Object-Oriented Analysis and Design

Simple SIP

HW#5

電機碩一 104318070 林侃賦

電機碩一 104318079 何柏寬

電機碩一 104318144 周冠亨

1. Requirement Document
   1. Change History

|  |  |  |
| --- | --- | --- |
| **Revision** | **Description** | **Date** |
| **Iteration I** | | |
| 1 | Problem statement. | Mar 4, 2016 |
| 2 | Problem Statement refinement. System Context Diagram. Summary of System Features. Use Case Diagram. Use Cases. Non-functional Requirements and Constraints. Glossary. Software Environments refinement. | Mar 18, 2016 |

* 1. Problem Statement

本軟體提供網路語音及簡訊服務，主要應用於行動裝置與個人電腦上。隨著無線通訊的普及與網路對談即時性的需求提高，SSIP (Simple SIP) 相較於傳統電話，提供網內互打可免費不限時數地區撥打與伺服器功能檢測。使用者只需具備上網能力便可操作 SSIP 進行撥打電話、傳送簡訊等 …… 相關服務；SIP Server 管理員也可透過 SSIP 檢測目前伺服器與其它 RTP Relay Server 、Gateway …… 週邊設備運作形況或於發生故障時，透過 SSIP 釐清問題。

使用軟體前，須先設定帳號、密碼、伺服器網址，進行登入伺服器的操作，完成登入後便可進行撥打、傳送簡訊，撥打市話也提供 Server 管理員伺服器的功能測試選項例如：1000 個帳號同時註冊、撥打、簡訊。

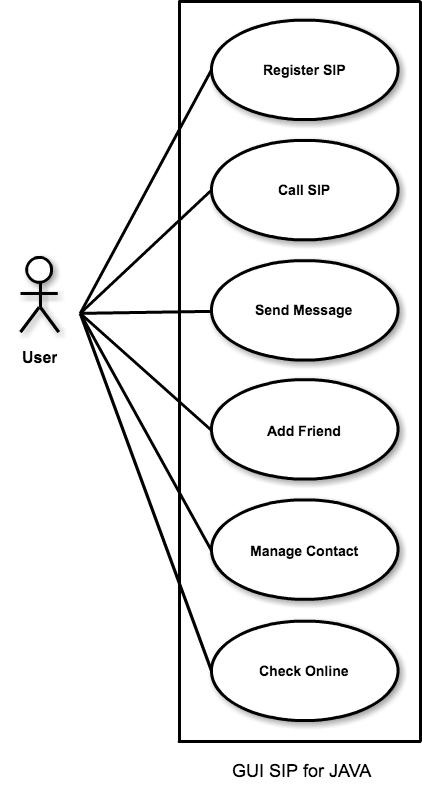
* 1. System Context Diagram



* 1. Summary of System Features

|  |  |
| --- | --- |
| Feature ID | Description |
| FEA-01 | 註冊 SIP |
| FEA-02 | 撥打 SIP 電話 |
| FEA-03 | 傳送 SIP 簡訊 |
| FEA-04 | 加入好友 |
| FEA-05 | 建立通訊錄 |
| FEA-06 | 是否上線 |

* 1. Use Case Diagram



SIP FOR JAVA

* 1. Use Cases
     1. Register SIP

|  |  |
| --- | --- |
| Scope | SIP for JAVA |
| Level | User-goal |
| Primary Actor | 使用者 |
| Stakeholders and Interests | * 使用者:可以完成註冊 |
| Preconditions | 1. 裝置具備上網能力 2. 使用者需要知道伺服器位置 3. 使用者需要知道帳號與密碼 |
| Success Guarantee | 1. 控制器註冊成功 2. 伺服器記錄註冊帳號 |
| Main Success Scenario | 1. 使用者輸入伺服器網路位置 2. 使用者填寫帳號、密碼 3. 控制器開始進行認證 4. 控制器啟用服務 |
| Extensions | 1a. 使用者輸入錯的伺服器位址   1. 控制器顯示訊息(連線錯誤) 2. Go to step 1.   4a. 連線伺服器超時   1. 控制器顯示訊息(連線超時) 2. 控制器詢問是否重新連線 3. Go to step 3.   4b. 登入失敗   1. 控制器顯示訊息(登入失敗) 2. Go to step 2. |
| Special Requirements | 使用 Domain URL 需使用 DNS 查詢伺服器 IP |
| Technology and Data  Variations List | None |
| Frequency of Occurrence | Often. |

* + 1. Call SIP

|  |  |
| --- | --- |
| Scope | SIP for JAVA |
| Level | User-goal |
| Primary Actor | 使用者 |
| Stakeholders and Interests | * 使用者:藉由撥打電話可以與其他使用者進行通訊 |
| Preconditions | 1. 裝置具備上網能力 2. 裝置需具備麥克風與喇叭 3. 使用者必須完成註冊手續 4. 對方必須完成註冊手續 5. 使用者必須知道對方SIP電話號碼 6. 使用者開啟程式並且已進入待機狀態 |
| Success Guarantee | 使用者可以與其它使SIP使用者進行通話 |
| Main Success Scenario | 1. 使用者選擇欲撥打的對象進行撥號 2. 裝置開始響鈴直到對方接通 3. 控制器開始進行通話 4. 其中一方結束通話 5. 程式回到待機畫面 |
| Extensions | 2a. 對方不在線上   1. 控制器結束通話 2. Go to step 6   3a. 裝置響鈴過時   1. 控制器結束通話 2. Go to step 6   4a. 裝置網路不穩   1. 控制器嘗試進行恢復 2. 控制器恢復成功 3. Go to step4 |
| Special Requirements | None |
| Technology and Data  Variations List | 音訊轉碼 |
| Frequency of Occurrence | Usually |

* + 1. Send Message

|  |  |
| --- | --- |
| Scope | SIP for JAVA |
| Level | User-goal |
| Primary Actor | 使用者 |
| Stakeholders and Interests | * 使用者:傳送簡訊 |
| Preconditions | 1. 裝置具備上網能力 2. 使用者必須完成註冊手續 |
| Success Guarantee | 1. 伺服器轉發簡訊成功 |
| Main Success Scenario | 1. 使用者點選簡訊頁面 2. 使用者選擇聯絡人 3. 使用者輸入簡訊內容 4. 控制器發送簡訊 5. 控制器顯示發送成功標記 6. 控制器記錄內容 7. 控制器清空輸入欄位 |
| Extensions | 2a. 對象不存在   1. 使用者新增聯絡人 2. 使用者輸入聯絡人屬性 3. 控制器完成設定 4. Go to step 2.   3a. 使用者填寫內容空白   1. 控制器清除空白 2. Go to step 3.   4a. 控制器傳送簡訊失敗   1. 控制器顯示發失敗標記 2. Go to step 7. |
| Special Requirements | None |
| Technology and Data  Variations List | CHARACTER VARYING |
| Frequency of Occurrence | Usually. |
| Open Issues | 1. 對方的帳號? |

* + 1. Add Friend

|  |  |
| --- | --- |
| Scope | SIP for JAVA |
| Level | subfunction |
| Primary Actor | 使用者 |
| Stakeholders and Interests | * 使用者:加入對方好友時，並寄送邀請給對方 |
| Preconditions | 1. 裝置具備上網能力 2. 好友也在線上 3. 開啟程式並且已進入待機狀態 |
| Success Guarantee | 1. 控制器成功加入聯絡人 |
| Main Success Scenario | 1. 使用者按下加入好友 2. 使用者輸入對方帳號 3. 使用者完成並送出邀請 4. 控制器更新聯絡人清單 |
| Extensions | 3a. 對方拒絕   1. 控制器顯示失敗訊息 2. Go to 待機畫面   3b. 對方不再線上   1. 控制器顯示失敗訊息 2. Go to 待機畫面 |
| Special Requirements | 帳號必須為數字或英文 |
| Technology and Data  Variations List | None |
| Frequency of Occurrence | Often. |
| Open Issues | None |

* + 1. Manage Contact

|  |  |
| --- | --- |
| Scope | SIP for JAVA |
| Level | User-goal |
| Primary Actor | 使用者 |
| Stakeholders and Interests | * 使用者:編輯聯絡人 |
| Preconditions | 至少有一筆聯絡人資料 |
| Success Guarantee | 1. 使用者修改成功 |
| Main Success Scenario | 1. 使用者點選聯絡人頁面 2. 使用者選擇聯絡人 3. 使用者輸入修改資料 4. 程式更新資料並刷新聯絡人頁面 |
| Extensions | 3a. 輸入資料無效   1. 控制器顯示失敗訊息 2. Go to 3 |
| Special Requirements | None |
| Technology and Data  Variations List | None |
| Frequency of Occurrence | Usually. |
| Open Issues | None |

* + 1. Check Online

|  |  |
| --- | --- |
| Scope | SIP for JAVA |
| Level | user-goal |
| Primary Actor | 使用者 |
| Stakeholders and Interests | * 使用者:查詢對方是否上線 |
| Preconditions | 1. 裝置具備上網能力 2. 使用者必須完成註冊手續 |
| Success Guarantee | 1. 伺服器回傳結果 |
| Main Success Scenario | 1. 使用者點選聯絡人頁面 2. 使用者選擇聯絡人 3. 控制器取得對方在線資訊 |
| Extensions | 2a. 對象不存在   1. 控制器新增聯絡人 2. 控制器輸入聯絡人屬性 3. 控制器完成設定 4. Go to step 2. |
| Special Requirements | 帳號必須為數字或英文 |
| Technology and Data  Variations List | None |
| Frequency of Occurrence | often |
| Open Issues | None |

* 1. Non-functional Requirements and Constraints

|  |  |  |
| --- | --- | --- |
| NFR ID | Category | Description |
| NFR-01 | Performance | 每新增聯絡人， 就要立刻更新聯絡人資料。 |
| NFR-02 | Usability | 操作流程簡單 |

* 1. Glossary

|  |  |
| --- | --- |
| Item | Definition or Description |
| SIP | SIP為Session Initiation Protocol的簡寫，中文翻譯為會話初始協議，用於建立、修改和終止影像、音訊或即時通訊之協議。 |
| IPv4 | 32位二進制數組成，爲便於使用，常以XXX.XXX.XXX.XXX形式表現 |
| DNS | 把網域名稱翻譯成IP位址(也可以逆向操作) |
| URL | 格式為 X.X.X，如：www.google.com |

* 1. Software Environments

The project will be written in JAVA programming language.

