ALMA MATER STUDIORUM
UNIVERSITÀ DEGLI STUDI DI BOLOGNA

Scuola di Ingegneria e Architettura di Cesena
Corso di Laurea in Ingegneria Informatica

LIBRERIA PER COMUNICAZIONI AD-HOC PER ANDROID BASATE SU BLUETOOTH

Elaborata nel corso di: Fondamenti di Informatica A

Tesi di Laurea di:
LUCA SANTONASTASI

Relatore:
Prof. MIRKO VIROLI

Co-relatori:
Prof. ALESSANDRO RICCI

ANNO ACCADEMICO 2013-2014
SESSIONE II

PAROLE CHIAVE

Bluetooth

Android

Peer 2 Peer

Comunicazioni Ad-Hoc

Don't waste your time or time will waste you..

Indice

Introduzione

La tesi sviluppa, attraverso un processo definito e gestito, una libreria Android che permette di far comunicare diversi dispositivi mobili tramite Bluetooth. Inoltre gestisce:

- le **richieste di connessione** sia che esse provengano dall'esterno, sia che partano dal dispositivo stesso;
- la **dinamicità del sistema** utilizzando dispositivi mobili, cioè capaci di muoversi in diverse direzioni costantemente. In tal senso, la lista dei dispositivi vicini è costantemente aggiornata.
- la **comunicazione in stile Peer-to-Peer(P2P)**. In questo caso si viene a formare un gap in quanto la libreria Android di Bluetooth usa lo stile *Master-Slave* per i suoi dispositivi. Per colmare questo gap, la libreria sviluppata crea un layer soprastante la libreria Bluetooth di Android e maschera lo stile di comunicazione di quest'ultima a favore di una comunicazione paritaria, nella quale un dispositivo può sia accettare le richieste di connessione, sia connettersi ad altri dispositivi.

Prima di sviluppare la libreria, ci si è concentrati sulla spiegazione e descrizione dei punti cardine del progetto. Nel capitolo 1, chiamato **Background**, si sono descritte le parole chiave utilizzate nel progetto. Successivamente si è iniziato con *Bluetooth*, in cui si è analizzata la storia e le caratteristiche tecniche della tecnologia. Successivamente si è parlato di *Android*, descrivendone i punti cardine che definiscono questo sistema operativo, come ad esempio la sua storia e la sua architettura di sistema. Di seguito si è discusso il *P2P*, descrivendone le applicazioni, il funzionamento, i vantaggi e gli svantaggi di tale tecnologia e infine citando le funzionalità

che derivano da questo tipo di architettura logica di rete. Per concludere il capitolo si è introdotto il discorso delle *reti ad-hoc*, indicando anche qui le applicazioni, i requisiti tecnici e citando un esempio come le MANET.

Una volta definiti i concetti base, si è passati nel secondo capitolo alla descrizione del **Processo di sviluppo**, seguendo la metodologia classica insegnata nel corso di Ingegneria del Software, che ha portato a un primo prototipo funzionante della libreria. Il primo passo riguarda la trascrizione dei requisiti che la libreria deve rispettare. In seguito si è passati all'*Analisi dei requisiti* dove per disambiguare alcuni concetti descritti nei *Requisiti*, è stato definito un *Glossario*. In seguito sono stati definiti i *Casi d'uso* della libreria, seguiti dagli *Scenari* di ognuno di questi. Come ultimo passo di questa prima analisi si è definito un primo *modello del dominio*. Il passo successivo è l'*Analisi del problema* in cui è stata analizzata l'*Architettura logica* del sistema, definita dal trittico *Struttura*, *Interazione* e *Comportamento*. Successivamente si è puntualizzato l'*Abstraction Gap* del sistema insieme ai punti critici determinati nell'*Analisi dei rischi*, concludendo la fase di analisi con la definizione del *Piano di lavoro*. La fase successiva è quella del *Progetto*, in cui si è presa l'intera architettura logica definita nell'analisi e si è cercato di dare una soluzione al problema analizzato. Infine si è passati all'ultima fase, cioè *Implementazione* in cui si sono mostrati alcuni dei passaggi critici descritti nella fase di progetto e rappresentati attraverso il codice.

Concluso il secondo capitolo, si è ottenuto un primo prototipo funzionante della libreria da cui si possono creare alcuni **Scenari applicativi** descritti nel terzo capitolo. Data questa libreria, si è implementato un progetto denominato *ChatBluetooth*, ossia un sistema di messaggistica basata su Bluetooth che ricerca i dispositivi vicini aventi tale applicazione. Questi sono mostrati tramite una lista ordinata secondo distanza dal nostro dispositivo in maniera crescente (ossia prima quelli più vicini). Questa lista oltre a essere ordinata è anche dinamica in quanto è costantemente in discovery e se un dispositivo entra o esce dal range, viene segnalato. Inoltre permette sia l'accettazione delle richieste di connessione da altri dispositivi, sia la connessione verso altri dispositivi vicini. I messaggi ricevuti vengono segnalati e salvati in un database locale interno all'applicazione stessa. Una volta conclusa la discussione dell'applicazione, si è passati a definire degli

scenari che riguardano da vicino la libreria o che potrebbero riguardarla per il futuro. Il primo è uno scenario reale ossia *FireChat*, un'applicazione di messaggistica basata su Bluetooth che funziona anche senza rete Wi-Fi o senza rete cellulare. Gli altri sono possibili scenari futuri il cui sviluppo si può basare semplicemente sulla libreria.

Infine sono state descritte le conclusioni relative alla libreria progettata e all'applicazione *ChatBluetooth* con tutti i dettagli in modo da fornire una visione globale di quanto inserito e descritto nell'elaborato.

Capitolo 1

Background

Lo sviluppo del progetto é stato basato su una serie di tecnologie senza cui sarebbe stato impossibile realizzare la libreria che sono:

- Bluetooth
- Android

1.1 Bluetooth

Figura 1.1: Logo Bluetooth

Bluetooth é uno standard tecnico industriale di trasmissione dati per reti personali senza fili (WPAN: Wireless Personal Area Network). Fornisce un metodo standard, economico e sicuro per scambiare informazioni tra dispositivi diversi attraverso una frequenza radio sicura a corto raggio.

Bluetooth (a volte abbreviato in BT) cerca i dispositivi coperti dal segnale radio entro un raggio di qualche decina di metri mettendoli in comunicazione tra loro. Questi dispositivi possono essere ad esempio palmari,

telefoni cellulari, personal computer, portatili, stampanti, fotocamere digitali, console per videogiochi purché provvisti delle specifiche hardware e software richieste dallo standard stesso.

La specifica Bluetooth é stata sviluppata da Ericsson e in seguito formalizzata dalla Bluetooth Special Interest Group (SIG). La SIG, la cui costituzione é stata formalmente annunciata il 20 maggio 1999, é un'associazione formata da Sony Ericsson, IBM, Intel, Toshiba, Nokia e altre societá che si sono aggiunte come associate o come membri aggiunti.

