第五章 整合 GUI 介面

本章的主要目的為介紹本論文所實作的範例程式如何和 GUI 介面整合，其中包括了實作介面介紹、主要的 GUI 類別，以及最後的整合實作。

5.1 實作介面介紹

本論文實作的介面包含幾個主要部份，其中包含以下五大項目：

1. 登入畫面

如圖 5.1 所示，登入畫面包含 UserName 和 Password 兩個欄位，其中 UserName 的欄位格式為 ID@伺服器位址，主要功能為讓上線的使用者登入 SIP 伺服器，當輸入完成即可按「Login」鍵向 SIP 伺服器傳送 REGISTER 訊息。

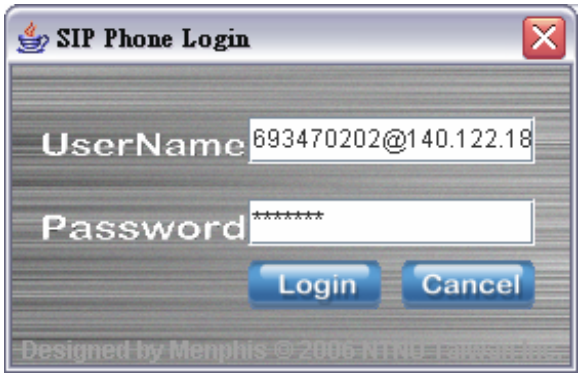

圖 5.1 登入畫面

2. 呼叫欄位

如圖 5.2 的紅色方框所示，當需要呼叫聯絡人加入語音互動會議時，只需在呼叫欄位裡填入聯絡人的 ID，即可按下「Call」按鍵發送訊息，此時網路電話便會傳送 INVITE 訊息給聯絡人。

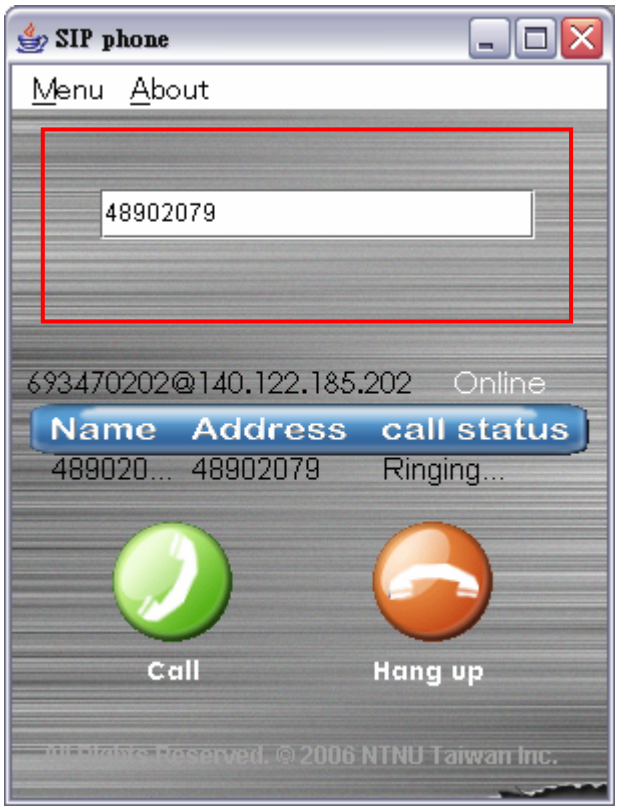

圖 5.2 呼叫欄位

3. 呼叫鍵

如圖 5.3 的紅色方框所示，綠色「call」按鍵如同一般手機裝置的撥叫鍵，當使用者為呼叫端時，「call」會將 INVITE 訊息傳送到被呼叫端，而當使用者為被呼叫端時，按下綠色「call」按鍵代表被呼叫端接受邀請，此時網路電話裝置會回傳 OK 訊息給呼叫端。