1. Domain Class Model
   1. Domain Class Diagram Showing Only Concepts
      1. Classes Identified

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Contact | Message | Success Identify | Online Information | Add Friend |
| Callee profile | Failed Message | Contact attribute | Network instability | End call |
| Stand by | microphone | Register  process | Register account | stand-by image |
| Audio | Server IP | Authenticate | Sip Call | Register |
| User | Profile | Controller | Network | Maker |
| Parser | Option | Log File |  |  |

* + 1. Bad Classes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Contact | stand-by image | Success Identify | Online Information | Add Friend |
| Callee profile | Failed Message | Contact attribute | Network instability | End call |
| Stand by | microphone | Register  process | Register account |  |

* + 1. Good Classes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| User | Profile | Controller | Network | Maker |
| Parser | Message | Log File | Option | Audio |
| Sip Call | Register |  |  |  |

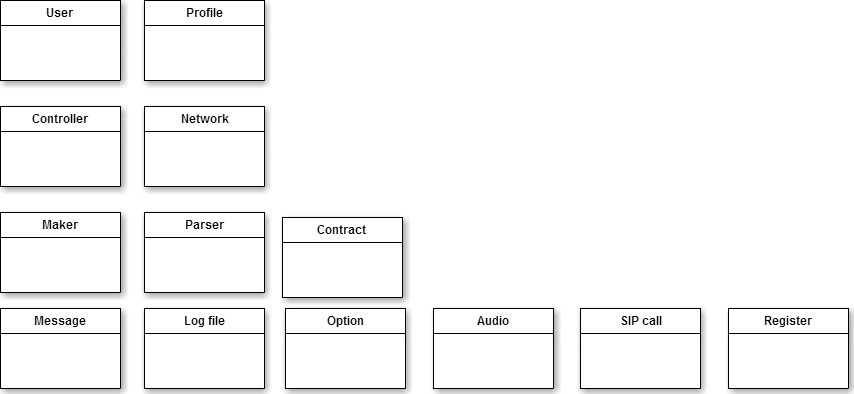


Figure 2-1 Domain Class Diagram Only Concepts

* 1. Add Associations
* User Execute Controller
* User Configure Profile
* Maker Own Profile
* Controller Go through Network
* Controller Log LogFile
* Controller Make Maker
* Controller Analyses Parser
* Controller Call SIP\_Call
* Controller Active Register\_Service
* Controller Send Option
* Controller Send Messager
* SIP\_Call Set Audio
* Audio Callback Controller

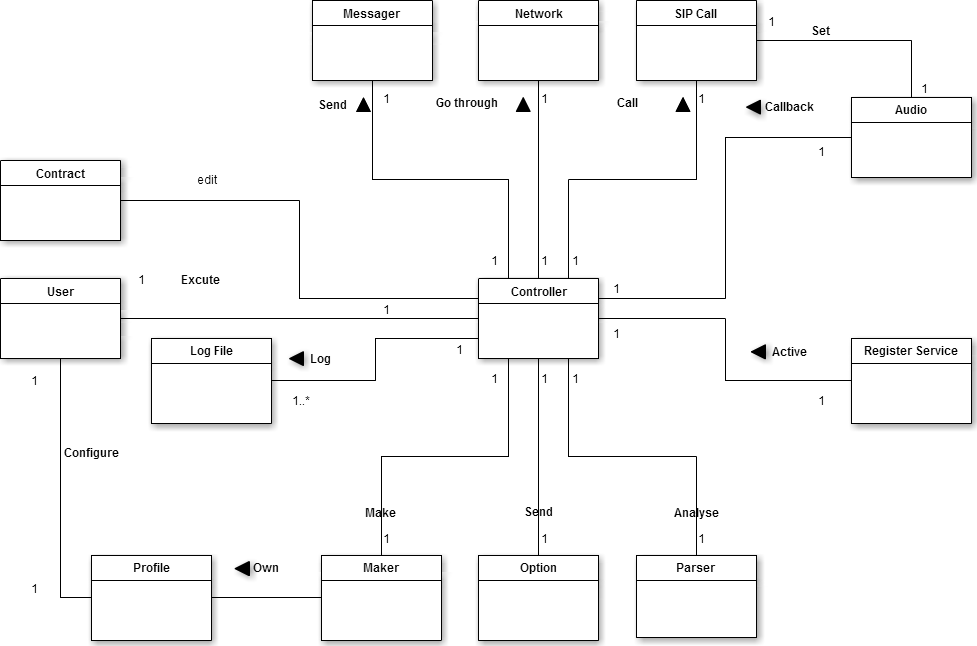


Figure 2-2 Domain Class Diagram with Associations

* 1. Add Attributes

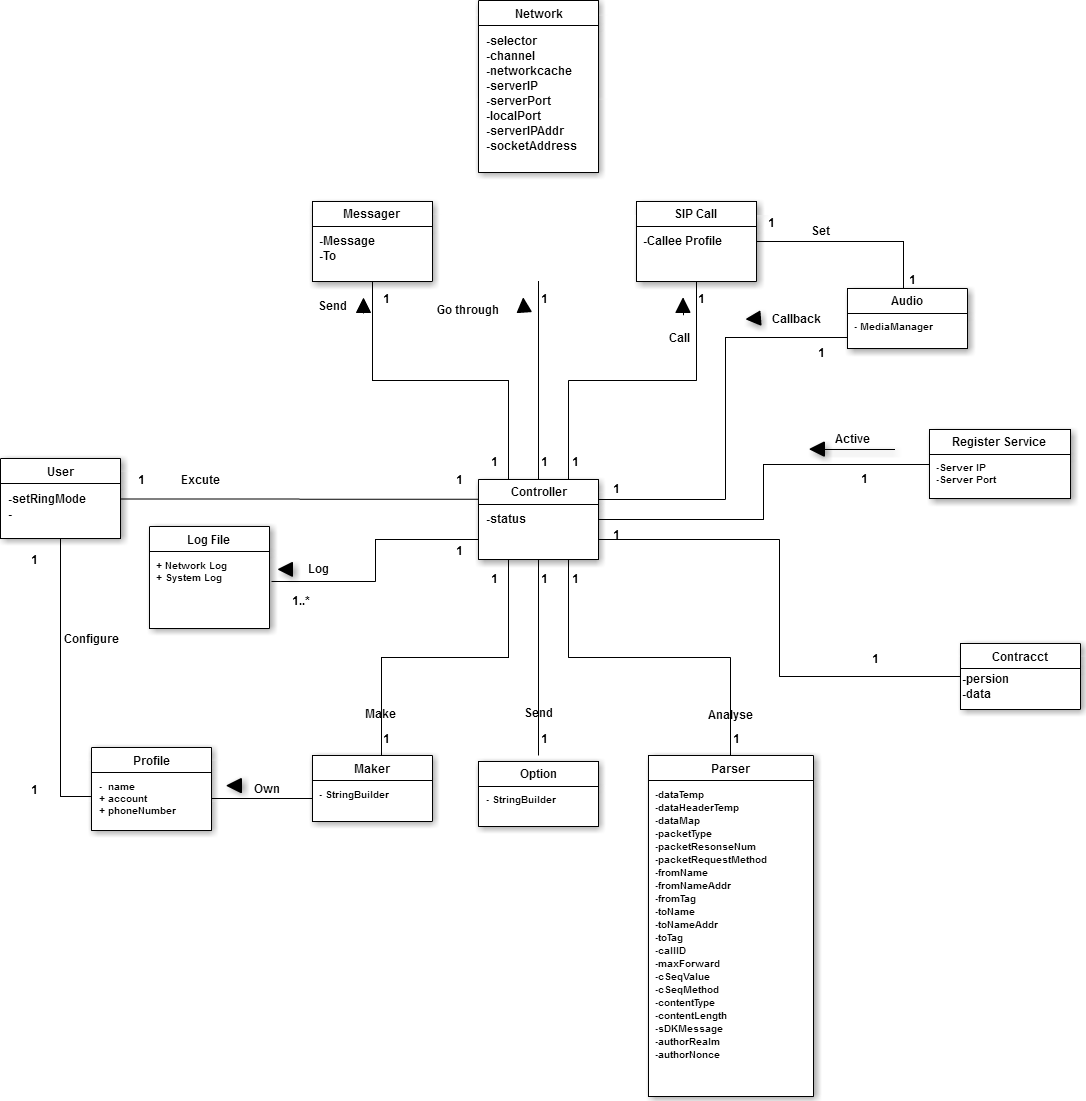


Figure 2-3 Domain Class Diagram with Attributes

1. Design
   1. Logical Architecture

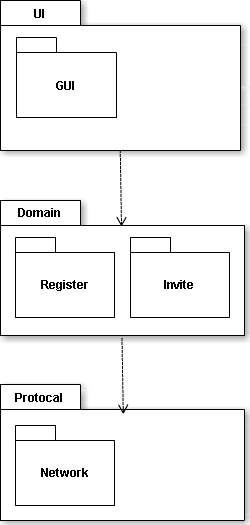
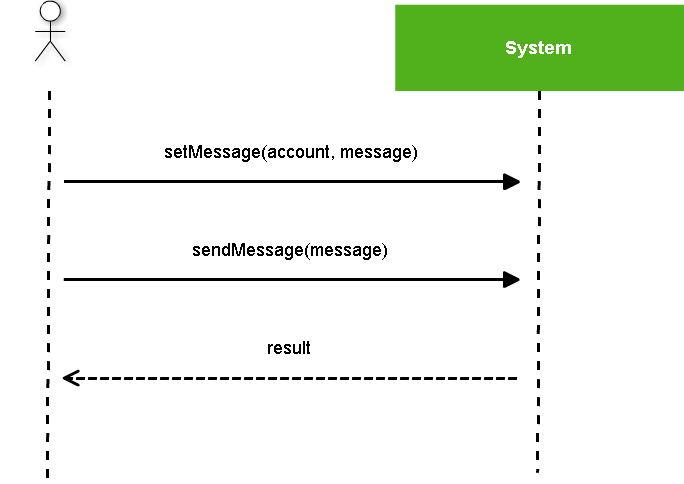


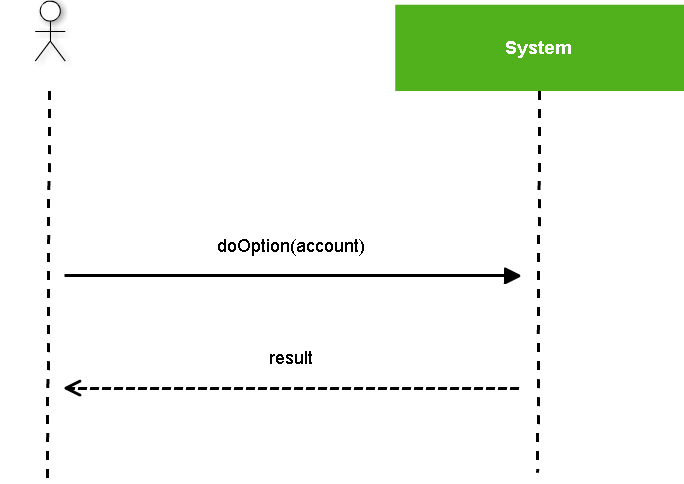
Figure 3-1 ***Logical Architecture in UML package diagram***

* 1. Use-Case Realizations with GRASP Patterns
     1. Send Message



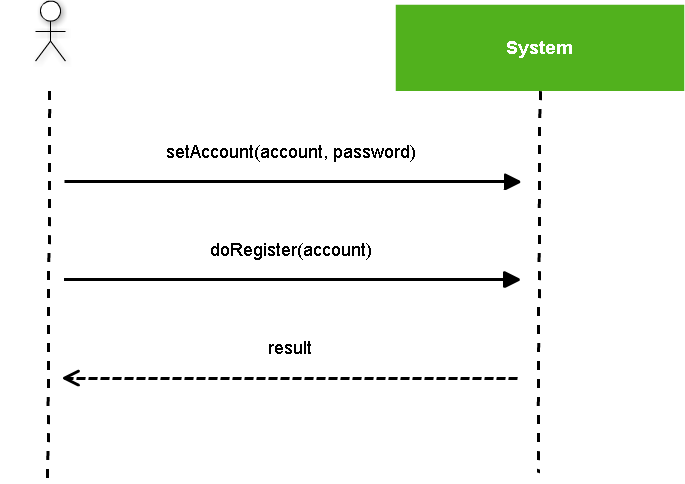
**Figure 3-2 System Sequence Diagram of use case: Send Message**

* + 1. Check Online



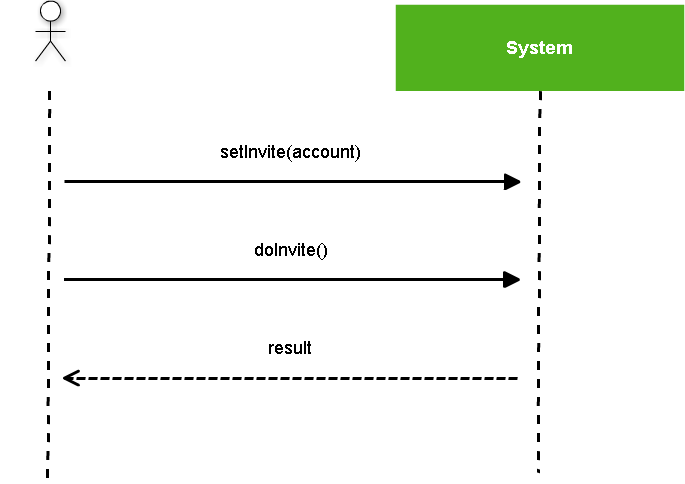
**Figure 3-3 System Sequence Diagram of use case:** **Check Online**