Il nome é ispirato a Harald Blatand (Harold Bluetooth in inglese), re Aroldo I di Danimarca (901 - 985 o 986), abile diplomatico che uní gli scandinavi introducendo nella regione il cristianesimo. Gli inventori della tecnologia devono aver ritenuto che fosse un nome adatto per un protocollo capace di mettere in comunicazione dispositivi diversi (così come il re unì i popoli della penisola scandinava con la religione).

Il logo della tecnologia unisce infatti le rune nordiche Hagall e Berkanan, analoghe alle moderne H e B. È probabile che l'Harald Blåtand a cui si deve l'ispirazione sia quello ritratto nel libro *The Long Ships* di Frans Gunnar Bengtsson, un best-seller svedese ispirato alla storia vichinga.

Questo standard è stato progettato con l'obiettivo primario di ottenere bassi consumi, un corto raggio d'azione (fino a 100 metri di copertura per un dispositivo di Classe 1 e fino ad un metro per dispositivi di Classe 3) e un basso costo di produzione per i dispositivi compatibili.

Lo standard doveva consentire il collegamento wireless tra periferiche come stampanti, tastiere, telefoni, microfoni, ecc. a computer o PDA o tra PDA e PDA. Attualmente più di un miliardo di dispositivi montano un'interfaccia Bluetooth.[3]

I telefoni cellulari che integrano chip Bluetooth sono venduti in milioni di esemplari e sono abilitati a riconoscere e utilizzare periferiche Bluetooth in modo da svincolarsi dai classici cavi in rame.

Il protocollo Bluetooth lavora nelle frequenze libere di 2,45 GHz. Per ridurre le interferenze il protocollo divide la banda in 79 canali e provvede a commutare tra i vari canali 1.600 volte al secondo (frequency hopping).

La versione 1.1 e 1.2 del Bluetooth gestisce velocità di trasferimento fino a 723,1 kbit/s. La versione 2.0 gestisce una modalità ad alta velocità che consente fino a 3 Mbit/s. La versione 4.0 arriva ad una velocita di 24 mbps (3 MB al secondo). Questa modalità però aumenta la potenza assorbita.

La nuova versione utilizza segnali più brevi, e quindi riesce a dimezzare la potenza richiesta rispetto al Bluetooth 1.2 (a parità di traffico inviato).

Bluetooth non è uno standard comparabile con il Wi-Fi, dato che questo è un protocollo nato per fornire elevate velocità di trasmissione con un raggio maggiore, a costo di una maggior potenza dissipata e di un hardware molto più costoso. Infatti il Bluetooth crea una personal area network (PAN) mentre il Wi-FI crea una local area network. Il Bluetooth può essere paragonato al bus USB mentre il Wi-FI può essere paragonato allo standard ethernet.

Ad ogni modo lo standard Bluetooth include anche comunicazioni a lunga distanza tra dispositivi per realizzare delle LAN wireless. Ogni dispositivo Bluetooth è in grado di gestire simultaneamente la comunicazione con altri 7 dispositivi sebbene, essendo un collegamento di tipo master-slave, solo un dispositivo per volta può comunicare con il server. Questa rete minimale viene chiamata piconet. Le specifiche Bluetooth consentono di collegare due piconet in modo da espandere la rete. Tale rete viene chiamata scatternet. Ogni dispositivo Bluetooth è configurabile per cercare costantemente altri dispositivi e per collegarsi a questi. Può essere impostata una password per motivi di sicurezza se lo si ritiene necessario.

I dispositivi dotati di Bluetooth si dividono in 3 classi:

Classe	**Potenza (mW)**	**Potenza (dBm)**	**Distanza (m)**
Classe 1	100	20	100
Classe 2	2,5	4	10
Classe 3	1	0	1

1.1.1 Caratteristiche tecniche

Temporizzazione e clock

La tecnologia Bluetooth prevede di sincronizzare la maggior parte delle operazioni con un segnale di clock in tempo reale. Esso serve, ad esempio, a sincronizzare gli scambi di dati tra i dispositivi, distinguere tra pacchetti ritrasmessi o persi, generare una sequenza pseudo-casuale predicibile e riproducibile. Il clock Bluetooth è realizzato con un contatore a 28 bit che viene posto a 0 all'accensione del dispositivo e subito dopo continua senza fermarsi mai, incrementandosi ogni 312,5 microsec (metà slot quindi). Il ciclo del contatore copre approssimativamente la durata di un giorno (312,5 microsec x 228 = 23,3 ore).

Ogni dispositivo Bluetooth ha il suo native clock (CLKN) che controlla la temporizzazione di quel dispositivo. Oltre a questo valore, proprio di ogni dispositivo, Bluetooth definisce altri due clock:

- CLK: questo è il clock della piconet, coincide con il CLKN dell'unità master della piconet. Tutte le unità attive nella piconet devono sincronizzare il proprio CLKN con il CLK. La sincronizzazione avviene aggiungendo un offset al CLKN dello slave per farlo coincidere con il CLK della piconet.

- CLKE: anche questo clock è derivato tramite un offset dal CLKN ed è usato dal master nel caso specifico della creazione di una connessione verso uno slave, e prima che tale slave si sia sincronizzato con il master (ovvero quando si tratta di un nuovo slave).

I primi 2 bit del contatore vengono usati direttamente per delimitare gli slot e i cosiddetti mezzi slot, per la trasmissione e ricezione dei pacchetti; essi servono anche a stabilire nel tempo gli slot Tx (trasmissione) o Rx (ricezione) a seconda se il dispositivo in questione stia funzionando da master o da slave. Una trasmissione da parte del master comincerà sempre quando CLK[1:0] = 00 (slot di indice pari), mentre una trasmissione da parte di uno slave comincerà sempre quando CLK[1:0] = 10 (slot di indice dispari).

Connessioni

I collegamenti che possono essere stabiliti tra i diversi dispositivi sono di due tipi: orientati alla connessione e senza connessione. Un collegamento orientato alla connessione richiede di stabilire una connessione tra i dispositivi prima di inviare i dati; mentre, un link senza connessione non richiede alcuna connessione prima di inviare i pacchetti. Il trasmettitore può in qualsiasi momento iniziare ad inviare i propri pacchetti purché conosca l'indirizzo del destinatario. La tecnologia Bluetooth definisce due tipi di collegamenti a supporto delle applicazioni voce e trasferimento dati: un servizio asincrono senza connessione (ACL, Asynchronous ConnectionLess) ed un servizio sincrono orientato alla connessione (SCO, Synchronous Connection Oriented).

