Alma Mater Studiorum · Università di Bologna

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

SVILUPPARE APPLICAZIONI VoIP SU SYMBIAN: UNA CORSA AD OSTACOLI

Tesi di Laurea Specialistica in Reti di Calcolatori

Presentata da:
Ricciardelli Marco

Relatore:
Dott. Ghini Vittorio

III Sessione
Anno Accademico 2009/2010

Parole Chiave: *mobilità, reti di calcolatori, symbian, qt, qml*

Indice

Elenco delle tabelle

Elenco delle figure

Introduzione

Questa tesi di laurea specialistica tratterà lo sviluppo di applicazioni VoIP (Voice Over IP) per il sistema operativo Symbian sviluppato e mantenuto da Nokia. Si effettuerà una panoramica generale sulle tecnologie attualmente disponibili per la comunità degli sviluppatori.

Si introdurrà la tecnologia di comunicazione VoIP, che molto probabilmente soppianterà nel futuro prossimo l'attuale telefonia tradizionale analogica.

Successivamente tratteremo il protocollo opensource SIP, utilizzato per stabilire questo tipo di connessioni multimediali.

Dopo l'introduzione di queste basi ci soffermeremo sugli obiettivi da raggiungere, per lo sviluppo di un'applicazione che rispetti tutti i canoni di una tipica applicazione volta al mondo del mobile. Tra gli obiettivi principali c'è sicuramente quello di essere il meno vincolati possibile dall'architettura e dal sistema operativo, sebbene si stia parlando di sviluppo in ambiente specifico Symbian. Si utilizzeranno le ultime tencologie messe a disposizione da Qt come QtQuick e QtMobility.

La necessità di sviluppare applicazioni per un ambiente mobile, ci ha portato a cercare framework di sviluppo già compatibile con le piattaforme su cui saremmo andati ad installare la nostra applicazione. Per la scelta del protocollo SIP, ci si è appoggiati alla ormai famosa libreria opensource PjSip, le cui caratteristiche peculiari sono l'efficienza e la portabilità. La libreria in questione avrebbe permesso già da sè di sviluppare applicazioni molto performanti a livello console, ma un'interfaccia grafica chiara e funzionale al giorno d'oggi è indispensabile. Questa nuova esigenza ha portato all'adozione di Qt framework come ambiente di sviluppo, sia perchè mantenuto direttamente da Nokia e quindi per Symbian, sia perchè offre la possibilità di portare l'interfaccia in maniera automatica su altre piattaforme.

Nel sesto capitolo tratteremo l'applicazione SmartQtVoip. Si partirà dall'architettura di base andando via via ad analizzare ogni componente fondamentale dell'applicazione.

Infine faremo alcune considerazioni sulle difficoltà incontrate per lo sviluppo di applicazioni per ambienti Symbian, arrivando poi a fare un'analisi sulle funzionalità su cui Nokia dovrebbe concentrare maggiormente i propri sforzi, favorendo per primo chi fa già parte di queste comunità di sviluppo, aumentando la capacità produttiva, e consentendo anche allo sviluppatore alle prime armi di poter creare facilmente le sue prime applicazioni.

Capitolo 1

VoIP (Voice over IP)

Voice over Internet Protocol o semplicemente Voice over IP (VoIP) è una famiglia di metodologie, protocolli di comunicazione e tecnologie di trasmissione per la gestione di comunicazioni vocali e sessioni multimediali attraverso reti basate sul protocollo IP (Internet Protocol). Altri termini, frequentemente utilizzati e spesso usati come sinonimi di VoIP sono IP telephony, Internet telephony, voice over broadband (VoBB), broadband telephony and broadband phone. Il termine "Internet telephony", o telefonia internet, si riferisce dunque ai servizi di comunicazione quali voce, fax, SMS e messaggi vocali trasportati attraverso la rete internet (possono anche usare come mezzo trasmissivo una qualsiasi rete privata basata sul protocollo IP, per esempio una LAN all'interno di un edificio o di un gruppo di edifici) invece che attraverso le normali reti telefoniche (PSTN). Una chiamata VoIP si sintetizza nell'apertura di un canale di comunicazione tra i due dispositivi, la digitaliz-

zazione della voce, ovvero un segnale audio analogico e la trasmissione sotto forma di pacchetti attraverso la rete. Il vantaggio principale di questa tecnologia sta nel fatto che essa elimina l'obbligo di riservare della banda per ogni telefonata (commutazione a circuito), sfruttando l'allocazione dinamica delle risorse, caratteristica dei protocolli IP (commutazione a pacchetto). Ne deriva infatti un minore costo delle infrastrutture (basta una semplice rete che supporti IP), un minor costo per le telefonate specialmente quelle a lunga distanza (internazionali) e l'aggiunta di nuove funzionalità mantenendo invariato l'hardware. La gestione delle chiamate voce sulla rete IP è, al momento, indirizzata verso due differenti proposte, elaborate in ambito ITU e IETF, che sono rispettivamente H.323 e SIP (Session Initiation Protocol). Non parleremo dello standard H.323 poichè è un protocollo relativamente datato ed è in corso di sostituzione a favore del protocollo SIP (Session Initiation Protocol), detto questo si può ben immaginare che le apparecchiature VoIP di ultima generazione seguiranno nella quasi totalià lo standard SIP.

1.1 Digitalizzazione del flusso audio

Per la digitalizzazione del flusso audio viene utilizzato un codec (selezionabile), ovvero un algoritmo che si occupa di codificare e/o decodificare (digitalmente) un segnale, effettuando eventualmente una compressione (e/o decompressione in lettura) dei dati, in modo da poter ridurre lo spazio di memorizzazione occupato, a vantaggio della portabilità o della trasmissività del

flusso codificato. Per valutare effettivamente la qualità della chiamata esistono svariati metodi ma in questo caso, per semplicità ci baseremo sul PESQ acronimo di "Perceptual Evaluation of Speech Quality", tradotto letteralmente con "valutazione percettiva della qualità della discussione", che si basa sul giudizio della qualità della telefonata dato da colui che la effettua. Viene utilizzato dagli operatori telefonici VoIP per testare la qualità del loro servizio. Per le telefonate da internet il problema della qualità è decisamente più sentito rispetto alla telefonia tradizionale; chi offre servizi VoIP infatti, deve garantire la miglior qualità possibile della telefonata: per farlo le compagnie telefoniche devono aver la certezza che vi sia una certa quantità di banda sempre disponibile, in modo tale da evitare il verificarsi di latenze, ritardi o disturbi nella conversazione che indiscutibilmente ne peggiorano la qualità. Vi sono strumenti oggettivi che forniscono l'indice PESQ (sottoforma di un valore compreso tra 0 e 5) di una conversazione telefonica, dando la possibilità ai gestori di modulare al meglio la qualità del servizio offerto. Vediamo nell'immagine sottostante la qualità percepita (PESQ) dei maggiori codec audio utilizzati nelle comunicazioni VoIP. Come si può ben vedere, codec come Speex (codec opensource nato per la compressione prettamente vocale) raggiungono una qualità vocale molto alta (circa 4.5) con poco meno di 15 kbps.

Capitolo 2

SIP (Session Initiation Protocol)

Il protocollo SIP (Session Initiation Protocol) è un protocollo di controllo a livello applicazione per creare, modificare e terminare sessioni multimediali tra due o più partecipanti. È stato accettato come standard dalla IETF nel marzo del 1999 quando è stata rilasciata la prima stesura delle specifiche (RFC 2543)[Bib03]. La seconda, nonché attuale, stesura risale a maggio 2002 (rfc 3261). Il protocollo è tuttora in via di sviluppo e vengono frequentemente pubblicati aggiornamenti delle precedenti Request for Comments. Le applicazioni più comuni di tale protocollo sono la telefonia su IP, lo streaming audio-video, le conferenze e la messaggeria istantanea. Grazie alla completezza del protocollo SIP e dello stack con cui di norma viene implementato, è possibile non solo inviare richieste di connessione ad uno o più utenti, ma

all'interno di queste richieste è possibile incapsulare le descrizioni dei tipi di media utilizzabili e dei protocolli di trasporto supportati, nonché risulta possibile stabilire una tabella di routing per gli attraversamenti ottimizzati della rete. Sono inoltre presenti tutti i comuni mezzi per la locazione degli utenti all'interno della rete, per l'autenticazione e la registrazione. Esso risulta totalmente indipendente dai protocolli di trasporto sottostanti. È strutturato come un protocollo a strati, consentendo una descrizione dei comportamenti dei singoli strati in termini di un insieme di processi semplici ed indipendenti, limitando lo studio sull'interazione tra livelli adiacenti. Non è necessario che gli elementi specificati dal protocollo contengano tutti gli strati definiti. Inoltre spesso tali strati sono soltanto logici e non fisici.

Transaction User (TU)
Transaction
Transport
Syntax/Encoding

Tabella 2.1: Stack SIP

Il livello più basso dello stack si occupa della sintassi e della codifica delle informazioni che seguono il sistema di codifica di Bakus-Naur (BNF). Il secondo strato, contenuto in tutti i componenti dell'architettura SIP, è quello di trasporto, che definisce come vengono inviate le richieste e ricevute le risposte all'interno della rete. Il terzo strato è quello di transaction, che si occupa

della supervisione delle transazioni di informazione tra client e server, tenendo traccia di richieste, risposte e timeout, consentendo il collegamento tra richieste e risposte e la ritrasmissione dei messaggi per i quali si è raggiunto il tempo di timeout. Lo strato di transaction si trova negli User Agent Client (UAc) e nei proxy stateful. Sullo strato di transazione si stabilisce un quarto strato detto TU (Transaction User). Ogni entità SIP, ad esclusione del proxy stateless, è un'entità di tipo TU. Quando viene effettuata una richiesta, il TU crea un'istanza di transazione a cui associa un indirizzo ip di destinazione, una porta ed un protocollo di trasporto. Il TU stesso può cancellare l'istanza creata bloccando tutto il processo e generando una risposta d'errore specifica per quella transazione. SIP è un protocollo orientato al messaggio ed utilizza messaggi di testo codificati in UTF-8[Bib05], è orientato al Web, e la sua struttura, simile ad HTTP/1.1 consente l'utilizzo secondo paradigmi "Client-Server", nonché connessioni Punto-Punto. Tra i principali punti di forza del protocollo SIP, che lo rendono di grande interesse, sia scientifico, sia economico, sono la scalabilità intrinseca dell'architettura su cui si basa e l'interoperabilità con sistemi preesistenti e sistemi "SIP-based". La scalabilità del protocollo deriva dal suo sistema di instaurazione delle sessioni di comunicazione, infatti la presenza di un server che gestisce le connessioni tra gli utenti è necessaria solo durante le fasi di instaurazione, modifica e rilascio della connessione. Durante la conversazione il flusso di dati avviene direttamente tra gli elementi terminali, non aumentando il carico della rete e permettendo al server di gestire numerose connessioni. L'interoperabilità

invece scaturisce dalla struttura modulare del protocollo, che consiste di un limitato set di funzionalità che lo rendono facilmente con sistemi SIP-based, su cui è possibile basare delle estensioni che ne consentono utilizzi più complessi, nonché il dialogo con sistemi preesistenti quali la fonia tradizionale o protocolli VoIP compatibili. Inoltre il dialogo anche tra sistemi diversi è facilitato dai sistemi di negoziazione di protocollo e dall'interrogazione di servizi disponibili. La descrizione del protocollo contenuta nella rfc 3261 è da considerare un insieme di direttive non imperative, mentre l'implementazione delle funzionalità è lasciata libera. Grazie all'assenza di limitazioni pertanto, nel rispetto dello standard "de facto", la progettazione e lo sviluppo risultano facilitati permettendo una più veloce diffusione del protocollo.

2.1 Messaggi SIP

I messaggi si dividono principalmente in richieste (Request) da client a server e risposte (Response) da server a client. Sia le richieste, sia le risposte scambiate dal protocollo seguono il formato standard (definito nella rfc 2822) anche se integrano sintassi e set di caratteri anche non previsti in questo standard. Entrambe le tipologie di messaggio sono costituite da una start-line, uno o più campi di intestazione (header), una linea vuota usata come separatore tra le intestazioni ed il corpo vero e proprio del messaggio (body), che è da considerarsi opzionale.

<generic-message> := <start-line>

```
*<message-header>
<CRLF>
[ <message-body> ]
```

```
<start-line> := <Request-Line> / <Status-Line>
```

2.1.1 Request

I messaggi Request hanno come start-line una request-line costruita come segue:

```
<Request-Line> :=        <Method> <SP> <Request-URI>
                         <SP> <SIP-Version> <CRLF>
```

contenente il metodo di richiesta, l'URI a cui è indirizzata la richiesta e la versione del protocollo utilizzata. Secondo le specifiche di SIPv2 sono previsti sei metodi: REGISTER, INVITE, ACK, CANCEL, BYE, OPTIONS, tuttavia è possibile estendere il protocollo con l'aggiunta di ulteriori metodi di richiesta. Il campo Request-URI è un indirizzo o un insieme di indirizzi URI che seguono gli schemi generali dell'rfc 2396 (Uniform Resource Identifiers acronimo appunto di URI).

2.1.2 Response

I messaggi Response hanno come start-line una status-line costruita come segue:

<Status−Line>:= <SIP−Version> <SP>
<Status−Code> <SP>
<Reason−Phrase> <CRLF>

costituita dalla versione del protocollo usata, un numero intero di tre cifre (status-code) ed una frase opzionale a commento della risposta. Lo status-code indica il tipo di risposta contenuta nel messaggio. In base al codice le tipologie di risposta si possono distinguere in sei gruppi:

1xx: Provisional: risposte provvisorie necessarie ad interrompere i timer di ritrasmissione;

2xx: Success: risposte che confermano il successo dell'operazione richiesta;

3xx: Redirection: richieste di redirezione della richiesta;

4xx: Client Error: avvisi di errore da parte del client nella sintassi nella richiesta;

5xx: Server Error: avvisi d'errore da parte del server che non può effettuare l'operazione richiesta;

6xx: Global Failure: fallimenti critici dovuti a motivazioni diverse dalle precedenti.

2.1.3 Campi header

L'header SIP è simile a quelli dell'HTTP (in maniera conforme all'rfc 2234) ed assume la forma:

```
<header > :=      "header-name"
                  <HCOLON>
                  <header-value>
                  *(<COMMA> <header-value>)
```

in cui i campi sono separati da virgole nell'ordine "nome-campo: valore" seguiti da una lista eventualmente vuota di parametri.

2.1.4 Corpo del messaggio

Il corpo del messaggio (Body) dipende strettamente dal tipo di richiesta o risposta. La lunghezza e la tipologia dei contenuti devono essere definite nell'intestazione del messaggio. Può contenere anche dati binari secondo gli standard MIME (rfc 2045 ed rfc 2046). Quando non diversamente specificato nell'intestazione del messaggio, il set di caratteri è UTF-8.

2.2 Componenti

Il protocollo SIP definisce le entità coinvolte durante l'instaurazione di una sessione di comunicazione.

2.2.1 User Agent (UA)

Lo User Agent (UA) è l'entità che interagisce con l'utente. Di norma ha un interfaccia amichevole orientata all'instaurazione delle sessioni di comunicazione tra utenti. Lo UA rende trasparente all'utente l'utilizzo dello stack e si occupa di determinare il tipo di media da utilizzare. SIP (rfc 3261) definisce gli UA come periferiche telefoniche che sono date dalla combinazione di uno UAc (User Agent Client) e un UAs (User Agent Server). L'UAc è l'unica entità all'interno di un'architettura basata su SIP in grado di generare richieste di connessione. L'UAs invece è in grado di ricevere richieste ed inviare risposte. Nella norma si intende per UA un'entità unica che svolge entrambe le funzioni sopraelencate. Uno UA SIP può essere implementato sia fisicamente come IP Phone, sia in una soluzione software in esecuzione su un computer. È possibile mettere in comunicazione due User Agent in maniera diretta senza l'ausilio di nessun software di interfaccia, ovvero può esserci interposta una grossa rete a complessità elevata. Uno UA rappresenta un sistema terminale. Contiene un UAc (client) che genera le request ed un UAs (server) che genera le risposte.

