Alma Mater Studiorum
Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea Magistrale in Informatica

Un'applicazione VoIP per Symbian: un'interfaccia utente

Tesi di Laurea in Architettura

Autore:
Giampiero Calma

Relatore:
Dott. Vittorio Ghini

Anno Accademico 2009-2010
Sessione III

Indice

Elenco delle figure

Introduzione

Nel campo della tecnologia, l'ultimo decennio è stato caratterizzato da significativi sviluppi nel mondo dei dispositivi mobili. Si è passati dal tradizionale telefono cellulare, ai più recenti palmari e Smartphone che integrano al tradizionale stereotipo di telefono cellulare, funzionalità avanzate su hardware molto sofisticato.
Con un moderno dispositivo mobile infatti, è possibile collegarsi ad Internet, leggere mail, guardare video, scaricare applicazioni e installarle per poterne così fruire.
L'International Telecommunications Union (ITU-T) ha stimato che alla fine del 2010 il numero di utenti Internet a livello mondiale ha raggiunto i 2 miliardi e che gli accessi alla rete cellulare hanno raggiunto circa 5,3 miliardi di sottoscrizioni. Se si considera inoltre che le reti 2G verranno sostituite da quelle 3G (che consente una connessione alla rete a banda larga tramite dispositivi cellulari), è inevitabile che nel prossimo futuro, gli utilizzatori di Internet tramite rete mobile potranno arrivare ad essere anche qualche miliardo [1].
Le applicazioni disponibili in rete sono spesso scritte in linguaggio Java che su dispositivi embedded, dove è cruciale il consumo di energia, mettono in crisi la durata della batteria del dispositivo. Altre applicazioni scritte in linguaggi meno dispendiosi in termini di consumi energetici, hanno un'interfaccia scarna, a volte addirittura ridotta a semplice terminale testuale, che non è indicata per utenti poco esperti. Infine altre applicazioni sono state eseguite solo su simulatori o emulatori, perciò non forniscono riscontri su dispositivi reali.
In questa tesi verrà mostrato come su un dispositivo mobile sia possibile utilizzare, tramite un'interfaccia "user-friendly", una tecnologia già esistente e

diffusa come il VoIP in maniera tale che qualunque tipologia di utente possa utilizzarla senza conoscerne i dettagli tecnici.
Tale applicazione, dovendo utilizzare una connessione dati, sfrutterà o una connessione a una rete WLAN o una connessione a una rete cellulare (GPRS, UMTS e HSDPA ad esempio) a seconda della dotazione hardware dell'apparecchio mobile e della locazione dello stesso in una rete accessibile dall'utente.
Il client VoIP è stato sviluppato in collaborazione con il collega tesista Giulio Massaccesi che si è concentrato più sull'architettura dell'applicazione, mentre chi scrive si è occupato più della parte frontend, in particolare la creazione dell'interfaccia e l'interazione tra questa e gli strati sottostanti.
Il documento di tesi è diviso in quattro capitoli di cui di seguito viene fornita una breve descrizione.
Nel primo capitolo, dopo una breve descrizione della tecnologia VoIP, verrà presentato il funzionamento del protocollo di segnalazione SIP. Successivamente verrà presentata la tecnica del "multihoming" per estendere le funzionalità mobili dei dispositivi e l'architettura ABPS, una possibile applicazione di tale tecnica, proposta e sviluppata presso il dipartimento di Scienze dell'Informazione dell'Università di Bologna.
Nel secondo capitolo verranno fissati gli obbiettivi e verranno elencate e dettagliate le scelte progettuali dopo aver analizzato le possibili alternative esistenti. In particolare verrà descritta l'architettura della piattaforma Symbian, di PJSIP, un insieme di librerie per gestire le comunicazioni multimediali attraverso SIP, e del framework Qt.
Nel terzo capitolo, dopo aver descritto le specifiche desiderate, si effettuerà un'analisi dei requisiti funzionali e non funzionali e successivamente si mostrerà l'applicazione realizzata utilizzando gli screenshot effettuati su un dispositivo mobile.
Il quarto capitolo fornirà i dettagli sull'implementazione dell'applicazione, dunque com'è stata strutturata, l'ambiente di sviluppo utilizzato con le istruzioni per l'installazione su un calcolatore e il processo di certificazione e installazione su un dispositivo mobile. In ultima analisi saranno evidenziate le problematiche riscontrate durante il processo di sviluppo.
Infine verrà ricapitolato il lavoro svolto e i possibili sviluppi futuri.

Capitolo 1

Scenario

1.1 VoIP

VoIP, acronimo di Voice over Internet Protocol, è una tecnologia che rende possibile effettuare una conversazione telefonica utilizzando il protocollo IP. Più specificatamente con VoIP si intende l'insieme delle tecnologie che descrivono la trasmissione digitale della voce nelle reti a commutazione di pacchetto.

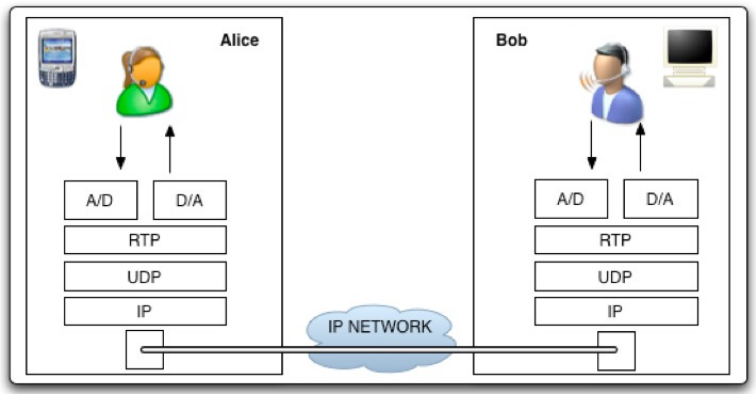


Figura 1.1: Conversazione VoIP

Le conversazioni VoIP non devono necessariamente viaggiare su Internet, ma possono anche usare come mezzo trasmissivo una qualsiasi rete privata basata sul protocollo IP, per esempio una LAN.
La tecnologia VoIP richiede che vengano utilizzati simultaneamente due tipo-

logie di protocolli di comunicazione, una per il trasporto dei dati (pacchetti voce su IP), ed una per i messaggi di segnalazione della conversazione (ricostruzione del frame audio, sincronizzazione, identificazione del chiamante, ecc...). Per il trasporto dei dati, nella grande maggioranza delle implementazioni VoIP viene adottato il protocollo *RTP* (Real-time Transport Protocol), mentre per quanto riguarda la seconda tipologia di protocolli, vi sono due differenti proposte elaborate in ambito ITU e IETF, che sono rispettivamente *H.323* e *SIP* (Session Initiation Protocol).
H.323 è un protocollo creato nel 1996 che propone uno standard per le sessioni di comunicazione audio, video e dati, attraverso reti orientate alla connessione nelle quali non è garantita la qualità del servizio (QoS). Si tratta di un protocollo indubbiamente stabile e collaudato, ma realizzare una rete che sfrutti tale standard risulta molto costoso e complesso.
SIP nasce come alternativa ad H.323, ed è stato creato come protocollo per la comunicazione voce in tempo reale su IP. Possiede un'architettura modulare e scalabile, ovvero capace di crescere con il numero degli utilizzatori del servizio e a differenza di H.323 è caratterizzato da una minore complessità. Queste caratteristiche hanno permesso a SIP di essere attualmente il protocollo VoIP più diffuso nel mercato residenziale e business.

1.2 Il protocollo SIP

Session Initiation Protocol (SIP) è un protocollo di segnalazione definito dalla IETF (Internet Engineering Task Force) nel Marzo 1999, poi aggiornato dalla RFC 3261 del Giugno 2002. Si tratta di un protocollo testuale operante a livello applicazione all'interno dello stack ISO/OSI, ed è basato su un modello di richiesta/risposta il cui scopo è quello di permettere a due o più partecipanti di stabilire una sessione, cioè uno scambio di dati tra un'associazione di partecipanti, composta da flussi multimediali multipli, come audio, video o altre tipologie di comunicazione come ad esempio i giochi distribuiti e le applicazioni condivise.
SIP è fortemente ispirato al protocollo HTTP e definisce la tipologia di messaggi, metodi ed entità sfruttando protocolli preesistenti per compiti specifici. Le entità logiche definite sono: User-Agent (UA), Redirect Server, Proxy

Server e Registrar Server. La comunicazione tra le varie entità è basata principalmente sul modello client-server ed alcune di queste entità possono svolgere entrambi i ruoli in base al tipo di operazione in corso.

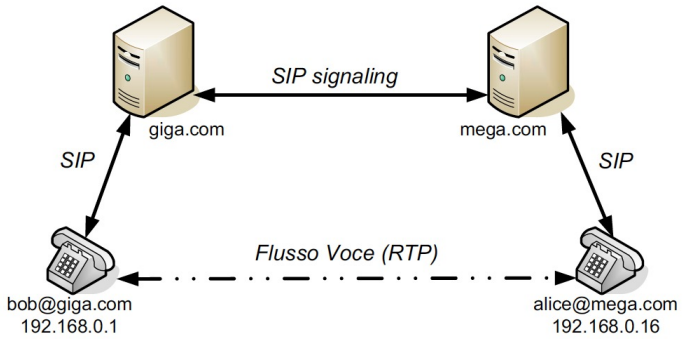


Figura 1.2: Trapezio SIP

L'architettura di sistema punta a concentrare la logica di protocollo presso gli User-Agent, con lo scopo di mantenere l'infrastruttura intermedia (Proxy Server, Registrar Server, ecc.) il più semplice e minimale possibile. In questo modo si ottiene un sistema molto flessibile e scalabile. Gli UA rappresentano, per l'utenza, il punto di accesso ad un sistema SIP e si dividono in UAC (User Agent Client) ed UAS (User Agent Server). Ogni UA può svolgere entrambi i ruoli, in base al tipo di operazione in corso. Esempi tipici di User-Agent sono i softphone e i dispositivi dotati di supporto VoIP, tramite i quali l'utente può instaurare sessioni di varia natura, come audio, video e messaggistica istantanea.
Il compito di descrivere e gestire le caratteristiche di ogni tipologia di sessione è affidata ad un ulteriore protocollo denominato *SDP* (Session Description Protocol). Più precisamente tale protocollo è usato da SIP durante la fase di creazione o modifica di una sessione per negoziarne le caratteristiche specifiche.
Il protocollo più usato nel trasporto dei messaggi di segnalazione è di tipo connectionless e si chiama UDP (User Datagram Protocol). UDP non gestisce il riordinamento dei pacchetti e la ritrasmissione di quelli persi ed è quindi generalmente considerato di minore affidabilità, ma in compenso è molto rapido ed efficiente e quindi ideale per le applicazioni multimediali che

richiedono di spedire una grande quantità di pacchetti. Recenti revisioni di SIP permettono comunque di utilizzare TCP (Transmission Control Protocol) anzichè UDP e TLS (Transport Layer Security) per la cifratura dei dati tramessi sulla rete.
I flussi multimediali sono invece gestiti dal protocollo RTP (Real-time Transport Protocol).
In sintesi ogni protocollo utilizzato da SIP ha un compito molto specifico e questa caratteristica fornisce un elevato grado di flessibilità, favorendo inoltre la possibilità di future espansioni dei servizi forniti.

1.2.1 Indirizzi

Un SIP-URI (SIP Uniform Resource Identifier) rappresenta lo schema di indirizzamento utilizzato per chiamare un altro soggetto attraverso il protocollo SIP. In altre parole, un SIP-URI è il recapito telefonico SIP di un utente.
Il formato standard per definire un indirizzo SIP è simile a quello per identificare gli indirizzi di posta elettronica:

```
sip:<username>@<hostname>
```

username: può essere un nome utente o un numero telefonico. Esistono infatti provider che propongono un servizio a pagamento di instradamento delle chiamate SIP su rete telefonica tradizionale;

hostname: rappresenta un dominio o un indirizzo IP.

La porta UDP di default è la 5060, ma può essere cambiata specificandola all'interno dell'indirizzo. Inoltre possono anche essere inseriti altri parametri che descrivono ulteriormente le caratteristiche del contatto. Di seguito alcuni esempi:

```
sip:giorgio@mioDominio.it;transport=TCP
sip:carlo@194.150.20.21:5068
sip:+39-81-12345@mioGateway.com;user=phone
sips:luca@mioDominio.it
```

Come avviene per gli indirizzi e-mail, anche per SIP è possibile inserire un collegamento ipertestuale all'interno di una pagina web.

1.2.2 Entità

I componenti del network SIP sono molteplici e a ciascuno è assegnato un particolare compito.

SIP User Agent

Le entità fondamentali di SIP sono gli User Agent (UA), dispositivi in grado di accettare oppure richiedere una comunicazione. Gli UA sono distinti in client UAC (User Agent Client) e server UAS (User Agent Server). Tali entità logiche si scambiano richieste e risposte consentendo a due o più terminali di comunicare tra loro. La sessione ha inizio quando si risponde positivamente al messaggio di *Invite* e termina con un messaggio di *Bye*.
Lo User Agent rappresenta l'unica entità con la quale l'utente interagisce direttamente, ed è quindi il punto di accesso ad un sistema SIP. Ad ogni utente possono essere associati molteplici UA: un utente può, ad esempio, essere rintracciabile contemporaneamente presso il proprio palmare, smartphone e computer.
Il compito di uno UA è quello di fornire un'interfaccia per la gestione dei servizi offerti dal sistema che nel caso di VoIP sono tipicamente conversazioni audio o video, messaggistica istantanea o conferenze. Di conseguenza, l'interfaccia deve contenere gli strumenti utili a gestire tali servizi, il cui numero e tipologia varia in base al terminale utilizzato.

SIP Registrar Server

Un Registrar Server è un server che accetta richieste di tipo REGISTER e memorizza le informazioni ricevute per la gestione del servizio di localizzazione. Il suo compito è quindi quello di accettare o rifiutare l'ingresso di uno UA all'interno del network SIP durante la fase di autenticazione.

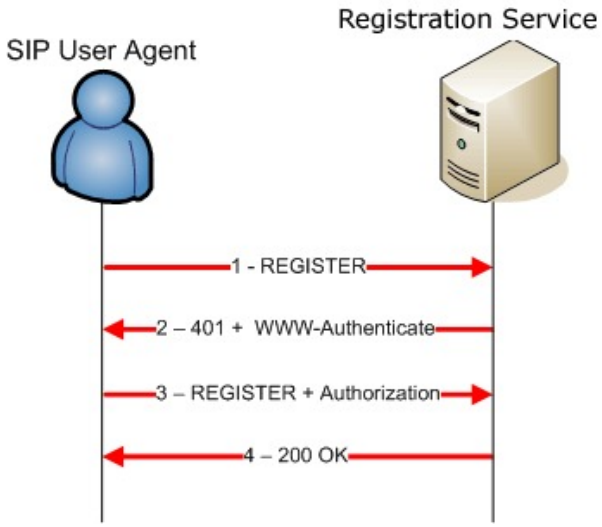


Figura 1.3: Autenticazione

Nel messaggio di REGISTER è specificata una lista di indirizzi ai quali l'utente può essere contattato. Questo permette all'utente di essere rintracciato su più dispositivi contemporaneamente.

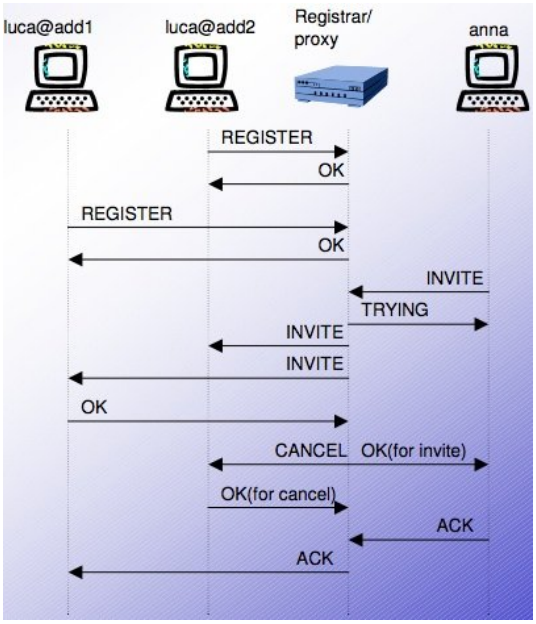


Figura 1.4: Utente registrato a più indirizzi

All'interno del messaggio è inoltre specificato il periodo di validità della registrazione, scaduto il quale il Registrar Server considera l'utente disconnesso. Di conseguenza, uno UA può inviare multipli messaggi di REGISTER, sia per rinnovare una registrazione prossima alla scadenza, sia per aggiornare la lista degli indirizzi. Quest'ultima operazione avviene quando l'utente disattiva uno UA precedentemente registrato oppure quando cambia il punto di accesso e dunque l'indirizzo IP.

SIP Proxy Server

Un SIP Proxy Server è una entità intermedia che agisce sia da server che da client. Fondamentalmente inoltra i messaggi tra gli UA coinvolti nella conversazione, replicandone i comportamenti. Per questo motivo, i messaggi generati sono percepiti dagli UA di destinazione come trasmessi dal proxy stesso. Richieste e risposte possono attraversare più Proxy Server prima di raggiungere la destinazione finale.

