SIP 實作架構

本專題架構底層即是以 SIP 通訊協定來完成訊息的溝通，為了能符合業界之規格標準，實作方面是採用 JAIN(Java Application Integrated Network)計劃中的 SIP 協定堆疊架構，其中 JAIN 這個組織主要的負責的工作在於制定各種通訊協定(特別是網路電話方面的協定)的呼叫介面，讓各廠商能依照其規定的呼叫流程實作其下的動作。本專題在這個部分則是依照該呼叫介面將實作的功能完成，以銜接上層的端點協定(Peer Protocol)。

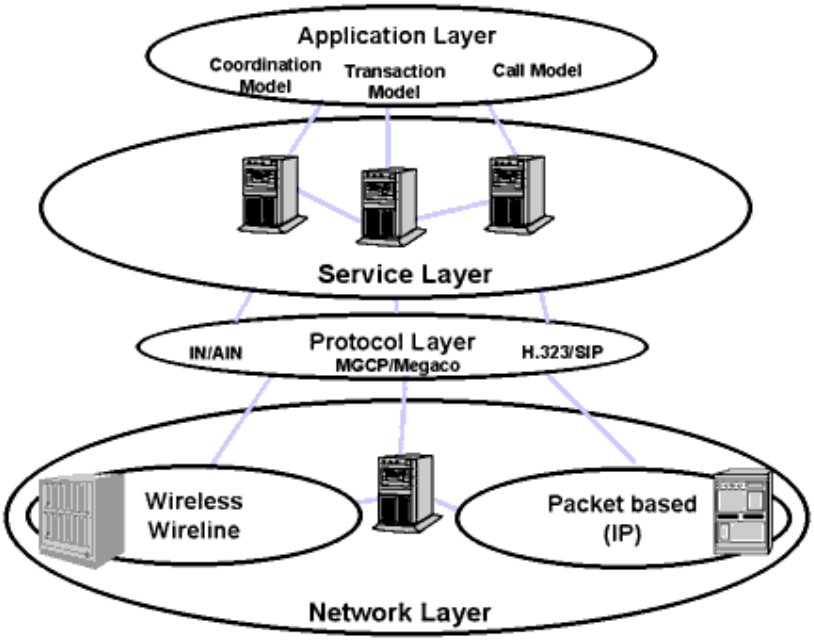
JAIN APIs 是一系列以 Java 虛擬機器為平台的網路通訊技術，以該技術為基礎平台所發展的應用程式從網路連線或存取資料便能擁有足夠的可攜性、完整性以及安全性。

JAIN 提供了抽象與關聯性的介面，可在公眾交換電話網路〔Public Switched Telephone Network – PSTN〕、封包交換網路 〔例如 Internet Protocol〔IP〕或 Asynchronous Transfer Mode〔ATM〕〕和無線網路之間建立服務元件，這便是所謂的整合式網路，除此之外，允許 Java 應用程式在安全的網路裡存取資源的優勢在於能夠同時提供各種功能不相同服務元件。因此，相較於其他以單一網路架構為基礎的封閉式系統，JAIN 技術改變了電話通信市場的思維，它使用服務元件的建立和配置更為簡便。

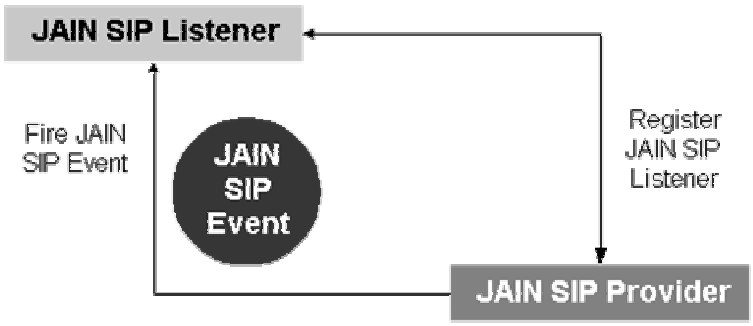
JAIN 技術是 Java 平台的一種延伸，它的發展是在昇陽公司的 Java Specification Participation Agreement〔JSPA〕、Java Community ProcessSM〔JCP〕和 Community Source Code Licensing〔SCSL〕規範下進行的，進一步的資訊可連接至 http://jcp.org 參考。

JAIN 的發展提案包括兩種 API 規格類型：

1. 協定 API 規格定義了有線、無線與 IP 協定訊息。
2. 應用程式API規格訂定了以協定API規格為基礎架構的服務元件建立。



該套件依照 Session Initiation Protocol(SIP) - RFC2543 的通訊協定規格制訂了一套完整的訊息呼叫流程。下圖為 SIP 的事件處理模型：



JAIN SIP API Specification 的模組套件名稱可依功能分為：

- jain.protocol.ip.sip
  定義該套件中的核心元件：SipStack、SipProvider、ListeningPoint。SipStack 功能在於建立、摧毀和管理其下所屬的 SipProvider 物件，SipProvider 和 ListeningPoint 為一對一的關係，每個 SipProvider 僅擁有單一 ListeningPoint，該類別負責解析接收到的協定訊息並將事件(SipEvent)傳至上層聆聽者(SipListener)。

- jain.protocol.ip.sip.address
  負責 SIP 通訊協定的定址，包括三種定址格式：URI、SipURL、NameAddress。

- jain.protocol.ip.sip.header
  定義 SIP 通訊協定訊息的各類型標頭資訊，依照其功能可分為：General、Request、Response 和 Entity Header，而其建立方式需透過 HeaderFactory 來執行。

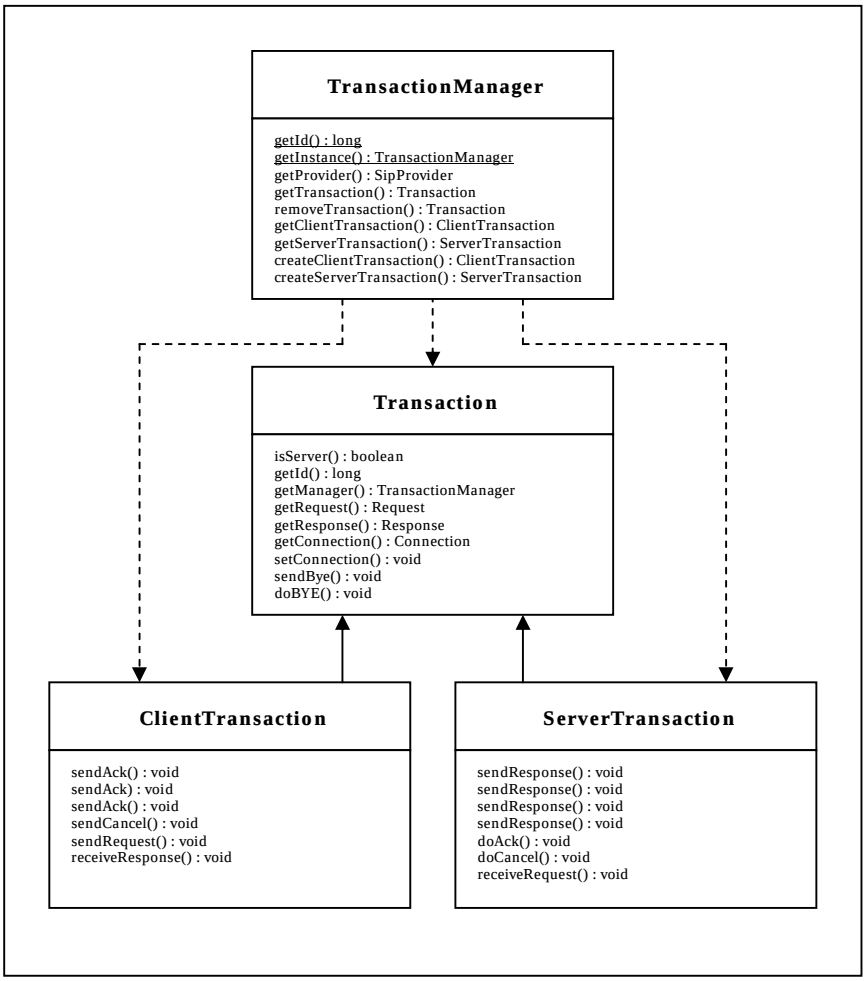
- jain.protocol.ip.sip.message
  定義 SIP 通訊協定的請求(Request)和回應(Response)訊息，依照兩種協定訊息的特性可放入特定功能的標頭資訊，例如請求訊息可包括 General、Request 和 Entity Header；回應訊息可包括 General、Response 和 Entity Header，其詳細規格可參考 RFC2543。

在本專題實作 JAIN-SIP 規格介面期間，發現到以上幾種功能上的缺失，造成上層建立其物件時會造成不易於使用的問題，以下幾點為這個版本所無法支援的問題及本專題提供的解決方案：

1. 協定規格僅支援 RFC2543，而未支援目前最新的 RFC3261
   本專題實作規格之版本編號是 JAIN/SIP 1.0，儘管目前已公佈最新的規格 JAIN/SIP 1.1，但是當時專案正在實作期間，因此無法及時更新至這個版本。另外，因為本專題主要的功能在於端點間的訊息傳遞，並沒有建立通話連線的需求，所以使用的請求方法類型為其後提出的 MESSAGE，而舊有版本中並未定義此方法，因此本專題特定為此加入該訊息類型，並遵循 RFC3428 - Session Initiation Protocol (SIP) Extension for Instant Messaging 的規範實作簡短訊息的傳遞。

2. 不提供訊息交易層(Transaction)架構的功能

本專題所實作的版本並未提供交易層的機制，即管理該請求訊息傳送或接收之後的狀態監控與訊息重傳，此版本對交易機制僅僅利用一個長整數值做為鍵值的方式管理，因此使用者在呼叫或管理上會帶來不便，為此，本專題又另外參考新版本的做法將交易機制納入原舊有版本之中，以下為交易層之 UML 架構圖：



3. 對於 SipStack、SipProvider 的建立無法依所需設定其運作方式

因為這個版本的標準規格在建立以上兩種物件時並無法設定其屬性和運作方式，例如底層傳輸的類型或是否將該端點向鄰近的註冊管理者(Registrar)進行註冊。面對這個問題，舊有版本的介面中並未包含任何選項設定的方法，也因此必須對原規格做稍微的修改，即加上建立 SipStack 時的選項設定，此法不但能讓使用者能依照自己所需設定物件的運作方式，實作廠商也能將本身擁有的額外服務功能加至其中供使用者叫用。

3.2 SOCKS 協定及實作

3.2.1 SOCKS 協定緣由

由於現今網路組態常使用防火牆使內部網路隔離於外部網路或使用網路位址轉譯器(NAT)建立虛擬私有網路(VPN)，這些防火牆系統可提供傳統的協定如 HTTP、TELNET、FTP 隧穿無虞，但對於其它提供特定應用的協定則會出現功能障礙，為了提升網路的透通性，SOCKS 提供了標準化、安全化的機制。

3.2.2 SOCKS 協定簡介

SOCKS 定義三種指令，connect、bind、udp associate，分別介紹如下：

a. Connect – connect 指令封包指示 client 欲連往的目標主機位置及埠號，SOCKS Server 負責連往目標主機並為雙方建立連線。
b. Bind – bind 命令 SOCKS Server 開啟指定的埠號，並等候指定的目標主機連入，一旦目標機連入 SOCKS Server 開啟之埠號，即為雙方建立連線。
c. UDP Associate – 類似 bind 指令運作方式，不過此時負責轉傳的是 UDP Datagram。

SOCKS 又分兩種版本，version 4 及 5，在 version 5 中，SOCKS Server 執行命令前，必須通過認証機制，而主機位置也可以是 IPv4、IPv6、Domain name。

3.2.3SOCKS 實作

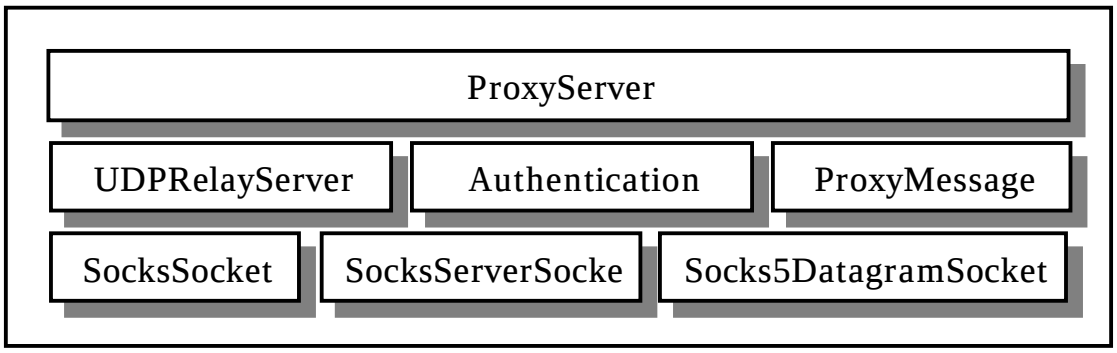


圖 3.2.3.1 SOCKS Server系統堆疊圖

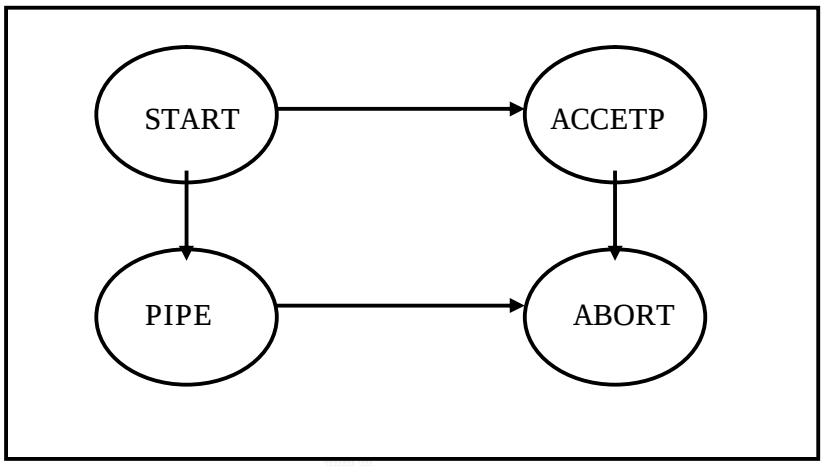


圖 3.2.3.2 ProxyServer狀態流程圖

運作流程 -

ProxyServer 開啓 SOCKS 公認埠，一般爲 1080，當收到連線後，首先判別該請求爲 SOCKSv4 或 SOCKSv5，若爲 version 5 則進行 SOCKS Server 有實作之認証程序，認証通過後將依指令進行對應動作。

a. Connect) ProxyServer 連線至 ProxyMessage 指示之目標位置及埠號，再產生一 ProxyServer 物件，該物件進入 Pipe Mode，負責兩端點雙向資料傳輸。

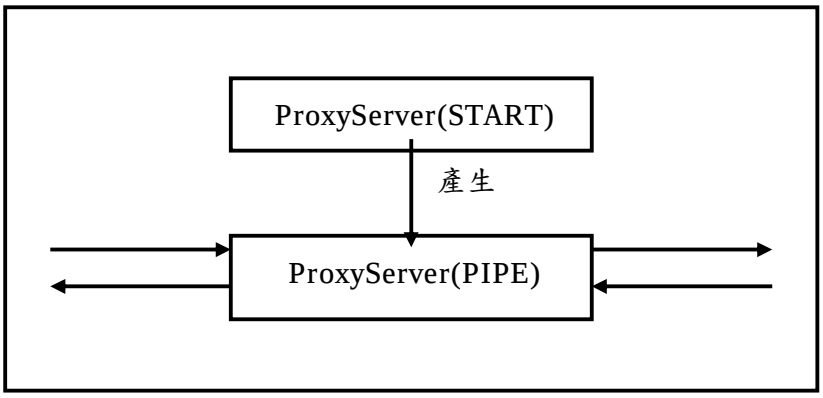


圖 3.2.3.3

b. Bind) ProxyServer 於本地開啟一聆聽埠，並將埠號包成 Response 告知 Client，產生一執行緒負責等候遠端的連線，當正確的遠端連線後，該執行緒再次回傳 Response 告知，並再產生一執行緒，共兩執行緒負責雙向資料傳輸。

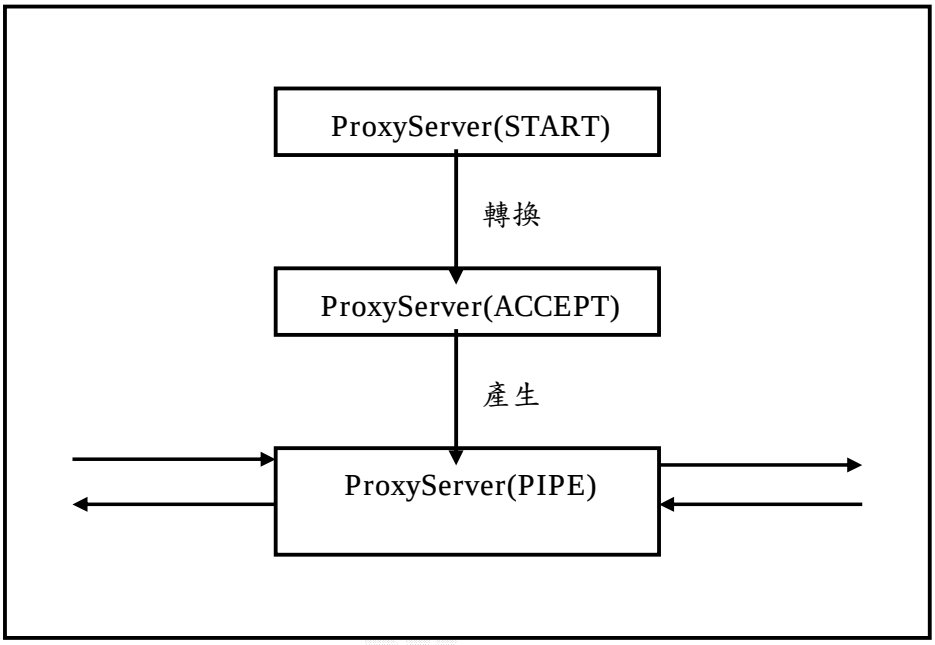


圖 3.2.3.4

c. UDP Associate) ProxyServer 產生 UDPRelayServer，其於本地開啟 Socks5DatagramSocket 聆聽埠，回傳該 Response 告知其聆聽埠，當收到雙方的 KEEP Datagram 後，即產生兩執行緒負責雙向資料傳輸，KEEP Datagram 用來保留雙方真實網路繞路資訊，以便傳輸 UDP Datagram。

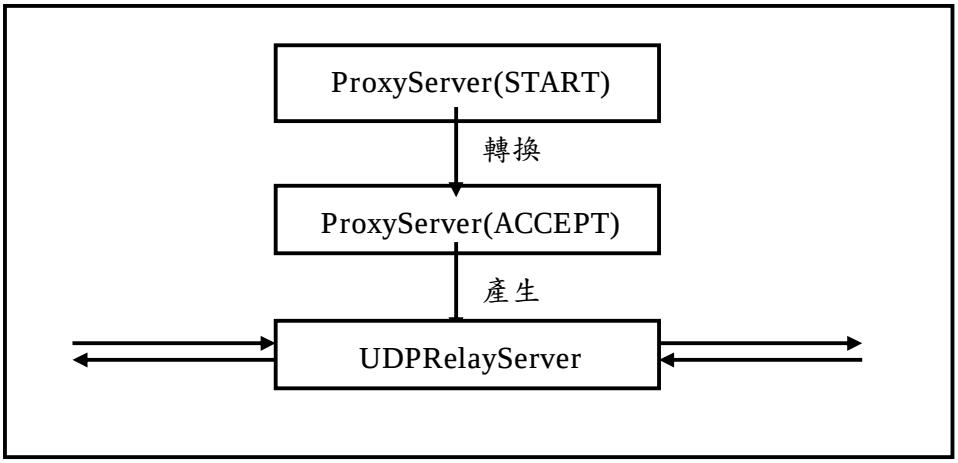


圖 3.2.3.5

3.3 JMF 架構及研究

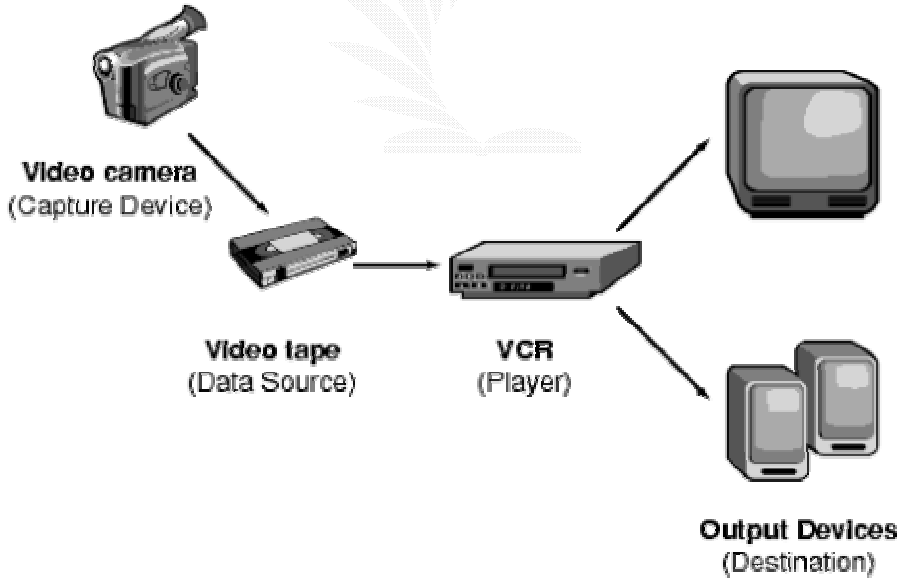
JMF 是一種應用程式介面，整合了 time-based media 到 Java applications 和 Applet。如果程式設計者，對 JMF 的格式興趣也可以自行延伸 JMF，以 plug-in 的方式加入新的 media-supported type，並可自行實作設計 transport-layer 層的傳輸方式。

以下是 JMF 的設計目標

- 容易去設計。
- 對於 Capture device media 的存取。
- 使 Java 能傳輸多媒體串流和網路電話的程式
- 提供進階程式師和技術去實作和延伸。
- 提供對 raw media data 存取能力。
- 提供發展 downloadable demultiplexers, codecs, effects processors, multiplexers, and renderers 的能力。

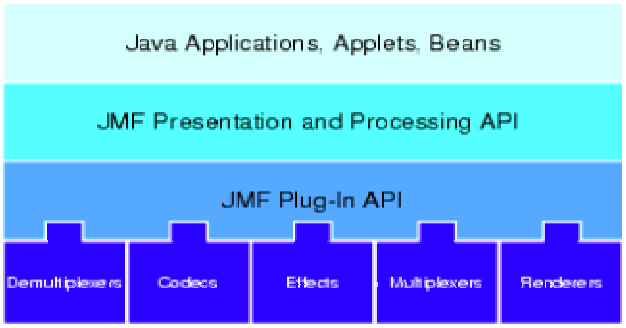
3.3.1 High-Level Architecture

像 tape decks 和 VCRs 這此裝置，提供了常見的 recording model，processing、和呈現 time-based media。當我們用 VCR 播放 movie，我們藉由放入 video tape 來提供 media stream，而 VCR 讀取和解釋在 tape 中會資料，然後傳送適合的訊號到 TV 和 Speaker.



JMF 使用和這一種相同的 basic model，***Data source*** 結合入 media stream 如同 tape，***Player*** 就如同 VCR 控制和處理的機器，而 Playing 和 Capturing audio 和 video 的輸入裝置和輸出裝置就是如同 microphones, cameras, speakers, and monitors。

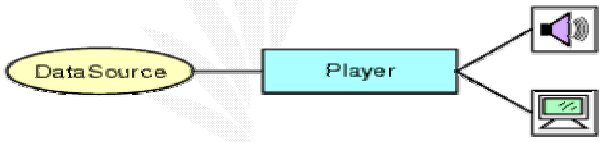
Data sources 和 players 是 **JMF's high-level API** 中不可或缺的部份，來處理 Caputre Device,及呈現和運作以時間為基礎的多媒豐。**JMF** 也提供了低階的 API，使設計者可以進而延伸和作細部的設計。



High-level JMF achitecture.

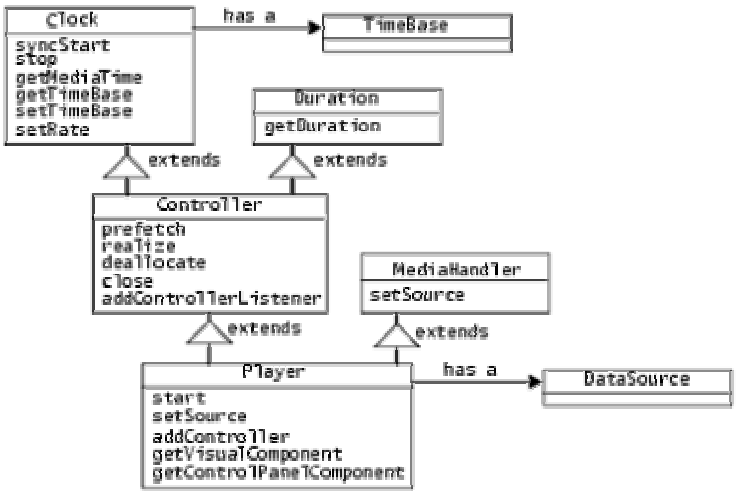
3.3.2 Players

Player 處理 media data 的 input stream 和以精確的時間呈現它。DataSource 被用來來傳送 input media-stream 到 Player，而 render 則依靠多媒體的形態來呈現。



JMF player model.

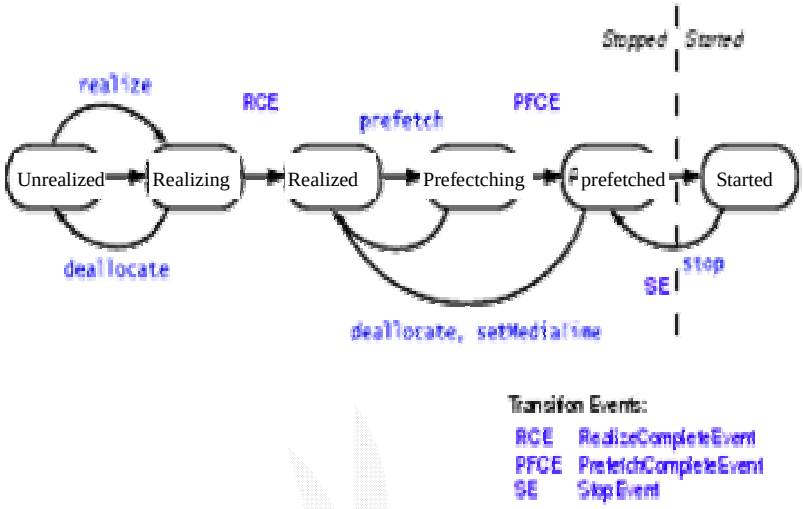
Player 沒有提供任何在控制和如何處理呈現 media data。而 Player 提供了一個標準化的 User Control 和藉由 Clock 和 Controller 來簡作上層的控制。



JMF players.

3.3.2.1 Player States

一個 Player 可以有 6 個 state. Clock 介面定義了 2 個主要 state: *Stopped* and *Started*.為了要使資源的管理容易，所以 Controller 分開了 *Stopped* state 到 5 個 tates: *Unrealized, Realizing, Realized, Prefetching*, and *Prefetched*.



Player states.

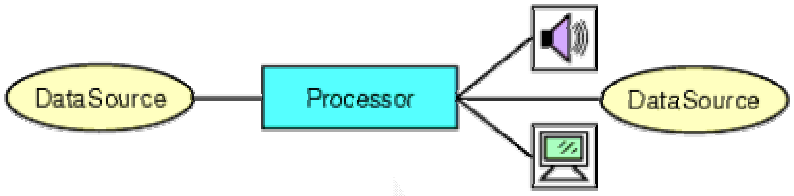
在正常的運作過程，一個 Player 可以到達每一個 state 直到 Started State:

- Player 在 Unrealized state 被起始化，但對 media 的資訊還不了解，當 Player 被建立後，它就在 Unrealized State.
- 常 realize 被呼叫，Player 開始由 Unrealized state 移到 Realizing state，Realizing Player 是處理決定資源的需求，在 realization 階段，Player 得到它只需得到一次的資源，可能包含呈現資源，一個 Realizing Player 常由網路得到這些資訊。
- 當 Player 完成 Realizing，它移動到 Realized state，Realized Player 了解那些資源需要和資訊，由於 Realized Player 了解如何去呈現，所以可以得到 visual components and controls 的資訊。
- 當 prefetch 被呼叫，Player 由 Realized state 到 Prefetching state，Player 正準備去呈現 media ，在這一階段，Player 會先 load 它的 media 資料，和惟一使用的資料，Prefetching 如果 Player 回到先前的狀態，會再發生，因此 Player 需要額外的 buffer，來交換狀態。

- 當 Player 完成 Prefetching，它移到 Prefetched state，Prefetched Player 就進去等待開始的狀態。
- 呼叫 start 使 Player 進入 Started state.，Started Player 的 time-base time 和 media time 作 mapping 的動作，且 clock 也開始 running，即使 Player 可能還在等待一個特定的時間去呈現。

3.3.3 Processors

Processors 也可以用來呈現 media data.Processor 只是一種 Player 的特別形態，提供了處理 input media stream。Processor 提供了所有和 Player 一樣的 controls。

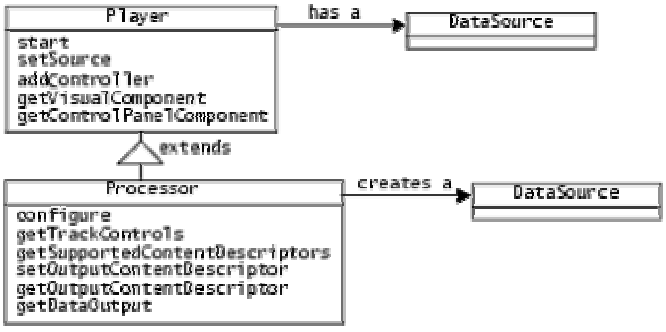


JMF processor model.

除了呈現 media data 到輸出裝置,Processor 可以輸出 media data 以 DataSource 的方式，將 DataSource 指定到下一個 Player or Processor，更進一步 Processor 可以傳送另一目的地，如傳送到 File 中。

3.3.3.1Processing

Processor 是一種 Player 以 DatatSource 為輸入，執行一些使用者定的程序在 media data，然後輸出和處理 media data

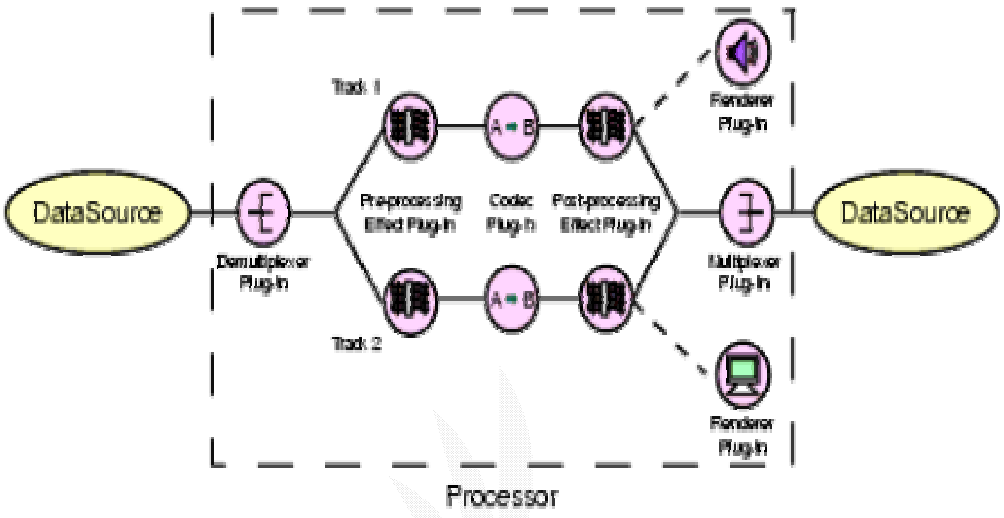


JMF processors.

Processor 可以送輸出資料到資料呈現的裝置或輸出到 DataSource，如果資料是輸出到 DataSource，DataSource 可以被使用如同另一個 Player 或 Processor 的輸入，也可以輸出到 DataSink。

當由 Player 處理執行之前所定義的工作，Processor 允許應用程式發展者去定義處理的形態，對應到相對的 media data，它使得應用程式的 effects、mixing、compositing 以即時的方式來處理。

處理 media data 分為下列幾個 stages

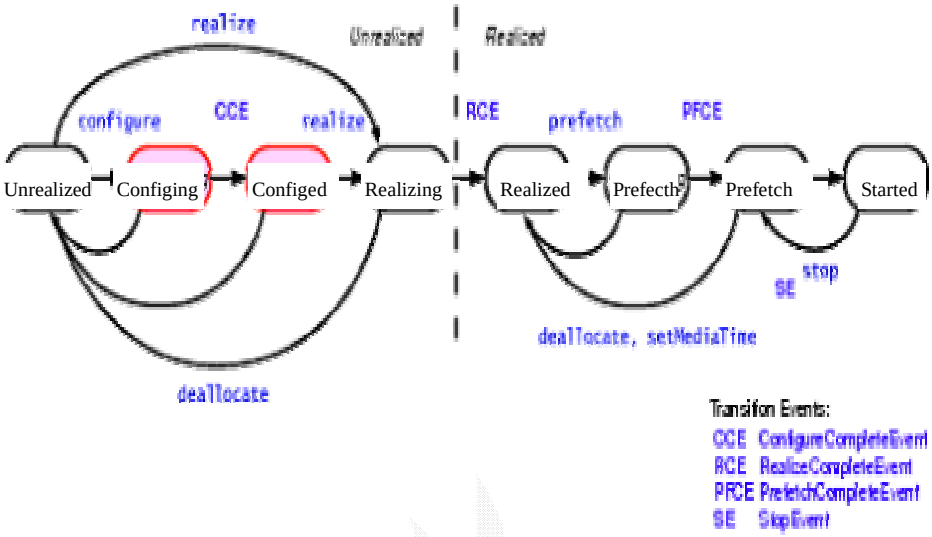


Processor stages.

- Demultiplexing 是 parsing input stream 的過程，如果 stream 包含了多個 tracks，它們分出和輸出個別地。
- Pre-Processing 是處理 effect algorithms 到 tracks 抽出到 input stream.
- Transcoding 是處理轉換每一 media data 的 track 的過程，當 data stream 從 compressed type 到 uncompressed type。
- Post-Processing 處理加入有效率的演算以去 decoded tracks.
- Multiplexing 是處理內部轉碼 media tracks 到單獨的 output stream
- Rendering 處理資料給 the user.

3.3.3.2 Processor States

Processor 有二個額外的等待 state, *Configuring* and *Configured*，它們發生在 Procssor 進入 *Realizing* state 之前。

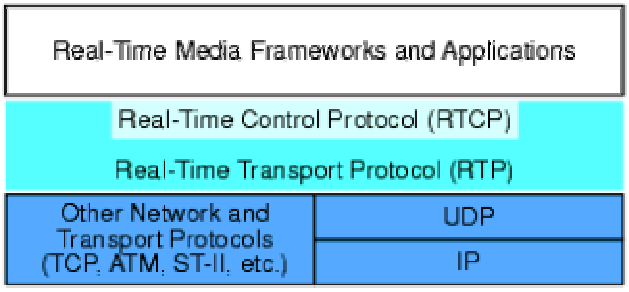


Processor states.

- Ø 當 configure 被叫 Processor 進入 Configuring state，. 它連接到 DataSource, demultiplexes 的 input Stream，且存取關於輸入資料的格式資料。
- Ø 當已連接到 DataSource，Processor 移到 Configured state，且資料的格式已經決定，當 Pocessor 到達 Configured state，會發生 ConfigureCompleteEvent。
- Ø 當 Realize 被呼叫，Processor 轉到 Realized state，Processor 只會有一次建構完整的 Realized State.

3.3.4 Real-Time Transport Protocol

在 JMF 中也提供了 RTP(Realtime Transport Protocol)的運作控制，RTP 提供了點對點的及時性資料傳輸 Service。RTP 是和網路和 transport-protocol 獨立的協定，即使它一般是以 UDP 來傳輸。



:RTP architecture.