ACL supporta traffico di tipo dati e si basa su un servizio di tipo best-effort. L'informazione trasportata può essere di tipo utente o di controllo. SCO, invece, è un collegamento che supporta connessioni con un traffico di

tipo real-time e multimediale. Il collegamento ACL supporta connessioni a commutazione di pacchetto, connessioni punto-multipunto e connessioni simmetriche o asimmetriche. Nel caso di connessioni simmetriche il data rate massimo è di 433,9 kbit/s in entrambe le direzioni; mentre, per connessioni asimmetriche si raggiungono 723,2 kbit/s in una direzione e 57,6 kbit/s in quella opposta. Uno slave può trasmettere solamente se nello slot precedente aveva ricevuto un pacchetto dal master. In questi tipi di collegamenti, in genere, viene applicata la ritrasmissione dei pacchetti.

La connessione SCO prevede connessioni a commutazione di circuito, connessioni punto-punto e connessioni simmetriche. Questo tipo di connessione è generalmente utilizzata per il trasporto della voce in canali da 64 kbit/s. Il master può supportare fino a tre collegamenti SCO verso lo stesso slave o verso slave differenti appartenenti alla stessa piconet. Uno slave, invece, può supportare fino a tre connessioni SCO verso lo stesso master, o due se i collegamenti sono stati creati da diversi master. A causa della sensibilità al ritardo di questi di pacchetti (trasportano dati di natura real-time), non è prevista alcuna ritrasmissione in caso di errore o perdita.

Tipologia della rete

Due o più dispositivi collegati tra loro formano una piconet. I dispositivi all'interno di una piconet possono essere di due tipi: master o slave. Il master è il dispositivo che all'interno di una piconet si occupa di tutto ciò che concerne la sincronizzazione del clock degli altri dispositivi (slave) e la sequenza dei salti di frequenza. Gli slave sono unità della piconet sincronizzate al clock del master e al canale di frequenza.

Le specifiche Bluetooth prevedono 3 tipi di topologie: punto-punto, punto-multipunto e scatternet. Diverse piconet possono essere collegate tra loro in una topologia chiamata scatternet.

Gli slave possono appartenere a più piconet contemporaneamente, mentre il master di una piconet può al massimo essere lo slave di un'altra. Il limite di tutto ciò sta nel fatto che all'aumentare del numero di piconet aumentano anche il numero di collisioni dei pacchetti e di conseguenza degradano le prestazioni del collegamento. Ogni piconet lavora indipendentemente dalle altre sia a livello di clock che a livello di salti di frequenza. Questo perché ogni piconet ha un proprio master. Un dispositivo Bluetooth può partecipare sequenzialmente a diverse piconet come slave attraverso

l'uso di tecniche TDM (Time Division Multiplexing), ma può essere master solo in una.

Modalità operative

Un dispositivo Bluetooth si può trovare essenzialmente in due stati: in quello di connessione o in quello di standby. L'unità si trova nello stato di connessione se è connesso ad un altro dispositivo ed è coinvolto con esso alle normali attività. Se il dispositivo non è connesso, o non è coinvolto alle attività della piconet, allora esso si trova automaticamente nello stato di standby. Questo stato è stato concepito come un modo per far risparmiare energia ai dispositivi. Se uno di essi non è coinvolto attivamente all'interno di una connessione, non c'è motivo che assorba picchi di potenza pari a quelli dei dispositivi attivi. Quando un'unità si trova in standby ascolta il canale ogni 1,28 secondi per eventuali messaggi dal master.

Quando un dispositivo passa dallo stato di standby a quello di connessione, esso può essere collocato in una delle seguenti modalità:

- Active mode: l'unità partecipa attivamente alla piconet, sia in ricezione che in trasmissione, ed è sincronizzata al clock del master. Il master trasmette regolarmente per mantenere la sincronizzazione del sistema. Gli slave hanno un indirizzo di 3 bit AMADDR (Active Member Address).

- Hold mode: il master può mettere i dispositivi slave nello stato di Hold per un tempo determinato. Durante questo periodo nessun pacchetto può essere trasmesso dal master anche se il dispositivo mantiene il suo AMADDR e la sincronizzazione con il master. Questa modalità operativa è utilizzata generalmente nel momento in cui non si devono inviare pacchetti ad un dispositivo per un periodo relativamente lungo (questo ci fa capire che tale modalità operativa è supportata solamente nel caso in cui tra due dispositivi bluetooth ci sia un collegamento di tipo ACL). Durante questo periodo, il dispositivo si può spegnere per risparmiare energia. L'Hold mode può essere utilizzata anche nel caso in cui un'unità vuole scoprire o essere scoperta da altri dispositivi bluetooth o vuole partecipare ad altre piconet.

- Sniff mode: lo slave che passa in questo stato si trova in una modalità di risparmio energetico. Per entrare nello sniff mode, master e sla-

ve devono negoziare due parametri: uno "sniff interval" ed uno "sniff offset". Con il primo si fissano gli slot di sniff, mentre con il secondo si determina l'istante del primo slot di sniff. Quando il collegamento entra in sniff mode, il master può inviare pacchetti solamente all'interno degli sniff slot. Quindi lo slave ascolta il canale ad intervalli ridotti. Il master può costringere lo slave ad entrare in sniff mode, ma entrambi possono chiedere il passaggio. L'intervallo di sniff mode è programmabile.

- Park mode: il dispositivo è ancora sincronizzato alla piconet ma perde il suo indirizzo di dispositivo attivo (AMADDR) e riceve un nuovo indirizzo di 8 bit (PMADDR, Park Mode Address). Questa modalità è stata ideata per avere la possibilità di costituire piconet con più di sette slave. Infatti si possono avere fino ad massimo di 255 (28-1) dispositivi in modalità Park. Utilizzando tale indirizzo il master è in grado di identificare un particolare dispositivo in tale modalità ed effettuare il passaggio all'active mode. Le unità in questo stato ascoltano regolarmente il traffico sulla rete per risincronizzarsi e ricevere messaggi di broadcast. Questi ultimi, infatti, sono gli unici messaggi che possono essere inviati ad uno slave in park mode. La richiesta di passaggio in park mode può avvenire indifferentemente da parte del master o dello slave. Lo slave per richiedere l'attivazione al master riceve un indirizzo di Active Request (ARADDR) non unico.

Architettura

Come avviene per l'architettura OSI, Bluetooth specifica un approccio a livelli nella sua struttura protocollare. Differenti protocolli sono utilizzati per differenti applicazioni. Indipendentemente del tipo di applicazione, però, lo stack protocollare Bluetooth porta sempre all'utilizzo dei livelli data-link e fisico. Non tutte le applicazioni usano tutti i protocolli dello stack Bluetooth, infatti, esso è rappresentato su più livelli verticali, al di sopra dei quali c'è un'applicazione specifica.

Scendendo un po' più in dettaglio è possibile identificare le funzioni principali svolte dai protocolli più importanti dello stack Bluetooth:

- **Bluetooth Radio**: definisce i requisiti della parte in radio frequenza. Qui è dove i segnali radio vengono processati.