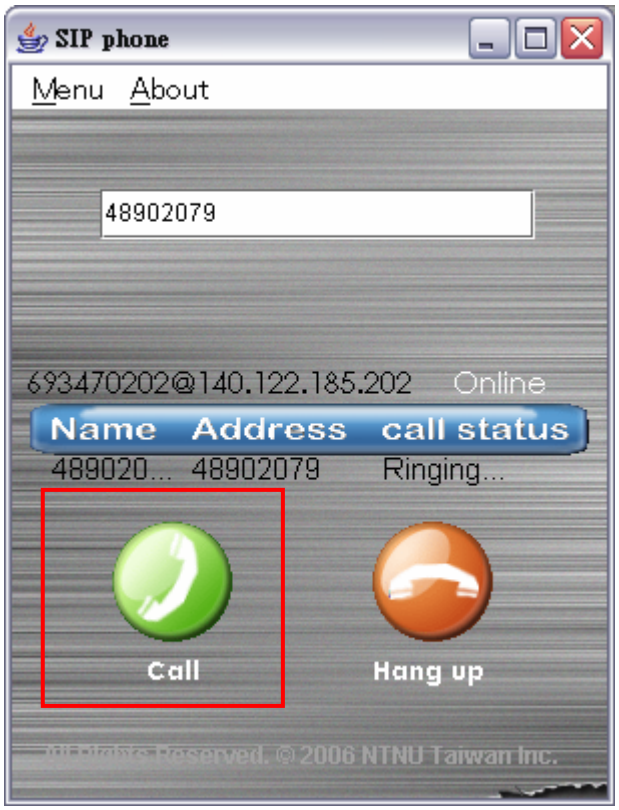

圖 5.3 呼叫鍵

4. 掛斷鍵

如圖 5.4 的紅色方框所示，紅色「Hang up」鍵如同一般手機裝置的掛斷鍵，使用者可用它結束連線中的會議，此時網路電話裝置會傳送 BYE 訊息給交談中的聯絡人。

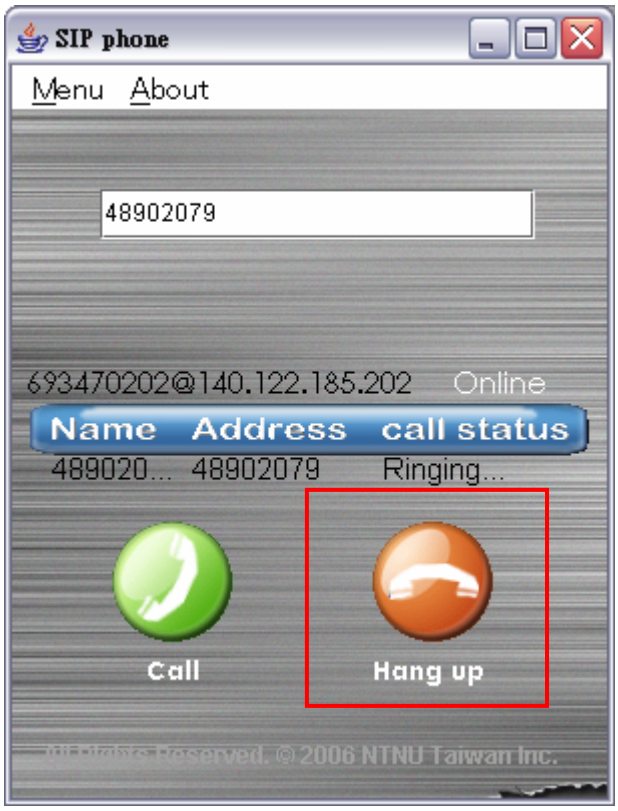

圖 5.4 掛斷鍵

5. 連線狀態顯示

如圖 5.5 所示，藍色橫條之上顯示的是目前使用者的註冊狀態，包括使用者 ID、位址以及登入狀態，而藍色橫條之下顯示的則是被呼叫端的使用者資訊，包括被呼叫端的 ID、位址以及目前的連線狀態，例如 Dialing、Ringing、Connected 等。