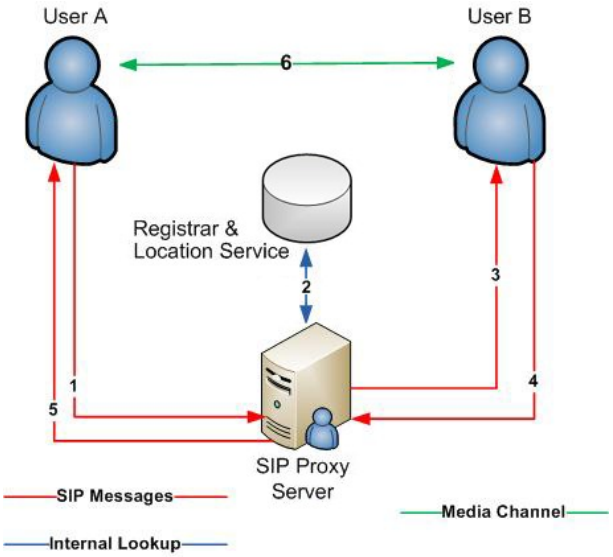


Figura 1.5: SIP Proxy Server

In generale i Proxy Server possono essere classificati in base alla quantità di informazioni che preservano ed utilizzano durante una sessione. Una gestione passiva (Stateless Proxy) prevede infatti che il proxy si limiti ad in-

stradare opportunamente il messaggio, mentre una gestione attiva (Stateful Proxy) può prevedere funzionalità aggiuntive e arbitrariamente complesse, che si sviluppano per tutta la durata della sessione stessa e che richiedono il salvataggio di opportune informazioni di stato:

Stateful Proxy: ciascuna richiesta SIP viene eseguita considerando le informazioni ottenute dalle richieste precedenti. Vengono mantenute informazioni sullo stato delle sessioni esistenti, permettendo l'uso di alcuni servizi aggiuntivi, come la ritrasmissione delle richieste in mancanza di risposta (liberando l'UA da questo compito e riducendo l'utilizzo delle risorse di rete), oppure l'autenticazione. Un tipo di Stateful Proxy è il Transaction Stateful Proxy, che si limita a tenere traccia dei messaggi fino a quando la transazione non viene eseguita, cioè dalla ricezione della richiesta alla ricezione della risposta definitiva. Rientra in questa categoria anche il Call Stateful Proxy, che partecipa alla gestione di una sessione per tutta la sua durata ed è quindi in grado di collezionare ed utilizzare informazioni sullo stato di un'intera sessione.

Stateless Proxy: tale tipologia di proxy non conserva alcuna informazione di stato. Per ogni richiesta o risposta pervenuta si limita a inoltrare il messaggio all'entità SIP successiva, sfruttando unicamente informazioni di instradamento fornite nel messaggio stesso.

SIP Redirect Server

Il compito di un Redirect Server è quello di semplificare la localizzazione di un utente, fornendo informazioni opportune su dove si possa trovare il destinatario ricercato. Per ricercare gli utenti si appoggia direttamente ad un Database oppure ad un Location Service. Quando il Redirect Server riceve una richiesta, per determinare dove essa vada inviata consulta il Location Service, il quale fornisce il legame utente-indirizzo IP o il riferimento ad altri URI, in modo analogo al funzionamento del sistema DNS di Internet.

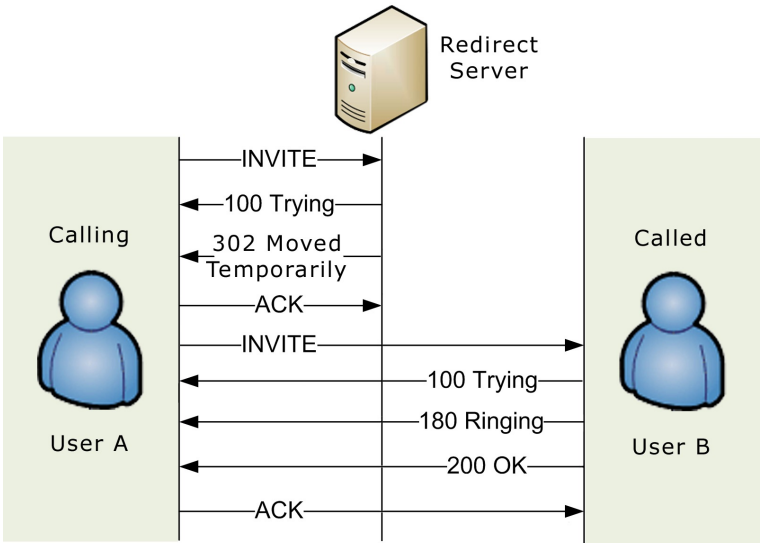


Figura 1.6: Redirect Server

Un Redirect Server non si limita a fornire l'intera lista dei contatti attivi (ovvero degli UA) appartenenti ad un utente, ma suggerisce anche dove è più probabile che quest'ultimo si trovi al momento della richiesta. Ogni utente può infatti specificare dove può essere rintracciato ad una determinata ora della giornata, o in un determinato giorno settimanale. Nel caso in cui il Redirect Server non riesca a recuperare il contatto diretto dell'utente ricercato, può restituire l'indirizzo di un ulteriore Redirect Server in possesso di informazioni più dettagliate. La possibilità di interrogare una serie di server a catena permette una gestione molto articolata e complessa della localizzazione di uno UA. Tale proprietà è valida non solo per i Redirect Server ma anche per i Proxy Server.
É interessante notare che l'utilizzo di un Redirect Server al posto di un Proxy Server ha spesso lo scopo di non sovraccaricare il dominio di pertinenza dell'utente cercato. Un Proxy Server si occupa direttamente della ricerca e del contatto dell'utente, impegnando quindi determinate risorse di rete. Al contrario, un Redirect Server si limita a fornire opportune informazioni, lasciando allo UA il compito di contattare l'utente in base alle informazioni ricevute.

Location Server

Teoricamente per effettuare una chiamata è sufficiente conoscere il contatto di destinazione. Nella maggioranza dei casi un utente non riesce ad instaurare una connessione diretta con l'UA di destinazione e ciò può succedere perchè il chiamante non riesce a contattare un DNS o non è capace di utilizzare le informazioni che ha a disposizione poichè si interpone un firewall o il chiamato si trova dietro un NAT e utilizza un IP privato.
Il Location Server consente l'instradamento di una chiamata anche qualora un chiamante non ha i mezzi per localizzare la destinazione. Tipicamente risiede sulla stessa macchina dove è presente il Registrar Server e contiene un database aggiornato costantemente dalle registrazioni degli utenti. Nel caso in cui non riesca a localizzare direttamente l'utente cercato restituisce gli indirizzi di proxy, gateway o altri Location Server che potrebbero conoscerne la posizione.

1.2.3 Messaggi

Il modello usato per la sintassi del protocollo SIP è text-based e trae ispirazione dal protocollo HTTP, di conseguenza ogni messaggio è anch'esso testuale e composto da un insieme di header più un eventuale body. Per instaurare una sessione si effettua un three-way handshaking concettualmente simile a quello che avviene con il protocollo TCP. La similitudine con HTTP risiede anche nella tipica logica di richiesta-risposta.

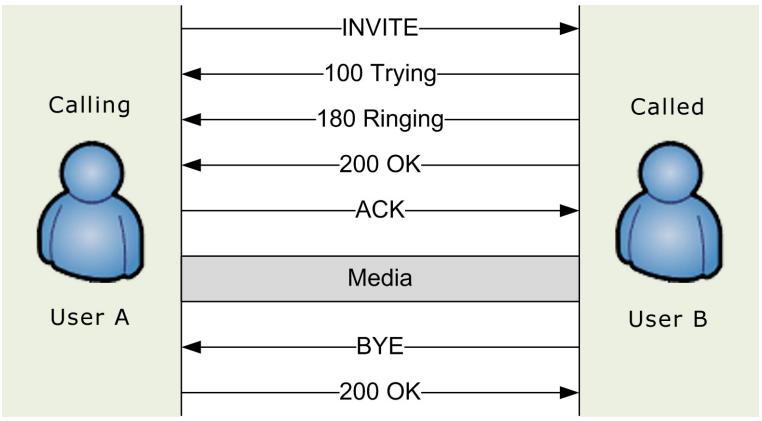


Figura 1.7: Esempio di sessione SIP

Messaggi di richiesta

Una richiesta è composta da una linea iniziale che specifica la release del protocollo SIP utilizzato, l'intestazione del messaggio, una linea vuota e il corpo del messaggio. Il formato di intestazione di una richiesta è composto da:

- **Method:** indica il tipo di operazione richiesta;
- **Request-URI:** indica il destinatario, cioè l'entità che dovrebbe processare l'operazione;
- **SIP Version:** indica la versione del protocollo SIP.

Un metodo definisce l'azione che si vuole intraprendere e le possibili operazioni possono essere:

INVITE: richiede l'apertura di una sessione. All'interno della richiesta sono presenti informazioni relative alla sessione che si vuole stabilire, definite tramite il protocollo SDP (Session Description Protocol). L'identificativo di sessione viene generato dallo UA che intende richiedere l'apertura della sessione e sarà utilizzato in tutti i messaggi successivi. É possibile generare un ulteriore messaggio INVITE, denominato re-INVITE, con lo scopo di rinegoziare oppure modificare i parametri di sessione. Un re-INVITE non può essere inviato durante l'instaurazione di una sessione. In questo caso è necessario utilizzare il metodo UPDATE.

ACK: conferma la ricezione di una risposta definitiva relativa ad una precedente richiesta di INVITE. Nel momento in cui il destinatario riceve un INVITE, viene generata immediatamente una risposta provvisoria (tipicamente 180, il telefono sta squillando), mentre l'eventuale risposta definitiva (200 OK) viene generata dal destinatario solo quando l'utente ha effettivamente accettato la chiamata. A quel punto l'UA genera un ACK per confermare la sua presenza e la ricezione del messaggio.

OPTIONS: richiede le capacità di uno UA. Un client si può servire di questo metodo per ottenere informazioni relative allo stato di un utente e alle funzionalità da esso supportate.

BYE: richiede la terminazione di una sessione attiva. Nel caso in cui la sessione coinvolga due partecipanti, l'abbandono da parte di uno di questi comporta la terminazione della sessione. Quando invece sono coinvolti più partecipanti non si hanno conseguenze sulla sessione.

CANCEL: richiede la cancellazione di una richiesta precedentemente trasmessa. Tipicamente viene utilizzato quando uno UA decide di interrompere un tentativo di chiamata. Una richiesta CANCEL non ha effetto nel caso in cui la sessione sia già stata instaurata.

REGISTER: richiede la registrazione di uno UA, ovvero informa il relativo Registrar Server che l'utente identificato da un determinato SIP-URI è contattabile sull'indirizzo fornito.

INFO: questo metodo viene usato per scambiare informazioni senza modificare lo stato della sessione. Le informazioni di interesse possono essere inserite in un apposito campo di intestazione INFO o semplicemente nel corpo del messaggio.

PRACK: consente ad un client di riscontrare la ricezione delle risposte provvisorie, che possono essere così trasmesse in maniera affidabile, al pari di quelle definitive.

SUBSCRIBE: richiede la sottoscrizione ad un evento al fine di ricevere notifiche attraverso richieste di tipo NOTIFY. La sottoscrizione implica la creazione di una sessione. Il messaggio contiene anche un campo che specifica la validità temporale della sottoscrizione, che può essere rinnovata all'interno della stessa sessione.

NOTIFY: rende possibile la notifica di un evento verificatosi su un nodo appartenente al network. La richiesta viene trasmessa all'interno di una sessione precedentemente creata mediante una richiesta SUBSCRIBE.

UPDATE: consente ad un client di aggiornare i parametri di una sessione (quali tipologie di flussi multimediali e relative codifiche) senza modificare lo stato della sessione. Il metodo è usato dopo un messaggio di INVITE, ma prima che la sessione sia stata instaurata.

MESSAGE: consente ad un utente di inviare messaggi istantanei nell'ambito di una sessione già stabilita. Il contenuto dei messaggi viene inviato all'interno del corpo della richiesta utilizzando il formato MIME (Multipurpose Internet Mail Extensions), tipico della posta elettronica e adatto al trattamento di dati multimediali.

REFER: con questo messaggio un client può invitare il destinatario a contattare un altro indirizzo, indicato in un apposito campo di intestazione denominato Refer-To. Questo meccanismo consente a SIP di supportare una serie di funzionalità, come il trasferimento di chiamata.

Messaggi di risposta

Il formato di un messaggio di risposta è il seguente:

- versione SIP;
- codice di stato;
- indicazione della ragione.

Il codice di stato è un numero di tre cifre che indica quale è stato l'esito della richiesta effettuata. La prima cifra identifica la classe a cui appartiene tale risposta, le restanti identificano una specifica risposta all'interno di tale classe. Lo standard prevede che per ogni richiesta invocata si possano ottenere una o più risposte provvisorie, ma al più una risposta definitiva.
Le classi di risposta, corredate da alcuni esempi di messaggio, sono le seguenti:

1XX In ricerca, squillo, accodato: indica una risposta informativa e provvisoria. Le risposte appartenenti a questa classe sono:

- **100 Trying:** il server SIP sta cercando l'utente destinatario della richiesta;
- **180 Ringing:** l'utente è stato localizzato e lo User-Agent in esecuzione sul suo terminale sta cercando di avvertirlo della chiamata. Si è in attesa di una risposta;

- **181 Call Is Being Forwarded:** si sta inoltrando la chiamata verso una diversa destinazione;
- **182 Queued:** l'utente non è al momento disponibile. La chiamata è stata messa in coda;
- **183 Session Progress:** questo messaggio può essere usato per comunicare al chiamante informazioni relative allo stato di una chiamata.

2XX Successo: indica che la richiesta è stata eseguita con successo. La risposta definitiva inviata dal server è 200 OK.

3XX Forwarding: indica che la richiesta è stata inoltrata. Questa classe di risposte fornisce informazioni riguardanti la nuova posizione dell'utente o i servizi alternativi che potrebbero portare a buon fine la chiamata.

- **300 Multiple Choices:** l'indirizzo fornito nella richiesta ha condotto all'individuazione di più terminali. L'utente dovrà quindi effettuare una scelta;
- **301 Moved Permanently:** l'utente cercato non è più raggiungibile all'indirizzo specificato nella richiesta. Il chiamante deve quindi inviare una nuova richiesta all'indirizzo fornito nel campo Contact dell'intestazione;
- **302 Moved Temporarily:** l'utente cercato non è temporaneamente raggiungibile all'indirizzo specificato nella richiesta. Il chiamante deve inviare una nuova richiesta all'indirizzo fornito nel campo Contact dell'intestazione. Può essere indicata la validità temporale di questo nuovo indirizzo in un apposito campo;
- **305 Use Proxy:** la risorsa a cui è destinata la richiesta deve essere acceduta attraverso un Proxy Server, il cui indirizzo è fornito nel campo Contact;
- **380 Alternative Service:** la chiamata non ha avuto successo, ma sono disponibili servizi alternativi descritti nel corpo del messaggio.

4XX Errori lato client: indica il fallimento, imputabile al client, della richiesta.

- **400 Bad Request:** la richiesta non è stata compresa a causa di una sintassi errata. La descrizione associata al codice di stato può fornire ulteriori informazioni sull'errore;
- **401 Unauthorized:** la richiesta necessita di un'autorizzazione da parte di un UAS o di un Registrar Server;
- **403 Forbidden:** la richiesta è stata rifiutata;
- **404 Not Found:** l'utente cercato non può essere trovato;
- **405 Method Not Allowed:** il metodo tentato non è consentito sull'indirizzo specificato. La risposta deve contenere un campo di intestazione con l'elenco dei metodi ammessi;
- **407 Proxy Authentication Required:** la richiesta necessita di un'autenticazione presso un Proxy Server;
- **415 Unsupported Media Type:** il corpo del messaggio si presenta in un formato non supportato;
- **420 Bad Extension:** il server non supporta o non comprende l'estensione del protocollo richiesta dal metodo e specificata nel campo Require o Proxy-Require dell'intestazione;
- **485 Ambiguous:** l'indirizzo indicato nella richiesta è ambiguo e non può quindi essere associato ad un utente in maniera univoca. Una serie di indirizzi simili non ambigui viene fornita dal campo Contact della risposta.
- **486 Busy Here:** il destinatario è stato contattato con successo, ma al momento è occupato.

5XX Errori lato server: indica un errore da parte del server.

- **500 Server Internal Error:** il server ha incontrato degli ostacoli imprevisti nel tentativo di soddisfare la richiesta. Se tale condizione è momentanea, il server può usare il campo di intestazione Retry-After per indicare al client quando ripetere la richiesta;

- **501 Not Implemented:** il server non supporta la funzionalità necessaria per soddisfare la richiesta. Questa risposta viene generata quando il metodo usato nella richiesta non è stato riconosciuto, distinguendosi pertanto dal codice 405 (Methot Not Allowed);
- **503 Service Unavailable:** il server al momento non è in grado di elaborare la richiesta a causa di un sovraccarico o di operazioni di manutenzione in corso. Il client può rivolgere la richiesta ad un altro server e non dovrebbe tentare di ricontattare il primo server prima del termine stabilito dal campo Retry-After della risposta;
- **504 Server Time-out:** il server non ha ricevuto una risposta in tempo utile da un server esterno.

6XX Occupato, rifutato, non disponibile: indica un errore globale.

- **600 Busy Everywhere:** il destinatario è stato contattato su ogni indirizzo disponibile, ma non può rispondere perché occupato. Il campo Retry-After dell'intestazione può suggerire al client un momento migliore per ritentare.
- **603 Decline:** questa risposta viene inviata se il destinatario, contattato con successo, non vuole o non può accettare la richiesta di comunicazione.
- **604 Does Not Exist Anywhere:** il server sa con certezza che l'utente indicato nel Request-URI non esiste all'interno della rete SIP.
- **606 Not Acceptable:** lo User-Agent cercato è stato contattato con successo, ma alcuni aspetti della descrizione della sessione presenti nella richiesta (quali mezzi necessari per la comunicazione, ampiezza di banda o stile di indirizzamento) non sono stati accettati. La risposta può contenere un elenco delle ragioni per cui la richiesta non può essere accolta.