- **Baseband**: abilita il collegamento fisico tra dispositivi all'interno di una piconet. Tale livello si basa sulle procedure di inquiry e di paging per la sincronizzazione e la connessione di dispositivi bluetooth. Permette di stabilire i due diversi tipi di connessione (ACL e SCO).

- **LMP**: è responsabile dell'organizzazione del collegamento, del controllo tra dispositivi bluetooth e del controllo e negoziazione della dimensione dei pacchetti. È anche utilizzato per quanto riguarda la sicurezza: autenticazione e crittografia, generazione, scambio e controllo chiavi. Effettua anche il controllo sulle diverse modalità di gestione della potenza (park, sniff, hold) e sullo stato della connessione di un dispositivo all'interno della piconet. I messaggi LMP sono filtrati ed interpretati dal link manager in sede di ricezione, di conseguenza non saranno mai trasmessi ai livelli superiori. Questi messaggi hanno priorità maggiore rispetto ai pacchetti che trasportano dati utenti.

- **L2CAP**: esegue il multiplexing dei protocolli di livello superiore, la segmentazione e il riassemblaggio dei pacchetti e il trasporto di informazione relativa alla QoS (Quality of Service) ovvero è possibile richiedere una certa QoS da riservare ad un determinato link. L2CAP permette ai protocolli dei livelli superiori ed alle applicazioni di trasmettere e ricevere pacchetti di dati di dimensione superiore a 64Kbyte. Esso definisce solamente un collegamento di tipo connectionless. I canali audio di solito vengono fatti girare su collegamenti SCO; per ovviare a questo problema dati audio possono essere inviati all'interno di pacchetti di protocolli che girano su L2CAP.

- **RFCOMM**: emula una porta seriale (RS-232) sul protocollo L2CAP. Questo livello è necessario in quanto esistono applicazioni (come per esempio OBEX) che utilizzano un meccanismo di trasmissione seriale.

- **TCS BIN**: opera a livello bit e definisce i segnali di controllo per le chiamate voce e dati tra dispositivi Bluetooth e le procedure per gestire gruppi di dispositivi TCS.

- **SDP**: è un elemento importante all'interno della tecnologia Bluetooth, in quanto permette alle applicazioni di avere informazioni sui dispositivi, sui servizi offerti e sulle caratteristiche dei servizi disponibili.

Dopo aver individuato il dispositivo che implementa un determinato servizio è possibile stabilire una connessione.

- **AUDIO**: la funzione di questo strato è quella di codificare il segnale audio. Due tecniche possono essere adottate: log PCM e CVSD; entrambe forniscono un flusso di bit a 64kbit/s. La codifica log PCM (Pulse Code Modulation) consiste in una quantizzazione non uniforme a 8 bit. Nella codifica CVSD (Continuous Variable Slope Delta Modulation) il bit d'uscita indica se il valore predetto è maggiore o minore del valore della forma d'onda in ingresso, costituita da un segnale PCM con quantizzazione uniforme. Il passo è determinato dalla pendenza della forma d'onda.

Gli adopted protocols sono così chiamati perché sono protocolli definiti da altre organizzazioni di standardizzazione e incorporati nell'architettura Bluetooth: PPP (lo standard Internet per trasportare i pacchetti IP su una connessione punto a punto), TCP/UDP-IP (le fondamenta della suite TCP/IP), OBEX (object exchange, un protocollo a livello sessione sviluppato dalla Infrared Data Association per scambio di oggetti, simile all'HTTP ma più semplice; usato ad esempio per trasferire dati in formato vCard e vCalendar, cioè biglietto da visita e calendario degli impegni) e WAE/WAP (Wireless Application Environment e Wireless Application Protocol).

1.2 Android

Android è un sistema operativo per dispositivi mobili sviluppato da Google Inc. sulla base del kernel Linux. Android è stato progettato principalmente per smartphone e Tablet, con interfacce utente specializzate per televisori (Android TV), automobili (Android Auto), orologi da polso (Android Wear), occhiali (Google Glass), e altri.

Android è per la quasi totalità Free and Open Source Software (ad esclusione per esempio dei driver non-liberi inclusi per i produttori di dispositivi) ed è rilasciato sotto i termini della licenza libera Apache 2.0.

Figura 1.2: Logo Android

1.2.1 Storia

Nell'ottobre 2003 Andy Rubin (co-fondatore di Danger), Rich Miner (co-fondatore di Danger e di Wildfire Communications), Nick Sears (vicepresidente di T-Mobile) e Chris White (principale autore dell'interfaccia grafica di Web TV), fondarono una società, la Android Inc. per lo sviluppo di quello che Rubin definì *«...dispositivi cellulari più consapevoli della posizione e delle preferenze del loro proprietario»*. Da ciò probabilmente scaturì la scelta del nome Android.

Inizialmente la società operò in segreto, rivelando solo di progettare software per dispositivi mobili. Durante lo stesso anno il budget iniziale si esaurì, motivo per cui fu fondamentale un finanziamento di 10 000 dollari da parte di Steve Perlman (amico intimo di Rubin) per poter continuare lo

sviluppo. Steve Perlman consegnò a Rubin il denaro in una busta e rifiutò la partecipazione nella società.

Il 17 agosto 2005 Google ha acquisito l'azienda, in vista del fatto che la società di Mountain View desiderava entrare nel mercato della telefonia mobile. È in questi anni che il team di Rubin comincia a sviluppare un sistema operativo per dispositivi mobili basato sul kernel Linux. La presentazione ufficiale del robottino verde avvenne il 5 novembre 2007 dalla neonata OHA (Open Handset Alliance), un consorzio di aziende del settore Hi Tech che include Google, produttori di smartphone come HTC e Samsung, operatori di telefonia mobile come Sprint Nextel e T-Mobile, e produttori di microprocessori come Qualcomm e Texas Instruments Incorporated. Il primo dispositivo equipaggiato con Android che venne lanciato sul mercato fu l'HTC Dream, il 22 ottobre del 2008.

Dal 2008 gli aggiornamenti di Android per migliorarne le prestazioni e per eliminare i bug delle precedenti versioni sono stati molti.

Ogni aggiornamento o release, similmente a quanto accade per molte versioni di Linux, segue un ordine alfabetico e una precisa convenzione per i nomi, che in questo caso sono quelli di dolci: la versione 1.5 prese il nome Cupcake che venne seguita dalla versione 1.6 Donut, la 2.1 venne chiamata Eclair, la 2.2 Froyo, la 2.3 Gingerbread, la 3.0 Honeycomb, la 4.0 Ice Cream Sandwich, la 4.1 Jelly Bean, la 4.4 Kit Kat in seguito ad un accordo con la Nestlé, mentre la più recente, lanciata il 14 ottobre 2014, è la 5.0, col nome Lollipop.

Nel marzo 2013 Larry Page annuncia che Andy Rubin ha lasciato la presidenza di Android per dedicarsi ad altri progetti di Google. Viene rimpiazzato da Sundar Pichai.