RTP 可以被使用在 unicast and multicast 的網路 Service.藉由 *unicast network* 的 Service 環境，可以分別資料複製由來源到每一個目的端。藉由 *unicast network* 的 Service 環境，可以傳送 Stable 的多個目的地。Multicasting 對於許多多媒體應用程式是相當有效率的，如在多人的視訊會議。在標準的 Internet Protocol (IP)對於 multicasting 的提供。

3.3.4.1 RTP Services

RTP 可以給使用者定義要被傳送的資料型態，決定每一資料封包要以何種順序來傳送，和如何同步來自於不同的多媒體來源。

RTP 資料封包並不保證到達的順序和它們的送出的順序相同，事實上，它們一點也不保證是否到達目的地。再來就是在接收端必需藉由加在封包 header 的資訊去重新建構傳送端的封包順序，並偵測遺失的封包

由於 RTP 沒有提供任何機智去保證即時性的傳輸或提供其它 Service 的品質，因此它加入了 control protocol (RTCP)，使用者可以 monitor 資料傳輸的品質，RTCP 也提供了對於 RTP 傳輸的控制和定義機智。

如果 Service 的品質對於特別的應用程式是必需的，RTP 可以使用 resource reservation protocol 來提供 connection-oriented 的 Service。

3.3.4.2 RTP Architecture

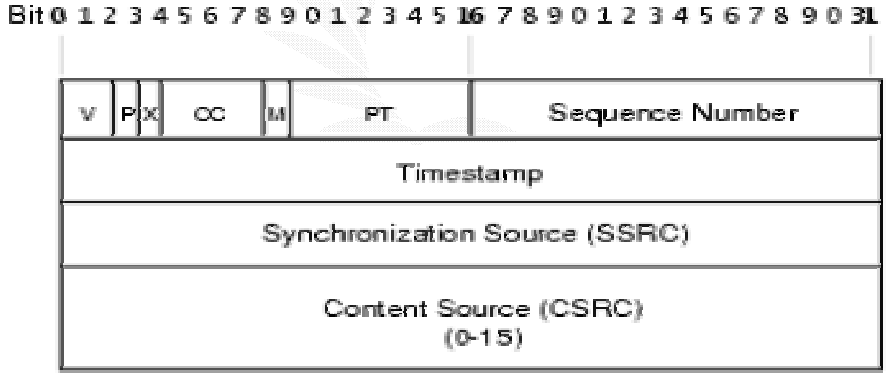
一個 RTP session 是以 RTP 應用程式的傳輸關聯性集合，session 是以網路位置和一組 ports 組成。一個 port 是使用在多媒體資料，另一個是用來傳輸 control (RTCP)資料。

Participant 是一個單獨的運作個體，或是使用者 participating 在這一個 session 中，*Participant* 在 session 中可以由被動的接受資料，主動的傳送資料組成。

每一種多媒體形態以不定的 session 來傳輸。所以如果在網路電話會議中如果包含了 audio 和 video ，一個 session 用來傳輸 audio 資料，另外一個 session 用來傳輸 video 資料。因此它使得 participants 可以去選擇他們想接收那一種多媒體形態，如果某人只有一個 low-bandwidth 的網路連線環境，他可能只要接收網路電話會議得 audio 部份。

3.3.5 Data Packets

一個 session 的多媒體資料傳輸如同一連串的封包。一連串的封包資料是以 RTP stream 來自某一特定的來源。每一個 RTP 在 Stream 中資料封包含二個部份，一個是結構化的 header，和另一個真正的資料 (packet's *payload*)。



RTP data-packet header format.

RTP 資料的 header 包含了以下容:

- **The RTP version number (V)**: 2 bits. 目前的 Version 是 2.0
- **Padding** (P): 1 bit 如果 padding bit 是一個集合，有一個 or 多個 bytes 在封包的後面，它不是 payload 的一部份，在封包中的最後一個 byte 指出了 padding 的 byte 數目。在 padding 的部份是以一些加密的演算法來使用。
- **Extension** (X): 1 bit. 如果 extension 的 bit 被設定，固定的 header 會隨著另一 header extension。Extension 的機智使得加入資訊在 header 得以完成
- **CSRC Count** (CC): 4 bits. CSRC 的數字定義了接下來固定的 header，如果 CSRC Count 是 0，同時性的來源是來源的 payload
- **Marker (M)**: 1 bit. Marker bit 以特別的多媒體資料定義
- **Payload Type** (PT): 7 bits. 在 media profile table 的 index ，描述 payload 的格式，Payload 對應於 audio 和 video 在 RFC 1890。
- **Sequence Number**: 16 bits. 一個惟一的 packet number 用來定義在封包中的位置須序。Packet number 的每送一次增加一次。
- **Timestamp**: 32 bits. 表達在 payload 中的第一個取樣的 byte 。
- 如果它們在 logic 上是相同時間的，一些連續的封包可以有相同的 timestamp，像是全部相同部的 video frame。
- **SSRC**: 32 bits. 同義了同時性的來源，如果 CSRC count 是 0，payload source 就是同時性的 source，如果 CSRC count 是非 0，SSRC 則定義為 mixer。
- **CSRC**: 32 bits each. 定義了貢獻給 payload 的來源，contributing sources 的數目在 CSRC count 欄位中指出，可以達到 26 個 contributing sources，如果有許多部份所組成的 contributing sources，則 payload 就是那些 sources 的 mixed 資料。

3.3.6 Control Packets

除了在 session 多媒體資料之外，control data (RTCP)也是週期性傳給所有在 session 的 participants，RTCP packets 可以包含給 session participants 的 Service 品質資訊，來源資訊在 data port 傳輸，統計目前已傳的的資料。

RTCP packets 有下列的形態

- Sender Report
- Receiver Report
- Source Description
- Bye
- Application-specific

RTCP packets 是可以堆疊的，它傳送了混合的封包，包含了至少二個封包，一個是 report packet，另一個是 source description packet。

所有在 session 的 participants 傳送 RTCP packets，一個最近有傳送的資料封包的的 participant，會產生 *sender report*。Sender report (SR) 的內容包括了全部數的封包和 byte 數目，如同資訊會被使用來自不同 session 的同步多媒體串流

Session participants 會週期性地產生 *receiver reports* 給所有接收資料封包的來源，Receiver report (RR)的內容包括了關於封包遺失的數目，最大已接收的 sequence number，和一個 timestamp，可以用來減少在 Sender 和 Receiver 之間來回的 delay。

在混點的第一個 RTCP packet 必需是 report packet，即使沒有資料被傳送 or 接收，在這一種情況，一個空的 receiver report 被傳送了。

所有混合的 RTCP packets 必需包含了有定義來源的 canonical name (CNAME) 的 source description (SDES) 元素。除此之外，資訊可能包含在 source description，像是 source 的名稱,電子郵件的位址，電話號碼，來源的位址,應用程式名稱, or 訊息內容，來描述目前的來源狀態。

當來源不再運作時，它傳送一個 RTCP BYE packet。在 BYE 說明它離開 session 的原因。

RTCP APP packets 提供了給應用程式定義了資訊經由 RTP control port 傳送的機智。

3.3.6.1Receiving Media Streams From the Network

一些形態的應用程式必需能夠接收 RTP streams，如:

- 網路電話會議必需能夠接收來自 RTP session 的 media stream，並且在使用者的控制環境下呈現。
- 電話答錄機應用程必需能多接收來自 RTP session 的 media stream，並儲存成檔案。
- 一個紀錄交談內容或網路電話會議必需能夠接收來自 RTP session 的 media stream，可以在使用者的控制環境下呈現，且能儲存到檔案。

3.3.6.2 Transmitting Media Streams Across the Network

RTP Server 應用程式傳送由 capture device 所得到的資料或由已儲存的 media streams 到網路。

例如:

在網路電話會議應用程式中，其中一個 media stream 可能擷取來自 video camera，然後傳送出一個或多個 RTP session。且 media streams 可能被碼成複合的多媒體格式，然再以送出幾個 RTP session 到網路電話會議不同形式的接收者，多人網路電話會議以許多的 unicast RTP session 來實作 multicat。

3.3.8 H.263 與 H.261 的差異

事實上，H.263 只是 H.261 的進一步延伸，而延伸方式則採用了類似 MPEG 的方法，其間差異主要在於兩者目標的不同：

- 兩者鎖定的目標位元率(Target Bit-rate)不同，H.261 是 px64Kbps，H.263 則希望在 64Kbps 以下，這是因為前者係利用 ISDN，而後者則希望透過現有的電話線網路來做傳輸。
- H.263 與 H.261 影像的格式不同，H.261 只提供 CIF 或 QCIF 的影像格式，而 H.263 則提供更多選擇，希望將來在不同傳輸媒介上的影像資訊都能夠互通。

除了上述這兩個明顯的差別外，在系統上，H.263 也做了很多的改變以增進編碼效率，在這裡我們把幾個比較重要的特色說明一下：

- 位移估計(Motion Estimation)的精確度可以到達半個像點(Half-pixelaccuracy)；這種方式和 MPEG 所使用的方式類似，主要是因為人眼視覺的特性，使得我們可以利用這種方法將位移估計的準確度提高。也就是說，利用影像在時間上的相關性將影像作更多壓縮。

- 可重疊式的區塊位移補償(Overlapped Block Motion Compensation;OBMC)；當我們利用位移估計找出最相似的區塊後，在傳統的方法中就直接把這個區塊貼回，再加上估計的誤差就成為目前之影像。然而在 H.263 中，所有目前的區塊都會是三個區塊之加權平均值再加上誤差，而這三個區塊具有相鄰的關係。在利用了這種方式之後，區塊的大小就不再是傳統 16×16 的大小而成為 8×8 了。也就是說，這會使得壓縮後的影像品質更佳清晰，但是卻還能夠保持極佳的壓縮比。這些動作規定於 H.263 中一個新的增強模式中(AdvancedPrediction Mode;AP)。

- 算術編碼方式；以往的壓縮方式在對付統計上的多餘性，通常是利用霍夫曼編碼(Huffman Coding)來達成可變長度編碼(Variable Length Coding)，但是使用了新的算術式編碼可以更有效地把資料壓縮到接近其亂度(Entropy)，因此在 H.263 中有一個增強模式(Syntax-based Arithmetic Coding Mode;SAC)是可以使用此種新的編碼方式。

- 無限制的位移向量模式(Unrestricted Motion Vector Mode ; UMV)；使用這個模式的話，位移向量將可以指到超過螢幕範圍的位置，當然，有些像點的值是用類似內插之方法得到。這個增強模式在螢幕畫面不大時相當有用，例如 sub-QCIF 或 QCIF 等可能用在液晶顯示的影像格式，因為當影像不大時，我們會發現影像中大部份的區塊都是位於邊緣位置。也就是說，當我們在做位移估計時，若使用傳統方式，有許多區塊的搜尋範圍會變少，所以這種增強模式將能夠彌補這方面的問題，如果影像中的東西正好是在螢幕邊緣位置移動時，影像品質的改善會更加明顯。

PB-frame 模式是類似 MPEG 中 B-frame 的一種方法。在 MPEG 中，P-frame 就是利用以前的影像來預測目前影像之方法，但是除了這種方式之外，還有一種 B-frame 會利用以往或其後的影像來做預測，這樣雖然付出了硬體或是複雜度的代價，卻換來高壓縮比以及更好的影像品質。在 H.263 中，也有一種類似的增強模式－PB-frame。顧名思義，一個 PB-frame 包含了一個 P-frame 和一個 B-frame，不過不像傳統的 B-frame 會利用之前或之後的影像計算位移向量，H.263 中 B-frame 的兩個向量都是利用 P-frame 的位移向量內插來得到，雖然這樣做會損失一些影像的品質，但是卻可以提高影像壓縮比大約一倍左右。

3.3.9 其他編碼方法

3.3.9.1 G.723.1[10]及G.729[11]

這兩種編碼方式，與G.711 的編碼方法有很大差異。G.711 的編碼方法，基本上是根據時間軸上的聲音波形來編碼，像這種編碼方式一般可稱之為聲源編碼器(Waveform Coders)，其特性是壓縮率不大，但運算也不致太複雜。此外，我們也可以根據聲音的特性來編碼。研究人員根據人類發聲的原理，建立適當的模型和參數，在收到聲音訊號時，試著分析其中的模型參數之值，利用這些參數值和模型，可以將聲音做近似的還原。取出這些參數值的過程稱為分析 (Analyze)，可視之為編碼的過程，接收端接到這些參數後，利用模型將這些參數值還原為聲音，此過程稱之為合成 (Synthesis)，為解碼的過程。像這種編碼器一般亦稱之為語音編碼器(Vocoders) ， 典型的代表為線偵測編碼演算法 (LinearPredictive Coding Algorithm) 。除此之外，還可以混合以上兩種編碼方式的特長來編碼，稱之為混合編碼器(Hybrid Coders) 。其基本原理是利用分析方法先找出一部份聲音特徵，再利用合成的方式，從各種可能的參數之合成結果，中，尋找最接近原來聲音波形的參數組合，這種過程稱為以合成方式所作分析 (Analysis-by-Synthesis) 。由於在尋找參數的過程，中必須做聲音的合成，而此部份正是解碼所做的動作，因此在編碼器即含有解碼器的單元。典型的代表是編碼激發線偵測(Code-Excited LinearPredictive；CELP)編碼，像G.723.1、G.728 和G.729 均屬於此類。關於CELP 的編碼方式，其原理簡述如下。在語音合成的部分(即解碼器)，解碼器根據聲音的參數選擇適當的字碼 (Codeword)，而後取其總和，以形成適當的激發源（這部份可視為模擬人體發體的氣息來源，以及聲帶週期性的振動），而後將此激發源經過一個適當的濾波器(可視為經過聲道、嘴唇、鼻腔等發聲器官產生的效果)，並給多適當的能量大小，即產生合成的輸出聲波。關於語音分析的部分（即編碼器），因為解碼器會根據聲波的一個框架，利用線性偵測演算法，找出適當的濾波器參數，來模擬此段聲波的波包部分（去掉週期性的振動部分所餘之波包，可視為發聲器官產生的效果），得到這些參數之後，再逐一選擇字碼組合為激發源，將其合成輸出聲波，再將此合成之聲波與原來之聲波比對，並根據人耳對音感之認知調整加權，以計算其差異。如此對所有可能的字碼組合一一試過，最後找出最合適之編碼參數，即完成編碼動作。

3.3.9.2 G.711

G.711[9]之編碼即脈衝編碼調變，其編碼率為64 Kpbs，這是最基本的語音編碼，廣泛用在世界各地。PCM 係將輸入之聲波，以每秒8K 的取樣速度，對每一取樣值使用8 bits 做不等量之量化來編碼。根據研究指出，人耳對於同樣的聲音變化，在不同的環境下，會有不同的感受。在安靜的環境下，人耳能感受到細微的變化；而在吵雜的環境下，只能感受到較大的變化。因此在對聲波取樣時，在振幅小的地方應該使用較小的量化值(Quantization Size)，而在振幅大的地方，則使用較大的量化值。若以橫軸為聲波之值，縱軸為取樣之值，則兩者的關係類似一對數曲線，故有時亦稱 log-PCM 編碼。一般又將其細分成兩種，稱為A-law 和μ-law，目前在美國和日本、台灣使用的都是μ-law。

3.4 NAT 與區域網路的限制與解決方式

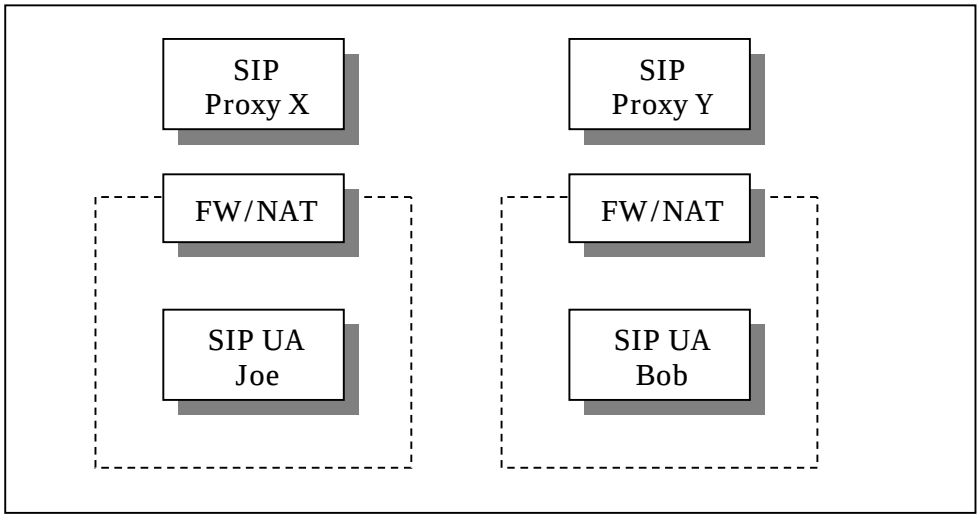
區域網路架構遭遇之限制與困難處

本專題架構底層所實作的 SIP 通訊協定並沒有明確定義區域網路內與其他端點進行協定訊息上傳遞的標準流程，也因此 SIP 通訊協定在這個環境之下也遇到了許多問題，加上區域網路內的使用者無法對防火牆或 NAT 進行操控，在這個環境裡解決使用者對外溝通的情況也變得相當具有挑戰性。現今已經有 IPv6 協定標準的制定，因此解決這個問題的時候，會希望 NAT 會因整個網路架構皆支援此標準而無用武之地，但是這種情況並不可能，因為許多企業或組織會利用此架構來達到安全的機制。

若本身 NAT 架構無法改變的話，解決此問題的另外一個方法是讓 NAT 或防火牆能支援 SIP 協定的 ALG(Application Level Gateway)，但是目前就算是支援最完整的網路裝置仍未必支援其協定封包的處理，因此絕大部分網路裝置皆無法判斷這類型封包的標頭資訊，若要完成大部分網路裝置對此封包的支援，或許還須等待幾年的時間，這種解決方式絕非明智之舉，也因此我們必須另外尋求其他解決方案。

解決方式

我們對此問題的解決方案是以不改變目前網路架構的考量之下進行，因此在實作上必須尋找其他針對該協定擴充的方法，而本專題針對此問題做出的解決方式則是採用 Internet Engineering Task Force SIP Work Group 由 J.Rosenberg,H.Schulzrinne 所提出的 **SIP Traversal through Residential and Enterprise NATs and Firewalls**，其所處的網路架構如下圖所示：



其解決方案的說明分為「訊息傳送方面」、「訊息接收方面」與「串流傳輸方面」，因此本章節也將分為這三個部分做說明。

- **訊息傳送方面**

傳送端將協定訊息透過 NAT/Firewall 傳送至代理伺服器令其轉送並接收代理伺服器傳回的回應訊息。

區域網路內部的 NAT 允許 TCP 和 UDP 封包傳送至外部網路的通訊埠 5060，因此傳送請求訊息並不會遇到任何問題；然而，接收回應訊息方面卻會有下面的問題：根據協定規格對 UDP 傳輸情況所述，回應訊息回傳送至封包中 Via Header 的位址與通訊訊埠，但是封包的標頭已經被 NAT 修改，所以此標頭的位址資訊是錯誤的，也就是回應訊息無法正常傳回至傳送端；對於 TCP 傳輸情況，回應訊息是傳送至接收到請求訊息時所使用的原來連線，因此在這個情況之下可以成功將回應訊息傳回。

根據上面的結論，傳送端以 TCP 傳輸方式進行，傳送與接收並無問題；若以 UDP 傳輸方式進行，接收請求訊息者(User Agent Server)可將動作改成將回應訊息傳送至原接收請求訊息的來源位址和通訊埠，儘管這改變會使流程不符合協定規格，但是卻可以解決這種情況所遇到的問題。

- **訊息接收方面**

接收訊息的情況並不如同前者狀況這樣簡單，因此需要更多處理該問題的步驟。

為了解決這個情況的問題，使用者需要先對註冊管理者進行註冊(Register)訊息的傳送進行註冊手續，如上圖之網路架構所示，傳送端 Bob 會先對註冊管理者(這個裝置與代理伺服 X 部署在同一網域內)傳送註冊訊息，下以即為註冊訊息的內容：

```
REGISTER sip:Y.com SIP/2.0
From: sip:bob@Y.com
To: sip:bob@Y.com
Contact: sip:bob@10.0.1.100
```

很明顯地協定訊息中之 Contact 標頭所記錄的並非廣域的位址，因此其他使用者無法將請求訊息傳送給該使用者，為了解決這個問題，使用者必須以 TCP 傳輸方式傳送註冊訊息並將這個連線保持開啟，以接收代理伺服器轉送過來的請求訊息，而註冊管理者與代理伺服器共存的裝置則在接收到註冊訊息之後，將其連線記錄在連線記錄表之中，其連線記錄包含三個欄位(M, N, O)，其中 M 為網路位址；N 為通訊埠；O 為傳輸方式，這個表格是用來往後查詢特定連線所須。

此時，與使用者 Bob 進行訊息上之連線已經建立完成，而這個連線的鍵值為 sip:bob@Y.com，然而這個位址在往後查詢時會轉換為 sip:bob@10.0.1.100，這並非正確的網路位址，因此註冊管理者在接收註冊訊息時須另外判斷使用者正確的來源位址並建立更正後的 Contact 標頭，由於大部分使用者皆無法確切了解其正確的網路位址，因此在傳送註冊訊息時需告知註冊管理者本身無法知道自己的網路位址，為了解決這個方面的問題，本方案是要求使用者在註冊時在 Contact 標頭的設定為 **jibufobutbmpu** 以告知註冊管理者，有趣的是這個字串所意思是「I hate NATs a lot」，即每個字元往後位移，以這個方式傳送的註冊訊息便是如下所示：

REGISTER sip:Y.com SIP/2.0
From: sip:bob@Y.com
To: sip:bob@Y.com
Contact: sip:bob@jibufobutbmpu

註冊管理者接收到這個註冊訊息之後便會將 Contact 標頭資訊更改為連線的來源位址與通訊息埠，並將此資訊存入連線記錄表，往後如果有其他使用者把請求訊息傳送至 sip:bob@Y.com 時，代理伺服器會轉換為 sip:bob@77.2.3.88:2397 並對應至正確的連線以進行傳送。

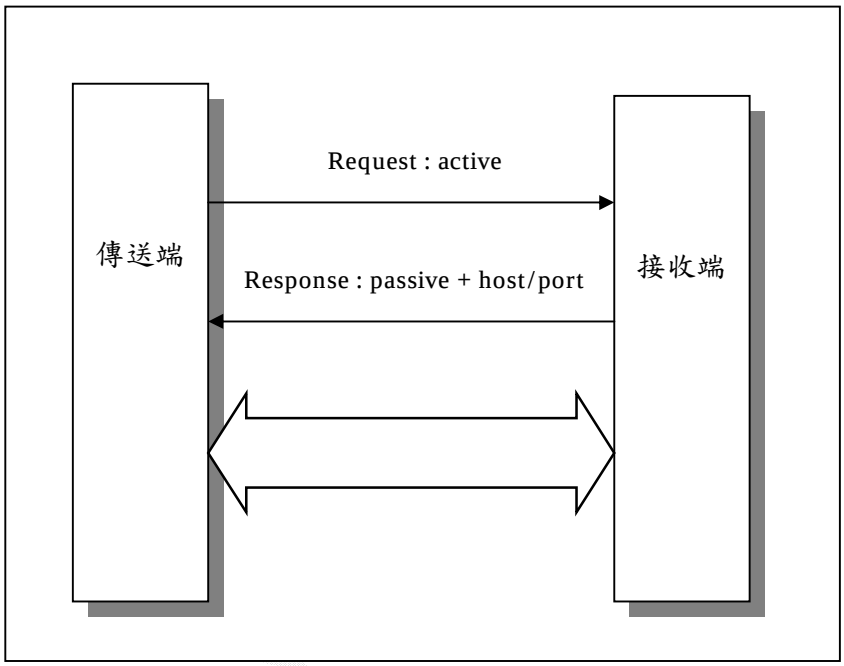
- **串流傳輸方面**

前面兩個主題說明的 SIP 訊息傳遞是屬於較單純的情況，但是對於串流傳輸方面卻顯得複雜許多，本章節之參考文章所討論的利用 SDP 描述建立 RTP 串流之方式亦可應用在一般資料串流的建立上，因此本專題所實作的方式還是使用原文所討論的解決方案，以下將針對兩種種情況分別做介紹與解決方式之說明。

在對這兩種情況做說明之前，有兩個前提必須遵循方能達成此問題之解決方案：1. 位於區域網路內部的使用者必須主動傳送第一個封包以告知 NAT 建立內外網域的繫結；2. 串流的傳送與接收都使用相同的一條通道或管線。

² 其中一個使用者位於區域網路內部

若傳送者位於區域網路內部，其請求訊息中包括 **active** 標記說明傳送端欲主動(Active)對接收端建立連線可告知接收者建立被動(Passive)的連線，而接收者在完成建立被動連線之後，可在回應訊息中加入此被動連線的網路位址與通訊埠，下圖為表示兩個使用者間進行串流建立的流程。

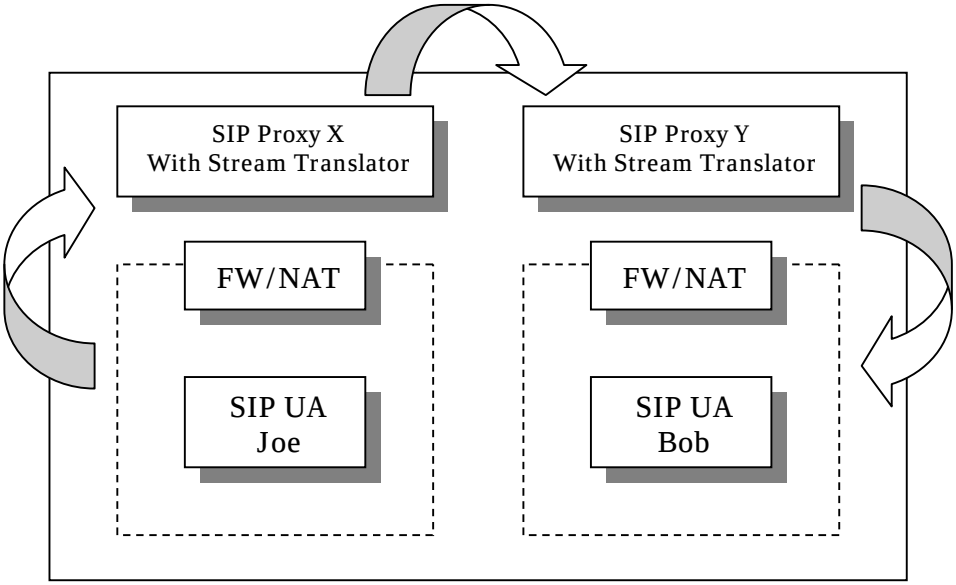


此情況必須確定傳送端是位於區域網路內部，若無法確定其真的網路架構，則須將此視為第二種情況。

² 兩個使用者皆位於區域網路內部

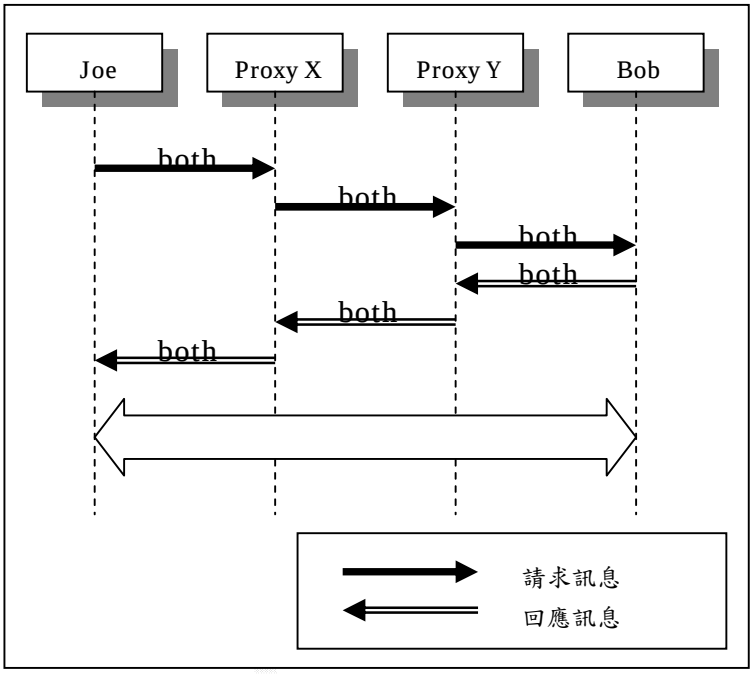
在這種更為複雜的情況之下，整個網路協定架構必須稍做功能上的加強，即代理伺服器必須支援協定訊息內容的解析，並判斷傳送端的網路狀況決定其運作的流程，我們將處理這個運作流程的元件稱為 Stream Translator(原文所稱為 RTP Translator，但本專題所討論的是一般資料串流的建立，因而如此稱之)。

當傳送端欲與接收端進行串流的建立，可在請求訊息中加上 **both** 標記告知傳送端之 Stream Translator 判斷傳送端之網路特性將此標記做適當的修改(**active/passive**)再轉送至接收端之 Stream Translator 進行下一步的判斷及標記的修正，最後再將該請求訊息傳送給接收端並建立資料串流。下圖的網路架構與前面所提及的相近，主要差別僅在於傳送端與接收端的帶理伺服皆支援 Stream Translator 的功能，如下所示：

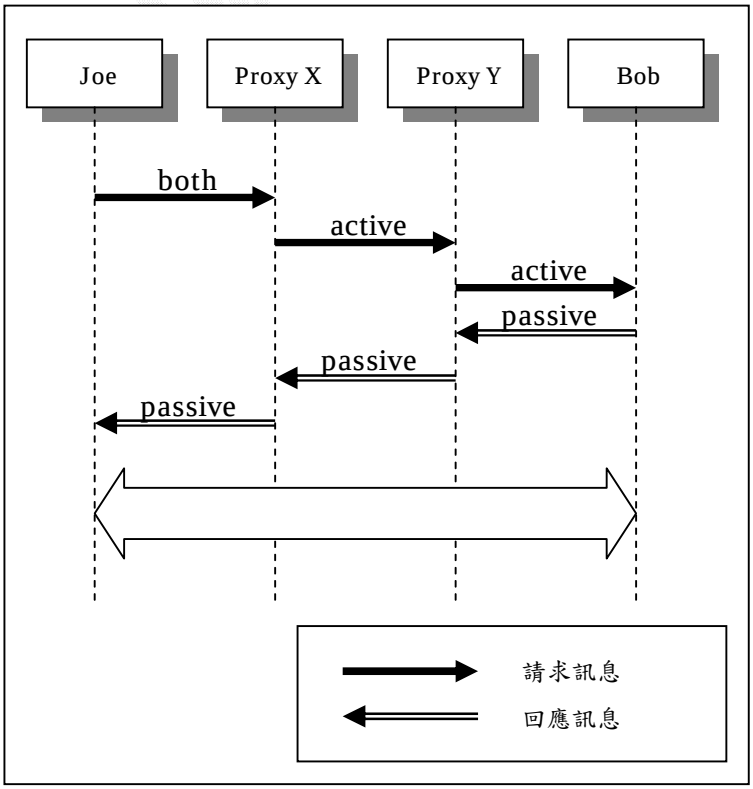


Stream Translator 會根據四種狀況來進行訊息的處理，其處理流程將以順序圖來表示：

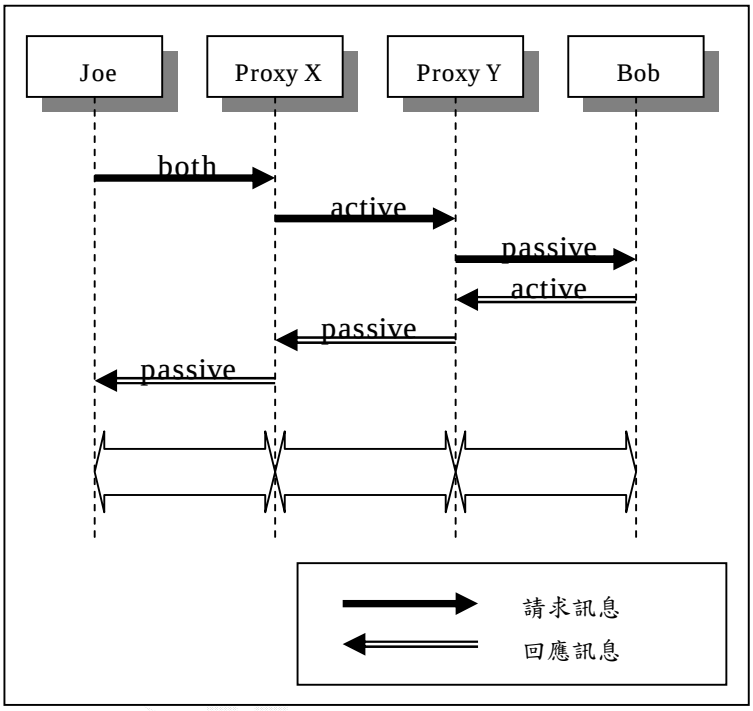
1. 兩者皆位於廣域網路



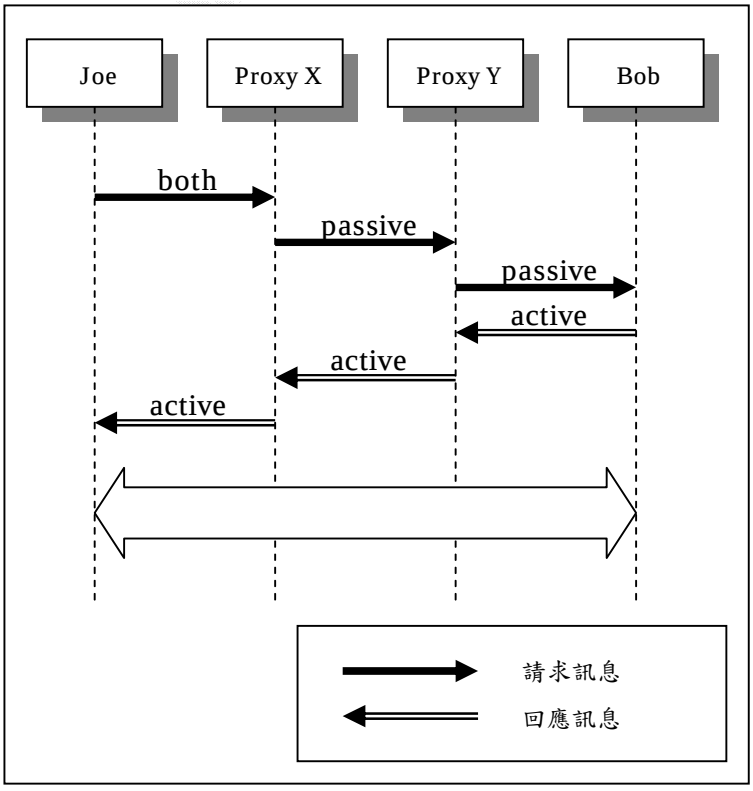
2. 傳送端位於區域網路，而接收端位於廣域網路



3. 傳送端位於廣域網路，而接收端位於區域網路



4. 兩者皆位於區域網路



本專題所使用的 SIP 協定架構中，其訊息內容並非使用原文討論的 SDP(Session Description Protocol)，而是另外定義之端點協定，其中，處理本部分串流傳輸的服務元件即為管線服務元件(Pipe Service)，因為對上層來說，用來傳送串流資料的媒介皆為本服務元件所封裝的管線物件(Pipe)，因此並無這方面的問題，而處理端點間的管線建立之流程即是參考本節之解決方案，其說明會在端點協定一章提出並介紹其使用方式與運作流程。

3.5 Java Web Start 架構說明

Java Web Start 是一種能和瀏覽器產生相關聯的輔助性應用程式，當使用者在網頁中點選一個指向某個特殊啟動檔案(JNLP 檔案)的連結時，瀏覽器就會執行 Java Web Start 並從啟動檔案所描述的路徑下載或是從快取目錄中執行以這種技術為基礎的應用程式。

當然，前面提到的 JNLP URLs 可以從 JAWS 應用程式管理員(JAWS Application Manager)直接開啟並儲存到書籤，此種檔案的副檔名甚至也可以是.html 或.jnlp。

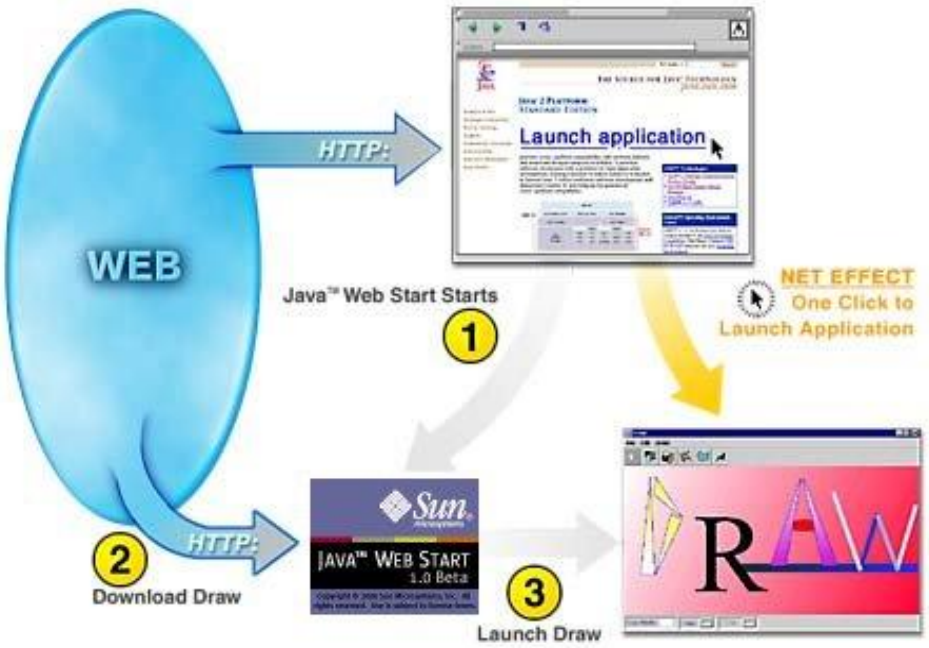
從一個技術的立場來說，Java Web Start 在佈署應用程式方面之所成為一個吸引人的平台在於以下幾種關鍵優勢：

- Java Web Start 的功能在於啟動由 Java 2 SE 平台發展而成的應用程式，因此，每個應用程式可存放在網頁伺服器以供使用者存取，這種方法的好處在於程式發展者可以將應用程式佈署在各種作業平台中，包括 Windows98/NT/2000/ME/XP、Linux 和 Salaris™ 。由於這樣的跨平台特性，以此為開發平台的應用程式便相當地強韌且具生產力，加上發展與測試的花費減低，便節省了許多發展經費。

