Alma Mater Studiorum · Università di Bologna

facoltà di scienze matematiche, fisiche e naturali
Corso di Laurea Triennale in Informatica

Interfaccia user-friendly per applicazioni VoIP in ambiente Android

Tesi di Laurea in Architettura degli Elaboratori

Relatore:
Chiar.mo Prof.
Vittorio Ghini

Presentata da:
Paolo Torresi

Seconda Sessione
Anno 2009/2010

INDICE

Introduzione

Il 1969 è stato un anno importantissimo nella storia dell'uomo e sicuramente un periodo fatto di scoperte rivoluzionarie per il mondo intero. Se, infatti, da un lato lo sbarco dell'uomo sulla luna ha fatto sognare milioni di persone, dall'altro la nascita di ARPANET, forma embrionale di internet, ha rivoluzionato l'idea di comunicazione tra individui distanti, quasi come la scoperta del telefono. Negli ultimi trent'anni l'area di maggior sviluppo della rete internet riguarda l'area della comunicazione mediante tale servizio. Se da un lato abbiamo visto, quindi, l'espansione dell'utilizzo della posta elettronica, streaming audio-visivi e servizi utilizzabili direttamente da browser web, dall'altro l'utilizzo di servizi VoIP stanno modificando totalmente l'idea di comunicazione tra utenti. Partendo appunto da applicazioni VoIP e cercando l'elemento di congiunzione tra quest'ultimo e le nuove tecnologie mobili, basate sul nuovo sistema operativo Android, ho cercato nella mia tesi di creare un'interfaccia veloce e fruibile da tutti per un'applicazione VoIP per ambienti Android.

Il creare una nuova interfaccia prende spunto dall'esigenza di programmare uno strumento che faciliti l'utilizzo delle applicazioni VoIP e ne consenta un uso agevole anche ai non esperti del settore. Condizione imprescindibile per la messa a punto del mio progetto è, quindi, la realizzazione di un applicazione che riesca a conciliare immediatezza d'uso e chiarezza espositiva. Curare l'aspetto grafico è, infatti, estremamente importante, in quanto la semplicità dell'uso passa anche attraverso una semplificazione grafica. Programmare un'interfaccia che risulti innanzitutto ordinata e ben organizzata dal punto di vista estetico, significa attuare una prima ed immediata semplificazione operativa. L'innovazione della mia tesi non consiste, infatti, nella creazione di nuove funzionalità per le applicazioni VoIP, ma nella programmazione di un'interfaccia che, mantenendo le potenzialità di complesse applicazioni VoIP già esistenti, garantisca una maggiore e più eterogenea fruibilità.

A partire da un progetto VoIP (Sipdroid) già esistente, contenente la rubrica ed il registro delle chiamate, ho deciso di sostituire le suddette caratteristiche, implementandole.

Ho, quindi, sovrapposto le mie “features” a quelle preesistenti, in modo da garantirne una maggiore compatibilità al mio programma di interfaccia VoIP. Grazie ad una maggiore compatibilità e ad un processo di semplificazione di alcune caratteristiche contenute in Sipdroid, ho cercato di aumentare la velocità delle prestazioni delle mie applicazioni. Eliminando alcune delle funzionalità presenti in Sipdroid (bluethoot, webcam), ho preferito concentrarmi sulla buona applicabilità delle funzioni di base, magari a discapito della molteplicità delle funzioni. Questo per ricollegarmi all'obbiettivo primario della tesi, ossia creare un'interfaccia VoIP che consenta un utilizzo immediato.

Per poter realizzare gli obiettivi sopra esposti, mi sono dedicato in via preliminare allo studio dell'ambiente di sviluppo e del linguaggio necessario alla programmazione in Android. Questo mi ha consentito di acquisire le nozioni di base, indispensabili alla programmazione della mia interfaccia. Ho dovuto, inoltre, installare l'emulatore di un dispositivo mobile Android tale da testare il corretto funzionamento delle applicazioni create.

Mi sono dedicato successivamente alla ricerca di un'applicazione VoIP in ambiente Android, che fosse open-source e funzionante. Dopo averla trovata e testata la compatibilità, ho proceduto con l'eliminazione delle componenti superflue del progetto Sipdroid e con l'aggiunta della mia interfaccia. In fase conclusiva ho testato il corretto funzionamento della suddetta interfaccia.

In fase espositiva, per rendere la spiegazione del progetto altrettanto agevole ed immediata, ho ritenuto opportuno suddividere la presente tesi in quattro capitoli.

Il primo, intitolato *Scenario*, e a sua volta suddiviso in due sotto-capitoli, contiene alcune informazioni di carattere teorico riguardante il VoIP e il suo funzionamento, nonché il sistema operativo Android e la sua architettura. Il paragrafo dedicato ad Android è a sua volta suddiviso in due sotto-capitoli: il middleware e l'architettura di Android. Quest'ultimo contiene a sua volta l'elenco esplicato delle varie componenti dell'architettura Android: applicazioni, framework, librerie, runtime e kernel linux.

Il secondo capitolo, intitolato *Obiettivo*, esplica più dettagliatamente gli obiettivi che mi sono prefissato in fase di progettazione della tesi, già esposti sinteticamente nella presente introduzione. Tale capitolo è suddiviso in quattro sotto-capitoli di cui il primo

contiene alcune delle considerazioni preliminari all'utilizzo del sistema operativo Android; il secondo elenca gli strumenti utilizzati, il terzo si concentra sulla descrizione dell'emulatore Android e delle sue componenti software; il quarto, infine, introduco il centralino VoIP PBX.

Il terzo capitolo, intitolato *Progettazione*, può essere definito il nucleo della tesi in quanto è dedicato all'analisi delle varie fasi della programmazione della mia interfaccia. Esso è a sua volta diviso in sei sotto-capitoli: il primo contiene alcuni accenni alla programmazione di applicazioni per dispositivi Android; il secondo consiste in una valutazione del progetto, che ha voluto essere il più possibile obiettiva; il terzo espone le scelte progettuali; il quarto contiene i risultati della ricerca operata in relazione ad una applicazione VoIP, da cui ho ricavato il progetto Sipdroid, successivamente esplicato; il quinto, invece, espone il servizio PBXes, ossia un centralino VoIP necessario al corretto funzionamento del programma Sipdroid; il sesto, infine, descrive lo stack SIP, utilizzato dal programma Sipdroid, denominato MjSip.

Il quarto ed ultimo capitolo, intitolato *Problematiche riscontrate e soluzioni adottate*, tratta delle difficoltà incontrate nello svolgimento del progetto e delle soluzioni di volta in volta adottate al riguardo. Esso è a sua volta suddiviso in tre sotto-capitoli: il primo contiene le modalità di salvataggio dei contatti e del registro chiamate; il secondo accenna alle difficoltà incontrate nella configurazione del servizio PBXes; il terzo propone una possibile alternativa all'utilizzo del servizio PBXes (Asterisk); mentre il quarto ed ultimo espone, sinteticamente, delle possibili migliorie al progetto in questione.

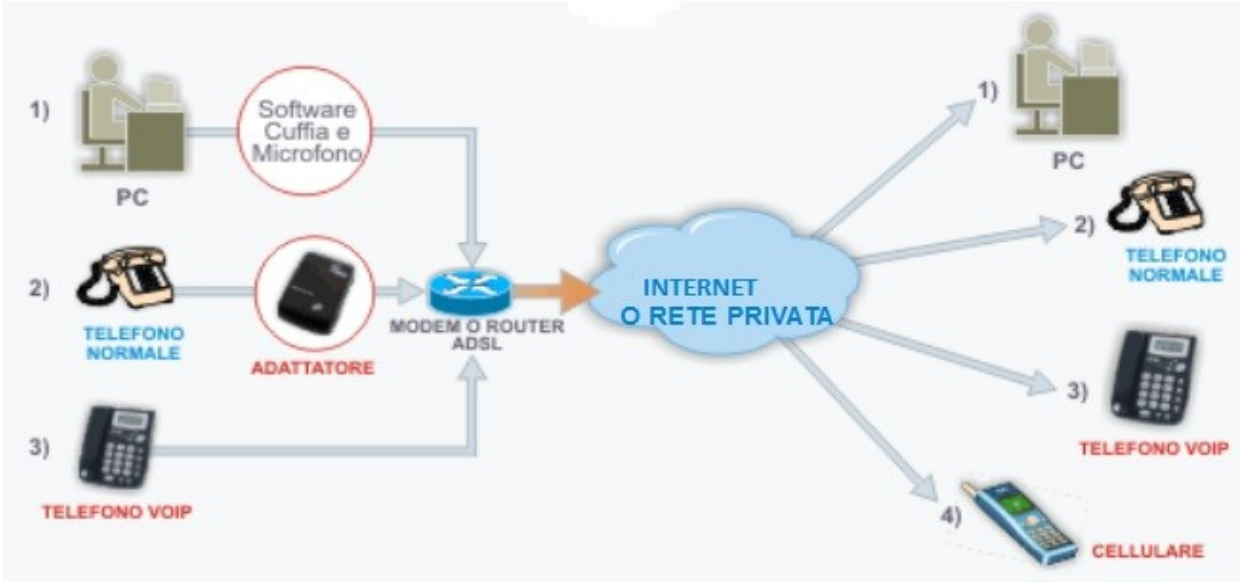
Le note conclusive saranno in parte dedicate alla valutazione dei risultati raggiunti ed in parte alla conseguente valutazione dell'esperienza di programmazione effettuata. Sempre nella conclusione mi chiederò, infatti, se i risultati attesi sono conformi ai risultati raggiunti e se questa esperienza di programmazione autonoma mi ha coinvolto e ha soddisfatto le mie aspettative. Tenterò di fare, infine, una valutazione, il più possibile obiettiva, delle competenze maturate nel corso degli studi universitari, che mi sono state utili per la realizzazione del suddetto progetto.