* + 1. Register SIP



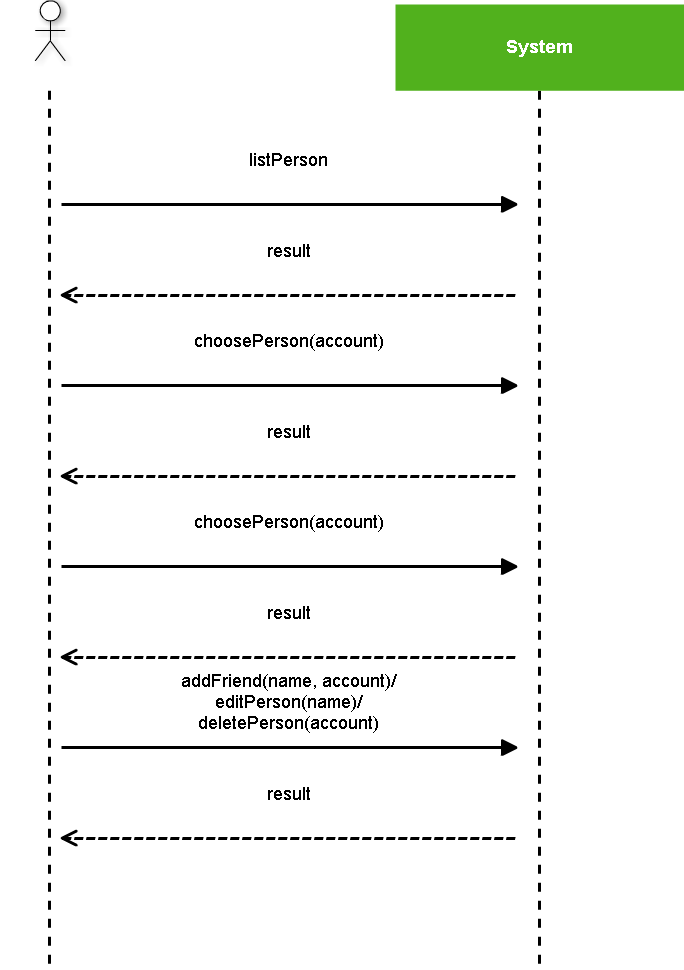
**Figure 3-4 System Sequence Diagram of use case: Register SIP**

* + 1. Call SIP



**Figure 3-5 System Sequence Diagram of use case: Call SIP**

* + 1. Manage Contact



**Figure 3-6 System Sequence Diagram of use case: Manage Contact**

* + 1. Operation Contract

|  |  |
| --- | --- |
|  | doRegister (String account, String password) |
|  | doInvite (String account) |
|  | doOption(String account) |
|  | sendMessage (String message, String account) |
|  | listPerson() |
|  | addFriend(String name, String account) |
|  | editPerson(String name) |
|  | deletePerson(String account) |

1. doRegister

|  |  |
| --- | --- |
| Operation | doRegister (String account, String password) |
| Cross References | Use Case: Register SIP |
| Preconditions | -none |
| Postconditions | -set register account  -make invite package |

1. doInvite

|  |  |
| --- | --- |
| Operation | doInvite () |
| Cross References | Use Case: Call SIP |
| Preconditions | -Device has network ability  -account is not null |
| Postconditions | -send invite package  -make invite package  -do proxy authorization |

1. doOption

|  |  |
| --- | --- |
| Operation | doOption (String account) |
| Cross References | Use Case: Check Online |
| Preconditions | -Device has network ability  -account is not null |
| Postconditions | -send option package |

1. sendMessage()

|  |  |
| --- | --- |
| Operation | sendMessage (String message, String account) |
| Cross References | Use Case: Send Message |
| Preconditions | -Device has network ability  -account is not null |
| Postconditions | -send option package |

1. listPerson()

|  |  |
| --- | --- |
| Operation | listPerson () |
| Cross References | Use Case: Manage Contact |
| Preconditions | -none |
| Postconditions | -show all contract |

1. addFriend(String name, String account)

|  |  |
| --- | --- |
| Operation | addFriend(String name, String account) |
| Cross References | Use Case: Call SIP, Manage Contact |
| Preconditions | -none |
| Postconditions | -new friend  -updata Contact page |

1. editPerson ()

|  |  |
| --- | --- |
| Operation | editPerson(String name) |
| Cross References | Use Case: Manage Contact |
| Preconditions | -There are at least one contact |
| Postconditions | -edit contact page  - updata Contact page |

1. deletePerson()

|  |  |
| --- | --- |
| Operation | deletePerson(String account) |
| Cross References | Use Case: Manage Contact |
| Preconditions | -there are at least one contact |
| Postconditions | -detect selected contact  - updata Contact page |