1.2.4 Attraversamento NAT

Il Network Address Translation (NAT), ovvero traduzione degli indirizzi di rete, è una tecnica che consiste nel modificare gli indirizzi IP dei pacchetti ed è stata messa a punto per rispondere alla penuria di indirizzi IP forniti da IPv4. Spesso presente nei router e nei firewall, si divide in source NAT (SNAT) e destination NAT (DNAT), a seconda che venga modificato l'indirizzo sorgente o l'indirizzo destinazione del pacchetto con quello del router stesso. In tal modo è possibile per più terminali connessi ad una LAN condividere un unico indirizzo IP pubblico (NAT Dinamico). La traslazione di indirizzo permette di collegare solo delle richieste che provengono dalla rete interna verso quella esterna, il che significa che è impossibile per un terminale esterno inviare un pacchetto verso un terminale della rete interna. Per questa ragione, esiste un'estensione del NAT detta "ridirezionamento di porta"" (in inglese Port Forwarding o Port mapping) che consiste nell'inoltrare tutti i pacchetti ricevuti su una particolare porta verso un terminale specifico della rete interna.

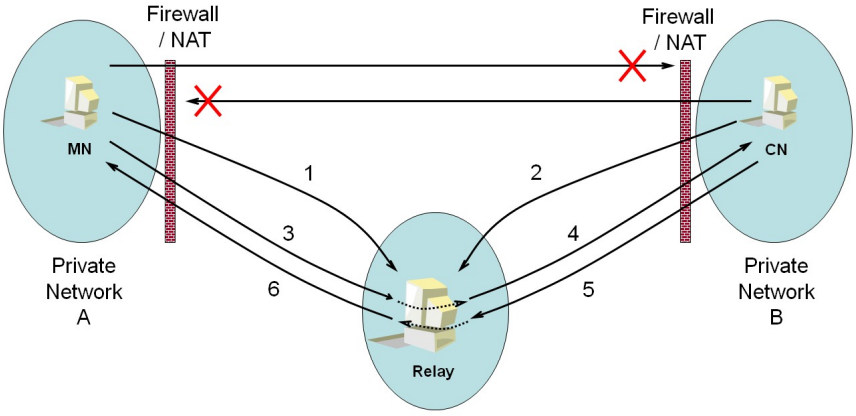


Figura 1.8: NAT e Firewall

L'utilizzo del NAT consente anche di proteggere gli utenti da attacchi dolosi provenienti dall'esterno, impedendo a nodi esterni di iniziare connessioni con quelli mascherati dal NAT, ma rappresenta anche uno dei maggiori ostacoli per le comunicazioni VoIP basate su SIP (e su protocolli associati, come RTP).

Il problema nasce quando la connessione è originata dall'esterno. In questo caso, l'utente che vuole avviare una comunicazione può solo indicare come destinatario l'indirizzo IP pubblico del router ma quest'ultimo, non avendo alcuna connessione attiva di riferimento, non sa dove recapitare i pacchetti. Il problema viene ulteriormente complicato a causa della varietà di NAT disponibili nati a causa dell'ampio numero di possibili scenari operativi ed architetturali:

Full Cone: tutte le richieste provenienti da un IP address interno e una certa porta vengono mappate sulla stessa porta dello stesso IP address esterno. Di conseguenza, qualsiasi host esterno può inviare un pacchetto all'host interno, inviandolo all'indirizzo esterno mappato. Un NAT Full Cone mappa quindi le sessioni attive di una coppia indirizzo interno/porta su una specifica coppia indirizzo esterno/porta. Tali sessioni possono essere iniziate sia da host interni che esterni.

Restricted Cone: tutte le richieste provenienti da un IP address interno e una certa porta vengono mappate sulla stessa porta dello stesso IP address esterno. Differentemente dal NAT di tipo Full Cone, un host esterno (con un dato IP address) può inviare un pacchetto all'host interno solamente nel caso l'host interno abbia precedentemente inviato un pacchetto a quello stesso IP address. Di conseguenza, solamente host interni possono iniziare una sessione attraverso NAT di tipo Restricted Cone.

Port Restricted Cone: simile al Restricted Cone NAT. La restrizione è estesa ai numeri di porta, perciò un host esterno puà inviare un pacchetto a un host interno soltanto se il secondo ha precedentemente inviato un pacchetto al primo.

Symmetric: tutte le richieste provenienti da un IP address interno e una certa porta vengono mappate sulla stessa porta dello stesso IP address esterno. La differenza risiede nella destinazione e nella dinamicità del mapping: ogni volta che indirizzo IP o porta di destinazione cambiano, sarà utilizzato un mapping diverso sull'interfaccia esterna del router, ovvero una differente combinazione pubblica di indirizzo e porta.

Per risolvere il problema dell'attraversamento di NAT quando si vuole instaurare una sessione SIP si può utilizzare STUN, acronimo di Simple Traversal of User datagram protocol through Network address translators. Si tratta di un protocollo e di un insieme di funzioni che permettono alle applicazioni in esecuzione su un computer di scoprire la presenza ed i tipi di NAT e firewall che si interpongono tra il computer e la rete pubblica. STUN permette a queste applicazioni di conoscere gli indirizzi IP e le porte con cui il dispositivo NAT li sta rendendo visibili sulla rete pubblica. STUN opera con molti NAT preesistenti e non richiede particolari configurazioni. Come risultato, STUN assicura ad una grande varietà di applicazioni IP (ad esempio, i telefoni VoIP) di lavorare attraverso le varie strutture NAT preesistenti [2].
STUN è un protocollo client-server, perciò un telefono o un software VoIP include un client STUN per comunicare con un server STUN. Quest'ultimo fornisce l'indirizzo IP e la porta UDP corrispondenti al traffico entrante del client e il tipo di NAT in uso. Ci sono tre tipi di NAT che è possibile attraversare tramite STUN: Full Cone, Restricted Cone e Port Restricted Cone. Poichè l'indirizzo IP del server STUN è differente da quello del client, il NAT simmetrico effettuerà mappaggi differenti e tale protocollo in tal caso non funzionerà.

1.2.5 Instradamento dei messaggi

Un sistema SIP basilare è composto semplicemente da due utenti (e relativi User-Agent), che possono comunicare direttamente tra loro a patto di conoscere i rispettivi contatti. In realtà questo scenario è estremamente raro sia per i motivi spiegati in precedenza, sia perchè i contatti di un utente possono variare frequentemente se l'utente si connette mediante una rete differente oppure se ha disattivato alcuni dei suoi UA.
Sin dall'inizio lo standard SIP ha cercato di favorire una comunicazione peer-to-peer tra gli UA, demandando a Registrar Server, Proxy Server e Redirect Server solo funzionalità di supporto cioè permettere una facile localizzazione degli utenti.
La realtà attuale di Internet rende tuttavia improbabile l'utilizzo di SIP in questa maniera a causa dell'utilizzo di firewall e NAT.

1.3 Multihoming

Il VoIP viene tipicamente classificato come servizio P2P (peer-to-peer): una conversazione che coinvolga due terminali prevede tipicamente l'instaurazione di un canale di comunicazione diretto tra di essi. Affinchè la comunicazione possa aver luogo è necessario che entrambi i terminali conoscano l'indirizzo IP dell'altro interlocutore. Nel caso in cui uno di essi sia un terminale mobile, un eventuale cambio di indirizzo IP causa la distruzione del canale di comunicazione. Il terminale fisso dovrà quindi recuperare il nuovo indirizzo IP del terminale mobile prima di poter ripristinare la comunicazione. La velocità con cui tale aggiornamento avviene influenza completamente la possibilità di continuare la comunicazione per i due interlocutori. Risulta quindi necessario utilizzare un supporto che consenta ad un dispositivo che si sposta da una rete ad un'altra, di essere raggiungibile da nuove richieste e di mantenere una sessione già avviata.
Uno smartphone moderno dotato di più di una interfaccia di comunicazione può utilizzare contemporaneamente più connessioni. Questa capacità è denominata *multihoming* e permette ad un terminale di continuare la trasmissione dei dati anche nel caso in cui una delle interfacce smetta di funzionare. Il multihoming è inoltre utile per distribuire il carico delle trasmissioni dei dati tra le diverse interfacce e per evitare episodi di inattività causati da disconnessioni non immediatamente rilevate, causate per esempio da una eccessiva diminuzione del segnale di ricezione o da interferenze temporanee.
In questo contesto, presso il dipartimento di Scienze dell'Informazione dell'Università di Bologna si sta lavorando nella realizzazione di un'architettura dedicata proprio al multihoming nota come *Always Best Packet Switching* (ABPS) [3].

1.3.1 L'architettura ABPS

Stato dell'arte

Le architetture per l'integrazione di reti eterogenee, note con il nome di *Seamless Host Mobility Architectures*, sono responsabili di identificare univo-

camente un nodo multihomed (MN), permettendogli di poter essere raggiunto dagli altri nodi con cui ha già effettuato una connessione e selezionando la rete migliore in modo da proseguire la conversazione. Queste architetture non hanno assunto una posizione ben definita all'interno del classico stack ISO/OSI e possono essere dunque implementate in ogni livello.
Nelle comunicazioni VoIP, l'indirizzo IP ha il compito di identificare il nodo, in quanto rappresenta l'univoca destinazione per i pacchetti provenienti dagli altri nodi con cui comunica. Trattandosi di un nodo mobile, esso riceverà un diverso indirizzo IP ad ogni ricollegamento, perdendo ad ogni selezione l'identità precedente. In questo modo, i nodi corrispondenti (CN) non riusciranno nuovamente a contattarlo prima di essere a conoscenza del suo nuovo indirizzo IP. Ne risulterà quindi una discontinuità all'interno della comunicazione.
La soluzione comune a tutte le architetture di *Mobile Management*, anche se implementate in strati ISO/OSI differenti, si basa su due semplici principi:

1. definire un identificatore univoco per il nodo, indipendente dalla provenienza dell'host;

2. fornire un servizio di localizzazione, sempre raggiungibile dal nodo corrispondente, per mantenere un'associazione tra l'identificatore univoco del nodo e il suo reale indirizzo di provenienza.

Il servizio di localizzazione è fornito dal Location Registry (LR), attivo su un server con indirizzo IP fisso e pubblico e raggiungibile da qualsiasi host. Se il CN è a conoscenza dell'identificatore univoco del MN, gli sarà sufficiente mettersi in contatto con il Location Registry per recuperarne l'indirizzo IP, per poi inizializzare o ripristinare una comunicazione diretta. Il Location Registry utilizza una funzione di mapping simile al DNS ed è un servizio esterno alla rete di provenienza dei nodi. Quando un MN necessita di aggiornare la propria locazione, invia un messaggio REGISTER al server, utilizzando il protocollo SIP. Quest'ultima soluzione non risulta efficiente perché il tempo impiegato dal nodo mobile per comunicare al server l'aggiornamento introduce un ritardo inaccettabile, durante il quale viene interrotta la comunicazione.

Requisiti del multihoming

I requisiti essenziali per un completo supporto alla seamless mobility nello scenario appena descritto sono i seguenti:

Trasparenza a livello utente: il roaming deve essere concluso il più velocemente possibile. L'utente non deve notare interruzioni nella comunicazione e nel caso in cui questo non risulti possibile, è necessario ridurre al minimo tali interruzioni.

Trasparenza a livello rete: non deve essere richiesto un esplicito supporto nelle varie reti di accesso. Queste devono solo garantire connettività su protocollo IP.

Compatibilità: la soluzione deve essere completamente compatibile con lo scenario preesistente, ovvero con relative entità e protocolli. In una comunicazione tra un terminale mobile ed uno fisso non deve essere richiesto supporto specifico da parte del terminale fisso. Questo quindi deve rimanere ignaro della mobilità del terminale corrispettivo.

QoS: la mobilità del MN deve essere gestita rispettando adeguatamente i requisiti di QoS.

Full-Mobile: deve essere supportata la possibilità che entrambi i terminali in comunicazione siano mobili. Essi devono quindi essere in grado di proseguire una comunicazione indipendentemente dai rispettivi spostamenti.

NAT-Friendly: la soluzione deve essere compatibile con la presenza di politiche di NAT sulle reti di accesso, in modo da non risultare di ostacolo alle preesistenti tecniche di NAT-Traversal.

Si noti che, in base al requisito *trasparenza a livello rete*, le reti di accesso sulle quali si sposta il terminale devono solo fornire connettività su protocollo IP. Non si vuole infatti porre alcun limite alle tipologie di roaming gestibili, in modo da fornire una mobilità completa. L'idea è quella di sfruttare il multihoming per fornire la continuità della comunicazione in piena mobilità, gestendo opportunamente eventuali riconfigurazioni dell'indirizzo IP utilizzato.

Scenario

Lo scenario a cui fa riferimento il modello *Always Best Packet Switching* (ABPS) [3] ha come oggetto principale la comunicazione VoIP su un dispositivo mobile, che può essere un laptop o, con maggiore probabilità, un dispositivo di telefonia mobile, equipaggiato con più di un'interfaccia di rete (NIC) wireless come WiMax, WiFi (IEEE802.11/a/b/g/n), GPRS, EDGE, 1xRTT, ZigBee o altre.

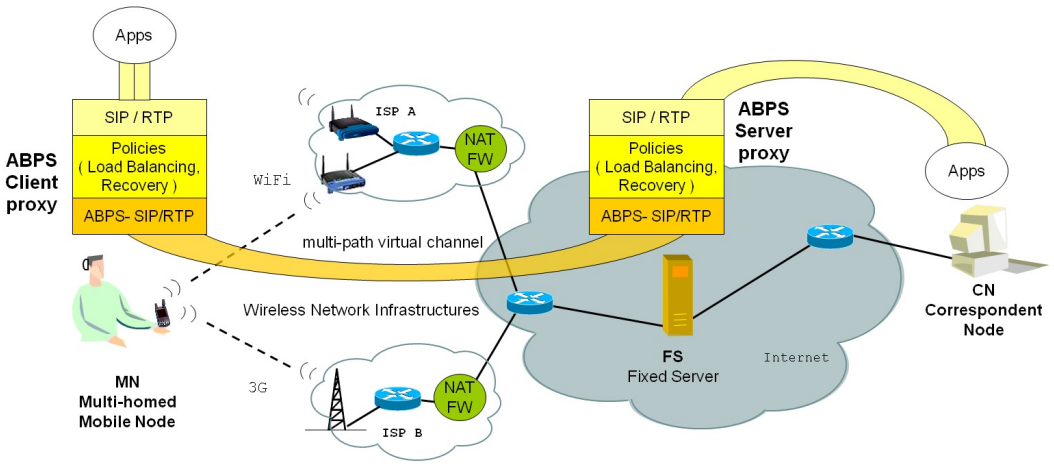


Figura 1.9: Il sistema ABPS

La figura 1.9 illustra lo scenario composto da un terminale VoIP multihomed equipaggiato con due o più interfacce wireless, una Wi-Fi 802.11b/g/n e una 3G, posizionato in una tipica area metropolitana che offre sia copertura 3G che WiFi. Il vantaggio di avere a disposizione interfacce di rete eterogenee sullo stesso dispositivo mobile è evidente soprattutto nel caso di interfacce wireless, perché rende possibile la connettività nel caso venga meno una delle connessioni. Una situazione del genere può verificarsi spesso in ambienti urbani, dove un utente si sposta in diversi punti della città cambiando access point e interfacce di rete, e viene definita dalle specifiche 3GPP come *Voice Call Continuity* (VCC) [4]. Un altro vantaggio è la selezione, nel caso sia disponibile più di un'interfaccia, del NIC preferito per accedere ad Internet. Quest'ultimo approccio è utilizzato dalla funzione di mobility management delle reti wireless, seguendo il modello di *Always Best Connected* (ABC) [5],

che suggerisce appunto di selezionare la migliore interfaccia NIC da usare come singolo punto di accesso a Internet. Nel caso in cui le prestazioni di un'interfaccia degradino eccessivamente, il dispositivo mobile rileverà una nuova interfaccia NIC preferita e la sostituirà alla prima.
L'architettura ABPS si basa su questo modello, ponendosi come scopo quello di offrire all'utente mobile il massimo della qualità di comunicazione, sfruttando al meglio le capacità aggregate di tutte le interfacce di rete disponibili.
A causa di limitazioni tecniche ed economiche, l'utilizzo di servizi multimediali come le conferenze audio/video su Internet mediante i dispositivi precedentemente descritti presentano ancora problematiche irrisolte, nonostante le continue innovazioni messe in campo. Le applicazioni multimediali VoIP e Video on Demand (VoD) risentono maggiormente di queste limitazioni tecniche perché hanno esigenze di Quality of Service (QoS) restrittive per poter offrire all'utente un buon livello di Quality of Experience (QoE). Ad esempio, le raccomandazioni ITU-T contenute nel documento G.1010 [6] stabiliscono che l'utente VoIP necessiti, per ottenere un servizio soddisfacente, di una latenza end-to-end minore di 150ms e una percentuale di pacchetti persi inferiore al 3%.
Il sistema ABPS è una soluzione completa ed efficace di QoS e Terminal Mobility per il VoIP wireless basato su SIP, ed è composta da due entità:

SIP Mobility: un'entità realizzata a livello applicazione che gestisce le conseguenze di un handover layer-3 (ovvero la riconfigurazione dell'indirizzo IP), che rispetti i requisiti precedentemente elencati.

Vertical Mobility: un'entità realizzata a livello data-link e network per monitorare lo stato di ogni interfaccia di rete presente sul dispositivo, gestendo in base ad opportune metriche e politiche la selezione dell'interfaccia di trasmissione.