Capitolo primo

Scenario

1.1 VoIP e suo funzionamento

Il Voice over IP o VoIP è una tecnologia che permette di effettuare una conversazione telefonica utilizzando una connessione Internet o un'altra rete dedicata che utilizza il protocollo IP. Per VoIP si intende l'insieme dei protocolli di comunicazione di strato applicativo che rende possibile la suddetta tipologia di comunicazione. Per mezzo di un provider VoIP è possibile altresì effettuare telefonate anche verso la rete tradizionale, fissa o mobile. La tecnologia VoIP è utilizzata in qualsiasi sistema atto a scambiare dati tra due o più dispositivi. La modalità di funzionamento del VoIP consiste nell'instradare sulla rete pacchetti di dati, codificati in formato digitale, contenenti le informazioni necessaria alla comunicazione tra le parti.



Le comunicazioni VoIP hanno molti aspetti positivi, tra i quali è opportuno sottolineare:

- minore costo per chiamata, talvolta nullo, soprattutto per quanto riguarda le grandi distanze (telefonate internazionali)
- minori costi delle infrastrutture:creato un sistema di scambio dati, è possibile utilizzarlo come mezzo di comunicazione
- nuove funzionalità avanzate come, ad esempio, la video chiamata
- l'implementazione di future opzioni non comporterà la sostituzione dell'hardware, ma l'aggiornamento o la sostituzione degli elementi software.

Le comunicazioni tramite VoIP non hanno necessariamente bisogno di un server riservato su internet, come i provider, ma possono essere effettuate anche sfruttando reti private, siano esse infrastrutturali, con l'ausilio di router e cavi, oppure wireless, per mezzo di un access point privato o tramite connessioni adHoc tra dispositivi: il VoIP è essenzialmente una comunicazione fatta mediante pacchetti dati, quindi un server esterno dà solo la possibilità di rintracciare dispositivi esterni alla propria rete privata e spesso irraggiungibili senza il suo ausilio. Il server esterno, inoltre, permette di comunicare con dispositivi tradizionali, che non sarebbero altrimenti collegabili con dispositivi VoIP.

I protocolli usati per codificare e trasmettere le conversazioni VoIP sono chiamati Voice over IP protocols. Il vantaggio nell'adottare questa tecnologia consiste quindi sia nell'economicità della struttura, qualsiasi struttura di rete preesistente può essere riadattata per il VoIP, sia nell'economicità del servizio. Spesso il servizio tra dispositivi VoIP è gratuito, escludendo i costi di utilizzo della rete.

In ambito aziendale la stessa rete di comunicazione dati può essere utilizzata anche per le comunicazioni vocali aumentando così il livello di interazione tra uffici tra loro distanti. È necessario ricordare che, se in precedenza veniva utilizzata a pieno una connessione dati, essa potrebbe generare problemi di saturazione della banda con l'aggiunta della comunicazione VoIP.

In ambito privato l'adozione del VoIP annulla le spese di comunicazione tra dispositivi VoIP e offre tariffe particolarmente vantaggiose per chiamate a circuiti tradizionali, soprattutto per chiamate internazionali. A tal proposito sarà comunque necessario un collegamento internet a banda larga che garantisca un servizio apprezzabile. In rari casi la comunicazione tra dispositivi VoIP presenta dei costi dipendenti dal provider in uso. Vorrei riportare come esempio la scelta fatta dalla società di comunicazione "3 Italia". Quest'ultima permette, ad utenti in possesso del dispositivo mobile "3 Skypephone", di telefonare ad altri utenti per mezzo del servizio VoIP in maniera gratuita per 600 minuti al giorno, ulteriori ore di comunicazione telefonica giornaliera saranno a pagamento.

La tecnologia VoIP necessita di due protocolli di comunicazione paralleli: uno per il trasporto dei dati (pacchetti voce), un altro per la "segnalazione" della conversazione (ricostruzione del frame audio, sincronizzazione, identificazione delle chiamante, etc…). Per il trasporto dei dati, nella maggior parte delle implementazioni VoIP, viene adottato il protocollo RTP (Real-time Transport Protocol). Questo protocollo viene comunemente scelto in servizi necessitanti di trasferimenti in tempo reale ed è normalmente basato sul protocollo UDP (User Datagram Protocol). Per la seconda tipologia di protocollo, che interessa la telefonia via Internet, il processo di standardizzazione non è ancora concluso. Al momento, la gestione delle chiamate voce sulla rete IP si orienta su due differenti proposte, elaborate in ambito ITU (International Telecommunications Union) e IETF (Internet Engineering Task Force), che sono rispettivamente H.323 e SIP (Session Initiation Protocol), quest'ultimo utilizzato per il mio progetto.

Le tecnologie di compressione vocale utilizzano una banda che varia dai 4 kbit/sec agli 82 kbit/sec (nei formati di compressione meno efficienti). La voce umana ha uno spettro di circa 2,7 kHz che richiede una larghezza di banda di almeno 5,4 kHz. Possiamo, quindi, ricostruire un segnale vocale dividendo un suono in valori discreti e mettendoli in successione secondo una scala temporale. Ad un segnale analogico di X Hz di banda corrisponde un segnale digitale di 2*X*N bit/sec, troppo grande per essere trasmesso in totale in una banda fornita da una linea di comunicazione media. Con i formati di compressione, tuttavia, diventa possibile ridurre drasticamente la banda richiesta; la

compressione interviene dopo la digitalizzazione della voce e prima del suo invio, poiché è impossibile comprimere un segnale analogico.

Esistono frequenze del segnale digitale anche superiori alla frequenza della voce naturale; il segnale analogico rimane, tuttavia, il limite qualitativo da raggiungere per il campionamento digitale, poiché il salvataggio e la trasmissione di altre frequenze non sarebbero percepibili dall'orecchio umano e, quindi, non aggiungerebbero informazioni al segnale di partenza.

Per la trasmissione di video le tecnologie attuali richiedono una larghezza di banda di almeno 50 kbit/sec., per evitare lo “sfarfallio” e/o una pessiama risoluzione delle immagini.

Una chiamata via Internet richiede una banda maggiore rispetto alla frequenza di campionamento della voce sia in upstream, dati che inviamo nella rete, che in downstream, dati che riceviamo dalla rete, essendo la comunicazione telefonica bidirezionale, detta anche full duplex. Se ciò non avviene, uno dei due interlocutori riceverà i pacchetti in un ordine scorretto e ascolterà parole senza senso, con sillabe accostate non nella giusta sequenza. Con il trasferimento di elementi “statici” come pagine Web o file, il modem ha la possibilità di fare dei controlli, ricostruire o chiedere il rinvio di pacchetti danneggiati o mai arrivati a destinazione. Non esiste, invece, metodi o controlli che possono risolvere i difetti di trasmissione vocale.

Utilizzando un adeguato protocollo di comunicazione, ossia un format di compressione vocale che limiti il campionamento intorno ai 12-13 kbit/sec. è possibile, con un pacchetto-voce trasmesso al secondo, avere pacchetti di circa 1,5 Kbyte, ossia al di sotto della soglia critica che creerebbe problemi con la connessione Internet. Con una linea ISDN si dispone di una connessione simmetrica su due linee: la velocità di trasferimento dati è la medesima sia in entrata che in ricezione. Con una larghezza di banda di 64 kbit/sec., impegnando anche una sola linea, è possibile stabilire una chiamata VoIP anche con formati di compressione vocale meno efficienti, ossia che campionano a 50-60 kbit/sec., sempre in riferimento a pari valore di indice di qualità della riproduzione. Con l'utilizzo dei due canali ISDN, quindi, è possibile gestire due chiamate VoIP contemporaneamente.

Dal punto di vista operativo, per poter effettuare chiamate VoIP, oltre a disporre di un programma di Voice Over IP, occorre installare un codec di compressione della voce e dovrà essere installato ed operativo su tutti i dispositivi che vogliono utilizzarlo. Fra questi codec rientrano: GSM 6.10, ILBC, Speex 15.2k, Speex 8.0k, G.711, G.723, G.729, G.771 A-Law, G.771 U-Law.

Concludendo, il servizio VoIP può essere un ottima alternativa alle comunicazioni tradizionali, quali l'ausilio della rete fissa o mobile GSM, poiché permette costi contenuti, talvolta nulli, e garantisce un soddisfacente livello di comunicazione vocale e video, sia per privati sia per aziende.

1.2 Android e sua architettura

Android è una piattaforma per dispositivi mobili che include sistema operativo, middleware e applicazioni di base.