1.2.2 Descrizione

Android adotta una politica di licenza di tipo open source (escluse alcune versioni intermedie), e si basa su kernel Linux. La licenza (Licenza Apache) sotto cui è rilasciato consente di modificare e distribuire liberamente il codice sorgente. Inoltre, Android dispone di una vasta comunità di sviluppatori che realizzano applicazioni con l'obiettivo di aumentare le funzionalità dei dispositivi. Queste applicazioni sono scritte soprattutto in linguaggio di programmazione Java.

Nell'ottobre 2012 le applicazioni disponibili presenti sul market ufficiale Android (Google Play) hanno raggiunto le 700.000 unità. Questi fattori hanno permesso ad Android di diventare il sistema operativo più utilizzato in ambito mobile, oltre a rappresentare, per le aziende produttrici, la migliore scelta in termini di bassi costi, personalizzazione e leggerezza del sistema operativo stesso, senza dover scrivere un proprio sistema operativo da zero.

Android Open Source Project

Il progetto Open Source Android è guidato da Google e, con il compito di mantenimento e allo sviluppo di Android secondo il progetto l'obiettivo è quello di creare un vero e proprio successo, in modo da migliorare l'esperienza mobile per gli utenti, AOSP mantiene anche la compatibilità dei programmi per Android, la definizione di un dispositivo Android compatibile, per esempio quel dispositivo su cui è possibile eseguire qualsiasi applicazione scritta da sviluppatori di terze parti che utilizzano Android SDK e NDK., per prevenire implementazioni incompatibili in Android. Il programma di compatibilità è facoltativo e gratuito, e la suite che consente di effettuare test di compatibilità è sempre gratuita e open-source.

Linux Kernel

Android è costituito da un Kernel basato sul kernel Linux 2.6 e 3.x (da Android 4.0 in poi), con middleware, Librerie e API scritte in C (o C++) e software in esecuzione su un framework di applicazioni che include librerie Java compatibili con librerie basate su Apache Harmony. Android utilizza la Dalvik virtual machine con un compilatore just-in-time per l'esecuzione di Dalvik dex-code (Dalvik Executable), che di solito viene tradotto da codice bytecode Java. La piattaforma hardware principale di Android è l'architettura ARM. L'architettura x86 è supportata grazie al progetto Android x86 e Google TV utilizza una speciale versione x86 di Android.

Il kernel Linux di Android mette a disposizione modifiche all'architettura create da Google al di fuori del ciclo di sviluppo del kernel. Un tipico sistema Android non possiede un X Window System nativo, né supporta il set completo standard di librerie GNU, e nel caso del C++ vi è solo una implementazione parziale delle STL. Tutto ciò rende difficile il porting di applicazioni Linux o librerie per Android. Per semplificare lo sviluppo

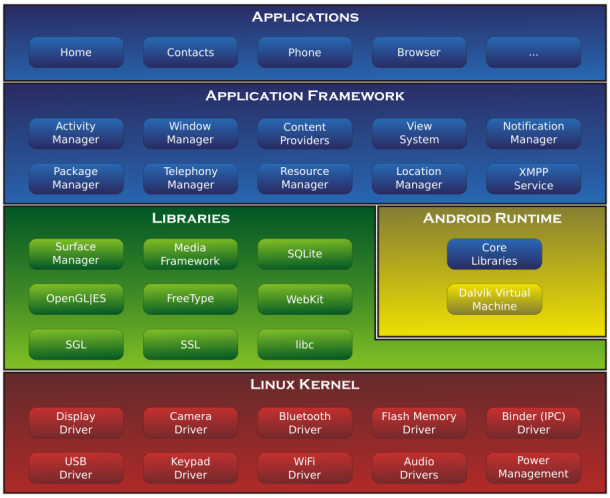


Figura 1.3: Schema di architettura

di applicazioni C/C++ sotto Android si usa SDL che tramite un piccolo Wrapper java permette l'utilizzo della JNI dando un'idea di utilizzo simile a un'applicazione nativa in C/C++.

Le applicazioni Android sono Java-based; in effetti le applicazioni scritte in codice nativo in C/C++ devono essere richiamate dal codice java, tutte le chiamate a sistema fatte in C (o C++) devono chiamare codice virtual machine Java di Android: infatti l'API multimediale SDL sotto Android richiama metodi di Java, questo significa che il codice dell'applicazione C/C++ deve essere inserito all'interno di un progetto Java, il quale produce alla fine un pacchetto Android (APK).

Alcune funzionalità implementate da Google hanno contribuito al kernel Linux, in particolare una funzione di gestione dell'energia denominata wakelocks, che però è stata respinta dagli sviluppatori del kernel mainline, in parte perché gli sviluppatori del kernel hanno ritenuto che Google non manifestasse alcuna intenzione di mantenere il proprio codice.[Anche se Google ha annunciato nel mese di aprile 2010 che avrebbero assunto due dipendenti per lavorare con la comunità del kernel Linux,[27] Greg Kroah-Hartman, l'attuale responsabile del kernel di Linux per il ramo stabile, ha dichiarato nel dicembre 2010 che era preoccupato del fatto che Google non sembrava più intenzionata a far includere le modifiche al codice nel ramo principale di

Linux. Alcuni sviluppatori di Google Android hanno fatto capire che il team di Android si è stancato del processo, perché erano una piccola squadra e avevano molto lavoro urgente da fare su Android.

In Linux è stato incluso l'autosleep e le capacità wakelocks nel kernel 3.5, dopo molti precedenti tentativi di fusione. Le interfacce sono le stesse ma la realizzazione di Linux ha due diverse modalità di sospensione: a memoria (le sospensione tradizionale che utilizza Android), e su disco (ibernazione, come è noto sul desktop). Nel mese di agosto 2011, Linus Torvalds ha detto che alla fine Android e Linux sarebbero venuti di nuovo ad un nucleo comune, ma probabilmente non sarà per quattro o cinque anni[30] Nel mese di dicembre 2011, Kroah-Hartman ha annunciato l'inizio del progetto mainlining Android, che mira a mettere un po' di Android driver, le patch e le caratteristiche, nel kernel di Linux a partire da Linux 3.3.

La memoria flash sui dispositivi Android è divisa in diverse partizioni, ad esempio /system per il sistema operativo stesso e /data per i dati utente e le installazioni delle app. Diversamente rispetto alle tradizionali distribuzioni GNU/Linux, agli utenti di dispositivi Android non sono disponibili i privilegi di superutente, o root, per l'accesso al sistema operativo e alle sue partizioni, quali /system, per le quali l'utente dispone dei permessi di sola lettura. Tuttavia, l'accesso come root sul dispositivo è quasi sempre possibile: in certi casi tramite richiesta al produttore, in altri sfruttando certe falle di sicurezza di Android. L'accesso root viene utilizzato frequentemente dalla comunità open source, per migliorare le capacità dei loro dispositivi, ma anche da soggetti malintenzionati per installare virus e malware.