2.2.1.1 Comportamento dello User Agent Client

Una richiesta ben formata da uno UAc deve contenere almeno i seguenti campi nell'header: To, From, CSeq, Call-ID, Max-Forwards, e Via, a cui si aggiungono altri tre campi obbligatori che sono: metodo, Request-URI e SIP version. Inizialmente il valore di Request-URI corrisponde con l'indirizzo URI

del campo “To” ad eccezione del metodo REGISTER che contiene nel campo Request-URI l’indirizzo del dominio di registrazione. In alcune circostanze particolari, stabilire preventivamente un percorso di routing influisce sulla Request-URI. Il campo To è l’indirizzo, o la lista di indirizzi, target della request, e può contenere un SIP o SIPS URI, ovvero anche altri schemi di URI (es. indirizzi telefonici) a seconda dei casi. Può essere popolato in svariati modi o manualmente, o da una rubrica di indirizzi, ovvero con una etichetta rappresentante un URI. Il campo From indica l’indirizzo dell’origine della request. Contiene un indirizzo URI ed opzionalmente un’etichetta. Il campo Call-ID identifica univocamente una serie di messaggi, infatti questo identificativo rimane invariato per tutti i messaggi inviati in una sessione da uno UA compresa la registrazione. È necessario fare in modo che questo identificatore non possa essere duplicato accidentalmente da altri UA. Per generare questi identificatori si usano dei sistemi di cifratura randomici, affinché si possa evitare la collisione di Call-ID tra UA diversi. Il campo Cseq è un identificativo della transaction. È composto da un numero progressivo e da un metodo (es.Cseq: 4356 INVITE). Il campo Max-Forwards indica il numero massimo di hop che si possono effettuare durante la trasmissione del messaggio. Ad ogni hop il contatore viene decrementato. Di norma si sceglie il valore massimo 70 affinchè i messaggi riescano ad arrivare a destinazione anche in reti di dimensioni considerevoli in assenza di cicli infiniti, e nel caso di presenza di cicli non si prendano troppe risorse ai proxy. Utilizzare un numero più basso può compromettere la QoS della rete: è pertanto scon-

sigliato a meno di una buona conoscenza della rete che permette di prevedere il numero di hop massimi. Il campo VIA contiene gli elementi attraversati durante una transaction ed il metodo di trasporto utilizzato. Per spedire una request bisogna innanzitutto determinare la destinazione. Nel caso in cui il primo target sia uno strict router bisogna inoltrare la request al Request-URI. Nel caso contrario l'inoltro avviene al primo elemento del route. Nel caso in cui il route non sia presente o sia vuoto, si utilizza come target il Request-URI.

2.2.1.2 Comportamento dello User Agent Server

Le richieste in arrivo allo UAs vengono prima processate dal transport layer e quindi passate al transaction layer il quale, appena svolte le proprie azioni, passa la response al TU. Se in una request sono presenti più di un campo VIA il messaggio viene scartato poiché è evidente che ci sono stati degli errori di trasmissione. Nel caso di response di tipo 3xx ovvero di redirect, il client dovrebbe utilizzare gli indirizzi URI contenuti nel contact header della response per le prossime request allo stesso target. Ogni nuova request modificata viene trattata come una nuova request in tutto e per tutto. Tuttavia è buona norma utilizzare gli stessi valori di Call-ID, To e From della richiesta precedente, aggiornando il valore di Cseq con un valore maggiore. Se tra i campi dell'header vengono riscontrati campi sconosciuti o non supportati, lo UAs deve limitarsi ad ignorarli e continuare a processare il messaggio. Nella formulazione delle risposte bisogna utilizzare gli stessi

valori dei campi Call-ID, From, Cseq e Via (compreso l'ordine) dell'header della request. Se è presente anche il campo To, anche questo deve essere ricopiato. Uno UAs stateless è uno UA che non mantiene lo stato di transaction. Risponde normalmente alle request non memorizzando tuttavia lo stato d'invio della response, ma scarta tutte le informazioni. Pertanto su una richiesta di ritrasmissione di un messaggio ne crea uno nuovo e lo inoltra come se fosse un nuovo messaggio. Uno UAs non utilizza il transaction layer, ma riceve le richieste direttamente dal transport layer ed invia direttamente le risposte dal transport layer. Tra i comportamenti particolari di un UAs stateless:

- non vengono spedite response 1xx provisional;
- non vengoono ritrasmesse le responses;
- vengono ignorate le ACK requests;
- vengono ignorate le CANCEL requests.

La request CANCEL viene utilizzata per annullare una richiesta precedentemente inviata da un client. Questo tipo di richiesta non ha effetto su richieste già processate dallo UAs. Le CANCEL request possono essere create sia dai proxy che dai UAc. Un UAc non può costruire request CANCEL per annullare altro che INVITE request. Se l'INVITE request contiene un percorso di Route, anche la CANCEL request avrà lo stesso route, affinché anche i proxy stateless possano inoltrare correttamente la cancel-

lazione. Non devono invece essere contenuti nella CANCEL request i campi require e proxy-require. La request CANCEL può essere inviata soltanto se si è già ricevuta una risposta provisional. Dal punto di vista del server il metodo CANCEL influisce su una transaction pendente. Infatti la transaction corrispondente viene cancellata. Uno UAs si limita a rispondere ad una richiesta di cancellazione con un messaggio di risposta 200 OK.

2.2.1.3 Gestione degli errori

In base ai messaggi d'errore ritornati allo UA i comportamenti possono essere differenti. Ecco alcuni tipi di risposte d'errore:

401 o 407 unauthorized o proxy authentication required: eseguire la procedura di autenticazione poiché il server Proxy richiede un riconoscimento esplicito;

413 request entity too large: rimandare la request con una dimensione minore rispetto alla dimensione massima supportata dallo UAs;

415 unsupported media type: il tipo di media utilizzato non è supportato dall'UAs pertanto è necessario rinviare la richiesta con un media supportato. Nel messaggio d'errore è previsto un campo Accept nell'header che individua i media supportati;

416 unsupported URI scheme: è necessario inviare nuovamente la request con un indirizzo URI valido.

Per l'elenco completo dei metodi di processamento delle risposte si può prendere come riferimento l'RFC 3261.

2.2.2 Locator server

Il locator server viene richiamato dal redirect server o dal proxy server per ottenere informazioni circa le possibili locazioni dell'utente chiamato. Non è specificato il funzionamento del servizio nella rfc 3261. Gli unici requisiti sono:

1. La possibilità di accesso in lettura e scrittura da parte di un server Registrar;

2. La formattazione dei dati in modo da essere letti dai Proxy e dai Redirect server.

Il locator server non utilizza protocollo SIP per comunicare con il registrar ed il proxy: alcuni di essi utilizzano diverse tecnologie quali LDAP .

2.2.3 Proxy server

Un Proxy Server, chiamato anche semplicemente proxy, è un programma intermediario che agisce sia da server che da client. Le richieste vengono gestite internamente oppure trasferite su altri server. Un proxy interpreta e se necessario riscrive le richieste prima di inoltrarle. Un proxy server a seconda della configurazione può anche funzionare in modalità di biforcazione (fork),

ovvero può inoltrare una richiesta di connessione a due o più entità in parallelo consentendo un rintracciamento più veloce degli utenti. I proxy server possono essere raggruppati in tre categorie: call-stateful, stateful, stateless. I proxy call-stateful memorizzano tutte le informazioni sugli stati da quando la sessione viene stabilita, finché questa non viene terminata. Un proxy stateful invece memorizza tutti gli stati dall'inizio alla fine di una transaction. Un proxy stateless non memorizza nessuna informazione sullo stato. Viene ricevuta ed inoltrata una richiesta al nodo successivo ed immediatamente vengono cancellate tutte le informazioni riguardanti quella richiesta. I proxy SIP sono degli elementi all'interno di una rete che instradano SIP request verso UAs e SIP response verso UAc. Le richieste possono attraversare numerosi proxy prima di giungere allo UAs ognuno dei quali può decidere delle regole di instradamento prima di inoltrare la richiesta, mentre le risposte ripercorrono il percorso in senso inverso rispetto alla richiesta. Il proxy SIP in realtà è un componente logico, infatti esso può comportarsi da proxy ed inoltrare le richieste ovvero processarle. Per esempio, supponendo di avere una richiesta malformata, il proxy può comportarsi da UAs e rispondere direttamente al mittente con un'informazione sull'errore. Un proxy per ogni richiesta può decidere se utilizzare la modalità stateless o stateful. Nel caso stateless opera semplicemente come elemento di redirezionamento, inoltrando ogni richiesta e cancellando le informazioni riguardanti tutti i pacchetti inoltrati. Un proxy stateful invece conserva le informazioni di tutti i messaggi inoltrati, in modo da poter riutilizzare i dati di instradamento per eventuali

messaggi successivi, ovvero decidere di biforcare un messaggio inoltrandolo a due o più destinatari. In qualsiasi momento è possibile passare alla modalità stateless. In modalità stateful opera secondo transazioni di tipo client-server. Un server stateful ha all'interno un core proxy detto "higher layer proxy" che si comporta da server per rispondere alle richieste di uno o più client. Il core proxy stabilisce il percorso di indirizzamento scegliendo passi diretti o con più salti. Il server proxy attiva una nuova server transaction per ogni richiesta ricevuta. Questo tipo di entità può comportarsi come un UAs creando nuovi messaggi (es. 100 Trying), tuttavia non può farlo senza aver ricevuto una richiesta di invito. Azioni principali del proxy sono:

- validazione della richiesta;
- preprocessamento delle informazioni di reindirizzamento;
- determinazione dei target per la richiesta;
- inoltro della richiesta ai target;
- processamento delle risposte.

2.2.3.1 Validità della richiesta

Per validare una richiesta, il messaggio deve superare i seguenti test:

- test di sintassi;
- correttezza degli URI;

- numero di salti (max-forwards);
- controllo di loop (opzionale);
- superamento dei requisiti richiesti dal proxy;
- autorizzazione sul proxy.

Nel caso in cui uno dei precedenti test non venga superato, il proxy risponde alla richiesta con un messaggio d'errore appropriato. Il controllo di sintassi deve essere estendibile e non devono essere scartati messaggi che contengano nell'header campi sconosciuti o metodi non previsti, ma solo messaggi formalmente errati. Se la struttura dell'indirizzo URI nella request non è riconosciuta dal proxy, questo rifiuta la richiesta mandando un messaggio di errore (416 Schema URI non supportato). Tra i campi dell'header è presente l'indicazione sul numero massimo di salti permessi. Se questo numero viene superato, nel caso in cui il messaggio è una richiesta di OPTION il proxy può rispondere direttamente comportandosi da UAs; se il messaggio invece deve ulteriormente essere inoltrato, il proxy risponde con un messaggio d'errore (483 too many hops). È possibile, ma non necessario, che un proxy faccia un controllo di loop, ovvero controlli se il suo next hop sia contenuto nel campo VIA dell'header del messaggio. In questo caso, se non ci sono altri percorsi possibili, viene mandata una risposta 482 (loop detect). Future estensioni del protocollo possono introdurre nuovi metodi con requisiti particolari, pertanto è prevista la possibilità di un controllo dei requisiti. Nel caso in cui questo test non venga passato, il messaggio d'errore che viene rimandato indietro è

420 (Bad extension). Alcuni proxy possono avere dei criteri di autenticazione per l'inoltro dei messaggi: in questo caso se il messaggio non supera il test di autenticazione viene scartato.

2.2.3.2 Preprocessamento delle informazioni di reindirizzamento

Quando un proxy riceve una richiesta di inoltro deve andare a controllare nelle tabelle di routing l'URI per il reindirizzamento. Nel caso in cui il messaggio contenga nell'header nel campo Record-Route il Request-URI, il proxy si preoccupa di sostituire questo campo con quello della sua tabella di routing e processa il messaggio come se fosse arrivato con questo nuovo header modificato. Questo succede soltanto quando l'elemento che ha fatto la richiesta è uno strict router. Se il messaggio contiene nella request-URI parametri di tipo "maddr", il proxy deve controllare se nelle sue tabelle di routing ha la possibilità di processare l'inoltro per quell'indirizzo. In caso positivo il proxy inoltra il messaggio all'indirizzo segnato e cancella il parametro maddr. È possibile che un messaggio arrivi con un maddr che indichi proprio l'indirizzo del proxy, tuttavia con una porta diversa. In questo caso il proxy deve inoltrare il messaggio a se stesso sulla porta opportuna. Se nel campo route è presente l'indirizzo del proxy, il proxy deve preoccuparsi di cancellare il proprio indirizzo prima di inoltrare il messaggio.

2.2.3.3 Determinazione dei target

Successivamente al preprocessamento, il proxy deve stabilire il successivo o i successivi hop. Per determinare questi target, deve controllare tra gli inoltri precedenti o eventuali nuovi dati ricevuti da un servizio di locazione astratto. Ogni target è rappresentato da un indirizzo URI. Se la Request-URI contiene parametri "maddr", l'indirizzo della request-URI deve essere posto come unico target. Se la request-URI non compare tra gli indirizzi direttamente processabili dal proxy, viene impostato l'indirizzo della request-URI come unico target e viene processato il messaggio come una richiesta di inoltro. Se la request-URI non contiene sufficienti informazioni per la decisione dei target, il proxy risponde con un messaggio d'errore (485 Ambiguous). Un target può essere inserito nell'insieme dei target una ed una sola volta affinché si riduca traffico di rete non necessario evitando ricorsioni infinite.

2.2.3.4 Inoltro dei messaggi

Non appena l'insieme dei target contiene degli indirizzi, il proxy può procedere con l'inoltro. Un proxy stateful può seguire un qualsiasi ordine, serialmente attendendo ogni singola risposta, o in parallelo, o arbitrariamente suddividendo i target in gruppi. Un comune metodo di ordinamento è l'utilizzo del parametro qvalue all'interno dell'header, inoltrando i messaggi dal qvalue più alto a quello più basso. Nei proxy stateful deve essere memorizzato tutto il set dei target a cui vengono inoltrati i messaggi, in modo da poter

mandare indietro le risposte ai client che avevano fatto la richiesta di inoltro. Per ogni target vengono seguiti i seguenti passi:

- copia della richiesta di inoltro;
- aggiornamento del request-URI;
- aggiornamento del counter degli hops;
- aggiunta di un campo record-route all'header (opzionale);
- aggiunta opzionale di altri campi all'header;
- post processamento delle informazioni di routing;
- determinazione del successivo hop, della porta e del protocollo di trasporto;
- aggiunta di un indirizzo nel campo VIA dell'header;
- aggiunta se necessario di un campo Content-lenght;
- inoltro della richiesta;
- partenza di un timer C.