Il middleware

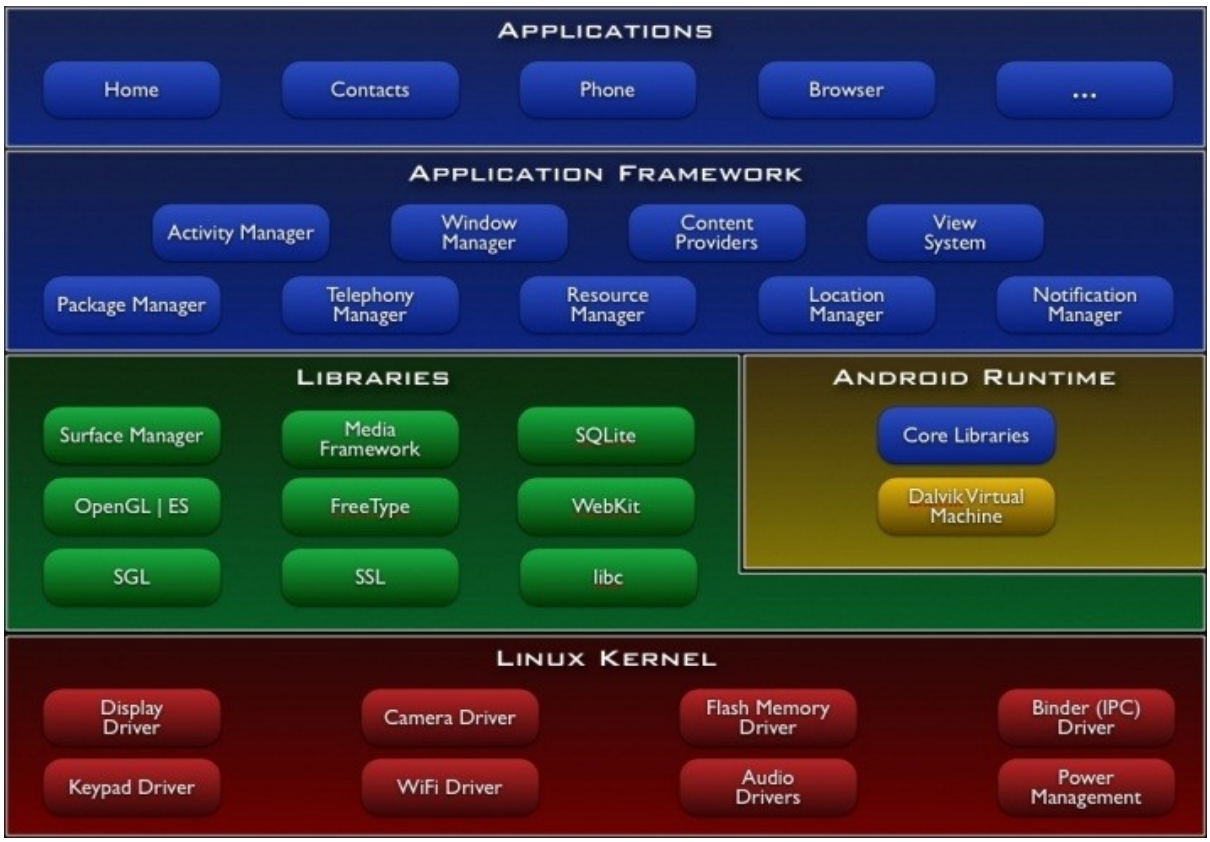
Il Middleware nasce circa dieci anni fa per indicare, in maniera generica, un'entità (programma, protocollo o altro) che si interpone principalmente tra l'hardware della macchina e il software applicativo. Il termine si è poi evoluto nel tempo assieme agli stessi concetti di applicazione e sistema operativo, assumendo varie connotazioni e/o interpretazioni a seconda dell'implementazione effettiva di un middleware. A prescindere dalla collocazione protocollare nello standard OSI, si può pensare ad un middleware Layer come a tutto ciò che si interpone tra il sistema operativo (quindi software di basso livello che controlla direttamente l'hardware) e i software applicativi di alto livello, garantendo principalmente l'indipendenza tra questi due livelli.

Un protocollo di Middleware fornisce un preciso modello di programmazione per la generazione di applicazioni distribuite, sollevando il programmatore dalla necessità di occuparsi di tutti i dettagli di basso livello della comunicazione e lo scambio di messaggi tra processi distribuiti.

Architettura di Android

Un sistema operativo è un programma software in grado di interfacciare le applicazioni con l'hardware su cui è installato, garantire un certo grado di sicurezza operativo per mezzo di strutture di controllo e dare strutture suppletive utilizzabili dalle applicazioni di alto livello.

Il diagramma seguente mostra le componenti principali del sistema operativo di Android. Ogni sezione sarà descritta in dettaglio di seguito.



Applicazioni

Android funziona con un set di applicazioni di base che comprende un e-mail client, un programma SMS, calendario, mappe, browser, contatti e altro. Tutte le applicazioni sono scritte in linguaggio Java.

Framework per applicazioni

Gli sviluppatori hanno pieno accesso alle stesse framework API usate dalle applicazioni di base. L'architettura delle applicazioni è progettata per semplificare il riutilizzo dei componenti; ogni applicazione può rendere pubbliche le sue capacità, così che tutte le altre applicazioni possano farne uso, ma sono soggette ai limiti imposti dalla sicurezza del framework. Per acquisire permessi speciali ad API e componenti protetti, è necessario aggiornare e modificare il file "AndroidManifest" della nostra applicazione. Questo stesso meccanismo consente all'utente di sostituire i componenti standard con versioni personalizzate.

Alla base di ogni applicazione si trova un set di servizi tra cui:

- un gruppo ricco ed estensibile di Viste che possono essere usate per costruire un'applicazione; esso contiene liste, caselle di testo, pulsanti, e addirittura un browser web integrato
- dei Content Providers che permettono alle applicazioni di accedere a dati da altre applicazioni (come i Contatti), o di condividere i propri dati
- un Manager delle risorse, che offre l'accesso a risorse non-code come strings localizzate, grafica, files di layout
- un Manager delle notifiche, che permette a tutte le applicazioni di mostrare avvisi personalizzati nella status bar
- un Manager delle attività, che gestisce il ciclo di vita delle applicazioni e consente un backstack di navigazione comune

Librerie

Android comprende un set di librerie C/C++ usate da vari componenti del sistema di Android. Tali elementi sono presentati allo sviluppatore attraverso il framework per applicazioni di Android.

Ecco alcune delle principali librerie:

- System C library, un'implementazione BSD-derived della libreria standard C system (libc), disegnata per dispositivi basati su Linux
- Media Libraries, basate sull'OpenCORE di PacketVideo. Le librerie supportano la riproduzione e la registrazione di formati audio e video (anche molto popolari), compresi file di immagini, come MPEG4, H.264, MP3, AAC, AMR, JPG, e PNG
- Surface Manager, gestisce l'accesso al display subsystem e compone layer grafici 2D e 3D da applicazioni multiple
- LibWebCore, un motore di browser moderno che fa funzionare sia il browser Android sia la visualizzazione web implementata
- SGL, il motore grafico 2D sottostante
- 3D libraries, un'implementazione basata su APIs OpenGL ES 1.0. Le librerie usano sia accelerazione hardware 3D (quando disponibile) sia quella inclusa, un rasterizer software 3D altamente ottimizzato
- FreeType, rendering di bitmap e vector font
- SQLite, un motore di database relazionale potente e leggero, disponibile per tutte le applicazioni

Runtime

Android comprende un set di librerie centrali che fornisce la maggior parte delle funzionalità disponibili nelle librerie di base del linguaggio di programmazione Java.

Kernel Linux

Android si appoggia sulla versione 2.6 di Linux per servizi del sistema centrale, come sicurezza, gestione della memoria, esecuzione, network stack e driver model. Il kernel funziona anche da abstraction layer tra l'hardware e il resto del software.

Capitolo secondo

Obiettivo

2.1 Considerazioni preliminari

L'utilizzo di componenti elettronici portatili vengono sempre più utilizzati nella vita di tutti i giorni, sia in ambito privato sia in quello lavorativo.

Android è un Sistema Operativo Open Source ideato dalla società Google e viene sempre più scelto da varie case di costruzione di apparecchi elettronici. Il VoIP è, inoltre, una tecnologia di comunicazione sempre più in espansione e in futuro potrebbe diventare l'unico mezzo di comunicazione. Le applicazioni attuali VoIP, che permettono la comunicazione per mezzo di un dispositivo Android, sono ancora poche e spesso non facili da configurare e da utilizzare.

L'obiettivo che mi sono prefissato in questa tesi è quello di scegliere l'applicativo VoIP più stabile presente in rete e di migliorarlo, sia per quanto riguarda le prestazioni sia per quanto riguarda la fruibilità dell'utente. Attualmente la maggior parte degli utenti VoIP non dispongono di una conoscenza approfondita del servizio e per questo si affidano spesso ad applicativi, talvolta a pagamento. Il risultato è che riescono ad utilizzare il programma VoIP in maniera semplice, ma poco configurabile. A questo proposito ho creato un interfaccia elementare che permetta, mediante pochi pulsanti, di iniziare e ricevere comunicazioni VoIP, salvare e gestire contatti in una Rubrica e richiamare un contatto dal registro delle chiamate.

Per quanto riguarda l'aspetto estetico ho deciso di creare un'interfaccia sobria, per non appesantire il dispositivo e per rendere, grazie ad una buona programmazione dei file XML, l'interfaccia il più possibile adattabile alle diverse risoluzioni esistenti tra i vari dispositivi fisici presenti in commercio.

2.2 Strumenti utilizzati

Il modo più efficiente di programmare applicazioni per dispositivi Android è utilizzare l'ambiente di sviluppo Eclipse, il quale permette di installare il plugin ADT(Android Development Tools). È necessario, inoltre, salvare nel disco rigido le librerie di Android scaricabili dal sito http://developer.android.com/sdk/index.html.

Ho scelto di utilizzare questi strumenti a disposizione perché rendono subito disponibile un emulatore di un dispositivo Android, tramite il quale è possibile testare la compatibilità e il programma stesso.

2.3 Emulatore Android

L'SDK Android include un emulatore di un dispositivo mobile. Quest'ultimo permette di testare le proprie applicazioni senza usare un dispositivo fisico.

L'emulatore Android simula tutti i componenti hardware e software di un tipico dispositivo, ad eccezione del fatto che non può mandare o ricevere reali chiamate telefoniche ed alcune altre piccole limitazioni riportate in dettaglio sotto. Dispone di molti mezzi di controllo e navigazione, come usare il click del mouse o utilizzare la propria tastiera come input per il dispositivo. In alternativa può essere utilizzata la tastiera virtuale presente a fianco o sotto l'emulatore. Per permettere di testare le proprie applicazioni in modelli di dispositivi diversi, l'emulatore supporta la configurazione dell'AVD (Android Virtual Device). Quest'ultimo permette di specificare la versione Android che si vuole utilizzare nel simulatore.