La prima entità è realizzata mediante un server posto sulla rete pubblica, mentre la seconda è una applicazione eseguita direttamente sul terminale mobile. Il server rappresenta il punto d'ancora del terminale mobile ed ogni comunicazione VoIP di quest'ultimo passa attraverso di esso. In questo modo qualunque entità VoIP-SIP che debba comunicare con un altro terminale contatta il server stesso, credendo di dialogare effettivamente con la destinazione,

dunque la mobilità del terminale viene gestita direttamente ed esclusivamente dal server.
In conseguenza di una scelta progettuale dovuta soprattutto a questioni pratiche, la versione corrente del sistema ABPS prevede che l'applicazione includa diversi livelli con diverse funzioni. L'idea originale era infatti quella di utilizzare un proxy server ABPS in locale, sul dispositivo Mobile Node (MN), per aggiungere ai pacchetti SIP ed RTP gli header ABPS, interfacciandosi con il Fixed Server (FS) in modo del tutto trasparente all'applicazione client. In questa nuova implementazione invece viene integrato tutto in un'unica applicazione client specifica per ABPS, divisa in più livelli, ad ognuno dei quali è assegnato un compito specifico dell'architettura. In particolare l'applicazione completa è composta da un livello più basso di selezione dell'interfaccia di rete per effettuare handover e load balancing, un secondo livello formato da un insieme di funzioni e strutture per effettuare autenticazione mutuale e firma dei pacchetti, e di un terzo e un quarto livello che riguardano il client SIP vero e proprio, ovvero l'utilizzo dei livelli sottostanti per comunicare col Proxy Server FS e lo sviluppo della GUI di un softphone SIP.
Le due entità descritte, il client ABPS e il Proxy Server creano un tunnel logico tra esse permettendo la continuità della comunicazione in modo completamente trasparente, sia al MN che alle altre entità in comunicazione con quest'ultimo.

Capitolo 2

Obiettivi e Scelte Progettuali

Data la crescente diffusione nella vita di tutti i giorni di dispositivi mobili in grado di offrire funzionalità una volta esclusive dei soli pc, le aziende produttrici si sono dovute impegnare nello sviluppo di sistemi operativi sempre più flessibili ed affidabili in modo da garantire lo sviluppo di soluzioni software sempre più complete ed estese in grado di poter rispondere alle esigenze degli utenti più disparati.
Inizialmente uno smartphone era un terminale che integrava alle funzionalità di telefono cellulare, funzioni di gestione di dati personali come gli appuntamenti, i messaggi di posta elettronica e così via. Oggi sono anche personalizzabili poichè vi è la possibilità di installare nuovo software sviluppato dallo stesso produttore del terminale o da terze parti, dando così la possibilità di aggiungere nuove funzionalità a quelle già presenti. Inoltre la maggior parte degli smartphone presenti sul mercato attuale, anche se prodotti da diverse case costruttrici, offre tecnologie come touch-screen, bluetooth, connessione WiFi, connessione 3G, fotocamera/videocamera digitale che consentono agli sviluppatori di creare una vasta gamma di applicativi che possono spaziare in diversi settori di interesse. Ad esempio tecnologie come il VoIP, ampiamente testate e utilizzate su postazioni fisse, possono ora essere impiegate anche in contesti mobili.
Nel settore degli smartphone esistono diversi sistemi operativi ognuno con caratteristiche diverse e i più diffusi sono:

- Symbian OS di Nokia

- Android di Google
- iOS di Apple
- Windows Phone di Microsoft
- BlackBerryOS di Research In Motion

L'obiettivo di questa tesi è stato quello di fornire un software in grado di utilizzare le tecnologie presenti per poter permettere all'utente di comunicare tramite la rete. La piattaforma scelta è stata Symbian, poichè è presente nella maggioranza dei cellulari Nokia, uno dei maggiori produttori mondiali di dispositivi mobili, ma non solo. La scelta di Symbian è stata inoltre dettata dai seguenti vantaggi:

- è open-source (EPL Eclipse Public License);
- la Serie S60 di Nokia è molto diffusa in tutto il mondo ed i modelli di base che la utilizzano hanno costi relativamente moderati;
- fornisce strumenti di sviluppo validi;
- ha una comunità molto attiva.

Analizzando le applicazioni VoIP esistenti per questa piattaforma, si può affermare che non esistono soluzioni *open* in grado di fornire uno strumento user-friendly per effettuare comunicazioni VoIP.
Per questo motivo si è deciso di creare un client facile da usare che possa essere utilizzato in uno scenario wireless e che possa utilizzare in futuro il sistema *ABPS* in corso di sviluppo presso il dipartimento di Scienze dell'Informazione dell'Università di Bologna.
Affinchè ABPS possa essere facilmente integrato all'applicazione mobile che si intende sviluppare, si è deciso di utilizzare come backend SIP le librerie PJSIP anch'esse utilizzate da tale sistema.
Un altro punto cruciale per la realizzazione di un applicazione VoIP user-friendly per dispositivi mobili è la scelta di un framework per la gestione dell'interfaccia grafica che sia performante e portabile nelle architetture dei principali cellulari o smartphone.

Il mercato degli smartphone è ancora molto immaturo quando si parla di interoperabilità tra sistemi di diverso tipo e sta ripercorrendo gran parte degli errori che in passato vennero fatti nel mondo dei desktop. Infatti ogni azienda produttrice rilascia il proprio SDK e di conseguenza costringe lo sviluppatore ad utilizzare un set specifico di API per accedere alle funzionalità del particolare sistema operativo e per la creazione di interfacce grafiche. Oltretutto oramai quasi tutti i dispositivi richiedono la firma digitale che impedisce l'utilizzo di molti tools o di framework con licenze open-source, mentre per motivi di performance non possono essere presi in considerazione framework scritti in linguaggi interpretati come *Java*. Le possibilità di scelta prese in considerazione sono state dunque solo due:

- Nokia Qt
- wxWidgets

Sia Qt che wxWidgets sono framework multipiattaforma per la gestione di GUI (Graphical User Interface) scritti in C++, ma largamente utilizzati anche in altri linguaggi di programmazione grazie agli innumerevoli bindings offerti. Qt e wxWidgets sono disponibili gratuitamente come software open-source e dispongono di molteplici licenze per quel che riguarda sia sorgenti a codice aperto, che chiuso. In passato Qt era utilizzato con più diffidenza dagli sviluppatori viste le restrizioni insite nella sua licenza open/closed source, per questo motivo la Nokia ha deciso di rilasciarne una versione sotto LGPL. Entrambi questi framework supportano una vasta gamma di s.o. per pc: Linux, Mac, Windows, e BSD. Avendo raggiunto ormai due decadi di vita sono entrambi considerati maturi e stabili e per quanto riguarda la gestione delle GUI entrambi rendono disponibili i principali componenti grafici con la possibilità di crearne di personalizzati aggiungendo eventualmente il supporto 3D con openGL, l'internazionalizzazione ed una libreria multimediale.
La scelta è ricaduta sulle librerie Qt perchè offrono un pieno supporto per Symbian OS e perchè sono disponibili nativamente sui più recenti dispositivi Nokia (Maemo, MeeGo...) senza il bisogno di aggiungere librerie esterne.

2.1 Symbian OS

Symbian OS è l'erede del sistema operativo EPOC creato dalla Psion alla fine degli anni novanta per la sua linea di palmari. La sua nascita risale a giugno del 1998 quando fu creata la compagnia indipendente Symbian Limited nata dalla cooperazione di diverse compagnie telefoniche. Nel 2008 Nokia rileva tutte le quote azionarie delle altre società diventandone così l'unico proprietario e rende il sistema open source con la creazione di Symbian Foundation, dando così la possibilità a chiunque di poter sviluppare software per questo sistema.
Symbian OS venne realizzato sulla base di tre fattori stringenti:

- integrità e sicurezza dei dati;
- facilità d'uso;
- scarsa disponibilità delle risorse.

Per garantire questi principi, Symbian è stato concepito su un architettura microkernel e dispone di funzionalità di multithreading, multitasking e protezione della memoria. Ha un approccio di tipo *request and callback* per i servizi, tiene separata l'interfaccia utente dal sistema, salvaguarda il consumo di energia ed implementa il pattern di progettazione orientato agli oggetti Model View Controller. Nelle ultime versioni del sistema operativo sono stati integrati anche dei moduli per la sicurezza ed una versione real-time del kernel.
La programmazione in Symbian OS è basata su eventi e sulla tecnica *active-object*, dunque quando le applicazioni in esecuzione non generano eventi, la CPU si può porre nello stato di "basso consumo" salvaguardando le risorse energetiche. Allo stesso modo la gestione dei thread è realizzata in modo da non creare un eccessivo consumo di risorse.
Il kernel di Symbian OS, EKA2, è un micro-kernel che integra le funzioni essenziali per garantire massima robustezza, sicurezza e reattività, ed inoltre è ottimizzato per avere dei buoni tempi di risposta quando ha a che fare con applicazioni di tipo real-time.
Com'è possibile vedere dalla figura seguente, la piattaforma Symbian OS è costituita da più livelli [7]:

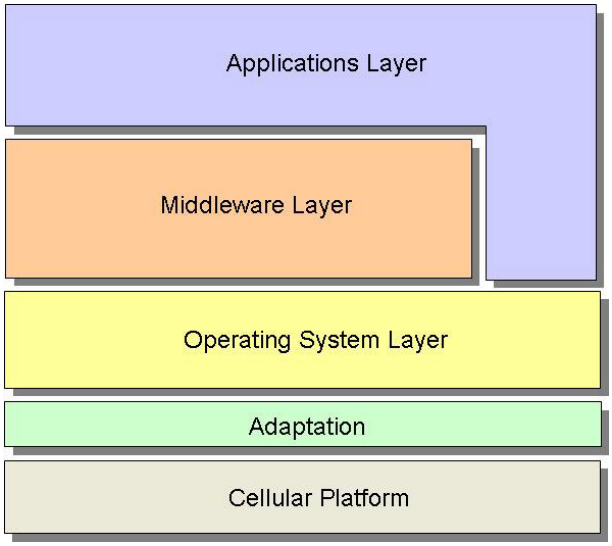


Figura 2.1: Architettura della piattaforma Symbian

- Application Layer: comprende i componenti e l'interfaccia utente di ogni applicazione e usa i servizi messi a disposizione dai livelli Middleware e Operating System;
- Middleware Layer: è suddiviso in domini applicativi (per es. multimedia, networking, ecc.) che forniscono servizi al livello superiore;
- Operating System Layer: fornisce una serie di servizi di alto livello che spaziano in un esteso range di domini tecnologici quali servizi di comunicazione, networking, grafica, multimedia. Inoltre fornisce anche una serie di servizi di basso livello come framework, librerie e utility che consentono di trasformare un dispositivo con hardware e sistema operativo specifici in un generico dispositivo programmabile;
- Adaptation Layer: Integra la piattaforma software generica con la piattaforma del telefono cellulare. Questo livello è implementato dal produttore del dispositivo;
- Cellular Platform: è l'hardware e il software specifico del device che esegue i servizi richiesti dalla piattaforma Symbian.

2.1.1 Sicurezza in Symbian OS

La piattaforma Symbian supporta lo sviluppo di applicazioni utilizzando il linguaggio C/C++. Se si vuole eseguire un programma sviluppato per tale piattaforma, è necessario "impacchettarlo" in un file di installazione ed infine installarlo sul device.
Prima dell'introduzione di un piano di sicurezza nell'architettura, le applicazioni potevano sfruttare liberamente tutte le risorse hardware del dispositivo e accedere anche a dati personali. Ciò rendeva estremamente vulnerabili i dati presenti nel dispositivo, come le impostazioni e il sistema stesso.
Security Platform è un insieme di tecnologie la cui funzione primaria è quella di controllare l'accesso ai dati e ai servizi di sistema da parte delle applicazioni. Essa è formata da tre componenti interconnessi:

- *Capability Model.* Questo modello impone che ogni applicazione che viene eseguita nel dispositivo debba dichiarare le sue "capacità".

- *Process Identity.* Ogni applicazione certificata sul dispositivo ha un security identifier univoco a livello globale. Tutti i server sono abilitati ad esaminare questo identificativo univoco e in questo modo si conoscerà sempre il processo che richiederà un particolare servizio. Inoltre ogni applicazione certificata sarà sempre accompagnata da un "vendor identifier" che identifica l'organizzazione che l'ha realizzata.

- *Data caging.* Il file system è suddiviso in parti con differenti livelli di accesso. In questo modo solo le applicazioni con opportune "capacità" possono avere accesso in lettura e scrittura a particolari zone del file system.

Con una piattaforma così costituita, l'accesso a servizi sensibili, come la connessione ad Internet e la rubrica, è garantito solo ad applicazioni certificate e dotate di opportune capabilities.
In Symbian OS vengono identificate due tipologie di applicazioni: trusted e untrusted, rispettivamente verificate e non. Le prime hanno il vantaggio di poter adottare tutte le capabilities di cui hanno bisogno, le seconde invece, non essendo verificate, possono disporre solo delle seguenti capabilities: ReadUserData, WriteUserData, Location, LocalServices, NetworkServices e

UserEnvironment. Nokia impone sui propri dispositivi l'esecuzione di sole applicazioni firmate (signed). In questo caso lo sviluppatore deve firmare il pacchetto con un certificato valido solo per il suo dispositivo ottenibile comunicando il proprio codice IMEI al servizio online fornito da Symbian. In questo modo sarà in grado di effettuare le operazioni di test necessarie prima di distribuire la propria applicazione.
Nel capitolo riguardante l'implementazione del progetto verrà ulteriormente descritta questa fase.

2.2 PJSIP

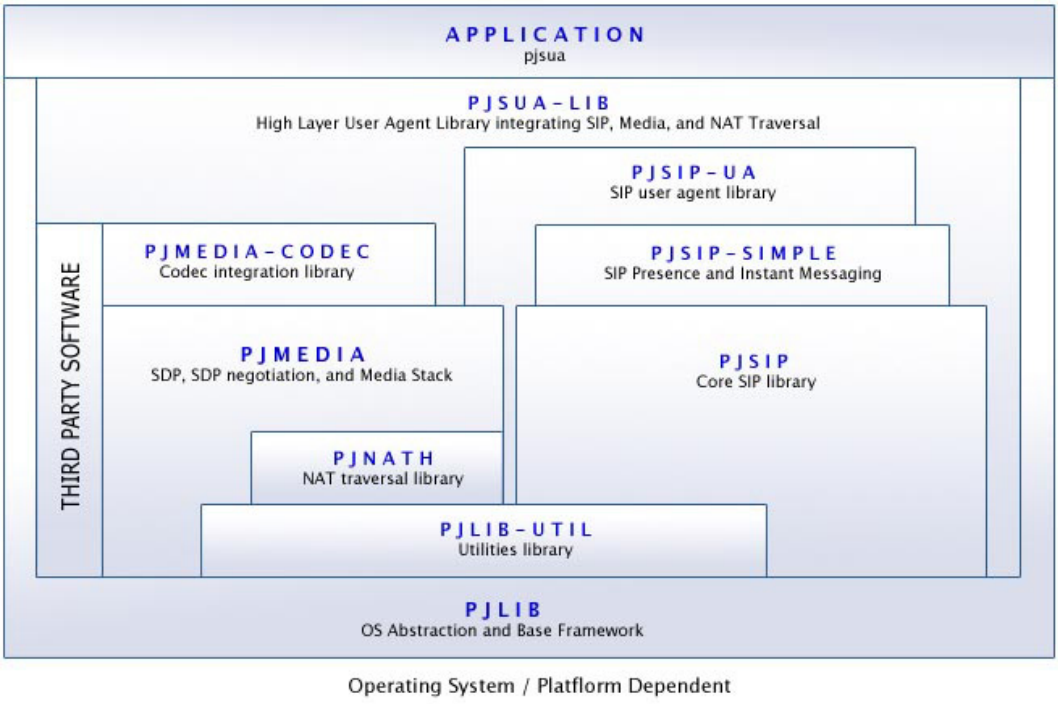


Figura 2.2: Architettura PJSIP

PJSIP è un gruppo di librerie per gestire comunicazioni multimediali attraverso il protocollo SIP [8]. Sono pensate per essere estremamente portabili e performanti, adatte per questo ad essere eseguite su architetture critiche, come ad esempio i dispositivi mobili, dove il consumo energetico e le risorse

hardware sono ridotte. Di seguito verrà descritto lo stack delle principali tecnologie che ne fanno parte e le particolarità che hanno fatto di PJSIP un backend ideale per le applicazioni VoIP su dispositivi mobili.

PJLIB

PJLIB è la libreria su cui tutte le altre si appoggiano, si occupa di garantire funzionalità di base quali l'astrazione rispetto al sistema operativo sottostante. Contiene una replica pressochè completa della libreria LIBC ed una serie di feature aggiuntive come la gestione dei socket, delle funzionalità di logging, dei threads, della mutua esclusione, dei semafori, delle critical section, funzioni di timing, nonchè un costrutto per la gestione delle eccezioni, e la definizione di varie strutture dati di base, come ad esempio liste, stringhe e tabelle di hash.

PJMEDIA

PJMEDIA è una libreria complementare e si occupa del trasferimento e della gestione dei dati multimediali. Tra le sue caratteristiche principali vi è la gestione degli stack RTP/RTCP, buffer adattivo per ridurre i problemi legati al jitter, rimozione degli echi, silence detector, generazione dei toni, supporto ad un'ampia varietà di codec tra cui speex/iLB-C/GSM/G.711.
Supporta l'encoding e il decoding di frequenze di 16/32 Khz e la conversione di qualunque campione verso queste frequenze, essa inoltre riesce a tollerare bene i problemi classici legati alle reti basate su IP che sono il jitter e la perdita di pacchetti.