Una copia della request va conservata e non va in alcun modo modificata. La request-URI della richiesta viene sostituita con l'indirizzo del nuovo target. Se l'URI contiene parametri non previsti, questi vengono rimossi. Il proxy decrementa il max-forward counter di uno. Se il campo risulta essere vuoto viene impostato il valore di default 70, se il counter raggiunge il valore

0 il messaggio non viene inoltrato ulteriormente. Se il proxy vuole essere inserito nel percorso per eventuali messaggi di richiesta, può inserire il proprio indirizzo nel record-route anche se è presente già un record-route nella richiesta. Il proxy può aggiungere al messaggio copiato altri campi nell'header a seconda delle informazioni necessarie da inoltrare. È possibile che un proxy abbia certe regole di inoltro, come per esempio dei percorsi prestabiliti da fare seguire al messaggio prima di essere inoltrato. Per effettuare questo meccanismo, il proxy deve prima valutare se il set di proxy da attraversare prima di inoltrare il messaggio sia composto da loose router. In genere risulta facile effettuare questo controllo, poiché di norma questi proxy sono tutti appartenenti allo stesso dominio e quindi configurati opportunamente. Una volta accertata questa condizione, il proxy aggiunge nell'header route gli indirizzi da seguire prima dell'inoltro. Se il campo route non è presente viene aggiunto. È possibile avere policy locali per l'inoltro dei messaggi a specifici indirizzi, porte o con specifici protocolli di trasporto, indipendentemente dai valori stabiliti nei campi route o nella request-URI. Anche in questo caso, condizione necessaria per questo tipo di meccanismo è che gli indirizzi bersaglio siano loose router. Tuttavia questo processo è fortemente sconsigliato, preferendo l'utilizzo di porte e protocolli standard. Quando uno dei target genera un errore, in caso di fallimento o di timeout, il proxy inoltra il successivo messaggio al successivo target nel set. Nel caso in cui tutto il set sia esaurito e tutti gli inoltri sono andati in timeout, il proxy risponde con un messaggio 408 (request timeout). Prima di inoltrare il messaggio il proxy

deve aggiungere il proprio indirizzo nel VIA header. Questo campo viene utilizzato da alcuni proxy che si occupano di scoprire eventuali loop. Nel caso in cui sia richiesto, il proxy deve essere in grado di leggere la lunghezza del campo content-lenght, aggiornandolo prima di inoltrare il messaggio. Al fine di evitare la possibilità di messaggi che rimangano in fase di spedizione per un tempo infinito, il proxy per ogni messaggio inoltrato stabilisce un tempo di timeout, allo scadere del quale l'inoltro risulta fallito. Questo tempo deve essere superiore ai 3 minuti poiché è contemplato anche il tempo di ringing oltre a quello di routing.

2.2.3.5 Processamento delle risposte

Quando un elemento riceve una response, la prima azione da compiere è quella di determinare se esiste un client transaction corrispondente. Nel caso in cui non venga trovato un processo appropriato, la response viene inoltrata in modalità stateless. I passi da seguire nell'inoltro della risposta sono i seguenti:

- determinazione del context;
- aggiornamento del timer;
- rimozione dell'indirizzo in testa nel campo VIA (se il campo VIA è vuoto il proxy era destinatario del messaggio e non è necessario effettuare ulteriori inoltri);
- aggiunta della response nel context;

- determinazione della tipologia di inoltro (se necessario inoltro immediato);
- se necessario determinare la risposta migliore;
- aggiungere parametri di autenticazione se necessario;
- riscrittura del header route (opzionale);
- inoltro della response;
- generare tutte le richieste CANCEL necessarie.

Se il transport layer notifica degli errori nell'inoltro di una request, il proxy deve comportarsi come se avesse ricevuto una response di tipo 503 (service unaviable). Se la notifica d'errore invece viene ricevuta nell'inoltro di una response, la response viene eliminata. Un proxy stateful può generare un CANCEL in qualsiasi momento per qualsiasi request inoltrata. Ed in questo caso tutti i proxy che ricevono un CANCEL devono cancellare tutte le richieste corrispondenti.

2.2.4 Redirect server

Un redirect server è un'entità che accetta richieste SIP e ne rimappa gli indirizzi ritornando questi indirizzi al client che ha inviato la richiesta. A differenza del proxy, non può accettare delle chiamate, ma può generare risposte che diano informazioni allo UAc, circa l'instradamento da effettuare per contattare altre entità. Un redirect server aiuta il SIP User Agent, fornendo

informazioni sulla collocazione degli utenti chiamati, nonché possibilità di reindirizzamenti alternativi, rendendo maggiore la raggiungibilità degli utenti. Spesso questa funzione può tornare utile soprattutto per utenti mobili. I server redirect si utilizzano all'interno di una rete SIP per ridurre il carico dei proxy server che sono responsabili degli instradamenti. I server redirect permettono agli altri server di aggiungere informazioni sui percorsi nei messaggi di risposta ai client e si tirano fuori dal ciclo per il resto della connessione. Infatti il client che ha composto la richiesta, una volta ricevuto l'instradamento dal server redirect comporrà i successivi messaggi con l'URI ricevuto. Questo procedimento permette una grossa scalabilità anche per reti molto grandi. Il redirect è composto sostanzialmente da due entità:

- server transaction layer;
- transaction user.

Il transaction user contatta il database contenente la mappatura degli indirizzi URIs, mentre il server redirect mantiene lo stato transaction per l'intera transazione SIP. Nelle risposte di tipo 3xx il server redirect popola la lista dei Contact con una o più locazioni alternative. Inoltre è possibile aggiungere dei parametri "expires" nell'header, che indicano il tempo di vita dei dati di reindirizzamento. Il contact header può contenere URI delle nuove locazioni o semplicemente delle informazioni su nuovi protocolli di trasporto utilizzati (es. passaggio da UDP a TCP). Il redirect server controlla che l'URI da redirigere non sia lo stesso dell'URI-Request, affinché non si creino cicli

infiniti di instradamento. I parametri di expire nel contact header indicano per quanto un URI è valido e sono espressi in numero di secondi. Il valore di default è 3600. È possibile che il client che richiede un redirect dia un canale alternativo di redirezione come un canale su linea pstn o un indirizzo email. In quel caso il redirect server deve comunicare ai client che hanno fatto una request per quel target un response message contenente le informazioni riguardanti la nuova location ed il tipo.

2.2.5 Registrar server

Un server di registro accetta richieste di tipo REGISTER. Questo componente immagazzina tutte le informazioni necessarie al rintracciamento degli utenti che all'inizio di una connessione sono tenuti a registrare la loro presenza all'interno della rete. Il procedimento di registrazione inizia con una richiesta di tipo REGISTER. Il server Registrar è un tipo particolare di UAs. Questo funziona come interfaccia tra la rete ed un server locator e si occupa della lettura e della scrittura delle mappature di rete in base alle request REGISTER. Queste request possono aggiungere, rimuovere e ricercare una corrispondenza nel database del server locator dei contatti relativi ad un utente. La costruzione di un messaggio di tipo REGISTER per uno UAc è esattamente identica a quella della creazione di un messaggio INVITE. Una richiesta REGISTER non stabilisce dialogo. È possibile che un UAc inserisca nell'header della request un percorso stabilito di routing. Tuttavia questo viene semplicemente ignorato una volta seguita la procedura di re-

gistrazione. I campi dell'header NECESSARI in una request REGISTER sono:

Request-URI nella forma "sip:domain.it" contenente informazioni sul dominio di appartenenza (sono esclusi lo username e @);

To e From che devono avere forma di SIP URI o SIPS URI;

Call-ID identificativo di chiamata che distingue univocamente una procedura di autenticazione su uno specifico server registrar da parte di un UAc;

Cseq numero di sequenza utilizzato dallo UA ed incrementato ogni volta che viene emessa una nuova request con lo stesso Call-ID;

Contact questo campo è opzionale e può contenere informazioni di una o più corrispondenze di indirizzi. All'interno di questo campo possono essere presenti i parametri "action" (obsoleto) e "expires" (tempo di validità della corrispondenza: es. sono raggiungibile presso mario.rossi@ufficio.it per 6 ore).

Uno UA non deve mandare ulteriori richieste finché non ha ricevuto una risposta finale da parte del server Registrar, ovvero se la request REGISTER è caduta in timeout. La request REGISTER invia informazioni al server registrar includendo l'indirizzo al quale essere contattato. Una volta stabilita una corrispondenza di questo tipo, il client può cancellare o modificare

la sua registrazione. Lo UAc può inoltre aggiungere nella REGISTER request informazioni circa la durata della sua registrazione. Questo è possibile tramite il campo dell'header Expires oppure con il parametro "expires" del campo Contact. Nel caso in cui non sia espressamente specificata la durata di registrazione è compito del registrar occuparsi di stabilire un tempo di default. Se vengono effettuate registrazioni diverse da uno stesso client, il server registrar le ordina secondo un parametro "q" all'interno del campo contact header. Una corrispondenza può essere rimossa in base al valore di expire oppure per limiti dovuti alle policy del registrar server. Inoltre è possibile rimuovere un binding esplicitamente. Un meccanismo di rimozione è quello di inoltrare una request REGISTER con parametro expires = "0". La response 200 OK ad una request REGISTER conterrà tutte le corrispondenze relative alla contact list dell'utente che ha richiesto la registrazione. Per effettuare una registrazione bisogna determinare la locazione del server registrar. Un UAc per determinare questa posizione può leggerlo nella configurazione statica. Oppure può richiedere la registrazione al proprio dominio (es. "sip:sipdomain.com"). La terza possibilità è quella di inoltrare una richiesta in multicast. Questo tipo di richiesta viene scartata da tutti gli elementi diversi dal server registrar. Un server registrar è un particolare UAs che può rispondere a delle request di tipo REGISTER ed ha in memoria una lista di corrispondenze accessibili ai proxy ed ai redirect all'interno del proprio dominio amministrativo. Un registrar non può generare risposte di classe 6xx. Un server registrar può processare le request REGISTER nell'or-

dine di arrivo e può comportarsi da redirect inoltrandole in caso di messaggio 302 (Moved temporarily). Il server registrar ignora i campi route dell'header. Ogni messaggio viene processato indipendentemente dagli altri. Passi da seguire per processare una request REGISTER:

- controllo Request-URI, il server controlla che la richiesta sia effettuata per un dominio di sua appartenenza. Nel caso in cui non sia verificata la corrispondenza, se il server funziona anche da proxy, ridirige la richiesta al registrar opportuno, in alternativa scarta la richiesta;
- per compatibilità con successive estensioni è necessario processare i campi require dell'header;
- esecuzione del meccanismo di autenticazione;
- determinazione delle policy dello UA (controllo delle azioni accessibili);
- controllo dell'address-to-record. Se questo campo non è compatibile con il dominio nella request-URI viene inoltrata una risposta d'errore 404 (not found);
- processamento del contact field. Se il campo non è presente si passa al passo successivo. Se il campo è presente si controlla innanzi tutto se sono presenti parametri per la rimozione. Nel caso di incongruenze viene mandato un messaggio d'errore 400 (Invalid Request);
- processamento del contact address nel contact header. Per ogni contatto viene controllato il tempo di validità e, nel caso in cui questo

parametro non sia settato, viene impostato ai valori di default o di configurazione del registrar. Se il tempo di expires è inferiore al tempo minimo di registrazione supportato dal server registrar, il server può inoltrare una risposta d'errore 423 (interval too brief);

- per ogni indirizzo viene ricercata la corrispondenza. Se la corrispondenza non è presente, viene aggiornata la tabella delle corrispondenze aggiungendo il nuovo record. Se la corrispondenza è presente si controlla il Call-ID che se corrispondente viene accettato, se differente viene scartato, in modo da ignorare le richieste di REGISTER out of order;

- modifiche o aggiornamenti della tabella delle corrispondenze avvengono solo quando tutte le modifiche sono eseguite con successo. Nel caso in cui una delle modifiche fallisca, vengono scartate tutte le modifiche e viene inoltrato un messaggio 500 (server error).

Una volta eseguiti tutti i passi precedenti con successo, viene inoltrata la risposta "200 OK" contenente tutte le corrispondenze.

2.2.5.1 Autenticazione

L'autenticazione in SIP avviene tramite autenticazione HTTP. Ogni volta che viene ricevuta una request da uno UA è possibile far partire la procedura di autenticazione per assicurarsi dell'identità del richiedente. Una volta riconosciuta l'origine della request è possibile stabilire se può ricevere una risposta. L'UAs può emettere una risposta 401 (Unauthorized) quando è

necessaria l'autenticazione. Lo stesso dicasi per registrar e redirect. Il proxy invece emette un messaggio 407 (proxy authentication required). Se un server non necessita di autenticazione per particolari tipi di request, queste possono essere emesse con username "anonymous" e password vuota. È possibile che ci siano singoli UAc a cui accedono diversi utenti. Ad esempio i PSTN gateway, si autorizzano sul dominio SIP con un unico username identificativo, ma gestiscono le proprie policy all'interno per gestire i diversi utenti. Il processo non è specificato nell'rfc 3261. All'interno dell'header dei messaggi vengono aggiunti dei campi credenziali utili all'autenticazione. Il processo di autenticazione può essere di tipo server-client, user-to-user, proxy-to-user. Lo schema di autenticazione, nonostante sia basato sull'autenticazione http, ha subito dei cambiamenti. In principio si utilizzava un meccanismo di autenticazione di tipo base ormai obsoleto e non più accettato dalle nuove architetture. Tra le specifiche aggiuntive della digest authentication:

- formattazione dell'URI durante la negoziazione secondo la BNF:

  <URI> := <SIP−URI> / <SIPS−URI>

- mentre nel protocollo http l'URI non è incluso nel quotation mark, nell'autenticazione sip questo è previsto;

- la BNF per il digest-uri-value è: ¡digest-uri-value¿ := ¡Request-URI¿.

Esistono molte applicazioni su internet che richiedono la creazione ed il mantenimento di una sessione per uno scambio di dati tra due o più uten-

ti. Per questo sono molteplici le possibilità di applicazione dello stack sip. Infatti spesso sorgono problematiche circa la posizione e la mobilità degli utenti all'interno di una rete ovvero nelle modalità di connessione, e quindi sono stati sviluppati numerosi protocolli che supportano svariate forme di multimedia in tempo reale, come voce, video, testo, spesso utilizzati contemporaneamente. SIP permette di invitare nuovi partecipanti su sessioni pre-esistenti, supportando anche le conferenze in multicast. È possibile anche a sessione aperta aggiungere nuovi servizi, quali il supporto di nuovi media o di nuovi protocolli di trasporto. Durante la creazione, il mantenimento e la terminazione di una sessione di comunicazione, SIP contiene le informazioni riguardo:

User location: per la determinazione del posizionamento, all'interno della rete, dei punti di accesso dei sistemi coinvolti e dei partecipanti;

User availability: per la disponibilità degli utenti ad iniziare una sessione di comunicazione;

User capabilities: per stabilire i tipi di media supportati ed i parametri utilizzabili durante la sessione di comunicazione;

Session setup: fase di start-up in cui la connessione viene stabilita e vengono fissati i parametri di comunicazione tra utenti chiamanti e chiamati;

Session management: mantenimento della sessione, informazioni riguardo il trasferimento e la terminazione della sessione stessa, e delle modifiche avvenute successivamente allo start-up, inclusi l'invocazione di nuovi servizi o nuovi protocolli di trasporto.

SIP può essere integrato nelle architetture multimediali esistenti, essendo compatibile con gli altri protocolli standard IETF come H.323, permettendo la costruzione di architetture complete che supportino protocolli Real-Time come RTP (Real-Time Protocol) e RTPS (Real-Time Streaming Protocol), o la commutazione sulla rete di telefonia pubblica tramite l'utilizzo di MEGACO (Media Gateway Control Protocol) , o la descrizione di sessioni multimediali per mezzo del protocollo SDP (Session Description Protocol). SIP non fornisce direttamente dei servizi, piuttosto stabilisce le primitive sulle quali i servizi possono essere implementati. SIP garantisce un buon livello di sicurezza fornendo una suite di servizi che includono prevenzione da DoS (Denial of Service), sistemi di autenticazione, controllo di integrità, crittografia e garanzie sulla riservatezza e segretezza dei dati. È previsto l'utilizzo di SIP sia su IPv4 che su IPv6.