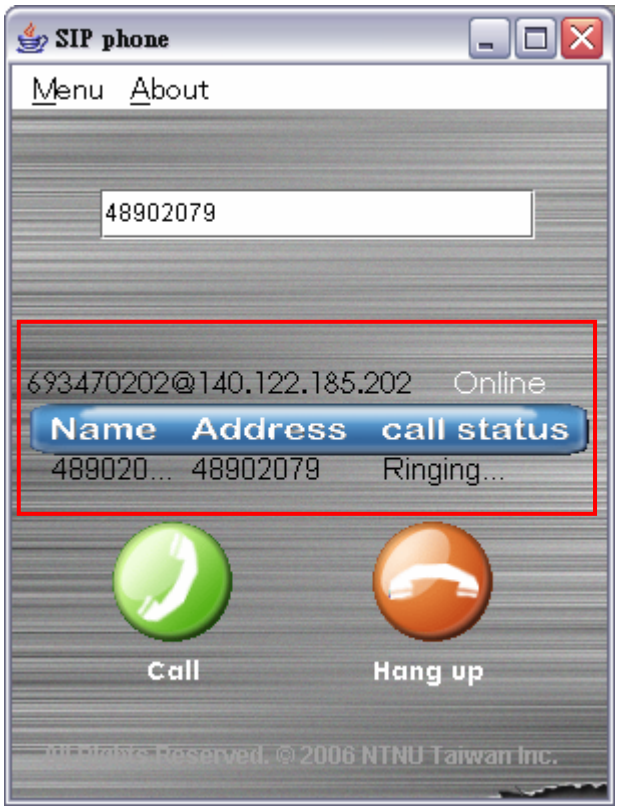

圖 5.5 連線狀態

5.2 GUI 程式類別

在我們所實作的 GUI 程式裡，主要分成三大部分，以下是這三個 class 的介紹：

1. main.java

位於 gui.main 的 package 裡，是整個網路電話程式的起始點，確保程式初始時的狀態，主要的功能為控制其他物件，例如登入視窗或目前的連線狀態顯示等。

2. login.java

位於 gui.login 的 package 裡，主要功能為初始、控制圖 5.1 的登入畫面。

3. ui.java

位於 gui.ui 的 package 裡，主要功能為初始定義整個 GUI 介面，包括欄位建立、呼叫按鍵位置的配置等。

5.3 整合 GUI 介面與網路電話程式

本論文實作之網路電話其中的 GUI 介面是由台灣師範大學資教系學生蘇于瑄、顏妙純、陳冠宇以及賴鵬羽四人協助完成的，其中整合至網路電話的部份依照 5.2 的五個主要部份分列如下。

1. 連線狀態顯示

連線狀態主要定義在 ui.java 程式裡，其中提供了幾個主要的 method，以供我們使用，以下是這些 method 的介紹：

a. print_login()

印出使用者的 ID 和位址，並且顯示目前為 online 狀態，印出的文字將顯示在圖 5.5 的藍色橫條之上。

b. print_logout()

印出使用者的 ID 和位址，並且顯示目前為 logout 狀態，印出的文字將顯示在圖 5.5 的藍色橫條之上。

c. print_name_and_address()

印出聯絡人的 ID 和位址，印出的文字將顯示在圖 5.5 藍色橫條之下的 ID 和位址欄位。

d. print_dialing()

印出目前的連線狀態為呼叫中，印出的文字將顯示在圖 5.5 藍色橫條之下的 call Status 欄位裡，並且顯示"Dialing... "字樣。

e. print_connent()

印出目前的連線狀態為已連線，印出的文字將顯示在圖 5.5 藍色橫條之下的 call Status 欄位裡，並且顯示"Connected..."字樣。

f. print_disconnent()

印出目前的連線狀態為連線中斷，印出的文字將顯示在圖 5.5 藍色橫條之下的 call Status 欄位裡，並且顯示"Disconnected..."字樣。

g. print_gotCall()

印出目前的連線狀態為對方來電中，印出的文字將顯示在圖 5.5 藍色橫條之下的 call Status 欄位裡，並且顯示"Ringing..."字樣。

2. 登入畫面

登入介面的目的是向 SIP 伺服器註冊，如圖 5.1 所示，當使用者鍵入 UserName 和 Password 並按下「login」鍵，則我們實作的網路電話便需傳送 REGISTER 訊息給 SIP 伺服器，在 5.2 節裡有提到，gui.login package 裡的 login.java 程式主要

功能為控制登入畫面，而此程式會將 UserName 和 Password 分別儲存在 get_username 和 ger_password 參數裡，以供 gui.main package 裡的 main.java 使用，當使用者按下「login」鍵時，程式便會呼叫 main.java 裡的 login_get_username() method，以用來處理登入動作，在 main.java 程式裡則會呼叫實作在主程式 JainSipPhone 的 register()，以下為此 login_get_username() method 的原始程式碼：

main.java

```
public static void login_get_username(){
//將 login 儲存的 get_username 參數傳遞給 main 程式裡的 username 參數
        username = login.get_username;
//將 login 儲存的 get_password 參數傳遞給 main 程式裡的 password 參數
        password = login.get_password;
//呼叫 JainSipPhone 程式提供的 register() method，呼叫時必須給定 port number
//(5060)和 transport (UDP)
        JainSipPhone.register ( 5060, "UDP")
        }
```

在 JainSipPhone.java 程式裡的 register() method 主要目的是建立並傳送 REGISTER 訊息給 SIP 伺服器，以下為 register() method 的原始程式碼：