PJSUA

PJSUA è una libreria di alto livello che fa da wrapper verso le altre librerie e consente di scrivere più agevolmente le applicazioni.

PJLIB_UTIL

PJLIB_UTIL è invece una libreria ausialiaria che fornisce supporto a PJMEDIA e PJSIP. Alcune sue funzioni sono: small footprint di parsing xml,

caching asincrono di risolver DNS, funzioni di hashing e di crittografia, ecc...

Nel dettaglio i punti principali che fanno di PJSIP una scelta ottimale sono i seguenti:

- **Portabilità:** può essere eseguita in svariati tipi di processori e sistemi operativi che siano a 16, 32 o 64 bit, little o big endian, singolo o multi-processore, con o senza capacità di calcolo in virgola mobile e supporto multi-threading del S.O. Può essere eventualmente eseguita anche in ambienti dove le librerie C ANSI non sono disponibili

- **Dimensioni:** uno degli obiettivi primari della libreria PJSIP è avere ridotte dimensioni in modo da essere utilizzata anche in ambiti embedded. É possibile infatti aggiungere o rimuovere funzionalità non necessarie in determinati ambiti o architetture, arrivando ad avere funzionalità VoIP minimali in appena 100Kb di spazio

- **Performance e gestione della memoria ottimizzata:** un punto fondamentale nello sviluppo di PJSIP è che la sua esecuzione sia più veloce possibile su ogni architettura supportata e per questo è stato deciso di evitare l'allocazione dinamica della memoria e di conseguenza l'utilizzo di funzioni quali malloc() utilizzando al suo posto un pool preallocato

- **Astrazioni del sistema operativo:** normalmente esistono alcune caratteristiche dei vari S.O. che rendono lungo e complesso lo sviluppo di applicazioni multipiattaforma. La libreria PJLIB, per ovviare ai classici problemi che la portabilità può causare, astrae alcune delle caratteristiche proprie dei vari sistemi operativi quali threads, sezioni critiche, semafori, eventi, manipolazione di date, ecc..

- **I/O di rete a basso livello:** PJLIB ha un set proprio di API per gestire le comuni azioni attraverso la rete.

Tali API rendono possibile ulteriori astrazioni quali:

- **Socket:** l'astrazione della socket permette di aggiungere nuove funzionalità di rete mantenendo la compatibilità sui sistemi supportati

- **Funzioni di risoluzione dell'indirizzo**
- **Strutture dati comuni:** fornisce tutte le operazioni comuni per creare e manipolare stringhe, array, hash table, linked list, alberi bilanciati
- **Gestione delle eccezioni:** un conveniente sistema di propagazione dell'errore ispirato al costrutto TRY/CATCH, semplificando così la scrittura del codice
- **Sistema di Logging:** le librerie PJLIB contengono un piccolo framework che permette di standardizzare la gestione dei messaggi di log. Possono essere configurati, per esempio, alcuni aspetti come la verbosità (messaggi informativi, di debug o di errore critico) o l'output (stdout, printf, file, etc..) mantenendo invariata la sintassi tra le diverse architetture supportate.

2.3 QT Framework

La realizzazione di interfacce grafiche in Symbian ha subito notevoli cambiamenti nel tempo: si è partiti con l'utilizzo del framework Eikon che supportava unicamente caratteri ad 8 bit e che permetteva la realizzazione di semplici interfacce grafiche monocromatiche, poi si è passati ad Uikon che supportava i caratteri Unicode, sino ad arrivare all'introduzione di un ulteriore livello al framework Uikon: Avkon.
Oggi è possibile sviluppare applicazioni per il sistema operativo Symbian utilizzando una particolare versione del C++, o utilizzando le librerie *Qt* [9], prima di TrollTeck poi dal 2010 di Nokia.
Qt nasce da un progetto di Haavard Nord e Eirik Chambe-Eng (i fondatori di Trolltech) nell'ambito della realizzazione di un framework per la creazione di interfacce grafiche in C++. Tale progetto divenne multipiattaforma quando venne loro commissionata la realizzazione di un'applicazione che consentisse di utilizzare le stesse API sotto ogni sistema operativo.
Nel 1993 venne rilasciata dai due programmatori la prima GUI realizzata col framework di loro produzione e successivamente, nel 1995 venne pubblicata la prima versione di Qt. Tale denominazione, che letteralmente significa "ca-

rino", nacque perchè ad Haarvard Nord piaceva come la lettera "Q" appariva nell'editor Emacs, mentre la "t" sta ad indicare *toolkit*.
Qt è una libreria multipiattaforma per definizione, infatti segue la filosofia *"write once, compile anywhere"*, che significa letteralmente *scrivi una sola volta e compila ovunque* e tramite il linguaggio di programmazione C++, consente di sviluppare applicazioni ad interfaccia grafica.
Grazie all'adozione di un'API (interfaccia di programmazione delle applicazioni) unica ed indipendente dall'hardware e dal software di sistema. Insieme con la libreria di classi, vengono forniti strumenti di supporto per il design grafico (Qt Designer), per la traduzione linguistica (Qt Linguist) ed il manuale in linea (Qt Assistant). Molte distribuzioni Linux sfruttano il toolkit Qt, poichè è la base su cui è stato costruito l'intero desktop environment KDE.
Qt è attualmente usato sia in ambito open-source, sia in ambito commerciale e supporta un numero sempre crescente di dispositivi mobili quali MeeGo, Maemo, Windows Phone, Symbian OS.
Nella programmazione di interfacce grafiche si richiede una buona efficienza a tempo di esecuzione ed un elevato grado di flessibilità. Per soddisfare questi requisiti il framework estende il C++ introducendo:

- un potentissimo meccanismo di comunicazione tra oggetti chiamato *Signals and Slots*;
- un sistema di eventi e filtraggio degli stessi;
- la possibilità di reperire dinamicamente le proprietà degli oggetti;
- internazionalizzazione delle applicazioni;
- un sistema di timer;
- una nuova gerarchia di oggetti, dove *QObject* ne è la radice.

La comunicazione mediante Signals and Slots e il sistema dinamico delle proprietà sono fornite dal Meta-Object System. Tale sistema estende il linguaggio C++ introducendo tre nuovi elementi:

1. la classe *QObject*, che è la classe radice su cui si basa il framework Qt;

2. la macro *Q_ OBJECT*, che, posta all'interno della sezione privata della dichiarazione di una classe, consente di utilizzare le funzionalità estese di Qt;

3. *il Meta-Object Compiler* (MOC), che effettua il parsing delle classi che utilizzano Q_OBJECT e produce un nuovo file sorgente contenente il codice del meta oggetto.

Una tipica dichiarazione di una classe che sfrutta il framework Qt e che verrà compilata dal MOC è la seguente:

```
class MyClass : public QObject {
  Q_OBJECT
  public:
    MyClass(QObject *parent = 0);
    ~MyClass();
  signals:
    void mySignal();
  public slots:
    void mySlot();
};
```

La creazione dei sorgenti che contengono i metadati può avvenire secondo due diverse modalità: una che prevede l'utilizzo diretto del MOC, gravando il programmatore dell'onere di inserire delle regole per la gestione del MOC all'interno dei makefile, l'altra che prevede l'utilizzo del tool *qmake* il quale crea automaticamente i makefile con le regole idonee per il corretto funzionamento del MOC.

2.3.1 La comunicazione tra oggetti: il sistema Signals and Slots

Tipicamente quando si sviluppano delle applicazioni ad interfaccia grafica, al variare dei dati si aggiorna qualche elemento della GUI e solitamente i framework dedicati a tale scopo implementano questo meccanismo utilizzando dei riferimenti a delle funzioni detti *callback*. L'utilizzo delle callback ha però i seguenti svantaggi:

- non sono type-safe, nel senso che non vi è la certezza che la funzione chiamante invochi una funzione callback con i parametri adeguati;
- una funzione callback è strettamente legata alla funzione chiamante, quindi quest'ultima deve conoscere con precisione la funzione da invocare.

Il sistema Signals and Slots di Qt, rappresenta una valida alternativa al meccanismo delle callback. Ogni oggetto Qt è fornito di una serie di signal e di slot, ma il programmatore può aggiungere di nuovi.
Un segnale viene emesso da un oggetto al verificarsi di un determinato evento, mediante il metodo `emit()`, mentre lo slot rappresenta l'azione, o la sequenza di azioni, da intraprendere quando questo viene attivato dal segnale al quale è collegato.
A differenza del meccanismo di callback, Signals and Slots è innanzitutto type-safe poichè la segnatura del segnale emesso deve coincidere con la segnatura dello slot che deve essere eseguito ed inoltre i segnali e gli slot sono disaccoppiati, perciò un segnale non deve conoscere lo slot a cui è collegato.
La connessione tra un segnale ed uno slot avviene mediante il metodo

```
connect(&sender, SIGNAL(signal()), &receiver, SLOT(slot()))
```

dove i parametri assumono il seguente significato:

- `&sender`: rappresenta l'oggetto che invia il segnale;
- `SIGNAL(signal())`: rappresenta il segnale che viene emesso dal sender;
- `&receiver`: rappresenta l'oggetto che riceve il segnale;
- `SLOT(slot())`: rappresenta lo slot che il receiver deve eseguire alla ricezione del segnale.

É anche possibile interrompere un collegamento mediante il metodo

```
disconnect(&sender, SIGNAL(signal()), &receiver, SLOT(slot()))
```

Il meccanismo di Signals and Slots prevede non solo la connessione tra segnale e slot, ma anche tra due segnali, tra un segnale e più slot e tra più segnali ed uno slot come viene mostrato nella figura sottostante.

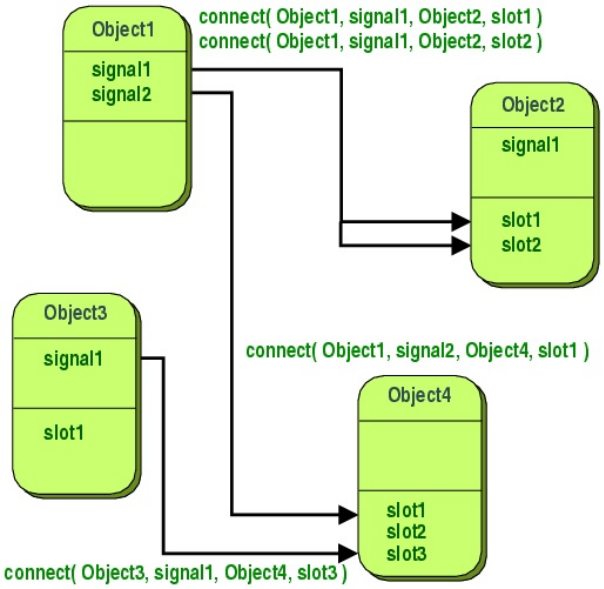


Figura 2.3: Esempio di connessioni tra QObject

Quando un segnale viene emesso, di solito l'esecuzione dello slot avviene istantaneamente e solo al termine della sua esecuzione il chiamante riprende il controllo, ma esiste anche la possibilità di creare una coda di connessioni e in tal caso l'esecuzione dello slot associato all'emissione di un segnale non avviene istantaneamente.
Tale tecnica risulta però essere circa 10 volte più lenta rispetto a chiamare direttamente la funzione da eseguire poichè è necessario localizzare l'oggetto, creare la connessione ed effettuare il marshalling dei parametri.
Di seguito viene mostrato un esempio di creazione di un segnale e di uno slot nella dichiarazione di una classe:

```
signals:
  void valueChanged(int newValue);
public slots:
  void setValue(int value);
```

A differenza del segnale, la creazione di uno slot oltre a richiedere la dichiarazione necessita dell'implementazione come avviene per qualunque altro metodo di una classe:

```
void Counter::setValue(int value){
  if(m_value != value){
    m_value = value;
    emit valueChanged(m_value);
  }
}
```

Nell'esempio di seguito oltre a definire due oggetti di tipo Counter viene creata una connessione tra il segnale `valueChanged(int)` dell'oggetto `a` e lo slot `setValue(int)` dell'oggetto `b`:

```
Counter a, b;
connect(&a, SIGNAL(valueChanged(int)), &b, SLOT(setValue(int)));

a.setValue(12);  // a.value() == 12, b.value() == 12
b.setValue(48);  // a.value() == 12, b.value() == 48
```

Quando verrà invocato il metodo `setValue(int)` di `a` (uno slot può essere utilizzato tranquillamente come un metodo) si provocherà la variazione dell'attuale valore di `a` e l'emissione del segnale `valueChanged(int)` che essendo collegato allo slot `setValue(int)` di `b` indurrà anche la modifica del valore di `b`.

2.3.2 QMake

É un tool che semplifica il processo di build di applicazioni su differenti piattaforme automatizzando la creazione dei "makefile". Il compito dello sviluppatore è scrivere il cosiddetto "project file" (un file con estensione .pro) dove vengono specificate le informazioni di base per la compilazione del progetto [10].
Un esempio di project file contiene le seguenti variabili:

```
QT += core gui
HEADERS += hello.h
SOURCES += hello.cpp
FORMS += hello.ui
```

Dove:

- `QT` serve per specificare quali moduli di Qt si vogliono utilizzare
- `HEADERS` serve per specificare la lista di file contenenti i prototipi delle funzioni da utilizzare durante la compilazione del progetto
- `SOURCES` serve per specificare la lista dei sorgenti da utilizzare durante la compilazione del progetto
- `FORMS` serve per specificare la lista di file ui che devono essere processati dal tool UIC

QMake prevede molte altre variabili per semplificare la fase di building del progetto, tra cui `LIBS` e `INCLUDEPATH` che servono rispettivamente per il *linking* e l'inclusione di librerie esterne.

2.3.3 Meta-Object Compiler

Il "Meta-Object Compiler" MOC, è un programma che gestisce alcune estensioni C++ proprie di Qt. Quando si lancia il tool, questo si occupa di leggere i file header e se trova una o più classi contenenti la macro Q_OBJECT produce un file sorgente contenente dei meta oggetti riferiti a quella classe.
Il file sorgente generato dal tool viene compilato e "linkato" all'implementazione della classe stessa.

2.3.4 User Interface Compiler (UIC)

Per velocizzare la creazione di interfacce grafiche e il loro successivo mantenimento, il framework Qt offre alcuni tool WYSIWYG (what you see is what you get) che le descrivono attraverso file XML (con estensione .ui). Il tool si occupa di convertire questi file di descrizione XML in file header C++, perciò è più facile per lo sviluppatore separare la parte di codice per generare la GUI da quella per implementare i requisiti funzionali dell'applicazione.

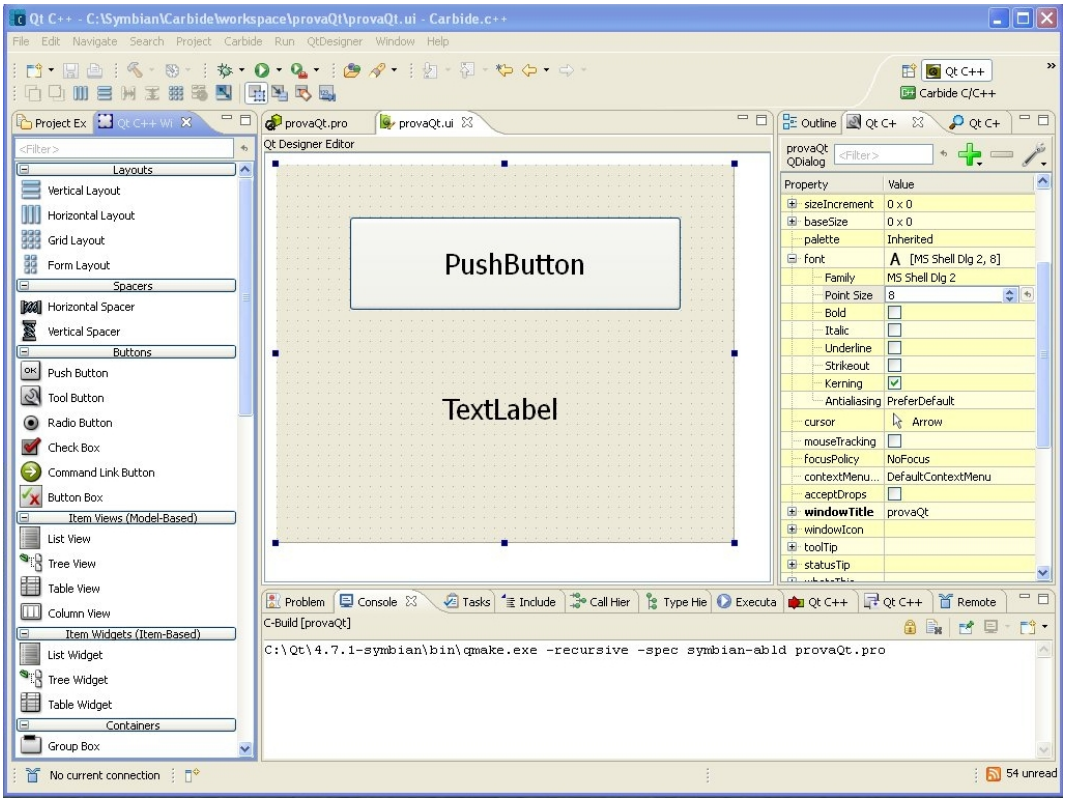


Figura 2.4: Tool integrato in Carbide per la creazione di interfaccie Qt

Capitolo 3

Progettazione

3.1 Analisi dei requisiti

3.1.1 Descrizione generale

Si vuole realizzare un'applicazione portabile per dispositivi mobili che consenta di effettuare chiamate e di utilizzare un servizio di messaggistica istantanea utilizzando la rete. Tale applicazione deve essere di facile utilizzo per l'utente medio, presentando un'interfaccia semplice e intuitiva. Il software inoltre deve essere facilmente configurabile dai meno esperti, ma allo stesso tempo deve fornire anche la possibilità di impostare dei parametri avanzati.