Capitolo 3

Obiettivi e scelte progettuali

Il mondo della telefonia, sia mobile che fissa, ha subito una forte crescita negli ultimi decenni ed è entrato ormai nella vita quotidiana di ognuno di noi. Dietro i gesti, che ora ci sembrano normali, c'è un secolo di evoluzione tecnologica, una rete fitta e capillare distribuita in tutto il globo fatta di centrali di smistamento, giunzioni in fibra ottica, collegamenti che stabiliscono quei percorsi che rendono così semplice la comunicazione. Anche accedere alla propria posta elettronica, o navigare in internet, è diventato altrettanto semplice e sempre più di uso comune, e tutto in un tempo circa dieci volte più breve. Dietro c'è di nuovo una rete, altrettanto fitta e sovrapposta a quella telefonica. Il telefono, con tutte le sue varianti, è ormai arrivato alla saturazione in quanto crescita, mentre internet è nel pieno dello sviluppo. Negli ultimi anni sono state raggiunte velocità di trasmissione più elevate, basti pensare alla diffusione dell'ADSL, sono stati sviluppati algoritmi più

efficienti di compressione dell'audio e del video, sono stati pensati protocolli per velocizzare le trasmissioni di contenuti multimediali, i tempi di risposta sono diventati più brevi. Questa evoluzione oggi consente, proprio attraverso tecnologie nate per il trasporto di dati, una comunicazione multimediale in tempo reale. In questo contesto nasce VoIP (Voice Over IP) un insieme di tecniche che consente la comunicazione telefonica attraverso internet. Usando la propria connessione a internet, attraverso un PC o un dispositivo mobile dotati di sistemi I/O audio, si possono effettuare vere e proprie telefonate a singoli o conferenze con molti partecipanti. Dal suo debutto nel 1995, VoIP ha continuato a suscitare interesse:

- presso privati e aziende per il forte risparmio che introduce, soprattutto se le chiamate sono a lunga distanza;
- per gli operatori ed i tecnici invece, perchè segna l'inizio di un processo di convergenza tra i servizi internet e di telefonia su un'unica rete basata sulla trasmissione dei dati.

Avere una rete multiservizi apre nuovi orizzonti alla comunicazione e al supporto di nuove tecnologie, tutto questo senza modifiche alle infrastrutture preesistenti. Da qui nasce l'esigenza di avere un sistema facile ed intuitivo che dia la possibilità all'utente di eseguire telefonate utilizzando sistemi VoIP su dispositivi mobili. Si è inotre sentita l'esigenza di aggiungere una funzionalità sperimentale che consenta di migliorare in maniera significativa la gestione del traffico VoIP, sfruttando tutte le interfacce di cui è dotato

il dispositivo. Finora un dispositivo mobile, nonostante fosse dotato di diverse interfacce di rete (NICs), ne sfruttava solo una per volta costituendo un grande limite alla comunicazione e allo sfruttamento ottimale delle risorse a disposizione. Per ovviare a questo problema, si è pensato di prendere in considerazione una tecnologia relativamente nuova, sviluppata presso il dipartimento di Scienze dell'Informazione dell'Università di Bologna, chiamata ABPS (Always Best Packet Switching). L'applicazione che supporta questa funzionalità può usare simultaneamente tutte le interfacce di rete disponibili ed è in grado di differenziarle a seconda del tipo di datagramma, aggiungendo così al nostro dispositivo la possibilità di indirizzare, in qualsiasi momento, i pacchetti verso l'interfaccia di rete più consona. La figura 3.1 mostra come il nostro dispositivo utilizzi le diverse interfacce di rete (WiFi e 3G) per poter raggiungere il server ABPS (Fixed Server) che a sua volta indirizzerà i pacchetti pervenuti all'host destinatario (Corrispondent Node). Più nel dettaglio, il nostro client incapsulerà i pacchetti SIP ed RTP inglobandoli in un ulteriore protocollo proprio, riconosciuto e gestito dal Server ABPS. Quest'ultimo, ricevuti i pacchetti rimuoverà l'header ABPS e li indirizzerà al corretto destinatario nascondendo gli indirizzi del mittente.

Vista la volontà di voler supportare la tecnologia ABPS nella nostra applicazione mobile, si è deciso di utilizzare come backend le librerie PjSIP per cui è già esistente tale supporto ed è noto il suo livello di massima portabilità nelle più svariate architetture. In maniera più approfondita si parlerà delle caratteristiche di PjSIP nel prossimo capitolo.

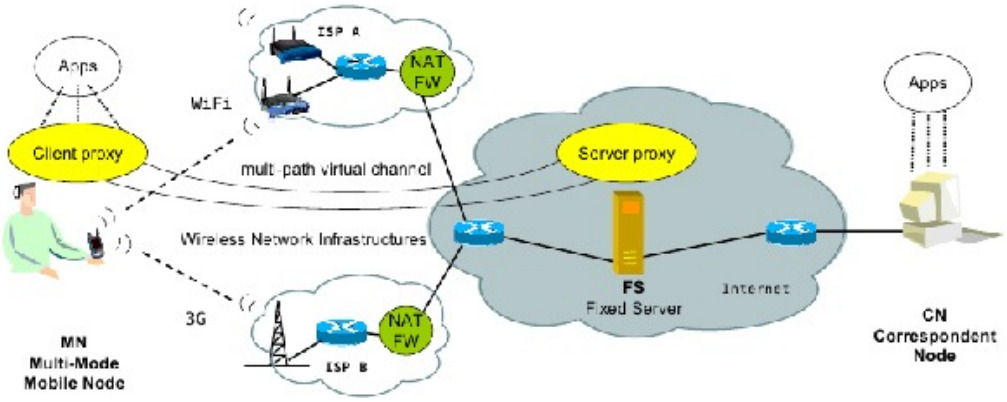

Figura 3.1: Tecnologia ABPS

Un altro punto cruciale per la realizzazione di un'applicazione VoIP per dispositivi mobili è la scelta di un framework per la gestione dell'interfaccia grafica che sia il più possibile semplice, 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. 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 sistema operativo (thread, mutex, semafori, I/O, etc...) e per la creazione di interfacce grafiche. Non aiuta il fatto che ormai quasi tutti i dispositivi siano dotati di firme digitali, compreso il famoso sistema operativo Android, che seppur a codice sorgente aperto può essere compilato ed emulato su un normale PC ma non eseguito direttamente sull'hardware specifico.

Questo particolare non è da sottovalutare ed è di grande attualità, infatti questo taglia fuori una gran fetta di tools o di framework con licenze opensource, ricordando che sono proprio le comunità open ad aver sempre spinto sulla portabilità tra sistemi e molto spesso ad aver portato vere e proprie innovazioni. Detto questo, sembrerebbe impossibile trovare un framework che sia portabile tra i vari dispositivi: infatti per problemi di portabilità sono stati subito scartati sia gli SDK specifici per iOS, Android e Windows Mobile sia i framework scritti in linguaggi interpretati (es. java, per motivi di performance).

Le scelte sono dunque rimaste principalmente due:

1. Nokia Qt Library;

2. wxWidgets.

Sia Nokia Qt Library 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 opensource e dispongono di molteplici licenze per quel che riguarda sia sorgenti a codice aperto, che chiuso. In passato Qt era utilizzato con più diffidenza da parte degli 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 sistemi operativi per pc come Linux, Mac, Windows, e

BSD. Avendo raggiunto ormai due decadi di vita, sono entrambi considerati maturi e stabili.

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 progettuale è quindi ricaduta sulle librerie Qt perchè sono nativamente supportate dai dispositivi Nokia (Symbian OS, Maemo, MeeGo, etc...) senza il bisogno di aggiungere ulteriori librerie esterne, ed è in corso il porting per i sistemi operativi più gettonati del momento come Android (sviluppato da Google) ed iOS (sviluppato da Apple per i device iPhone). Nei prossimi capitoli verranno spiegate le caratteristiche in maniera più approfondita.

Altro obiettivo alla pari della realizzazione dell'applicazione, forse anche di più, è stato quello di utilizzare le ultime tecnologie di sviluppo messe a disposizione da Nokia. Si è verificato quanto queste siano mature per dare nuova linfa allo sviluppo per ambienti Symbian e correlati. Si sono fatte delle prove con i vari Nokia Qt SDK. Questi utilizzano compilatori molto più moderni ed efficienti che producono applicazioni più ridotte in termini di spazio occupato, rispetto a quelli utilizzati dalla ormai vecchia piattaforma di sviluppo Carbide C++. Si è inoltre testato il funzionamento di nuove tecnologie da poco introdotte da Nokia: QtQuick e QtMobility. La prima permette uno sviluppo più rapido ed orientato al mondo mobile di

applicazioni dall'interfaccia utente molto accattivante e personalizzabile. La seconda permette di scrivere applicazioni su di un livello di astrazione più alto, tralasciando i dettagli hardware specifici di ogni device e consentendo di ususfruire di servizi tipici di device mobili in maniera semplice.

Capitolo 4

PjSIP

PjSip è un gruppo di librerie per gestire comunicazioni multimediali attraverso il protocollo SIP. Sono pensate per essere estremamente portabili e performanti, adatte per questo ad essere eseguite su architetture più 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, in dettaglio, che hanno fatto di PjSip un backend ideale per le applicazioni voip su dispositivi mobili.

4.1 PjLib

PjLib è la libreria base su cui tutte le altre del gruppo si appoggiano, si occupa di garantire funzionalità di base quali l'astrazione rispetto al sistema operativo sottostante, contiene una replica pressoché completa della libreria libc ed una serie di feature aggiuntive come la gestione dei socket, delle

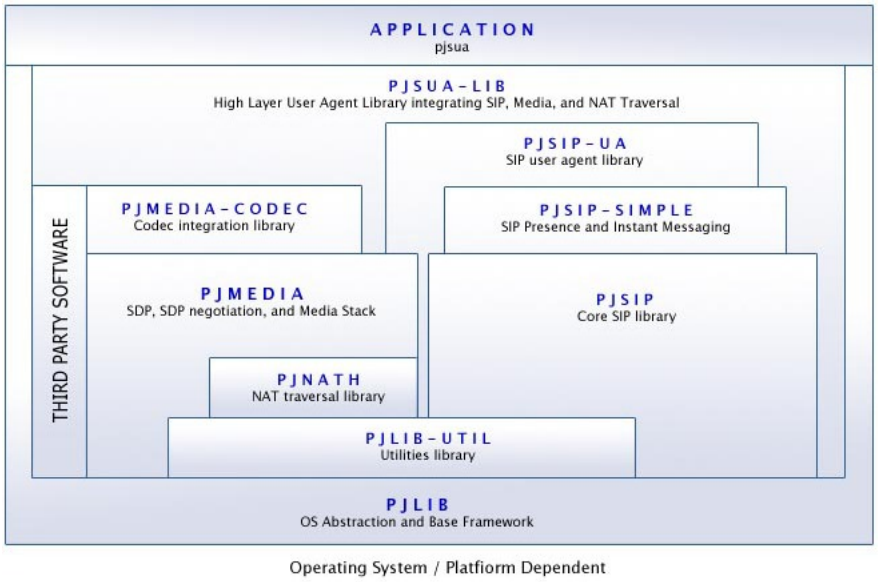

Figura 4.1: Stack PjSip

funzionalità di logging, dei threads, della mutua esclusione, dei semafori, delle critical section, funzioni di timing, nonché 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. Al contrario di quello che si può pensare, queste non usano le librerie ANSI C e le LIBC per motivi di portabilità.

4.2 PjMedia

PjMedia è una libreria complementare a PjSip e si occupa del trasferimento e della gestione dei dati multimediali. Tra le sue caratteristiche principali abbiamo 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/iLBC/GSM/G.711. Alta qualità: supporta l'encoding e il decoding a frequenze di 16/32Khz e la conversione di qualunque campione verso queste frequenze; essa inoltre riesce a tollerare bene i problemi classici legati alle reti ip che sono il jitter e la perdita di pacchetti .

4.3 PjSUA

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

4.4 PjLib_Util

PjLib_Util è invece una libreria ausiliaria che fornisce supporto a PjMedia e PjSip. Alcune sue funzioni sono: funzionalità small footprint di parsing xml, caching asincrono di risolver DNS, funzioni di hashing e di crittografia, etc...

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

Portabilità: possono essere eseguite su 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 eseguito

anche in ambienti dove le librerie ANSI C non sono disponibili, attraverso l'implementazione a carico dello sviluppatore di determinate funzioni strettamente legate all'architettura specifica. Le architetture supportate sono:

- Win32_x86 (Win95/98/ME, NT/2000/XP/2003, mingw);
- Linux_x86 (anche come modulo del kernel);
- Linux_alpha,x86, eCos ultra-II;
- Solaris;
- powerpc;
- MacOS;
- m68k;
- PalmOS;
- arm;
- PocketPC;
- SymbianOS.

Dimensioni: uno degli obiettivi primari della libreria PjSip è avere ridotte dimensioni in modo da essere utilizzata anche in ambiti embedded. È 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: l'idea centrale che verte nello sviluppo delle PjSip è che possano essere eseguite il più velocemente 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 un pool preallocato:

- alloc() ha costo computazionale O(1);
- nessun sistema di sincronizzazione è usato all'interno della funzione alloc(), si assume infatti che la sincronizzazione venga usata ad un livello di astrazione più alto;
- non è necessaria la funzione free() di liberazione della memoria. Tutti i segmenti allocati saranno liberati in una sola operazione quando il pool di memoria verrà deallocato;
- le performance di questo tipo di gestione della memoria possono essere, in certi sistemi, fino a 30 volte superiori rispetto all'utilizzo classico di malloc e free.

Astrazioni del sistema operativo: normalmente esistono alcune caratteristiche dei vari sistemi operativi che rendono lungo e complesso lo sviluppo di applicazioni multi piattaforma. 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, etc...

I/O di rete a basso livello: PjLib ha un set proprio di API per gestire le comunicazioni attraverso rete. Queste ultime 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;
- select() API, per comodità è stata implementata la funzione select() normalmente utilizzata nei sistemi *nix.

Strutture dati comuni: si hanno:

- operazioni su stringhe;
- array;
- hash table;
- linked list;
- alberi bilanciati (Red/Black trees).

Gestione delle eccezioni: un conveniente sistema simil TRY/CATCH di propagazione dell'errore, semplificando così la scrittura di alcune funzioni interne:

```
define SYNTAX_ERROR   1
```

```
PJ_TRY {
        msg = NULL;
        msg = parse_msg(buf, len);
}
PJ_CATCH ( SYNTAX_ERROR ) {
        .. handle error ..
}
PJ_END;
```

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, printk, file, etc..) mantenendo invariata la sintassi tra le diverse architetture supportate. Un esempio di quanto detto sopra può essere sintetizzato con questo blocco di codice in puro stile PjSip:

```
/* Tipi di dato specifici di pjsip per avere
* il massimo della portabilita'.
* Da notare che esistono anche i tipi di dato
* piu' comunemente usati quali
* stringhe e le relative funzioni per la loro
* creazione e gestione
```

```
 */

pj_hostent he;
pj_status_t rc;
pj_str_t host = pj_str("host.example.com");

// funzioni ad hoc per le operazioni di rete piu'
// comuni
rc = pj_gethostbyname( &host, &he);
if (rc != PJ_SUCCESS) {
    char errbuf[80];
    pj_strerror( rc, errbuf, sizeof(errbuf));

    // Framework per semplificare la gestione
    // del log generato
    // dall'applicazione

    PJ_LOG(2,("sample",
              "Unable to resolve host,error=%s",
              errbuf)
          );
```

```
        return rc;
    }
```

Capitolo 5

Nokia Qt Framework

Qt è una libreria di classi ed un insieme di strumenti per lo sviluppo di inferfacce utente grafiche (GUI) multi piattaforma, ed è conosciuto anche con il nome di toolkit Qt. Sviluppato in origine dalla norvegese Trolltech, ora acquistata da Nokia, Qt permette lo sviluppo di software secondo la filosofia "write once, compile anywhere", che significa letteralmente "scrivi una sola volta e compila ovunque". In altre parole, Qt permette agli sviluppatori di scrivere codice sorgente perfettamente compatibile su diversi sistemi operativi, quali ad esempio MS Windows, Mac OS X, Linux, Solaris HP-UX, molte versioni UNIX con grafica X11 ed embedded Linux. Il tutto è reso possibile grazie all'adozione di un'interfaccia di programmazione delle applicazioni (API) 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 contengono il toolkit Qt: esso è la base su cui è stato costruisto l'intero desktop environment KDE.