- Java Web Start 支援多種版本的 Java 2 SE 平台，因此應用程式可以要求它所需要的特定平台版本，例如 J2SE™ 1.4.0。在 Java Web Start 之上的各種應用程式可各自以不同版本的平台同時執行而且不會導致之間的衝突，如果應用程式發現用戶端並沒有安裝所需要的平台版本時，Java Web Start 也可以自動下載並重新安裝平台的更新版。

- Java Web Start 允許應用程式由瀏覽器各自獨立執行，當瀏覽器關閉或是之後再次啟動特定應用程式時，它也可以用來處理這些功能，因為從瀏覽器呼叫 Java Web Start 並間接啟動特定應用程式對使用者來說是相當不方便且不可行的。

- Java Web Start 直接沿用 Java 平台安全性的優點，應用程式預設上是在保護模式的環境(sandbox)之下執行，所有本地端磁碟和網路資源的存取都會受到限制，當應用程式並未受到信任時，使用者仍可執行該應用程式而無安全上的虞慮。
- 透過 Java Web Start 執行的應用程式是在本地端快取的，因此先前已經下載過的應用程式便直接在本地端快取目錄呼叫。

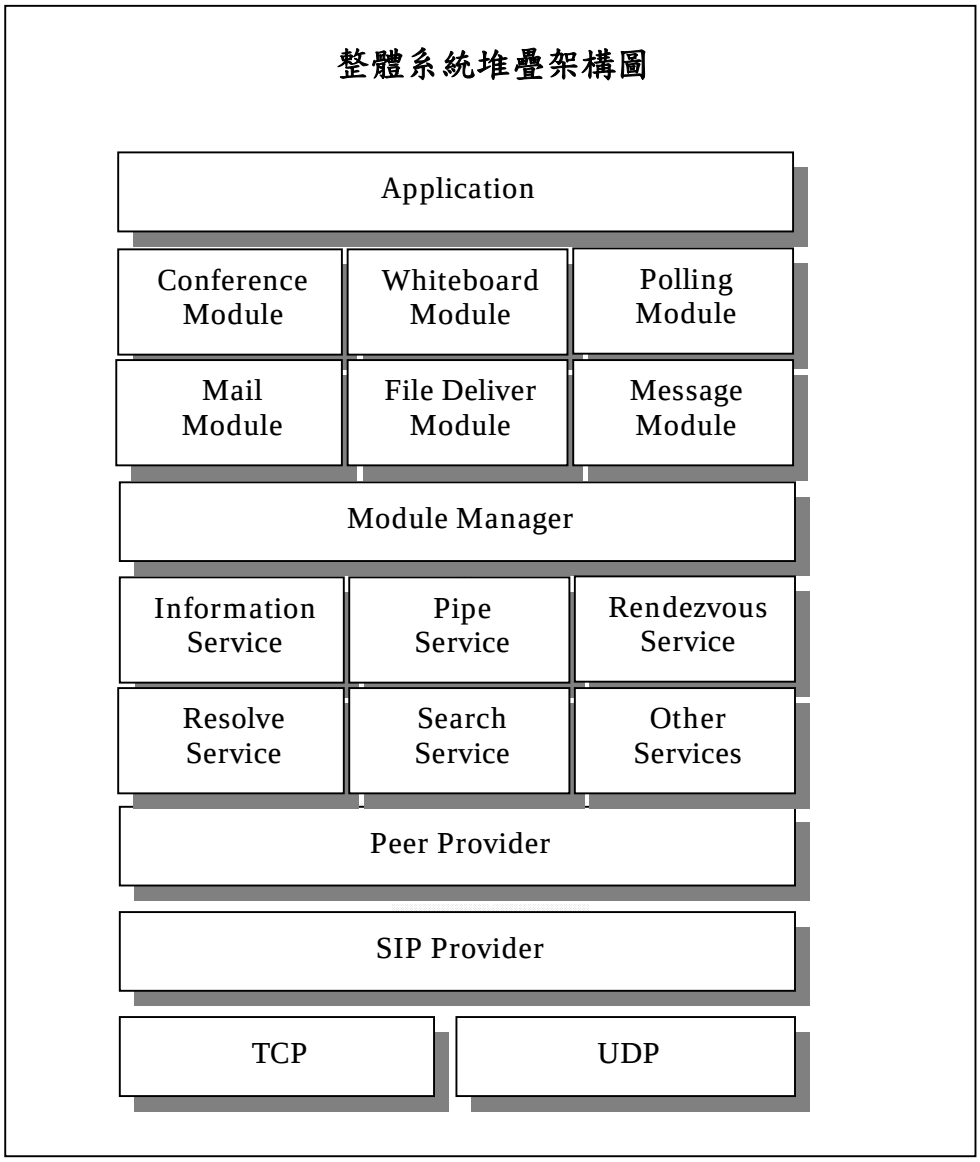


Java Web Start 架構圖

第四章　系統架構及實作

4.1 整體架構概觀

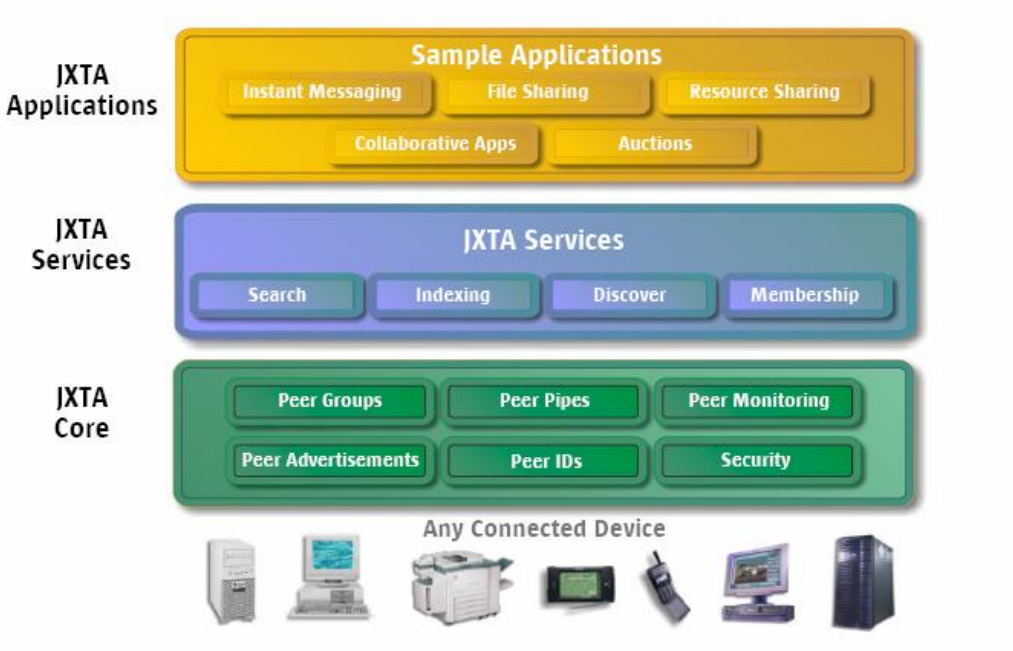
整體系統堆疊架構圖



4.2 端點協定

本專題底層端點協定架構是建構在 SIP 通訊協定之上，而在端點訊息的傳遞流程與基礎服務功能上是參考 JXTA 之端點協定技術，以下將對 JXTA 技術與本專題參考實作的架構做進一步的介紹。

JXTA 端點技術提供了一系列開放式通訊協定，該通訊協定允許小至手機和 PDA、大至個人電腦和伺服器等連線裝置的點對點通訊。在 JXTA 端點所建立的虛擬網路之內，即使有些端點位於防火牆或 NAT 之後或是使用不同架構的通訊協定，任何一個端點仍然可以直接和其他端點和資源溝通。



JXTA 整體軟體架構共分為三種層次：

- 平台層(Platform Layer)
  平台層又稱為 JXTA 的核心，其主要是將所有 P2P 網路架構所必須支援的共通基礎功能封裝其中，包括端點搜尋、媒體傳輸(例如通過防火牆之管控)、端點與端點群組之建立及其他安全性的機制。

- 服務層(Service Layer)
  服務層包括 P2P 網路內部必須支援才能正常運作的網路服務，但是不同的 P2P 環境內皆有不同支援程度的各類服務，其服務包括搜尋、建立目錄索引、儲存系統、檔案分享與分散式檔案系統、資源集結與租用、協定間之轉換、認證與 PKI(Public Key Infrastructure)等服務。

- 應用層(Application Layer)
  應用層包括功能整合的應用程式之實作，例如點對點訊息傳送、文件與資源分享、娛樂媒體內容管理與發佈、點對點電子郵件、分散式拍賣系統以及其他可在端點網路上發揮的功能。

JXTA 網路中是由一群相互連結的節點(或稱為端點)所組成，而端點之間可選擇與其他端點組成一個具有共通服務之端點群組，端點群組提供的服務則包括文件分享與聊天等。JXTA 端點發佈其支援服務是利用 XML 文件處理，該文件稱為 **Advertisement**，其主要功能在於讓其他端點能透過其他描述瞭解如何對該端點服務進行連線。關於訊息的傳遞方面，JXTA 端點是利用其特有的**管線(pipe)**將訊息傳遞給其他端點，管線是一種非同步且不具方向性的傳送機制，端點服務便是透過該機制進行溝通，其溝通的訊息也是一種將路由、摘要與憑證資訊封裝其中的 XML 文件，另外，管線是與特定端點(Endpoint)繫結而成，及 TCP 通訊埠及其位址。

JXTA 協定架構的主要特色：

以下三種特色是 JXTA 協定架構與其他分散式運算架構之主要相異處。

- 以 XML 文件描述(Advertisement)特定端點所提供的網路資源。
- 以特定的抽象介面提供給上層使用(Pipe -> Peer -> Endpoint)。
- 統一的端點定址方式(Peer IDs)

本計畫著重於點對點之間的直接通訊，有鑑於 JXTA 技術的架構因必須處理各技術領域內通用的溝通機制，致使整體架構相當龐大，加上該技術有部分端點協定並不適用本計畫的實作(如端點群組的機制)，因而，本計畫以 JXTA 技術架構作為範本再加以簡化，發展出自訂的點對點通訊協定〔JEMI Peer Protocol〕。

本端點協定架構是由各自分散的端點所組成，當一個端點欲向其他端點請求服務時，可對之進行直接的要求，但是因為端點在進入端點網路時無法得知其他端點的實際位址，所以端點網路內需要一個負責管理各端點的中央伺服器，其功能主要負責存放加入該端點網路的所有端點清單，當一個端點欲向其他端點請求服務前，必須對中央伺服器請求該遠端端點的位址〔本地端點也可在快取目錄中搜尋該端點的位址以減低搜尋時間〕，方能對其他端點進行服務請求，但是該快取的端點位址須設定其有效的時限，以防止某特定端點位址並非實際的目標位址。

4.2.1. 端點位址之表示方式〔Peer Addressing〕

本專題所設計的端點協定架構中並沒有特定的實作方式，因此在端點定址上僅提供一個簡單的端點位址物件(Peer Address)表示，該物件提供兩個存取位址資訊的方法：端點識別子(Identifier)與端點位址之解析資訊(Resolved Data)，這兩個端點位址資訊會因為底層實作上的不同而有所差異，為了避免上層架構會因為實作上的不同而導致有平台相依的問題，因此僅提供這個物件給上層元件使用，藉此降低使用上的困難度及管理上的複雜度，以下是端點位址之格式：

PeerAddress := Identifier @[ResolvedData]
Identifier := (A-za-z0-9)+
ResolvedData := URI

4.2.2. 端點協定訊息〔Peer Protocol Message〕

本專題所使用的端點協定訊息傳遞格式也是 XML-based，其優點在於無作業平台之間的差異性，因而具有足夠的可攜性與延伸性，加上協定訊息裡的資訊是以人類可以辨識的文字表示，所以對於發展上層應用程式的測試與除錯，無疑是一大幫助。

端點協定訊息資訊是存放在 SIP 協定訊息的訊息內容〔message body〕之內，共分為請求訊息〔Request〕和回應訊息〔Response〕，一般而言，傳送請求訊息並接收回應訊息期間稱為一個會議〔Session〕，協定封包訊息裡的標頭功能在於控制該封包訊息的傳送流程：來源端點位址〔FromAddress〕表示建立並傳送該封包訊息的端點位址；目的端點位址〔ToAddress〕則表示傳送欲傳送至的端點位址；轉送位址〔ViaAddress〕記錄該協定封包訊息傳送經過的端點位址；每個協定封包訊息皆有屬於封包訊息的識別碼〔SessionId〕，藉以辨識出每個封包訊息所在的會議編號，以下為端點協定訊息的標頭資訊列表：

1. FromAddress — 傳送該協定封包訊息的端點位址。
2. ToAddress — 接收該協定封包訊息的端點位址。
3. SessionId — 該協定封包訊息的唯一識別碼。
4. Parameter — 該協定封包訊息的參數。

- ü 請求訊息〔Request〕
  請求訊息除了共通之基本協定訊息標頭外，另外加上請求訊息所特有的兩種訊息標頭，以指定請求之服務名稱與方法：
  - n 端點服務名稱〔Request Service〕
    指定端點欲對其他端點進行請求的服務名稱。
  - n 服務請求方法〔Request Method〕
    指定請求端點服務的方法類型。

下圖為請求訊息之範例：

```
<?xml version="1.0" encoding="utf-8"?>

<Msg Type="Req">
 <FromAddr>jemi:lamer@[sip:Presario700:3035]</FromAddr>
 <ToAddr>jemi:server@[sip:Presario700:5080]</ToAddr>
 <SID>bT272jlYCkHxemAo3inkXHh5VLB321B7</SID>
 <Srv>jemi.peer.rendezvous.RendezvousService</Srv>
 <Mthd>LOGIN</Mthd>
 <Param>
  <RdvIvl-0>***</RdvIvl-0>
  <RdvInm>UserId</RdvInm>
 </Param>
</Msg>
```

- 回應訊息〔Response〕
  回應訊息除了共通之基本協定訊息標頭外，另外加上回應訊息所特有的兩種訊息標頭，以指明回應服務之名稱與狀態：

  - 端點服務名稱〔Response Service〕
    指明被要求之端點服務所回應的服務名稱。

  - 服務回應狀態〔Response Status〕
    指明被要求之端點服務所回應之狀態，該狀態為 4 bytes 的整數，其數字格式為「0xdddddd」。在本專題之端點協定架構中，共有以下幾種內建的回應狀態：

    - STATUS_SUCCESS(0x00)
      表示服務處理請求完成。
    - STATUS_MESSAGE_FORMAT_UNKNOWN(0x10)
      表示無法解析請求訊息之格式。
    - STATUS_MESSAGE_TYPE_UNKNOWN(0x11)
      表示無法分辨接收訊息之類型。
    - STATUS_MESSAGE_VERSION_UNSUPPORTED(0x12)
      表示不支援訊息之版本編號。
    - STATUS_MESSAGE_INCOMPLETE(0x13)
      表示接收到的訊息不完整。
    - STATUS_MESSAGE_TOO_LONG(0x14)
      表示接收到得訊息長度過長。

    - STATUS_ADDRESS_CONFLICT(0x15)
      表示訊息中含有衝突的位址。
    - STATUS_PEER_NOT_FOUND(0x20)
      表示欲傳送的端點位址不存在。
    - STATUS_PEER_TIMEOUT(0x21)
      表示無法在特定時間內收到回應訊息。
    - STATUS_SERVICE_NOT_FOUND(0x30)
      表示端點中找不到指定的服務。
    - STATUS_SERVICE_METHOD_UNKNOWN(0x31)
      表示服務不支援指定的請求法。
    - STATUS_SERVICE_PARAMETER_REQUIRED(0x32)
      表示服務需要取得特定參數才能正常運作。
    - STATUS_SERVICE_ERROR(0x33)
      表示服務處理期間發生錯誤。
    - STATUS_UNAUTHORIZED(0x40)
      表示端點需通過認可才能執行某服務之功能。

下圖是回應訊息之範例：

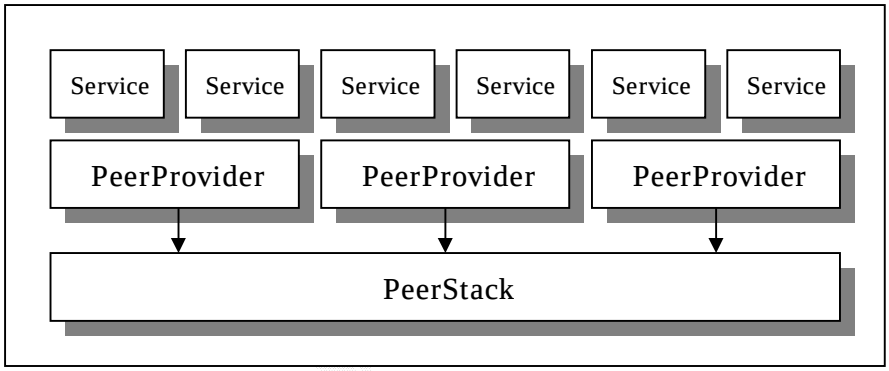
```
<?xml version="1.0" encoding="utf-8"?>

<Msg Type="Res">
 <FromAddr>jemi:lamer@[sip:Presario700:3511]</FromAddr>
 <ToAddr>jemi:server@[sip:Presario700:5080]</ToAddr>
 <SID>9J1GRGrb30D9RWC4GZzUD8iPXnttCL19</SID>
 <Srv>jemi.peer.rendezvous.RendezvousService</Srv>
 <Stat>0x00000000</Stat>
 <Param>
  <RdvIvl-4>%C2%BEG</RdvIvl-4>
  <RdvIvl-3>%C3%80s%C2%B3%C3%8D</RdvIvl-3>
  <RdvIvl-2>LaMer</RdvIvl-2>
  <RdvIvl-1>%22SKL2%22+%3Cskl2%3E</RdvIvl-1>
  <RdvIvl-0>0</RdvIvl-0>
  <RdvInm>Gender</RdvInm>
  <RdvInm>ContactUsers</RdvInm>
  <RdvInm>NickName</RdvInm>
  <RdvInm>LastName</RdvInm>
  <RdvInm>FirstName</RdvInm>
 </Param>
</Msg>
```

4.2.3. 端點堆疊〔Peer Stack〕

端點堆疊的功能在於建立、刪除並管理其下所有的端點提供者，建立端點提供者期間，上層程式可對端點堆疊進行選項設定，以對端點提供者控制其連線或運作流程，例如端點繫結的位址和通訊埠、傳輸模式、位址解析伺服器與代理伺服器等設定，以下是端點堆疊的基本選項：

- PATH：端點堆疊指定建立的實作類別。
- HOST：端點繫結的位址。
- PORT：端點繫結的通訊埠。
- TRANSPORT：端點訊息的傳輸模式。
- RESOLVER：端點進行位址解析的伺服器位址。
- PROXY：端點所使用的代理伺服器位址。

下圖即為端點堆疊與其上層端點提供者之所屬關係：



4.2.4. 端點提供者〔Peer Provider〕

端點提供者是本專題中整體端點架構中的核心元件，即表示一個端點的主體，其功能主要有以下兩種：

- 管理、建立與刪除其下的服務元件

  因為端點提供主要功能在於接收訊息並傳遞至上層的特定服務元件，所以端點提供者在建立服務元件時須提供服務元件進行端點訊息聆聽的介面，以完成訊息的傳遞的機制。

  另外，端點提供者在建立時皆會預先建立五種內建的服務元件，InformationService、PipeService、ResolveService、RendezvousService 及 SearchService，上層程式可利用端點提供者提供的五種方法取得這些內建服務元件：

  - getInformationService() : InformationService
  - getPipeService() : PipeService
  - getResolveService() : ResolveService
  - getRendezvousService() : RendezvousService
  - getSearchService() : SearchService

  若端點提供者欲搜尋或取得其他單一服務或是所有服務名稱清單，以下有存取各服務元件的方法：

  - getServiceNames() : String[]
  - hasServices() : boolean
  - hasService(String serviceName) : boolean
  - getService(String serviceName) : Service

  端點提供者另外提供了新增或刪除特定服務元件的方法，以下即為完成這些功能的方法：

  - addService(String serviceName) : Service
  - removeService(String serviceName) : Service
  - removeServices() : void

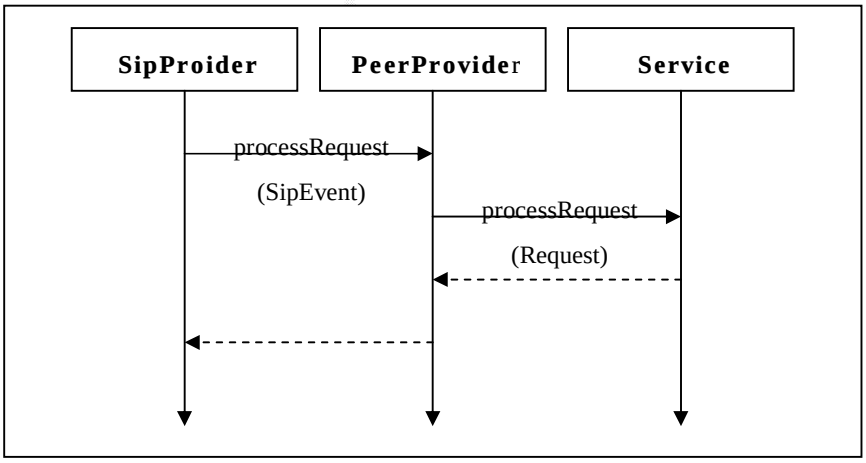
- 傳送及接收其他端點的協定訊息

上層服務元件對端點提供者提出的要求主要是協定訊息的傳遞，這也是端點提供者工作量最為頻繁的動作之一，提供傳遞之訊息類型有要求訊息(Request)及回應訊息(Response)，而其提供之方有為下列兩種(請求/回應)：

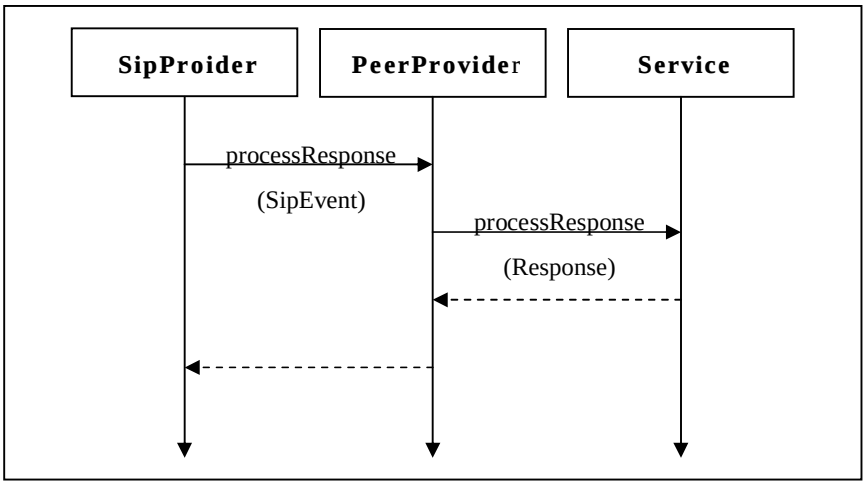
- sendRequest(Request request) : void
- sendResponse(Response response) : void

端點提供者也負責聆聽下層 SipProvider 所傳送的事件，當端點提供者被底層告知接收到 SipRequest 和 SipResponse 協定訊息，則從 SIP 協定訊息內容中解析出端點協定訊息，當端點提供者成功解析出正確的訊息後，便判斷該端點協定中如請求或回應等訊息類型，端點提供者需依照端點訊息指示請求或回應的服務名稱將接收到的訊息傳至上層的進服務層，以下為端點提供者對訊息(分為請求與回應兩種情況說明)進行處理的順序圖：

處理請求訊息的順序圖：

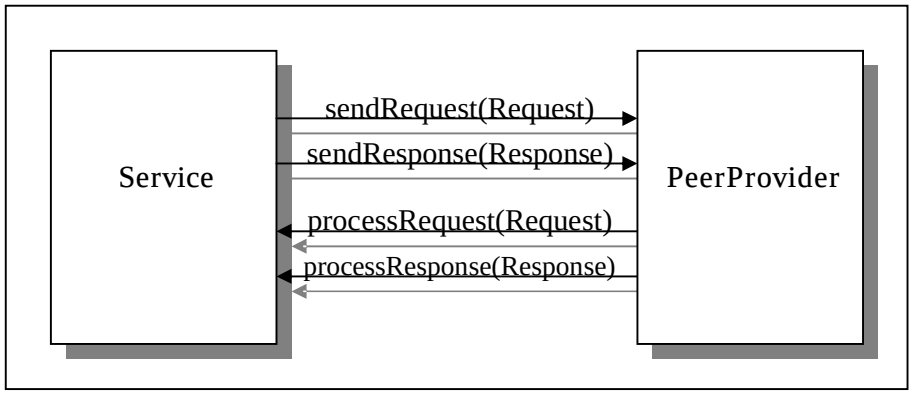


處理回應訊息的順序圖：



4.2.5. 服務元件〔Service〕

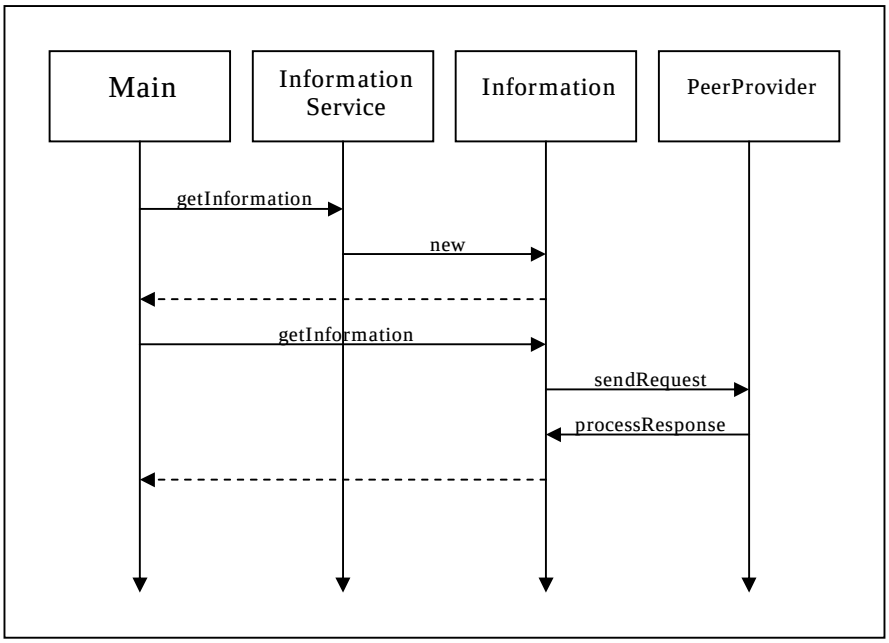
服務元件位於端點提供者之上的服務層，該層中包含端點支援所有服務的集合，服務元件的呼叫透過端點提供者進行，服務元件的建立、摧毀或對服務元件本身的其他管理也是由端點提供者負責，而服務元件與端點提供者之間的訊息傳遞可由下圖表示：



本端點協定架構依照端點支援的內建或額外服務分為六種服務類型，以下將對這六種服務元件做完整的介紹：

4.2.5.1. 資訊服務元件(Information Service)

資訊服務元件主要功能在提供端點查看其他端點所擁有的資訊，如端點名稱、使用者基本資料、端點協定版本編號、登入端點網域名稱、可支援服務清單與功能等，另外，應用程式也可依照本身需求將其他特定的應用程式資訊也加至其中以供其他程式取得，下圖為端點對其他端點取得資訊的順序圖。

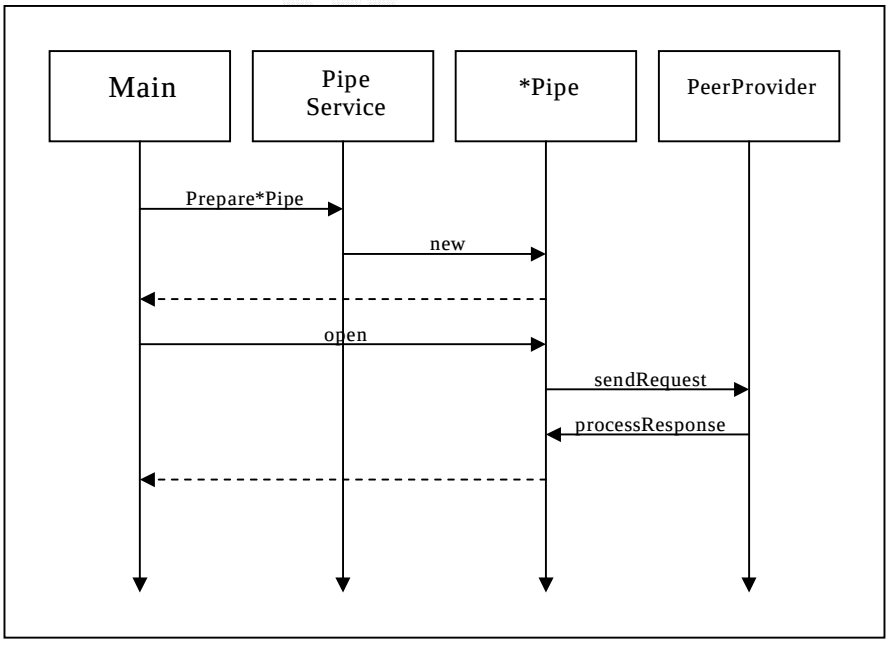


4.2.5.2. 管線服務元件(Pipe Service)

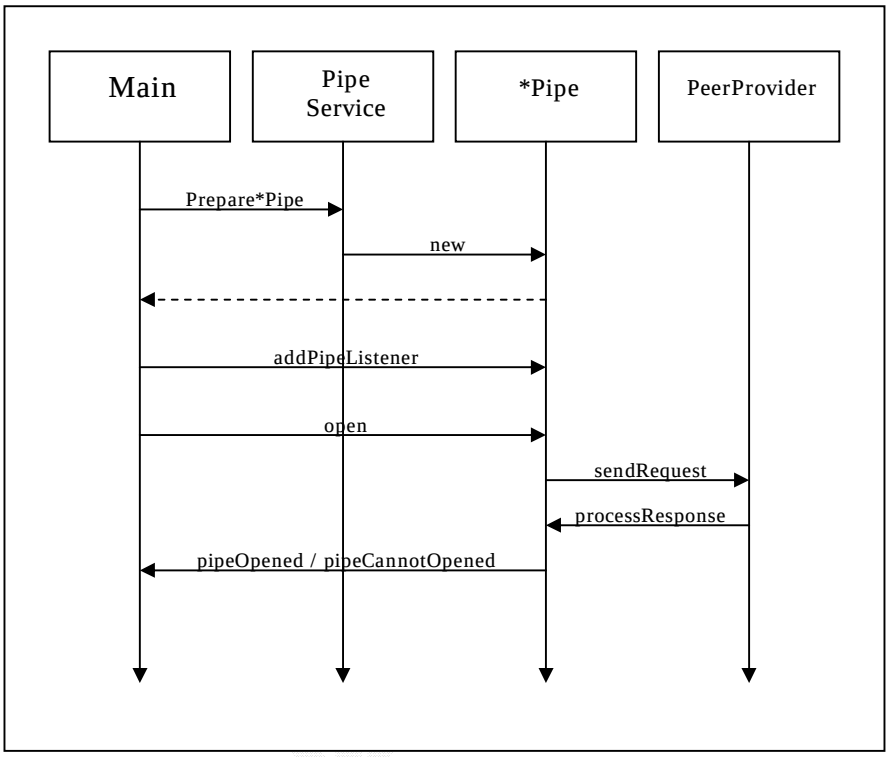
管線服務元件負責管理端點和其他端點之間傳送資料串流的所有管線，其功能有建立、摧毀及查詢目前所開啟的管線資訊，這個服務元件也是整個端點架構中使用最為頻繁的部分，因為端點在與其他端點以特定資料做相互溝通時，必須彼此建立對等的通道以進行訊息的傳送與接收，常用的資料傳送有文字訊息、檔案傳輸甚至是影音媒體串流的傳遞等。

管線服務元件所支援的管線類型共有三種：**輸入管線(Input Pipe)**、**輸出管線(Output Pipe)**及**雙向管線(Bidirectional Pipe)**，與其他端點建立管線的方式皆分為兩種類型：**同步(Synchronous)**與**非同步(Asynchronous)**建立類型，此兩種建立方式的差別僅在於準備管線時是否註冊聆聽管線之事件及呼叫開啟管線時是否指定管線建立的時限，以下為這兩種管線建立類型之執行順序圖(三種管線類型的建立方式皆相同，所以此順序圖適用所有類型之管線建立)。

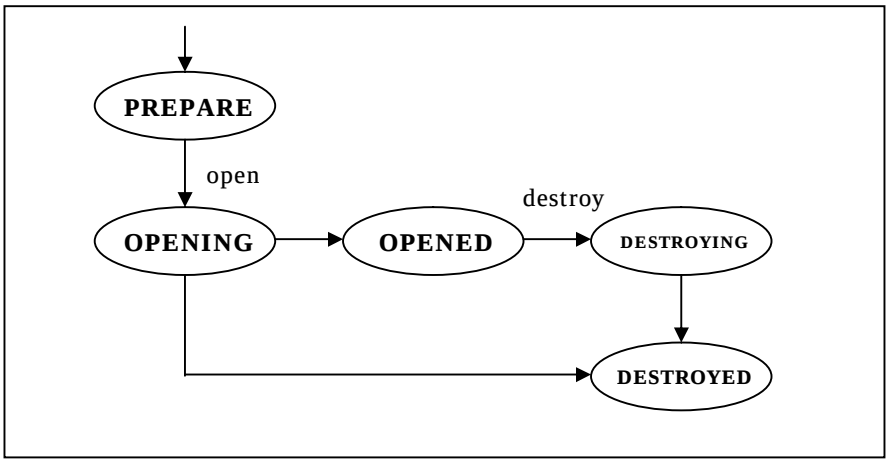
同步類型管線建立之順序圖：



非同步類型管線建立之順序圖：



端點與端點之所建立的管線會依使用者呼叫不同的方法而有狀態的轉變，下圖即為管線之狀態轉換圖：



4.2.5.3. 集結服務元件(Rendezvous Service)

集結服務元件提供端點向鄰近端點管理者進行端點網路內的登入、登出或註冊之服務，其支援的方法有 LOGIN、LOGOUT、MODIFY 及 REGISTER 等，功能分別為登入、登出、修改及註冊，另外，集結在傳遞資訊時是利用 RendezvousContext 存放，藉以指示其他端點需提供的服務項目。

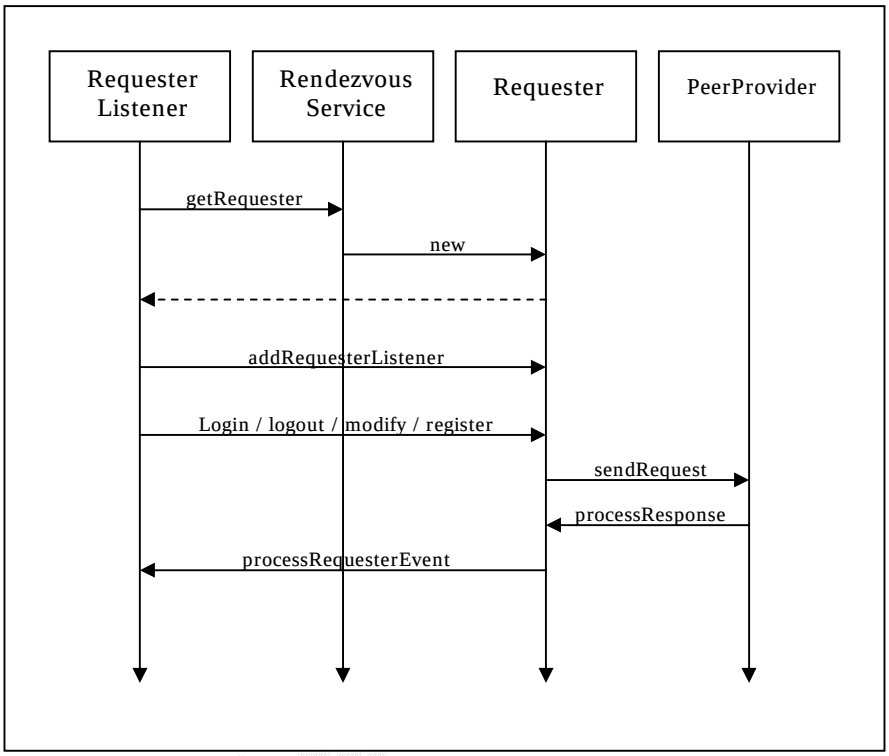
這個服務元件所管理的物件又可分為兩種類型：**請求者(Requester)**與**回應者(Responsor)**兩種，故名思義這兩種物件的功能分別為對其他端點請求服務與提供其他端點對服務的請求，而集結服務元件對於以上兩種物件之管理如下：

1. 請求者的取得方式
   getRequester(PeerAddress) : Requester
   【註】這個方法呼叫時所使用的參數為端點位址，表示請求者欲進行集結服務請求的對象。