JainSipPhone.java

```
public void register(int registrarPort, String registrarTransport)
        throws InvalidArgumentException{
//程式一開始必須先獲得使用者輸入的 username，並記錄在 User 參數裡，
//username 的格式為 ID@伺服器位址
        String User = main.username;
//接下來開始分離字串中的 ID 和伺服器位址，首先先取得@符號在字串裡的位
//置
        int mouse = User.indexOf('@');
//使用 substring() method 獲得使用者 ID，並記錄在 fromUser 參數裡
        fromUser = User.substring(0,mouse);
//使用 substring() method 獲得伺服器位址，並記錄在 registrarAddress 參數裡
        registrarAddress = User.substring(mouse+1);
//設定使用者的 display name，在這裡我們將它設成使用者 ID (fromUser)
        fromDisplayName = fromUser;
//獲得 main.java 程式裡的 password 參數，並記錄在 password 參數裡
        password = main.password;
//開始建立 REGISTER 訊息，其中 REGISTER 訊息的檔頭包括有 From Header、
//To Header、RequestURI、Call ID Header、Cseq Header、Via Header、Max Forwards
//Header 和 Expire Header，以下先介紹建立這些檔頭的方法
```

```
        try{
//建立 requestURI 物件，建立時需給定 SIP 伺服器位址(registrarAddress)，例如
//192.168.1.1，使用者名稱則不需給定
        SipURI requestURI =
            addressFactory.createSipURI(null,registrarAddress);
//設定註冊時使用的 port number，例如 5060
        requestURI.setPort(registrarPort);
//建立新的 CallIdHeader，CallIdHeader 可從 SipProvider 物件的 getNewCallID()
//method 獲得
        CallIdHeader callIdHeader = SipProvider.getNewCallId();
//建立新的 CseqHeader
        //CseqHeader
        CseqHeader cSeq = headerFactory.createCSeqHeader(1,
                Request.REGISTER);
//在 REGISTER 訊息裡的 FromHeader 記錄了使用者 ID 和註冊位址，首先使用
//AddressFactory 的 createSipURI() method 建立 SipURI 物件，建立時必須給定
//呼叫端使用者名稱(fromUser)以及註冊位址(registrarAddress)
        SipURI fromURI =
            (SipURI)addressFactory.createSipURI(fromUser,
```

registrarAddress);

//使用 addressFactory 的 createAddress() method 建立 Address 物件，建立時必須給

//定之前所建立的 SipURI (fromURI)物件

Address fromAddress = addressFactory.createAddress(fromURI);

//Address 物件和 SipURI 物件的不同點在於 Address 物件包含了使用者的顯示名

//稱(DisplayName)，在這裡利用 Address 物件的 setDisplayName() method 設定

//Display Name

fromAddress.setDisplayName(fromDisplayName);

//有了 Address 物件(fromAddress)之後，我們便可藉由 HeaderFactory 的

//createFromHeader() method 建立 FromHeader 物件，代表呼叫端的 tag 參數則藉

//由 hashCode() method 亂數產生

fromHeader = headerFactory.createFromHeader(fromAddress,

Integer.toString(hashCode()));

//建立 ToHeader，REGISTER 訊息中 ToHeader 的位址和 fromHeader 相同，只有

//代表接收端的 to tag 先設成 null

ToHeader toHeader = headerFactory.createToHeader(fromAddress,

null);

//使用 headerFactory 的 createViaHeader() method 建立 ViaHeader 物件，建立時必

//須給定本地的 IP 位址 (ipAddress)、port number (listeningPort)、使用的傳輸協

//定(transport)，branch 會在日後由系統自動設定，在這裡我們先設為 null。

ViaHeader viaHeader =

headerFactory.createViaHeader(ipAddress, listeningPort,

transport, null);

//使用 HeaderFactory 的 createMaxForwardsHeader() method 建立

//MaxForwardsHeader，在這裡建議值為 70

MaxForwardsHeader maxForwards =

headerFactory.createMaxForwardsHeader(70);

//有了以上建立的檔頭之後，我們便可以建立 REGISTER 訊息

Request registerRequest = messageFactory.createRequest(requestURI,

Request.REGISTER,

callIdHeader,

cSeq, fromHeader, toHeader,

viaHeaders,

maxForwardsHeader);