3.1.2 Requisiti funzionali

Il sistema che si vuole realizzare deve consentire all'utente di:

- inserire i parametri minimi per potersi registrare a un fornitore di servizi VoIP come lo username, la password e il dominio;
- poter effettuare chiamate tramite la rubrica;
- poter effettuare chiamate digitando da tastiera il contatto del destinatario;
- ricevere chiamate;

- inviare dei messaggi istantanei a un contatto;
- ricevere dei messaggi istantanei da un contatto.

3.1.3 Requisiti non funzionali

Inoltre il sistema consentirà di:

- accedere alla rubrica del telefono inserendo, modificando o cancellando i contatti in essa presenti;
- prevedere un registro delle chiamate in entrata e in uscita;
- consentire di personalizzare dei particolari parametri per la comunicazione come il protocollo utilizzato (UDP, TCP, SRTP), la porta locale su cui rimane in ascolto il client, lo STUN server e il PROXY server;
- consentire di pubblicare il proprio status di utente "online" o in caso contrario risultare "offline";
- visualizzare lo stato dei contatti registrati in rubrica;
- notificare all'utente eventuali errori di connessione o di utilizzo dell'interfaccia;
- visualizzare il log di PJSIP.

3.2 Realizzazione dell'applicazione

L'interfaccia utente dell'applicazione progettata consiste di un insieme di finestre che consentono la navigazione tra esse da parte dell'utente. In particolare le finestre utilizzate sono dette "di dialogo", cioè sono finestre che vengono solitamente utilizzate per brevi task e quando si vuole che l'utente e l'applicazione interagiscano con brevi comunicazioni.
La classe base fornita da Qt per realizzare questo tipo di finestre è chiamata *QDialog*. Una caratteristica di tale tipologia di finestra è che a differenza delle altre è di tipo top-level, cioè si trova sempre in cima allo stack delle finestre. Una finestra di dialogo può essere *modale*, cioè bloccare eventuali

input verso le altre finestre visibili nella stessa applicazione, o *non modale*, cioè può operare indipendentemente dalle altre finestre nella stessa applicazione. A sua volta una dialog modale può, una volta aperta, costringere l'utente a finire prima la sua interazione con essa e chiuderla prima di poter accedere a qualunque altra finestra nell'applicazione, o bloccare l'accesso alle sole finestre associate ad essa.[11]
Di seguito verranno mostrate le funzionalità dell'applicazione e i relativi screenshot ricavati dal dispositivo utilizzato per il testing: il Nokia E52.

Figura 3.1: Avvio dell'applicazione sul device Nokia E52

La finestra più importante è quella che contiene l'*Area Contatti*, in cui è possibile visualizzare la rubrica. Ogni dettaglio è costituito da nome, cognome, contatto e da una icona rappresentante il suo status (online oppure offline). Oltre alla lista dei contatti è presente un'area editabile dove è possibile digitare un contatto per chiamarlo direttamente.

Figura 3.2: Area Contatti

Per accedere a tale area è necessario essersi prima registrati al fornitore di servizi VoIP, per questo è prevista l'*Area di Login* in cui l'utente può inserire i suoi dati d'accesso

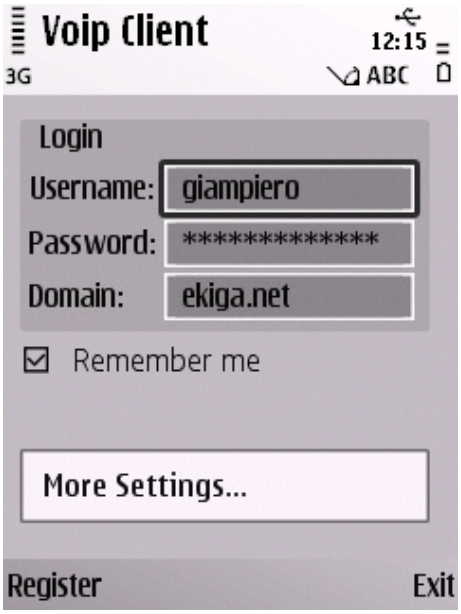


Figura 3.3: Area di Login - dettaglio login

e impostare i parametri per la comunicazione.

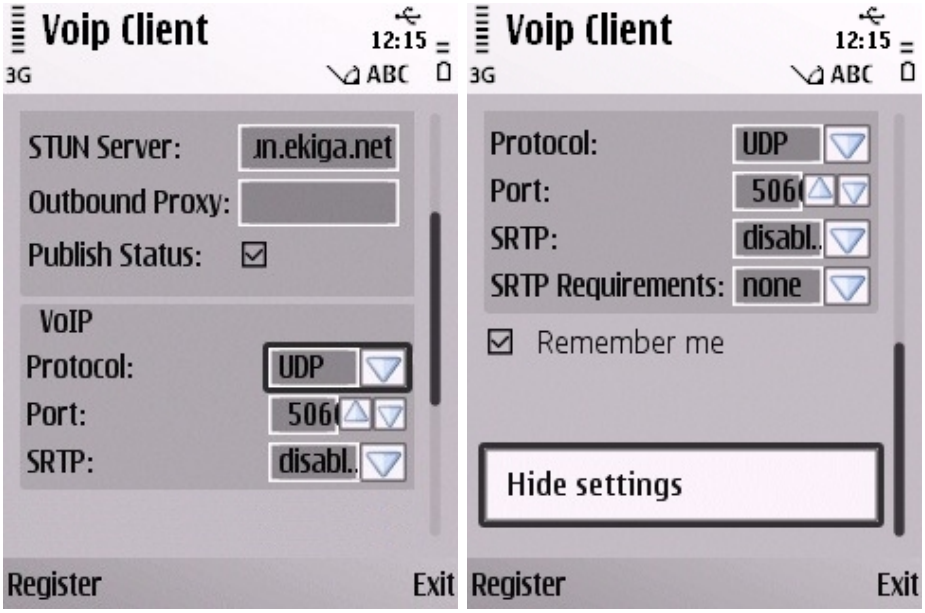


Figura 3.4: Area di Login - dettaglio configurazione

Dopo aver premuto il tasto “register” per la richiesta di registrazione, all’utente viene notificato il risultato dell’operazione.

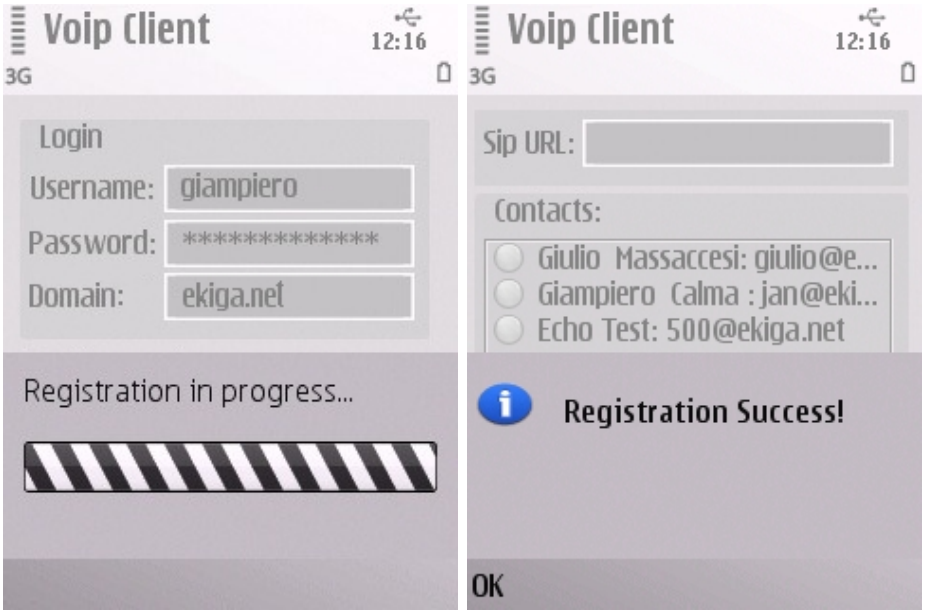


Figura 3.5: Area di Login - dettaglio registrazione

I progetti open-source esistenti per la comunicazione VoIP utilizzano soluzioni limitate per la gestione della rubrica. In particolare viene creata una rubrica dell'applicazione che non ha nessun tipo di relazione con quella del dispositivo. Questa soluzione consente di gestire i contatti solo quando l'applicazione è in esecuzione e ciò vuol dire che le modifiche ai contatti non sono visibili nella rubrica del dispositivo e viceversa.
L'Area Contatti progettata visualizza ed opera invece sugli effettivi contatti della rubrica del dispositivo mobile. Ciò significa che è possibile non solo effettuare tutte le operazioni di base che un utente medio si aspetta di trovare quali la modifica, la cancellazione e l'inserimento di contatti, ma anche di vederne i risultati nella rubrica del dispositivo. Inoltre è anche possibile effettuare l'operazione inversa, cioè modificare la rubrica del dispositivo e visualizzare, una volta lanciata l'applicazione VoIP, le variazioni che sono state fatte in maniera del tutto trasparente. Per operare sulla rubrica dall'Area Contatti è sufficiente accedere al menu a tendina utilizzando il tasto "options" del dispositivo.

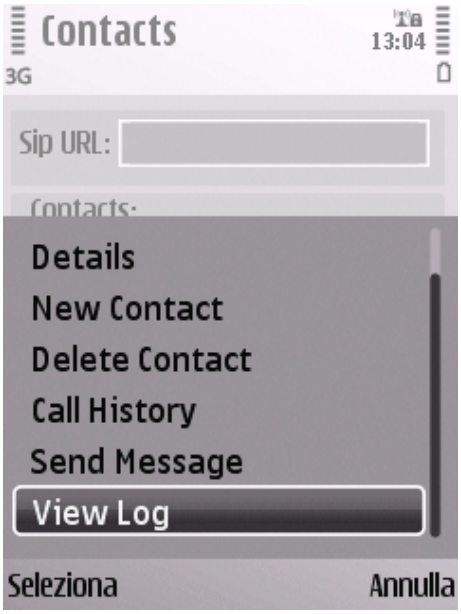


Figura 3.6: Area Contatti - menu delle opzioni

Se l'utente decide di inserire un nuovo contatto, viene visualizzata una form con tutti i campi vuoti, alcuni obbligatori (nome o cognome e contatto) ed altri opzionali,

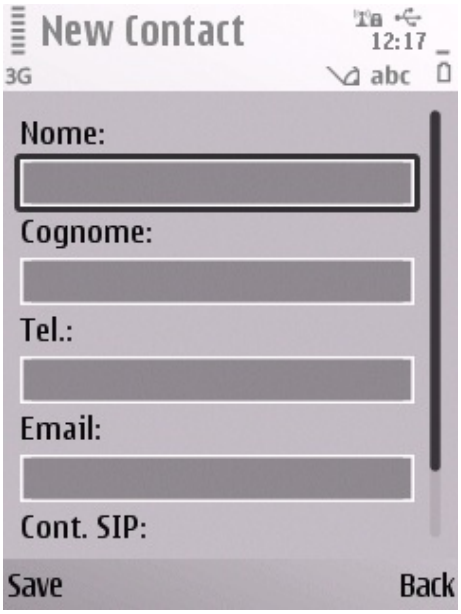


Figura 3.7: Area Contatti - dettaglio nuovo contatto

se invece l'utente decide di modificare un contatto già esistente, viene visualizzata una form con i campi già compilati, ma editabili

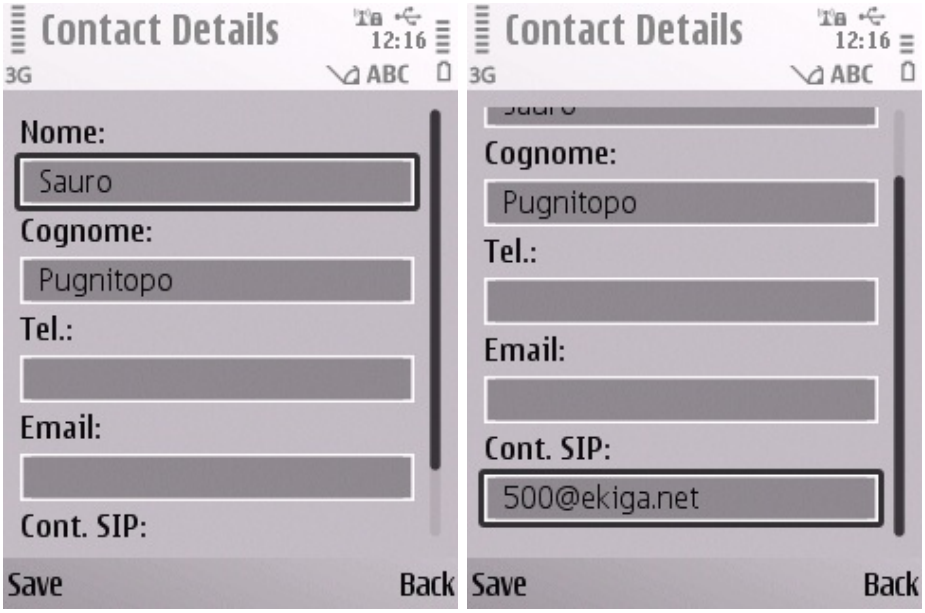


Figura 3.8: Area Contatti - dettaglio modifica contatto

e infine se decide di cancellare un contatto, l'operazione viene effettuata in maniera trasparente selezionando la corrispondente voce nel menu.
Il tasto "options" consente di effettuare anche altre operazioni, la più importante delle quali è l'invio di messaggi istantanei a un contatto della rubrica. Quando l'utente decide di effettuare tale operazione, viene visualizzata una nuova finestra in cui è possibile visualizzare lo stato della conversazione e inserire nuove stringhe di testo.

Figura 3.9: Dettaglio conversazione testuale

Per effettuare nuove chiamate è sufficiente selezionare un contatto dalla rubrica o digitarne uno nella linea editabile e premere il tasto di avvio chiamata o il tasto centrale del dispositivo. Per notificare all'utente l'avvio della chiamata, viene visualizzata una nuova finestra di chiamata in uscita

Figura 3.10: Avvio di una chiamata in uscita

mentre quando il contatto risponde, e dunque la chiamata è effettivamente in corso, viene aggiornato il contenuto della finestra

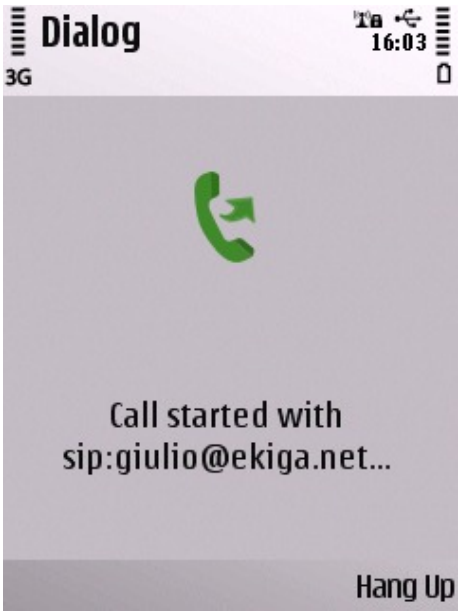


Figura 3.11: Chiamata in uscita instaurata

Per chiudere la conversazione è stato creato un tasto personalizzato, poichè negli ultimi modelli della Nokia (tra cui il dispositivo in nostro possesso) non è possibile personalizzare il comportamento del tasto di default per la chiusura della chiamata.
La ricezione delle chiamate segue lo stesso principio appena esposto, dunque ogni volta che se ne riceve una, viene visualizzata una nuova finestra di chiamata in entrata e il dispositivo emette una suoneria. L'utente può rispondere e attivare una nuova conversazione o rifiutare e tornare alla finestra che stava visualizzando.

Figura 3.12: Chiamata in entrata

Sia le chiamate in entrata che quelle in uscita vengono annotate in un registro delle chiamate esclusivo dell'applicazione. Si è scelta questa soluzione perchè le conversazioni attraverso la rete sono distinte dalle normali conversazioni vocali e non avrebbe senso inserire le chiamate vocali all'interno di un'applicazione VoIP e viceversa. Per accedere al registro è sufficiente utilizzare la voce corrispondente del menu delle opzioni della finestra dei contatti.

Figura 3.13: Registro delle chiamate

É possibile selezionare uno dei contatti nel registro ed effettuare una chiamata senza tornare alla rubrica.
Infine è stata aggiunta un'ultima voce al menu dei contatti che consente all'utente più esperto e allo sviluppatore di visualizzare una finestra di log per controllare le operazioni effettuate da PJSIP durante la registrazione e durante le operazioni di chiamata e di messaggistica istantanea.

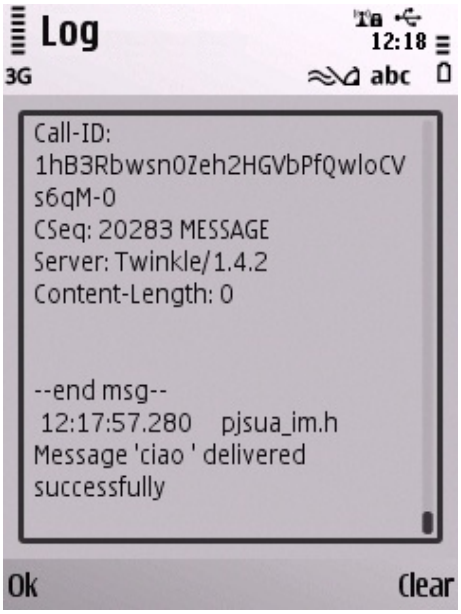


Figura 3.14: Log di PJSIP

Per chiudere l'applicazione e inviare una richiesta di sconnessione dal fornitore del servizio VoIP, l'utente può utilizzare il tasto di chiusura personalizzato o quello di default del dispositivo.