* + 1. Sequence Diagram
       1. doRegister

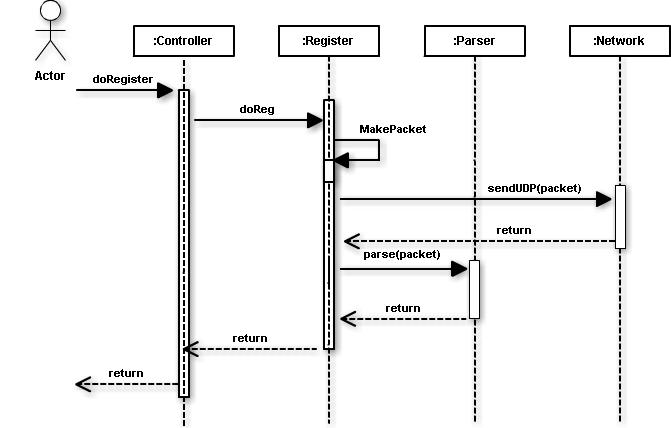


Figure 3-7 Sequence Diagram : doRegister

doInvite

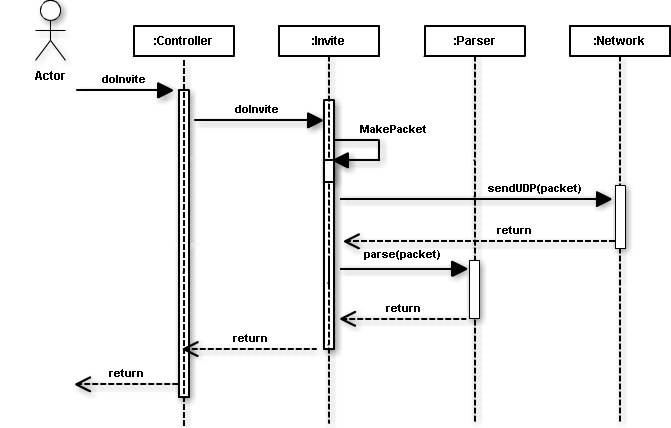
****

Figure 3-8 Sequence Diagram : doInvite

doOption

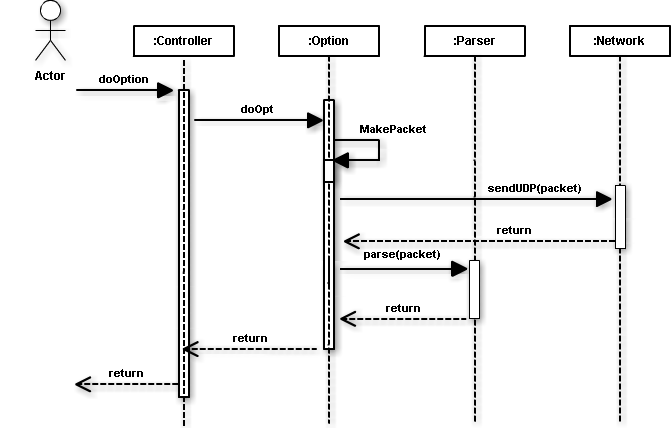


Figure 3-9 Sequence Diagram : doOption

sendMessage

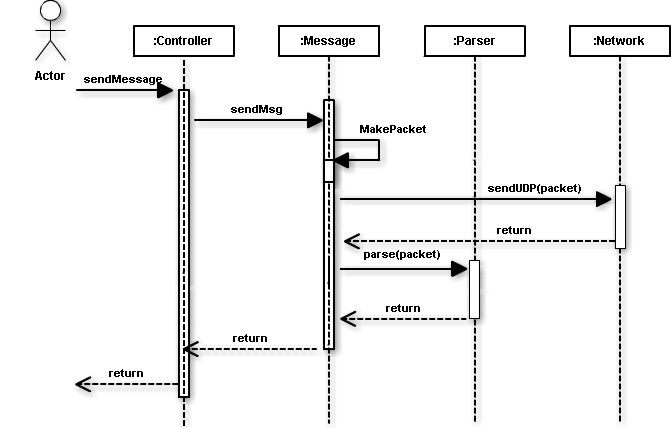


Figure 3-10 Sequence Diagram : sendMessage

listPerson

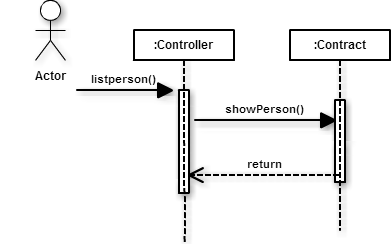
****

Figure 3-11 Sequence Diagram :listPerson

addFriend

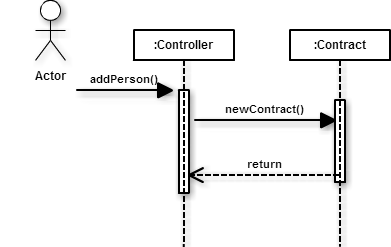
****

Figure 3-12 Sequence Diagram : addFriend

editPerson

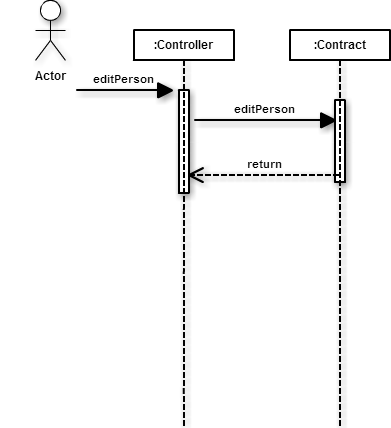


Figure 3-13 Sequence Diagram : editPerson

deletePerson

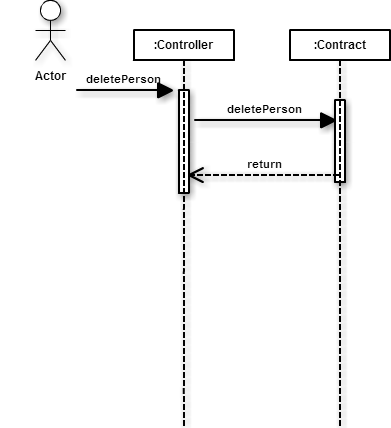
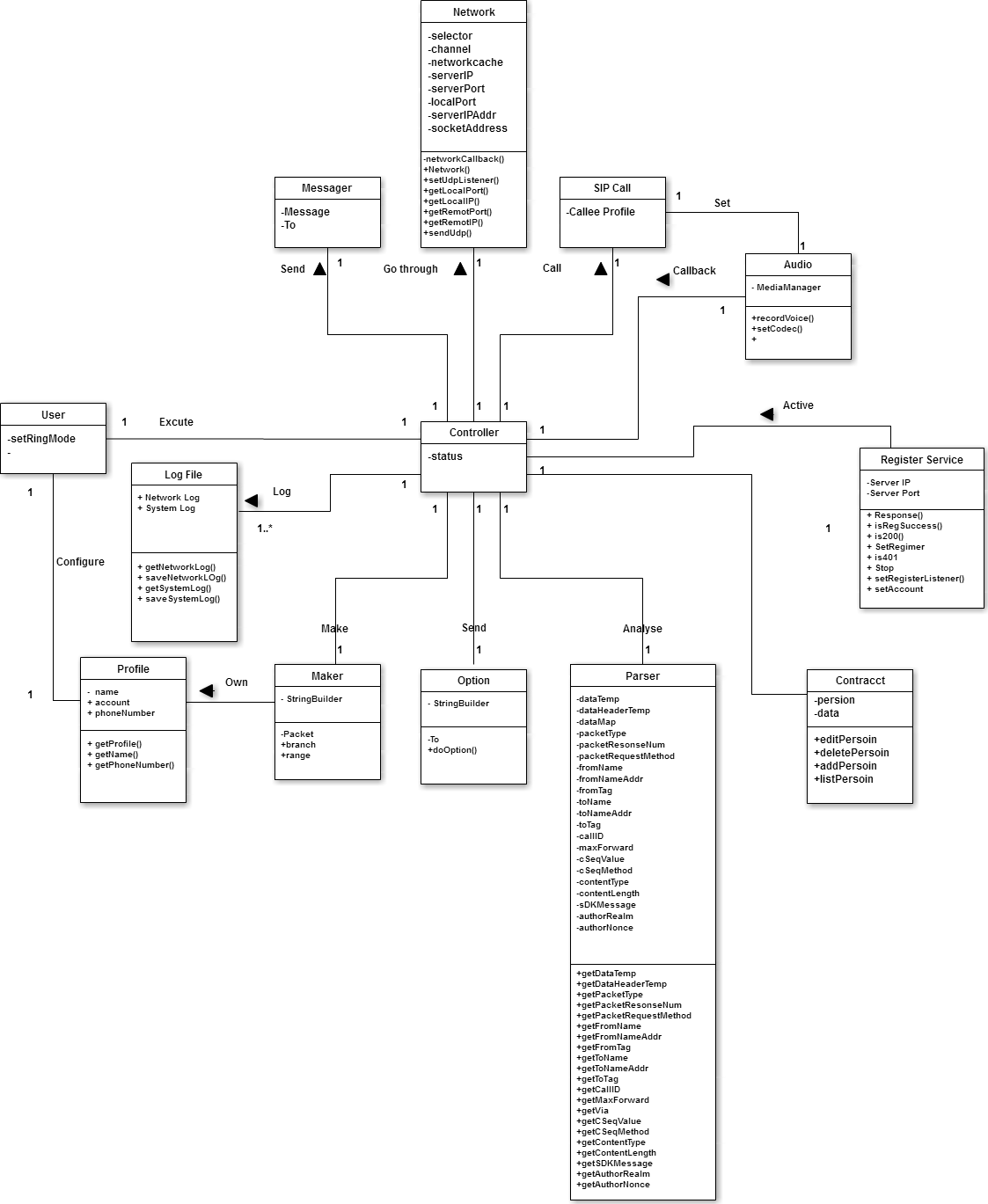


Figure 3-13 Sequence Diagram : deletePerson

* 1. Design Class Model

Figure 3-15 Design Class Model

1. Implementation Class Model
   1. Draw an implementation class diagram for your system(Including associations, attributes and methods)

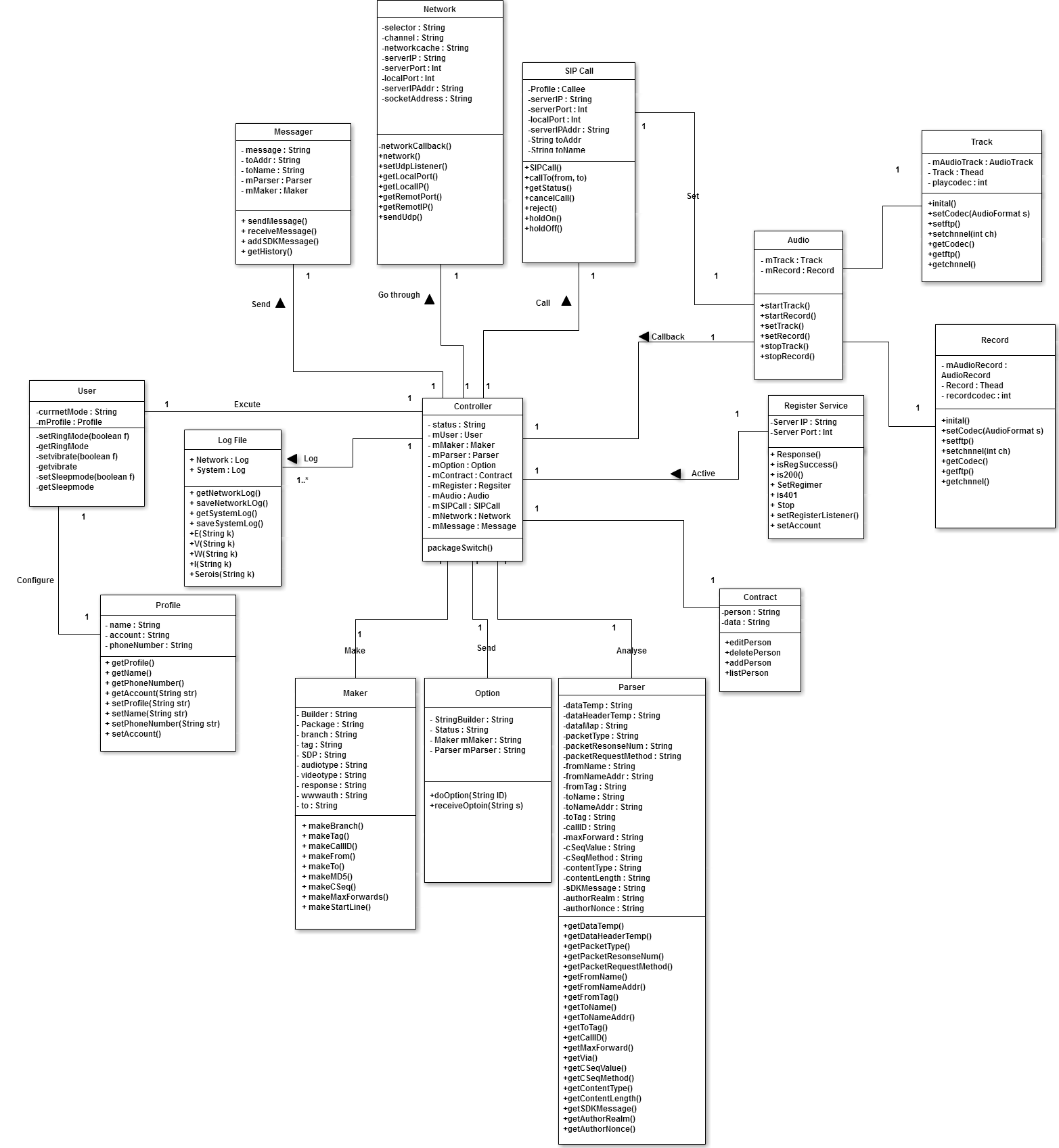


Figure 4-1 Domain

* 1. Implementation class model and design class model difference

Table 4.2.1: Comparison with design and implementation class

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **Method** | **Design** | **Imp.** |
| Controller | packageSwitch() | No | Yes |
| Messager | sendMessage() | No | Yes |
| receiveMessage() | No | Yes |
| addSDKMessage() | No | Yes |
| getHistory() | No | Yes |
| Network | networkCallback() | Yes | Yes |
| network() | Yes | Yes |
| setUdpListener() | Yes | Yes |
| getLocalPort() | Yes | Yes |
| getLocalIP() | Yes | Yes |
| getRemotPort() | Yes | Yes |
| getRemotIP() | Yes | Yes |
| sendUdp() | Yes | Yes |
| SIP Call | SIPCall() | No | Yes |
| callTo(from, to) | No | Yes |
| getStatus() | No | Yes |
| cancelCall() | No | Yes |
| reject() | No | Yes |
| holdOn() | No | Yes |
| holdOff() | No | Yes |
| Audio | startTrack() | No | Yes |
| startRecord() | No | Yes |
| setTrack() | No | Yes |
| setRecord() | No | Yes |
| stopTrack() | No | Yes |
| stopRecord() | No | Yes |
| recordVoice() | Yes | No |
| setCodec() | Yes | No |
| Track | inital() | No | Yes |
| setCodec(AudioFormat s) | No | Yes |
| setftp() | No | Yes |
| setchnnel(int ch) | No | Yes |
| getCodec() | No | Yes |
| getftp() | No | Yes |
| getchnnel() | No | Yes |
| Record | inital() | No | Yes |
| setCodec(AudioFormat s) | No | Yes |
| setftp() | No | Yes |
| setchnnel(int ch) | No | Yes |
| getCodec() | No | Yes |
| getftp() | No | Yes |
| getchnnel() | No | Yes |
| Register Service | Response() | Yes | Yes |
| isRegSuccess() | Yes | Yes |
| is200() | Yes | Yes |
| SetRegimer | Yes | Yes |
| is401 | Yes | Yes |
| Stop | Yes | Yes |
| setRegisterListener() | Yes | Yes |
| setAccount | Yes | Yes |
| Contract | editPerson | Yes | Yes |
| deletePerson | Yes | Yes |
| addPerson | Yes | Yes |
| listPerson | Yes | Yes |
| Parser | getDataTemp() | Yes | Yes |
| getDataHeaderTemp() | Yes | Yes |
| getPacketType() | Yes | Yes |
| getPacketResonseNum() | Yes | Yes |
| getPacketRequestMethod() | Yes | Yes |
| getFromName() | Yes | Yes |
| getFromNameAddr() | Yes | Yes |
| getFromTag() | Yes | Yes |
| getToName() | Yes | Yes |
| getToNameAddr() | Yes | Yes |
| getToTag() | Yes | Yes |
| getCallID() | Yes | Yes |
| getMaxForward() | Yes | Yes |
| getVia() | Yes | Yes |
| getCSeqValue() | Yes | Yes |
| getCSeqMethod() | Yes | Yes |
| getContentType() | Yes | Yes |
| getContentLength() | Yes | Yes |
| getSDKMessage() | Yes | Yes |
| getAuthorRealm() | Yes | Yes |
| getAuthorNonce() | Yes | Yes |
| Option | doOption(String ID) | Yes | Yes |
| receiveOptoin(String s) | No | Yes |
| Maker | makeBranch() | No | Yes |
| makeTag() | No | Yes |
| makeCallID() | No | Yes |
| makeFrom() | No | Yes |
| makeTo() | No | Yes |
| makeMD5() | No | Yes |
| makeCSeq() | No | Yes |
| makeMaxForwards() | No | Yes |
| makeStartLine() | No | Yes |
| Log File | getNetworkLog() | Yes | Yes |
| saveNetworkLOg() | Yes | Yes |
| getSystemLog() | Yes | Yes |
| saveSystemLog() | Yes | Yes |
| E(String k) | No | Yes |
| V(String k) | No | Yes |
| W(String k) | No | Yes |
| I(String k) | No | Yes |
| Serois(String k) | No | Yes |
| User | setRingMode(boolean f) | Yes | Yes |
| getRingMode | No | Yes |
| setvibrate(boolean f) | No | Yes |
| getvibrate | No | Yes |
| setSleepmode(boolean f) | No | Yes |
| getSleepmode | No | Yes |
| Profile | getProfile() | Yes | Yes |
| getName() | Yes | Yes |
| getPhoneNumber() | Yes | Yes |
| getAccount(String str) | No | Yes |
| setProfile(String str) | No | Yes |
| setName(String str) | No | Yes |
| setPhoneNumber(String str) | No | Yes |
| setAccount() | No | Yes |