//建立重要的 ExpiresHeader，ExpiresHeader 的主要功能為告知伺服器使用者要求

//的停留時間，其最大值為 3600 秒

ExpiresHeader expHeader = headerFactory.createExpiresHeader(

3600);

```
//將 ExpiresHeader 加入 REGISTER 訊息裡
            registerRequest.addHeader(expHeader);
//建立用來傳送 REGISTER 訊息的 ClientTransaction
            ClientTransaction    regTrans
                =udpProvider.getNewClientTransaction(registerRequest);
//將此 REGISTER 訊息傳送到網路上
            regTrans.sendRequest();
            }catch (Exception ex) {
                System.out.println(ex.getMessage());
                ex.printStackTrace();
                usage();
                }
        }
```

---

3. 呼叫欄位

呼叫欄位的主要目的是讓使用者填入聯絡人的 ID，並儲存在系統裡以供傳送 INVITE 訊息時使用，呼叫欄位定義在 ui.java 程式裡，如圖 5.6 所示，在 ui.java 的程式裡建立了 JTextField 物件代表供聯絡人欄位，在這個物件裡，我們可以使用 Swing API 提供的 getText() method 獲得使用者所填入的文字。並儲存在

main.java 程式裡的 remotename 物件裡，remotename 物件可供傳送 INVITE 訊息時使用，我們將會在第四部份呼叫鍵做介紹。

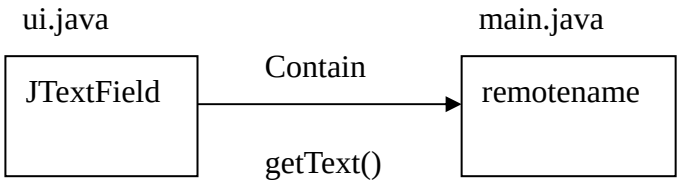

圖 5.6 遠端使用者欄位

其定義的原始程式碼如下：

Ui.java

//新增 JTextField 物件，其初始的名稱為 jTextFiled1

static JTextField jTextField1 = new JTextField();

//獲得聯絡人名稱，並儲存在 main.java 程式裡的 remoteuser 物件裡

main.remoteuser = jTextField1.getText();

---

4. 呼叫鍵

呼叫鍵的主要功能為撥打或接聽電話，方便使用者建立會議，呼叫鍵定義在 main.java 的類別裡，當使用者在第二項的呼叫欄位裡填入聯絡人 ID 後，並按下「Call」綠色按鍵，此時便會呼叫 main 類別裡的 dialing() method，如圖 5.7 所示，在 dialing() method 裡程式會判斷使用者是呼叫端還是被呼叫端，並呼叫定義在 JainSipPhone 相對應的 method，以下為 dialing() method 的原始程式碼：