Altre caratteristiche

L'interfaccia utente di Android è basata sul concetto di direct manipulation per cui si utilizzano gli ingressi mono e multi-touch come strisciate, tocchi e pizzichi sullo schermo per manipolare gli oggetti visibili sullo stesso. La risposta all'input dell'utente è stata progettata per essere immediata e tentare di fornire un'interfaccia fluida. Sensori hardware interno come accelerometri, giroscopi e sensori di prossimità sono utilizzati da alcune applicazioni per rispondere alle azioni da parte dell'utente, ad esempio la regolazione dello schermo da verticale a orizzontale a seconda di come il dispositivo è orientato o che consentono all'utente di guidare un veicolo in una corsa virtuale ruotando il dispositivo, simulando il controllo di un volante.

La cosiddetta Homescreen è simile al desktop trovato su Windows ed è la schermata principale che ci si trova appena il device è stato avviato, oppure premendo il tasto Home. L'homescreen di Android è in genere occupata sia dalle icone delle applicazioni che dai widget cioè delle sorti di gadgets con varie funzioni; ci sono widget che mostrano vari stili di orologi, quelli che mostrano gli ultimi video di YouTube, altri che visualizzano informazioni meteo, quelli relativi alle email. La homescreen può essere costituita da più pagine tra cui l'utente può scorrere avanti e indietro.

Sempre presente nella parte superiore dello schermo si trova una barra di stato, che mostra le informazioni sul dispositivo e la sua connettività. Trascinando la barra di stato verso il basso compare una schermata di notifica in cui le applicazioni possono visualizzare notifiche relative ad informazioni importanti o aggiornamenti come ad esempio una e-mail appena ricevuta o un SMS, in modo da non interrompere immediatamente l'utente.[35] Nelle prime versioni di Android tali notifiche potevano essere sfruttate esclusivamente per aprire l'applicazione in questione, ma gli aggiornamenti più recenti hanno fornito maggiori funzionalità, come ad esempio la possibilità di chiamare un numero direttamente dalla notifica della chiamata persa, senza dover aprire l'applicazione telefono[36] Le notifiche sono persistenti fino alla loro lettura o cancellazione da parte dell'utente.

La piattaforma usa il database SQLite, la libreria dedicata SGL per la grafica bidimensionale (invece del classico server X delle altre distribuzioni linux) e supporta lo standard OpenGL ES 2.0 per la grafica tridimensionale. Le applicazioni vengono eseguite tramite la Dalvik virtual machine, una macchina virtuale adattata per l'uso su dispositivi mobili.

Android è stato progettato principalmente per smartphone e tablet, ma il carattere aperto e personalizzabile del sistema operativo permette che sia utilizzato anche su altri dispositivi elettronici tra cui portatili e netbook, smartbook, eBook reader, fotocamere e smart TV (Google TV). Il mercato delle smart things è cresciuto in maniera notevole in questi ultimi periodi a tal punto da stimolare la creativitá delle persone. Un esempio è lo Smartwatch dotato di sistema operativo Android in versione light cuffie, lettori auto CD e DVD, occhiali intelligenti (Project Glass o google glass), frigoriferi, sistemi di navigazione satellitare per veicoli, sistemi di automazione per la casa, console di gioco, specchi, telecamere, lettori MP3/MP4 e tapis roulant.

Il logo di Android è stato progettato insieme con la famiglia di caratteri

(font) Droid di Ascender Corporation, il verde è il colore del robot che rappresenta il sistema operativo Android. Il colore di stampa è PMS 376C e il colore RGB in valore esadecimale è A4C639, come specificato dalle linee guida del Marchio di Android Il carattere personalizzato di Android si chiama Norad. Viene utilizzato solo nel logo di testo.

1.3 Peer-to-Peer(P2P)

Il termine **Peer-to-Peer** (*abbr.* **P2P**) indica un'architettura logica di rete informatica in cui i nodi non sono gerarchizzati unicamente sotto forma di client o server fissi (clienti e serventi), ma sotto forma di nodi equivalenti o paritari (in inglese peer) che possono cioè fungere sia da cliente che da servente verso gli altri nodi terminali (host) della rete. Essa dunque è un caso particolare dell'architettura logica di rete client-server.

Mediante questa configurazione qualsiasi nodo è in grado di avviare o completare una transazione. I nodi equivalenti possono differire nella configurazione locale, nella velocità di elaborazione, nella ampiezza di banda e nella quantità di dati memorizzati. L'esempio classico di P2P è la rete per la condivisione di file (File sharing).

In Microsoft si tende a definire con il termine peer-to-peer una rete di un piccolo gruppo di persone (non più di 10 persone), dove la protezione non costituisce un problema, modalità normalmente conosciuta con il termine gruppo di lavoro, in antitesi alle reti cliente-servente in cui è presente un dominio centralizzato.

1.3.1 Applicazioni

Il termine può essere tecnicamente applicato a qualsiasi tipo di tecnologia di rete e di applicazioni che utilizzano questo modello, come per esempio il protocollo Nntp utilizzato per il trasferimento delle notizie Usenet, ARPANET, applets java, live chat decentralizzate o le BBS di Fido Net. Il termine frequentemente viene riferito alle reti di file sharing (condivisione file, per questo definita Peer to peer filesharing o P2P-Filesharing) come Gnutella, FastTrack, e l'ormai defunto Napster che forniscono, o per quanto riguarda Napster forniva, il libero scambio (e qualche volta anonimo) di file tra i computer connessi a Internet.

Alcune reti e canali, come per esempio Napster, OpenNap o IRC usano il modello client-server per alcuni compiti (per esempio la ricerca) e il modello peer-to-peer per tutti gli altri. Proprio questa doppia presenza di modelli, fa sì che tali reti siano definite ibride. Reti come Gnutella, Freenet o quella di EMule, vengono definite come il vero modello di rete peer-to-peer in quanto utilizzano una struttura peer-to-peer per tutti i tipi di transazione, e per questo motivo vengono definite pure.

Quando il termine peer-to-peer venne utilizzato per descrivere la rete Napster, implicava che la natura a file condivisi del protocollo fosse la cosa più importante, ma in realtà la grande conquista di Napster fu quella di mettere tutti i computer collegati sullo stesso piano. Il protocollo peer era il modo giusto per realizzarlo.