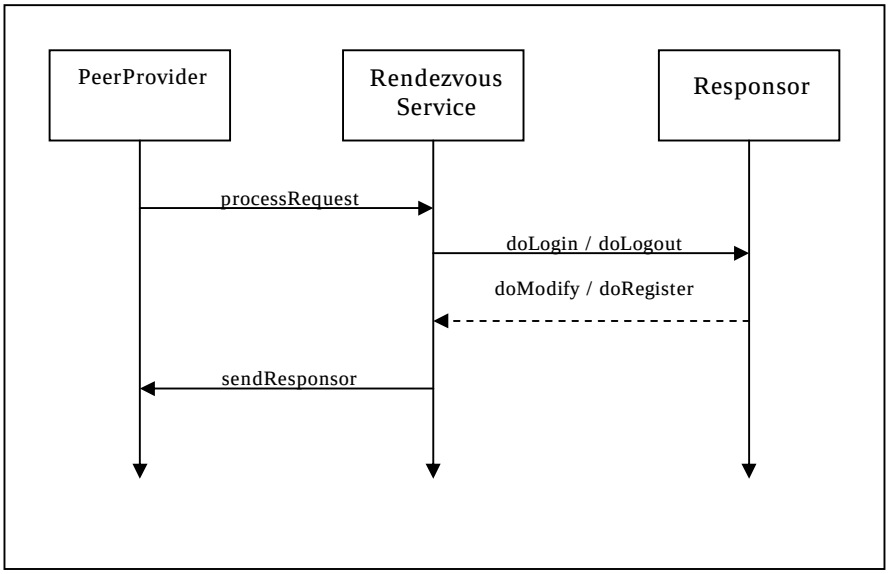
2. 回應者的取得與設定方式
   hasResponsor() : boolean
   getResponsor() : Responsor
   setResponsor(Responsor) : void
   removeResponsor() : void
   【註】回應者所提供的服務可依照實作者的需求而有所不同，因此端點間須事先知道彼此所使用的實作是否相同。

請求者和要求者在支援的方法上提供相對的呼叫方式，以下即為兩者在支援的方法上的執行順序圖(四種方法的流程皆相同)。

請求者之執行順序圖：



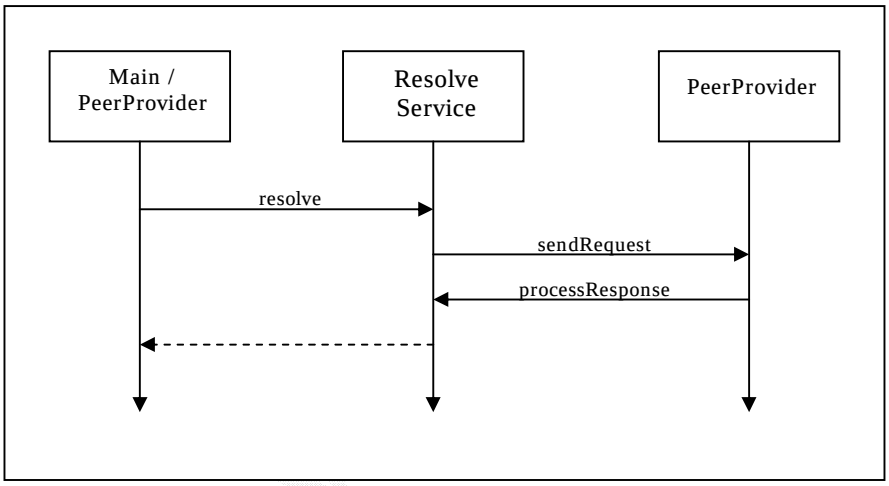
回應者之執行順序圖：



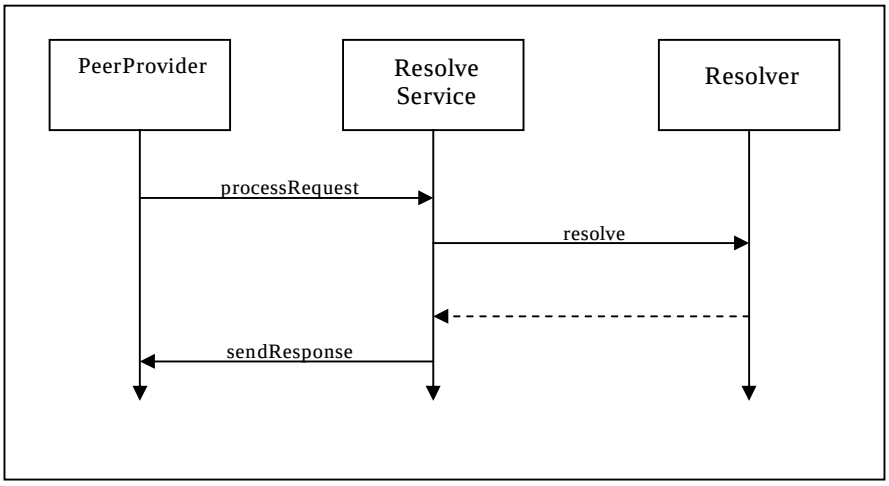
4.2.5.4. 解析服務元件(Resolve Service)

解析服務元件通常不直接提供服務給上層元件使用端點，其使用情況是當傳送協定訊息時發現目的端點位址尚未取得解析後的資料，端點提供者會透過其本身所有的解析服務元件對其他端點管理者(通常是端點所登入端點網路之管理者)請求該端點位址的解析服務，而後完成該協定訊息的傳送，其解析的功能便是端點位址對應至實際網路位址的轉換，負責的物件即為 Resolver，同樣地，該物件的處理方式會因為端點網路架構的實作不同而有所差異。以下是解析服務元件在執行時的順序圖：

請求端點位址解析之執行順序圖：



解析者處理端點位址之執行順序圖：



4.2.5.5. 搜尋服務元件(Search Service)

搜尋服務元件提供端點對鄰近端點管理者進行其他端點資源的搜尋，例如端點所分享的檔案或媒體串流等資源。為了讓上層的元件能簡易地使用該搜尋服務元件所提供的搜尋功能，在其下又定義了兩種搜尋物件：**搜尋條件(Search Term)**及**搜尋結果(Search Result)**，前者的功能在於讓上層能將欲搜尋之條件陳述句設定至該物件中，並以此物件為參數啟動搜尋服務元件的執行；而後者則是當處理搜尋條件的服務元件完成搜尋後，將搜尋結果包裝其中讓上層取用，以下將說明這兩種物件的屬性與使用方式：

1. 搜尋條件(Search Term)

   搜尋條件中含有三種欄位資訊：搜尋選項(Option)、條件陳述句(Expression)、搜尋筆數限制(Threshold)，其中，搜尋選項之用途在於讓應用程式能依其所需指定其特有的搜尋方式；條件陳述句則是該次搜尋所描述的搜尋過程；而搜尋筆數限制則可讓搜尋服務元件所搜尋的資料筆記縮減至原設定之值，以降低網路頻寬的耗費。下圖為該物件的呼叫介面：

SearchTerm
hasOption(String name) : boolean
getOptions() : String[]
getOption(String name) : String
addOption(String name, String value) : void
removeOption(String name) : void
getExpression() : String
setExpression(String expression) : void
hasThreshold() : boolean
getThreshold() : int
setThreshold(int threshold) : void
removeThreshold() : void

2. 搜尋結果(Search Result)

搜尋結果存放的是搜尋服務元件完成搜尋動作之後的結果，其存放的方式是採用關聯式隨機存取，即類似關聯式資料庫之存取方式，此方式有便於存取的優點，所以該物件便採用此種存取方法，另外，由於 JDBC 介面在使用上的廣泛性及其便利性，本物件之呼叫介面是參考其 ResultSet 類別並簡化得來，以下將介紹其主要的功能與用法。

對搜尋結果物件進行存取是採用游標(Cursor)指位的方式將目前正在處理的記錄指定，當使用者將游標指定至某筆記錄之後，往後的存取皆是針對該筆記錄做處理，在搜尋結果物件被初始化時，其游標位置是在第一筆記錄之前，此設定的主要原因是讓使用者能以簡單的迴圈達到讀取所有資料的目的，下面範例即說明讀取搜尋結果物件中所有記錄之方法：

```
SearchResult searchResult = ***;
while (searchResult.next()) {
    // 存取此筆記錄之特定欄位
}
```

另外，若使用者欲對該搜尋結果物件進行反向的存取，以下程式碼即可達到反向存取的需求：

```
SearchResult searchResult = ***;
searchResult.afterLast();
while (searchResult.previous()) {
    // 存取此筆記錄之特定欄位
}
```

除了以上兩種循序存取的方式之外，搜尋結果物件另外提供六個以隨機方式設定游標位置的方法：absolute、relate、beforeFirst、first、last 和 afterLast 等。下表為各方法呼叫之後游標所設定的位置(令記錄筆數為 x)

呼叫方法	游標位置(呼叫前)	游標位置(呼叫後)
absolute(n)	m	n
relate(n)	m	m+n
beforeFirst()	m	-1
first()	m	0
last()	m	x-1
afterLast()	m	x
previous()	m	m-1
next()	m	m+1

搜尋結果物件對於某記錄中特定欄位的存取則提供兩種 get/set 的方法：其一是透過指定的欄位名稱回傳所要的值；其二則是以欄位之索引編號來取得所要的值。因為記錄中所有欄位型態僅限定於字串型態，所以在取用資料時需自行處理方能使用，以下即為搜尋結果物件所提供存取的方法：

- get(String columnName) : String
- get(int columnIndex) : String
- set(String columnName, String columnValue) : void
- set(int columnIndex, String columnValue) : void

搜尋結果物件另外提供一個方法讓使用者能取得該物件所存放的欄位清單，以便往後對單一欄位存取時之用：

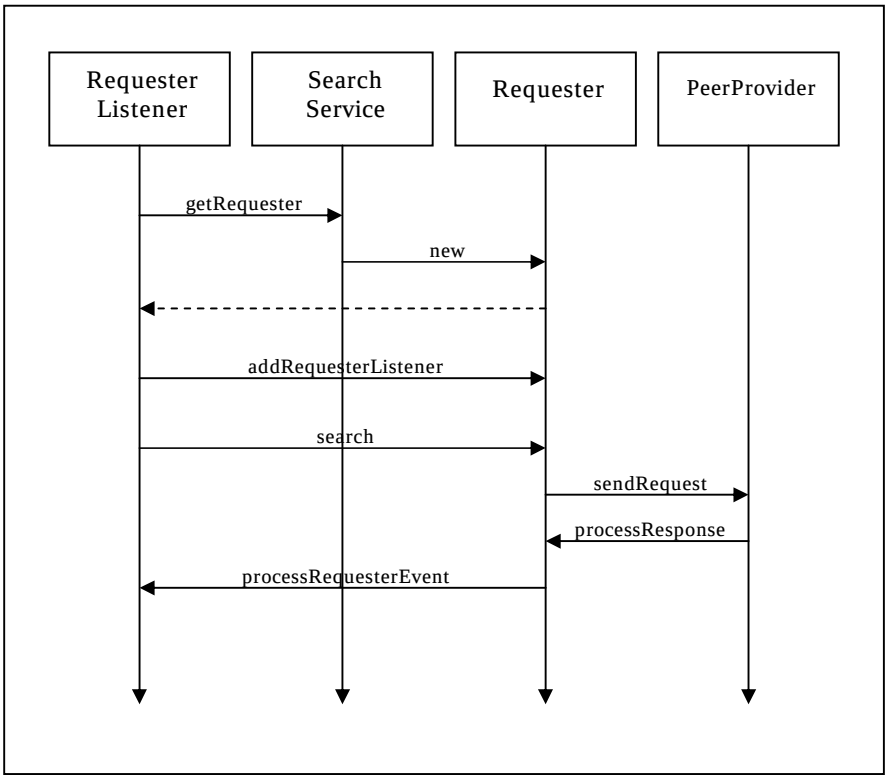
- getColumnNames() : String[]

下圖為該物件的呼叫介面：

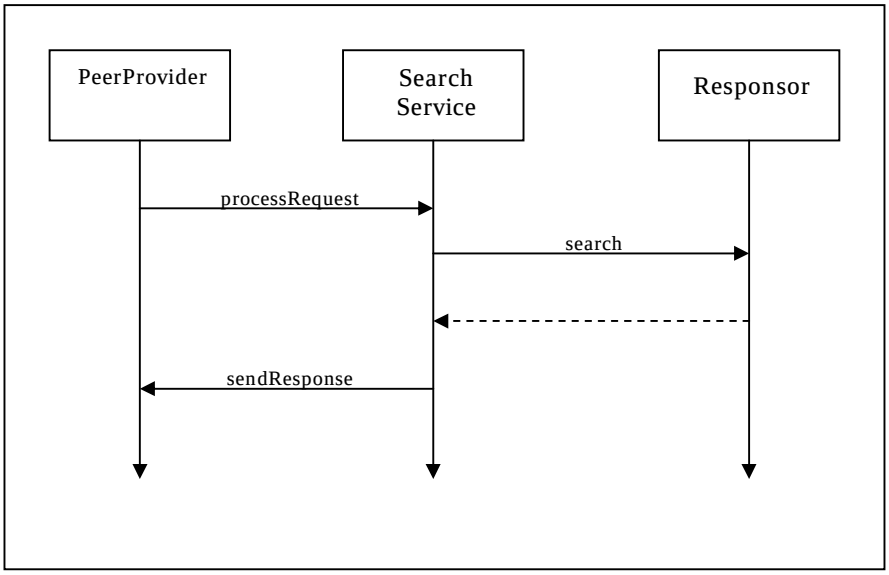
SearchResult
isBeforeFirst() : boolean isFirst() : boolean isLast() : boolean isAfterLast() : boolean getSize() : int getRow() : int absolute(int row) : boolean relative(int rows) : boolean beforeFirst() : void first() : boolean previous() : boolean next() : boolean last() : boolean afterLast() : void addRow() : void getColumnNames() : String[] setColumnNames(String[] columnNames) : void get(String columnName) : String get(int columnIndex) : String set(String columnName, String columnValue) : void set(int columnIndex, String columnValue) : void

搜尋服務元件共管理請求者和回應者兩種元件，這點和集結服務元件的架構相當類似，因此在本段中不贅述，以下為搜尋服務元件在執行期間的順序圖：

請求者之執行順序圖：



回應者之執行順序圖：



4.2.5.6. 其他服務元件(Other Services)

為了應用程式所需之功能額外擴充之用，本協定架構預留服務元件與端點提供者之間溝通的呼叫介面，程式發展者便可藉由繼承此服務元件將特定服務加至端點提供者中，以擴大服務支援的範圍。而此呼叫介面在本節開始便已經提到，所以在此僅對該呼叫介面做簡略的介紹，下圖即為服務元件所定義的呼叫介面(圖中以粗體表示的文字便是端點提供者與服務元件進行溝通的呼叫方法)：

Service
getName() : String setName(String name) : void getProvider() : PeerProvider addServiceListener(ServiceListener serviceListener) : void removeServiceListener(ServiceListener serviceListener) : void **processRequest(Request request) : void** **processResponse(Response response) : void**

4.3 模組管理系統

本專題應用程式層是以模組的機制依照其支援功能之服務元件做切割，讓整體應用程式在管理上更為簡便，在程式實作期間，也能達到分工合作的目的，而主程式則提供給模組其本身的基本功能，如主選單項目(Menubar Item)、工具列按鈕(Toolbar Button)或跳出選單項目(Popup Menu Item)的加入，另外，主程式也提供模組的掛載(load)與卸載(unload)動作，達到模組管理的便利性及彈性。

因為每個模組中皆定義其本身所擁有的視覺元件、服務元件及其他模組資源，所以必須利用特定方式將這些資源完整描述，而本專題所使用的方式是採用類似 Java Applet、Java Web Start、WAR、EAR 或類似格式的程式檔案包裝，以下為一個模組目錄架構及其中必須包含的描述檔清單：

- /classes
  這個目錄裡存放的是模組中所有的程式類別檔案，模組載入器在讀取這個模組目錄或檔案會先將這個目錄設到類別路徑(Class Path)中以進行類別的載入。

- /meta-inf
  這個目錄裡存放的是描述模組資源的描述檔(Module Description File)，這個檔案的功能在於記錄此模組中定義的各類資源；若模組開發者欲新增自有的設定檔則可以將這些檔案存放在此目錄中。

- /meta-inf/module.xml
  這個檔案是整個模組最為重要的模組描述檔，其內容主要包含模組類別路徑、名稱、描述說明、作者、主要版本編號、次要版本編號、最後修改日期、語言設定及模組資源；另外，這個檔案定義與主程式間元件新增或刪除的動作，如主選單、工具列按鈕或跳出選單等。

- /meta-inf/lang/lang-*.xml
  這個目錄中裡存放的檔案為模組之視覺元件所呈現的文字語系，若使用者欲進行語言的變更，則僅需要重新讀取指定文字對應檔的內部，之後對程式進行重新載入，例如英文語系的檔案名稱為 lang-English.xml、繁體中文為 lang-TraditionalChinese.xml、簡體中文為 lang-SimplifiedChinese.xml。

下圖為(1)模組描述檔與(2)文字對應檔的範例：

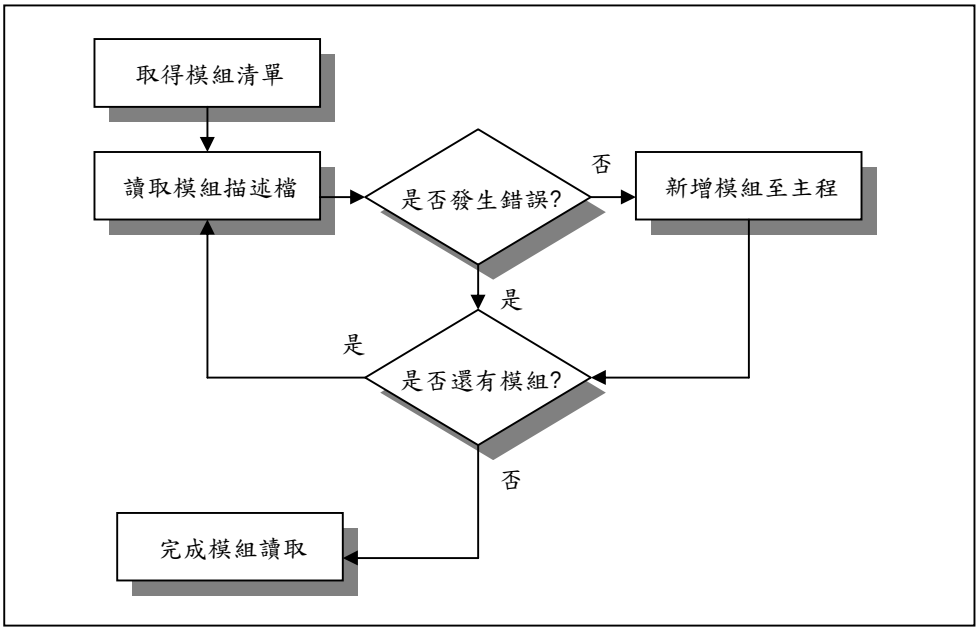
模組描述檔範例

```
<?xml version="1.0" encoding="utf-8"?>

<module className="jemi.app.client.module.mail.MailModule">

    <name>MailModule</name>
    <description>This is a module for MailService</description>
    <author>JEMI Development Team</author>
    <version>1.0</version>
    <language>TraditionalChinese</language>
    <modified>2003/4/7 20:00:00 PST</modified>

    <menubar>
        <menu name="Contact" mnemonic="C">
            <menu-item name="MailBox" mnemonic="H"/>
        </menu>
    </menubar>

    <toolbar>
        <toolbar-button/>
    </toolbar>

    <popupmenu>
        <menu-item name="MailService" mnemonic="M" trigger="login">
            <menu-item name="AudioMail" mnemonic="M" />
            <menu-item name="VideoMail" mnemonic="M" />
                    <menu-item name="BinaryMail" mnemonic="M"/>
            <menu-item name="TextMail" mnemonic="M"/>
        </menu-item>
    </popupmenu>

</module>
```

文字對應檔範例

```
<?xml version="1.0" encoding="utf-8"?>

<language>
        <name>TraditionalChinese</name>
        <description>Language Definition for Traditional-Chinese</description>
        <text name="jemi.app.client.main.Menu.MailService" value="多媒體留言(M)"/>
        <text name="jemi.app.client.main.MenuItem.MailService.AudioMail" value="語音"/>
        <text name="jemi.app.client.main.MenuItem.MailService.VideoMail" value="影像"/>
        <text name="jemi.app.client.main.MenuItem.MailService.BinaryMail" value="圖片"/>
        <text name="jemi.app.client.main.MenuItem.MailService.TextMail" value="文字"/>
        <text name="jemi.app.client.main.MenuItem.Contact.MailBox" value="收件匣"/>
        <text name="jemi.app.client.main.MenuItem.Add" value="新增"/>
        <text name="jemi.app.client.main.MenuItem.Delete" value="刪除"/>
        <text name="jemi.app.client.main.MenuItem.Forward" value="轉送"/>
        <text name="jemi.app.client.main.MenuItem.Reply" value="回覆"/>
        <text name="jemi.app.client.main.MenuItem.Play" value="播放"/>
</language>
```

主程式開始進行初始化期間會讀取模組目錄中的子目錄或是副檔案為.jar 的包裝檔並進行每個模組的掛載，若讀取模組描述檔發生格式錯誤或其他未知情況時，目前處理的模組會發出例外事件告知錯誤的發生及其說明，下圖為模組載入器在執行期間的流程圖。



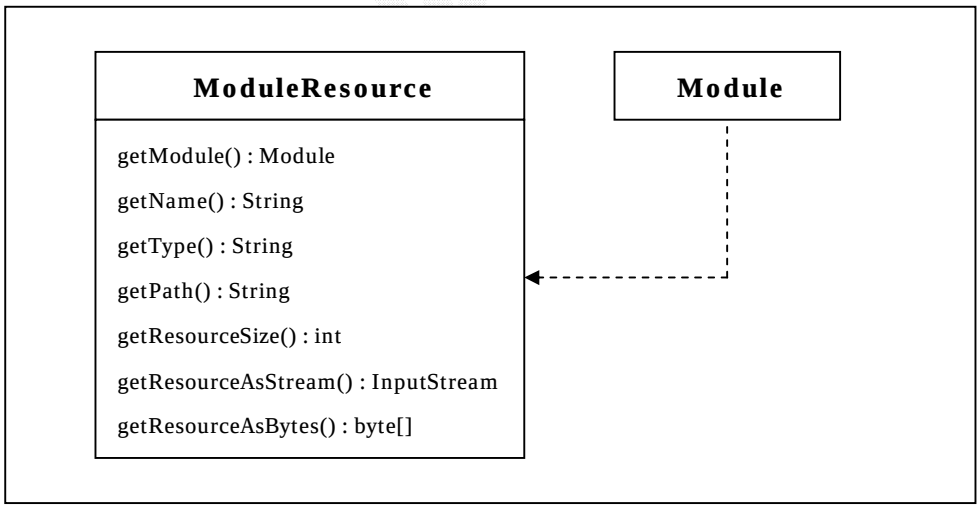
模組除了支援對主程式視覺元件的建立與摧毀，另外，對於其他類型的檔案皆以模組資源(Module Resource)表示，模組發展者可以利用此物件將特定設定檔或視覺元件所須的圖示檔進行讀取，以避免模組在無法確定特定檔案之正確路徑的問題。模組資源主要定義了三種資訊：名稱(name)、類型(type)和相對路徑(path)，其中名稱主要用來做為取得特定模組資源物件的鍵值；類型在目前版本中並未具有任何意義，在此僅做為模組發展者對特定資源的分類，在往後版本的模組載入器預計會支援類型的判別並且對之建立相對類型的模組資源存取介面；相對路徑指出特定資源相對於模組目錄之路徑。

模組發展者在讀取模組資源時可利用兩種方式進行：

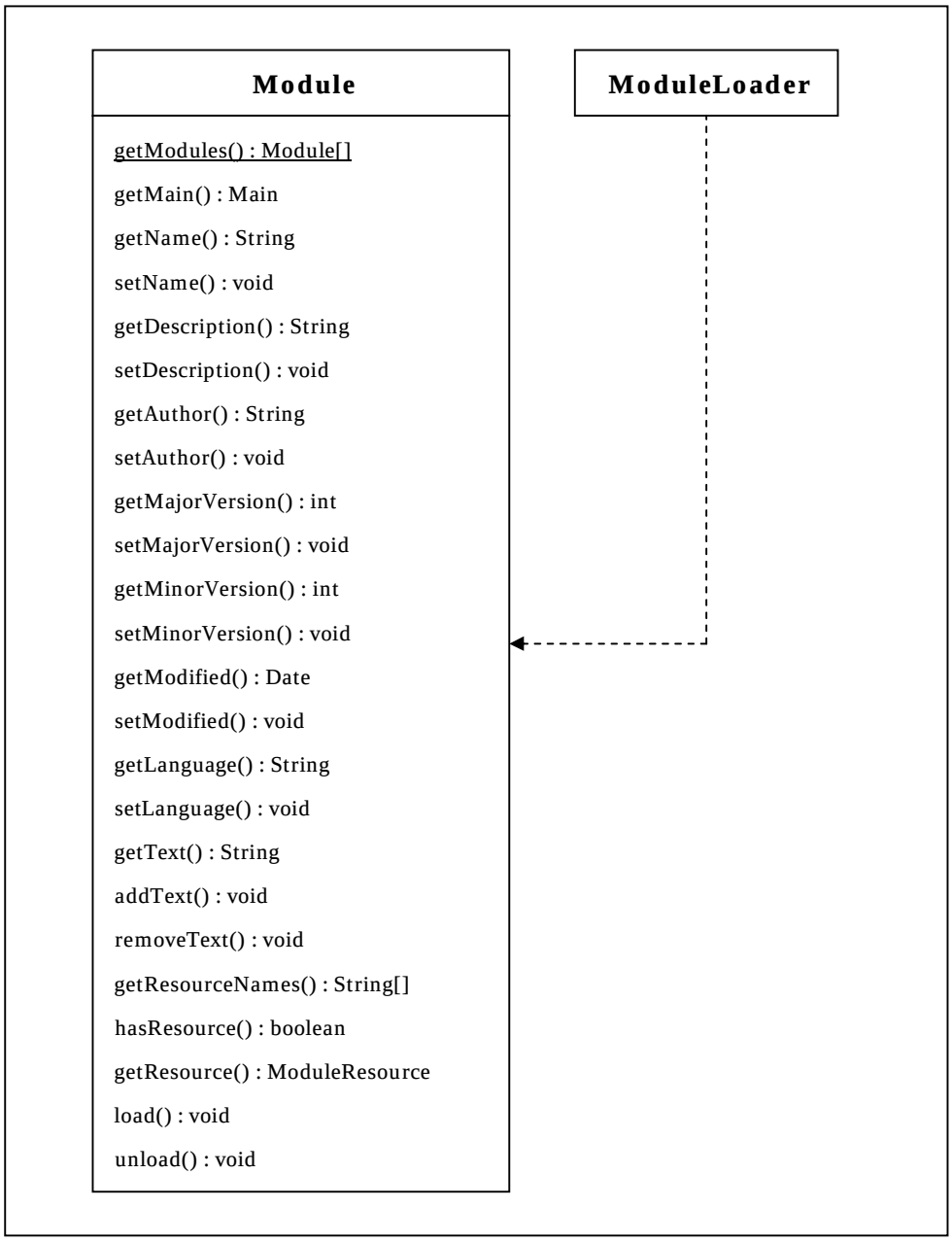
- 直接取得該資源內容的位元組陣列。
  每個模組資源在建立的時候皆會將其內容讀取並存放在位元組陣列中，因此該方式回傳的資料正是其原始內容。

- 先取得該模組資源的輸入串流再進行循序的讀取。
  這個取得方式僅是將原先的位元組陣列包裝成輸入串流，以提供使用者進行循序的讀取，而此法也可以簡化模組資源的讀取方式。

下圖為模組資源的 UML 架構圖：



下圖為模組物件之 UML 示意圖：



4.4 視訊會議系統

4.4.1運作規則

1. 會議參與者權力分為主席(chairman)與一般(normal)
2. 擁有主席權限的端點為
   I. 建立該會議者
   II. 可使用投票系統
   III. 驅逐參與者
   IV. 主席關閉會議，所有人一並離開會議。
3. 產生視訊會議的方式有：邀請使用者及加入會議，當端點無參與視訊會議並邀請其它端點建立會議時，即成為主席。
4. 當某端點已身在某一會議，即無法加入其它端點的會議，或被邀請，也就是說，每一端點僅能建立一個會議。
5. 當會議僅剩一人時，會議即結束。

4.4.2 元件概觀

元件名稱	主司
ConferenceModule	使用者透過此，與Conference及ConferenceWindow互動，也負責告知Main 如何關於會議的選單文字
ConferenceWindow	視訊會議的使用者界面
ConferenceService	負責視訊會議的流程，管理Conference元件
Conference	管理多個Participant元件及媒體元件
Participant	記錄其它參與者之會議相關資訊

表 4.4.2.1 元件概觀

ConferenceModule –

使用者按下『邀請使用者』或『加入會議』ConferenceModule即呼叫ConferenceService處理使用者的要求，並要為ConferenceService向主程式要求需要的媒體裝置元件。

ConferenceService –

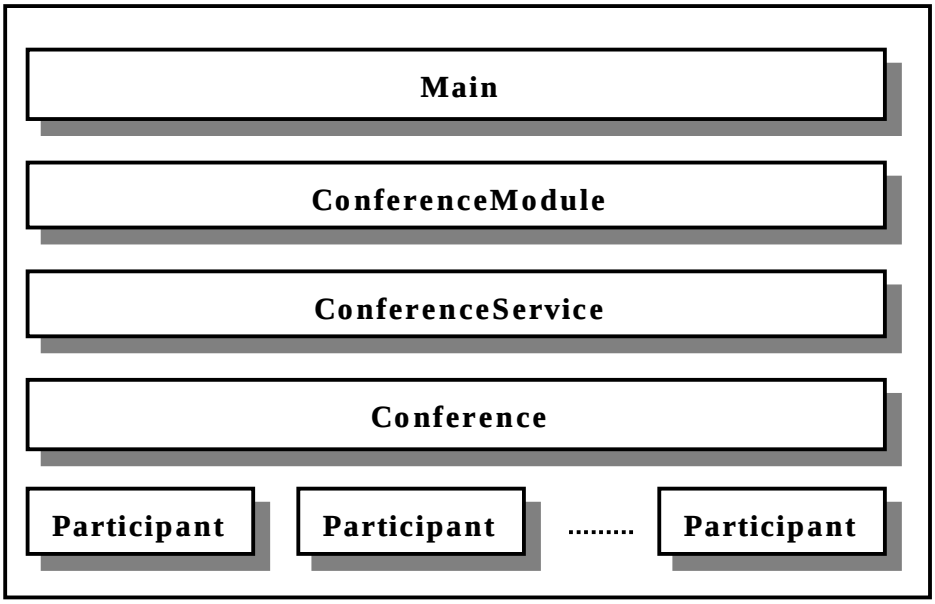
透過PeerProvider收發相關視訊會議的訊息，依視訊會議邏輯，呼叫Conference以執行流程。

Conference –

1. 記錄端點的會議流程狀態，以避免例外情況發生
2. 操作媒體裝置元件，產生視訊及聲訊
3. 記錄同一會議之參與者及彼此的關係

Participant –

記錄其它參與者的權限及端點會議與其它參與者間，媒體裝置的傳收關係。



4.4.3 系統狀態及流程

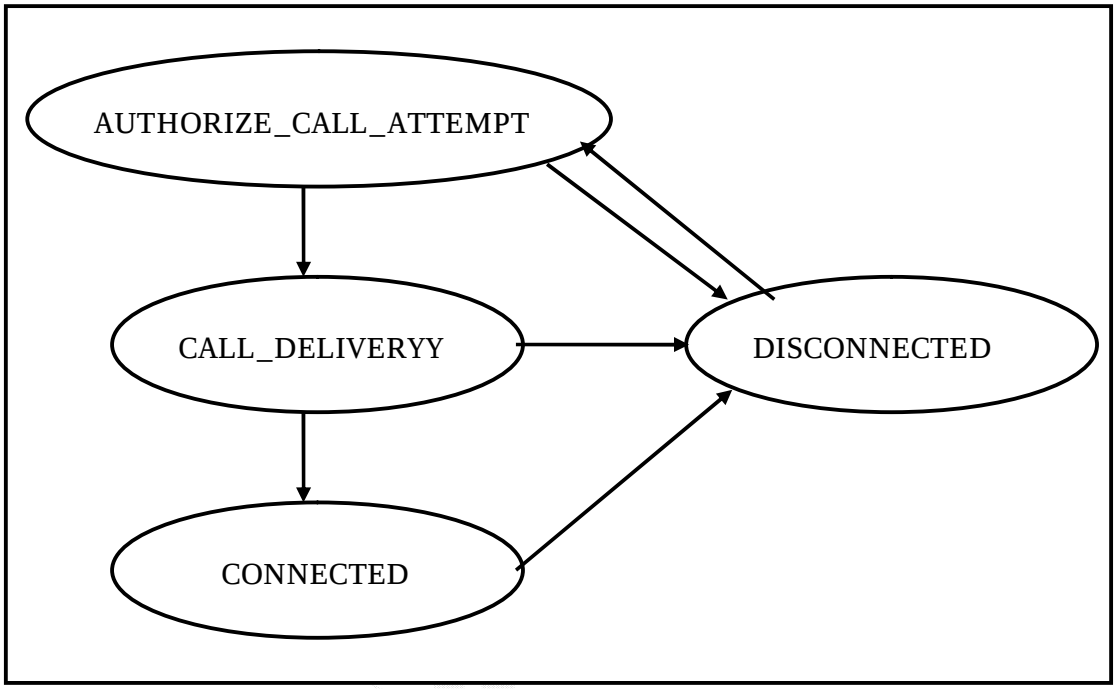


圖4.4.3.1 系統狀態流程圖

使用狀態記錄目前會議的邏輯流程，以避免進入無法預測的狀態，例如：使用者在加入某會議的同時，又同意了另一使用者的邀請，當狀態處於AUTHORIZE CALL ATTEMP或CALL DELIVERY時，便無法加入其它人的會議或邀請他人加入會議。

4.4.4 以邀請建立會議的各種狀態流程

實際上，邀請也可用在使用者已建立會議，被邀請者將會進入邀請者已存在的會議，並不會另建一個會議，當邀請動作下達後，正常可能發生情況有。

1. 對方接受
2. 對方拒絕
3. 對方忙碌中(可能正與他人建立會議)

4.4.4.1 非主席參與者邀請第三者(接受)進入會議

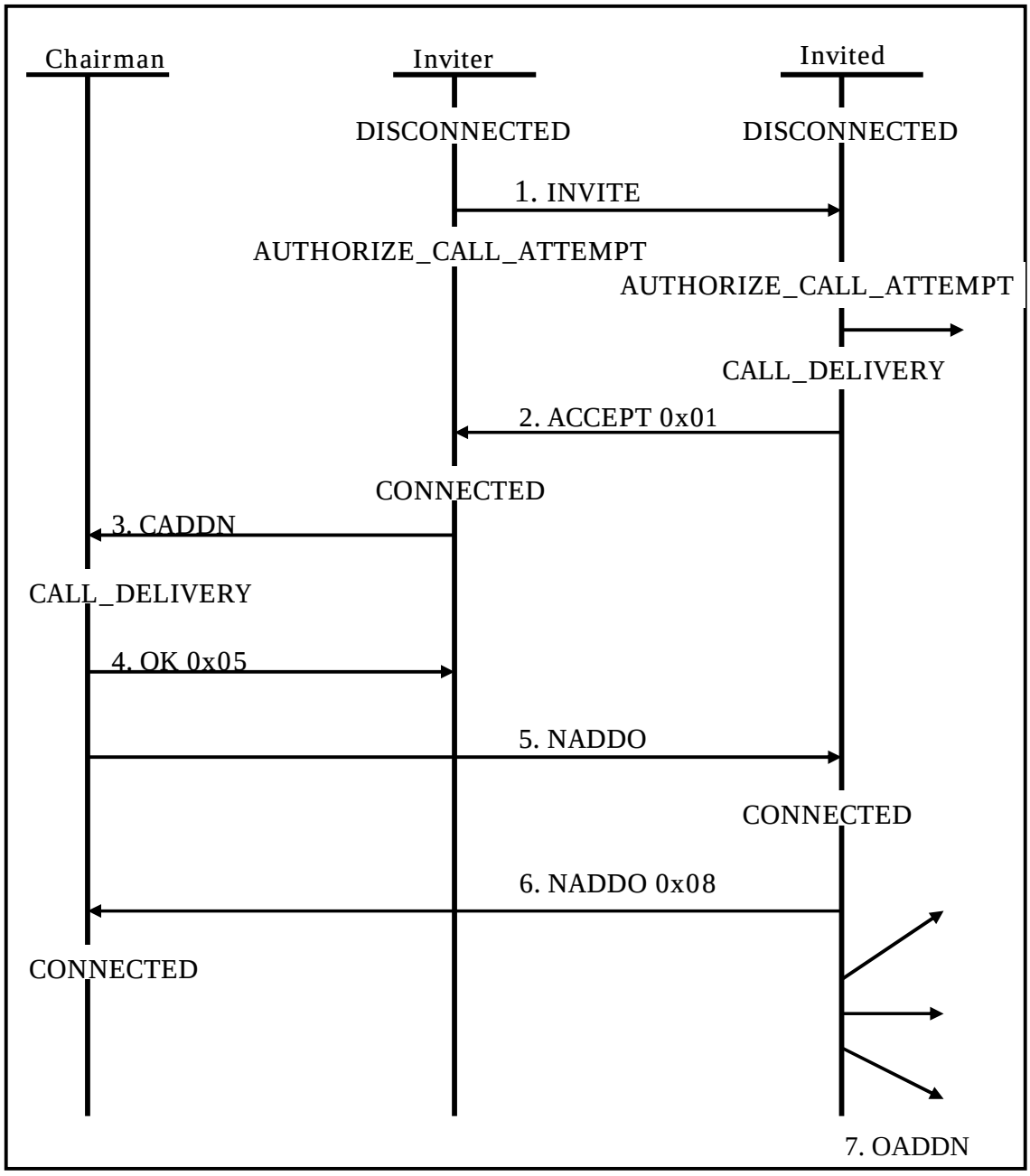


圖 4.4.4.1 非主席參與者邀請第三者(接受)進入會議

1. 邀請方在INVITE Request中，指出會議相關訊息，包括會議編號、主席端點位地、會議主題、是否有視訊聲訊會議，邀請方的起始狀態可以是DISCONNECTED或CONNECTED。
2. 被邀請方接受邀請送回ACCEPT Response前，將主席加入己方的會議、並與主席建立訊息管線，若建立視訊聲訊,則建立視訊聲訊網路接收裝置以等候主席端傳送媒體串流。
3. 邀請方收到ACCEPT Response後，將Response內參數加入CADDN Requeset，並對主席端發出請求。
4. 主席端先回應邀請方，再與被邀請方建立相關媒體連線，並以NADDO Request告知被邀請方目前會議的成員。
5. 被邀請方
   a. 加入各會議成員
   b. 與各成員建立訊息管線
   c. 以OADDN Request告知各成員，自己已經加入