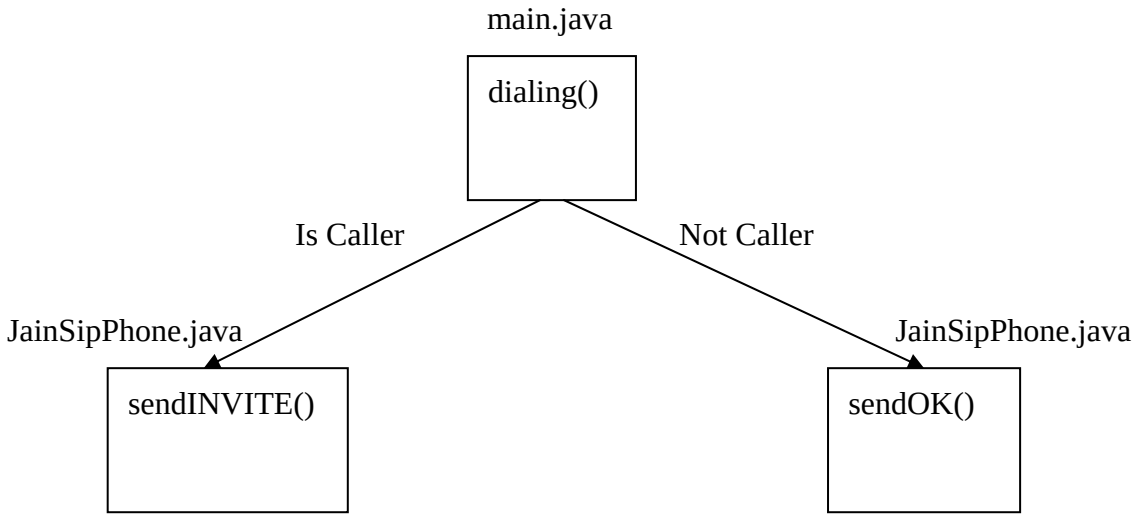

圖 5.7 呼叫鍵程式流程圖

main.java

```
//dialing() method，呼叫時必須傳入 JainSipPhone 和 ui 物件，以使用所提供的
//method
    public static void dialing(JainSipPhone jainSipPhone, ui text){
//判斷是否為呼叫端
            if (jainSipPhone.isCaller==true){
//若為呼叫端則呼叫 jainSipPhone 所實作的 senINVITE() method
              jainSipPhone.sendINVITE();
//使用 ui 程式的 print_dialing() method 印出目前狀態為呼叫中
            text.print_dialing();
            }
            else{
```

//若為被呼叫端則呼叫 jainSipPhone 所實作的 sendOK() method

jainSipPhone.sendOK();

}

}

程式在 dialing() method 裡呼叫了兩個定義在主程式 JainSipPhone 的 method，以下先介紹這兩種 method：

a. sendINVITE() method

在第三章的 3.4.3 UAC Create and Send INVITE Request 裡介紹了如何建立和傳送 INVITE 訊息給聯絡人的方法，我們可以將這些程式碼改寫成 sendINVITE() method，改寫的部份為將儲存在 main.java 的 remotename 參數傳遞給 INVITE 訊息，如此一來，我們才能將 INVITE 訊息正確的傳送到被呼叫端，以下是此 method 的原始程式碼：

//sendINVITE() method，呼叫時必須傳入位於 gui.main package 的 main 物件，以

//獲得聯絡人 ID

public void sendINVITE(main Main){

//將被呼叫端的使用者 ID (toUser)和對方的顯示名稱(toDisplayName)設成

//main.java 程式裡的 remotename 參數，remotename 參數可在第三部份的呼叫欄//

位獲得

toUser = Main.remotename;

toDisplayName = Main.remotename;

//以下程式碼皆和第三章的 3.4.3 相同

//FromHeader 記錄了呼叫端位址，首先使用 AddressFactory 的 createSipURI()

//method 建立 SipURI 物件，建立時必須給定呼叫端使用者名稱(fromUser)以及使

//用者位址(fromSipAddress)

SipURI fromURI = (SipURI) addressFactory.createSipURI(fromUser,

fromSipAddress);

//使用 addressFactory 的 createAddress() method 建立 Address 物件，建立時必須給

//定之前所建立的 SipURI (fromURI)物件

Address fromAddress = addressFactory.createAddress(fromURI);

//Address 物件和 SipURI 物件的不同點在於 Address 物件包含了使用者的顯示名

//稱(DisplayName)，在這裡利用 Address 物件的 setDisplayName() method 設定

//Display Name

fromAddress.setDisplayName(fromDisplayName);

//有了 Address 物件(fromAddress)之後，我們便可藉由 HeaderFactory 的

//createFromHeader() method 建立 FromHeader 物件，代表呼叫端的 tag 參數則藉

//由 hashCode() method 亂數產生

fromHeader = headerFactory.createFromHeader(fromAddress,

Integer.toString(hashCode()));

//使用 AddressFactory 的 createSipURI() method 建立 SIPURI 物件，在我們的範例

//裡因為此 INVITE 訊息需透過 Proxy 傳送，所以被呼叫端的位址為 Proxy 位址

//(registrarAddress)，並給定被呼叫端的使用者 ID (toUser)，以供 Proxy 尋找

SipURI toURI =

addressFactory.createSipURI(toUser, registrarAddress);

//使用 AddresFactory 的 createAddress() method 建立 Address 物件，並給定建立好

//的 SipURI 物件 (toURI)。

Address toAddress = addressFactory.createAddress(toURI);

//設定 Address 物件(toAddress) 的 DisplayName

toAddress.setDisplayName(toDisplayName);

//有了 Address 物件後便可使用 HeaderFactory 的 createToHeader() method 建立新

//的 ToHeader 物件，而代表接收端的 toTag 則須由接收端設定，所以我們先設成

//null

ToHeader toHeader =

headerFactory.createToHeader(toNameAddress, null);

//建立代表 request 主題的 requestURI，其中目的地位址為 Proxy 位址

//(registrarAddress)，toUser 則為被呼叫者 ID

SipURI requestURI =

addressFactory.createSipURI(toUser, registrarAddress);

// 建立 ViaHeader 物件，因為 ViaHeader 通常包含許多經過的位址資訊，所以先新增一個 ArrayList 屬於 ViaHeaders

ArrayList viaHeaders = new ArrayList();