L'emulatore include, inoltre, molte capacità di debugging, come una consol dalla quale è possibile visualizzare i log del kernel, simulare interrupt di una applicazione (come l'arrivo di un sms), simulare e gestire la latenza ed eventuali cadute di informazioni del canale dati e, infine, permette di visualizzare e gestire i file nel dispositivo con un semplice

“esplora risorse” ad albero.

L'emulatore Android emula un dispositivo mobile ARM e carica uno stack totale di sistema Android fino a livello kernel; questo include un set di applicazioni preinstallate. Alla creazione del dispositivo, è possibile scegliere quale versione di Android si vuole eseguire nel dispositivo stesso: dimensione dello schermo, presenza e, quindi, emulazione di una SD Card ed altro ancora.

Oltre l'emulatore in sé, l'SDK contiene la libreria nativa, la virtual machine Dalvik, il frameworks android.

Oltre alle strutture sopra descritte, l'emulatore supporta ed emula molti componenti Hardware come:

- un processore ARMv5 e la sua corrispondente MMU (Memory Menagement Unit)
- un Display a 16bit
- una tastiera virtuale composta da una “qwerty” e bottoni speciali utilizzati normalmente in un dispositivo Android come il tasto “Home” o il tasto di accensione e spegnimento
- un chip audio capace di riprodurre e assimilare suoni
- una SD-Card emulata con una cartella residente nel computer locale
- un modem GSM insieme ad una simulata carta SIM

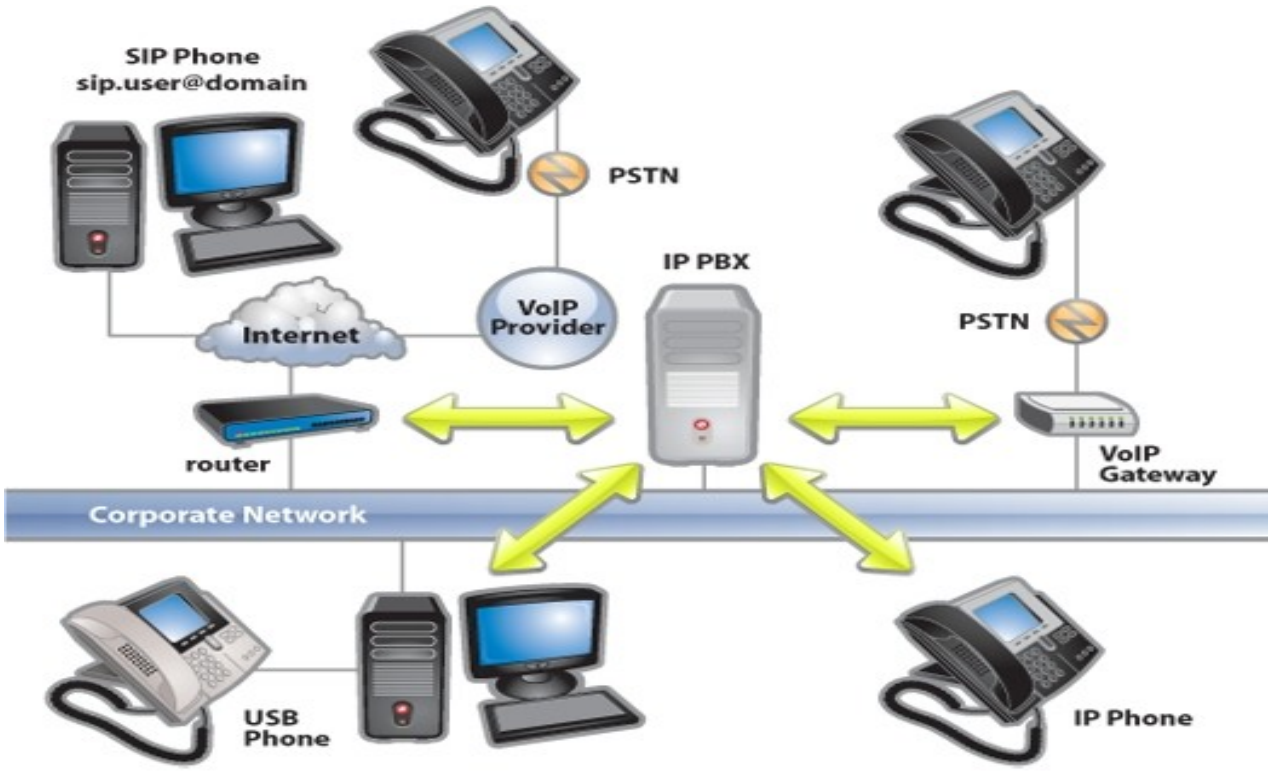
Anche se l'emulatore è un ottimo prodotto per testare nuove applicazioni create, è importante ricordare le limitazioni di questo software. Il dispositivo emulato non possiede:

- la possibilità di iniziare o ricevere reali chiamate telefoniche: è possibile soltanto simulare telefonate per mezzo della console dell'emulatore
- una connessione USB.
- un input per videocamera (webcam)
- un dispositivo per auricolari
- un supporto per conoscere lo stato della batteria ed un metodo di simulazione di ricarica
- la possibilità di determinare se la SD-Card sia inserita o sia stata espulsa
- un servizio Bluetooth

La mancanza di una webcam e di un servizio Bluetooth nel dispositivo emulato mi ha causato non pochi problemi, in quanto le librerie che venivano utilizzate per questi scopi non venivano riconosciuti dal mio emulatore. Per questo motivo volendo utilizzare Sipdroid come motore VoIP della mia interfaccia, ho dovuto modificare i sorgenti di quest'ultimo al fine di eliminare i servizi bluetooth e WebCam dal programma originale.

2.4 Il centralino PBX

Nell'attuazione del mio progetto ho voluto creare un'interfaccia per un'applicazione Voip che possa interfacciarsi anche con un centralino PBX. Il PBX (Private Branch eXchange) è una rete telefonica privata, in cui gli utenti condividono un numero di linee esterne per effettuare chiamate telefoniche. Il VoIP PBX, anche chiamato IP PBX, viene utilizzato dalla mia applicazione ed usa il protocollo SIP per trasmettere le chiamate.



Tra le funzioni tipiche dei centralini IP-PBX è opportuno ricordare l'hold, ovvero la messa in attesa di una telefonata, la deviazione di chiamata, ovvero immettere la chiamata in rotte

alternative in caso di apparecchio occupato, l'utilizzo di IVR (risponditori automatici), la gestione delle code di attesa e la possibilità di registrare le chiamate e di associare ad ogni utente una segreteria telefonica. L'IP PBX ha anche funzionalità non previste nei tradizionali centralini PBX: poter rinviare una chiamata verso un determinato utente a seconda dell'orario, poter scegliere quale compagnia telefonica utilizzare in una certa fascia oraria in base alle tariffe più convenienti.

Capitolo terzo

Progettazione

3.1 Accenni alla programmazione di applicazioni per Android

Ogni volta che si programma un'applicazione Android bisogna prendere in considerazione due ambiti fondamentali: uno riguardante il layout che l'applicazione dovrà avere, l'altro le azioni che voglio far svolgere all'applicazione che creo. La prima parte deve essere implementata per mezzo del file ".xml" nel quale è possibile aggiungere elementi attivi, come bottoni e stringhe di inserimento (EditText), o elementi di cornice come immagini o stringhe. E' consigliabile inserire tutti gli elementi in uno o più Layout.

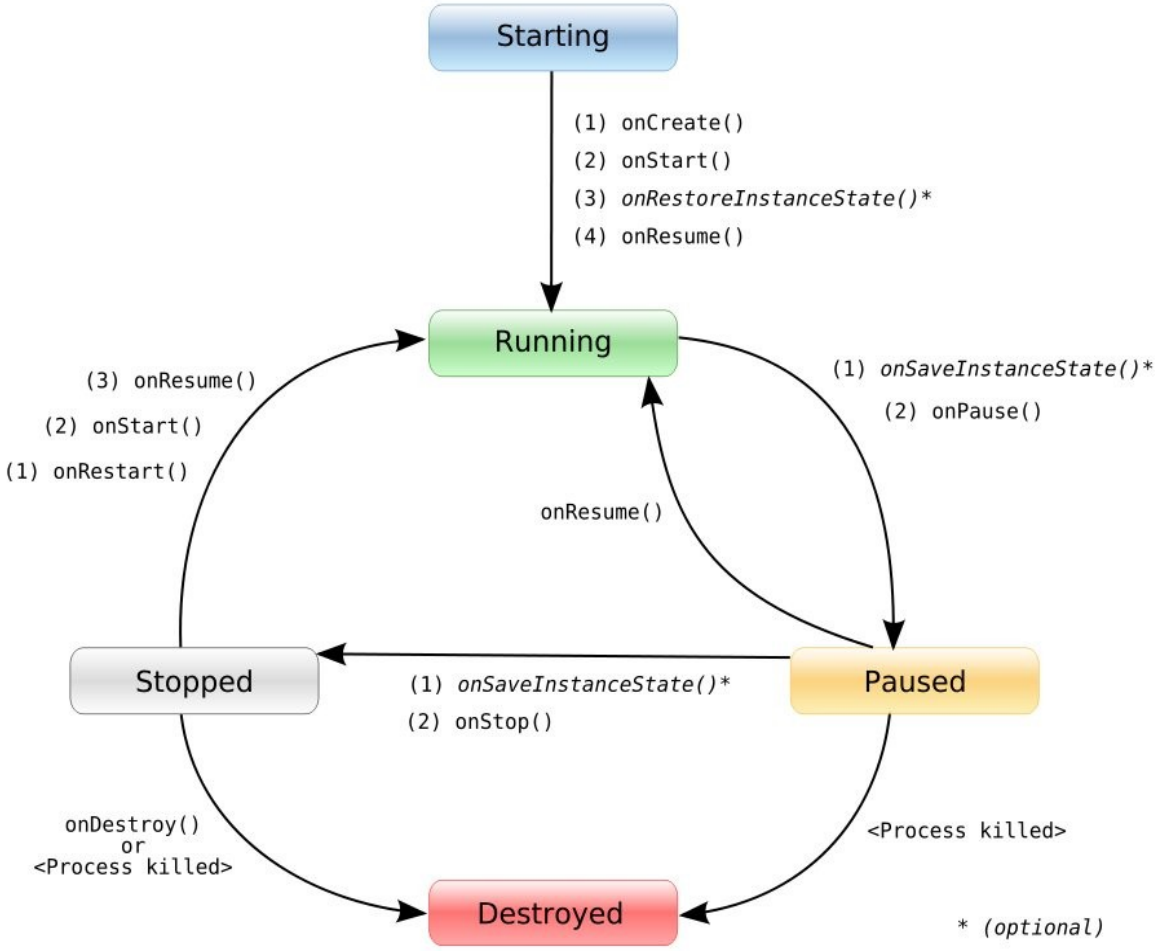
Nel mio progetto utilizzo generalmente il "LinearLayout" poiché mi permette di mantenere una certa proporzione tra i vari componenti aggiunti e di rendere visivamente stabile gli oggetti inseriti anche in caso di cambiamento delle dimensioni del display. Tutti gli oggetti aggiunti nei vari file ".xml" vengono riportati automaticamente da Eclipse nel file R.Java nel modo seguente: (**public static final int** *RegTel*=0x7f060099;). Nel mio progetto questo file possiamo trovarlo al "/telefono/gen/org/sipdroid/sipua/R.Java". Questo passaggio è necessario come ponte di collegamento tra gli oggetti inseriti nei file ".xml" e il loro uso da parte del programma.