Qt è attualmente usato in molti campi per applicazioni sia opensource che commerciali, supportando un numero sempre crescente di dispositivi mobili quali MeeGo, Maemo, Windows Mobile, Symbian OS e presto anche la piattaforma Android. Le librerie di classi modulari Qt sviluppate in C++ permettono di sfruttare un ricco set di blocchi applicativi già costruiti, lasciando tutte le funzionalità per la costruzione di applicazioni avanzate crossplatform.

5.1 Gestione Segnali (Signals & Slot)

Il framework Qt offre un metodo flessibile per far comunicare tra loro gli oggetti C++ della libreria senza per questo dover dipendere dai metodi di comunicazione nativi, quali ad esempio la gestione degli eventi di Windows o di X11, in modo da rimanere indipendenti dall'implementazione.

Prima di addentrarci nell'analisi del meccanismo di comunicazione adottato da Qt chiamato Signals & Slots, cerchiamo di capire in cosa consiste il problema: supponiamo di avere una finestra di dialogo contenente due oggetti grafici molto semplici, un bottone e un indicatore visivo. Supponiamo inoltre di voler dotare la nostra finestra di una semplice funzionalità: al click del mouse sul bottone grafico, il colore del nostro semaforo diventi

verde. Il semplice esempio sopra riportato, ci pone di fronte al problema di far comunicare tra loro i nostri due oggetti grafici: il pulsante e l'indicatore; in altre parole, quando il pulsante viene premuto, deve essere emesso un "segnale" verso l'indicatore, in modo che esso possa riconoscere l'evento e cambiare colore. In estrema sintesi, si tratta di inviare un segnale scaturito dal click sul bottone (detto signal in terminologia Qt) verso il nostro oggetto "indicatore" che chiamerà un apposito metodo (detto slot in terminologia Qt) per la gestione dell'evento. Allo stato attuale delle cose, abbiamo identificato un segnale e uno slot, ciò che ancora ci manca è un metodo per il loro collegamento. Il meccanismo di meta object system di Qt offre una funzione (connect), indipendente dalla piattaforma, per realizzare questa connessione.

I signal sono i segnali emessi dai widget Qt a fronte di eventi (interni o esterni), mentre gli slot sono praticamente identici alle funzioni membro di una classe; si può quindi parlare di slot pubblici, privati e protetti secondo l'accezione classica del linguaggio C++, e possono essere invocati così come avviene per tutti gli altri metodi tradizionali di una classe.

La differenza sostanziale è che uno slot può sempre essere collegato a un segnale e quindi sarà invocato ogni volta che il segnale verrà emesso. Questo collegamento signal & slot, viene realizzato con la funzione di libreria connect e ha la sintassi:

```
connect ( sender,
          SIGNAL(signal),
```

```
        receiver,
        SLOT(slot)
    );
```

Dove:

- sender e receiver sono i puntatori agli oggetti Qt (nel nostro esempio sender è il bottone e receiver l'indicatore);
- SIGNAL e SLOT sono funzioni (in questo caso senza parametri) appartenenti rispettivamente all'oggetto sender e receiver.

Nel caso invece di collegamento signal/slot con un parametro intero si avrà:

```
connect ( sender,
        SIGNAL(signal(int)),
        receiver,
        SLOT(slot(int))
    );
```

La macro SIGNAL() e la macro SLOT() sono essenziali per il funzionamento del tutto e verranno preprocessate dal meta object compiler (MOC) prima della compilazione di tutti i file del progetto.

Il meta object system di Qt, è un sistema di pre-processing in grado di generare codice C++ puro, partendo dalla definizione degli oggetti Qt. In

questo modo, il meccanismo Signals & Slots, potrà funzionare su qualsiasi sistema operativo: l'unica condizione necessaria è quindi la disponibilità di un compilatore C++ standard (es. GCC). L'uso di MOC e qMake, descritti in seguito, libera quindi il programmatore da tutti i dettagli di implementazione necessari al funzionamento multipiattaforma del meccanismo "Signals & Slots".

Così come esiste una funzione connect, esiste anche la funzione opposta, la disconnect, usata per disconnettere il segnale dallo slot precedentemente collegato. La sintassi della funzione è:

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

In questo modo è possibile attivare e disattivare il collegamento tra oggetti a runtime. Va inoltre ricordato che più segnali possono essere collegati a uno stesso slot, un segnale può essere collegato a più slot o a un altro segnale:

```
connect ( sender,
          SIGNAL(signal1),
          receiver,
          SLOT(signal2)
        );
```

In questo caso, quando viene emesso il segnale signal1, verrà anche emesso il segnale signal2.

5.2 Portabilità del sistema di build

Il framework Qt, offre allo sviluppatore una serie di tool per automatizzare le normali operazioni che garantiscono la portabilità tra un'architettura e l'altra. Discuteremo alcuni dei tool più importanti in dettaglio.

5.2.1 qMake

È un tool che semplifica il processo di build di applicazioni su differenti piattaforme automatizzando la creazione dei cosiddetti "makefile". Il compito dello sviluppatore è scrivere il cosiddetto "project file" (con estensione .pro) dove vengono specificate le informazioni di base per la compilazione del progetto. Un esempio di project file base contiene la specifica dei sorgenti da compilare e le librerie esterne che essi utilizzano. Esempio:

```
SOURCES = hello.cpp
HEADERS = hello.h
CONFIG += qt warn_on release
```

Ne daremo una breve spiegazione riga per riga omettendo il più possibile i dettagli tecnici.

```
SOURCES = hello.cpp
```

Questa linea specifica ogni singolo file che costituisce il codice sorgente della nostra applicazione. In questo caso abbiamo solo un file sorgente, ovvero hello.cpp. La maggior parte delle applicazioni necessitano, ovviamente, di molteplici file sorgente e nel caso è sufficiente elencarli separati da spazi:

```
SOURCES = hello.cpp main.cpp
```

La parola chiave HEADERS è usata, come è ovvio che sia, per specificare i cosiddetti headers file.

```
HEADERS += hello.h
```

CONFIG invece è usato per dare la possibilità allo sviluppatore di aggiungere alcune informazioni riguardo alla configurazione dell'applicazione

```
CONFIG += qt warn_on release
```

Nell'esempio appena discusso vediamo tre diverse impostazioni:

qt indica a qmake che la nostra applicazione fa uso del framework Qt e quindi provvede a rendere disponibile tale libreria durante le operazioni di linking;

warn_on imposta il compilatore in modo che stampi anche i messaggi di warning;

release indica a qmake che l'applicazione deve essere compilata senza le informazioni aggiuntive di debug e quindi per un rilascio ufficiale.

5.2.2 Meta-Object Compiler (MOC)

Il "Meta-Object Compiler", più comunemente chiamato MOC, è un programma che gestisce alcune estensioni C++ proprie di Qt. Il tool in questione (MOC) legge i file header C++ e se trova una o più classi contenenti la macro Q_OBJECT produce un file sorgente C++ contenente dei meta oggetti riferiti a quella classe. Questo è necessario, prima di tutto, al meccanismo di "signals and slots" precedentemente descritto, creando un codice di gestione segnali portabile tra un'architettura e l'altra. Il file sorgente generato dal tool moc deve essere, ovviamente, compilato e "linkato" con l'implementazione della classe stessa. Se si utilizza Qmake per la creazione dei makefiles, moc viene chiamato implicitamente e quindi non è necessario il suo utilizzo esplicito.

5.2.3 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 UIC converte questi file XML di descrizione in file header C++ rendendo più facile la suddivisione tra la parte puramente estetica e la parte puramente implementativa. Questo tool può essere lanciato esplicitamente oppure implicitamente attraverso Qmake aggiungendo al project file una linea di questo tipo:

```
FORMS = hello.ui
```

Come tutti i generatori di codice è da usare con parsimonia nel caso si voglia ottimizzare particolari aspetti della creazione dei componenti Qt.

5.2.4 Portabilità delle strutture dati

Qt, oltre al suo ottimo framework per costruire interfacce grafiche, offre anche alcune caratteristiche molto importanti, utilizzabili anche per applicazioni testuali su console:

- le classiche strutture dati usate normalmente durante lo sviluppo di applicazioni generiche quali liste, tabelle hash, alberi etc...
- funzioni per la creazione e la gestione dei thread nonché i sistemi di sincronizzazione quali mutex o semafori.

5.2.5 Gestione della memoria con Qt

In C++, la gestione dinamica della memoria avviene utilizzando l'operatore new per l'allocazione e l'operatore delete per la deallocazione della memoria. Solitamente, dunque, si può procedere come di seguito:

```
#include <string>
#include <iostream>

using namespace std;

class StringWrapper {
```

```
        public:
            StringWrapper(const string& str = "")
                : str(str){}
            string text() const { return str; }
            void setText(const string& str) {
                this->str = str; }
        private:
            string str;
};

int main(){
        StringWrapper* a = new StringWrapper();
        StringWrapper* b = new StringWrapper();

        a->setText("stringa a");
        b->setText("stringa b");

        cout << a->text() << " | " << b->text()
             << endl;

        delete a;
        delete b;
```

}

Nella libreria Qt il tutto avviene in maniera più pulita. Innanzittuto, tutte le classi Qt ereditano dalla classe QObject, dunque creeremo una classe da essa derivata. La classe QObject accetta un oggetto QObject chiamato parent indicante il genitore dell'oggetto. Tale oggetto si occuperà di deallocare la memoria per ogni QObject che ha come figlio:

```
#include <QObject>
#include <string>
#include <iostream>

using namespace std;

class StringWrapper : public Qobject {
  public:
        StringWrapper(const string& str = "",
                      QObject* parent = 0) : QObject(parent),
                      str(str){}
        ~StringWrapper() {
                cout << "StringWrapper::~StringWrapper()"
                      << endl;
        }
```

```
        string text() const {
                return str;
        }
        void setText(const string& str) {
                        this->str = str;
        }
    private:
        string str;
};

int main() {

    QObject parent;

    StringWrapper* a = new StringWrapper("", &parent);
    StringWrapper* b = new StringWrapper("", &parent);

    a->setText("stringa a");
    b->setText("stringa b");

    cout << a->text() << " | " << b->text() << endl;
}
```

Così facendo non richiameremo esplicitamente l'operatore delete poichè è stato richiamato dall'oggetto parent. Attraverso l'utilizzo di questo genere di approccio si possono evitare parte dei problemi riguardanti il corretto utilizzo della memoria, lasciando a Qt l'onere di liberare la memoria quando non è più effettivamente necessaria.

5.3 QtQuick

Qt Quick è una delle nuove tecnologie offerte da Nokia per lo sviluppo delle applicazioni. Fondamentalmente cambia la maniera in cui implementare interfacce utente, dando la stesso potere di personalizzazione ai designer quanto ne hanno gli sviluppatori della logica applicativa. Questa nuova modalità di sviluppo permette di astrarre lo sviluppatore dai problemi del C++ e della sua compilazione.

Il comportamento delle interfacce grafiche è gestito da un nuovo linguaggio dichiarativo chiamato QML molto simile a Javascript. QML ha di default diversi layout standard che permettono di realizzare facilmente interfacce grafiche touch-friendly. Qt Quick è molto meno rigido dei tradizionali toolkit per lo sviluppo di interfacce grafiche, perchè permette allo sviluppatore di realizzare semplicemente interfacce personalizzate e quindi di uscire dai canoni imposti dagli oggetti discreti tradizionali.

È possibile costruire intere applicazioni utilizzando unicamente QML e JavaScript, ma QML è ancora più potente quando viene usato in collab-

Figura 5.1: Esempio di interfaccia realizzata con QtQuick

orazione con C++. Qt 4.7 offre una visione dichiarativa dei widget che i sviluppatori possono utilizzare per caricare un layout QML all'interno di una finestra Qt convenzionale di un'applicazione C++. Le funzioni e le strutture dati sottostanti del programma in C++, possono essere esposte attraverso una visione dichiarativa e rese accessibili da QML, permettendo agli sviluppatori di implementare le parti più sensibili dal punto di vista prestazionale.

QML è stato progettato da Qt per sfruttare il modello MVC (Model View Controller), il che significa che le attuali applicazioni Qt progettate attorno a classi di Qt MVC, si potranno facilmente aggiornare all'interfaccia utente QML. La facilitazione appena citata, permetterà un porting senza patemi di applicazioni desktop ad applicazioni nell'ambito mobile. Gli sviluppatori

dovranno semplicemente aggiungere alcuni ruoli al loro modello per poi esporli alla vista QML, dove una interfaccia touchscreen amichevole può essere creata con un po' di scripting.

Un altro grande vantaggio di Qt Quick, è che gli sviluppatori possono facilmente supportare una vasta gamma di device, ognuno dei quali ha le proprie peculiarità come può essere la dimensione dello schermo. Si potranno quindi utilizzare molteplici layout QML o certe espressioni condizionali per dare visibilità ad alcuni elementi a seconda delle dimensioni dello schermo.

Ora mostriamo uno spezzone di codice QML con il rispettivo risultato grafico mostrato dalla Figura **??**:

```
import QtQuick 1.0

Rectangle {
    id: rectangle1
    width: 320
    height: 280
    gradient: Gradient {
        GradientStop {
            position: 0
            color: "#a1d2f7"
        }
```

```
        GradientStop {
            position: 1
            color: "#000000"
        }
    }
    Text {
        id: helloText
        x: 23
        y: 29
        text: "Hello World Qt!"
        opacity: 0.6
        horizontalAlignment: Text.AlignHCenter
        verticalAlignment: Text.AlignTop
        style: Text.Normal
        font.underline: true
        font.italic: true
        font.pointSize: 28
        font.bold: true
        anchors.verticalCenterOffset: -89
        anchors.horizontalCenterOffset: 1
        anchors.centerIn: parent
    }
```

```
    Image {
        id: imageQt
        x: 83
        y: 100
        width: 156
        height: 158
        anchors.verticalCenterOffset: 39
        anchors.horizontalCenterOffset: 1
        anchors.centerIn: parent
        smooth: true
        fillMode: Image.Stretch
        source: "qt.png"
    }
}
```

5.3.1 Disponibilità

Il Qt 4.7 SDK include: il toolkit, l'ambiente di sviluppo, e l'ambiente di sviluppo Qt Creator 2.1. È disponibile per Windows, Linux e Mac OS X. Per Symbian dobbiamo fare riferimento ad una versione specifica chiamata "Nokia Qt SDK" che prevede processi di building sia per Symbian sia per Maemo e dei simulatori appositi. Il supporto a MeeGo secondo Nokia ci sarà

Figura 5.2: Risultato grafico del QML

a breve.

Nokia prevede di fornire su tutti i suoi nuovi devices, la preinstallazione delle librerie Qt alla versione 4.6 o 4.7. Per chi non ha le librerie preinstallate può facilmente installarle attraverso pacchetti autoinstallanti .sis.

5.4 QtMobility

Il progetto Qt Mobility rende disponibile un nuovo insieme di API con funzionalità che sono ben note al modo dei dispositivi mobili, in particolare nei cellulari. Tuttavia, queste nuove API permettono allo sviluppatore di usare queste nuove caratteristiche con la facilità di un framework, ed applicarle ai cellulari, netbook e dispositivi non propriamente mobili. Il framework in questione non migliora solamente molti aspetti dell'esperienza mobile, perchè ne migliora anche l'utilizzo di queste tecnlogie; inoltre hanno utilizzo che va al di là del mondo mobile.