***Table 4.2.2: Summary of implementation class/method changed***

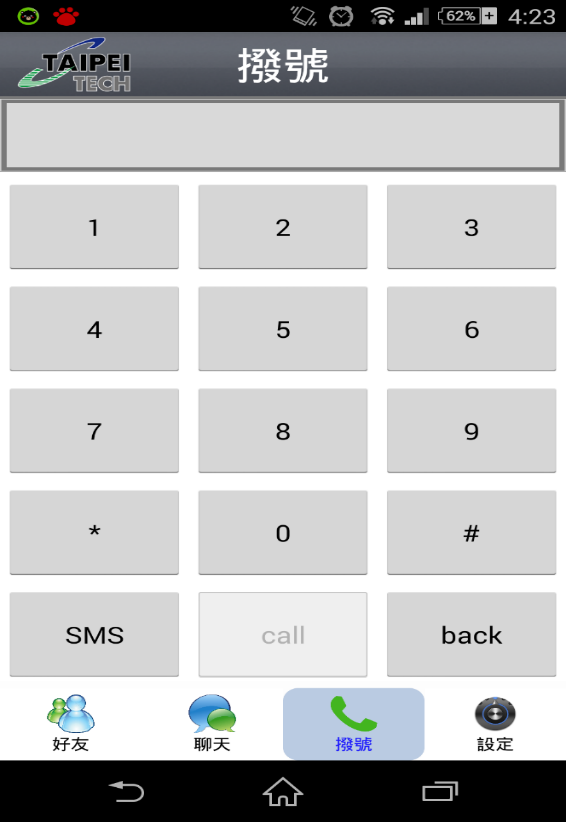
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Number of added** | | **Number of removed** | **Number of modified** |
| Class | 2 | 0 | | 0 |
| Method | 57 | 2 | | 0 |

* 1. Calculate Line of Code

***Table 4.3.1: Line of Code of classes***

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Class Name** | **Number of Methods** | **Line of Code in Class**  **(without comment)** |
|  | Controller | 1 |  |
|  | Messager | 4 | 0 |
|  | Network | 8 |  |
|  | SIP Call | 7 |  |
|  | Audio | 8 |  |
|  | Track | 7 |  |
|  | Record | 7 |  |
|  | Register Service | 4 |  |
|  | Contract | 4 |  |
|  | Parser | 21 | 245 |
|  | Option | 2 |  |
|  | Maker | 9 |  |
|  | Log File | 9 |  |
|  | User | 6 |  |
|  | Profile | 8 |  |

1. **Programming**
   1. Snapshots of system execution



* 1. Source Code Listing
     1. MainActive

**package** com.sample.android.sip.sipphone;  
  
**import** android.content.ComponentName;  
**import** android.content.Context;  
**import** android.content.Intent;  
**import** android.content.ServiceConnection;  
**import** android.os.Bundle;  
**import** android.os.IBinder;  
**import** android.support.design.widget.FloatingActionButton;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.support.v7.widget.Toolbar;  
**import** android.view.View;  
  
**public class** MainActivity **extends** AppCompatActivity {  
 **private static final** String ***TAG*** = **"MainActivity"**;  
 **private** LocalService **mService**;  
 **private boolean mBound** = **false**;  
 */\*\*  
 \* Defines callbacks for service binding, passed to bindService()  
 \*/* **private** ServiceConnection **mConnection** = **new** ServiceConnection() {  
 @Override  
 **public void** onServiceConnected(ComponentName className, IBinder service) {  
 *// We've bound to LocalService, cast the IBinder and get LocalService instance* LocalService.LocalBinder binder = (LocalService.LocalBinder) service;  
 **mService** = binder.getService();  
 **mBound** = **true**;  
 }  
  
 @Override  
 **public void** onServiceDisconnected(ComponentName arg0) {  
 **mBound** = **false**;  
 }  
 };  
  
 @Override  
 **protected void** onDestroy() {  
 **super**.onDestroy();  
 **if** (**mBound**)  
 unbindService(**mConnection**);  
 }  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_main***);  
 Toolbar toolbar = (Toolbar) findViewById(R.id.***toolbar***);  
 setSupportActionBar(toolbar);  
  
 *//Bind server* Intent intent = **new** Intent(**this**, LocalService.**class**);  
 bindService(intent, **mConnection**, Context.***BIND\_AUTO\_CREATE***);  
  
 FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.***fab***);  
 **assert** fab != **null**;  
 fab.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 **mService**.doReg();  
*// mService.doInvite();* }  
 });  
 }  
}

* + 1. LocalService

**package** com.sample.android.sip.sipphone;  
  
**import** android.app.Service;  
**import** android.content.Intent;  
**import** android.os.Binder;  
**import** android.os.IBinder;  
  
**import** java.util.Map;  
  
  
**public class** LocalService **extends** Service {  
 **private static final** String ***TAG*** = **"LocalService"**;  
 **final** Network **netowrk** = **new** Network(**"02.ntut.edu.tw"**, 5060);  
 *// Binder given to clients*  
**private final** IBinder **mBinder** = **new** LocalBinder();  
 Parser **parser** = **new** Parser();  
 **private** Register **regTest**;  
 **private** Invite **inviteTest**;  
 **private int testAccount** = 40110;  
  
 **private void** StopThread() {  
 *// regTest.Stop();*  
 *// regTest.interrupt();*  
}  
  
 **private void** filterPackage(String message) {  
 **parser**.pushData(message);  
 **if** (**parser**.getCSeqMethod().compareToIgnoreCase(**"REGISTER"**) == 0) {  
 **regTest**.Response(message);  
 } **else if** (**parser**.getCSeqMethod().compareToIgnoreCase(**"INVITE"**) == 0 ||  
 **parser**.getCSeqMethod().compareToIgnoreCase(**"ACK"**) == 0) {  
 **inviteTest**.Response(message);  
 }  
 }  
  
 @Override  
 **public** IBinder onBind(Intent intent) {  
 **return mBinder**;  
 }  
  
 @Override  
 **public void** onCreate() {  
 **super**.onCreate();  
 **netowrk**.start();  
 setNetworkCallback();  
 setParserCallback();  
 **regTest** = **new** Register(**netowrk**);  
 setRegisterAgentCallback();  
 **regTest**.start();  
 **inviteTest** = **new** Invite(**netowrk**);  
 }  
  
 @Override  
 **public void** onDestroy() {  
 **super**.onDestroy();  
 StopThread();  
 }  
  
 **public void** doReg() {  
 **if**(!**regTest**.isAlive())  
 **regTest**.start();  
 **regTest**.setAccount(Integer.*toString*(**testAccount**));  
 **testAccount**++;  
 **regTest**.doReg();  
 }  
  
 **public void** doInvite() {  
 **inviteTest**.setAccount(Integer.*toString*(**testAccount**), Integer.*toString*(30051));  
 **inviteTest**.doInvite();  
 **netowrk**.sendUdp(**inviteTest**.makePacket());  
 }  
  
 **public void** setNetworkCallback() {  
 **netowrk**.setUdpListener(**new** Network.NetworkCallback() {  
 @Override  
 **public void** udpListener(String message, **int** messageLength) {  
 filterPackage(message);  
 }  
 });  
 }  
  
 **public void** setRegisterAgentCallback() {  
 **regTest**.setRegisterListener(**new** Register.RegisterAgentCallback() {  
 @Override  
 **public void** PacketListener(String message, **int** messageLength) {  
 *// Reg.E(message);*  
**netowrk**.sendUdp(message);  
 }  
 });  
 }  
  
 **public void** setParserCallback() {  
 **parser**.setParserListener(**new** Parser.ParserCallback() {  
 @Override  
 **public void** parserListener(String type, Map<String, String> data) {  
  
 }  
 });  
 }  
  
 **public class** LocalBinder **extends** Binder {  
 LocalService getService() {  
 *// Return this instance of LocalService so clients can call public methods*  
**return** LocalService.**this**;  
 }  
 }  
}

* + 1. Network

**package** com.sample.android.sip.sipphone;  
  
**import** android.util.Log;  
  
**import** java.io.IOException;  
**import** java.net.InetAddress;  
**import** java.net.InetSocketAddress;  
**import** java.net.SocketAddress;  
**import** java.net.SocketException;  
**import** java.net.UnknownHostException;  
**import** java.nio.ByteBuffer;  
**import** java.nio.channels.ClosedChannelException;  
**import** java.nio.channels.DatagramChannel;  
**import** java.nio.channels.SelectionKey;  
**import** java.nio.channels.Selector;  
**import** java.nio.charset.Charset;  
**import** java.util.Iterator;  
  
*/\*\**  
 *\* Created by clarence- on 2016/4/8.*  
 *\*/*  
**public class** Network **extends** Thread {  
  
 **private static final** String ***TAG*** = **"Network"**;  
 Selector **selector**;  
 DatagramChannel **channel**;  
 **int networkCache** = 65536;  
 String **serverIP**;  
 **int serverPort** = 5060, **localPort** = 0;  
 InetAddress **serverIPAddr**;  
 SocketAddress **socketAddress**;  
  
 **private** NetworkCallback **networkCallback**;  
  
 **public** Network(String serverIP, **int** serverPort) {  
 **this**.**serverIP** = serverIP;  
 **this**.**serverPort** = serverPort;  
 }  
  
 ;  
  
 **public** Network(String serverIP, **int** serverPort, **int** localPort) {  
 **this**.**serverIP** = serverIP;  
 **this**.**serverPort** = serverPort;  
 **this**.**localPort** = localPort;  
 }  
  
 **public void** setUdpListener(NetworkCallback networkCallback) {  
 **this**.**networkCallback** = networkCallback;  
 }  
  
 **public int** getLocalPort() {  
 **return channel**.socket().getLocalPort();  
 }  
  
 **public** String getLocalIP() {  
 **return channel**.socket().getLocalAddress().toString().substring(1);  
 }  
  
 **public int** getRemotePort() {  
 **return serverPort**;  
 }  
  
 **public** String getRemoteIP() {  
 **return serverIPAddr**.getHostAddress();  
 }  
  
 @Override  
 **public void** run() {  
 **super**.run();  
  
 **try** {  
 **serverIPAddr** = InetAddress.*getByName*(**serverIP**);  
 **channel** = DatagramChannel.*open*();  
 **channel**.socket().bind(**new** InetSocketAddress(**localPort**));  
 **channel**.configureBlocking(**false**);  
 **socketAddress** = **new** InetSocketAddress(**serverIPAddr**, **serverPort**);  
 **channel**.connect(**socketAddress**);  
 **selector** = Selector.*open*();  
 **channel**.register(**selector**, SelectionKey.***OP\_READ***);  
 } **catch** (SocketException e) {  
 e.printStackTrace();  
 } **catch** (UnknownHostException e) {  
 e.printStackTrace();  
 } **catch** (ClosedChannelException e) {  
 e.printStackTrace();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 ByteBuffer byteBuffer = ByteBuffer.*allocate*(**networkCache**);  
 **while** (**true**) {  
 **try** {  
 **if** (**selector**.select() > 0) {  
 Iterator iterator = **selector**.selectedKeys().iterator();  
 **while** (iterator.hasNext()) {  
 SelectionKey selectionKey = (SelectionKey) iterator.next();  
 iterator.remove();  
 **if** (selectionKey.isReadable()) {  
 **channel** = (DatagramChannel) selectionKey.channel();  
 **int** packageSize = **channel**.read(byteBuffer);  
 **networkCallback**.udpListener(**new** String(byteBuffer.array()).substring(0, packageSize), packageSize);  
 *//Log.d(TAG, "receiveUdp: " + new String(byteBuffer.array()));*  
byteBuffer.clear();  
 }  
 }  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 **public void** sendUdp(**final** String message) {  
 **new** Thread(**new** Runnable() {  
 @Override  
 **public void** run() {  
 **try** {  
 **int** len = **channel**.send(Charset.*defaultCharset*().encode(message), **socketAddress**);  
 Log.*d*(***TAG***, **"Send: "** + message + **" Len: "** + len);  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }).start();  
 }  
  