4.4.4.2 主席邀請第三者(接受)進入會議

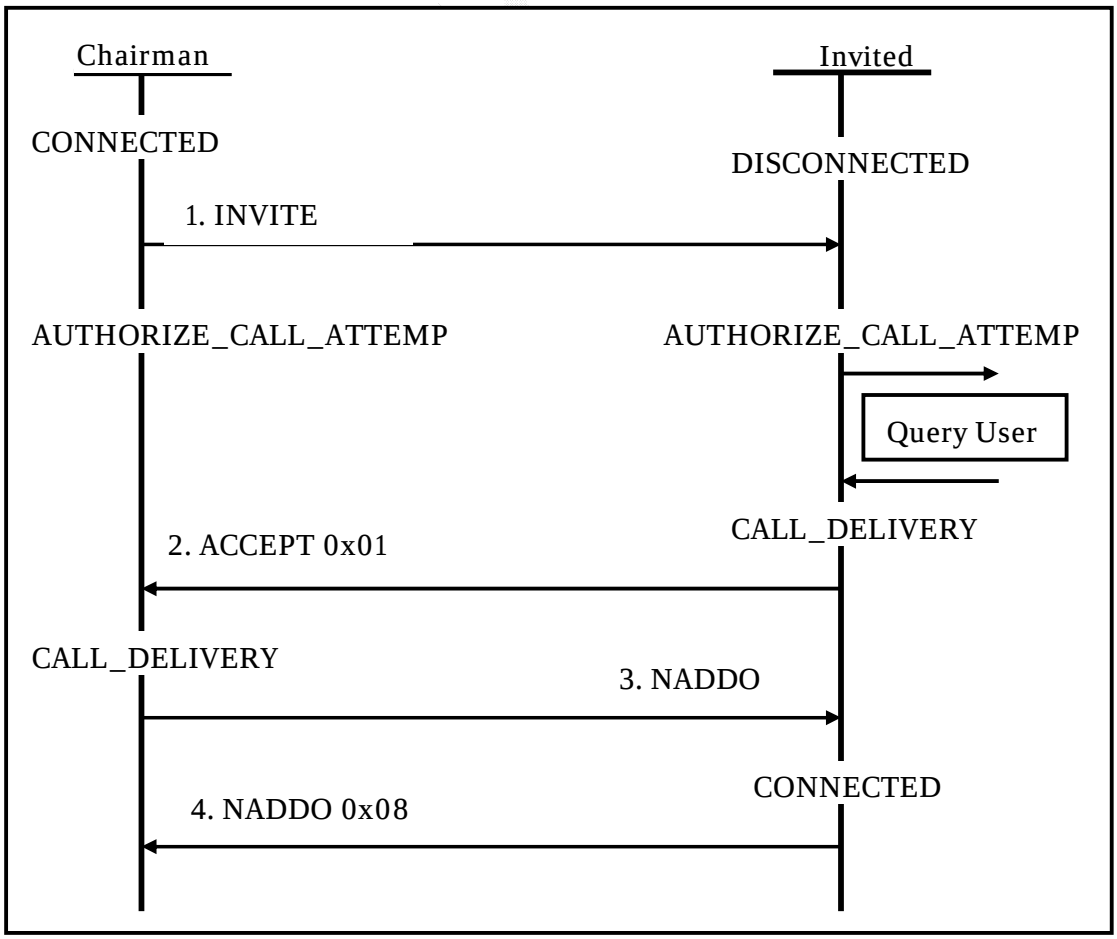


圖 4.4.4.2.1 主席邀請第三者(接受)進入會議

由於邀請人為主席，因此不用另行通知，其流程與非主席參與者邀請第三者(接受)進入會議類似且更為精簡，接下來是邀請流程的UML順序圖。

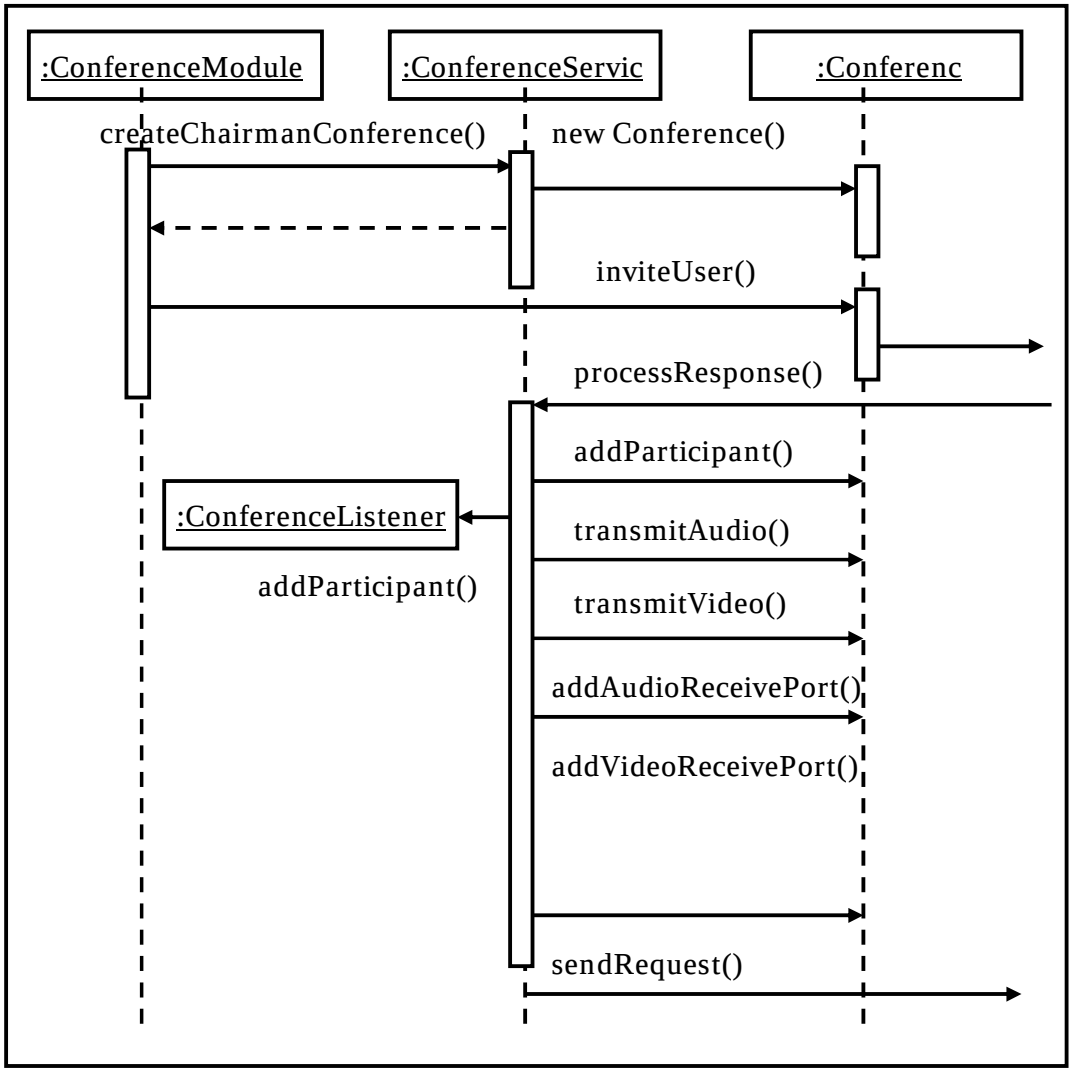


圖 4.4.4.2.1 邀請流程的UML順序圖

由於版面限制本圖最右方的是PeerProvider物件未畫出，本圖所描述情境是邀請方並未建立會議，因此一開始呼叫ConfereService的createChairmanConference()，如果邀請方已處於會議中，則直接呼叫Conference的inviteUsesr()，接下來介紹的是非主席參與者邀請被拒的狀態流程。

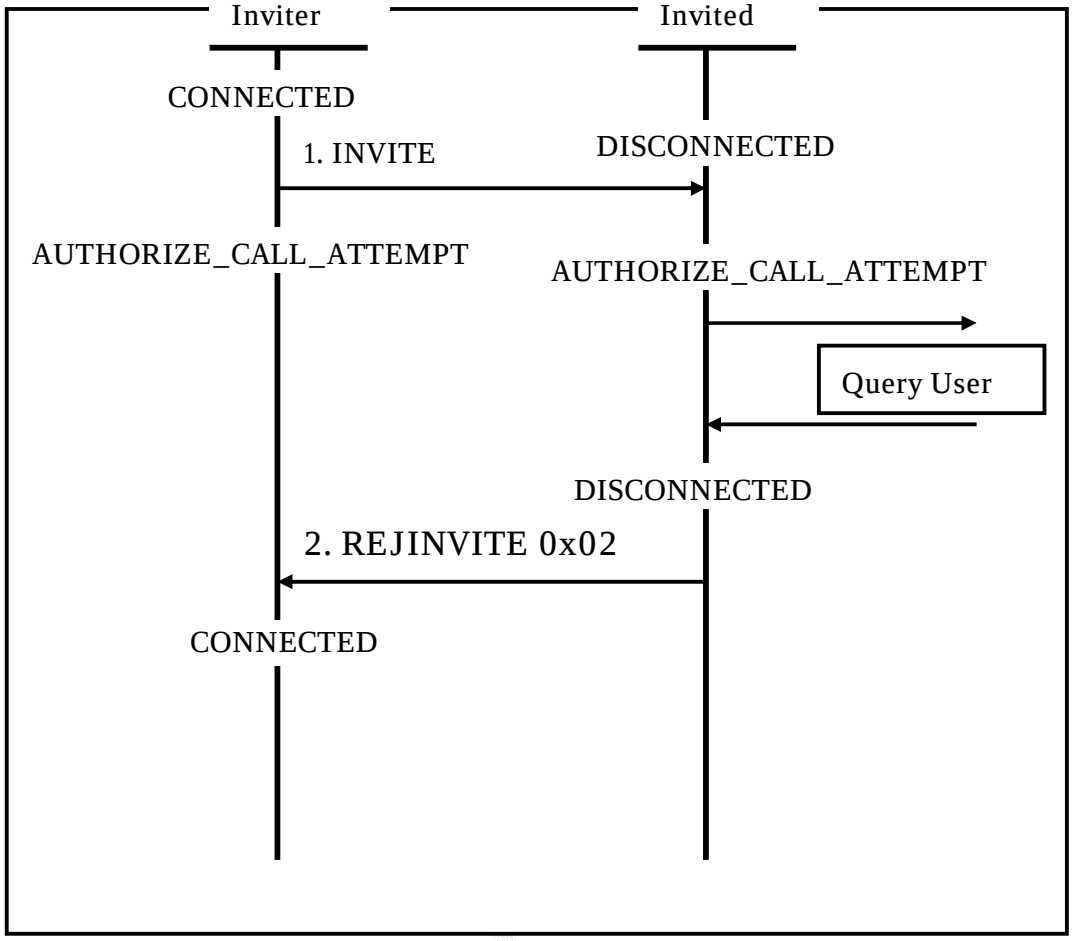


圖 4.4.4.2.2 非主席參與者邀請被拒狀態流程圖

由圖可知，參與者仍可保持送出邀請前及收到邀請前的狀態，使會議可繼續進行，這是視訊會議系統非常重要的特性，就是狀態的安全性，我們必須避免狀態超出掌控的情況，無論是受到使用者無意的不正常使用，或網路及執行環境的影響，這些影響將於後述。

4.4.5 以加入動作進入會議

加入動作只有在並沒有建立本地會議的情形下，才能動作，若已建立會議則無法使用此動作，加入動作下達後，可能發生的情況。

1. 本地會議已建立，動作中止
2. 對方同意
3. 對方拒絕
4. 對方忙碌中(可能正與他人建立會議)
5. .對方未建立會議

4.4.5.1 以非主席參與者為對象(接受)加入會議

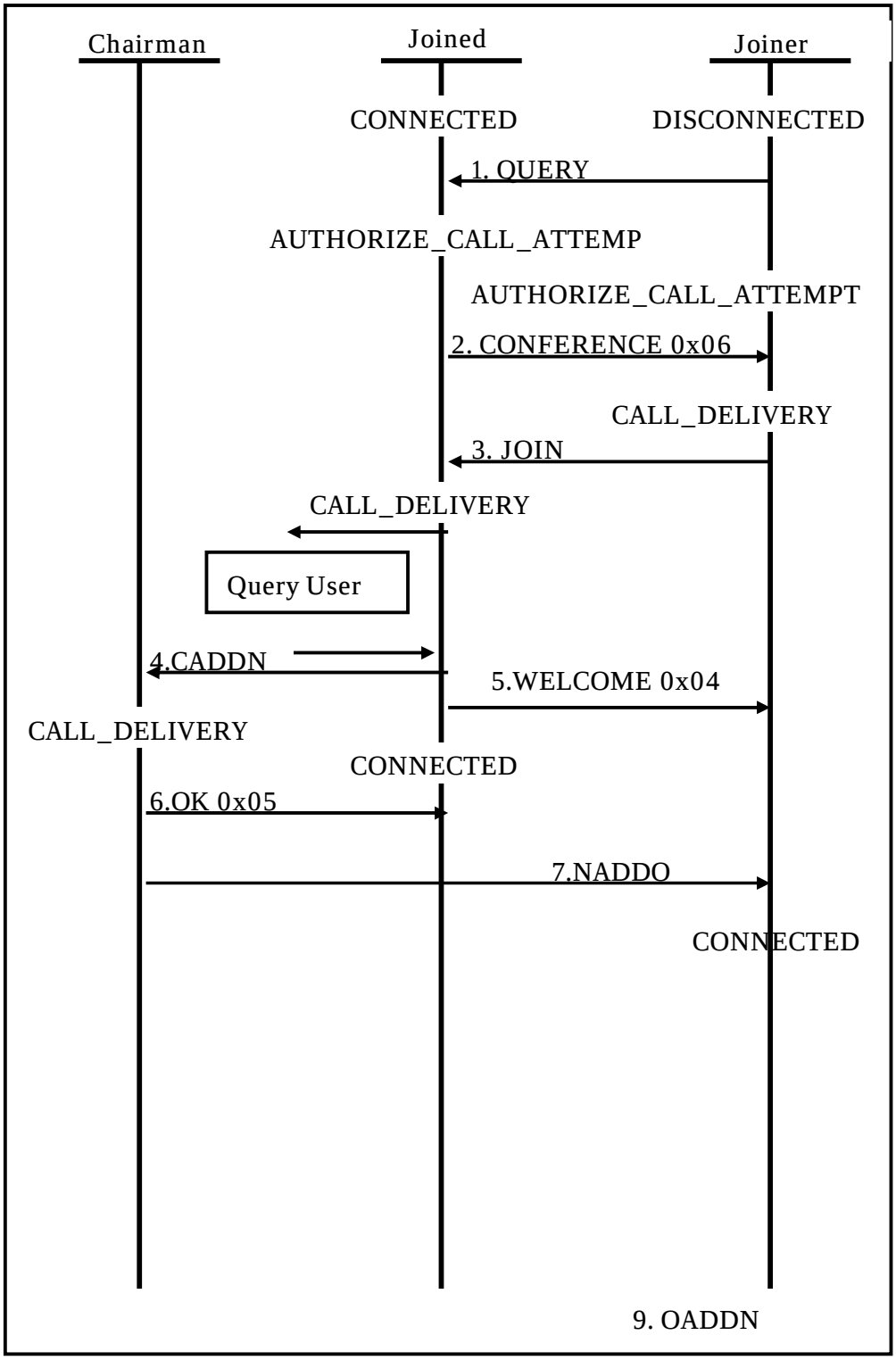


圖 4.4.5.1.1 以非主席參與者為對象(接受)加入會議

1. 首先查詢對方的會議狀況：
   - 對方會議狀態為DISCONNECTED則回應「對方為建立會議」
   - 對方會議狀態不為CONNECTED且不DISCONNECTED則回應「對方忙碌中」
   - 對方會議狀態為CONNECTED則進入下一步

2. 加入方送出加入請求，被加入方會議詢問使用者是否同意此加入請求。

3. 被加入方同意後將回應WELCOME Response並送一份CADDN Request至主席端。

4. 主席端先回應邀請方，再與被邀請方建立相關媒體連線，並以NADDO Request告知被邀請方目前會議的成員。

5. 被邀請方
   - 加入各會議成員
   - 與各成員建立訊息管線
   - 以OADDN Request告知各成員，自己已經加入

其中步驟4、5與非主席參與者邀請第三者(接受)進入會議相同，就是為了保持會議參與者列表的一致性及整體會議流程的一致性

4.4.5.2 以非主席參與者為對象(拒絕)加入會議。

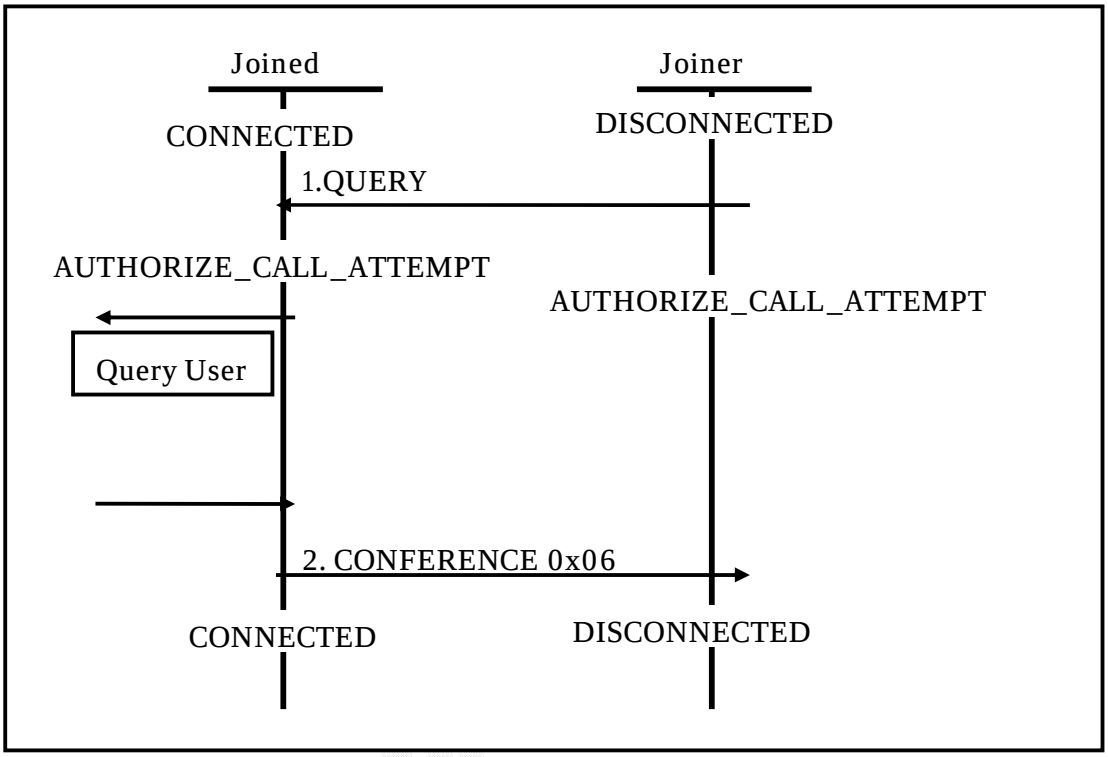


圖 4.4.5.2.1 加入被拒狀態流程圖

雖然同樣是回應CONFERENCE Response但與對方同意加入請求不同的是Response中，包含一參數FLAG值為N，同意則為Y，若參數中無SESSION則表示對方沒有建立會議。

4.4.5.3 加入會議的UML順序圖

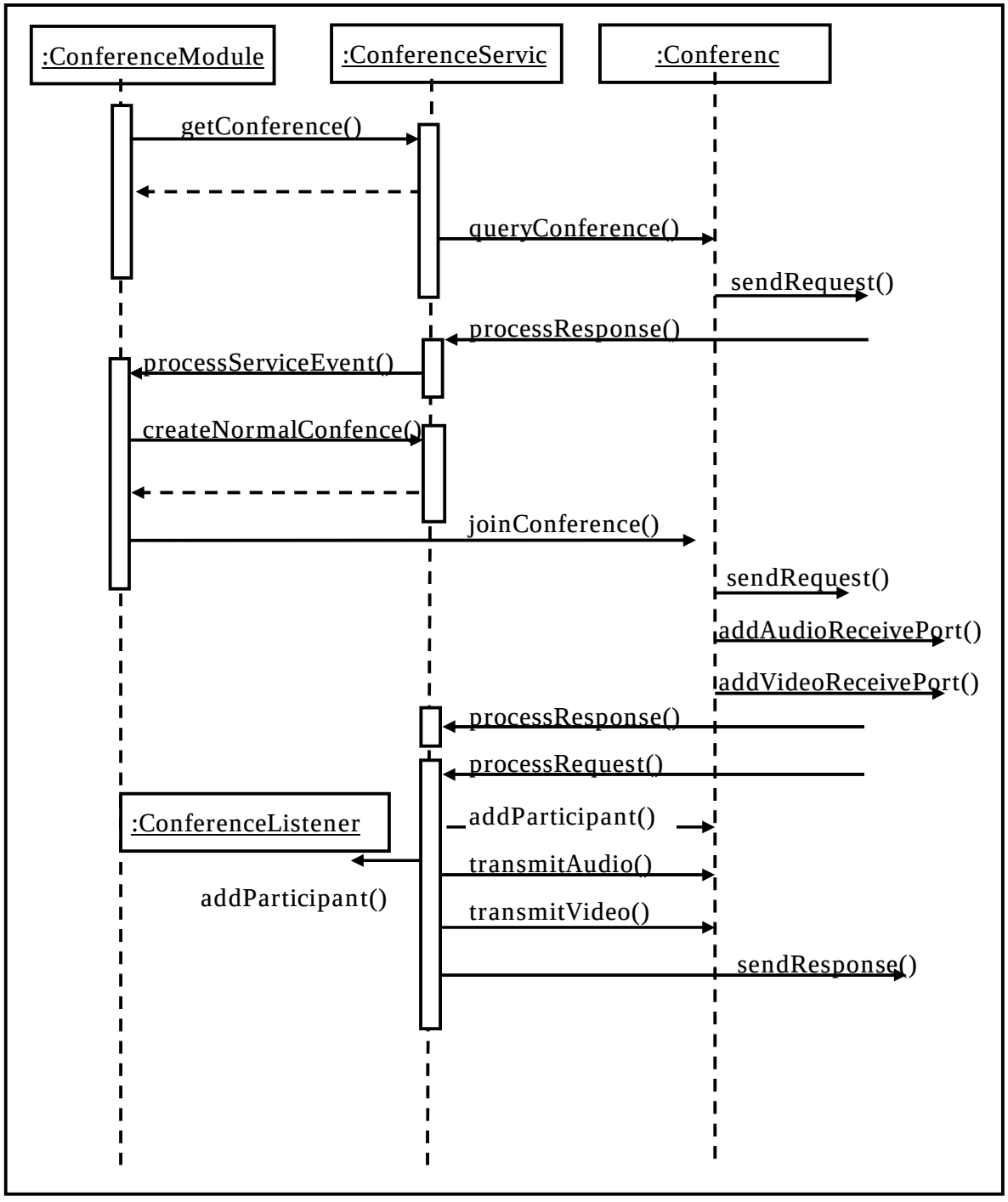


圖 4.4.5.3.1 加入會議的UML順序圖

上圖的觀點在於加入者本地會議系統的呼叫流程，一開始從ConferenceService取Conference物件，狀態DISCONNECTED，之後利用該物件查詢被加入者的會議狀態，一旦對方同意後再重新產生Conference物件，之後便利用此物件建立會議。

4.4.6 正常離開視訊會議

某會議參與者離開會議後，將關閉以下項目：

1. 視訊會議使用者界面
2. 所開啟視訊畫面
3. 對所有人的訊息管線
4. 對主席的聲視訊資料傳輸
5. 聲視訊的接收物件

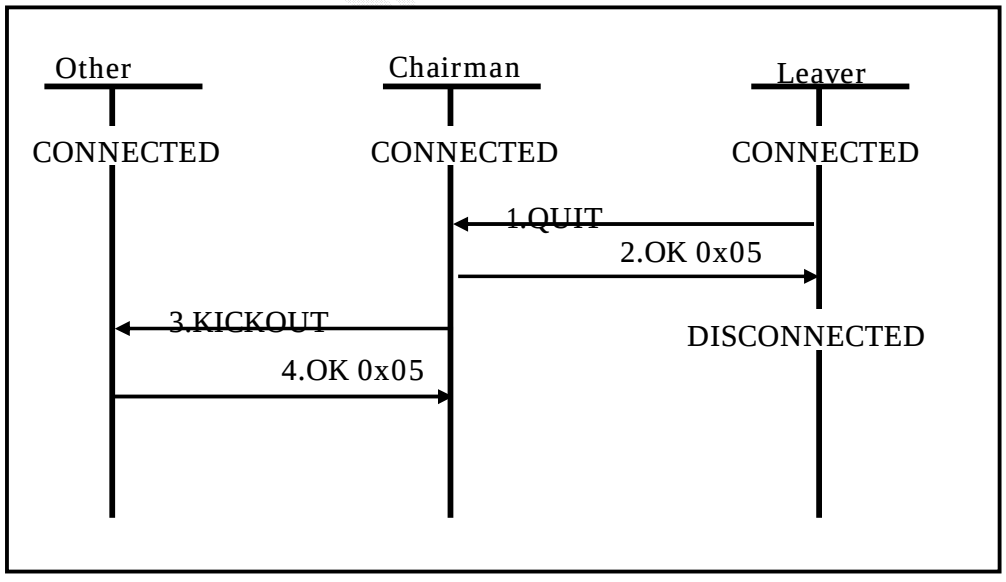
主席端收到參與者之離開請求後，將關閉以下項目：

1. 對該參與者的聲視訊資料傳輸
2. 對該參與者聲視訊的接收物件
3. 對該參與者的訊息管線
4. 告知其它參與者

其它參與者收到主席告知後，將關閉以下項目：

1. 對該參者的訊息管線

注意如果是主席參與者離開會議，則整個會議將關閉，也就是說其它參與者也將關閉一切，如果會議只剩一參與者時，也將關閉會議，換句話說，不存在一個人的會議，而在此制度下，該剩下的一個參與者其實就是主席。



同樣的，為保持一致性，參與者的離開請求需送至主席，再由主席轉告，主席轉由KICKOUT Request告知，KICKOUT Request本是用在強制驅離視訊會議的使用者，不過也可用於此，下圖為主席方收到離開請求的UML順序圖。

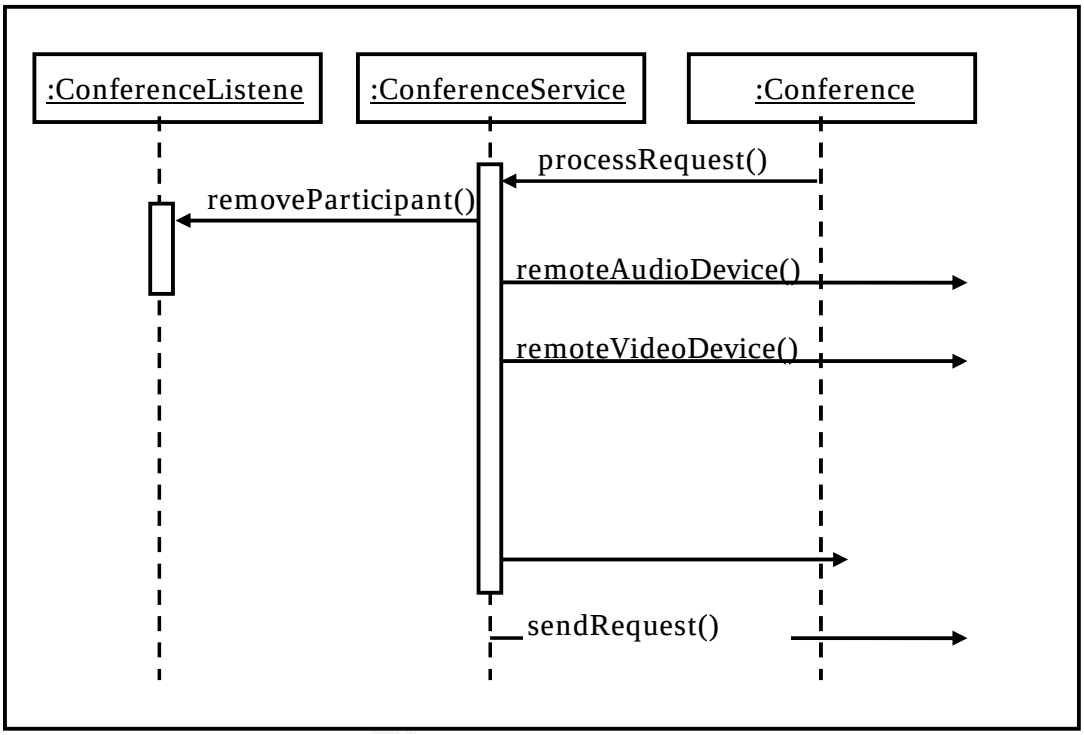


圖 4.4.6.2 主席方收到離開請求的UML順序圖

4.4.7 極端及例外狀況

極端狀況主要是發生在進入會議及離開會議時，例舉狀況如下：

(1) 非主席參與者邀請第三者進入會議時，非主席參與者送CADDN Request至主席，此時主席可能正處理其它人的CADDN Request或本身的邀請流程，如果不予理會而處理此請求，則可能造成整體會議中，有兩位參與者看不到彼此，如果回應忙碌則造成使用上的不便。

解決方法：使用一訊息佇列暫存請求，待主席處理其它事務完畢後再從訊息佇列取出並處理。

(2) 非主席參與者送離開請求至主席，此時主席可能正處理其它人的CADDN Request或本身的邀請流程，如果新進入之會議參與者已收到會議參與者名單並送出OADDN Request則離開者也會收到請求而不予理會，導至新參與者出現TimeOut，同時多媒體裝置的開啟及關閉也會出現不同步現象。

解決方法：同(1)

例外狀況則是因為使用者不正常使用或網路環境因素導至,可能發生的情況如下：

(1) 使用者不關閉視訊會議視窗而直接關閉主程式。
解決方法：由於主程式的關閉會告知JEMI-Server，JEMI-Server會轉告該使用者所有上線連絡人，ContactService將丟出ContactUserRemove事件，ConferenceService收到此事件後，直接將該參與者從本地參與者列表中移除。

(2)使用者網路中斷。
解決方法：無可避免的，其它會議參與者對該使用者的訊息管線將出現錯誤，某訊息管線出現錯誤，Conference將關閉該管線，並於視窗界面中告知無法傳送訊息至該使用者，但並不移除該參與者，直到JEMI-Server偵測不到該使用者而發出通知。

4.4.8 結論

本視訊會議的特點是decenterization，所有使用者的會議的管理並非集中至JEMI-Server而是由該會議中的某一參與者負責，也就是主席，這種作法的好處是避免集中管理的缺點，例如因集中管理程式或主機失誤而導至所有使用者無法建立視訊會議，也可避免集中管理主機負載過重而導至效能低落，因為具有這些優點所以視訊會議採用decenterization技術。

4.5 影音留言系統

4.5.1 多媒體影音留言

近年由於手機的普及，加上科技日益進步，過去我們只能以文字和語音的方式來留言，如今已大為改觀。在手機上已可以顯示多媒體訊息，包括了聲音和影像，因此各電信業者也加入了此項服務，也就是時下最流行得 Multimedia Mail Service(MMS)。而在手機上的即時影像和聲音，由於電信頻寬的限制(目前為止)，所以現在的 MMS 多以圖片加上聲音為主。而目前在 PC 上 P2P 軟體並沒有此項功能，加上電腦的網路頻寬已非常快速，所以在電腦上實作多媒體影音留言系統非常適合。

基於上述理由，我們因此規畫出一套多媒體影音留言系統在 P2P 架構下使用，使得每個客戶端可以快速地享受多媒體訊息所帶來的全新服務。

系統架構和流程:

因為是留言系統所以必需有伺服器來暫存目的地的 MMS 訊息，所以分為客戶端和伺服器。

Server 端每一使用者，都會有其所儲存的使用者留言資訊，其目錄結構如下:

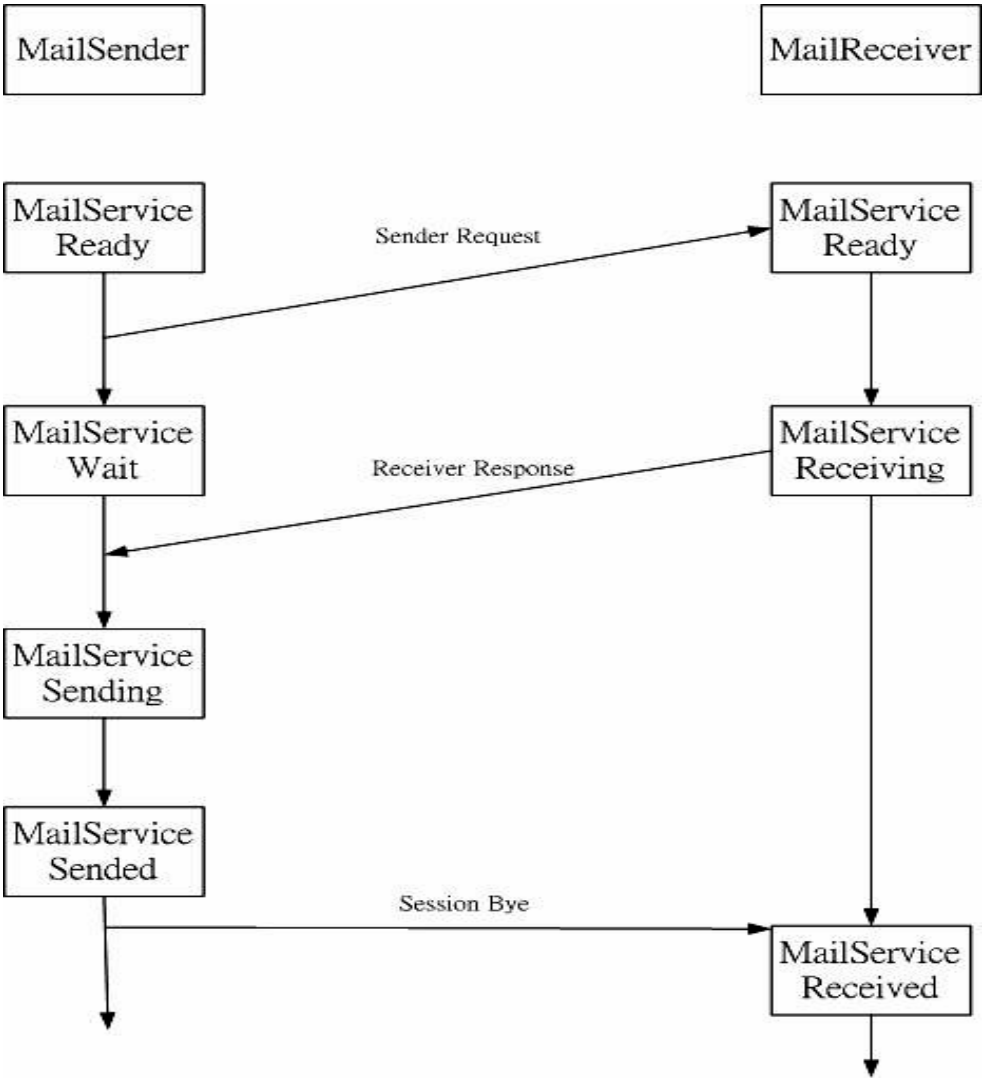
```
-Mail
    -UserName
            +Audio
            +Video
            +Binary
            +Text
            +Mix
            -mail.xml
```