Le funzionalità coperte dal framework sono molteplici dando diverse funzionalità e tecnologie, rendendo Qt Mobility una collezione di API e framework. L'ultima release disponibile è la 1.1.0 fornita con Qt 4.7 che permette un'interfacciamento anche con QML. Le API disponibil sono:

Bearer Management: un'API per controllare lo stato di connettività del sistema;

Contacts: un'API che abilita i client nel recupero dei contatti dai dispositivi locali o remoti;

Document Gallery: un'API per navigare e richiedere documenti usando i loro meta-data;

Feedback: un'API che permette al client di avere dei feedback tattili e audio generate dalle azioni degli utenti;

Location: questa API provede una libreria per recuperare la posizione, la gestione delle mappe e la navigazione;

Messaging: permette di accedere al servizio di messaggistica;

Mobility QML Plugins: fornisce un insieme di plugin compatibili con QML per il mobility project;

Multimedia: fornisce un insieme di API per riprodurre, registrare e gestire una collezione di media;

Organizer: fornisce un'API per il recupero di calendari e schedulazione di dati personali da locale o remoto;

Publish and Subscribe: permette alle applicazioni di leggere dei valori, a cui ci si può registrare, ricevendo poi notifica nel caso di un loro cambiamento;

Qt Service Framework: mette a disposizione un insieme di API che permette ai client di trovare ed istanziare servizi;

Sensors: prevede un'API per la gestione dei sensori;

System Information: permette di recuperare informazioni e capacità di sistema;

Versit: permette l'import e l'export in formato vCard e iCalendar;

Qt Mobility, riassumendo, rende possibile la gestione facilitatà di queste funzionalità:

- recuperare servizi locali o remoti, trovando una connessione ottimale per l'utilizzo di questi;
- servizi costruiti sopra queste API possono includere applicazioni internet come email e browser;
- funzioni multimediali che permettono di catturare immagini e video con l'audio, registrare/riprodurre audio e video;

- il servizio di location da la consapevolezza al dispositivo della sua posizione geografica;

- Publish-Subscribe permette la comunicazione tra diversi oggetti distinti sia in locale che in remoto.

Il framework QtMobility è utilizzabile all'interno del namespace QtMobility, con l'eccezione di Multimedia.

5.5 Risparmio energetico sui sistemi mobile

Ogni piattaforma mobile è progettata pensando prima di tutto all'efficenza del sistema ed è quindi ottimizzata per gestire al meglio l'uso della batteria. D'altro canto, applicazioni sviluppate senza tenere in considerazione l'effettivo utilizzo di queste risorse possono avere un impatto considerevole nell'efficienza energetica del dispositivo. Nonostante il framework Qt sia raccomandato per lo sviluppo di applicazioni mobili, sia per portabilità che per risparmio energetico, è bene considerare, specialmente per quegli sviluppatori che provengono dal mondo desktop, qualche piccola linea guida che permette di rendere l'applicazione utilizzabile sui dispositivi embedded, senza troppi limiti o compromessi dettati dallo spreco di risorse e quindi di batteria.

5.6 Strategie per la conservazione energetica in ambito mobile

5.6.1 Reagire in maniera appropriata ai cambiamenti di stato del dispositivo

Uno degli aspetti più importanti per la conservazione della batteria, è di evitare il più possibile operazioni non necessarie, specialmente quando producono risultati che non devono essere visualizzati all'utente. L'applicazione dovrebbe infatti accorgersi dello stato di inattività del dispositivo interrompendo temporaneamente le attività superflue (es. le animazioni). I principali momenti in cui l'applicazione in questione dovrebbe entrare in uno stato di inattività sono:

- esecuzione in background;
- risparmio energetico o modalità stand by.

Questi stati di inattività ci vengono segnalati dal framework Qt attraverso degli eventi appositi:

QEvent::WindowDeactivate viene richiamato quando:

- l'applicazione passa in background;
- lo schermo viene bloccato;
- viene avviato il salva schermo o viene attivata la modalità stand by.

QEvent::WindowActivate viene richiamato quando l'applicazione si riattiva ed esce da uno degli eventi descritti precedentemente.

Ci sono comunque altri eventi importanti che generano segnali utilizzabili per riconoscere lo stato dell'applicazione e la sua visibilità.

Segnali che ci aiutano a riconoscere se un determinato componente è visibile all'utente per evitare di mantenerlo aggiornato inutilmente:

QEvent::Show

QEvent::Hide

È utile sapere quando una finestra di dialogo ne blocca un'altra sottostante, in modo tale da interrompere l'aggiornamento di quella al momento non visibile.

QEvent::WindowUnblock

QEvent::WindowBlocked

Ecco un semplice esempio di come possono essere gestiti i suddetti segnali nel caso di una semplice animazione grafica:

```
// Viene avviata l'animazione
case QEvent::Enter: // necessaria per Maemo
case QEvent::WindowActivate:
                    timer.start(20, this);
                    text->setText("Started");
```

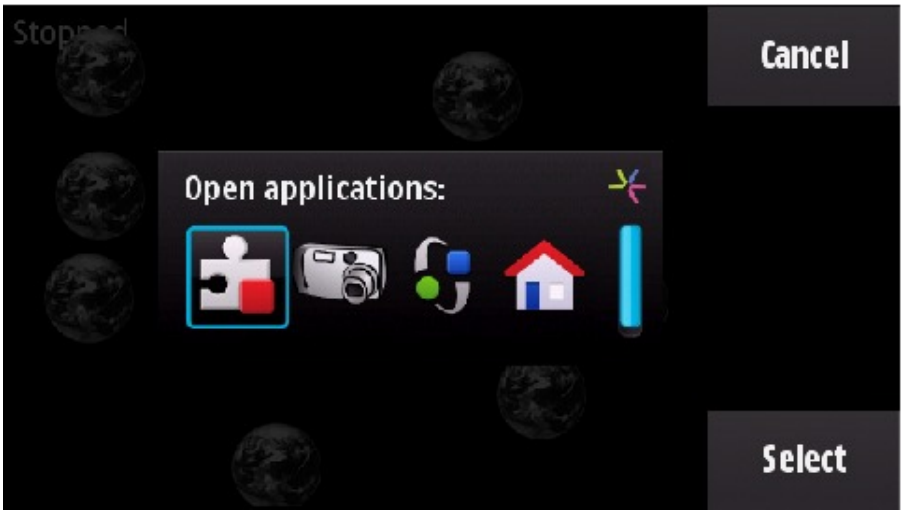

Figura 5.3: Blocco applicazione

```
                    break;

// Viene interrotta l'animazione
case QEvent::Leave: // necessaria per Maemo
case QEvent::WindowDeactivate:
                    timer.stop();
                    text->setText("Stopped");
                    break;
```

Come si può vedere dalla Figura 5.3, appena viene richiamato il gestore dei task la nostra applicazione mette in pausa l'animazione evitando così inutili sprechi di risorse. Il grafico sottostante mostra i consumi di un dispositivo dotato di accelerometro (mentre l'applicazione è in esecuzione in background) in due casi:

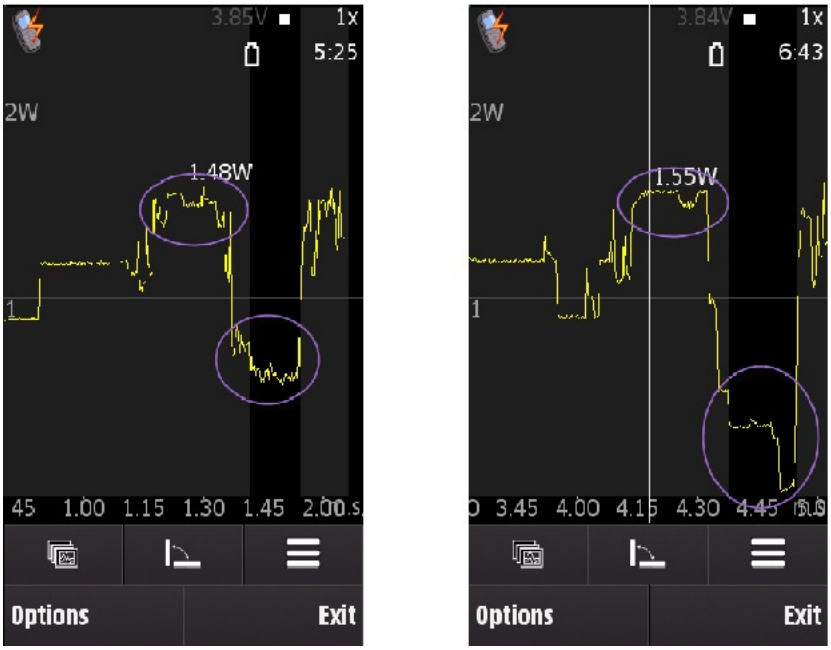

Figura 5.4: Grafico dei consumi

- nel caso in cui venga mantenuto attivo l'accelerometro;

- nel caso in cui venga invece disabilitato.

Entrambe le immagini di Figura 5.4 mostrano il consumo energetico di un'applicazione che fa uso dell'accelerometro. Il grafo mostra l'andamento dei consumi durante l'utilizzo dell'applicazione (zona grigia del grafico) e durante un periodo di inattività (zona nera del grafico). Come si può facilmente vedere, i consumi nel periodo di inattività calano drasticamente se l'hardware in questione viene disabilitato esplicitamente (grafo di destra).

5.6.2 Utilizzo di eventi

Se l'applicazione ha bisogno di gestire cambi di stato/notifiche, come per esempio la disponibilità di una connessione WiFi, si possono considerare due opzioni:

- controllare se la risorsa è disponibile attraverso un susseguirsi di richieste a tempi prefissati (questa modalità è più comunemente chiamata "polling");
- attendere un segnale apposito.

Il polling consuma cicli di CPU e di conseguenza diminuisce la vita della batteria a prescindere dall'effettivo arrivo dell'evento. Questo controllo, essendo dettato da un timer, può causare una reazione ritardata da parte del dispositivo (che può variare a seconda della frequenza in cui viene eseguito). I timer possono addirittura evitare al nostro dispositivo di entrare in modalità a basso consumo e questo può generare grosse controindicazioni che vanno al di là del consumo energetico vero e proprio. Il paradigma ad eventi corregge i principali problemi portati dall'utilizzo della soluzione con polling. Innanzitutto l'evento viene chiamato solo ed esclusivamente quando c'è un effettivo cambio di stato: i thread infatti attendono l'arrivo del segnale senza sprechi di risorse. Se tutto il codice in esecuzione su un determinato sistema gestisce in maniera corretta i segnali, il dispositivo può entrare nella modalità di risparmio energetico. Fortunatamente l'architettura Qt è orientata agli eventi e permette allo sviluppatore di implementarne dei propri,

utilizzando un comodo sistema portabile di “Signals & Slots” già discusso in precedenza.

5.6.3 Utilizzare le risorse hardware solo quando sono necessarie

L’impatto sul consumo energetico di un telefono di nuova generazione è dato dalla varietà del suo hardware e dalle sue caratteristiche. Il display, come del resto un’inferfaccia che trasmette e riceve segnali radio, incide pesantemente sui consumi. Alcune nuove caratteristiche come l’accelerometro e il magnetometro se usati in maniera inefficiente possono rendere la nostra applicazione inefficiente e quindi inutilizzabile. Non sono da scartare aspetti meno ovvi come ad esempio l’allocazione non necessaria della memoria: il dispositivo infatti deve mantenere allocati anche i banchi di memoria non utilizzati. Nella Tabella 5.1 sono elencate alcune delle caratteristiche hardware ed i rispettivi consumi:

Una batteria a grande capacità può esprimere 4.5 watt/ora e riferendoci alla Tabella 5.1 che riguarda i consumi possiamo notare che l’utilizzo della rete wireless può scaricare il dispositivo in meno di 3 ore e con il display attivo il tempo si riduce a poco più di 2 ore. Un errore ricorrente è utilizzare hardware con minore consumo come per esempio il GPS e poi aggiornare il display ogni qual volta si ha un cambiamento di stato.

Hardware	Consumo tipico (mW)
LCD display	190-360
OLED display	40-500
GPS	50
WLAN	1000-1600
Bluetooth	meno di 1mW in modalità “sniffing” 10 mW in modalità attiva
Accelerometro	n.d
Magnetometro	n.d

Tabella 5.1: Consumi Hardware

5.6.4 Ottimizzazioni

Vista la varietà di dispositivi e tecnologie utilizzate, è impossibile riuscire a gestire al meglio il consumo energetico compatibilmente con le diverse caratteristiche hardware. Sfortunatamente metodologie ottimali di gestione di uno specifico hardware, come per esempio un ricevitore GPS, possono non essere altrettanto performanti su altri dispositivi. Questo paragrafo parlerà, di conseguenza, di metodologie generiche per risparmio energetico applicabili alla maggior parte dei dispositivi mobili.

5.6.4.1 Radio ricevitori e trasmettitori

In questa famiglia sono inclusi, oltre al sistema radio classico del cellulare, WLAN e Bluetooth, anche il sistema GPS che è effettivamente un ricevitore radio ma che verrà discusso separatamente. Nell'utilizzo di questo tipo di hardware è importante tenere a mente alcune regole:

- ottimizzare le dimensioni dei dati in trasmissione, evitando di inviare dati ridondanti o non necessari;
- caching dei dati per agevolarne il riutilizzo;
- evitare protocolli che utilizzano il polling;
- utilizzare TCP invece di UDP. Particolarmente importante per le applicazioni che devono rimanere sempre attive ;
- trasferire i dati in blocco, quando possibile;
- abilitare il risparmio energetico della WLAN per evitare inutile traffico aggiuntivo.

5.6.4.2 Scrivere codice efficente

Una buona pratica di sviluppo è fondamentale per avere applicazioni efficienti da un punto di vista di consumi. Meno cicli di CPU richiedono le nostre funzioni, meno energia viene sprecata dal dispositivo. In questo paragrafo si descriveranno alcune tecniche per una corretta gestione delle risorse, con riferimento specifico al framework Qt. Qualche esempio tipico:

- scorrimento di liste ed accesso agli elementi:
  - preferire la tipologia di iteratore più efficiente. Qt rende disponibile allo sviluppatore differenti modi per scorrere una lista, come iteratori STL e la macro “foreach”. Considerare che la macro “foreach” può essere significativamente più lenta;
  - preferire Qlist::at() per l’accesso agli elementi di una lista in sola lettura. Utilizzando il metodo at() si ha un accesso più veloce rispetto a Qlist::operator[] , poichè non duplica i dati della lista;
  - usare gli iteratori di tipo costante “const iterators” quando gli elementi non debbono essere modificati. Entrambi i frammenti di codice sottostanti eseguono un iterazione completa del vettore sommando tutti i suoi elementi interni nella variabile foo (o bar, nel secondo esempio). Il primo frammento di codice utilizza un iteratore non costante:

```
QVector<qreal>::iterator it = container.begin();
while (it != container.end()) {
        foo += *it;
        ++it;
}
```

Il secondo frammento utilizza un iteratore costante:

```
QVector<qreal>::const_iterator it = container.constBegin();
while (it != container.constEnd()) {
```

```
        bar += *it;
        ++it;
}
```

Il tempo di esecuzione di entrambi i frammenti di codice cambia in maniera considerevole come illustrato nella Tabella 5.2 su più dispositivi. Il compilatore può ottimizzare in maniera più efficiente il codice prodotto quando viene utilizzato esplicitamente un iteratore costante.