 **public interface** NetworkCallback {  
 **void** udpListener(String message, **int** messageLength);  
 }  
}

* + 1. Parser

**package** com.sample.android.sip.sipphone;  
  
import android.os.Message;  
import android.util.Log;  
  
import org.json.JSONArray;  
  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.Map;  
  
/\*\*  
 \* Created by DP on 2016/4/22.  
 \*/  
public class Parser {  
  
 **private** String[] **dataTemp**;  
 **private** String[] **dataHeaderTemp**;  
  
 **private** Map<String, String> **dataMap** = **new** HashMap<String, String>();  
  
 **private** String **message**;  
 **private** String **packetType**, **packetResonseNum**, **packetRequestMethod**;  
 **private** String **fromName**, **fromNameAddr**, **fromTag**;  
 **private** String **toName**, **toNameAddr**, **toTag**;  
 **private** String **callID**;  
 **private** String **maxForward**;  
 **private** String **cSeqValue**, **cSeqMethod**;  
 **private** String **contentType**;  
 **private** String **contentLength**;  
 **private** String **sDKMessage**;  
 **private** String **authorRealm**, **authorNonce**;  
 **private** String **proxyAuthorRealm**, **proxyAuthorNonce**;  
 **private int viaCount**;  
 **private** String[] **via**,**viaBranch**,**viaRport**;  
  
 **private** ParserCallback **parserCallback**;  
  
 **public interface** ParserCallback {  
 **void** parserListener(String Type, Map<String, String> Data);  
 }  
  
 ;  
  
 **public void** setParserListener(ParserCallback parserCallback) {  
 **this**.**parserCallback** = parserCallback;  
 }  
  
 **public void** pushData(String Message) {  
  
 *//--Initial class*  
**message** = **new** String(Message);  
 **packetType** = **""**;  
 **packetResonseNum** = **""**;  
 **packetRequestMethod** = **""**;  
 **fromName** = **""**;  
 **fromNameAddr** = **""**;  
 **fromTag** = **""**;  
 **toName** = **""**;  
 **toNameAddr** = **""**;  
 **toTag** = **""**;  
 **callID** = **""**;  
 **maxForward** = **""**;  
 **cSeqValue** = **""**;  
 **cSeqMethod** = **""**;  
 **contentType** = **""**;  
 **contentLength** = **""**;  
 **sDKMessage** = **""**;  
 **authorRealm** = **""**;  
 **authorNonce** = **""**;  
 **proxyAuthorRealm** = **""**;  
 **proxyAuthorNonce** = **""**;  
 **viaCount** = 0;  
 *//--*  
**int** pos, count;  
  
 pos = **message**.indexOf(**"\r\n\r\n"**);  
 **sDKMessage** = **message**.substring(pos + 4);  
 **if**(pos > 0)  
 **message** = **message**.substring(0,pos);  
  
 **dataTemp** = Message.split(**"\r\n"**);  
 **int** dataLength = **dataTemp**.**length**;  
 **dataHeaderTemp** = **new** String[dataLength];  
 **for** (count = 0; count < dataLength; count++) {  
 **if** (count == 0) {  
 **if** (**dataTemp**[count].indexOf(**"SIP/2.0"**) > 5)  
 **packetType** = **"REQUEST"**;  
 **else**  
 **packetType** = **"RESPONSE"**;  
 } **else** {  
 pos = **dataTemp**[count].indexOf(**":"**);  
 **dataHeaderTemp**[count] = **dataTemp**[count].substring(0, pos);  
 **dataHeaderTemp**[count] = **dataHeaderTemp**[count].toUpperCase();  
 }  
 }  
 **for** (count = 0; count < dataLength; count++) {  
 **if** (count == 0)  
 **dataMap**.put(**"STARTLINE"**, **dataTemp**[count]);  
 **else**  
 **dataMap**.put(**dataHeaderTemp**[count], **dataTemp**[count]);  
 }  
 **int** pos1, pos2;  
 String temp;  
  
 **if** (**dataMap**.get(**"STARTLINE"**) != **null**) {  
 temp = **dataMap**.get(**"STARTLINE"**);  
 **if** (**packetType** == **"REQUEST"**) {  
 pos2 = temp.indexOf(**" "**);  
 **packetRequestMethod** = temp.substring(0,pos2);  
 }  
 **else if** (**packetType** == **"RESPONSE"**) {  
 pos1 = temp.indexOf(**" "**);  
 pos2 = temp.indexOf(**" "**,pos1 + 1);  
 **packetResonseNum** = temp.substring(pos1 + 1,pos2);  
 }  
 *//有錯*  
}  
  
 **if** (**dataMap**.get(**"FROM"**) != **null**) {  
 temp = **dataMap**.get(**"FROM"**);  
 pos1 = temp.indexOf(**": "**);  
 pos2 = temp.indexOf(**"<"**,pos1);  
 **fromName** = temp.substring(pos1 + 2, pos2);  
 pos1 = temp.indexOf(**"sip:"**, pos2);  
 pos2 = temp.indexOf(**">"**, pos1);  
 **fromNameAddr** = temp.substring(pos1 + 4, pos2);  
 pos1 = temp.indexOf(**"tag="**, pos2);  
 **fromTag** = temp.substring(pos1 + 4);  
 }  
  
 **if** (**dataMap**.get(**"TO"**) != **null**) {  
 temp = **dataMap**.get(**"TO"**);  
 pos1 = temp.indexOf(**": "**);  
 pos2 = temp.indexOf(**"<"**,pos1);  
 **toName** = temp.substring(pos1 + 2, pos2);  
 pos1 = temp.indexOf(**"sip:"**, pos2);  
 pos2 = temp.indexOf(**">"**, pos1);  
 **toNameAddr** = temp.substring(pos1 + 4, pos2);  
 pos1 = temp.indexOf(**"tag="**, pos2);  
 **if**(pos1 > pos2)  
 **toTag** = temp.substring(pos1 + 4);  
 **else**  
 **toTag** = **""**;  
 }  
  
 **if**(**dataMap**.get(**"CSEQ"**) != **null**) {  
 temp = **dataMap**.get(**"CSEQ"**);  
 pos1 = temp.indexOf(**": "**);  
 pos2 = temp.indexOf(**" "**, pos1 + 2);  
 **cSeqValue** = temp.substring(pos1 + 2, pos2);  
 pos1 = temp.indexOf(**" "**, pos2);  
 **cSeqMethod** = temp.substring(pos1 + 1);  
 }  
  
 **if**(**dataMap**.get(**"CALL-ID"**) != **null**) {  
 temp = **dataMap**.get(**"CALL-ID"**);  
 pos1 = temp.indexOf(**":"**);  
 **callID** = temp.substring(pos1 + 2);  
 }  
 **if**(**dataMap**.get(**"MAX-FORWARDS"**) != **null**) {  
 temp = **dataMap**.get(**"MAX-FORWARDS"**);  
 pos1 = temp.indexOf(**":"**);  
 **maxForward** = temp.substring(pos1 + 2);  
 }  
  
 **if**(**dataMap**.get(**"CONTENT-LENGTH"**) != **null**) {  
 temp = **dataMap**.get(**"CONTENT-LENGTH"**);  
 pos1 = temp.indexOf(**":"**);  
 **contentLength** = temp.substring(pos1 + 2);  
 }  
  
 **if**(**dataMap**.get(**"WWW-AUTHENTICATE"**) != **null**) {  
 temp = **dataMap**.get(**"WWW-AUTHENTICATE"**);  
 pos1 = temp.indexOf(**"realm=\""**);  
 pos2 = temp.indexOf(**"\""**,pos1 + 7);  
 **authorRealm** = temp.substring(pos1 + 7, pos2);  
 pos1 = temp.indexOf(**"nonce=\""**,pos2);  
 pos2 = temp.indexOf(**"\""**,pos1 + 7);  
 **authorNonce** = temp.substring(pos1 + 7, pos2);  
 }  
  
 **if** (**dataMap**.get(**"PROXY-AUTHENTICATE"**) != **null**) {  
 temp = **dataMap**.get(**"PROXY-AUTHENTICATE"**);  
 pos1 = temp.indexOf(**"realm=\""**);  
 pos2 = temp.indexOf(**"\""**, pos1 + 7);  
 **proxyAuthorRealm** = temp.substring(pos1 + 7, pos2);  
 pos1 = temp.indexOf(**"nonce=\""**, pos2);  
 pos2 = temp.indexOf(**"\""**, pos1 + 7);  
 **proxyAuthorNonce** = temp.substring(pos1 + 7, pos2);  
 }  
  
 }  
  
 **public** String getPacketType() { **return packetType**; }  
 **public** String getPacketResonseNum() { **return packetResonseNum**; }  
 **public** String getPacketRequestMethod() { **return packetRequestMethod**; }  
 **public** String getFromName() { **return fromName**; }  
 **public** String getFromNameAddr() { **return fromNameAddr**; }  
 **public** String getFromTag() { **return fromTag**; }  
 **public** String getToName() { **return toName**; }  
 **public** String getToNameAddr() { **return toNameAddr**; }  
 **public** String getToTag() { **return toTag**; }  
 **public** String getCallID() { **return callID**; }  
 **public** String getMaxForward() { **return maxForward**; }  
 **public** String getcSeqValue() { **return cSeqValue**; }  
 **public** String getcSeqMethod() { **return cSeqMethod**; }  
 **public** String getContentType() { **return contentType**; }  
 **public** String getContentLength() { **return contentLength**; }  
 **public** String getsDKMessage() { **return sDKMessage**; }  
 **public** String getAuthorRealm() { **return authorRealm**; }  
 **public** String getAuthorNonce() { **return authorNonce**; }  
 **public** String getProxyAuthorRealm() { **return proxyAuthorRealm**; }  
 **public** String getProxyAuthorNonce() { **return proxyAuthorNonce**; }  
 **public int** getViaCount() {**return viaCount**; }  
 **public** String getVia(**int** i) {  
 **if**(i < **viaCount** && i>=0)  
  
 **return ""**;  
 **else**  
 **return via**[i];  
 }  
 **public** String getViabranch(**int** i) {  
 **if**(i < **viaCount** && i>=0)  
  
 **return ""**;  
 **else**  
 **return viaBranch**[i];  
 }  
 **public** String getViaRport(**int** i) {  
 **if**(i < **viaCount** && i>=0)  
 **return ""**;  
 **else**  
 **return viaRport**[i]; }  
  
}

* + 1. Invite

**package** com.sample.android.sip.sipphone;  
  
*/\*\*  
 \* Created by lab214-clarence-pc on 2016/4/28.  
 \*/***public class** Invite {  
 **private static** Network *network*;  
 **private static** SIPHeaderParameter *packetMaker*;  
 **private** StringBuilder **packet** = **new** StringBuilder();  
 **private int cseqNum** = 1;  
 **private** String **fromAccount**, **toAccount**;  
 **private** String **method** = **"INVITE "**;  
 **private** String **sipURI** = **"sip:"**, **fromURI** = **"sip:"**, **toURI** = **"sip:"**, **contactURI** = **"sip:"**;  
  
 **public** Invite(Network network) {  
 **this**.*network* = network;  
 }  
  
 **public void** doInvite() {  
 **packet** = **new** StringBuilder();  
 *packetMaker* = **new** SIPHeaderParameter();  
 makeInvite();  
 }  
  