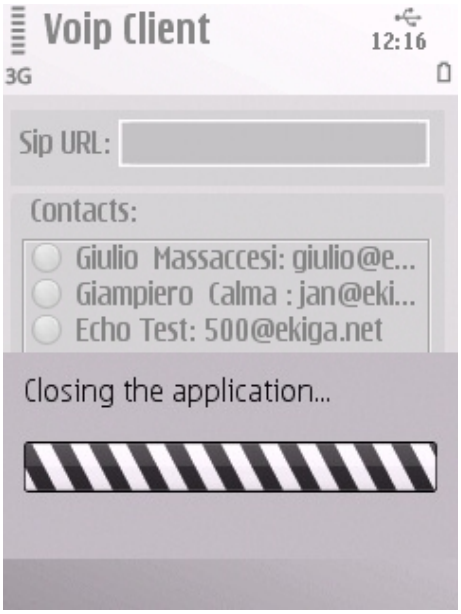


Figura 3.15: Chiusura dell'applicazione

Capitolo 4

Implementazione

Avendo precedentemente definito i requisiti che l'applicazione deve soddisfare e avendo descritto come l'applicazione dovrà interagire con l'utente nella fase di progettazione, in questo capitolo verrà descritta la fase implementativa del client realizzato.
Astraendo dai dettagli implementativi che riguardano il codice realizzato in C++, verrà inizialmente descritta l'architettura dell'applicazione elencandone le componenti principali e le interazioni che intercorrono tra loro. Successivamente verranno elencati gli strumenti che ci hanno permesso di realizzare il client VoIP (librerie, framework ma anche IDE, tool di debug, ecc...) descrivendone anche le procedure di utilizzo e installazione. Infine esamineremo in dettaglio le fasi di certificazione del pacchetto di installazione necessario per l'installazione dell'applicazione su dispositivi Symbian.

4.1 Architettura dell'applicazione

Per poter riutilizzare e mantenere le funzionalità implementate si è deciso di sviluppare l'applicazione su più strati intermedi così strutturati:

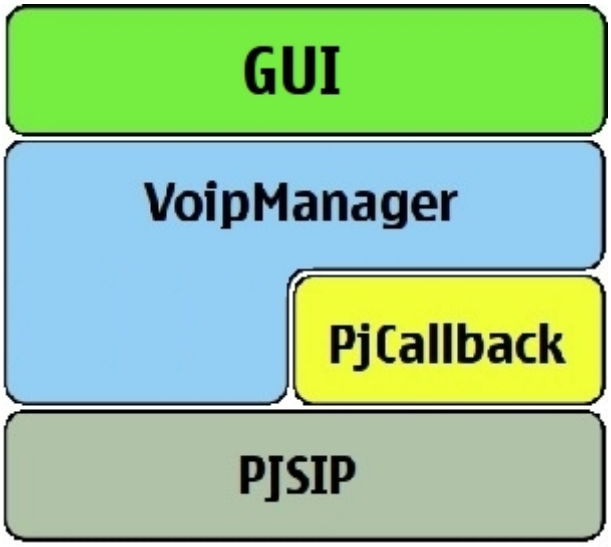


Figura 4.1: Architettura dell'applicazione

4.1.1 VoipManager

Questa classe permette all'applicazione di far dialogare l'interfaccia grafica sviluppata con le librerie Qt con gli strati sottostanti che implementano le funzionalità VoIP. L'obbiettivo è quello di creare una sorta di wrapper che nasconda tutti i dettagli implementativi relativi ai protocolli di comunicazione, utilizzo di codecs, utilizzo dell'hardware audio, ecc...
La classe VoipManager implementa metodi che verranno richiamati o tramite un'interazione con l'interfaccia grafica o tramite le notifiche provenienti da PjCallback. In questo modo l'utente potrà effettuare/ricevere chiamate, spedire/ricevere messaggi testuali, ecc... Di seguito elencheremo i metodi più rilevanti di VoipManager descrivendone il loro utilizzo:

- `register()` si occupa di registrare l'utente presso il server SIP utilizzato. Il client, prima di avviare la procedura di registrazione, andrà a leggere i parametri inseriti nei settaggi dall'utente che comprendono le credenziali di accesso (username e password) e il server registrar;
- `call(<destinatario>)` è usato per chiamare il contatto SIP passato come parametro. Se il contatto chiamato risponderà alla chiamata

allora sarà PJSIP che instaurerà un flusso audio tra i due utenti, nascondendo al programmatore tutti i dettagli relativi a socket, codecs, utilizzo della scheda audio del device, ecc...

- `answer()` è usato per rispondere a una chiamata in ingresso;
- `hangUp()` chiude la chiamata in corso;
- `sendIm(<destinatario>, <testo del messaggio>)` invia un messaggio testuale al destinatario specificato nel parametro;
- `ring()` si occupa di far squillare il telefono per segnalare una chiamata in arrivo;
- `subscribeBuddy(<nickname>, <indirizzo sip>)` per consentire la ricezione delle notifiche riguardanti il cambiamento di stato del contatto passato come parametro.

4.1.2 PjCallBack

Questo componente viene utilizzato per permettere alle librerie PJSIP di notificare agli strati superiori (VoipManager e Qt GUI) gli eventi SIP come per esempio una chiamata in ingresso, l'arrivo di un messaggio, il cambiamento di stato di un utente, ecc...
La notifica avviene convertendo i segnali provenienti da PJSIP in segnali Qt (i signal della tecnica signal-slot precedentemente mostrata).
Questi ultimi invocheranno gli slot corrispondenti nel VoipManager che si occuperanno di aggiornare l'interfaccia grafica.
Di seguito sono elencati i *signal* più importanti:

- `reg_state_signal()` notifica il risultato della procedura di registrazione sul server;
- `incoming_call(<utente>)` notifica una chiamata in ingresso;
- `seccallState()` notifica un cambiamento nello stato della chiamata (ad es. chiusura da parte dell'utente chiamato, rifiuto della chiamata);

- `new_log_message(<messaggio di log>)` notifica un messaggio di debug da visualizzare;
- `new_im(<mittente>, <testo>)` notifica l'arrivo di un messaggio testuale;
- `nat_detect()` notifica che tipo di NAT è stato rilevato;
- `buddy_state(<id contatto>)` notifica il cambiamento dello stato di un contatto.

4.2 Ambiente di sviluppo

Per poter sviluppare applicazioni Symbian è necessario disporre di un computer con Microsoft Windows. Nonostante sia possibile scrivere e compilare applicazioni anche su Linux e Mac OS, la creazione di un pacchetto di installazione è possibile solo su Windows.
Nei prossimi paragrafi saranno illustrati i passaggi necessari per la corretta installazione dell'ambiente di sviluppo. Ulteriori informazioni possono essere trovate a questo indirizzo:

`http://wiki.forum.nokia.com/index.php/SDK_and_Carbide.c%2B%2B_installation_guide`

Perl

ActivePerl è un interprete per Windows di script programmati in linguaggio Perl [1]. In pratica rende disponibile al programmatore il necessario per creare, testare e fare il debug dei programmi scritti con questo linguaggio.
La sua installazione è necessaria poichè l'SDK di Symbian fa un massiccio uso di questo linguaggio. É possibile scaricare ActivePerl al seguente indirizzo:

`http://www.activestate.com/activeperl/downloads`

[1] `http://it.wikipedia.org/wiki/Perl`

SDK

Il pacchetto Software Development Kit (SDK), fornito da Nokia, include tutte le risorse necessarie per lo sviluppo di applicazioni su Symbian, come compilatore, librerie di sistema (API), documentazione, esempi e un emulatore. É consigliabile installare il pacchetto all'interno della cartella C:
Symbian, per evitare problemi in seguito. Il Software Development Kit dà accesso alle API pubbliche, e lavorando esclusivamente su queste si ha la sicurezza che le applicazioni sviluppate funzioneranno sull'ampio ventaglio di device disponibili attualmente e in futuro. Il software si trova presso il sito di Nokia, al seguente indirizzo:

`www.forum.nokia.com/info/sw.nokia.com/id/S60_All_in_One_SDKs.html`

ADT

L'Application Development Toolkit (ADT) contiene gli strumenti necessari allo sviluppo delle applicazioni. All'interno del toolkit è presente un Integrated Development Environment (IDE) denominato Carbide.c++, basato su Eclipse[2], che costituisce il principale strumento per lo sviluppo C++ su questa piattaforma. I plugin di Carbide includono funzionalità di building, debugging, analisi statiche, analisi dinamiche e una varietà di strumenti ed utility specializzate. Carbide.c++ permette quindi allo sviluppatore di editare, costruire ed effettuare debugging di un'applicazione su emulatore o direttamente su smartphone. É possibile scaricare ADT al seguente indirizzo:

`https://developer-secure.symbian.org/main/tools_and_kits/downloads/view.php?id=2`

Nokia PC Suite

Nokia PC Suite consente di avere accesso al dispositivo Nokia attraverso il PC. Comprende una raccolta di programmi e consente anche di installare applicazioni direttamente sul cellulare. É possibile scaricare il software al seguente indirizzo:

`http://www.nokia.it/supporto/software/nokia-pc-suite`

[2] `http://it.wikipedia.org/wiki/Eclipse_(informatica)`

Qt

Il framework Qt come già descritto nei capitoli precedenti è stato utilizzato per la costruzione dell'interfaccia grafica.
Per installare il framework è necessario:

1. Scaricare il file da questo url:

   `http://qt.nokia.com/downloads/symbian-cpp`

2. Eseguire i semplici passi di installazione. É importante specificare che il drive di installazione sia lo stesso di quello dove è presente l'SDK precedentemente installata.

3. Per poter utilizzare le API di Qt su Symbian è necessario compilare il framework appena installato per questa architettura. Occorre quindi aprire il prompt dei comandi di Windows e all'interno della directory Qt digitare il comando seguente per eseguire il *configure* del framework:

   ```
   configure -platform win32-g++ -xplatform symbian-abld
   ```

4. Per compilare Qt per il dispositivo fisico eseguire il make tramite il seguente comando:

   ```
   make release-gcce
   ```

   Altrimenti per l'emulatore è necessario:

   ```
   make debug-winscw
   ```

Una volta compilato il framework, sarà possibile sviluppare applicazioni Qt per il target selezionato (device o emulatore).

QtMobility

Le QtMobility sono una serie di API che estendono le librerie Qt standard e danno la possibilità di accedere alle funzionalità più interne dei dispositivi mobili. Ad esempio attraverso le API di QtMobility è possibile mandare SMS, gestire i contatti e perfino controllare il ricevitore GPS.
Nel client sviluppato questa estensione si è rivelata necessaria per poter accedere in lettura e scrittura alla rubrica degli indirizzi del telefono. QtMobility fornisce infatti delle comode API per poter manipolare i contatti senza dover andare ad aprire e leggere i file che contengono le liste degli indirizzi memorizzati. Queste librerie sono costituite da molti moduli (Messaging, Multimedia, Location, ecc...) ma nel nostro caso abbiamo utilizzato solamente il modulo *Contacts*.
La procedura per poter utilizzare le API fornite dalle librerie QtMobility è simile a quella seguita per le Qt al paragrafo precedente. É necessario:

1. Scaricare il file

   `http://get.qt.nokia.com/qt/add-ons/`
   `qt-mobility-symbian-opensource-1.1.0.zip`

2. decomprimere il file

3. aprire il prompt dei comandi e spostarsi nella directory contenente i files appena estratti

4. lanciare il comando `configure` e attendere il completamento dell'operazione.

5. Per compilare le QtMobility per il dispositivo fisico eseguire il make tramite il seguente comando:

   ```
   make release-gcce
   ```

   Altrimenti per l'emulatore è necessario:

   ```
   make debug-winscw
   ```

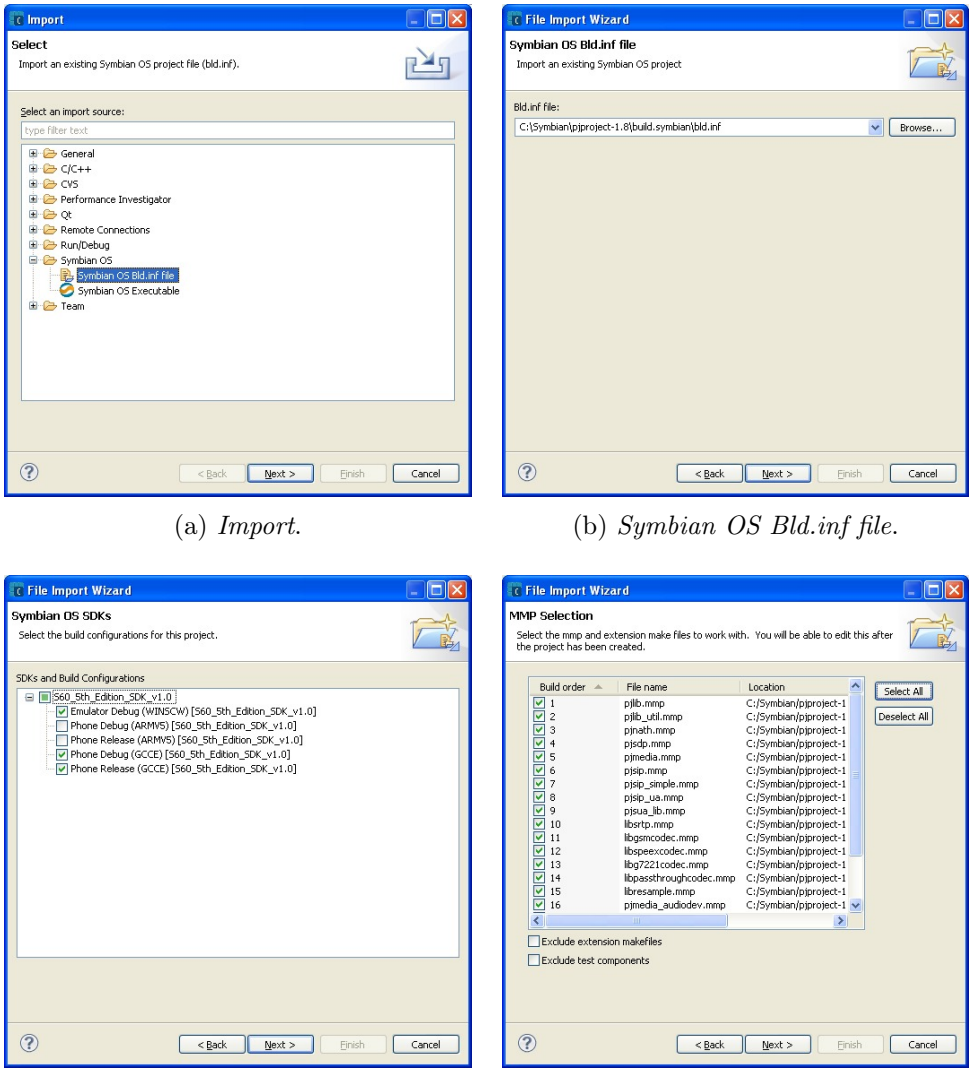
PJSIP

I sorgenti di PJSIP possono essere scaricati al seguente indirizzo:

`http://www.pjsip.org/download.htm`

Una volta decompresso il pacchetto, è possibile importarlo in Carbide.c++, attraverso i seguenti passaggi:

1. una volta eseguito Carbide.c++, selezionare *File* e poi *Import*;
2. all'interno della finestra di dialogo, selezionare dalla lista *Symbian OS*, e successivamente *Symbian OS Bld.inf file*; infine cliccare sul bottone *Next*;
3. cliccare quindi su *Browse* e scegliere il file *bld.inf* contenuto all'interno della cartella *build.symbian*;
4. a questo punto selezionare le configurazioni WINSCW (emulatore) e GCCE (telefono) e cliccare su *Next*;
5. all'interno della finestra di dialogo, cliccare su *Select All* ed infine su *Next*;
6. il progetto è quindi pronto per essere importato, premendo il bottone *Finish*. A questo punto Carbide esporterà tutti i file MMP all'interno del nuovo workspace. Attendere quindi che il processo sia terminato.



(a) *Import.* (b) *Symbian OS Bld.inf file.*

(c) *Symbian OS SDKs.* (d) *MMP Selection.*

Figura 4.2: Import del progetto PJSIP all'interno di Carbide.c++

Per poter effettuare il building del progetto è necessario selezionare all'interno del menu principale *Project*, *Build Configuration*, *Set Active* e quindi *Phone Debug (GCCE) [S60_5th_Edition_SDK_v1.0]*. É quindi possibile eseguire il building facendo click con il tasto destro sul progetto e poi click su *Build Project*. Finito questo processo (che può richiedere diversi minuti) si otterranno le librerie compilate per l'architettura Symbian e che potranno essere utilizzate per lo sviluppo della nostra applicazione VoIP. Ulteriori

informazioni sulla configurazione di PJSIP all'interno di Carbide.c++ sono presenti all'indirizzo:

`http://trac.pjsip.org/repos/wiki/Getting-Started/Symbian`

4.2.1 Creazione del pacchetto SIS

Il formato SIS è una particolare tipologia di file compresso usato sui telefoni cellulari Symbian per consentire l'installazione di applicazioni. Un file SIS può essere trasferito sul dispositivo mobile via OTA (over the air), BlueTooth ed infine tramite USB usando l'applicazione Nokia PC Suite. Alla fine della procedura di installazione, nella griglia delle applicazioni apparirà una nuova icona associata al software appena installato.
Attraverso Carbide è possibile compilare e creare il pacchetto di installazione seguendo questi passi:

1. lanciare Carbide.c++ ed aprire il progetto;
2. cliccare con il tasto destro del mouse sul progetto, ed infine su *Properties*;
3. selezionare *Carbide.c++*, ed poi cliccare su *Build Configurations*;
4. controllare che sia selezionato come configurazione attiva *Phone Debug (GCCE) [S60_5th_Edition_SDK_v1.0]*;
5. cliccare su *Add* e selezionare il file *.pkg* contenuto nella cartella del progetto;
6. all'interno della finestra *SIS Properties* cliccare su OK per utilizzare le configurazioni di default;
7. a questo punto cliccare con il tasto destro del mouse sul progetto, e poi su *Refresh*;
8. attendere il completamento del processo;
9. cliccare con il tasto destro sul progetto, ed infine su *Build Project* per compilare.