1.3.2 Vantaggi e svantaggi

Vantaggi e svantaggi sono relativi al tipo di ambiente in cui si decide di installare questo tipo di rete, ma sicuramente si deve tenere presente che:

- Non si deve acquistare un server con potenzialità elevate e quindi non se ne deve sostenere il costo, ma ogni computer deve avere i requisiti per sostenere l'utente in locale e in rete, ma anche gli altri utenti che desiderano accedere alle risorse di questo in remoto;

- Ogni utente condivide localmente le proprie risorse ed è amministratore del proprio client-server. Questo da un lato può essere positivo per una questione di indipendenza, ma dall'altro richiede delle competenze ad ogni utente, soprattutto per quel che concerne la protezione;

- La velocità media di trasmissione dei dati è molto più elevata di una classica rete con sistema Server / Client, dal momento che l'informazione richiesta da un Client può essere reperita da numerosi Client connessi in modo paritario (ossia peer), anziché da un unico server (questo tipo di condivisione diventa tanto più efficace tanti più sono i Client connessi, in antitesi con la rete tradizionale Server/Client, dove un elevato numero di Client connessi riduce la velocità di trasmissione dati per utente);

- La sicurezza degli accessi ai client viene gestita localmente su ogni macchina e non centralizzata, questo significa che una rete basata su utenti deve avere lo stesso archivio reimpostato in ogni client.

1.3.3 Funzionalità

La maggioranza dei programmi peer-to-peer garantisce un insieme di funzionalità minime, che comprende:

- supporto multipiattaforma, multiserver, multicanale: il programma è compatibile con tutti i sistemi operativi, server e dispositivi hardware (PC, laptop portatili, palmari, cellulari);
- supporto protocollo IPv6;
- download dello stesso file da più reti contemporaneamente;
- offuscamento dell'ID di rete;
- offuscamento del protocollo P2P;
- supporto proxy e Tor;
- supporto crittografia SSL;
- gestione da remoto, sia da PC/notebook che da cellulari e palmari.

Una funzionalità di recente sviluppo è l'assegnazione di una priorità delle fonti, privilegiando quelle con connessione a banda larga (ad esempio, BitTyrant per i file torrent).

Altri usi del P2P si vedono nel campo dei videogiochi, specialmente MMORPG quali World Of Warcraft e similari; al momento il P2P viene utilizzato per distribuire client di gioco o anche espansioni e patch in modo che i file vengano distribuiti più rapidamente e con meno carico sui server principali di distribuzione.

1.4 Reti ad-hoc

Una **wireless ad hoc network** (**WANET**) è un tipo di rete wireless decentralizzata. La rete si dice ad-hoc poichè non si basa su infrastrutture preesistenti, come i router in una rete collegata o punti d'accesso nelle reti wireless. Invece, ogni nodo partecipa al routing inoltrando dati agli altri nodi, in modo tale che la determinazione di quale nodo inoltra il dato è fatto dinamicamente sulla base della connettività di rete. Inoltre oltre al classico routing, le reti ad-hoc possono essere usate per il flooding dei dati inoltrati.

Una rete ad hoc si riferisce a ogni insieme di reti dove tutti i device hanno uno stesso stato su una rete e sono liberi di associare con ogni altra rete ad-hoc nel range di collegamento. Una rete ad-hoc spesso si riferisce al modo di operare delle reti wireless IEEE 802.11.

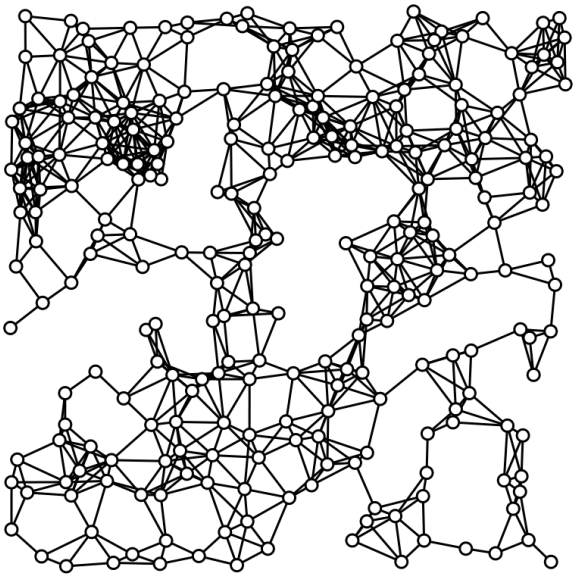


Figura 1.4: Distribuzione di una classica rete ad-hoc

1.4.1 Applicazioni

La natura decentralizzata delle reti wireless ad-hoc le rende adatte per una varietà di applicazione dove non si può contare su un nodo centrale e può migliorare la scalabilità delle reti rispetto le reti wireless gestite, ma teoricamente e praticamente i limiti all'intera capacità di tale rete devono essere identificati.

Una minima configurazione e un veloce rilascio rendono le reti ad-hoc adatte per situazioni di emergenza come disastri naturali o conflitti militari. La presenza di un protocollo di routing dinamico e adattativo rendono una rete ad-hoc capace di essere formata velocemente.

1.4.2 Requisiti tecnici

Una rete ad-hoc è una rete composta da una moltitudine di nodi connessi da collegamenti.

I collegamenti sono influenzati dalle risorse del nodo (e.g. potenza di trasmissione, potenza di computazione e memoria) e proprietà comportamentali (e.g. affidabilità), oltre a proprietà dei collegamenti (e.g. lunghezza del collegamento e perdità del segnale, interferenza e rumore). Dato che i collegamenti possono essere connessi o disconnessi in ogni momento, una rete funzionante deve essere capace di superare con questa ristrutturazione dinamica, preferibilmente in modo tale che è veloce, efficiente, affidabile, robusta e scalabile.

La rete deve permettere a due nodi di comunicare trasmettendo l'informazione attraverso altri nodi. Un percorso è una serie di collegamenti che connette due nodi. Più metodi due routing usano uno o due percorsi attraverso questi due nodi; il flooding usa tutti o la maggior parte dei percorsi disponibili.

1.4.3 Medium-access control

Nella maggior parte delle reti ad hoc wireless, i nodi competono per accedere a un medium wireless condiviso, spesso entrando in collisione(interferenza). Usando una comunicazione wireless cooperativa migliora l'immunità a problemi di interferenza avendo il nodo di destinazione combinato la propria interferenza e quella degli altri nodi per migliorare la decodifica del segnale desiderato.

1.4.4 Simulazione di reti wireless ad-hoc

Un problema chiave delle reti wireless ad-hoc è la previsione delle varie possibilità che possono avvenire. Come risultato, la modellazione e simulazione usando parametri estensivi generalizzata e l'analisi degli scenari diventa un paradigma estremamente importante da usare nelle reti ad-hoc.

La modellazione e simulazione agent-based offrono questo paradigma. Non bisogna confodersi però con i sistemi multi-agente e agenti intelligenti. La modellazione basata sugli agenti ha origine dalle scienze sociali, dove l'obiettivo era quello di valutare e vedere un sistema a larga scala con numerosi *AGENTI* o componenti interattivi in una larga varietà di situazioni random per osservare il fenomeno a livello globale. Diversamente dai tipi sistemi IA con agenti intelligenti, la modellazione agent-based è simile al mondo reale. I modelli agent-based sono così effettivi nella modellazione di sistemi bio-inspired e nature-inspired. In questi sistemi, le interazioni base dei componenti del sistema, anche chiamati *sistemi adattativi complessi*, sono semplici ma efficaci in un'avanzato fenomeno mondiale come emergenza.