Se per quanto riguarda l'aspetto visivo ci basiamo sui file ".xml", per implementare metodi che utilizzano gli elementi aggiunti in precedenza bisogna usare, invece, il linguaggio Java. Nel file Java si utilizza una specifica metodologia che permette il controllo dei vari stati in cui la nostra applicazione può andare.

I componenti applicativi infatti hanno un ciclo di vita che inizia quando Android avvia un'istanza del programma e finisce quando quest'ultimo viene distrutto. Durante la vita di un applicazione, essa potrà essere attiva o inattiva, visibile o invisibile all'utente. Un'attività ha essenzialmente tre stati:

- È attiva, o Running, quando l'applicazione è visibile nel display ed è utilizzabile

- È in pausa se l'applicazione non viene più visualizzata dall'utente o è passata in secondo piano, ma non è ancora stata chiusa. Un'attività in pausa è ancora completamente attiva nel dispositivo, mantiene quindi tutto il suo stato e rimane tracciabile dal Windows Manager, ma può essere chiusa dal dispositivo se la memoria si trova in uno stato estremamente basso.
- È ferma se l'applicazione è completamente oscurata da un'altra attività. Anche in questo caso viene mantenuto il suo stato e può essere terminato dal sistema in caso di necessità di memoria.



Ogni volta che l'applicazione passa da uno stato all'altro, il sistema operativo chiama automaticamente il metodo assegnatogli (`onCreate()`, `onStart()`, `onRestart()`, `onResume()`, `onPause()`, `onStop()`, `onDestroy()`). È possibile aggiungere in questi metodi operazioni che vogliamo vengano eseguiti dal nostro

programma in quel determinato stato. Ho dovuto quindi aggiungere nel metodo “onCreate” tutti gli oggetti utilizzati dal mio file Java e prendere tutti i riferimenti dal file R.Java dei componenti utilizzati dal file “.xml” utilizzando il metodo “findViewById()”. Per passare l'esecuzione del programma da un file Java all'altro creo ogni volta un nuovo oggetto “Intent” ed eseguo il metodo “startActivity()”. È inoltre molto importante ricordare che per ogni nuovo Intent creato c'è la necessità di aggiornare il file “AndroidManifest.xml”, riporto di seguito un esempio:

```
<activity android:name="cs.voip.CSVP" android:label="@string/app_name">
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
```

Ogni applicazione deve avere un file AndroidManifest.xml nella propria cartella principale. Il Manifest presenta le informazioni essenziali dell'applicazione per il sistema Android, queste informazioni devono essere date al sistema prima che l'applicazione venga eseguita. Tra le altre cose, il manifesto possiede le seguenti informazioni:

- il nome del Java Package dell'applicazione. Il nome del pacchetto è necessario per identificare univocamente l'applicazione.
- la descrizione dei componenti dell'applicazione come Activity (Intent) e servizi. Queste dichiarazioni permettono al sistema Android di sapere quali sono i componenti utilizzati e sotto quali condizioni possono essere lanciati.
- la descrizione dei processi che verranno ospitati dall'applicazione.
- le dichiarazioni di quali permessi l'applicazione possiede per permettergli l'utilizzo di API protette ed interagire con altre applicazioni.
- la lista dei permessi necessari per far accedere altre applicazioni alle risorse dell'applicazione del Manifest.

- la lista delle classi di strumentazione che forniscono profiling ed altre informazioni su come l'applicazione debba essere avviata.
- la dichiarazione della versione minima delle API necessarie per il corretto funzionamento dell'applicazione. La mia specifica applicazione ha necessita della versione Android 1.5.7 .
- la lista delle librerie linkate nell'applicazione.

3.2 Valutazione del progetto

Come mi sono prefissato negli obbiettivi, l'interfaccia risulta fin da subito estremamente facile da usare. Eliminando il tastierino numerico e dando la possibilità di inserire una qualsiasi stringa come recapito telefonico, è possibile chiamare sia i numeri tradizionali sia gli account VoIP con estrema facilità.

Ho cercato, inoltre, di rendere il menù principale dell'applicazione il più possibile semplice, per mezzo di quattro pulsanti, e leggera aumentando di poco le prestazione dell'intero programma.

L'utilizzo della rubrica è immediato, con la possibilità di inserire nome, cognome, telefono (nel quale è possibile aggiungere un recapito VoIP in formato stringa) e indirizzo e-mail. Dato uno specifico contatto è, inoltre, possibile chiamarlo facilmente mediante un pulsante specifico di chiamata. Nel caso si vogliano aggiungere altri campi per ogni contatto, è necessario aggiornare il file “addcontact.xml” e “DBManager.Java”.

Il registro chiamate contiene i record di tutte le chiamate effettuate e ricevute, elencate per ordine di arrivo. Selezionando un qualsiasi elemento verrà data la possibilità di richiamarlo automaticamente.

Fatta eccezione per l'icona personalizzata che contraddistingue il programma CSVP (Computer Science VoIP) nel menu dei programmi, ho scelto di non appesantire l'applicazione con animazioni o icone che diminuiscono la compatibilità del programma e la velocità del dispositivo su cui è installato.

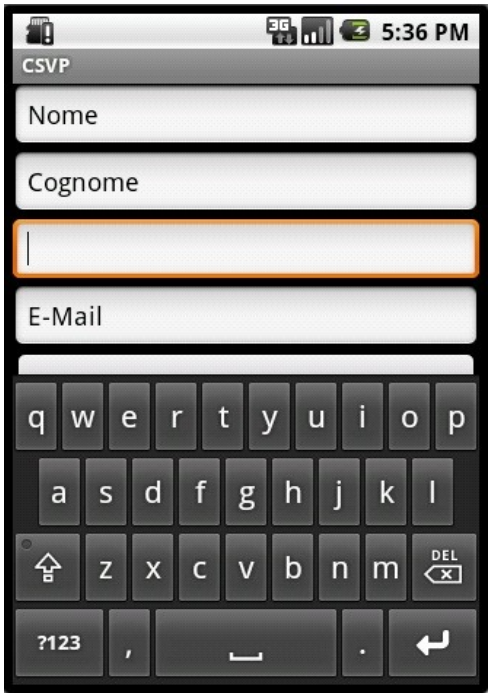
3.3 Scelte progettuali

Ho scelto di organizzare il progetto in modo tale da avere una file Java per ogni file ".xml"; ho utilizzato, inoltre, un file Java supplementare, chiamato "DBManager.Java", per la creazione e la gestione del database.

L'avvio dei thread per la connessione al server e l'attesa di una chiamata entrante, nonché la creazione del menù principale del programma, avviene nel file "CSVP.Java".

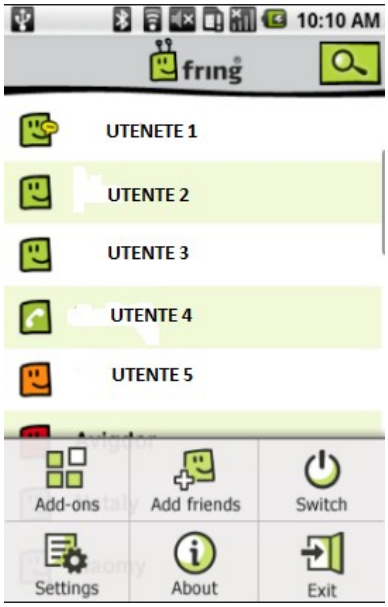
Sia il registro delle chiamate sia la rubrica sono organizzati in una lista di elementi selezionabili, la cui scelta porta alla telefonata diretta dell'elemento, se la selezione avviene nel registro delle chiamate, o dalla visualizzazione della scheda dell'elemento, se la selezione avviene dalla rubrica. Ho scelto di non modificare troppo il layout di queste due liste per rendere l'interfaccia il più semplice e familiare possibile.

In qualsiasi editText che compare nel mio programma ho predisposto l'inserimento automatico di una stringa che descrive a cosa corrisponde lo spazio dato; all'aggiunta di un contatto compariranno, per esempio, quattro stringhe di input ed in ognuna di esse un distinto valore (nome, cognome, telefono, E-mail). Ho creato per ogni classe interessata un listener di selezione che cancella automaticamente tutta la editText la prima volta che viene selezionata, affinché l'utente non debba ogni volta cancellare la stringa di indicazione.