 **private void** makeInvite() {  
 **packet**.append(**method** + **sipURI** + **" SIP/2.0\r\n"**);  
 **packet**.append(**"Via: SIP/2.0/UDP "** + *network*.getLocalIP() + **":"** + *network*.getLocalPort() + **";branch:="** + *packetMaker*.branch() + **";rport\r\n"**);  
 **packet**.append(**"From: <"** + **fromURI** + **">;tag="** + *packetMaker*.Integer() + **"\r\n"**);  
 **packet**.append(**"To: <:"** + **toURI** + **">\r\n"**);  
 **packet**.append(**"CSeq: "** + Integer.*toString*(**cseqNum**) + **" "** + **method** + **"\r\n"**);  
 **packet**.append(**"Call-ID: "** + *packetMaker*.Integer() + **"@"** + *network*.getRemoteIP() + **"\r\n"**);  
 **packet**.append(**"Max-Forwards: 70 \r\n"**);  
 **packet**.append(**"Content-Type: application/sdp\r\n"**);  
 **packet**.append(**"Content-Length: 0\r\n"**);  
 **packet**.append(**"Contact: <"** + **contactURI** + **":"** + *network*.getLocalPort() + **">\r\n\r\n"**);  
 }  
  
 **public void** setAccount(String fromAccount, String toAccount) {  
 **this**.**fromAccount** = fromAccount;  
 **this**.**toAccount** = toAccount;  
 **this**.**sipURI** = **this**.**sipURI** + **this**.**toAccount** + **"@"** + *network*.getRemoteIP();  
 **this**.**toURI** = **new** String(**this**.**sipURI**);  
 **this**.**fromURI** = **this**.**fromURI** + **this**.**fromAccount** + **"@"** + *network*.getRemoteIP();  
 **this**.**contactURI** = **this**.**contactURI** + **this**.**fromAccount** + **"@"** + *network*.getLocalIP();  
 }  
  
 **public** String makePacket() {  
 **return packet**.toString();  
 }  
  
 **public void** Response(String message) {  
  
 }  
}

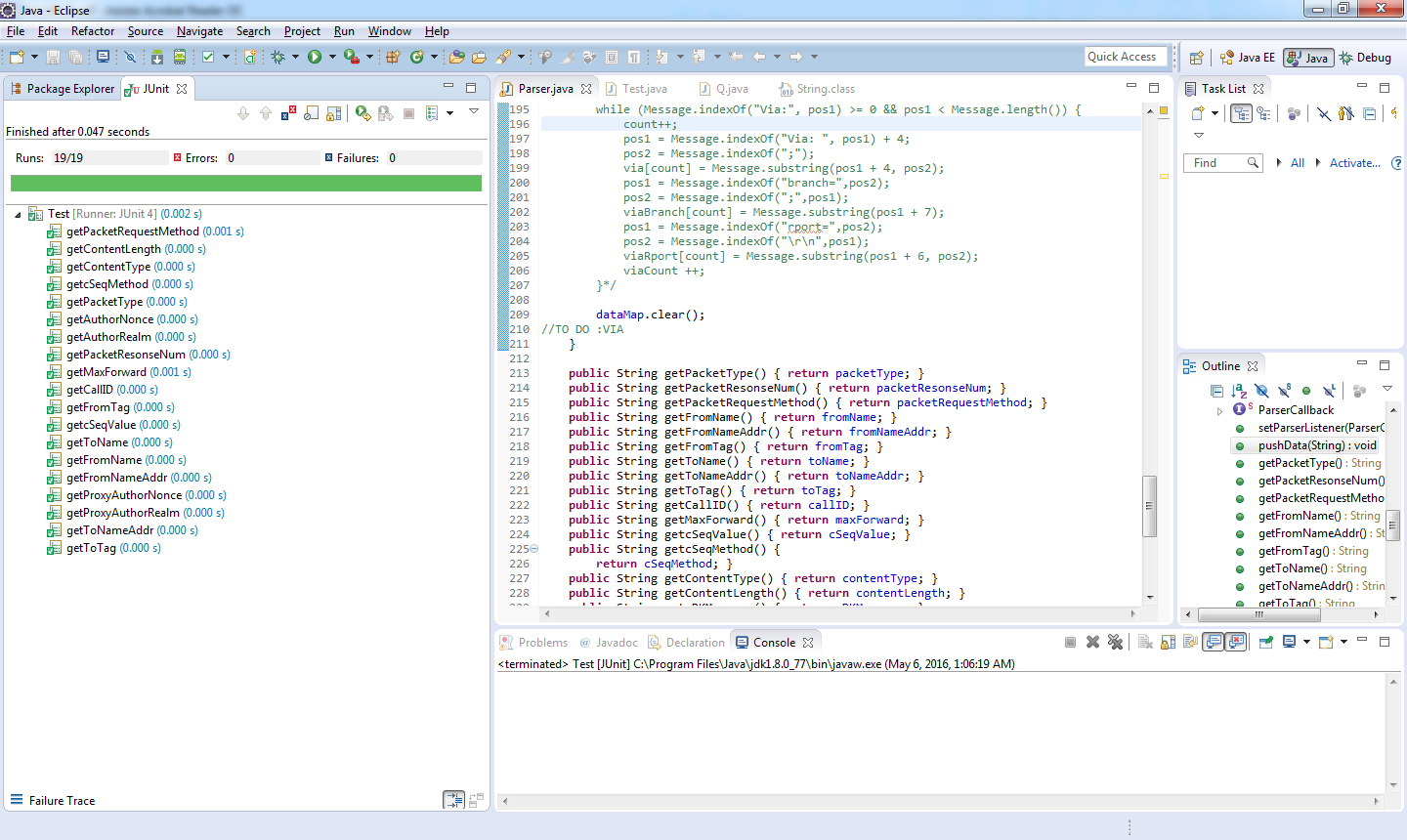
* + 1. Register

**package** com.sample.android.sip.sipphone;  
  
  
**import** java.io.UnsupportedEncodingException;  
**import** java.security.MessageDigest;  
**import** java.text.SimpleDateFormat;  
**import** java.util.Date;  
**import** java.util.Timer;  
**import** java.util.TimerTask;  
**import** java.util.concurrent.ConcurrentLinkedQueue;  
  
**public class** Register **extends** Thread {  
 **private static final** String ***Startline*** = **"REGISTER sip:"**;  
 **private static final** String ***TAG*** = **"Register"**;  
 **private static** String *UAgent* = **"hydar"**;  
 **private static** ConcurrentLinkedQueue<String> *queue* = **new** ConcurrentLinkedQueue<>();  
 **private** Self\_Loger **mSelf** = **new** Self\_Loger(***TAG***);  
 **private** StringBuilder **AuthPacket** = **new** StringBuilder(),**builder** = **new** StringBuilder();  
 **private** SIPHeaderParameter **Reg\_packetMaker**;  
 **private** String **ServerIP**, **mAccount**, **mip**, **mport**, **currrntid**, **nonce**, **ReceivePacket**,**password** = **"123456"**;  
 **private int Expires** = 60,**RegTimes**;  
 **private** Network **netWork**;  
 **private double coffet** = 0.2;  
 **private boolean RegSuccess** = **false**, **RepeatPacketcome** = **false**,**Wokring** = **true**;  
 **private** RegisterAgentCallback **Callback** = **null**;  
 **private** String **WWWResponse**;  
  
 **private** SimpleDateFormat **timeStamp** = **new** SimpleDateFormat(**"HH:mm:ss"**);  
 **private** Timer **REtimer** = **new** Timer();  
  
  
  
 **public** Register(Network n) {  
 **netWork** = n;  
 }  
  
 **public void** Response(String p) {  
 **mSelf**.close();  
 Date dateobj = **new** Date();  
 **boolean** js = *queue*.offer(p);  
 *// boolean js = queue.add(p);* **if** (js) **mSelf**.E(**"addqueue "** + **timeStamp**.format(dateobj) + **"ture"**);  
 **else mSelf**.E(**"addqueue "** + **timeStamp**.format(dateobj) + **"false and queue full"**);  
 }  
  
 **public boolean** isRegSuccess() {  
  
 **return RegSuccess**;  
 }  
  
 @Override  
 **public void** run() {  
 **while** (**Wokring**) {  
 **try** {  
 **if** (!*queue*.isEmpty()) {  
 **ReceivePacket** = *queue*.poll();  
 **if** (isbelongtoReg(**ReceivePacket**))  
 **if** (is401(**ReceivePacket**) && !**RepeatPacketcome**) {  
 Date dateobj = **new** Date();  
 **RepeatPacketcome** = **true**;  
 **mSelf**.E(**"removce success "** + **timeStamp**.format(dateobj));  
 handle401(**ReceivePacket**);  
 } **else if** (is200()) {  
 **RegSuccess** = **true**;  
 Stop();  
 **this**.interrupt();  
 SetRegtimer();  
 }  
 }  
 } **catch** (NullPointerException ex) {  
 System.***out***.println(ex);  
 }  
 }  
 **mSelf**.E(**"Reg close"**);  
  
 }  
  
 **private boolean** is200() {  
 **return ReceivePacket**.contains(**"200 OK"**);  
 }  
  
 **private void** SetRegtimer() {  
 **long** period=(**long**)(**Expires** \* **coffet**) \* 1000;  
 **REtimer**.schedule(**new** ReRigsterTimer(), period,period);  
 }  
  
 **private boolean** is401(String receivePacket) {  
 **return** receivePacket.contains(**"401 Unauthorized"**);  
 }  
  
 **public void** Stop() {  
 **Wokring** = **false**;  
 }  
  
 **public void** setRegisterListener(RegisterAgentCallback Callback) {  
 **this**.**Callback** = Callback;  
 }  
  
 **public void** setAccount(String s) {  
 **mAccount** = s;  
 }  
  
 **public void** doReg() {  
 **RepeatPacketcome** = **false**;  
 **RegSuccess** = **false**;  
 **RegTimes** = 1;  
 **builder** = **new** StringBuilder();  
 **AuthPacket** = **new** StringBuilder();  
 **Reg\_packetMaker** = **new** SIPHeaderParameter();  
 **mip** = **netWork**.getLocalIP();  
 **mport** = Integer.*toString*(**netWork**.getLocalPort());  
 **ServerIP** = **netWork**.getRemoteIP();  
 startLine();  
 regPacketProcess();  
 **netWork**.sendUdp(**builder**.toString());  
  
 }  
  
 *///private* **private** String Getnonce(String p) {  
 **int** n = p.indexOf(**"nonce="**);  
 **int** al = p.indexOf(**"algorithm"**);  
 String nodns = p.substring(n, al);  
 String[] h = nodns.split(**"\""**);  
 **return** h[1];  
 }  
  
 **private boolean** isbelongtoReg(String receivePacket) {  
 **return** receivePacket.contains(**currrntid**);  
 }  
  
 **private void** handle401(String p) {  
 **nonce** = Getnonce(p);  
 **WWWResponse** = WWWAuthResponse(**nonce**);  
 **AuthPacket**.append(**"Authorization: Digest "** + **"username=\""** + **mAccount** + **"\","** +  
 **"realm=\""** + **ServerIP** + **"\","** + **"nonce=\""** + **nonce** + **"\","** + **"uri=\""** +  
 **"sip:"** + **ServerIP** + **"\","** + **"response=\""** + **WWWResponse** + **"\","** + **"algorithm=md5"** + **"\r\n"**);  
 **AuthPacket**.append(**"CSeq: "** + Integer.*toString*(**RegTimes** + 1) + **" "** + **"REGISTER"** + **"\r\n"**);  
 **AuthPacket**.append(**"User-Agent:"** + *UAgent* + **"\r\n"**);  
 **AuthPacket**.append(**"Content-Length: 0"** + **"\r\n"**);  
 **AuthPacket**.append(**"\r\n"**);  
 **netWork**.sendUdp(**AuthPacket**.toString());  
  
 Date dateobj = **new** Date();  
 **mSelf**.E(**"finish"** + **timeStamp**.format(dateobj));  
  
 }  
  
 **private void** startLine() {  
 **builder**.append(***Startline***);  
 **builder**.append(**ServerIP** + **" "**);  
 **builder**.append(**"SIP/2.0\r\n"**);  
 }  
  