//使用 headerFactory 的 createViaHeader() method 建立 ViaHeader 物件，建立時必須給定本地的 IP 位址 (ipAddress)、port number (listeningPort)、使用的傳輸協定(transport)，branch 會在日後由系統自動設定，在這裡我們先設為 null。

ViaHeader viaHeader =

headerFactory.createViaHeader(ipAddress, listeningPort,

transport, null);

//將新增的 ViaHeader 加入 ViaHeaders 的 ArrayList 裡

viaHeaders.add(viaHeader);

//使用 HeaderFactory 的 createCseqHeader() method 建立新的 CSeqHeader，建立時必須給定 sequence number，其初始值為 1，以及 request 的名稱 (Request.INVITE)

CSeqHeader cSeqHeader

=headerFactory.createCSeqHeader(1,Request.INVITE);

//使用 HeaderFactory 的 createMaxForwardsHeader() method 建立

//MaxForwardsHeader，在這裡建議值為 70

MaxForwardsHeader maxForwards =

headerFactory.createMaxForwardsHeader(70);

//透過 SipProvider 的 getNewCallID() method 獲得新的 CallIDHeader，使用

//SipProvider 所提供的 method 可獲得代表此 dialog 的獨一獨二 Call ID，而

//HeaderFactory 提供的 method 則需手動輸入 Call ID 字串，並且不保證獨一性

CallIdHeader callIdHeader = sipProvider.getNewCallId();

//成功建立完 INVITE 訊息所需的 Header 之後，便可利用 MessageFactory 的

//createRequest() method 建立新的 INVITE 訊息

Request request =

messageFactory.createRequest(requestURI, Request.INVITE,

callIdHeader, cSeqHeader, fromHeader, toHeader, viaHeaders,

maxForwardsHeader);

//有了 INVITE 訊息之後，接下來產生新的 ClientTransaction 物件幫助我們將訊息

//傳送到網路上，而 ClientTransaction 物件可透過 SipProvider 獲得

//藉由 SipProvider 的 getNewClientTransaction() method 獲得新的

//ClientTransaction　物件，此物件是根據所給定的 request 訊息產生的，產生後便

//無法處理其他訊息

clientTransaction = sipProvider.getNewClientTransaction(request);

```
//利用 ClientTransaction 的 sendRequest() method 將訊息傳送到網路上，達成建立
//和傳送 INVITE Request 的目的
        clientTransaction.sendRequest();
}
```

---

b. sendOK() method

在第三章的 3.4.4 UAS Receive INVITE and Send Response 裡，我們有提到當被呼叫端接收到 INVITE 訊息之後，如何建立 Response 訊息，並回傳給呼叫端，在這個範例程式裡，我們介紹了建立 180 Ringing 和 200 OK 訊息的方法，現在我們要將傳送 200 OK 訊息的方法重新定義在 sendOK() method 裡，如此一來，當被呼叫端同意參與會議並按下綠色的「Call」按鍵後，便可傳送 OK 訊息給呼叫端，告知呼叫端同意建立連線，建立 200 OK 訊息的方法如圖 5.8 所示，首先先用 MessageFactory 物件的 createResponse()建立 200 OK 訊息，並設定關於被呼叫端的 toHeader 參數，有了 200 OK 訊息之後，我們便可以使用 ServerTransaction 物件將此 Response 傳送給呼叫端，以下為重新定義傳送 200 OK 的 sendOK() method。