É possibile controllare lo stato della compilazione all'interno del tab *Console*. Se terminata con successo verrà stampato:

```
Signing

***SIS Creation Complete

Total Time: 12 sec
```

Eventuali errori di compilazione possono essere visualizzati all'interno del tab *Problems*.

TRK

Carbide permette anche di effettuare il debug direttamente sul device. Questa tecnica, denominata *On-Device Debugging*, risulta fondamentale nello sviluppo, poiché l'utilizzo dell'emulatore pone grossi limiti per quanto riguarda la gestione della rete e dell'audio.
Seguendo le istruzioni riportate nel paragrafo precedente, al termine di una compilazione si ottiene un programma auto-certificato. Di norma Symbian non accetta l'installazione di programmi di questo tipo, per cui risulta necessario effettuare una particolare configurazione sul telefono[3]:

1. sul telefono, aprire il menu;
2. entrare nel *Pannello di Controllo*, ed infine su *Gestione Applicazioni*;
3. selezionare *Opzioni*, e poi *Impostazioni*;
4. all'interno di *Installazione software*, selezionare *Completa*.

Dopo aver completato la configurazione, è possibile installare il software *TRK* sul telefono, attraverso le seguenti istruzioni:

1. collegare il telefono al computer utilizzando un cavetto USB;

[3]Le istruzioni si riferiscono ad un Nokia E52.

2. all'interno di Carbide, selezionare *Help* e successivamente *On-Device Connections*;

3. nella finestra di dialogo, selezionare USB come *Connection Type* e cliccare quindi su *Next*;

4. selezionare il tab *Install remote agents* e quindi selezionare l'installer appropriato a seconda dell'SDK installato[4]

5. cliccare su *Install* e seguire le istruzioni di installazione fornite da Nokia PC Suite.

Al termine dell'installazione, è possibile eseguire l'applicazione TRK[5]:

1. sul telefono, aprire il menu e selezionare *Applicazioni*;

2. all'interno della lista visualizzata, selezionare *TRK*;

3. apparirà una finestra di dialogo, all'interno della quale viene chiesto di collegarsi via Bluetooth. Selezionare *NO*, in quanto verrà utilizzato un collegamento USB;

4. selezionare *Opzioni* ed infine *Settings*;

5. cambiare il tipo di connessione da Bluetooth a USB e premere *Indietro*;

6. selezionare *Opzioni*, ed infine *Connect*;

7. a questo punto dovrebbe essere visibile lo stato *Connected* all'interno del programma.

A questo punto si ritorna a Carbide e si clicca su *Set Connection Settings* selezionando così la porta seriale collegata al device e la versione dell'installer utilizzata in precedenza. Per testare la connessione, selezionare il servizio TRK e cliccare su *Initiate service testing*. Se il servizio è configurato in modo corretto, cliccare su *Finish*. A questo punto all'interno del tab *Remote connections* è possibile verificare il corretto funzionamento del servizio TRK. É quindi possibile effettuare il debug *On-Device*.

[4]Per il telefono Nokia E52 selezionare *Application TRK 3.1.0*.
[5]Le istruzioni si riferiscono ad un Nokia E52.

4.3 Certificazione dell'applicazione

Con l'uscita della terza edizione del sistema operativo Symbian è stata introdotto il sistema *Symbian Signed* [12] i cui obbiettivi sono:

- evitare modifiche malevoli dei file SIS (tampering);
- assicurare che lo sviluppatore dell'applicazione possa essere indentificato;
- consentire la revoca di un certificato che precedentemente era stato assegnato all'applicazione.

Tali operazioni vengono eseguite tramite un modello di sicurezza basato su un'infrastruttura a chiave pubblica (TKI) e introducendo caratteristiche di sicurezza ai file SIS. Il sistema in particolare rilascia una coppia certificato pubblico/chiave privata che consente la segnatura dell'applicazione. Quest'ultima si ottiene dal valore hash di lunghezza fissa generato tramite il SIS e dalla chiave privata e, insieme al certificato pubblico, viene aggiunta al pacchetto.
Per verificare l'attendibilità dell'applicazione, il dispositivo, utilizzando la segnatura e il certificato pubblico, genera l'hash value corrispondente e lo confronta con il valore originale: se coincidono il SIS non è stato modificato (per esempio per scopi malevoli) e l'installazione può proseguire.
Lo sviluppatore una volta ottenuta la coppia certificato pubblico/chiave privata deve aggiungere la seguente linea nel file *pkg* generato durante la compilazione:

```
*"myApp.key", "myApp.cer"
```

prima della lista dei file da installare, ma dopo l'header.
A questo punto lanciando da terminale il comando

```
makesis myApp.pkg
```

verrà generato il SIS signed untrusted.
Per ottenere il certificato e la chiave per una applicazione segnata, ma non verificata, lo sviluppatore deve registrarsi al servizio *Open Signed Online*

all'indirizzo `www.symbiansigned.com` e specificare il codice IMEI[6] del dispositivo poichè questo servizio consente di installare il software solo nel terminale prescelto e solo per fini di sviluppo. La coppia certificato pubblico/chiave privata è utilizzabile per l'installazione del SIS per 36 mesi, dopo i quali bisognerà eventualmente rinnovare la richiesta. In ogni caso il software non potrà utilizzare le seguenti capabilities: CommDD, MultimediaDD, NetworkControl, DiskAdmin, DRM, AllFiles, TCB.
Se si vuole distrubuire l'applicazione e quindi consentirne l'installazione su tutti i dispositivi è necessario acquistare un Publisher ID attraverso il servizio *Offline*.
Poichè si voleva soltanto verificare il funzionamento del client VoIP su un dispositivo Symbian, ci si è limitati ad effettuare solo la procedura per ottenere un SIS signed untrusted.
É possibile ottenere tale pacchetto affidando a Carbide l'operazione di modifica del file pkg affinchè venga utilizzata la coppia certificato pubblico/chiave privata in possesso dello sviluppatore. Per impostare opportunamente l'ambiente di sviluppo, è sufficiente portarsi sull'opzione "SIS Builder" nella configurazione per la compilazione del progetto di sviluppo, premere *Add* e aggiungere i percorsi al file *pkg*, al certificato e alla chiave come mostrato nella figura seguente.

[6]Il codice IMEI (acronimo di International Mobile Equipment Identity) è un codice numerico che identifica univocamente un terminale mobile (Mobile Equipment), che può essere un telefonino o un modem. Nei cellulari Nokia si può ottenere digitando `*#06#` sulla tastiera

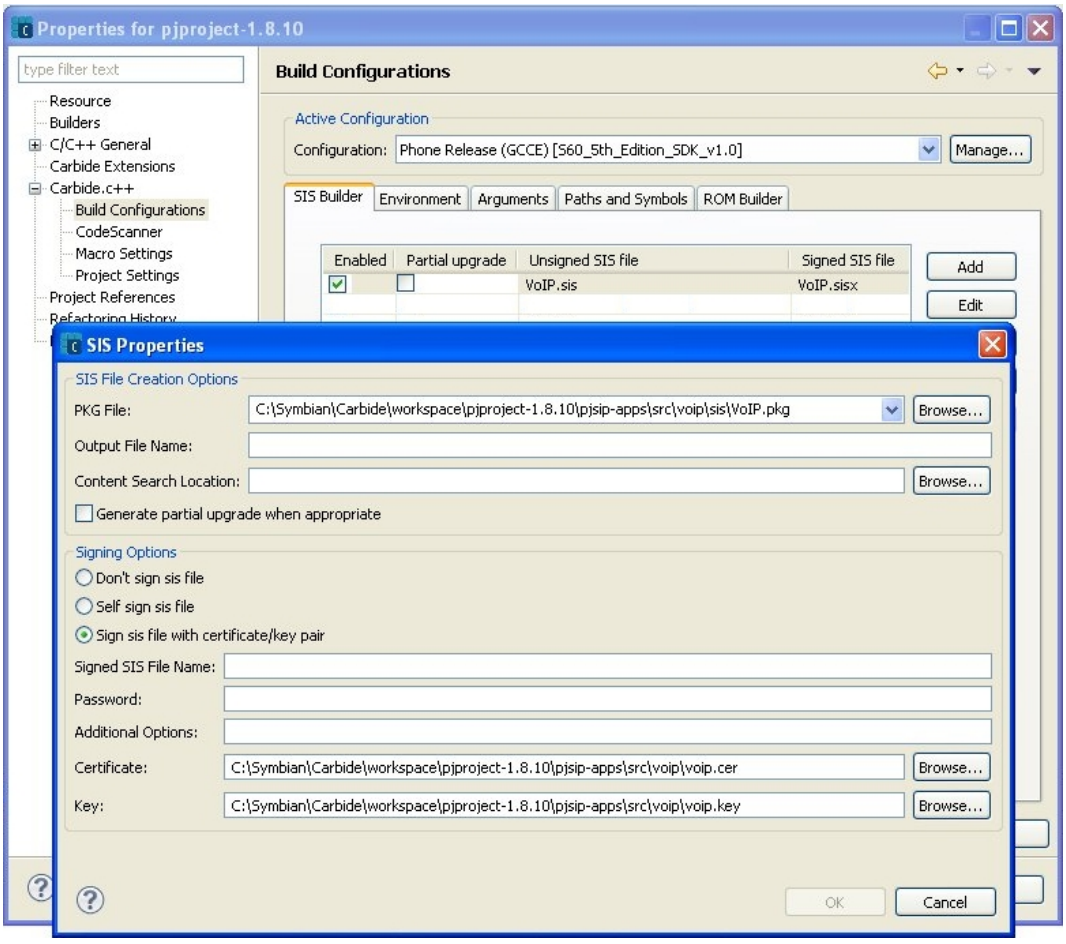


Figura 4.3: Build Configuration su Carbide.c++

4.4 Installazione

Dopo che la compilazione del programma, la creazione del pacchetto e la successiva certificazione sono andate a buon fine, non resta che installare il software nel dispositivo.
Prima di effettuare quest'ultima operazione occorre però installare due pacchetti di cui necessita il programma:

- `qt_installer.sis`: installa le librerie del framework.
- `qtmobility.sis`: installa le librerie che consentono di utilizzare funzionalità standard per la telefonia mobile.

Sono scaricabili entrambi dal sito della Nokia

`http://qt.nokia.com/products`

Una volta scaricati basta cliccare con il tasto destro del mouse sui pacchetti e cliccare su *'Installa con Nokia Application Installer'* e seguire la procedura di installazione direttamente sullo schermo del cellulare.
É importante che venga installato prima il pacchetto `Qt` e successivamente `QtMobility` per evitare problemi di dipendenze.
A questo punto è possibile installare il client VoIP come già fatto per i pacchetti precedenti. Finita la procedura sarà possibile lanciare il software cliccando su *Menu* poi su *Applicazioni* e infine sull'icona del programma.

4.5 Problematiche riscontrate

Le maggiori difficoltà incontrate si sono presentate durante la fase di testing e debugging del programma. Poichè non si aveva a disposizione da subito un dispositivo mobile, per queste due operazioni si è utilizzato inizialmente l'emulatore fornito con l'SDK di Symbian.
Nonostante sia uno strumento utile per lo sviluppo, l'emulatore ha comunque grossi limiti e differenze rispetto al dispositivo reale. Ad esempio, l'emulatore non mette a disposizione il quadro generale delle periferiche di cui il cellulare è provvisto, come la fotocamera e le schede di rete. É disponibile una sola interfaccia di rete, vista come una LAN, che non consente di sviluppare software in ambito multihoming. Poichè l'emulatore non rappresenta un device reale, alcune caratteristiche come la temporizzazione, le performance delle applicazioni e la gestione della memoria non sono valutabili tramite esso e questo può creare diversi problemi durante lo sviluppo.
Per questi motivi l'emulatore è stato impiegato solamente in una fase preliminare. In particolare utilizzando il dispositivo mobile non sono state riscontrate delle anomalie nel comportamento delle finestre di dialogo e di altri elementi grafici che nell'emulatore erano presenti.
Perciò si può concludere che non è consigliabile sviluppare software utilizzando come strumento di testing solo l'emulatore, poichè potrebbe presentare *bugs* non presenti effettivamente nel device reale.

Conclusioni e sviluppi futuri

La diminuzione dei costi delle infrastrutture delle reti mobili e di Internet ha consentito di diminuire anche i costi per il loro utilizzo da parte degli utenti finali. Col passare del tempo inoltre la rete si è evoluta ed è diventa sempre più un mezzo di comunicazione vasto e variegato e una fonte di condivisione di dati e di risorse di ogni tipo.
Il VoIP ha beneficiato a tal punto di tale progresso che oggi la comunicazione vocale in tempo reale tramite personal computer dotati di accesso a banda larga ad Internet è assai diffusa e ha costi ridotti all'abbonamento flat versato al fornitore dell'accesso alla rete.
L'ulteriore evoluzione in tale campo si ritiene possa essere quella di portare tale tecnologia anche sui dispositivi mobili che hanno un bacino d'utenza vasto. Ciò potrebbe significare da un lato una grande fonte di guadagno per i fornitori di accessi alla rete mobile, dall'altro un mezzo di comunicazione a basso costo, il servizio voce su Internet, alternativo a quello tradizionale.
Il lavoro svolto è consistito nella realizzazione di un'applicazione per dispositivi mobili che fornisse in maniera semplice e utilizzando il VoIP, le principali operazioni previste da un normale dispositivo mobile. Dopo aver analizzato quali fossero le tecnologie open-source disponibili in rete, si è dato risalto ai seguenti fattori:

- diffusione;
- portabilità;
- performance;

per intraprendere le seguenti scelte:

- target: dispositivi mobili che utilizzano Symbian OS, con Nokia, una delle aziende leader del settore, la principale esponente;
- GUI: Qt framework, essendo native in una grande varietà di dispositivi mobili e scritte in C++;
- backend per i servizi VoIP: librerie PJSIP, altamente portabili e con alte performance.

Successivamente si è passati ad analizzare quali dovessero essere i requisiti dell'applicazione ed è stato scelto l'ambiente di sviluppo su cui lavorare.
Si può affermare che l'applicazione risultante soddisfa sia tutti i requisiti funzionali che un client VoIP dovrebbe avere, sia tutti i requisiti non funzionali che ci si era prefissati di implementare e che consentono all'utente di fruire di uno strumento sufficientemente esteso e facile da utilizzare. In particolare è possibile effettuare, ricevere e visualizzare il registro aggiornato delle chiamate, è possibile gestire la rubrica del dispositivo tramite l'applicazione e instaurare delle conversazioni testuali con altri utenti.
In futuro sarebbe comunque utile implementare ulteriori funzionalità quali la gestione di conferenze vocali, per consentire a più persone di comunicare nello stesso momento, e la possibilità di mettere in attesa una chiamata passando agevolmente ad un'altra. In tal modo si ritiene che il client si potrà considerare completo se confrontato con quelli proprietari. Inoltre poichè i dispositivi mobili per loro natura non sono sempre agganciati ad una cella, ma si possono trovare in condizioni di "segnale assente", sarebbe interessante integrare la tecnica del multihoming, utilizzando ad esempio ABPS, per rendere l'applicazione funzionante in ogni contesto.

Riferimenti

[1] The World in 2010: ICT Facts and Figures. http://www.itu.int/ITU-D/ict/material/FactsFigures2010.pdf.

[2] STUN. http://it.wikipedia.org/wiki/STUN.

[3] Giorgia Lodi Vittorio Ghini and Fabio Panzieri. Always Best Packet Switching: the Mobile VoIP Case Study. *Journal of Communications*, 4(9), october 2009.

[4] 3GPP. Voice Call Continuity between Circuit Switched and IP Multimedia Subsystem. http://www.3gpp.org/ftp/Specs/html-info/23206.htm.

[5] Gustafsson E. and Jonsson A. Always Best Connected. *Wireless Communications, IEEE*, 10(1):49–55, february 2003.

[6] ITU-T Recommendation G.1010. End-user multimedia QoS categories, november 2001.

[7] Forum Nokia Library. Introduction to Symbianˆ3. http://library.forum.nokia.com/index.jsp.

[8] Benny Prijiono. PJSIP Developer's Guide, 2006. http://www.pjsip.org/release/0.5.4/PJSIP-Dev-Guide.pdf.

[9] Jasmin Blanchette and Mark Summerfield. *C++ GUI Programming with Qt 4 - The official C++/Qt book*. Prentice Hall, second edition, 2006. http://qt.nokia.com/developer/books/cpp-gui-programming-with-qt-4-2nd-edition/.

[10] Qmake Project Files. http://doc.qt.nokia.com/latest/qmake-project-files.html.

[11] QDialog Class Reference. http://doc.qt.nokia.com/latest/qdialog.html#details.

[12] Getting to Grips with Symbian Signed, september 2005. https://www.symbiansigned.com/developerguidetoSymbianSigned.pdf.