Dispositivo	**const iterator**	**non-const iterator**
Nokia N97	106 ms	128 ms
Nokia N97 Mini	109 ms	129 ms
Samsung i8910	60 ms	73 ms

Tabella 5.2: Costi dei vari metodi di iterazione

- evitare di istanziare oggetti eccessivamente grandi, come per esempio immagini da un trilione (10^{12}) di pixel con profondità a 32 bit. Utilizzare una quantità eccessiva di memoria, come nel caso sopra citato, può portare a rallentamenti ed aumenti drastici di consumi, costringendo il sistema ad eseguire la paginazione della memoria più frequentemente;
- utilizzare intelligentemente la cache per gli oggetti utilizzati di frequente, come le anteprime delle immagini, i font o le informazioni sui

file. Ricalcolare o ricreare questi oggetti ad ogni loro utilizzo può essere oneroso;

- utilizzare le funzioni di tipo costante in Qt per evitare la copia dei dati.

5.6.4.3 Accesso ai File

Quando si legge o scrive su un file, è importante ridurre al minimo le operazioni di trasferimento, evitando trasferimenti di piccole quantità di dati a favore di grandi trasferimenti. Anche gli accessi lineari saranno più efficienti degli accessi casuali. Alcune raccomandazioni:

- utilizzare QFile in modalità buffered (non usare la flag unbuffered nella fuzione open()). Utilizzando la modalità buffer si riduce il numero di letture e scritture sul disco;
- non utilizzare la funzione flush(): non è necessaria durante la scrittura;
- evitare un uso frequente della seek(): in caso di bisogno si può considerare il caricamento del contenuto del file dentro un QByteArray o QBuffer .

5.6.4.4 Stati della batteria

In certi casi è utile considerare lo stato della batteria in modo che la nostra applicazione reagisca in maniera differente a seconda dei casi. Registrando il segnale QsystemDeviceInfo::batteryLevelChanged si può essere avvisati quando la batteria del dispositivo ha una carica inferiore al 3%, 10%, 40%

oppure quando la stessa supera il 40%. Se per sua natura l'applicazione ha un consumo considerevole di energia, si può considerare di avvisare l'utente dandogli la possibilità di terminarla, o semplicemente avvisarlo di utilizzare il dispositivo direttamente da rete elettrica se sotto una determinata soglia.

Capitolo 6

SmartQtVoip

Ora analizzeremo le fasi che ci hanno portato allo sviluppo dell'applicazione SmartQtVoip. Svilupperemo la scelta dell'architettura utilizzata, andandone poi ad analizzare ogni singola parte.

6.1 Architettura

L'architettura a cui si fa riferimento è a layer comunicanti.

A livello più basso della nostra architettura abbiamo sicuramente il sistema operativo, che fornisce tutti i servizi base per poter utilizzare correttamente tutte le periferiche hardware. Nel nostro caso specifico abbiamo come sistema operativo Symbian.

Sopra alla base fornita da Symbian vi è la libreria PjSip che crea un livello di astrazione molto importante. Infatti permette di gestire ad un buon livello di astrazione la creazione di connessioni multimediali basate sul protocollo

SIP. PjSip utilizza direttamente chiamate a funzione di sistema, ed in qualche caso, come nella gestione dei socket basati su SSL, fa uso di librerie di sistema.

Allo stesso livello abbiamo le librerie Qt. Queste invece, forniscono l'astrazione necessaria per rendere il più possibile portabile l'interfaccia grafica. L'intero strato è fornito da Nokia, attraverso alcuni pacchetti .sis disponibili con ogni SDK.

Infine al livello più alto abbiamo la nostra applicazione SmartQtVoip. Essa si basa completamente sul sottostrato fornito da PjSip e Qt. All'interno l'applicazione è divisa in due parti:

backend è la parte di software che fa da cuscinetto tra le funzionalità di PjSip e l'interfaccia grafica basata su Qt fornita dal frontend: egli quindi formalizza le chiamate derivanti dal frontend in chiamate a livello PjSip. D'altra parte riceve e gestisce tutte le notifiche del sottolivello, inoltrando opportunamente i cambiamenti a tutti i componenti grafici coinvolti nel frontend;

frontend è la parte terminale dell'applicazione: è quello che l'utente vede e con cui interagisce. L'interazione è permessa attraverso un'interfaccia grafica che si aggiorna in base ai cambiamenti di stato del backend. Ogni interazione grafica che implica cambiamenti alla comunicazione SIP, viene notificata al backend.

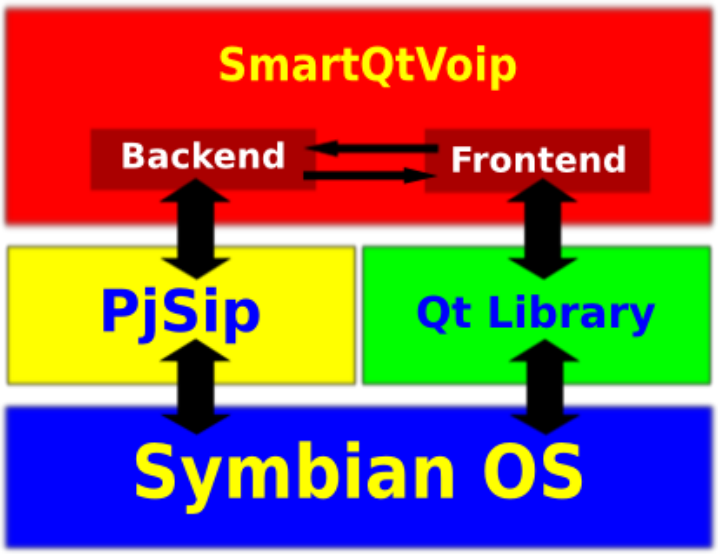

Figura 6.1: Architettura SmartQtVoip

6.2 Strumenti di sviluppo

6.2.1 SDKs utilizzati

Fino a poco tempo fa uscivano SDK specifici per Symbian basati sulla piattaforma di sviluppo Carbide C++.

Carbide C++ è basato sulle ultime versioni di Eclipse IDE ed Eclipse CDT estesa con caratteristiche specifiche per Symbian. Attualmente supporta le architetture x86 WINSCW compilatore C++ per la produzione di emulatore. Il WINSCW è un compilatore basato su GCC effettivamente fornito con l'SDK, ma non esplicitamente incluso nell'IDE Carbide. Carbide C++ supporta il Symbian Build System (SBS). Sono disponibili due versioni SBSv1 e SBSv2. Il primo è un sistema basato su Perl mentre il secondo sul

linguaggio Python compatibile con la nuova generazione di sistemi operativi Symbian OS. Il vantaggio principale di SBSv1 e SBSv2 è che possono fare il building dei progetti da riga di comando in parallelo all'IDE. SBSv1 è in via di abbandono perchè effettua il controllo delle dipendenze in via automatica ad ogni build e rebuild, e quindi può impiegare diverso tempo in progetti di grandi dimensioni.

Come possiamo vedere dalla Tabella 6.1 esistono diversi SDK compatibili con Carbide C++ per le varie versioni di Symbian:

SDK
Symbianˆ3
Nokia N97 mobile
S60 5th Edition
S60 3rd Edition, Feature Pack 2

Tabella 6.1: Lista SDK compatibili con Carbide C++

Carbide C++ ha raggiunto la versione 2.7 che risulta essere abbastanza stabile, molto di più rispetto alle versioni precedenti. Per poter sviluppare applicazioni Symbianˆ3 è necessario avere quest'ultima versione, mentre per tutte le altre versioni è possibile utilizzare anche versioni precedenti di Carbide, anche se lo sconsiglio vivamente.

Il testing delle applicazioni può essere svolto senza dover farne il deployment sul device. L'emulatore permette di avviare una macchina virtuale con

tutte le funzionalità di un device reale. L'inconveniente di tutto ciò, che non è da trascurare, è il tempo di avvio dell'emulatore, che richiede diversi minuti e a volte risulta anche instabile.

Carbide C++ seppur basato sull'IDE Eclipse (disponibile anche in ambiente Unix like) esiste solo per sistemi operativi Windows.

Alcuni SDK sono stati inizialmente utilizzati per provare le funzionalità di Carbide, ma questi sono risultati un po' datati rispetto alle versioni Symbian che si trovavno in commercio. Per tutto ciò abbiamo deciso di utilizzare il meno possibile lo strumento di sviluppo Carbide con i suoi SDK.

6.2.1.1 Nokia Qt SDKs

Come lo è stato per noi, anche la Nokia si è accorta che le fasi di sviluppo di un qualsiasi software per il suo sistema non aveva tempi ristretti di apprendimento. Quindi in qualità di proprietaria della libreria Qt, Nokia ha deciso di utilizzare questa per cambiare l'ambiente di sviluppo addottando Qt Creator. Qt Creator oltretutto è multipiattaforma, disponibile quindi, per vari sistemi operativi (es. Windows, Linux, Mac).

Durante il nostro lavoro sono state testate due versioni del Nokia Qt SDK:

- Nokia Qt SDK 1.0 (versione Stable);
- Nokia Qt SDK 1.1 (versione Beta).

Il primo SDK è il predefinito per il rilascio di applicazioni commerciali o gratuite da pubblicare in Ovi Store (portale Nokia delle applicazioni di

Symbian). Fa uso della librerie Qt 4.6, ma ha un supporto preliminare alle librerie Qt 4.7. Il secondo SDK è ancora in stato di beta ma già decisamente funzionale. Quest'ultimo dà pieno supporto alle Qt 4.7.2 e alla nuova tecnologia QtQuick. In entrambi gli SDKs sono supportati tutti i framework che compongono le QtMobility.

6.2.1.2 Impostazione Carbide C++ per utilizzo dei Nokia SDKs

Per avere a disposizione le librerie PjSip, e poterne poi fare il linking col nostro progetto realizzato in ambiente QtCreator, è stato necessario trovare il modo di poter compilare il PjProject con Carbide C++ abilitato ad usare i compilatori interni ai Nokia Qt SDK. Quindi sono stati impostati manualmente tutti i Nokia Qt SDK che avevamo a disposione. Questo è stato necessario perchè come vedremo più avanti le PjSip sono strutture per la compilazione attraverso delle build gestite da Carbide. Inoltre le librerie così ottenute, sono risultate più piccole nelle dimensioni ed efficienti per via dei compilatori di ultima generazione.

6.2.2 PjSip in SmartQtVoip

Il progetto si è basato sull'ultima versione stabile di PjSip che fa riferimento alla versione 1.8.10 del pacchetto PjProject.

6.2.2.1 La compilazione

Il pacchetto PjProject segue varie vie di compilazione a seconda del sistema operativo su cui si andrà a compilare la libreria.

Prima di compilare PjSip, su una qualsiasi piattaforma, vanno impostate delle macro. Queste macro vanno definite in un file "pjproject/pjlib/include/pj/config_site.h" da creare. Per semplificare il tutto esiste già un file contenente delle macro di compilazione "pjproject/pjlib/include/config_site_sample.h". Quindi nel file "config_site.h" si può inserire l'include del file di esempio:

```
#include <pj/config_site_sample.h>
```

In ambienti Unix like (es. Linux, MacOS X 10.2, mingw, FreeBSD), la compilazione è semplice ed immediata in quanto gestita attraverso il tool Autoconf. La sequenza dei comandi da seguire nella cartella root della libreria sarà quindi la seguente:

```
$ ./configure
$ make dep
$ make clean
$ make
```

In ambiente Windows e Windows Mobile, la compilazione viene gestita interamente dalla piattaforma di sviluppo Visual Studio. Quindi si può procedere aprendo il file di progetto con estensione .dsw o .sln a seconda dell'IDE a disposizione.

Il nostro caso riguarda però il sistema operativo Symbian, che è gestito in maniera completamente diversa. I files di build sono all'interno della cartella "pjproject/build.symbian". I files di progetto sono organizzati attraverso tanti files con estensione .mmp. Ogni .mmp file rappresenta una libreria o un'applicazione. La lista dei .mmp files che generano dei file di libreria con estensione .lib sono i seguenti:

pjlib.mmp : piattaforma di astrazione;

pjlib-util.mmp : per funzioni di testo, crittografia, DNS, XML, etc...

pjnath.mmp : per STUN, TURN, ICE, etc...

pjsdp.mmp : parte di pjmedia relativo a SDP;

pjmedia.mmp : framework per i media;

pjsip.mmp : nocciolo dello stack SIP;

pjsip_ua.mmp : funzionalità SIP per lo user agent;

pjsip-simple.mmp : modulo SIP SIMPLE per la presenza, instant messaging, etc...

pjsua_lib.mmp : libreria di alto livello per lo user agent che integra SIP, media e NAT;

libsrtp.mmp : libreria per il protocollo SRTP.

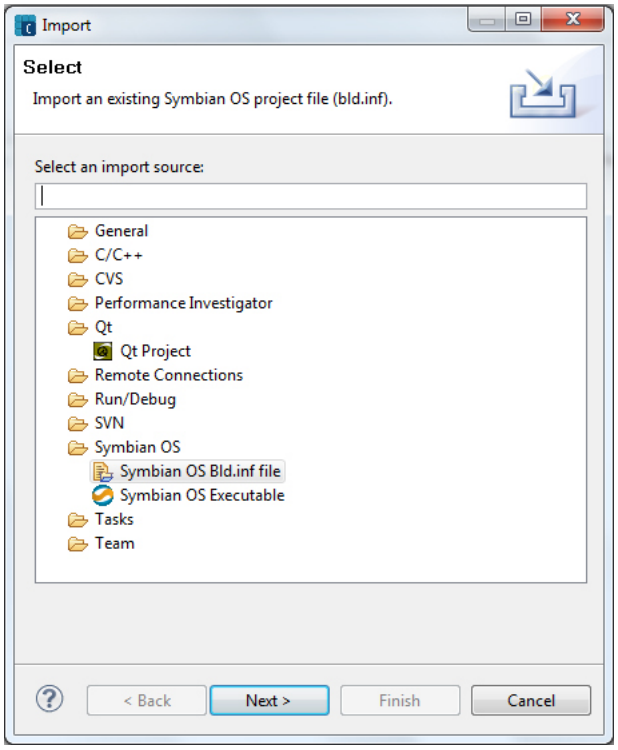

Figura 6.2: Carbide C++ 2.7 - Importazione PjProject 1.8.10 - Step 1

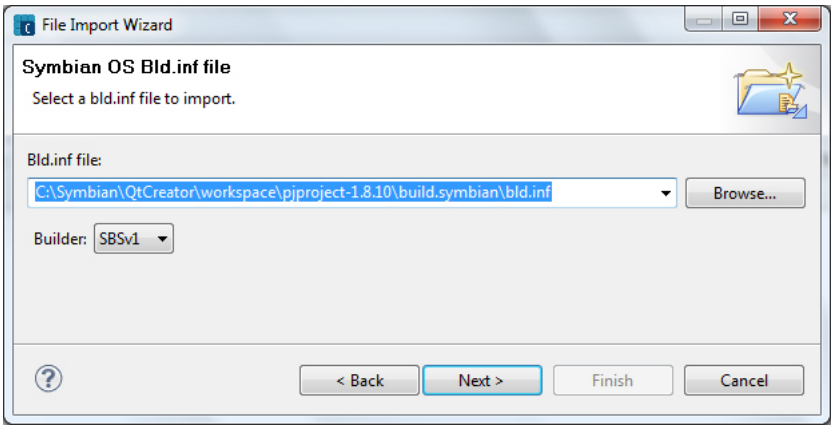

Figura 6.3: Carbide C++ 2.7 - Importazione PjProject 1.8.10 - Step 2

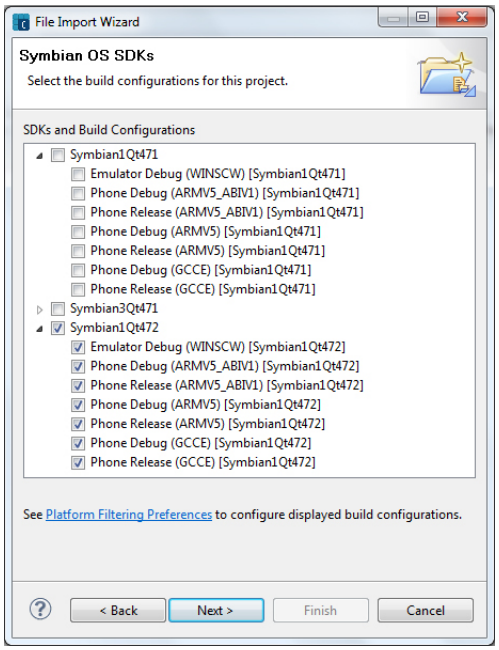

Figura 6.4: Carbide C++ 2.7 - Importazione PjProject 1.8.10 - Step 3

Esistono altri .mmp che si riferiscono ai codec, al campionamento, ai dispositivi audio e ad applicazioni di esempio.