在每個資料夾中，分別存放各種多媒體留言資料和相關資訊。

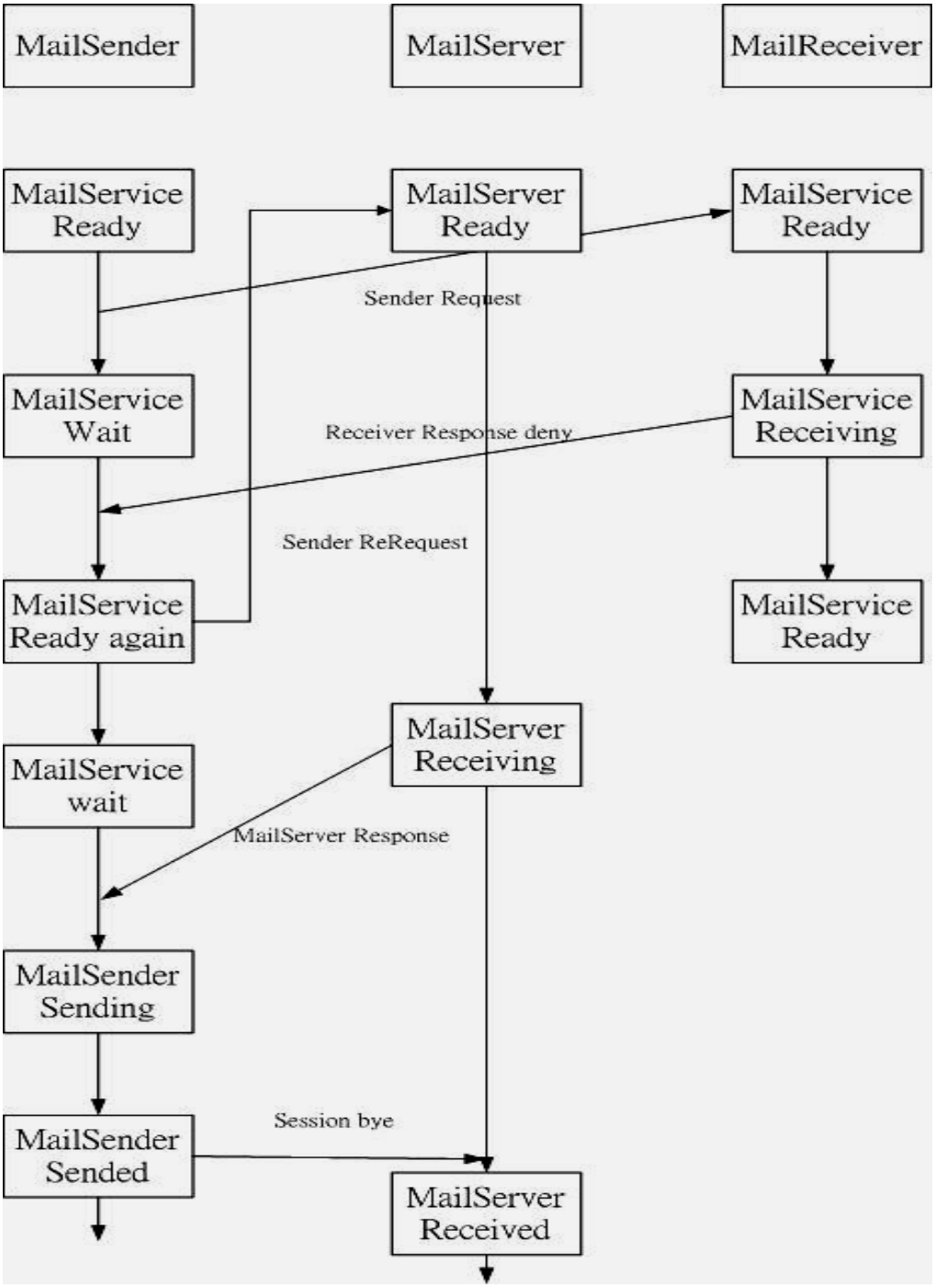
以上是基本的 mail 格式，我們可以分別設定各種我們想傳送格式，除此之外，也可以由使用者設定。

基本流程如下:

首先在來源端，必需先建立其所要想要的多媒體影音訊息類型，可以影像、聲音、圖片、文字和混合(可以是前面的各類型的結合)，當來源端決定好其所要傳送的多媒體影音訊息類型和目標後，來源端會先向伺服器的到接受端的位置相關資訊，直接向接受端送出其要求，當接受端接受其要求，會回傳其已接受和其所開的 port 等資訊，而來源端收到此回應，就會經由串流的方式，向接受端送出。接受端也會即時地將所收到的串流以對應格式輸出。



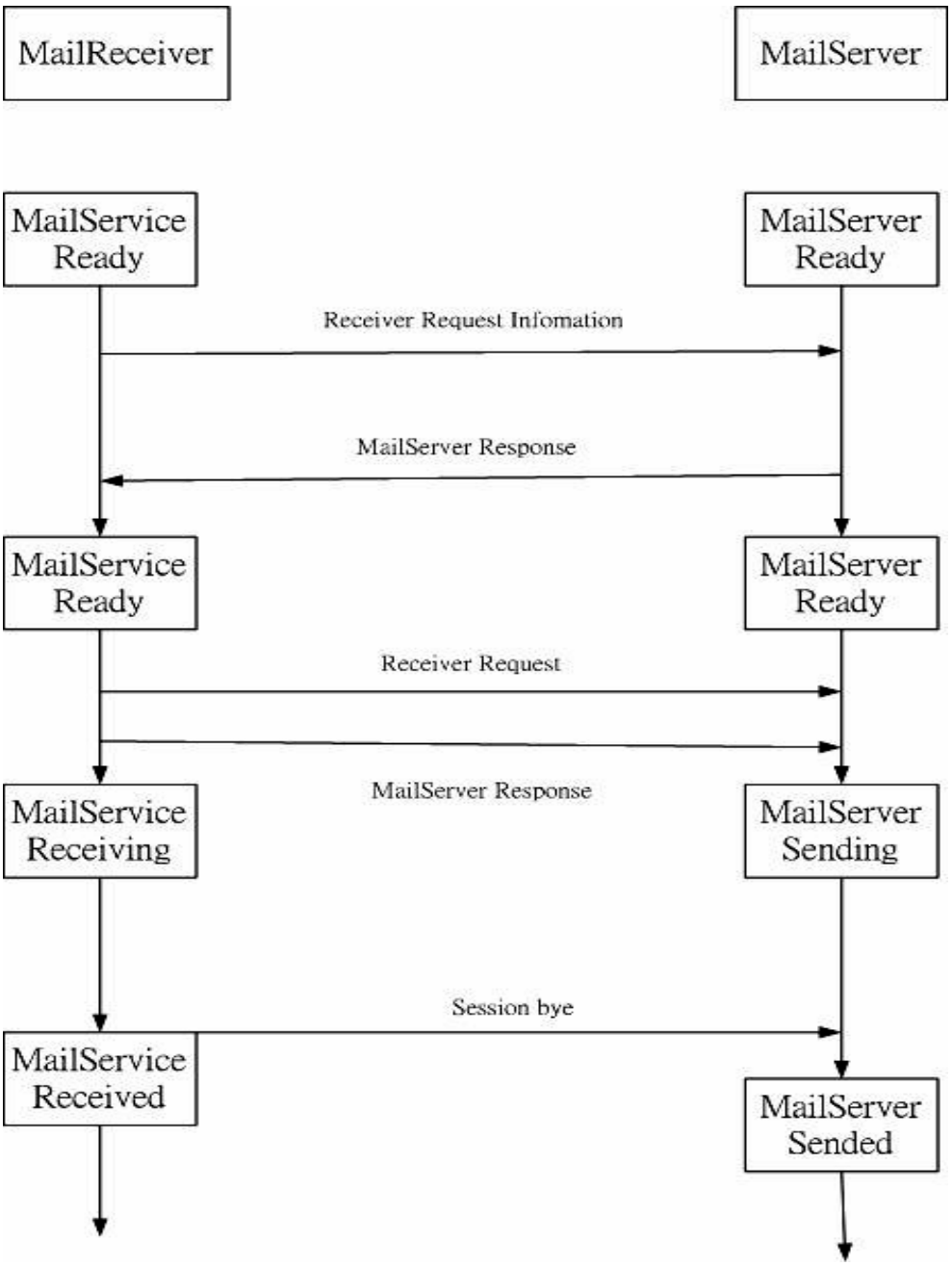
當目的端不在時，會回傳不存在的資訊，當來源端收到此訊息後，將向伺服器送出其要留言的要求，把留言先保留在伺服器端，待接受端上線後，再將其留言訊息送到目的端，而傳送的方式也是經由串流儲存在伺服器。



當使用者上線後，會先向伺服器端送出是否有其留言訊息，當伺服器有其留言時，會先回傳其所有的留言訊息到使用者端，使用者可選擇其所想聽的留言項目，當使用者選擇選定後，伺服器會回傳該留言，以串流直接到使用者端輸出。

4.5.2 留言訊息的保留及利用

當使用者收完在伺服器上留言後，可以選擇是否先保存在伺服器，本地端可以先不保留，所以使用者可以在不同地方重覆地接收其留言。當使用者要將其留言保留後，可以經由在本地端的 encoder 將其編碼為如 mp3、mpeg、mov、avi 等通用格式保存，以方便重覆收聽。也因為有保存其留言，所以使用者也可以將其留言自己送給其它使用者，以完成資訊共享的目的。

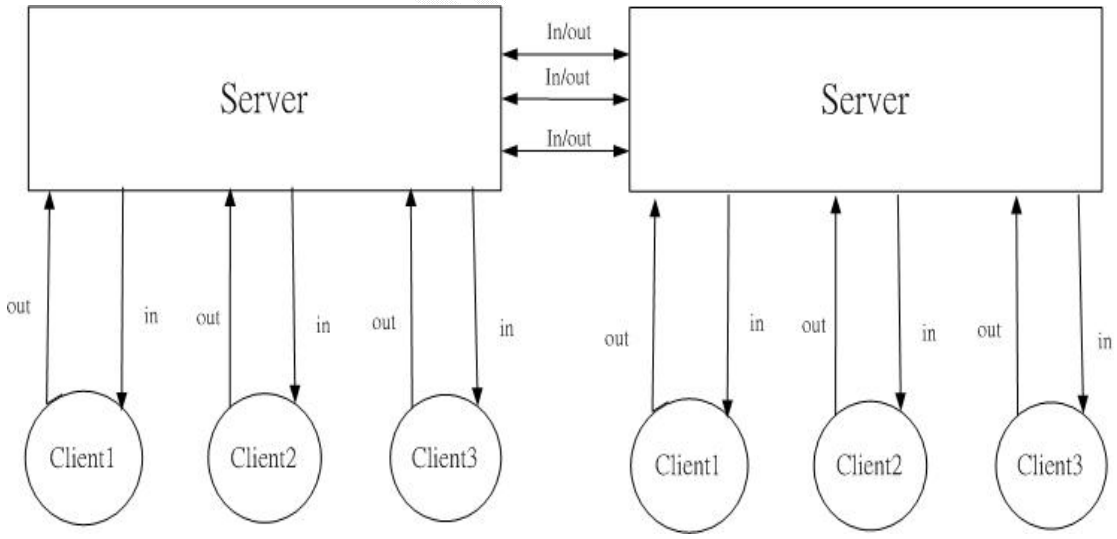


4.5.3 MailServer 之間的溝通

由於使用者每次所 Login 的 Server 可能不同，因此多媒體留言是使用分散式資料共享的方式來儲存，所以使用者端必需向 Server 詢問是否該使用者的 mail 訊息及所 mail 所儲存的多媒體留言所在 Server 位置.也因為如此，每個 MailServer 之間必需有良好溝通機智。在本系統中我們的 Server 是使用樹狀結構方式，每一 Server 在一段時間後會向 RootServer 詢問目前那些 Server 已啟動，及是否有使用者留言，如果有新的留言訊息，會將其訊息傳回 Server，等使用者得到其 mail 的資訊後，再向存有其真正存多媒體留言的 Server 接收其 mail 內容。

4.5.4 混音

在本計畫中的 Conference，也使用了混音的技術，來減少頻寬的使用。所謂的混音，就是將多個聲音來源，在一個集中的來作聲音的結合動作，將多個聲音來源，只以一個聲音頻寬的大小來輸出，大大地減少頻寬的使用。而混音最大的困難就是如何將多個來源快速地結合，以減少時間 Delay 的問題，所以在作混音的 Chairman，再作混音時，作每個來源必需有適合的緩衝區大小，以平衡各個來源不同頻寬的同步。除此之外，另一個問題就是當已混音的聲音頻寬，突然加入另一來源必需先對之前的來源暫存，並重新混音:，而且這些動作都是即時的，所以在設計上必需考慮其動作先後時序。



4.5.5 多人視訊流程會議簡化設計方法

首先必需先有人當視訊會議的的暫時伺服器(會議的建立者)，每個要加入會議的人，都先連線到此伺服器，如果有其它的可用暫時伺服器，也可以將所有加入會議的人可以平均分配到所有暫時伺服器，以簡化設計流程。這就是所謂的集中式管理方法。

4.5.6 多人視訊會議的多媒體特殊設計方法

本專題中為法達成多人視訊會議的目標,特別設計出一套可多人連線，自動轉接，自動分配封包的 Media Router，透過 Reorderable Multimedia Queue，來作多媒體串流的處理， 並且成功地以只用到二個 port 來傳輸所有人的影像及聲音，以下是本專題多媒體的架構

```
-jemi.media
    -device
            -Device
            -NetworkDevice
            -AudioDevice
            -VideoDevice
            -FileDevice
            -NetworkDevice
            -DeviceManager
    +player
    +recoder
    -rtp
        +Audio
        +Video
        +Adapter
        +moniter
        +file
    +util
```

套件名稱	功能
device	將每一種可能的來源與目的定義為一種 device，並且為其加上 Heart Processor 為其 heart，只要 heart 正常運作，就可以自動將 Stream 無限地分支傳輸。
player	為多媒體的呈現元件，並且可將所處理的 Stream 自動地轉傳為其 Output 給其它元件使用。
recoder	是可將 Stream 記錄至任何 Device 元件。
util	為 tool 元件，包含 device 的 detect，元件的安裝及註冊。
rtp	此為本專題中多媒體傳輸的主要套件，包含了 Audio、Video、 Adapter、moniter、file 這些子套件來處理各種多媒體傳輸。

以下是其主要功能元件功能說明:

1. DeviceManager:此為管理及 detect 各種裝置的主要物件，可由其來發現使用者使用環境上所有可用的裝置，如 Camera、Microphone、Audio Card、和所需的元件是否已經安裝，且可透過此物件來產生各種所需的 NetworkDevice，包含目的及來源的 NetworkDevice。

2. NetworkDevice:此為最處理底層 RTP 協定的主要元件，包含了 RTP Stream 的傳輸、混音、混影像、各種 RTP 事件的處理、來源串流的分析及分配、來源串流使用者的確認與比對，由於同一 NetworkDevice 可能處理多個到無限制來源的多媒體串流,所以其處理能力相當重要。

3. VideoTransmitter 及 AduioTransmitter:其分別為處理及管理 Video 與 Audio 使送目的主要物件，它接收由 NetworkDevice 來的各種事件及各個可用的多媒體串流，其中 VideoTransmitter 更可以和 AudioTransmitter 結合，可辨別出來自同一來源的串流來作組合，因為網路上的串流來的順序可能不一樣，其中 Audio 及 Video 都可能發生，所以處理這部份是本專題的重點之一。

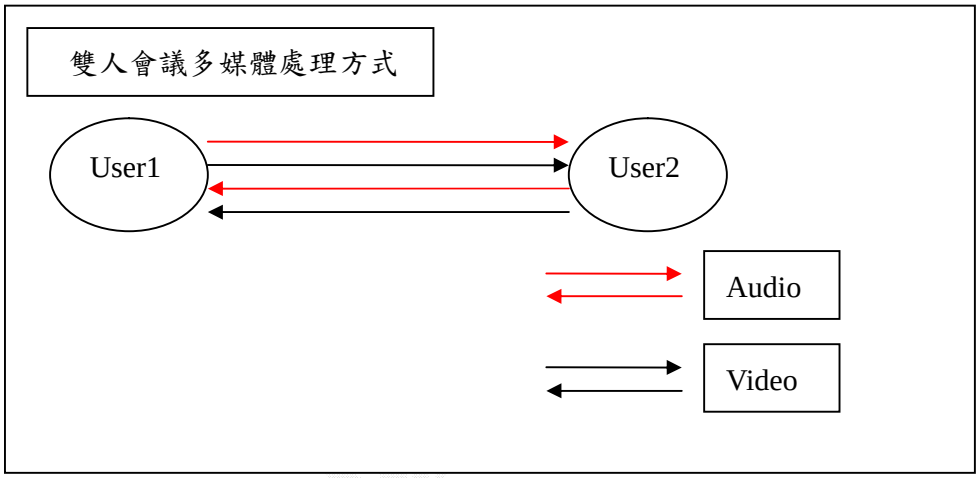
4. MediaRecoder:此為將各源的串流存為檔案的主物件，包含來自網路的 H.263、g.723、JPEG、mpeg 等 RTP 串流，和本地的 MicroPhone、Camera 等裝置的串流，編碼為各種檔案類型，如 mov、mp3、avi、mpeg 等常用的檔案，以方使用者存取。

5. VideoPlayer:其為呈現來自網路、本地的各種多媒體裝置，其可處理包含本地的各種聲音及影像檔案,另外它也可以處理來自網路上的各種 RTP 串流呈現給使用者。

4.5.7 傳輸架構

4.5.7.1 使用中的傳輸架構

本專題中是做用所謂的 Client-Server 架構，但 Client 與 Server 之間，和 Server 與 Server 卻都是以 Peer-to-Peer 的方式來模擬，當 Client 和 Server 之間建立一次 Video 及 Audio 的連線後，以後再有使用者加入到 Server 後，不必再建立另外的連線，就可以從原來已建立的連線，得到不同使用者的 Video 及 Audio 串流，以下是傳統會議的建立方式。



在傳統的架構中如果有使用者加入會議，都必需和使用者之間必需使用至少 4 個 udp port 來轉輸聲音和影像，所以當使用者和使用者之必需在建立另外的 4 個 udp port 來轉輸聲音和影像，所以如果使用者人數很大量時，將會使傳輸的架構變得相當複雜，使的每個使用者的連線數是 4*n 的方式遞增，所以本專題以不同以往的方式來設計，主要是以 Client-Server 端的模式為基礎，首先使用者必需先加入會議中，先連線到 Chairman 中，彼此先建立雙方會議，在此 Chairman 和使用者之會分別用 4 個 port 用來傳遞彼此的視訊和聲音，透過 RTP 協定來作溝通，將視訊編碼為 H.263/JPEG 等格式，而聲音則可編碼為 GSM/ULAW/G.732/mpeg 等格式，依使用者的環境和喜好來設定，且彼此間沒有必需使用相同的格式來傳輸，除此之外，雙方的視訊和聲音可能因為網路傳輸的關係，所以可能在傳輸的過程中有不同的次序的方式到來，每一端必需各個分析其所串流的內容為何，必需由 RTP 的內容得知為那一種格式的串流，並選擇對應的解碼器，來將串流解碼並呈現給使用者，另外由於視訊和聲音是在不同的串流中，且來的次序並不相同，所以必需作視訊和聲音的比對，如聲音先到則必需先等待視訊的到來才一起呈現，如此使用者看到時才不會只有一個串流呈現的情形，反之也是一樣必需等到聲音的到來才會輸出。

在雙方建立完連線後，雙方底層必需隨時監看串流的變化，且也必需注意 RTP 的訊息是否有變化。在連線的過程中，可以由使用者的需求決定串流的傳送、暫停、停止等情況，當使用者暫停時，接收端必需也暫停串流的輸出呈現，另外如果使用都停止傳送時，必需由 RTP 協定告知接收端必需結束串流的呈現，並將畫面結束。這些動作雙方的底層都會自動的處理和維護。

4.5.7.2 三方到 N 方會議的處理方式

在本專題中可以由 Client-Server 的模式來建立 N 方會議，以下是傳統上處理多人會議上所需面對的難題

- 串流的處理相當複雜，在多人連線的情況下，Sever 端必需結收所有人的連線。
- Server 端的連線數多，且必需將串流和其他使用者再建立連線來傳輸，流運非常大。
- 每人到 Server 端的視訊與聲音串流來的順序並不相同，必需有特別的設計來作處理。
- 和其它人的連線。流程複雜且效能不理想，使用者和使用者之不如直接建立連來得好。

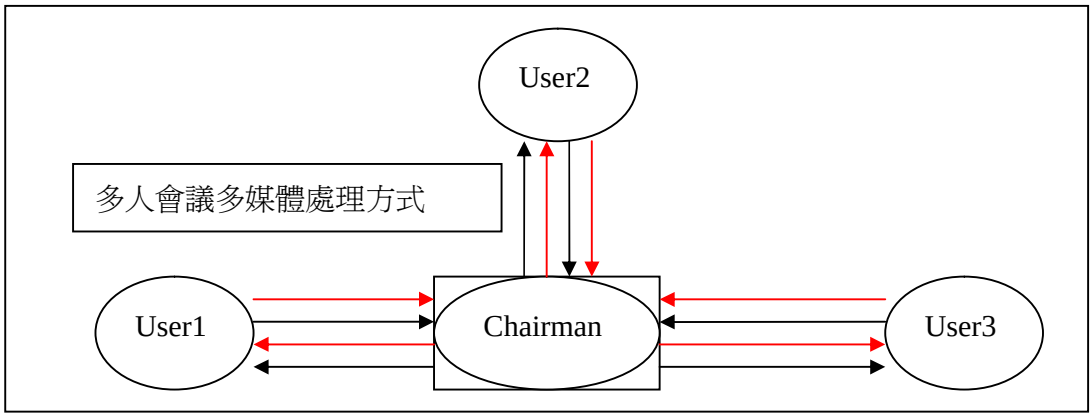
4.5.7.3 優點

1. 連線易管理，且連線方式單純，只需和 Server 建立連線，不必和其他使用者建立連線。
2. 特別是在 private ip 有 NAT 的環境下，使用者可藉由 Server 來使得使用者不管所處的網路環境為何，都可以連線(當然 Server 端必需有能處理該情況的能力)。
3. 使用者加入會議的方式簡化了，可以得到多方人的資料。

本專題中解決了以上的難題並加上特別的連線機智和方法，以下是**本專題多媒體傳輸特別的地方**

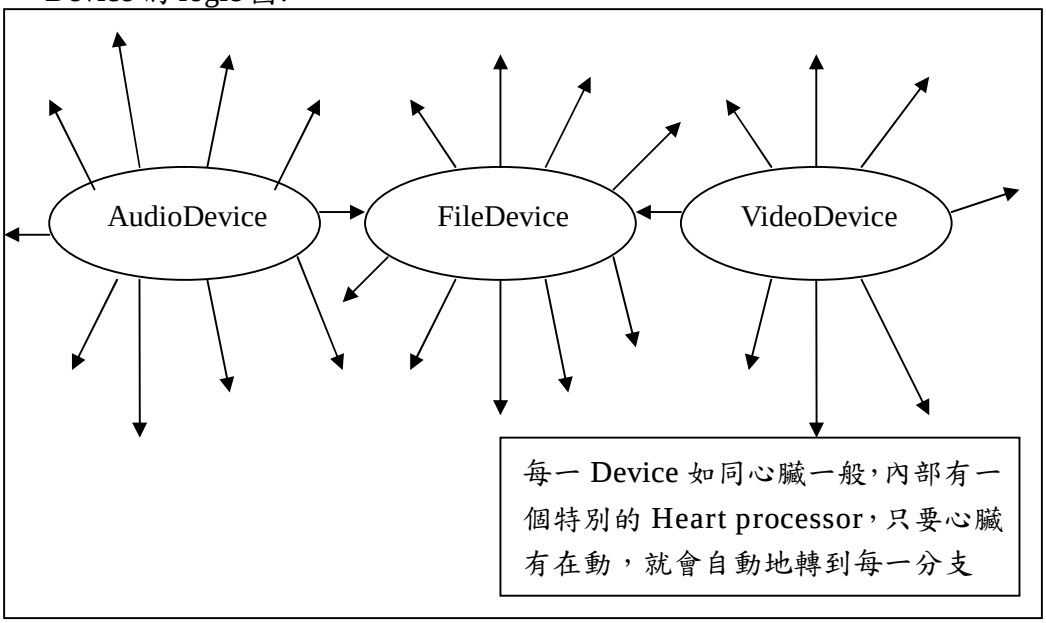
1. 雖是以 Client-Server 的方式連線，但只要 Client 和 Server 端建立基本的連線後，以後就不必再建立任何其它的連線了(此為特別設計)。
2. 以後有新的使用者加入，Client 和 Server 也只需建立一般的雙方通話連線即可。
3. Server 端會自動地將已加入連線的其它使用者，以已連線的通道，加入其它人的影像和聲音。
4. 使用了混音的機智和方法，和加上特別的混影像方法，使得每一人只要有一個聲音通道和一個影像通道即可傳輸無限制人數的資料。也減少了傳輸的頻寬。
5. 自動化的 Multicast 機智，大大地簡化了上層會議的設計機智，並提出特別的設計方法，使其可以自動且快速地將封包送致對應的使用者。
6. 可以不必依串流來的順序來使傳輸，會自動地作串流的分析和比對的工作。
7. 可配合底層的各動傳輸方法，可以和在各種環境下的使用者建立連線，包括 private ip to private ip、private ip to public ip、public ip to public 等環境。
8. 加上可減少 delay 的設計方法，使其連線速度接近一對一的連線。
9. 此設計方式更可以支援 VOD 的傳輸，使用者可以自已播放音樂、電影給其它的使用者觀看。每個使用者如同是 VOD Server 一般，增加了使用者使用上樂趣。
10. 支援 VOD 和會議並行的機智，使用者如同和 VOD Server 建立連線一般。
11. 加上了會議與其它會議結合的特殊設計，只需要兩個 Server 建立雙方通話，就會自動地用已建立連線的使用者用既有的連線得到另一會議的所有使用者資料。
12. 提出特別的設計架構，使得使用者端的多媒體資料如同心臟一般可以自由的分支和處理。

以下是多人會議時其連線架構:



4.5.8 使用者端 Device 的建立

首先對每一用者的裝置作了一個特別的架構，將每一種裝置如 Camera 定義為 VideoDevice，Audio Card 為 AudioDevice，各種多媒體檔定義為 FileDevice，Network Card 定義為 NetworkDevice，且在每一 Device 中加入一種 Heart processor 來作為其串流的開關和分支，當使用者建立了每一 Device 後，Device 會試著去測試連線，以確認該 Device 可用，之後會啟動其內部的串流作不斷地送出，之後只要對該 Device 要求就可得到該 Device 的串流，以後不管有少個要求，都可以動態地分支，重覆使用，如同有許多 Device 來作服務，而當使用者把 Heart Processor 停止時，所有分送出去的串流也會自動地停止，也可以動態地單獨暫停某一分支，然後再啟動。除此之外，也可以透過特別的介面，把二個 devce 的輸出轉為輸入，如可以把 AudioDevice 的輸出轉到 FileDevice 的輸入，就可以在把串流在到檔案去，更可以把如 AudioDevcie、VideoDevice、FileDevice 傳到 NetworkDevcie，如此就可以將串流傳輸到另一方的 NetworkDevice 來輸出呈現。以下是其 Device 的 logic 圖:



4.5.9 主席端內部的特別設計

本專題中主席必能有能力處理來自各方的串流,由於所有的串流都必需經由主席來傳輸到其它人,所以其流量非常大,且有相當複雜的情況,如串流來的順序,串流的來源的分析(因為所有人串流者經由主席,必需將串流傳到還沒有的會議中的使用者),所以設計在其內部設計了一個 Auto Reorder Queue,用來將不同順序到來的串流自動作 reorder 和分析其編碼格式,並將聲音混音,且將影像以多工器的方式模擬加入同一串流輸出,其中可以支援不同格式的聲音的混合,不同格式的影像的結合,更可結合來自檔案的串流輸出。

其中主要處理的工作如下:

1.分析來自不同使用者的串流,辨別出其編碼格式,還必需快速地確認是否以可以輸出。

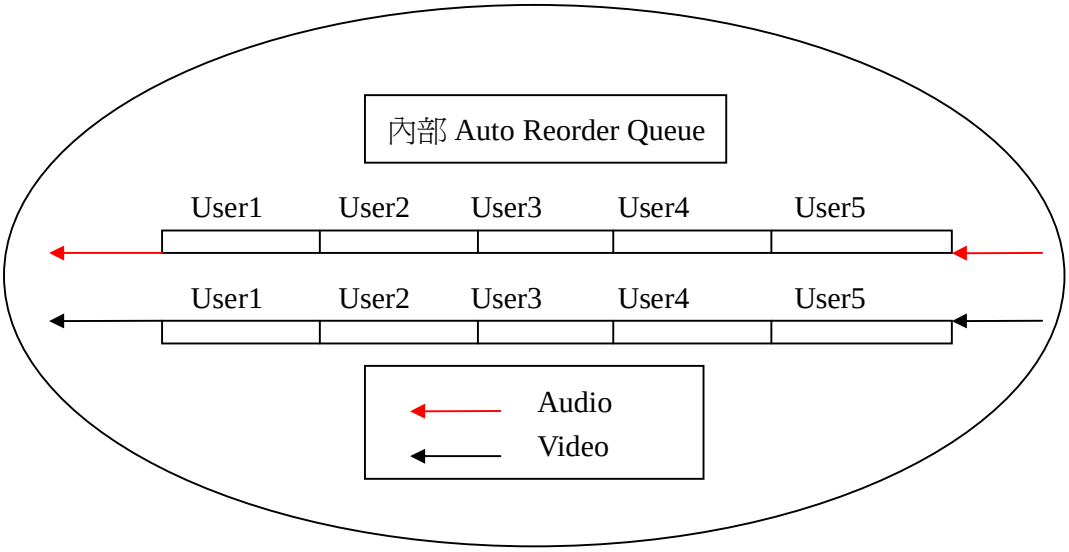
2.確認串流來的順序,因為使用者的網路環境可能不同,所以可以後加入的使用者串流比較早到,所以必需等待其他先加入使用者的串流到來,並且將要輸出的串流作 reorder 的動作,使其輸出來其它使用者時仍能維持一定順序,否則接收端可會看到不同的視訊畫面在另一個視訊畫面上交換出現,形成串流跑錯輸出地方。

3.當所有到達主席的串流都以依序排列後,再來就是將串流混合,其是採用了所謂的「交錯式混合」,將所有收到的聲音和影像,以不改變其原來的格式,交錯地將聲音輸出,當使用者端收到串流後,可依當時交錯的方式,取出對應的封包內容,所以當使用者收到時,可以得到當時每一使用者所傳輸的原本格式。

4.加入了聲音和影像的來源的分析和處理,因為每一使用者其實本身並不知道有其它的使用者加入會議中,且聲音和影像是分開傳輸的,所以必需能夠快速地確認那一個影像是對應那個的聲音,並且輸出給使用者,若比對錯誤時就可能出現聲音和影像的輸出是來自不同的使用者,出現錯位的現像。

5.當在會議中的某一方要結束會議時,主席會先得知,並要將其在 reorder Queue 中移除,且不可以使其它在傳輸中的使用者斷線、暫停等不良情況,使得使用者可以動態地加入和離開,但卻不會影響其它使用者的連線。

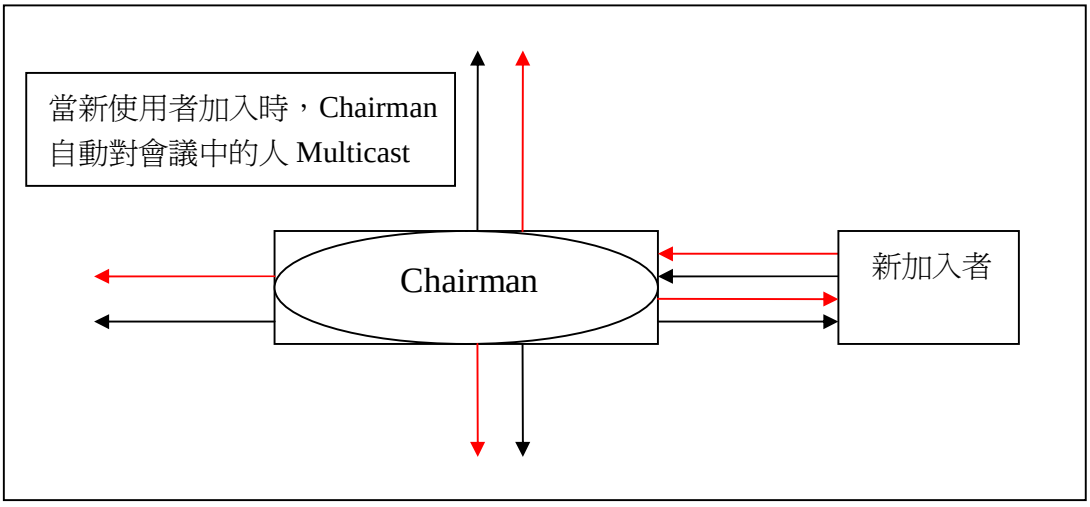
以下是主席端中 Reorder Queue 的架構



4.5.10 內部的 Multicast

當有新的使用加入到會議中時，主席會將得到的新串流送給其它已加入會議中的人，所中是軟體的方式模擬 Multicast 的傳送，主席會將收到的串流，自動作串流的分支的動作，將串流分送給其它人，這個過程相當快且不可以斷斷續續，必需一直維持其串流的穩定。

以下是其 Multicast 的過程：



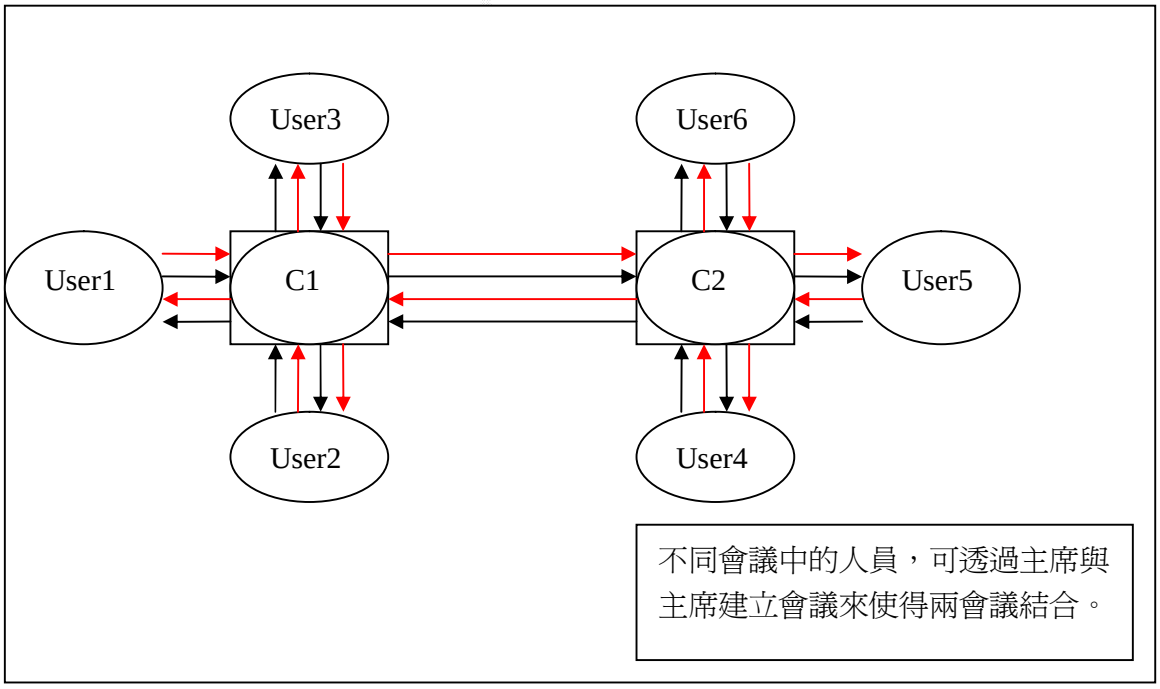
4.5.10.1 不同會議的結合

在多人會議的過程中，有可能在二個不同的會議有要結合的時侯，如在多人討論時，因為工作的需要，可能動態地要將二個會議中的人結合一起討論，但又不想浪費時間重新建立會議，和邀請其它人加入會議，所以出現了可能需要結合會議的情況發生，而結合會議的過程又是另一種情況和架構了，但本專題發展出的架構卻可以如同雙方通話的方式直接將二個不同的會議結合，其作法如下

- 一方的主席邀請另一方的主席
- 雙方的彼此會建立如同一般會議連線
- 雙方主席會各自地對加入會議的人們作 Multicast 的動作

經由以上的動作後，串流會自動地轉送到所以在會議中的人，所以另一方會議的人得到另一方人的所有串流，而且這些過程都是在原本以建立的通道來傳輸，不必再另外建立其它的連線了。