 **private void** regPacketProcess() {  
 **builder**.append(**"Via: SIP/2.0/UDP "** + **mip** + **":"** + **mport** + **";branch="** + **Reg\_packetMaker**.branch() + **"\r\n"**);  
 **builder**.append(**"From: <sip:"** + **mAccount** + **"@"** + **ServerIP** + **">;"** + **"tag="** + **Reg\_packetMaker**.Integer() + **"\r\n"**);  
 **builder**.append(**"To: <sip:"** + **mAccount** + **"@"** + **ServerIP** + **">"** + **"\r\n"**);  
 **currrntid** = **Reg\_packetMaker**.Integer();  
 **builder**.append(**"Call-ID: "** + **currrntid** + **"@"** + **mip** + **"\r\n"**);  
 **builder**.append(**"Expires:"** + Integer.*toString*(**Expires**) + **"\r\n"**);  
 **builder**.append(**"Max-Forwards: "** + **"70"** + **"\r\n"**);  
 **builder**.append(**"Contact: "** + **"<sip:"** + **mAccount** + **"@"** + **mip** + **":"** + **mport** + **">"** + **"\r\n"**);  
  
 **AuthPacket**.append(**builder**.toString());  
 **builder**.append(**"CSeq: "** + Integer.*toString*(**RegTimes**) + **" "** + **"REGISTER"** + **"\r\n"**);  
 **builder**.append(**"User-Agent:"** + *UAgent* + **"\r\n"**);  
 **builder**.append(**"Content-Length: 0"** + **"\r\n"**);  
 **builder**.append(**"\r\n"**);  
 }  
  
 **private** String WWWAuthResponse(String nonce) {  
 String url = **"sip:"** + **ServerIP**;  
 String a1 = **mAccount** + **":"** + **ServerIP** + **":"** + **password**;  
 String HA1 = getMD5(a1);  
 String a2 = **"REGISTER"** + **":"** + url;  
 String HA2 = getMD5(a2);  
 String re = HA1 + **":"** + nonce + **":"** + HA2;  
 String f = getMD5(re);  
 **return** f;  
 }  
  
 **private** String getMD5(String input) {  
 **byte**[] source;  
 **try** {  
 *//Get byte according by specified coding.* source = input.getBytes(**"UTF-8"**);  
 } **catch** (UnsupportedEncodingException e) {  
 source = input.getBytes();  
 }  
 String result = **null**;  
 **char** hexDigits[] = {**'0'**, **'1'**, **'2'**, **'3'**, **'4'**, **'5'**, **'6'**, **'7'**, **'8'**, **'9'**, **'a'**, **'b'**, **'c'**, **'d'**, **'e'**, **'f'**};  
 **try** {  
 MessageDigest md = MessageDigest.*getInstance*(**"MD5"**);  
 md.update(source);  
 *//The result should be one 128 integer* **byte** temp[] = md.digest();  
 **char** str[] = **new char**[16 \* 2];  
 **int** k = 0;  
 **for** (**int** i = 0; i < 16; i++) {  
 **byte** byte0 = temp[i];  
 str[k++] = hexDigits[byte0 >>> 4 & 0xf];  
 str[k++] = hexDigits[byte0 & 0xf];  
 }  
 result = **new** String(str);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** result;  
 }  
  
 **public interface** RegisterAgentCallback {  
 **void** PacketListener(String message, **int** messageLength);  
 }  
  
 **class** ReRigsterTimer **extends** TimerTask {  
 @Override  
 **public void** run() {  
 doReg();  
 }  
  
 }  
  
}

* + 1. SIPHeaderParameter

**package** com.sample.android.sip.sipphone;  
  
  
**import** java.util.ArrayList;  
**import** java.util.Iterator;  
**import** java.util.Random;  
  
  
  
**public class** SIPHeaderParameter {  
  
 **private** StringBuilder **builder**=**new** StringBuilder();  
  
 **public** String branch() {  
  
 **builder** = **new** StringBuilder();  
 **builder**.append(**"z9hG4bK"**);  
 **builder**.append(getrand(**"ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890"**));  
 **return builder**.toString();  
 }  
  
 **public** String Integer() {  
 **builder** = **new** StringBuilder();  
 **builder**.append(getrand(**"123456789"**));  
 **return builder**.toString();  
  
 }  
  
 **private** String getrand(String s) {  
 StringBuilder salt = **new** StringBuilder();  
 Random rnd = **new** Random();  
 **while** (salt.length() < 8) {  
 **int** index = (**int**) (rnd.nextFloat() \* s.length());  
 salt.append(s.charAt(index));  
 }  
 String saltStr = salt.toString();  
 **return** saltStr;  
 }  
}

1. **Snapshots of testing result**
   1. **Snapshots of testing result**



* 1. **Unit Testing Code Listing**

**import static** org.junit.Assert.\*;  
  
**public class** Test {  
 String **G**=**"INVITE sip:7170@iptel.org SIP/2.0\r\n"**+  
 **"Via: SIP/2.0/UDP 195.37.77.100:5040;rport\r\n"**+  
 **"Max-Forwards: 10\r\n"**+  
 **"From: \"jiri\" <sip:jiri@iptel.org>;tag=76ff7a07-c091-4192-84a0-d56e91fe104f\r\n"**+  
 **"To: <sip:jiri@bat.iptel.org>\r\n"**+  
 **"Call-ID: d10815e0-bf17-4afa-8412-d9130a793d96@213.20.128.35\r\n"**+  
 **"CSeq: 2 INVITE\r\n"**+  
 **"Contact: <sip:213.20.128.35:9315>\r\n"**+  
 **"User-Agent: Windows RTC/1.0\r\n"**+  
 **"Proxy-Authorization: Digest username=\"jiri\", realm=\"iptel.org\","** +**"algorithm=\"MD5\", uri=\"sip:jiri@bat.iptel.org\""** +**"nonce=\"3cef753900000001771328f5ae1b8b7f0d742da1feb5753c\","** +**"response=\"53fe98db10e1074b03b3e06438bda70f\"\r\n"** +**"Content-Type: application/sdp\r\n"** +**"Content-Length: 451\r\n"**;  
  
 Parser **p**=**new** Parser();  
 @org.junit.Test  
 **public void** getPacketType() {  
 **p**.pushData(**G**);  
 assertEquals(**"REQUEST"**,**p**.getPacketType());  
 }  
 @org.junit.Test  
 **public void** getPacketResonseNum() {  
 **p**.pushData(**G**);  
 assertEquals(**""**,**p**.getPacketResonseNum());  
 }  
 @org.junit.Test  
 **public void** getPacketRequestMethod() {  
 **p**.pushData(**G**);  
 assertEquals(**"INVITE"**,**p**.getPacketRequestMethod());  
 }  
 @org.junit.Test  
 **public void** getFromName() {  
 **p**.pushData(**G**);  
 assertEquals(**"REQUEST"**,**p**.getPacketType());  
 }  
 @org.junit.Test  
 **public void** getFromNameAddr() {  
 **p**.pushData(**G**);  
 assertEquals(**"jiri@iptel.org"**,**p**.getFromNameAddr());  
 }  
 @org.junit.Test  
 **public void** getFromTag() {  
 **p**.pushData(**G**);  
 assertEquals(**"76ff7a07-c091-4192-84a0-d56e91fe104f"**,**p**.getFromTag());  
 }  
 @org.junit.Test  
 **public void** getToName() {  
 **p**.pushData(**G**);  
 assertEquals(**""**,**p**.getToName());  
 }  
 @org.junit.Test  
 **public void** getToNameAddr() {  
 **p**.pushData(**G**);  
 assertEquals(**"jiri@bat.iptel.org"**,**p**.getToNameAddr());  
 }  
 @org.junit.Test  
 **public void** getToTag() {  
 **p**.pushData(**G**);  
 assertEquals(**""**,**p**.getToTag());  
 }  
 @org.junit.Test  
 **public void** getCallID() {  
 **p**.pushData(**G**);  
 assertEquals(**"d10815e0-bf17-4afa-8412-d9130a793d96@213.20.128.35"**,**p**.getCallID());  
 }  
 @org.junit.Test  
 **public void** getMaxForward() {  
 **p**.pushData(**G**);  
 assertEquals(**"10"**,**p**.getMaxForward());  
 }  
 @org.junit.Test  
 **public void** getcSeqValue() {  
 **p**.pushData(**G**);  
 assertEquals(**"2"**,**p**.getcSeqValue());  
 }  
 @org.junit.Test  
 **public void** getcSeqMethod() {  
 **p**.pushData(**G**);  
 assertEquals(**"INVITE"**,**p**.getcSeqMethod());  
 }  
 @org.junit.Test  
 **public void** getContentType() {  
 **p**.pushData(**G**);  
 assertEquals(**"application/sdp"**,**p**.getContentType());  
 }  
 @org.junit.Test  
 **public void** getContentLength() {  
 **p**.pushData(**G**);  
 assertEquals(**"451"**,**p**.getContentLength());  
 }  
 @org.junit.Test  
 **public void** getAuthorRealm() {  
 **p**.pushData(**G**);  
 assertEquals(**""**,**p**.getAuthorRealm());  
 }  
 @org.junit.Test  
 **public void** getAuthorNonce() {  
 **p**.pushData(**G**);  
 assertEquals(**""**,**p**.getAuthorNonce());  
 }  
 @org.junit.Test  
 **public void** getProxyAuthorRealm() {  
 **p**.pushData(**G**);  
 assertEquals(**"iptel.org"**,**p**.getProxyAuthorRealm());  
 }  
 @org.junit.Test  
 **public void** getProxyAuthorNonce() {  
 **p**.pushData(**G**);  
 assertEquals(**"3cef753900000001771328f5ae1b8b7f0d742da1feb5753c"**,**p**.getProxyAuthorNonce());  
 }  
}

1. **Measurement**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| HW #5 | | | | | |
| 104318070 林侃賦 | | 104318079 何柏寬 | | 104318144 周冠亨 | |
| 2016/05/02 17:00~21:00 | 4H | 2016/05/04 19:00~22:00 | 3H | 2016/05/04 17:00~22:00 | 3H |
| 2016/05/04 20:00~24:00 | 4H | 2016/05/05 17:00~22:00 | 5H | 2016/05/05 20:00~23:00 | 5H |
| Total | 8H | Total | 8H | Total | 8H |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| HW #4 | | | | | |
| 104318070 林侃賦 | | 104318079 何柏寬 | | 104318144 周冠亨 | |
| 2016/04/25 18:00~22:00 | 4H | 2016/04/26 19:00~22:00 | 3H | 2016/04/25 17:00~22:00 | 3H |
| 2016/04/29 19:00~23:00 | 4H | 2016/04/29 17:00~22:00 | 5H | 2016/04/28 20:00~23:00 | 5H |
| Total | 8H | Total | 8H | Total | 8H |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| HW #3 | | | | | |
| 104318070 林侃賦 | | 104318079 何柏寬 | | 104318144 周冠亨 | |
| 2016/03/30 18:00~22:00 | 4H | 2016/03/30 19:00~22:00 | 3H | 2016/03/30 17:00~22:00 | 3H |
| 2016/03/31 19:00~23:00 | 4H | 2016/03/31 17:00~22:00 | 5H | 2016/03/31 20:00~23:00 | 5H |
| Total | 8H | Total | 8H | Total | 8H |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| HW #2 | | | | | |
| 104318070 林侃賦 | | 104318079 何柏寬 | | 104318144 周冠亨 | |
| 2016/03/15 12:00~17:00 | 5H | 2016/03/17 20:00~22:00 | 2H | 2016/03/15 17:00~22:00 | 5H |
| 2016/03/17 20:00~23:00 | 3H | 2016/03/18 17:00~23:00 | 6H | 2016/03/17 20:00~23:00 | 3H |
| Total | 8H | Total | 8H | Total | 8H |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| HW #1 | | | | | |
| 104318070 林侃賦 | | 104318079 何柏寬 | | 104318144 周冠亨 | |
| 2016/02/28 10:00~18:00 | 8H | 2016/02/28 10:00~18:00 | 8H | 2016/02/28 10:00~18:00 | 8H |
| Total | 8H | Total | 8H | Total | 8H |