Tutti questi file sono organizzati e listati all'interno del file "bld.inf" che dà modo al compilatore di sapere quali librerie e applicazioni compilare. Ora si procede facendo l'import del progetto con Carbide C++, caricando il file "pjproject/buil.symbian/bld.inf". Durante l'importazione del progetto si potranno selezionare quali file .mmp rendere attivi nella compilazione e quali SDK utlizzare per la compilazione. Dopo l'import, per ogni SDK impostato, si potranno avere tre modalità di compilazione/esecuzione:

Debug: la modalità con tutte le informazioni necessarie per il debug;

Release: la modalità finale con tutte le ottimizzazioni possibili disposte dal

compilatore;

Emulator: la modalità che consente di poter eseguire le applicazioni o le librerie senza dover effettuare il deployment su di un device specifico, ma all'interno di un emulatore creato ad-hoc.

Ora non ci resta che compilare le librerie: selezioneremo la modalità release per ridurre al massimo le dimensioni di queste ed avere prestazioni massimali. Le librerie ottenute con estensione .lib verranno automaticamente copiate nella cartella predefinita per l'SDK correntemente utilizzato (es. "/SDKs/Symbian1Qt472/ epoc32/release/armv5/udeb).

6.2.2.2 Problematiche della compilazione

La compilazione segnala diversi warning e forse qualche errore, ma a questi ultimi si riesce ad ovviare facilmente con qualche accorgimento.

6.2.2.3 Impostazione del Project File di Qt

Il file di progetto di SmartQtVoip (SmartQtVoip.pro) risulta fondamentale per una corretto linking dopo la fase di compilazione. La cosa più importante di questo file, processato da qMake, è sicuramente la sezione che gestisce il path per gli include delle librerie di PjProject. Il file presenta diversi tipi di compilazione uno per ogni sistema operativo considerato (Symbian, Windows e Linux). Nella sezione Symbian si sono gestiti i nomi delle librerie base utilizzate da PjProject e le macro necessarie.

6.3 Backend basato su PjSip

Il backend, come mostrato nella figura rappresentante l'architettura di SmartQtVoip, comunica direttamente con il sistema operativo Symbian. Si è già visto come generare le librerie PjSip: ora vediamo come queste vengono utilizzate all'interno del backend di SmartQtVoip.

6.3.1 L'interfaccia Qt per PjSip

Per potersi interfacciare correttamente con PjSip è stata creata una classe ad hoc all'interno del nostro progetto. La classe in questione è PjCallback definita all'interno del file "PjCallback.h". Questa effettua una wrapper di tutte le funzioni che possono essere chiamate a PjSip, cioè praticamente converte tutti i segnali che giungono da PjSip in segnali Qt. Vediamo come è stato implementato questo wrapper di segnali:

```
....
....
PjCallback();

    virtual ~PjCallback();

    /* Callback logger function which emmits the signal */
    void logger_cb(int level, const char *data, int len);
    /* Callback logger function called by PjSip */
```

```
static void logger_cb_wrapper(int level,
                              const char *data,
                              int len);

void on_pager(pjsua_call_id call_id,
              const pj_str_t *from,
              const pj_str_t *to,
              const pj_str_t *contact,
              const pj_str_t *mime_type,
              const pj_str_t *text);
static void on_pager_wrapper(pjsua_call_id call_id,
                             const pj_str_t *from,
                              const pj_str_t *to,
                              const pj_str_t *contact,
                              const pj_str_t *mime_type,
                              const pj_str_t *text);

void on_call_state(pjsua_call_id call_id,
                   pjsip_event *e);
static void on_call_state_wrapper
      ( pjsua_call_id call_id,
        pjsip_event *e);
```

```
void on_incoming_call(pjsua_acc_id acc_id,
                      pjsua_call_id call_id,
                      pjsip_rx_data *rdata);
static void on_incoming_call_wrapper
      ( pjsua_acc_id acc_id,
        pjsua_call_id call_id,
        pjsip_rx_data *rdata);

void on_nat_detect
      (const pj_stun_nat_detect_result *res);
static void on_nat_detect_wrapper
      (const pj_stun_nat_detect_result *res);

/** Notify application when media state
  * in the call has changed. */
void on_call_media_state(pjsua_call_id call_id);
static void on_call_media_state_wrapper
      (pjsua_call_id call_id);

/** Callback function, called by wrapper */
void on_buddy_state(pjsua_buddy_id buddy_id);
```

```
    /** Callback wrapper function called by PjSip
      * Presence state of buddy was changed */
    static void on_buddy_state_wrapper
          (pjsua_buddy_id buddy_id);

    /** Callback function, called by wrapper */
    void on_reg_state(pjsua_acc_id acc_id);
    /** Callback wrapper function called by PjSip
      * Registration state of account changed */
    static void on_reg_state_wrapper(pjsua_acc_id acc_id);



signals:
    /** this signal forwards the log message
      *a-synchronous to the GUI thread */
    void new_log_message(QString text);
    /** this signal forwards the instant message
      * a-synchronous to the GUI thread */
    void new_im(QString from, QString text);
    /** this signal forwards the instant message
      * a-synchronous to the GUI thread */
```

```
    void nat_detect(QString text, QString description);
    /** this signal forwards the call state synchronous
      * to the GUI thread */
    void call_state_sync(int call_id);
    /** this signal forwards the call state
      * a-synchronous BLOCKING to the GUI thread */
    void setCallState(QString);
    /** this signal forwards the text of the call
      * button a-synchronous BLOCKING to the GUI thread */
    void setCallButtonText(QString);
    /** this signal forwards the buddy_id of the
      * buddy whose status changed to the GUI thread */
    void buddy_state(int buddy_id);
    /** this signal forwards the acc_id of the SIP
      * account whose registration status changed */
    void reg_state_signal(int acc_id);
};
....
....
```

I metodi pubblici con la parola wrapper nel nome del metodo effettuano delle chiamate alle funzioni C propriamente di PjSip. I metodi pubblici senza la parola wrapper, sono quelli che andranno a ricevere dati dal sottostrato

PjSip inoltrando i dati sotto forma di segnali al frontend.

6.4 Frontend basato su Qt C++, QtQuick e QtMobility

Nel frontend si è fatto l'utilizzo della tecnologia QtQuick per realizzare tutta l'interfaccia grafica. Vengono gestiti di questa gli eventi attraverso un substrato C++, il quale a sua volta invia notifiche sotto forma di segnali. La gestione dei contatti è stata realizzata utilizzando il framework QtMobility.

6.4.1 Interfaccia utente

L'interfaccia è stata resa il più possibile logica e chiara dividendola per aree di interesse. Sono stati individuati in cinque blocchi:

- home;
- impostazioni SIP;
- gestione contatti;
- gestione chiamate vocali;
- gestione dei messaggi istantanei.

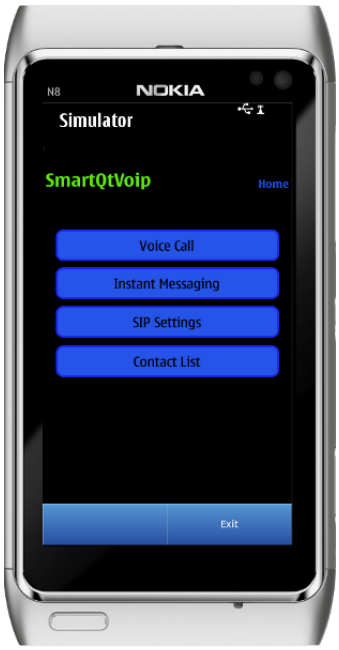

Figura 6.5: SmartQtVoip - Sezione Home

Figura 6.6: SmartQtVoip - Sezione Voice Call

Figura 6.7: SmartQtVoip - Sezione Instant Messaging

Figura 6.8: SmartQtVoip - Sezione Voip Setting

Figura 6.9: SmartQtVoip - Sezione Contatti

6.4.2 Gestione dei contatti

La gestione della rubrica dei contatti è stata eseguita attraverso il framework QtMobility. Questo ci ha permesso di utilizzare la rubrica di Symbian per la memorizzazione e gestione degli indrizzi SIP.

Ogni volta che l'applicazione viene avviata si esegue una scansione della rubrica, filtrandola per i contatti che hanno tra le informazioni memorizzate l'indirizzo SIP. Quando invece si andrà ad aggiungere o ad eliminare un nuovo contatto SIP, non si farà altro che aggiornare l'opportuno campo del contatto della rubrica in question.

L'utilizzo delle QtMobility ci ha permesso di non dover implementare un doppione della rubrica già presente in tutti i cellulari dotati di Symbian.

Si è in sintesi utilizzata l'informazione SIP uri come uno dei tanti campi informativi associati ai nostri contatti.

Capitolo 7

Limiti attuali di SmartQtVoip

I limiti attuali dell'applicazione sviluppata sono tutti principalmente dovuti ai problemi che vi sono nello strato inferiore dell'architettura che abbiamo adottato. Ogni qual volta vengono aggiornate le funzionalità del sottostrato fornito da PjSip e Qt, queste risultano immediatamente disponibili al livello più alto di sviluppo della nostra applicazione.

7.1 Limiti di PjSip

PjSip è molto flessibile, forse anche troppo. Il porting esistente di PjSip per Symbian, non è aggiornato alle ultime specifiche Symbianˆ1 e Symbianˆ3. Durante la compilazione con Carbide si hanno diversi warning che non vanno trascurati. Inoltre ci sono delle funzioni deprecate che vanno sostituite con le nuove adottate da Nokia. Ma sembra che la prossima release, la 2.0 prevista

per inizio 2012, abbia tra i vari task un porting completo all'ultima versione Symbianˆ3.

La gestione dell'audio non è ancora ottimale, presentando problemi di eco durante la comunicazione. Comunque anche per questo si è controllato nel sito di riferimento PjSip: si sta cercando di risolvere questo problema.

Un altro problema è il dover gestire le librerie di PjSip in maniera statica. Queste aumentano di molto la grandezza del nostro file "SmartQtVoip.sis".

7.2 Limiti di Qt

Qt rispetto a PjSip ha dato molti meno problemi di stabilità. Si è mostrato un framework di sviluppo molto stabile ed affermato. Purtroppo siccome molti progetti vecchi sono stati sviluppati in Carbide, non è così immediato effettuare il porting delle applicazioni. Infatti quando si effettua la compilazione del progetto, Qt Creator crea automaticamente il file di progetto .mmp che veniva utilizzato da Carbide per il building dell'applicazione. Ma molti progetti sviluppati in Carbide, come nel nostro caso PjSip, presentano delle personalizzazioni dei files .mmp per le compilazioni. Qt è riuscita a mettere una pezza a questo problema utilizzando delle macro apposite da inserire nel file di progetto .pro che permettono di aggiungere del testo nei file .mmp autogenerati.

Qt sebbene disponibile per Windows, Linux e Mac non ha per tutte e tre le versioni le stesse funzionalità. Infatti solo per Windows esistono SDK che

permettono la compilazione ed il conseguente deployment dell'applicazione. Inizialmente Qt forniva solo un simulatore per sopperire alla mancanza della compilazione, ma nelle ultime release di Nokia Qt SDK è disponibile la funzionalità Remote Compiler che permette, previa registrazione al sito di Nokia, di poter inviare i nostri sorgenti ad un server remoto che ne farà la compilazione e ci ritornerà il pacchetto .sis pronto per essere installato. Questa funzionalità è molto utile quando si fanno progetti che utilizzano solamente le librerie predefinite di Qt. Siccome nel nostro caso facciamo uso di librerie esterne a Qt, questa strada non è percorribile in ugualmente.

QtQuick si è mostrato davvero molto utile per lo sviluppo dell'interfaccia, con un grado di personalizzazione di questa molto elevato. Fino ad ora però esistono pochi widget immediatamente utilizzabili, che forniscono delle funzionalità base. L'integrazione tra loro di questi widget permette di ottenerne di nuovi molto funzionali, ma richiede comunque tempo di sviluppo.

Conclusioni

Nel capitolo precedente abbiamo trattato come ci siano stati molti ostacoli da superare per lo sviluppo di SmartQtVoip. Abbiamo raggiunto un risultato accettabile anche se si potrebbe fare molto di più.

La spinta fondamentale sarà data dal porting della comunità di PjProject a Symbianˆ3, che permetterà di avere meno problemi durante la compilazione delle librerie con i nuovi SDKs. Non si sa come questo porting verrà fatto, ma sicuramente se fosse gestito attravero un file di progetto Qt (file .pro), quindi da qMake, sarebbe un grande passo in avanti per abbandonare definitivamente la piattaforma di sviluppo Carbide. Per rendere le librerie indipendenti dal software che le utilizza sarebbe necessario poterle compilare ed installare come pacchetti distinti (files .sis) come avviene per le librerie Qt in Symbian.

Tornando a parlare di QtCreator, per aumentare il bacino possibile di sviluppatori, e mantenere testa agli altri sistemi operativi (es. Android di Google), sarà necessario avere SDKs che permettano la compilazione delle applicazioni anche in ambienti diversi da Windows. Per quanto riguarda le

QtMobility, si sono già rivelate ad un buon livello di affidabilità, garantendo funzionalità ad un alto livello di astrazione. Sarebbe utile espandere il loro utilizzo per quanto possibile all'interno del nostro software. QtQuick è ancora molto giovane, presentando ancora pochi widget pronti all'utilizzo, ed andando a guardare la lista di bug individuati si nota che sono stati molti, alcuni già risolti ed altri ancora da risolvere. Speriamo che Nokia investa ulteriormente su questa tecnologia, perchè ci è sembrata la via migliore per far tornare competitivo il mercato dello sviluppo Symbian.

SmartQtVoip presenta ancora dei problemi di stabilità: ne va fatto quindi un test accurato e sistematico per individuare più casi di errore possibile. Inoltre nella roadmap di sviluppo di PjProject è previsto il supporto video: si può quindi già pensare a sviluppare una sesta area di interesse che riguardi le chiamate video, utilizzando il più possibile le funzionalità dei frameworks che compongono le QtMobility.

Bibliografia

[1] Voice Over IP, http://en.wikipedia.org/wiki/Voice_over_IP

[2] Summerfield Blanchette, C++ Gui Programming with Qt 4 , 2006.

[3] Symbian Developer Wiki, Creating Energy Efficient Apps Using Qt. http://developer.symbian.org/wiki/Apps:Creating_Energy_Efficient_Apps_Using_Qt

[4] PJSIP Online-Documentation, http://www.pjsip.org/docs.htm

[5] PJSIP Development Wiki, http://trac.pjsip.org/repos/

[6] Introduction to Session Initiation Protocol (SIP) - A Beginners' Made Easy Tutorial, http://www.siptutorial.net/index.html

[7] Speex codec comparison, http://www.speex.org/comparison/

[8] Qt Online Documentation, http://doc.qt.nokia.com/