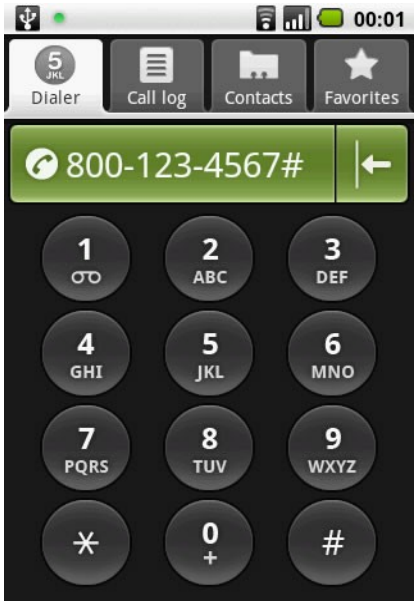
3.4 Il progetto Sipdroid

Dovendo creare un'interfaccia user-friendly per applicazione VoIP in ambiente Android, ho cercato una buona applicazione VoIP per poter inserire la mia interfaccia. Una delle più comuni applicazioni VoIP per Android è Fring.



Fring è un applicazione mobile che permette agli utenti di comunicare tra di loro usando la connesione ad internet. Con questo programma è possibile effettuare telefonate, video chiamate e live-chat e si può comunicare con altri servizi come MSN Messenger®, GoogleTalk™, AIM®, ICQ® , Facebook® & Twitter. Il motivo principale per cui ho deciso di non utilizzare Fring consiste nel fatto che non è open-source e, quindi, non avrei potuto modificare il codice sorgente dell'applicazione.

Continuando la mia ricerca ho, invece, trovato un progetto molto interessante, il sipdroid (sipdroid.org), cui mi sono riferito per creare la mia interfaccia.



SipDroid è il primo client sip interamente open source. SipDroid è stato realizzato per il dispositivo HTC Dream ed è stato sviluppato usando una tecnologia Java MjSIP. SipDroid è stato poi rilasciato sotto licenza GPL 3, questo progetto è ovviamente ancora in fase di sviluppo, ma in ogni caso può essere già usato, anche con diverse funzionalità, come ad esempio la scelta di una rete preferita, la gestione dei messaggi di testo e l'uso del client su diverse reti dati.

Sipdroid, infatti, permette di scegliere se collegarsi alla rete WLAN, 3G e EDGE. Per il suo corretto utilizzo Sipdroid ha bisogno una velocità di trasmissione di circa 80 kBits/s in ciascuna direzione: 1.2 Mb al minuto per una comunicazione vocale e di circa il doppio per una videochiamata. Se nelle opzioni è stato selezionato il codec GSM o se viene utilizzata una connessione EDGE, Sipdroid utilizzerà 30 kBit/s per ciascuna direzione. Sipdroid è stato progettato per funzionare nella versione 1.5 "Cupcake"di Android, il funzionamento su altre versioni del sistema operativo non è garantito.

Per poter compilare, eseguire e modificare Sipdroid è necessario installare il plugin aggiuntivo di Eclipse chiamato Subclipse. Personalmente, pur avendo installato il plugin, ho avuto difficoltà nell'importare il progetto poiché il compilatore non riusciva a generare correttamente il file "R.Java", dandomi errori critici nell'intero progetto. Dopo essermi documentato e vedendo che il mio problema era un evento isolato, ho deciso di riscrivere

l'intero codice facendo generare il file sopra descritto da Eclipse come "nuovo progetto" risolvendo quindi il problema.

Sipdroid utilizza un SIP standard e per avere una piena interoperabilità consiglio di registrarsi ed utilizzare il servizio PBXes (www1.PBXes.com), che descriverò più in dettaglio nelle pagine seguenti. Se viene utilizzato un diverso server SIP è possibile che:

- venga esaurita velocemente la batteria del dispositivo utilizzando la rete 3G/EDGE
- il servizio di chiamata entrante non venga effettuato
- si verifichi una comunicazione vocale ad una sola via
- non venga riconosciuta la connessione

Il motivo del forzato utilizzo del servizio PBXes è nella progettazione di Sipdroid stesso, ottimizzato per funzionare con quel servizio e non per qualunque servizio SIP. Se si è in possesso di altri account SIP è conveniente comunque registrarsi ed utilizzare il servizio PBXes ed aggiungere ai “trunk” del servizio il proprio account VoIP preferito. Il metodo di configurazione di un possibile account PBXes verrà descritta in seguito.

3.5 Il servizio PBExs

Il servizio PBXes è un centralino VoIP costantemente in esecuzione come un servizio di segreteria telefonica o il server di posta elettronica in un data-center. Come un classico servizio PBX, esso permette di fare da ponte di collegamento tra l'utente o l'organizzazione che lo utilizza (le estensioni) e la rete telefonica pubblica (i trunk o tronchi). Come tronchi possono essere utilizzati qualsiasi provider SIP in tutto il mondo e le tariffe per le telefonate a pagamento saranno quelle del provider SIP utilizzato senza nessun rincaro da parte del servizio PBXes.

I servizi aggiuntivi disponibili con il servizio PBXes sono molti, elencherò qui di seguito i più importanti:

- Risposta alle chiamate entranti
- Gruppi suoneria
- Risposta automatica in caso di assenza
- Inoltro di chiamata
- Forking, processo di divisione di una singola chiamata SIP a multipli account SIP finali. In pratica chiamando un singolo account SIP, la telefonata verrà inoltrata a più dispositivi e quindi a più account.
- Servizio di VoiceMail come e-mail
- Monitor, gestione e configurazione delle chiamate
- Conferenze tra più account
- Citofono virtuale
- creazione di account utente-interno@PBXes.org
- integrazione di gateway PSTN

Un possibile metodo di configurazione del servizio PBX verrà descritto successivamente.

3.6 Stack SIP MjSip

SipDroid utilizza lo stack SIP creato da MjSip (http://www.mjsip.org). MjSip è una libreria compatta e potente SIP che serve a costruire facilmente applicazioni SIP e servizi. Esso prevede nello stesso tempo l'implementazione SIP e SIP API.

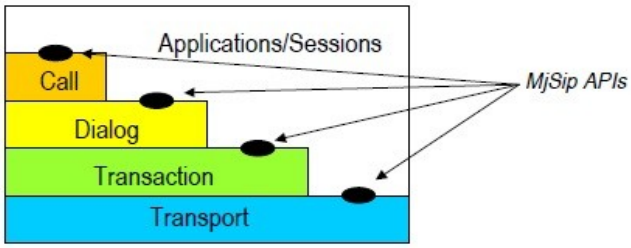
In particolare MjSip include:

- Oggetti standard SIP come messaggi SIP, transazioni, etc.
- Varie estensioni SIP definite all'interno del IETF (Internet Engineering Task Force)
- Call API di controllo
- Una implementazione di riferimento di alcuni sistemi SIP come server ed user agent

Le principali caratteristiche di MjSip sono:

- È basato su Java quindi è cross-platform, ovvero eseguibile in piattaforme differenti senza dover necessariamente cambiare il codice sorgente.
- Non è solo un API ma contiene una implementazione completa dello stack SIP.
- È molto leggero e può essere utilizzato contemporaneamente sia per l'implementazione del server sia per quella del client.

Secondo l'architettura SIP definita nella RTF 3261 il nucleo di MjSip è strutturata su tre livelli: trasporto, transazione e dialogo. In cima a questi livelli si prevede anche il controllo delle chiamate e il livello applicativo con le API corrispondenti.



Lo strato più basso è il livello di trasporto che fornisce il trasporto dei messaggi SIP. Il SipProvider è l'oggetto MjSip che fornisce il livello di trasporto a tutti i livelli superiori. Ogni elemento SIP deve utilizzare le API del SipProvider se vuole avere accesso al servizio di trasporto.

Il secondo livello è quello di transazione; esso è una componente essenziale poiché nel SIP una transazione è una richiesta spedita da un client (transaction client) verso un server e la sua relativa risposta. Il livello di transazione, quindi, gestisce la ritrasmissione dei livelli superiori, il controllo delle richieste e risposte delle parti e il timeout. Ovviamente il livello di transazione deve inviare e ricevere messaggi per mezzo del livello di trasporto.

Il terzo livello è il dialogo il quale lega differenti transazioni della stessa "sessione". Il dialogo è una relazione SIP peer-to-peer tra due user agent che persiste per un certo periodo di tempo. Il livello di dialogo facilita la successione dei messaggi e le routin di richiesta tra i vari user agent. Come definito nella RFC 2631, il metodo "INVITE" stabilisce un dialogo (denominato "invite dialog") ed è implementato in MjSip nella classe InviteDialog. Un dialogo è definito per mezzo della combinazione To Tag, From tag e Call-ID. InviteDialog gestisce anche il metodo CANCEL.

MjSip offre API per l'accesso a tutti i livelli precedenti SIP, dal SipProvider fino alla chiamata reale, e una implementazione di riferimento per vari sistemi a livello applicativo. Uno sviluppatore può scegliere di utilizzare tutte o in parte le API fornite da MjSip, le quali sono implementate dalle classi:

- Call e ExtendedCall
- InviteDialog
- ClientTransaction, ServerTransaction, InviteClientTransaction e InviteServerTransaction
- SipProvider

MjSip è intrinsecamente estendibile, i nuovi SIP heder e/o metodi possono essere facilmente inseriti ed integrati, per questo motivo è stato scelto ed utilizzato dal progetto Sipdroid. Lo stack MjSip è formato da due pacchetti principali: sip e sipx. Il primo pacchetto include tutti gli heder SIP standard, metodi, e livelli standard SIP (trasporto, transazione e dialogo), mentre il secondo comprende le estensioni possibili ed è aperto per nuove estensioni, quali nuovi SIP header, metodi e funzioni.