以下是其連線的架構圖：

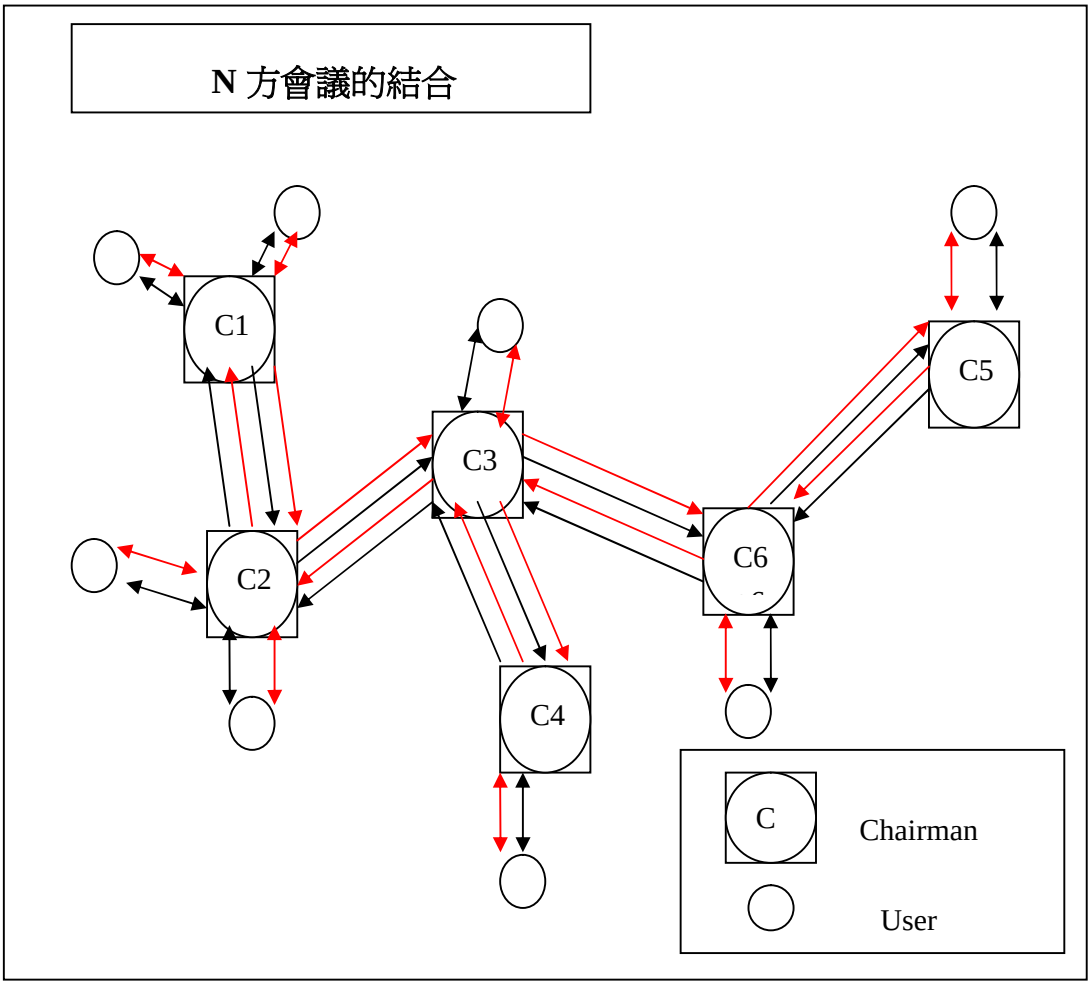


4.5.10.2 N 方會議的結合

經由以上的方式更可以發展成 N 方通話的架構，且如果將組合應用到 VOD 時，更可以形成另一種形態的「Media Router」，而當多方會義結合時要考慮的情況有些不一樣，因為可有使用者同時加入到不同中，而這些會議還沒結合，所以那些會議中的並不知道另一會議中有相同的使用者，所以在結合的過程中可能發生得到已在另一會議中，和本原會議中相同的人的串流，這一種情況，本專題中也有研究到，以下是其處理情況:

- 不同的會議彼此結合，傳輸串流到另一會議中
- 當其中一個和該會議結合的會議發現有一串流中有其連結會議中的人員時，會自動地省略其串流的傳輸，使其不會到另一個會議中重覆出現。
- Media Router 都已傳輸到另一會議中，會議結合完成。

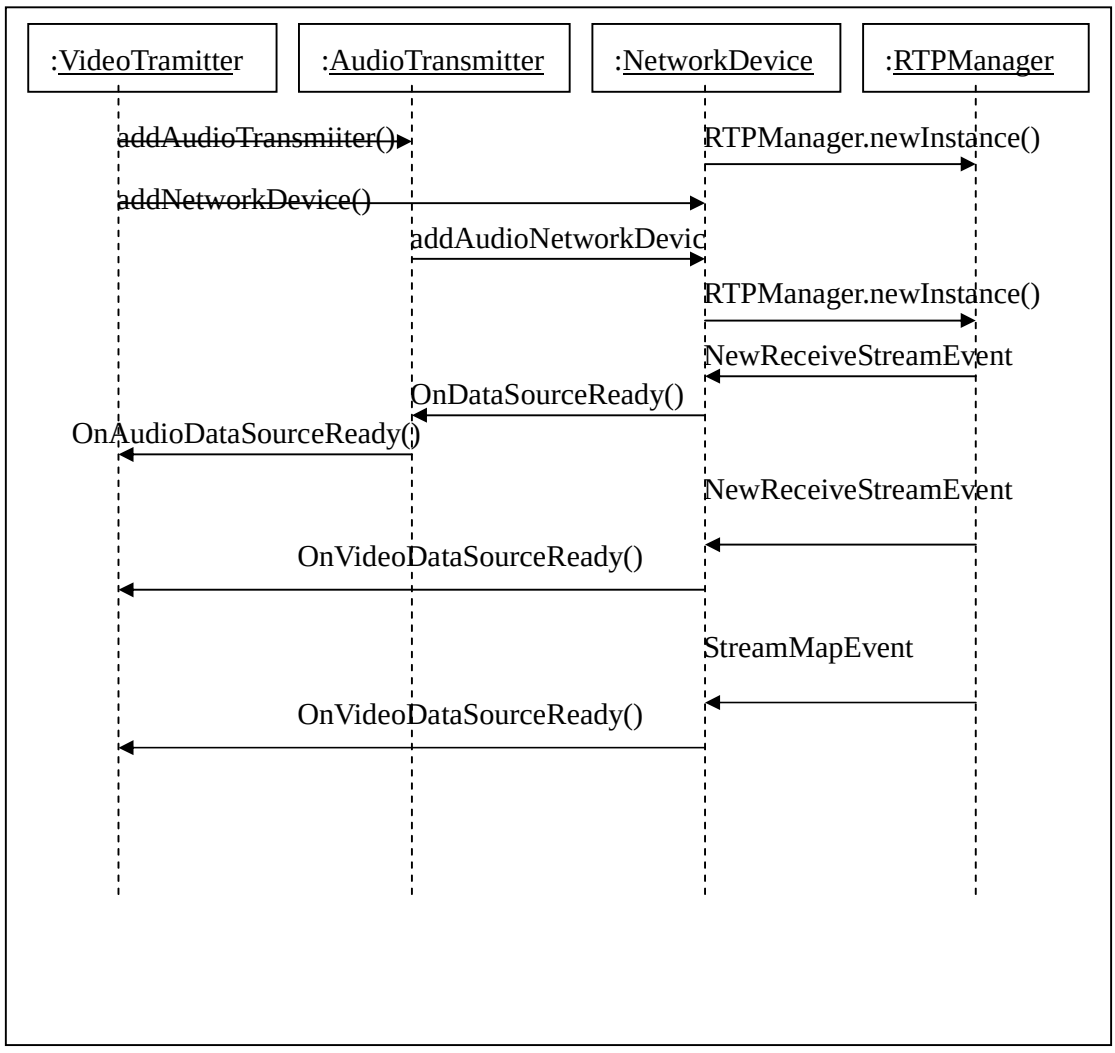
以下是 N 方會結合後連線的情況:



4.5.11 一般會議建立的細節:

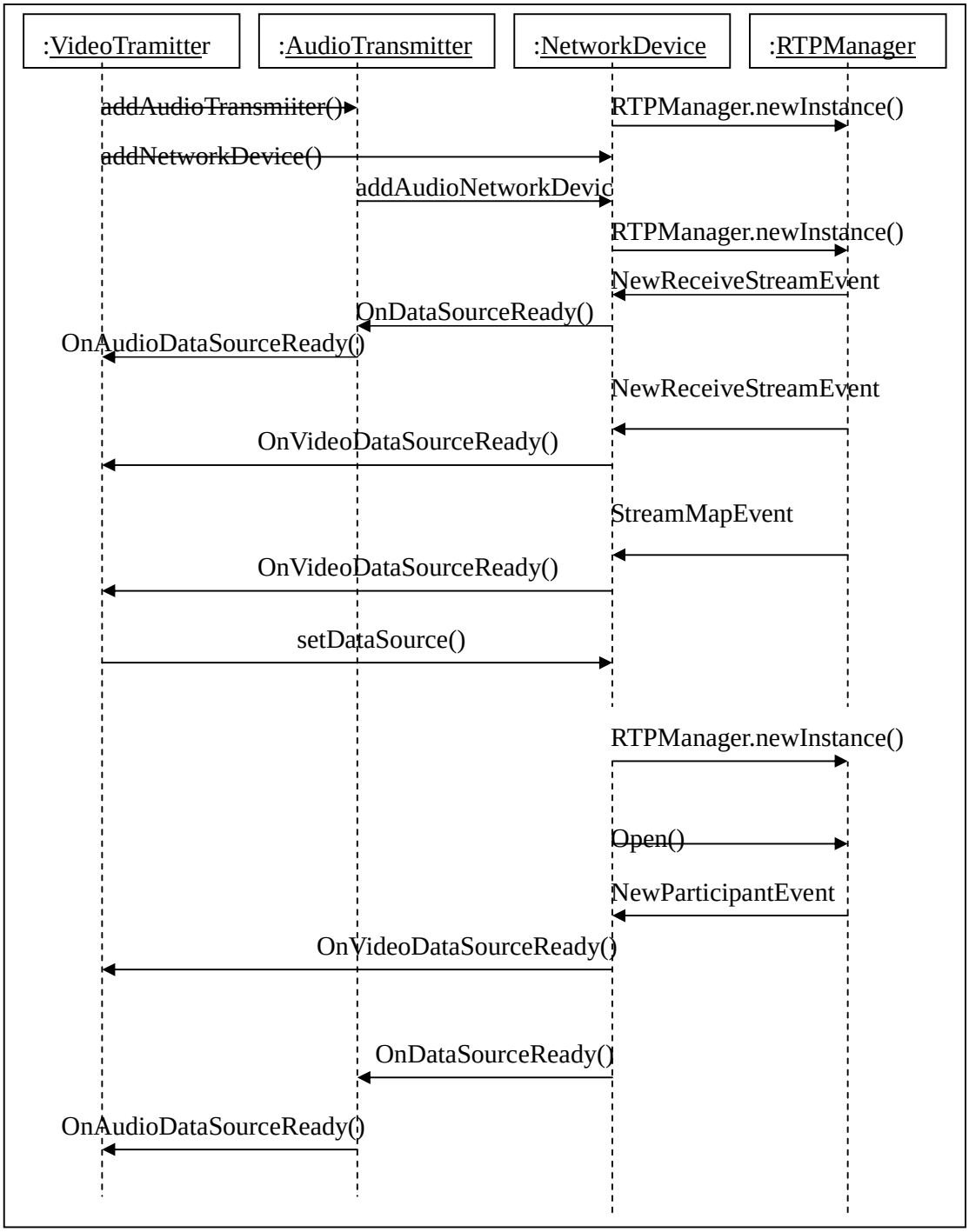
當使用者要和其它建立會議時，會依以下的過程來建立，首先必需知道另一使用者的位置，這一點可由本專是中的 BindirctionPipe 建立得到，並建立 NetworkDevice 結使用者，當建立完需要收和傳送的 NetworkDevice 之後，可由 VideoTransmitter，AudioTransmitter 來使用該 NetowrkDevice 傳輸，另外可由 AudioReceiver 和 VideoReceiver 來接收另一使用者的資料，之後每個 NetworkDevice 的底層會有一個 RTPManager 來監控所有 RTP 的事件，當有真正的新串流來是會向上回報給個 Receiver 來作處理，這一層就是主席最重要的部分，有串流的分析和處理，並有 Reorder Queue 和混音和影像的處理。

下圖是其動態的 UML 圖形說明:

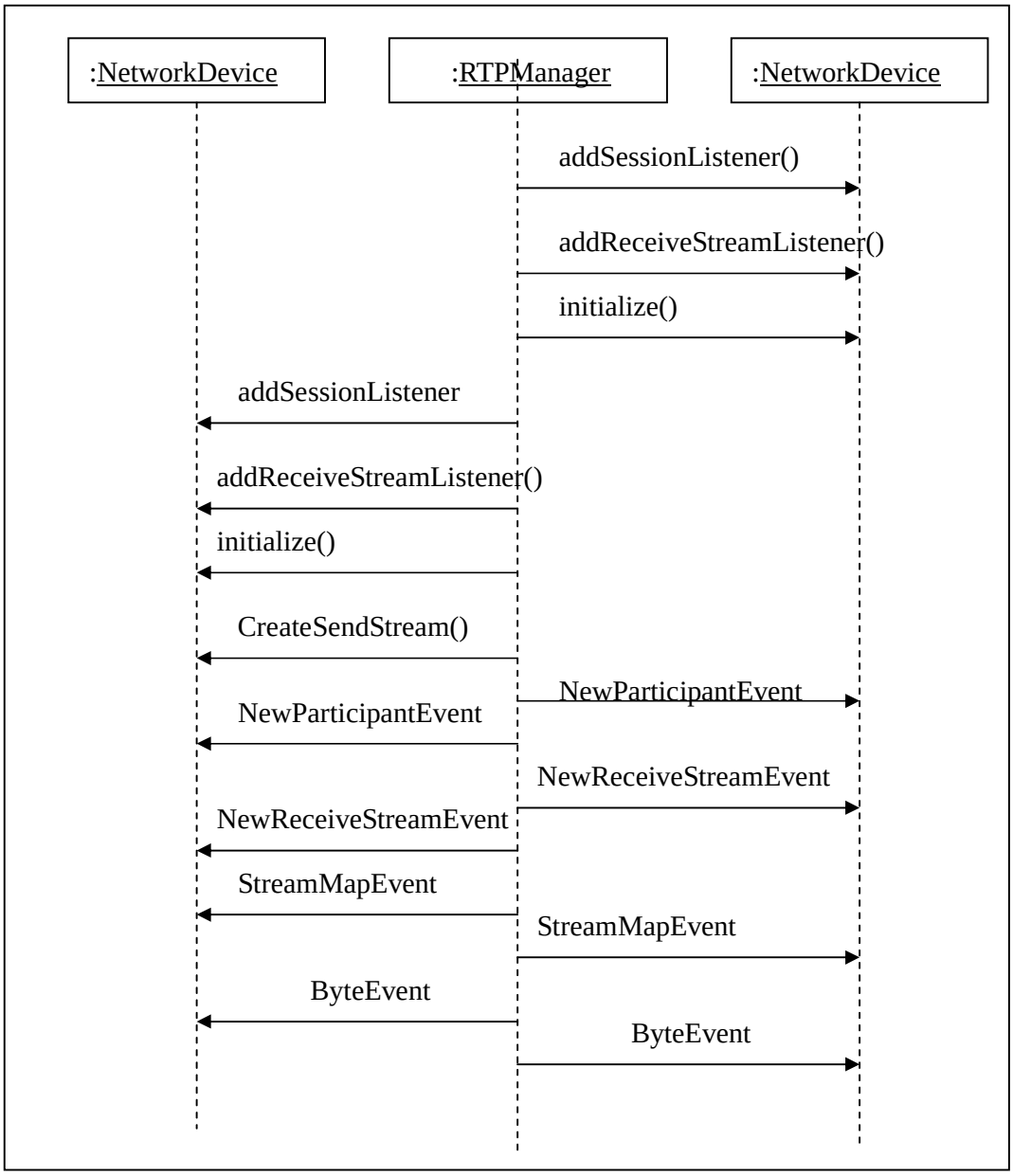


以下是主席的情況，除了基本的流程外，當所有的串流都已經排好時，就會重新指定新的串流給原本已在傳輸中的NetworkDevice，並在串流不斷的情況下，組合入原本的傳輸通道中，其中主要的工作就是呼叫setDataSource()這個動作，其中加入了重排後的串流和原本串流的來源。

以下是主席中的動態的UML



以下是當雙方建立了互相建立連線後，底層對雙方的Transcation過程，必需經由RTP協定互相說明彼此的狀態和內容，並由RTP的Sender Report和Receiver Report說明其所傳輸的資料是那種格式，加入者是誰和彼此之間的資訊,只要也必需監控所得到的多媒體的內容是否已可以使用，當到達一定的量時，就會告訴上層的VideoPlayer物件來解碼並輸出。



4.5.12 影響語音品質因素之探討

目前的網際網路僅能提供一種最簡單的服務品質(Quality of Service；QoS)，以QoS 並不能被保證。然而隨著網際網路的快速擴張及應用之普及，有著越來越多的網際網路應用需要網路能提供即時性的服務(Real-time Service)。這些應用包含了網際網路電話、視訊會議(VideoConferencing)等即時系統。因此，一些影響服務品質的因素，如封包遺失(PacketLoss)、封包延遲(Packet Delay)及延遲差異(Delay Jitter)等均將予以克服以提升服務品質。

4.5.12.1 語音編碼壓縮之影響

因為語音資料在網際網路上傳送需要經過壓縮過後，其語音之資料量才會變小，目的在於節省網際網路之傳送頻寬以提供其他更多之服務。要將聲音訊號做處理就一定得付出時間延遲的代價，因此在網際網路電話中，則以語音壓縮/解壓縮運算最為沉重，而且也是佔據處理器運算資源最多的部分，所以這是實現網際網路電話上是相當大的瓶頸。而解壓縮所需的時間大約是壓縮時間的一半，且壓縮/解壓縮的延遲可分為兩個部分：

1. 處理延遲：它是執行壓縮工作所需要花費的時間。
2. Lookahead 延遲：因為壓縮的時候要比較前後之訊框(Frame)才能在其大部分相同之情況下得到好處，這就得等下一個訊框的音訊都採樣完畢後才能開始比較並壓縮，而這段等待的時間就是Lookahead 延遲。因此，每一訊框的長度越短，不論是等待或是壓縮的時間都會越小。然而，每個標準之壓縮演算法的處理延遲及Lookhead 延遲的時間並不相同，如在網際網路電話中常用到的壓縮演算法有G.723.1 及G.729，它們的處理延遲分別為30ms 以及10ms，而其Lookahead 延遲則為7.5ms 以及5ms。因此，由此可知，G.723.1壓縮演算法較G.729 之處理時間來的大，但是其壓縮率卻是比G.279 來的大。

由於語音編碼與解碼、壓縮與解壓縮的處理過程將時間拉長的話，將造成網路的Jitter Delay 變長，這會造成接收端在應播放該封包而未播放時，便會將此封包跳過而往後播放，因而造成語音封包的遺失現象，因此應該要盡量避免此現象的產生導致影響語音的品質。

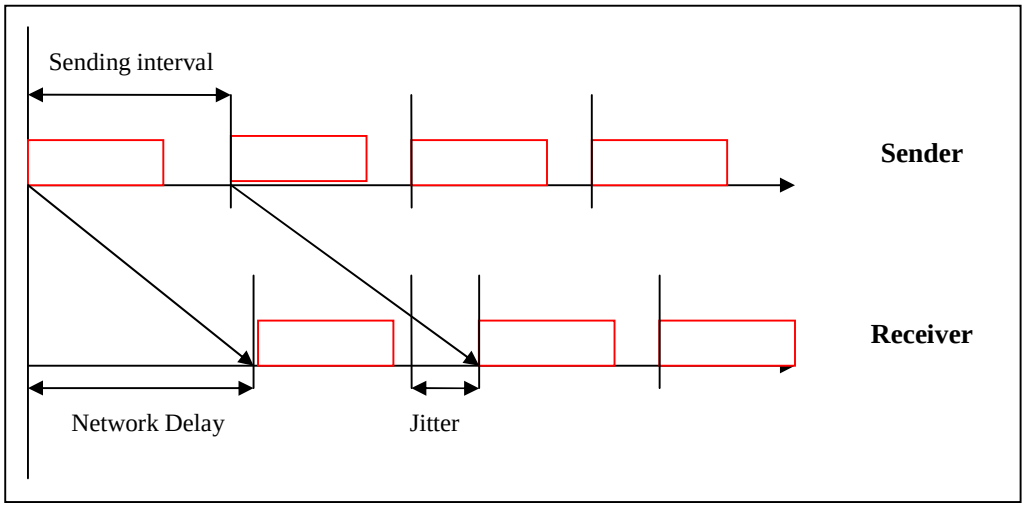
4.5.12.2 封包遺失之影響

語音封包在傳過程中會經過許多的路由器(Router)、閘道器(Gateway)等等設備。因此，當多個封包同時到達某一路由器時，在其內部會形成一個先進先出(First-In-First-Out；FIFO)的佇列。而由於網路的流量過多導致過度壅塞，使得網路中的節點(Router或是Gateway)其內部佇列因為飽和而將導致緩衝區溢位(Buffer Overflow)，此時，這些節點會將應該接收的語音封包丟棄因而造成語音封包的遺失。另外，在接收端這邊正一直連續不段地接收語音封包，若有些語音封包可能因為網路壅塞或是其他情形而比較晚到達接收端，使得該其播放而來不及播放，即RTP協定中之時間戳記的播放時間過了，便會被接收端視為封包遺失而將此晚到之語音封包丟棄。

因此，除了語音訊號的壓縮會影響通話品質之外，網際網路電話在通話品質上最大的問題則是在於語音封包資料的遺失和延遲。封包的遺失主要是因為網路傳輸上設計的錯誤所造成，而封包的延遲往往也跟演算法的設計有絕對的關聯，比如封包的組裝(Packetization)、傳遞的方式(如中間是透過寬頻之骨幹網路或是窄頻網路)、加解密的速度等等。而當語音封包因為網路壅塞或是因為較慢到達而未播放造成封包遺失之現象，這都會讓使用者聽起來覺得有點不舒服，若是語音封包遺失的遺失量持續增加的話，會造成使用者根本無法聽清楚對方在說些什麼，而達到無法使用的地步。因而語音封包的遺失率在使用者所能容忍的範圍是在5%以下，若是超過此一門檻，都會讓使用者無法辨識對方所說的話，因此本論文之主要目的便在減低語音封包因網路或其他因素之影響而造成語音封包遺失的現象，使得使用者能在網路壅塞的情況下也能聽到不錯的語音品質。

4.5.12.3 延遲差異之影響

延遲差異(Delay Jitter 或稱Delay Variance)是指封包經過網路傳輸過程中延遲時間的差異，如下圖說明



延遲時間差異乃表示封包傳送至播放裝置之時間間隔的差異。就整體來說，延遲差異可以定義為兩個不同封包在同一連線得傳輸過程中所經歷的最大絕對差值(Maximum Absolute Difference)。而造成延遲差異的主要原因有以下幾點：

- 語音資料在編碼、壓縮、解壓縮過程中的時間差異
- 作業系統排程(Scheduling)的時間差異
- 網路傳輸過程中的時間差異，而就網路部分的延遲差異而言，其主要原因又包含了以下幾點：
  - 實體的延遲差異(Physical Jitter)
  - 流量控制的等待時間(Flow Control Waiting Time)
  - 佇列的延遲時間(Queuing Dealy)

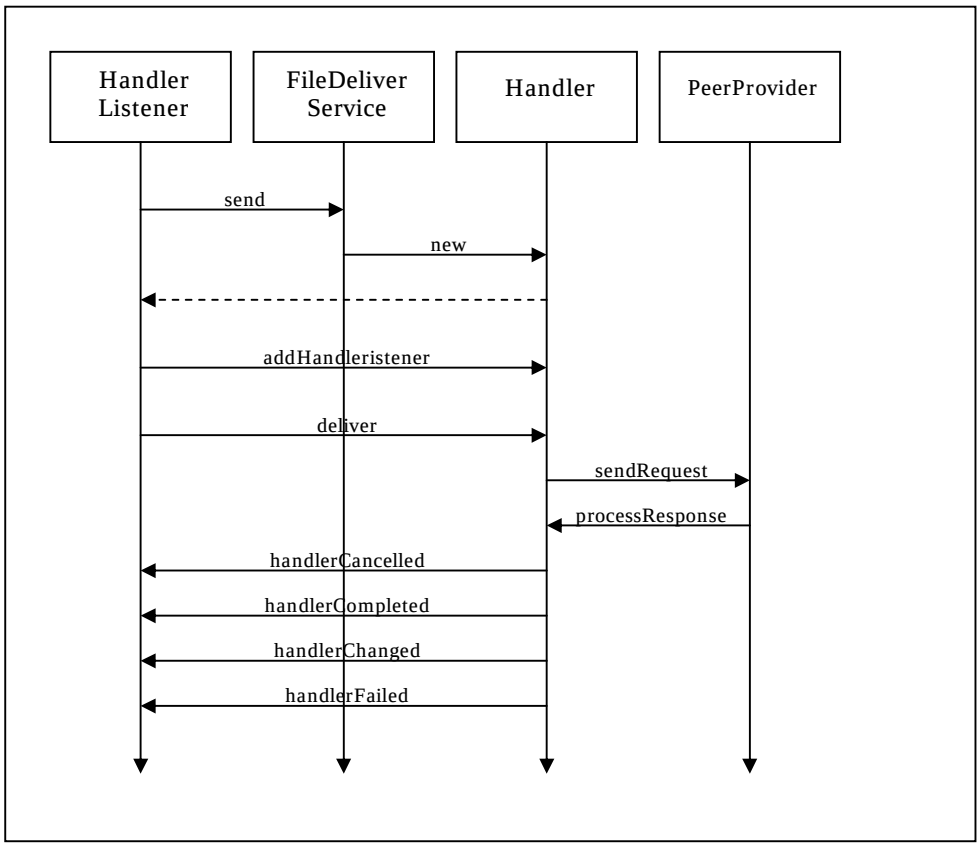
延遲差異對語音品質最直接的影響在於聲音播放的連續性，因為當接收端接收到的語音封包資料的延遲變異的時間是時長時短的話，播出的語音資料會產生斷斷續續的聲音，這會讓使用者聽起來會覺得有不舒服的感覺。而一般解決的方法是加入一個緩衝區(Buffering)的時間來做調整，讓播放出的聲音能夠較為平順地播放出來，不會讓使用者聽起來有不舒服的感覺。而一般而言，動態播放時間的調整可分為封包保留(Packet Preserving)及時間保留(Time Preserving)者兩種方式。封包保留方式乃保證大多數的封包到達接收端直到可被播放為主，對於慢來的封包接收端會等待其到來再行播放即可，因此播放時容易造成不連續的縫隙(Gap)，而且可能會增加太多的整體延遲時間。所以此方式比較適合使用在非互動(Non-interactive)的應用軟體上，如隨選視訊(Video on Demand；VoD)等等。而時間保留方式則是以減低整體的延遲時間為其主要的目的，利用程式去計算每個封包的播放時間，若某個封包之延遲超過其播放之時間則是為封包遺失而將之丟棄，如此一來就不會有不連續的播放縫隙及延遲過長的問題。所以此方式比較適合使用在互動(Interactive)的應用軟體上，例如網路電話以及視訊會議等等

4.6 檔案傳送系統

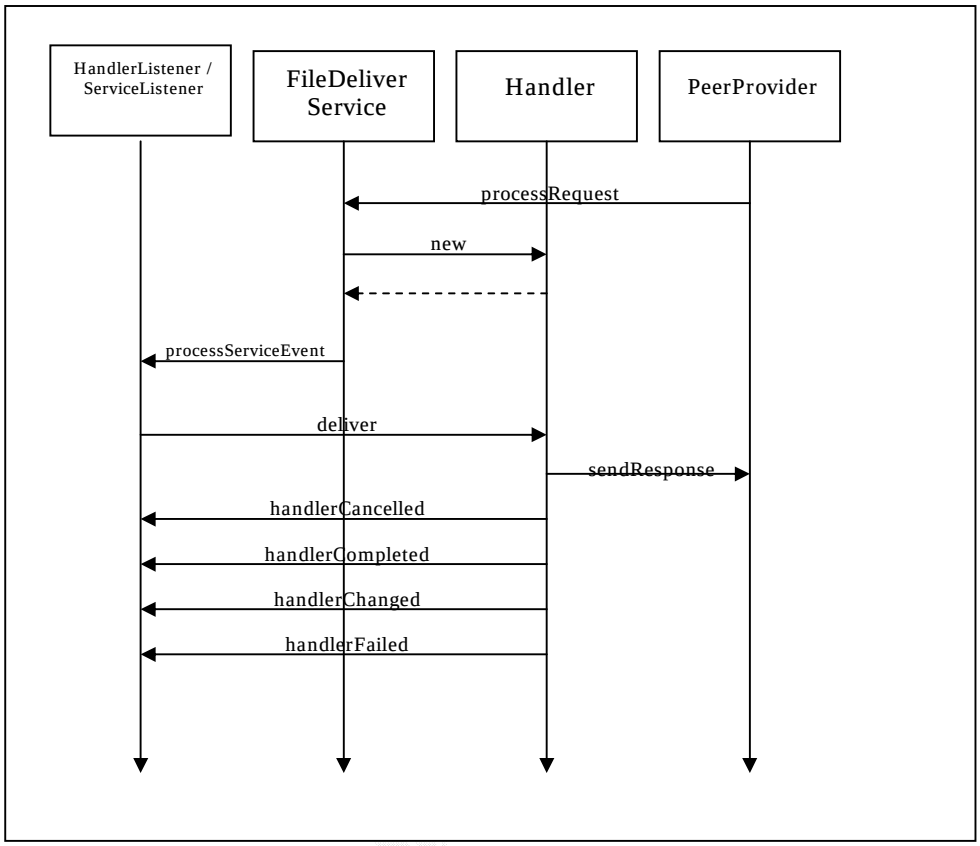
檔案傳送系統主要的功能是提供使用者在交談或參與會議期間將特定文件資料傳送給特定、多數甚至是所有連絡人。其使用方式相當地簡單，當使用者欲進行檔案的傳送時只要選取特定連絡人指定傳送的目標，而後再由檔案選取對話框選取欲傳送的檔案並等待對方接受或拒絕；當使用者接收到對方傳送檔案的請求時，會在主視窗上方顯示是否接受傳送的對話框，而使用者只要依據檔案是否為本身所須選擇接受或拒絕接收。

此系統底層所支援的檔案傳送之端點服務元件，即上層所發出的請求皆由這個服務元件進行處理，而上層所顯示的接受請求對話框也是由這個服務元件進行事件觸發產生的，而發出請求和確認接收與否的對話則是透過服務元件向端點提供者提出協定訊息的傳送達成的，而檔案傳送期間所使用的方式則是請求管線服務元件建立端點間的傳輸通道，下圖檔案傳送服務及其下所屬元件間的執行順序圖。

檔案傳送請求之順序圖：



檔案接收確認之順序圖：


HandlerListener / ServiceListener
FileDeliver Service
Handler
PeerProvider
processRequest
new
processServiceEvent
deliver
sendResponse
handlerCancelled
handlerCompleted
handlerChanged
handlerFailed

4.7 電子白板系統

本系統提供同一視訊會議的共同繪圖板，以表達文字描述不清處，其使用規則如下：

1. 同一視訊會議的使用者皆可啟用電子白板。
2. 當會議中某參與者啟用電子白板後，其它參與者也將建立電子白板。
3. 視訊會議只能建立唯一的電子白板。
4. 某一參與者關閉電子白板，其它參與者的電子白板也將關閉。

由以上可以看出電子白板系統依賴在視訊會議，若未建立視訊會議，則無法使用電子白板，因為電子白板的建立是利用視訊會議的參與者列表。

4.7.1 系統架構

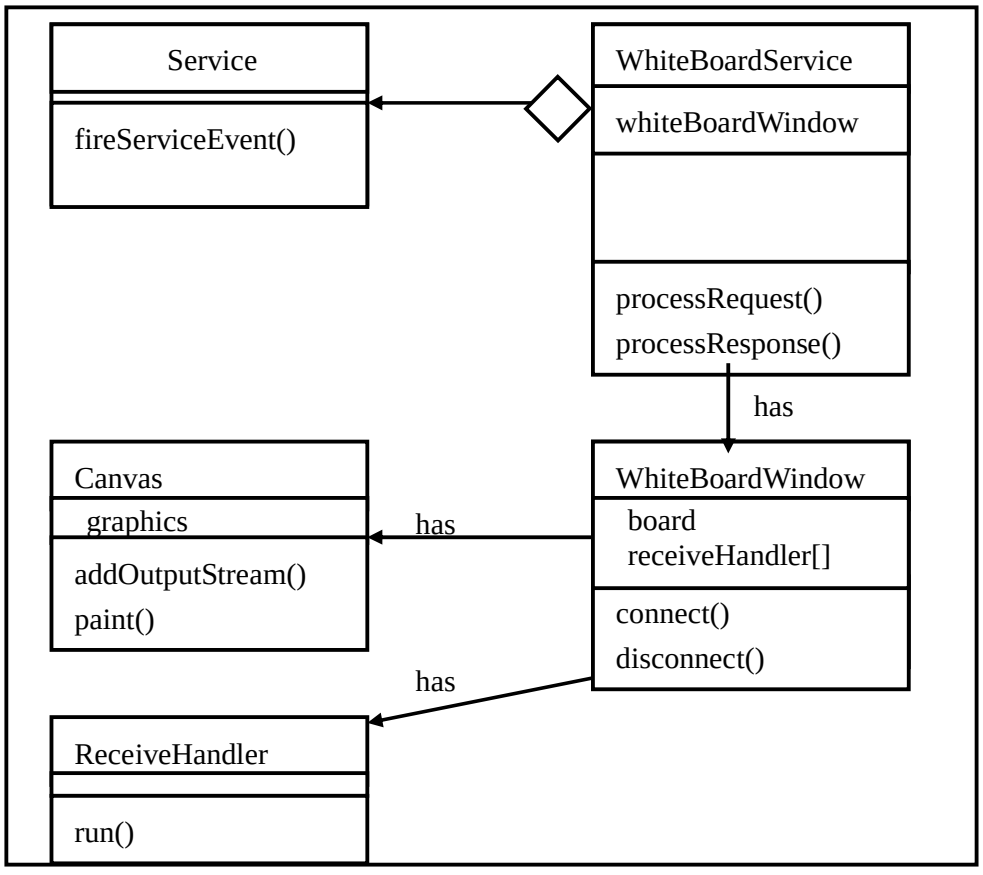


圖 4.7.1 電子白板類別圖

功能描述如下：

WhiteBoardService：

主要是負責電子白板建立及關閉的協定處理。

WhiteBoardWindow：

負責使用者視窗界面，產生管理ReceiveHandler，產生管理資料管線及視窗事件聆聽。

ReceiveHandler：

開啟繪圖資料聆聽埠，加以聆聽，並剖析資料使成有意義的繪圖資訊。

Canvas：

此類別與java.awt.Canvas並無相關，負責聆聽滑鼠事件及繪圖。

4.7.2 建立，運作及關閉

(1) 建立

首先由某同一視訊會議參與者啟動電子白板，該參與者的WhiteBoardService將對其它參與者建立輸入管線並送出CREATE Request，如下圖。

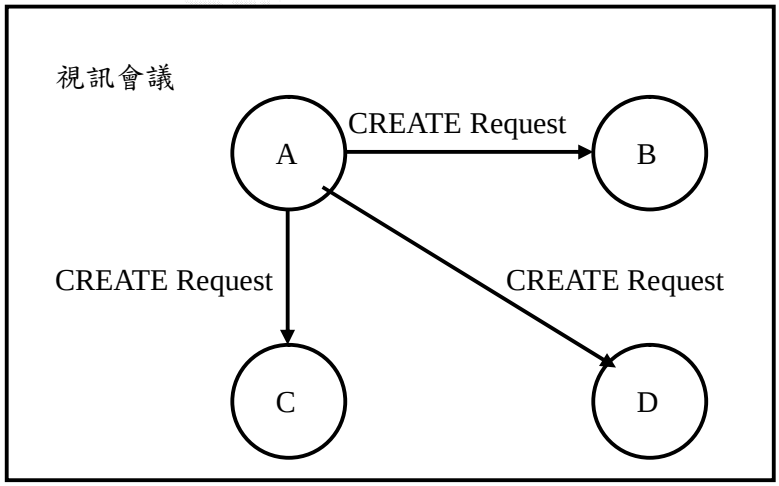


圖 4.7.2.1 建立電子白板

其它參與者收到CREATE Request後，利用請求內的參數建立與參與者A的輸出管線，建立與A的輸入管線並將管線參數加入回應OK內，然後開始向除了A以外的參與者發出INVITE Request，如下圖。

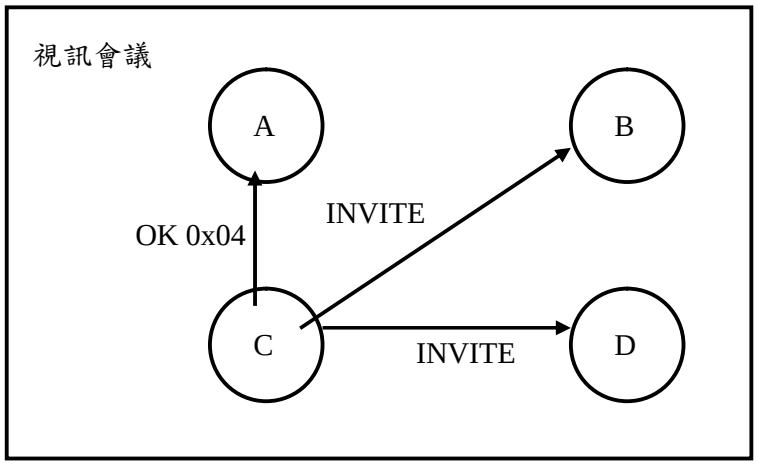


圖 4.7.2.2 建立電子白板

上圖只畫出某一參與者的情況，其它參與者(B、D)雷同，在送出INVITE Request前，先對該INVITE Request的接受者建立輸入管線，同樣的，將管線參數加入請求中，由於每位參與者皆如此做，最後網狀的管線分佈圖將被建立起來，如下圖。