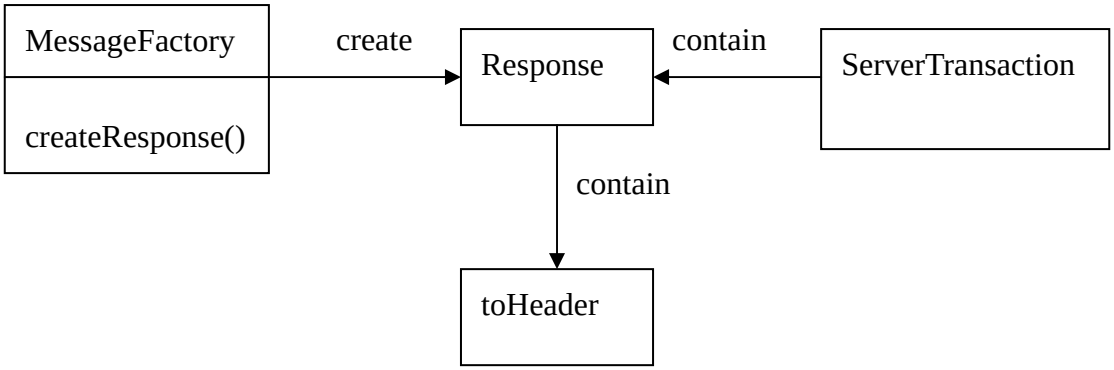

圖 5.8 建立 200 OK 訊息

JainSipPhone.java

```
public void sendOK(){
        try{
//建立 200 OK Response 訊息
            Response response = messageFactory.createResponse(200,request);
//取得此 Response 的 ToHeader 檔頭
            ToHeader toHeader =
                (ToHeader)response.getHeader(ToHeader.NAME);
//設定代表被呼叫端的 tag 參數
            toHeader.setTag("4321");
//使用 ServerTransaction 的 sendResponse() method，將此 Response 傳送到網路
//上
            serverTransaction.sendResponse(response);
        }catch(Exception e){
```

```
                e.printStackTrace();
                System.exit(0);
            }
        }
```

---

5. 掛斷鍵

掛斷鍵的主要功能為提供使用者結束連線中的會議，掛斷鍵定義在 main.java 程式裡，當使用者按下圖 5.4 的紅色「Hang Up」按鍵後，便會呼叫實作在 main.java 程式裡的 hang_up() method，hand_up() method 的功能為傳送 Bye 訊息給交談中的聯絡人，所以在 hang_up() method 裡則會呼叫位於主程式 JainSipPhone 的 sendBye() method，並且在 GUI 介面上印出”Disconnected”訊息，以下為 hand_up() method 的原始程式碼。

main.java

```
//hand_up() method，呼叫時需傳入 JainSipPhone 和 ui 物件，以使用提供的 mehod
public static void hang_up(JianSipPhone jainSipPhone, ui text){
//呼叫 JainSipPhone 裡的 sendBye() method，傳送 BYE 訊息給聯絡人
        jainSipPhone.sendBye();
//使用在第一部份提到位於 ui.java 程式裡的連線狀態 method，印出連線中
//斷 ”Disconnected”字樣
```

text.print_disconnent();

}

在 JainSipPhone.java 程式裡實作的 sendBye() method 主要目的在建立 BYE 訊息，並將它傳送到網路上，如圖 5.9 所示，我們可以使用 Dialog 物件建立一個新的 BYE 訊息，並利用 BYE 訊息建立一個新的 CleintTransaction 物件，最後則使用 Dialog 物件將此 ClientTransaction 物件傳送到網路上，以下介紹 sendBye() 的原始程式碼：

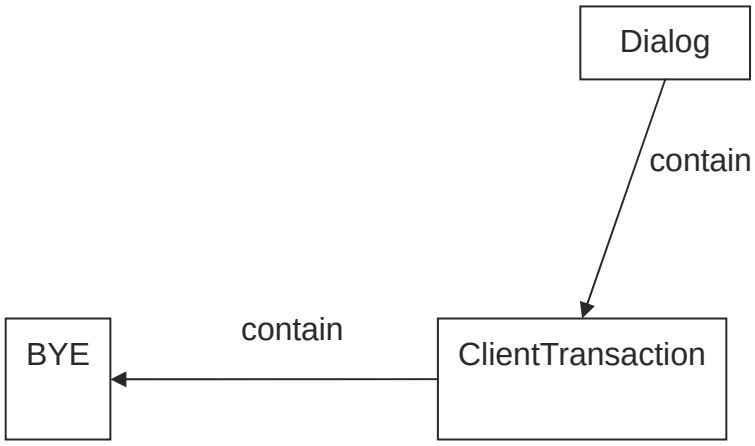

圖 5.9 建立 BYE 訊息

JainSipPhone.java

public void sendBye(){

//建立新的 BYE request 訊息

Request bye = null;

```
    try {
//使用 dialog 的 createRequest() method 建立新的 BYE request 訊息，建立時必
//須給定 Request 名稱(Request.BYE)
        bye = dialog.createRequest(Request.BYE);
//建立對應於 BYE 訊息的 ClientTransaction 物件
        ClientTransaction clientTransaction = null;
        clientTransaction =
            udpProvider.getNewClientTransaction(bye);
//利用 Dialog 物件將相對於 BYE 訊息的 CleintTransaction 物件傳送到網路上
        Dialog.sendRequest(clientTransaction);
        }
        catch (SipException ex1) {
        }
    }
```