Lo stack MjSip è contenuto nel mio progetto nei pacchetti “zoolu” ed è stato creato come attività di ricerca dal Dipartimento di Ingegneria dell'Informazione dell'Università di Parma e dal Dipartimento Di Ingegneria Elettronica dell'Università degli Studi di Roma "Tor Vergata" ed è attualmente sfruttato commercialmente da CREALAB.

Capitolo quarto

Problematiche riscontrate e soluzioni adottate

4.1 Salvataggio dei contatti e del registro delle chiamate

Durante la progettazione e la programmazione della mia interfaccia uno dei primi problemi riscontrati è stato il metodo di salvataggio da adottare nel dispositivo del registro delle chiamate e dei contatti.

Android fornisce varie opzioni che permettono di salvare i dati delle applicazioni in modo persistente.

Le opzioni di archiviazione dei dati sono i seguenti:

- Preferenze condivise: archivio privato di dati primitivi in coppie chiave-valore. La classe "`SharedPreferences`" fornisce un quadro generale che consente di salvare e recuperare i valori in tipi dati primitivi. Con questo metodo è possibile salvare booleani, float, int, long, e stringhe. Queste informazioni vengono salvate nella sessione utente e rimangono persistenti anche dopo l'eliminazione dell'applicazione
- Memoria Interna: è possibile salvare i file direttamente sulla memoria interna del dispositivo. Per impostazioni predefinite, i file salvati nella memoria interna sono privati per l'applicazione e sono inaccessibili per le altre applicazioni, o l'utente stesso. Quando l'utente disinstalla l'applicazione, questi file vengono rimossi
    - Salvataggio di file nella cache: se desideri memorizzare nella cache alcuni dati, piuttosto che conservarli in maniera persistente, si deve usare il metodo "`getCacheDir()`" per aprire un `File` che rappresenti la directory di cache interna utilizzabile per salvare dati temporanei. Quando il dispositivo è a corto di memoria, Android può eliminare questi file di cache per recuperare spazio. Non si dovrebbe fare affidamento sul sistema per ripulire automaticamente questi file,

ma è auspicabile eliminarli periodicamente e mantenerli sempre di dimensioni ragionevoli. Quando l'utente disinstalla l'applicazione, questi file vengono rimossi.

- Memoria esterna: ogni dispositivo compatibile con Android supporta una "memoria esterna" condivisa che è possibile utilizzare per salvare i file. Questo può essere un supporto di memorizzazione rimovibile, ad esempio una scheda SD, o una non rimovibile, interna al dispositivo. I file salvati sulla memoria esterna sono leggibili da tutti e possono essere modificati dall'utente, sia collegando il dispositivo al computer per mezzo della porta USB sia rimuovendo e accedendo direttamente alla memory card. Questo metodo è però inaffidabile, poiché i file esterni possono scomparire se l'utente li elimina accidentalmente o se viene rimossa la scheda di memoria. Le applicazioni esterne possono, invece, cancellare o modificare il contenuto di questi dati rendendo di fatto questo metodo di memorizzazione inaffidabile.
    - Salvataggio di file nella cache: anche questo metodo permette di salvare file temporanei. Anche in questo caso, durante la vita della nostra applicazione, è necessario gestire questi file di cache e rimuovere quelli che non sono necessari al fine di preservare lo spazio del file.
- SQLite Database: Android fornisce pieno supporto per il database SQLite. Ogni database creato, è accessibile in base al nome per qualsiasi classe del nostro programma, ma non al di fuori dell'applicazione. Il metodo consigliato per creare un nuovo database SQLite è quello di creare una sottoclasse di "`SQLiteOpenHelper`" e sovrascrivere il metodo "`onCreate()`", in cui è possibile eseguire un comando SQLite per creare le tabelle nel database. Android non impone alcuna limitazione al di là dei concetti standard SQLite. Ogni query SQLite restituisce un "`Cursor`" che punta a tutte le righe trovate dalla query. Un "`Cursor`" è il meccanismo con il quale è possibile scorrere i risultati di una query di un database e leggere righe e colonne di una tabella.
- Connessione di rete: è possibile utilizzare la rete, quando disponibile, per memorizzare e recuperare dati sul nostro servizio web-based. Per fare operazioni di

rete vengono utilizzate le classi appartenenti ai pacchetti “`Java.net.*`” e “`android.net.*`”.

Tra queste possibilità ho scelto di utilizzare il database SQLite messo a disposizione da Android, ritenendolo il metodo migliore per salvare le informazioni che mi interessavano. Documentandomi sul metodo di utilizzo del database ho notato che, se da un lato l'inizializzazione di un nuovo database con la struttura da me scelta risulta poco complesso, dall'altro la gestione delle tuple nelle varie tabelle è alquanto caotico. Per ovviare a questa difficoltà ho deciso, quindi, di creare un file Java apposito chiamato “DBManager.Java”, nel quale viene collocato un nuovo database, quando l'applicazione viene installata nel dispositivo per la prima volta, e vengono gestite diverse operazioni di lettura e scrittura nel database, per mezzo di funzioni specifiche da me sviluppate:

- DatabaseHelper = permette di creare un nuovo database nel dispositivo di nome “ripostiglio” e gli viene assegnata come versione del database il valore 1
- onCreate(SQLiteDatabase db) = è una funzione che viene richiamata solo se viene creato un nuovo database. Nel nostro caso specifico, viene richiamata solo la prima volta che l'applicazione viene salvata nel dispositivo. Questo metodo esegue due operazioni simili tra di loro: la prima creerà una tabella per il registro delle chiamate, la seconda creerà un altra tabella per la rubrica. Il metodo preso in esame esegue su un database una stringa di codice SQL. Espongo di seguito il codice SQL per la creazione della tabella del registro chiamate:

<<create table "+*DATABASE_TABLE1*+" ("+*KEY_ROWID*+" integer primary key autoincrement, " + ""+*KEY_TimeStamp*+" integer not null, "+*KEY_Telefono*+" text not null);>>

- insertTitle(String time, String telefono) = questo metodo permette di aggiungere un nuovo elemento nella tabella “registrochiamate”
- insertTitle(String nome, String cognome, String telefono, String mail) = questo metodo permette di aggiungere un nuovo elemento nella tabella “rubrica”
- deleteTitle(**long** rowId, int Type) = questo metodo elimina la tupla con posizione eguale a “rowId” nella tabella “registrochiamate” se il valore “Type” è uguale a 0, altrimenti eliminerà la tupla corrispondente nella tabella “rubrica”

- GetAllTitles(int Type) = questo metodo restituisce un oggetto di tipo “Cursor” dal quale possiamo successivamente recuperare tutti gli elementi o della tabella “registro chiamate” se “Type” è uguale a 0, oppure della tabella “rubrica”
- getTitle(**long** rowId, int Type) = questo metodo, come il precedente, restituisce un oggetto di tipo “Cursor”, dal quale possiamo ricavare però solo un particolare elemento della tabella coincidente alla posizione eguale a “rowId”. La scelta della tabella da cui estrarre l'elemento è sempre basata sul valore della variabile “Type”.

Oltre al problema dell'organizzazione delle procedure utilizzate sul database, nel corso della programmazione ho riscontrato, ad una prima prova, un problema circa l'effettivo funzionamento del database. Le tabelle create non potevano più essere modificate poiché il database rimane presente nel dispositivo anche se l'applicazione viene modificata e ricaricata nel dispositivo in uso. Per ovviare a questo problema ho trovato una possibile soluzione: bisogna eliminare totalmente l'applicazione dal dispositivo prima di installarne una nuova modificata. Questa procedura avrà conseguentemente l'effetto di eliminare ogni dato registrato nel database stesso. L'eliminazione manuale del singolo file, contenente il database, non sembra risolutiva per il problema dato.

4.2 Difficoltà nella configurazione del serviziuo PBXes

Dovendo fare delle prove per verificare il corretto funzionamento del programma ho dovuto creare un account PBXes e configurarlo. Come detto nel capitolo riguardante questo servizio, il suo utilizzarlo è conveniente, poiché sipdroid rischia di non funzionare con altri account SIP. Una volta creato un nuovo account PBXes, sarà necessario configurarlo al meglio per consentire il corretto funzionamento della nostra applicazione.

Entrando nel setup del servizio andiamo nella sezione “Interni”, creiamo un nuovo interno SIP, diamo un valore numerico al nostro interno ed un nome identificativo, diamo infine una password all'interno appena creato. D'ora in poi potremo collegarci con la nostra applicazione utilizzando come user (user dell'account PBXes) il valore numerico del nostro

interno.

Andiamo ora a configurare i tronchi. Creiamo un nuovo tronco SIP, diamogli un nome identificativo del tronco, selezioniamo come “dtmfmode” il valore “rfc2833”, diamo all'opzione “audio bypass” il valore “si”. Nella sezione “Account” possiamo aggiungere un diverso account SIP che verrà bypassato automaticamente dal servizio PBXes. Aggiungiamo quindi l'utente, la password ed il SIP server del nostro account alternativo.

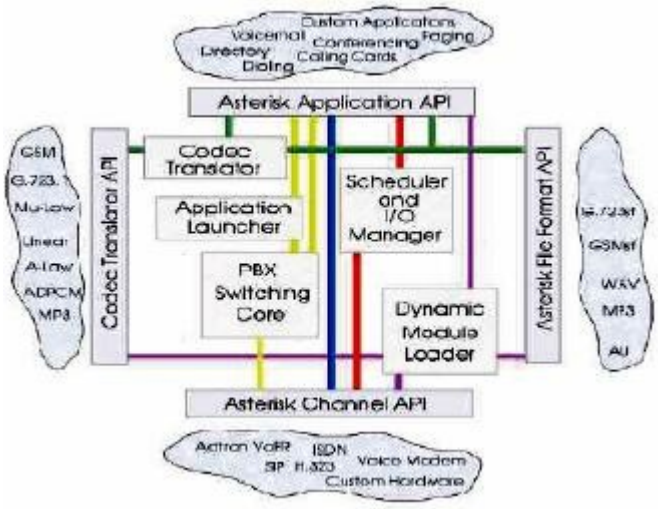
Creiamo una nuova rotta di entrata SIP, selezioniamo sia per il servizio giorno sia in quello notte, l'interno che abbiamo creato in precedenza e selezioniamo l'opzione “Forza servizio giorno”.