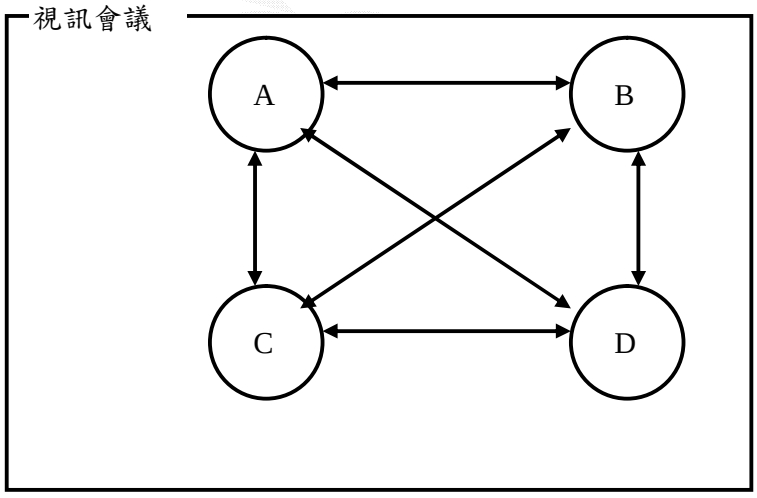


圖 4.7.2.3 電子白板網狀管線分佈圖

使用網狀分佈的好處是資料的傳送速率較集中管理再行轉傳快，由於繪圖資料封包雖小量但必須即時，故傳輸速率要求較高，以下是繪圖封包的格式：

- 第0位元組 – 繪圖模式，可分為畫筆、圓形、矩形、線形及橡皮擦模式。
- 第1位元組 – 畫筆大小，可分為大中小三種。
- 第2~5位元組 – RGB碼。
- 第6~7位元組 – 滑鼠X軸的位置。
- 第8~9位元組 – 滑鼠Y軸的位置。
- 第10~11位元組 – 寬。
- 第12~13位元組 – 高。

當然有些欄位在某些模式下是不必要的，但如果為了節省位元組空間，而必須在每次收到封包時加以判斷類型，是不符合效益的，在電子白板的建立中，最重要的任務就是建立網狀管線。

(2) 運作

電子白板的運作方式很單純，只要有繪圖事件發生，就執行兩件事，首先在繪圖板上繪圖然後將繪圖相關資訊包入封包送出，以上皆是由Canvas物件負責，當ReceiveHander物件收到封包後，予以解析後，直接呼叫Canvas物件繪圖，以下是兩種繪圖情形的UML順序圖。

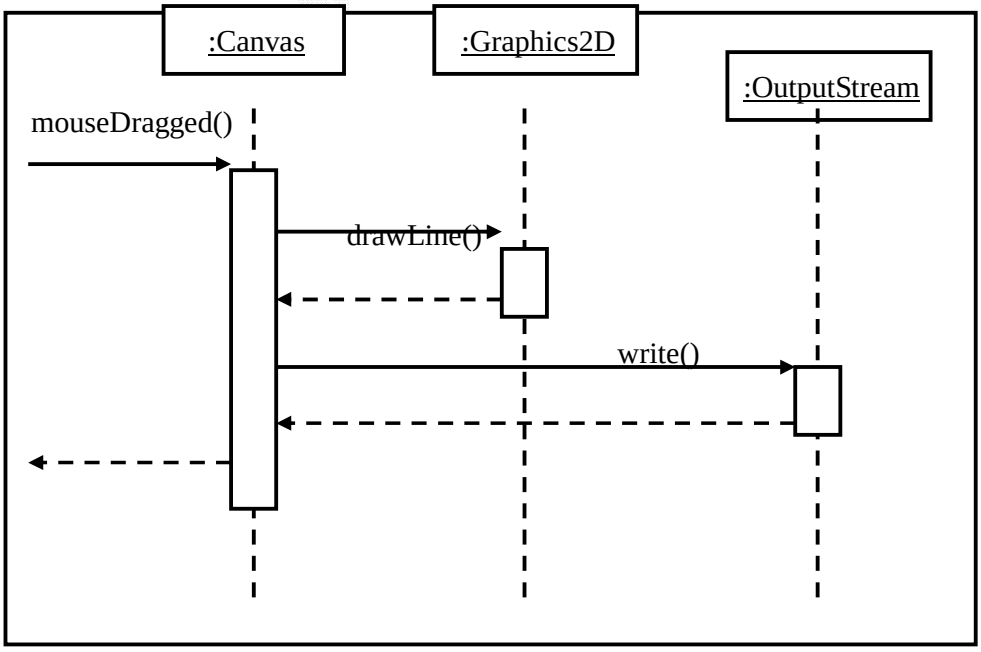


圖 4.7.2.4 畫筆模式

使用者拖曳每一像素都會將上面的流程會執行一次，也就是說在正常的使用下，可能一秒內就執行七八十次，這也是整個電子白板系統中，較為困難的部份，如何避免上一次流程尚未完成就進行下一次流程而造成資料封包的重疊，因此，必須減少以上流程的時間，不能使用昂貴的運作，所以電子白板使用陣列儲存資料輸出串流，能預先載入的動作及資料即預先載入，使用bit shift運算代替乘法等，否則即使有Java語言的同步化保護，也會造成延遲導至使用上的不順暢。

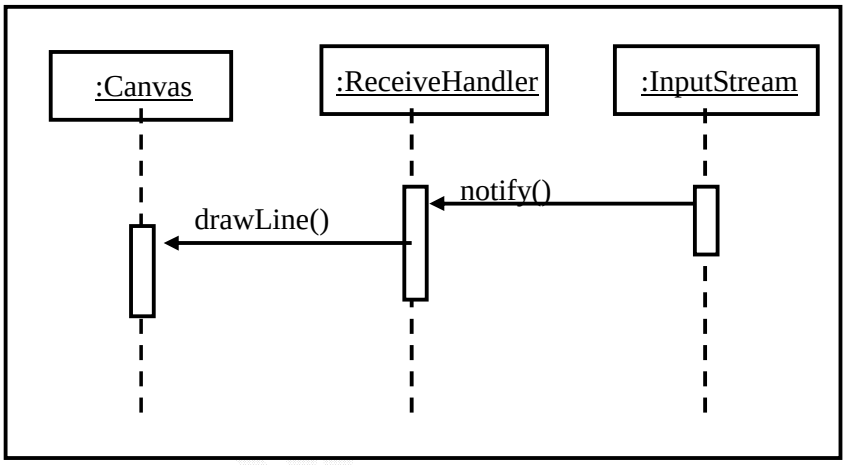


圖 4.7.2.5 遠端繪圖

(3) 關閉

電子白板的關閉由視訊會議中的某一參與者發啟，送出BYE Request至每一參與者，當電子白板系統收到BYE Request即關閉視窗界面及資料串流，然後回應BYE Response，以下是遠端關閉的UML順序圖。

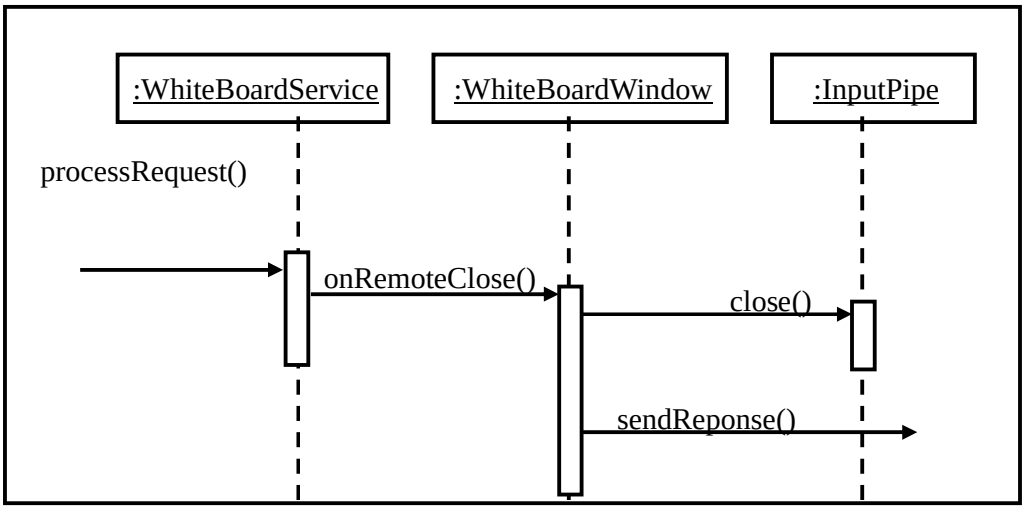


圖 4.7.2.6 收到BYE Request

4.7.3 討論

電子白板未來可能改進之處有：

(1) 不依賴視訊會議系統，任兩使用者可隨時建立電子白板會議。

(2) 任一參與者關閉電子白板，其參與者的電子白板也不需一併關閉，改為其它方式告知某參與者已關閉電子白板，最後剩一位參與者使用電子白板時才被迫關閉。

(3)在電子白板開啟後，新加入的參與者無法參與電子白板會議。

4.8 線上投票系統

本專題以商業視訊會議為取向，有鑑於一般會議常須要投票表決，故發展本系統，本系統提供有關投票動作的選項有：

(1) 該次投票的標題
(2)該次投票的簡要敘述
(3)每一參與者幾票
(4)匿名、不匿名投票
(5)公開、不公開投票結果
(6)該次投票時間

線上投票系統也是基於視訊會議的基礎上，只有主席權限的參與者可以舉辦投票並決定投票的相關選項，整體投票流程分為三步驟，首先，主席權限參與者填寫投票相關選項然後發佈投票公告，各參與者開始投票直到投票時間截止或全部的參與者皆投票完畢為止，最後是開票，如果選擇不公開投票結果則沒有此步驟，僅有主席參與者可看見投票結果。

4.8.1系統架構

線上投票系統主要是由PollingService負責處理協定並提供Method由三個視窗呼叫，PollingCreateDialog、PollingDialog、PollingResultDialog。

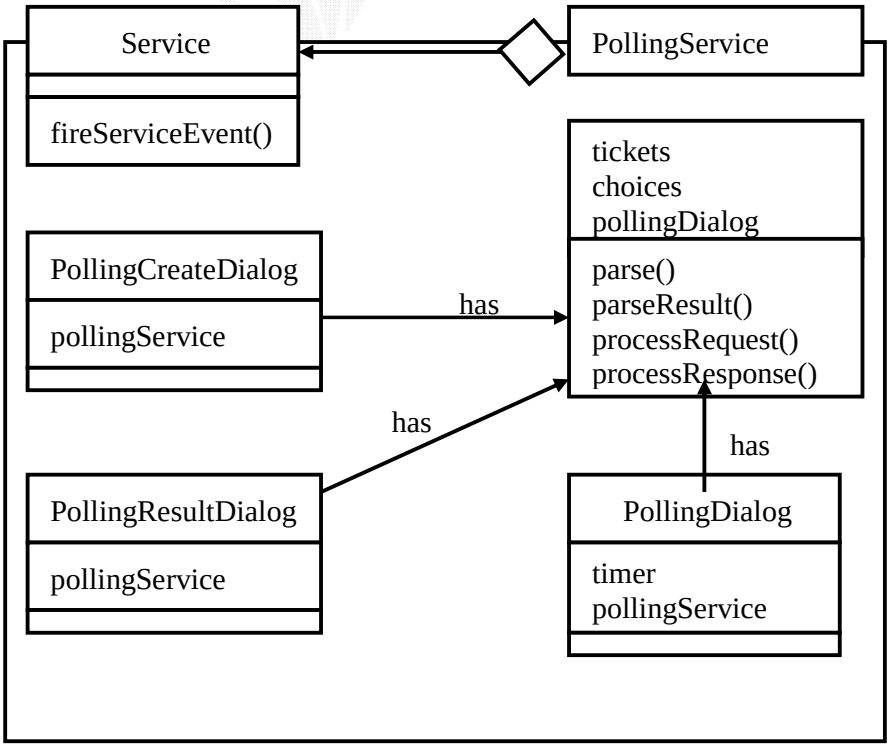


圖 4.8.1.1 線上投票系統類別圖

4.8.2 建立及終止

主席參與者在PollingCreateDialog視窗界面中，填好投票相關選項後，將呼叫PollingService予以初始化，並送出起始投票的請求給其它參與者，其它參與者之PollingService收到SPOLLING Request後，剖析其中的相關投票選項參數，建立PollingDialog。

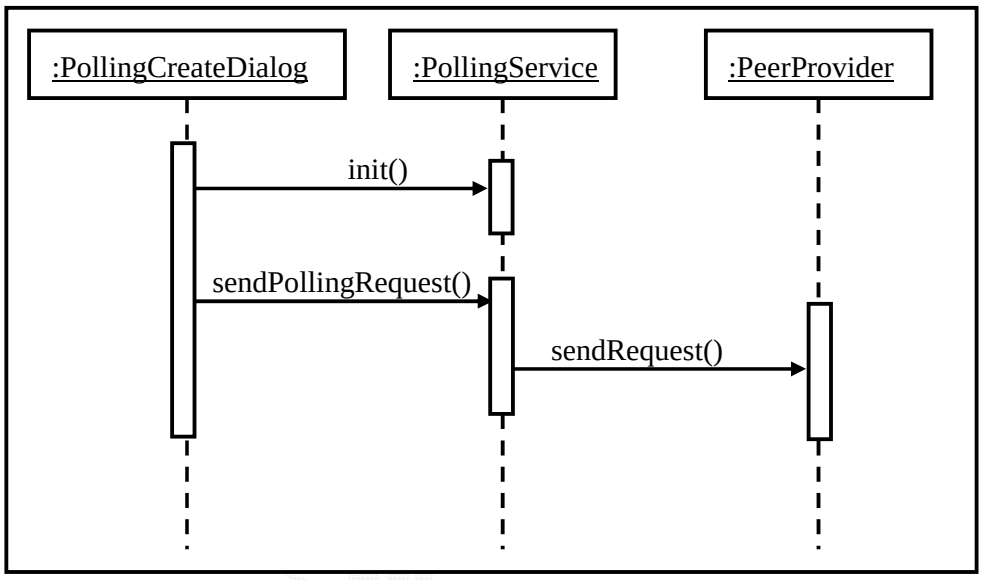


圖 4.8.2.1 主席參與者起始投票順序圖

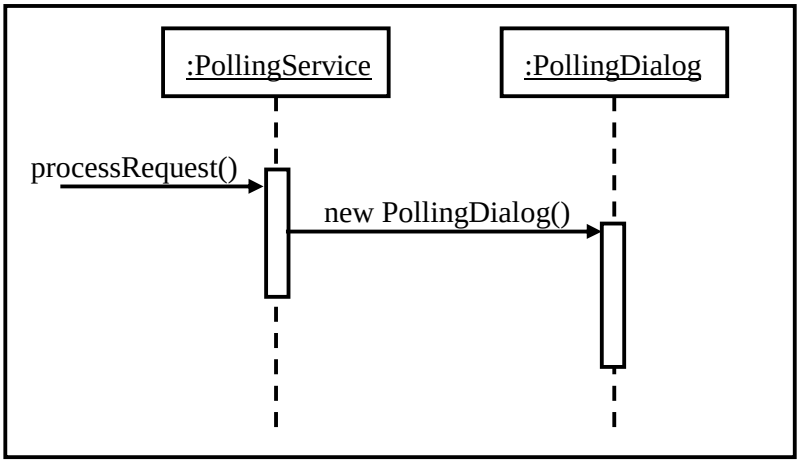


圖 4.8.2.2 其它參與者建立投票

在公開投票結果的條件下，參與者投票後或投票時間終了後，將送出投票結果請求至每位參與者，如果不公開投票結果，則僅送至主席權限參與者，每當PollingService收到投票結果請求後，便呼叫自己的parseResult()方法，產生一Ticket物件，在收到投票起始請求時，PollingService會向Conference取得目前視訊會議的參與者人數，存於一變數，每收到投票結果請求，該變數減一直到為零時，也就是收到最後一份投票結果請求就產生投票結果視窗界面。

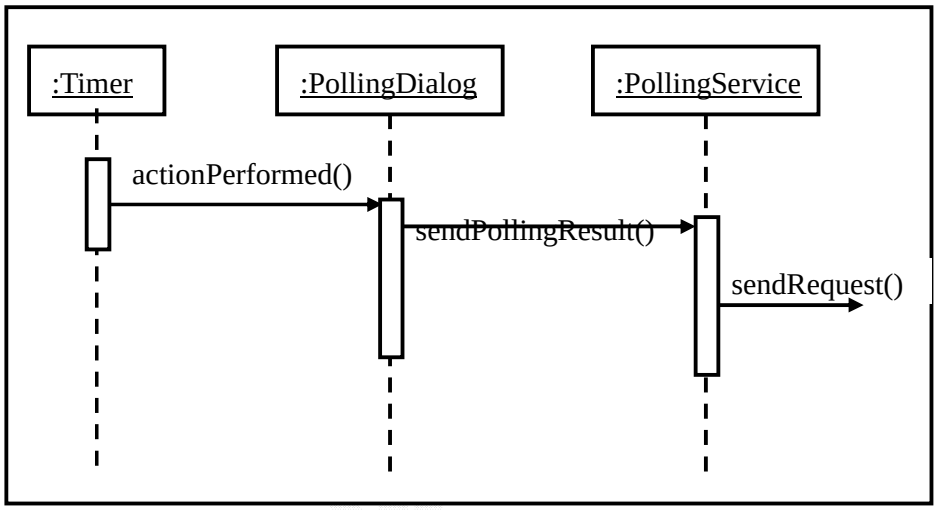


圖 4.8.2.3 投票時間終了

第五章 實驗結果比較

目前較多人使用的軟體有 ICQ、MSN Messenger(包含 NetMeeting)、Yahoo Messenger 這三大軟體，首先比較一下這幾套軟體跟本專題的主要差異。

分別從文字傳輸、網路電話、檔案傳輸、影像傳輸這四點討論起，因為這是最常使用到的基本功能。

- **文字傳輸**：這是最基本的功能，每種通訊軟體也都一定會有。

- **網路電話**：近年來 VoIP 因為它的低成本所以逐漸流行，因此每一套軟體也都將它列為必要的功能之一。

- **檔案傳輸**：點對點網路架構可以帶來資源的快速交換，所以每一套軟體皆有把這個功能放入，其中又以 ICQ 的檔案傳輸能力最強，不但可以同時傳送多個檔案而且支援序傳，再加上還可以控制流速，比起另外兩套不支援序傳又不能控制流速的軟體在這項功能強大多了，因此本專題是參考 ICQ 的模式來完成檔案傳輸的功能。

- **視訊傳輸**：由於寬頻網路的逐漸盛行，需要高頻寬的視訊傳輸需求也越來越多，除了 ICQ 之外的另外兩套軟體以及本專題皆有把這個功能放進去，不過其中 Windows Messenger 的視訊傳輸無法在一個多人的視訊會議中同時開啟多個影像視窗。Yahoo Messenger 在傳輸的時候頻寬要求十分嚴格，否則看起來就只是像是連續的照片。而本專題則是擷取這兩種軟體的優點、排除缺點，在一個多人的視訊會議中可以同時開啟多個視訊視窗，且品質可讓使用者依照當時的情況自己選擇。

再來更進一步的討論一些視訊會議可能會用的到的功能。

- **電子白板**：ICQ 本來就不是針對視訊會議所開發，所以在這方面的所有功能皆為空白，而其他的軟體以及本專題都有提供這個功能，讓使用者可以更清楚的表達自己的意見。

- **多人混音**：在這個部分，只有本專題有做，它可以提供多名使用者同時講話的功能，不像其他的軟體，一次只能有一個人講話。

- **會議秘書**：這個功能可以讓使用者立即把會議中的影音及文字資料即時紀錄到電腦中，目前其他的軟體都沒有做這個功能，但我們覺得這是一個很好用的功能，所以將它放入我們的專題中。

- **投票系統**：目前其他的軟體也沒有做這個功能，但是會議中多少會使用到投票表決，所以我們將它放入我們的專題中，提供給使用者一個快速、準確、有效率的投票功能

其他的功能還有即時變更語系以及多媒體串流留言。

- **即時變更語系**：這是一個國際化的功能，讓使用者只要按一個鍵就可以將介面完全切換成另一種語言，不過本專題目前只有提供英文跟繁體中文語系。
- **多媒體串流留言**：一般的留言都只能留純文字模式，且只有單向的留言，而本專題所做的是雙向的多媒體串流留言，使用者可以留言給要找他的人，而要找他的人在看到留言後可以留言給使用者，當然留言的模式除了最基本的純文字之外還包括了語音與視訊。

另外，由於本專題是由純 JAVA 寫成的，所以可以很容易的在各種平台上安裝、使用，不需要在另外做大幅度的更改程式碼。

綜合以上所討論的內容，可以整理成下表。

軟體 / 功能	ICQ	Windows Messenger	Yahoo Messenger	JEMI
文字傳輸	◎	◎	◎	◎
網路電話	◎	◎	◎	◎
檔案傳輸	◎	◎	◎	◎
視訊傳輸	—	◎	◎	◎
電子白板	—	◎	◎	◎
多人混音	—	—	—	◎
即時變更語系	—	—	—	◎
會議秘書	—	—	—	◎
多媒體留言	—	—	—	◎
投票系統	—	—	—	◎
跨平台	—	—	—	◎

第六章 操作手冊

本專題為一套多功能的應用軟體，以下有整個專題的介面外觀以及功能使用介紹。

6.1 主畫面

這是使用者最初的外觀。

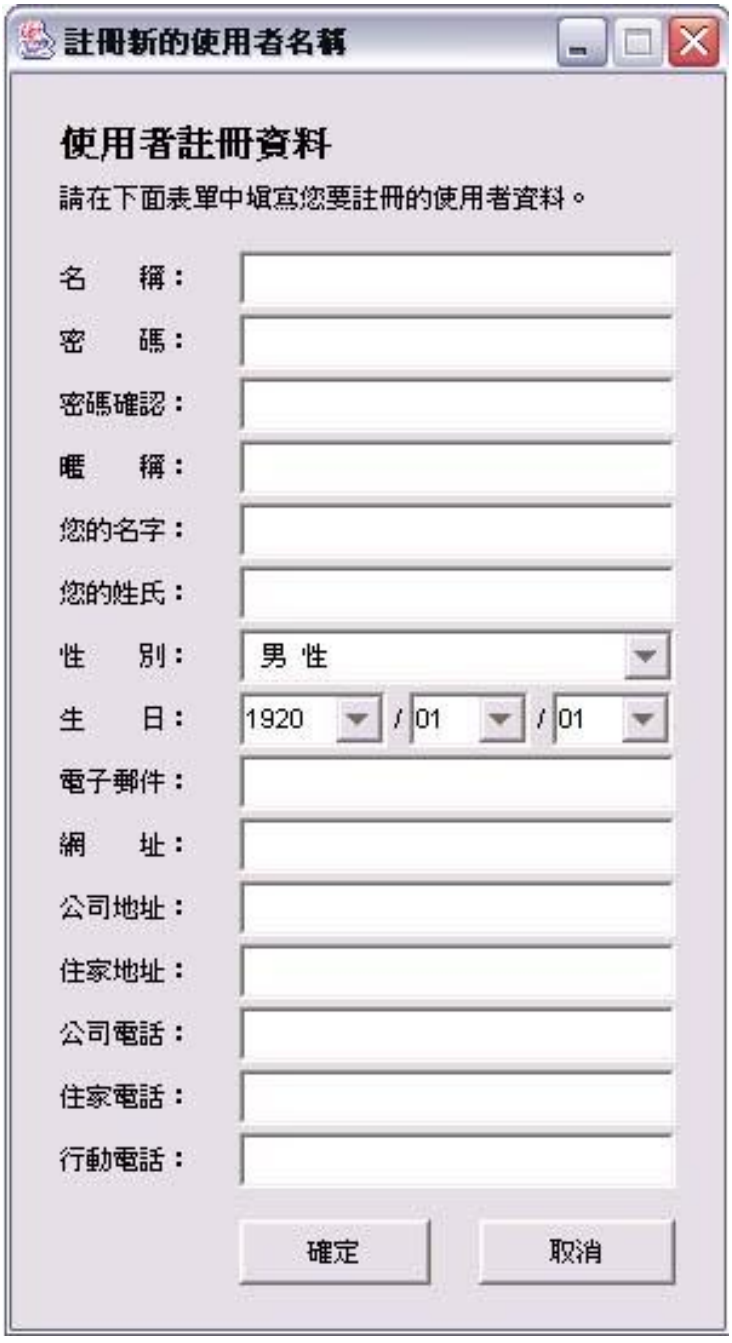
由於使用者尚未註冊及增加聯絡人，所以尚無聯絡人名單，至於如何註冊以及加入好友名單，之後會有詳細介紹。

6.2 註冊

一般使用者剛開始不會有帳號，所以要使用本專題必須先經過一個註冊的步驟，其註冊介面及方法如下。

- 在上方 Menu Bar 的部分選擇 ***網路(N)*** 後選擇 ***使用者登入(I)*** 就可以看到如下的視窗，或者直接按快速鍵 ***Ctrl+Shift+C*** 也可以有相同的效果。

- 在上半部按下 ***使用者註冊*** 則會跳出如下的視窗，輸入資料且按下確定鈕後，稍待片刻即可完成註冊程序(請先確定已連上網際網路)。



6.2 搜尋及增加聯絡人

首先在上方 Menu Bar 的部分選擇 ***連絡(C)*** 後選擇 ***新增連絡人(A)***

則會出現如下圖所示的搜尋視窗。

本專題一共提供四種搜尋模式，這裡就示範性的介紹其中最常用到的一種，依照使用者帳號搜尋，下圖以搜尋帳號為 skl 的當作例子。

當按下下一步之後則會出現顯示搜尋結果的視窗，左邊有個 List 因為這次是利用使用者帳號來搜尋，不會有重複的情況，所以只有一個結果；如果是利用 性別及年齡 來搜尋，則右方 List 會顯示所搜尋到所有符合條件使用者的帳號，而右半邊則顯示左邊 List 所選擇到使用者的詳細資訊。

6.3 會議

在會議的部分，本專題提供了純文字、語音、影像、電子白板、投票、會議秘書…等服務，其使用方式及介面如下。

- 會議的建立方法，在好友列表中按右鍵，選擇 ***邀請使用者(H)*** 即可，另外邀請者即是會議主席。

接著則雙方會出現以下的視窗，邀請方會跳出一個等候回應的視窗，而接受要請的使用者則會跳出一個接受邀請的視窗。

至於如果需要邀請多人參與會議，方法與上面一模一樣。

- 會議初始化的介面。如圖所示，使用者可以在左上方的框框看到所有參與會議的人以及主席。右上方是純文字的對話框。下方是本地端文字輸入的地方。兩旁的視訊視窗是要當使用者有提供視訊時才會出現。至於上方工具列則是其他服務的按鈕。

- 電子白板的介面如下，至於功能則不再贅述。

- 投票的介面如下，首先先由主席建立一個投票。

當主席建立完投票後所有參與會議的使用者接會收到如下的視窗。

選擇完並寫等待所有使用者皆投完票後(如果逾時未投票，則算廢票)即會開票，開票顯示結果如下圖所示。

- 會議秘書的介面如下，左上有目前有提供視訊的所有使用者，選擇使用者後按下 > 鍵即可將該使用者移動到欲錄製視訊的部分(同理 < 鍵可以將使用者從欲錄製視訊的部分移除)，再來選擇檔案的儲存路徑，從下方的 ... 按鈕按一下即會跳出檔案目錄選擇視窗，選好路徑後按下確定即可，

6.4 Mail 介面

Mail 留信息的方式有兩種，第一種是如下圖所示，直接在該使用者 ID 上按右鍵，然後選擇多媒體留言，並且選擇留言的種類即可留言。

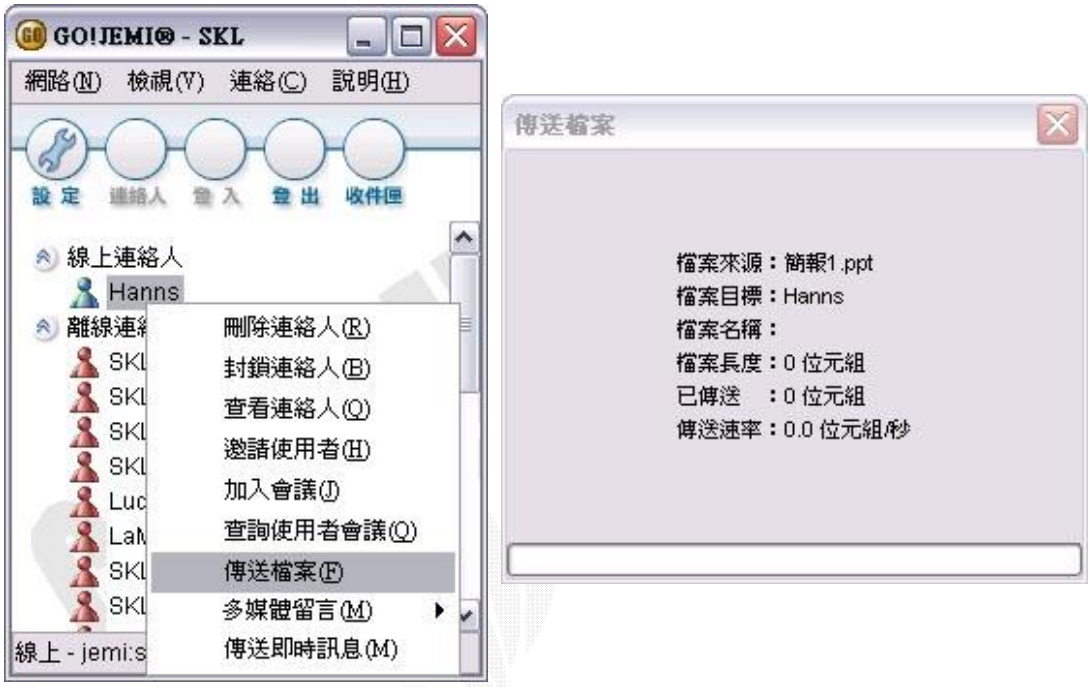
另外一種方式是如下圖所示先由收件匣按右鍵，然後選擇 ***新增*** 之後會跳出一個視窗(下圖是以影像留言為例子)，最後再選擇連絡人即可。

下圖為收到的影像留言畫面。

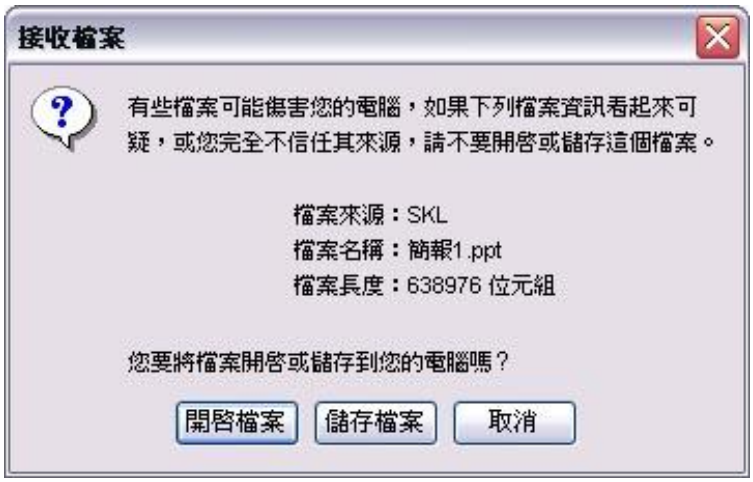
下圖為收到的影像留言畫面，因為此範例用的是標準 mpeg1 格式傳輸所以看起來畫質很不錯，不過相對所要求的頻寬也較大(約需要 15K Byte/s)，如果用一般的視訊裝置來留影像，若是本身頻寬不夠也可以選擇較低要求的格式來做傳輸。

6.5 檔案傳送

檔案傳送必須在欲傳送的連絡人 ID 上按右鍵，選擇 ***傳送檔案(F)***，此時會跳出一個檔案選擇視窗，當選擇好檔案且按下確定鍵後則會出現如下圖右的視窗，待對方確定接收後即開始傳送，該視窗下方的 Bar 可顯示傳送的進度。



而接收方會出現如下圖所示的視窗，選擇完畢後即開始傳送。



以上就是本專題簡易的操作手冊，其實本專題介面並沒有艱深的字彙，且盡量用淺顯易懂的文字來表示，所以使用者只要稍加摸索，應該不會有操作上的困難。

第七章 貢 獻

1. 完成 SIP,SDP,SOCK4,SOCK5,SIP Auth,RTP 等協定的實作。
2. 實作協定規格依國際 Java 組織 JCP 格式，如 Jain Sip ,Jain SDP 等，並取得國際認證中。
3. P2P Service 架構,依 Java JXTA 為基礎來發展,完成完整的 P2P 架構,可依實作 JEMI Service Protocol 來完成各種 P2P 的 Service 服務。
4. 解決 NAT 環境下的限制，完成 Socks-Server，SIP-Proxy Server,JEMI-Server.
5. 完成多人即時混音，和影像，並提出新的多媒體串流架構，以簡化使用者的會議的流程和多媒體串流個別處理。
6. 完成多人會議的建立和結合，包含多人語音和視訊的結合，且不限制使用者的編碼格式。
7. 新增多媒體留言系統，使使用者的訊息保留不受時間和地點的限制。
8. 完全個人化的設計，可自行設定語系、主題、增加新的 Service。
9. 透過 Web 介面來安裝及更新，並以 Java 技術為基礎，可以自動安裝和維護。
10. 及時性的多人電子白板，方便使用者之間快速的圖形方式交談。
11. OpenSource 概念，加入 SourceForge 組織 Project，全世界的人一起以 JEMI 架構來開發 P2P Service.
12. 近 600 類別，超過 120,000 行程式碼，開發時間歷時 2 年。
13. 加入個人秘書的設計，管理及協助使用者記錄其各種交談內容。

第八章 感 言

人們說:『人因夢想而偉大』,在人各年齡層都有屬於自己的夢想或理想,而才剛成年的我們,眼前有著無限的可能,也因為初識人生,不懂得可能橫在眼前的困難,夢也就更異想天間,不過最瘋狂的莫過於真的相信終有一天會美夢成真而去實現的人,幸運的是,我們就是一群這樣志同道合的人。

我們在心中一直有這樣的夢想,夢想有一天,我們的朋友將可以用我們的軟體,分享彼此的生活,我們的軟體可以放在網路上免費讓人們使用,而且十分滿意其效能,我想這是對我們最大的肯定,所以我們開了無數次會議,一遍又一遍的修正,希望能朝向目標前進,所以我們在三十度的室溫下,集合在一個小房間內寫程式,在夢想開始耕地播種的那個暑假,四個人除了吃飯,幾乎就是待在房間裡,邊討論邊寫程式,寫好一小部份就除錯,有問題了再討論再寫程式,像是掉入無窮迴圈啊,有時候真的好恨,為了心目中的完美,不得不退回幾個星期前的進度,重新來過,那時候,四個人真的想放棄、想妥協,但或許是我們對事情總抱著樂觀的態度,也就咬緊牙根堅持下去,也因此讓我們見到了不少美麗的日出,縱然橫在我們眼前的是時間和技術困難程度的不成比例,不過我們依然相信總有一天,我們的夢想將會發芽、開花、結果。

所謂有付出就有收穫,在這個專題中,我們真的學到不少課本的學術研究所學不到的經驗,不論是心理上或是技術上,

1. 增進 Java 2 Standard Edition 的熟練度。
2. 增進 Java 語言的 Design Patten 的了解。
3. 對 Java 的熱門開發工具 Jbuilder 也駕輕就熟。
4. 習慣面對英文的國外研究報告和國際標準。
5. 了解如何與同學間如何一起工作的方法及態度。

最重要的是我們學會了如何面對挑戰,如何自己想出辦法去解決問題,及達成一項不可能任務的過程中,會有多痛苦,其成功的果實又會有多甜美,真的很感謝這項專題,讓我們成長許多。

參考文獻

[1] M. Handley, H. Schulzrinne, E. Schooler, J. Rosenberg, “Session Initiation Protocol〔SIP〕”, RFC2543, March 1999.

[2] S. Petrack, L. Conroy, “The PINT Service Protocol: Extensions to SIP and SDP for IP Access to Telephone Call Services”, RFC2848, June 2000.

[3] M. Handley , V. Jacobson, “Session Description Protocol〔SDP〕”, RFC2327, April 1998.

[4] Audio-Video Transport Working Group, H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications”, RFC1889, January 1996.

[5] Ying-Da Lee, “SOCKS Version 4 Protocol”

[6] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, L. Jones, “SOCKS Protocol Version 5”, RFC1928, March 1996.

[7] M. Leech, “Username/Password Authentication for SOCKS V5”, RFC1929, March 1996.

[8] P. McMahon, “GSS-API Authentication Method for SOCKS Version 5”, RFC1961, June 1996.

[9] Mohamed Abdelaziz, Nicolas Adment, Akhil Arora, William R. Bauer, Daniel Brookshier, Mike Duigou, Carl Haywood, Jean-Christophe Hugly, Kuldip Singh Pabla, Eric Pouyoul, Gerry Seidman, Juan Carlos Soto, James Todd, Bernard Traversat, Bill Yeager, Oiching Yeung, Lauren Zuravleff, “JXTA v2.0 Protocols Specification”, February 27, 2003.

[10] British Telecom, Cisco Systems, Dynamicsoft, Inc., Ericsson Inc., Open Cloud, Sun Microsystems, Inc., Telcordia Technologies, Inc., Ulticom, “JAIN™ SIP API Specification 1.0”, JSR32, 23 Aug, 2001.

[11] Accenture, Dynamicsoft, Inc., Ericsson Inc., Nord, James, Ranganathan, Mudumbai, Sun Microsystems, Inc., “JAIN™ SDP API Public Review 2”, JSR141, 15 Mar, 2003.

[12] British Telecom, IBM, Motorola, Nokia Networks, Nortel Networks, Oracle, Sun Microsystems, Inc., Telcordia Technologies, Inc., Ulticom, “JAIN™ JCC Specification”, JSR21, 30 Jul, 2002.

[13] Sun Microsystems, Inc. “Java™ Media Framework API Programmer's Guide”, 29 Aug, 2002.

[14] Sun Microsystems, Inc. “Java™ Web Start v1.2 FCS”.