Creiamo, infine, una rotta di uscita (sempre nominandola) e selezionando il tronco creato in precedenza. La configurazione del servizio PBXes risulterà allora completata e riuscirete a collegarvi con la vostra applicazione per mezzo dell'account PBXes appena creato.

4.3 Asterisk come possibile alternativa all'uso del PBXes

Dover utilizzare necessariamente il servizio PBXes per il progetto Sipdroid è un grosso problema per qualsiasi sviluppatore che intende avere un programma SIP-client generico. Una possibile soluzione a questo problema può essere l'ausilio di un centralino elettronico PBX come Asterisk.

Asterisk è un software open-source sviluppato dalla Digium (www.digium.com) in ambiente Linux e permette di avere un proprio servizio di centralino elettronico PBX a basso costo. Asterisk è stato creato per interfacciare qualsiasi tipo di apparato telefonico standard e supporta tre protocolli VoIP tra i quali il SIP. Asterisk è creato per fornire la massima flessibilità, API specifiche sono definite attorno ad un centralio PBX che fa da nucleo, il quale si preoccupa delle interconnessioni interne. Questo permette ad Asterisk di essere compatibile e di utilizzare qualsiasi tecnologia disponibile ora o nel futuro.



Facendo ricerche nell'ausilio del servizio Asterisk per il progetto Sipdroid si evince la difficoltà nell'utilizzarlo in questo progetto. Spesso infatti Sipdroid non riesce a connettersi ad un servizio Asterisk dando un problema di “Timeout”. La motivazione più accreditata consiste nel fatto che Sipdroid non risponde alle richieste INVITE, BYE e NOTIFY. La soluzione, quindi, non è semplice ed è tuttora in elaborazione. Gli sviluppatori del progetto Sipdorid hanno chiaramente espresso la loro volontà di utilizzare solo il gataway PBXes; qualsiasi altro account SIP non verrà da loro gestito ed invitano ad utilizzare una diversa applicazione SIP, in pratica la risoluzione del problema è demandata allo sviluppatore che lo vuole risolvere.

4.4 Possibili migliorie

Il mio programma, se pur funzionante e conforme agli obbiettivi prefissati, potrebbe essere ampliato e migliorato. Potrebbe essere utile aggiungere al programma una client mail, con la possibilità di mandare e ricevere e-mail mediante la stessa interfaccia e gli stessi componenti. Il metodo di integrazione sarebbe molto semplice, basterebbe aggiungere nel menù principale il bottone per entrare nella gestione delle e-mail, azione molto semplice dato la buona programmazione del layout, ed un altro eventuale bottone nella scheda di un contatto per poter mandare immediatamente un'e-mail al contatto selezionato. Si potrebbe

inoltre aggiungere un metodo per l'esportazione o l'importazione di tutta la rubrica rendendo però di conseguenza l'applicazione più complessa e per alcuni più difficile da usare.

Le applicazioni VoIP per l'ambiente Android sono poche e ancor meno sono quelle Open-Source; la maggior parte di queste hanno, inoltre, ancora problemi di funzionalità e di compatibilità. Non sarebbe male poter inserire la mia interfaccia in una applicazione di gestione telefonate VoIP più stabile e, soprattutto, senza il necessario ausilio del servizio PBXes.

Conclusioni

Ho scelto come argomento della mia tesi la creazione di una interfaccia per applicazioni VoIP constatando la scarsa fruibilità delle applicazioni VoIP preesistenti; come già precedentemente esposto nell'*Introduzione* e nel corpo della tesi, l'obiettivo che mi sono prefissato è principalmente la programmazione di un interfaccia che risulti di facile utilizzo anche agli utenti non esperti nella comunicazione VoIP. La caratteristica della mia interfaccia è, infatti, quella di avere una grafica semplificata, in modo da esplorare l'intera applicazione per mezzo dell'ausilio di pochi pulsanti. Per fare questo è stato necessario studiare: la metodologia di sviluppo di applicazioni per i Dispositivi Android, il funzionamento del protocollo SIP, di una comunicazione VoIP e di un centralino IP PBX.

La programmazione di applicazioni fatte per ambiente Android è stata un'esperienza molto positiva. Non è stato eccessivamente difficile, infatti, adattarmi alla programmazione utilizzando il linguaggio Java, già appreso nel corso della mia carriera universitaria. L'ambiente di sviluppo scelto mi è stato fin da subito familiare, ciò mi ha permesso di sfruttare al meglio le risorse datemi. L'AVD, messo a disposizione per sviluppatori Android, è stato fondamentale non solo per l'autogenerazione del file R.java (procedimento che sarebbe risultato assai complesso), ma anche dagli strumenti di controllo, fondamentali per fare debugging ed avere un controllo diretto del dispositivo emulato. Fondamentale, infatti, è stato l'utilizzo dell'emulatore, senza il quale avrei potuto avere solo un approccio teorico all'esperienza di programmazione in Android. Pur essendo Android un sistema operativo nato da pochi anni, le API messe a disposizione ai programmatori sono molte, utili e ben documentate, aumentando di fatto le potenzialità che questo sistema operativo offre.

La ricerca di un buon programma VoIP per dispositivi Android non è stata altrettanto agevole. Purtroppo la “giovane età” di questo sistema operativo ha come conseguenza la carenza di applicazioni disponibili e quelle disponibili non evidenziano una sicura compatibilità con i vari dispositivi messi in commercio.

Il progetto Sipdroid è un buon progetto di applicazione di telefonia VoIP che

garantisce il collegamento con diversi circuiti di comunicazione, come GoogleTalk™ ed altri, ed un buon funzionamento operativo, ma al tempo stesso è ancora in fase di sviluppo creando a volte errori dipendenti dal dispositivo usato, sia esso fisico sia esso emulato. Essendo un progetto ancora in via di sviluppo, inoltre, è stato predisposto all'ausilio del solo centralino PBXes e non a qualsiasi, vincolando così gli utilizzatori ed i programmatori di tale progetto. A prescindere da questi ultimi problemi, il programma Sipdroid resta comunque un buon programma VoIP, che ha le potenzialità per diventare il miglior progetto open-source di telefonia VoIP per dispositivi Android.

La mi interfaccia risultante dallo studio della programmazione e dalla ricerca fatta sul mondo VoIP risulta pienamente soddisfacente riguardo gli obbiettivi prefissatomi. Fin da subito, infatti, l'utente si trova di fronte ad un'interfaccia sobria ed intuitiva, che con pochi tocchi soddisfa le necessità dell'utente.

Premendo il tasto "Telefona" è possibile fin da subito iniziare una telefonata vocale immettendo direttamente l'indirizzo SIP o il numero del dispositivo tradizionale che si vuole chiamare.

Avendo creato un database che accoglie i record delle telefonate effettuate e ricevute ed i record dei contatti in due tabelle differenti, sono riuscito ad avere una buona organizzazione della memoria, dando la possibilità, in futuro, di esportare o di importare l'intero database o parte di esso selezionando ciò che interessa all'utente.

L'interfaccia, che accoglie la Rubrica, è immediata e dà subito all'utente la lista dei contatti memorizzati in precedenza ed un tasto superiore che permette l'aggiunta di nuovi contatti. L'inserimento dei contatti avviene per mezzo di quattro editText nel quale inserire rispettivamente nome, cognome, telefono ed E-mail. Alla selezione di un singolo contatto vengono mostrati tutti i valori che lo compongono e due pulsanti che permettono la cancellazione o la comunicazione vocale diretta dell'elemento scelto.

Ho creato un registro delle chiamate con un impatto grafico molto simile alla rubrica, cercando di abituare l'utente ad un utilizzo immediato e familiare. All'apertura compare l'elenco delle chiamate ricevute ed effettuate. Con la selezione di un record, l'utente viene direttamente portato nella sezione "Telefona", col numero dell'elemento selezionato già inserito nella stringa di input.

Le impostazioni, infine, sono organizzate come una lista di menù in cui la cui selezione di un elemento apre in dettaglio i campi che lo riguardano. Se selezioniamo il menu "SIP account", ad esempio, vengono visualizzate tutte le componenti necessarie al corretto login tramite il servizio PBXes. Per mezzo delle impostazioni è possibile, oltre a configurare l'account SIP, gestire i codecs da utilizzare nelle comunicazioni vocali, selezionare il protocollo di trasmissione (TCP od UDP), selezionare le reti utilizzabili (Wifi, 3G, GPRS, VPN e auto-answert), selezionare la suoneria e le impostazioni audio, scegliere le impostazioni audio-video, selezionare impostazioni avanzate per la Wireless e le opzioni per il servizio PBXes.

La creazione di questo progetto mi ha consentito di maturare l'esperienza di una programmazione autonoma, a partire dalla necessità di organizzare il tempo e le risorse impiegate. L'argomento della tesi mi ha portato, inoltre, ad interessarmi e ad informarmi sul mondo delle applicazioni VoIP e Android, conosciute fino a quel momento solo come utilizzatore finale.

Durante il periodo di programmazione, è stato importante il pregresso studio del linguaggio di programmazione Java e dei linguaggi di gestione database come SQL. Le mie precedenti esperienze di creazione di file ".xml" hanno anch'esse contribuito alla creazione del mio progetto. Senza queste basi non avrei potuto portare a compimento il presente lavoro.