1.4.5 Esempio: Mobile ad hoc network (MANET)

Una **mobile ad hoc network** (**MANET**) è una rete continuamente autoconfigurante, senza infrastruttura di dispositivi mobili connessi senza fili.

Ogni device in una MANET è libero di muoversi indipendentemente in ogni direzione, e cambiare quindi i suoi collegamenti agli altri device frequentemente. Ogni device deve inoltrare il traffico che non gli serve e deve quindi agire da router. La prima sfida nella costruzione di una MANET è l'equipaggiamento di ogni device di un continuo mantenimento dell'informazione richiesta dall'appropriato traffico di routing. Questa rete può operare autonomamente o può essere connessa a una più larga rete. Possono contenere uno o più trasmettitori tra i nodi. Ciò che si ottine è una topologia altamente dinamica e autonoma.

Tipi

- **Vehicular Ad hoc Networks** (**VANETs**) sono usate per la comunicazione attraverso veicoli e equipaggiamento stradale. VANETs

intelligenti (InVANETs) sono un tipo di IA che aiuta i veicoli di comportarsi intelligentemente durante le situazioni pericolose che possono capitare a due veicoli.

- **Smart Phone Ad hoc Networks** (**SPANs**) valorizzano l'hardware (Bluetooth e Wi-Fi) degli smart-phones attualmente in commercio per creare reti peer-to-peer senza basarsi sulla rete del cellulare, punti di accesso wireless, o infrastrutture di rete tradizionali. Le SPANs differiscono dalle tradizionali reti hub-spoke, come Wi-Fi Direct, in quanto aiutano le multi-hop relay e non c'è una nozione di group leader in modo tale che i peers possono unirsi e lasciare la rete a piacimento senza distruggere la rete.

- **Internet based mobile ad hoc networks** (**iMANETs**) sono reti ad hoc che collegano nodi mobili e nodi Internet Gateway fissi. Ad esempio, più sotto-MANET possono essere collegati da un classico Hub-Spoke VPN per creare un MANET geograficamente distribuito. In questo tipo di reti normali ad hoc algoritmi di routing non si applicano direttamente.

- **Military / Tactical MANETs** sono utilizzati da unità militari con particolare attenzione alla sicurezza, gamma, e l'integrazione con i sistemi esistenti. Le forme d'onda comuni includono dell'esercito statunitense SRW, Harris's ANW2 and HNW, Persistent Systems' Wave Relay, Trellisware's TSM and Silvus Technologies' StreamCaster.

- Una MANET è una rete ad-hoc ma una rete ad-hoc non è necessariamente una MANET.

Capitolo 2

Processo di sviluppo

2.1 Introduzione

In questo capitolo illustrerò l'intero processo di sviluppo con cui è stata realizzata la libreria per Android per le connessioni Bluetooth. Per fare ciò verrà utilizzato il classico template appreso nel corso di *Ingegneria del Software* per lasciare un documento ufficiale su cui migliorare e aggiornare la libreria in futuro.

2.2 Requisiti

Si ha bisogno di una libreria scritta in Java per Android che permetta a più dispositivi mobili la comunicazione attraverso Bluetooth. Si vuole che la libreria permetta di scoprire i dispositivi, le relative distanze e lo stato della connessione. I dispositivi devono essere visti solo se hanno la stessa applicazione basata sulla libreria.
Una volta iniziata la discovery deve permettere la visualizzazione dei soli dispositivi che hanno un'applicazione basata sulla libreria stessa.Tali dispositivi saranno detti *Neighbors* (o *Vicini*).
La libreria deve garantire l'ordine decrescente della lista dei dispositivi in base al più vicino. Inoltre un'applicazione basata su di essa, per garantire il P2P, deve potersi sia connettere che accettare altri dispositivi. Inoltre si deve garantire l'invio di messaggi sia broadcast sia diretti a singoli peer.

2.3 Analisi dei requisiti

2.3.1 Glossary

Name	Description
Bluetooth	Tecnologia di trasmissione dati per reti personali richiesta come mezzo di comunicazione fra i vari device
P2P	Architettura logica di rete in cui i nodi singoli sono paritari e possono fungere sia da server che da client.
Device	Smartphone con Android incorporato il quale ha a disposizione un modulo Bluetooth sul quale è installata un'applicazione con la libreria
Neighbor	Dispositivi vicini che si basano sulla stessa libreria che entrano nel raggio di discovery del *Device*. Essi sono peer rispetto il nostro *Device*
Messaggi Broadcast	Modalità di invio di un messaggio a tutti i *Neighbors*
Messaggi singoli	Modalità di invio che riguarda un singolo *Neighbors*. L'invio deve essere preceduta da una accettazione o connessione di un altro *Device* con il quale si vuole comunicare
Modalità discovery	E' la modalità in cui entra un device quando intende usufruire del servizio di discovery di Bluetooth. Grazie a questo è possibile scoprire i dispositivi che sono *discoverable*, ossia che hanno un flag per cui sono visibili agli altri *Device*.

2.3.2 Casi d'uso

Di seguito vengono mostrati i casi d'uso che il sistema necessita ai fini di rispettare i requisiti indicati.

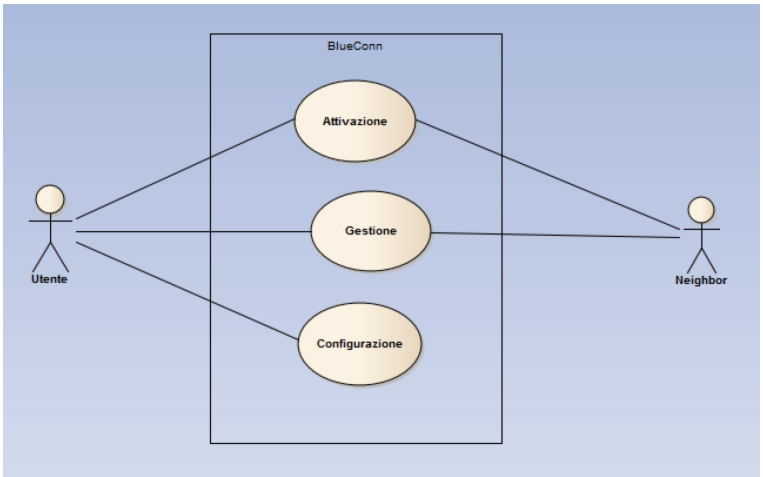


Figura 2.1: Use cases

2.3.3 Scenari

ID	Attivazione
Descrizione	Crea la connessione Bluetooth con altri device
Attori	*Neighbors*, *Utente*
Precondizione	Deve essere avvenuto prima lo scenario di Configurazione
Scenario principale	Un dispositivo incomincia la ricerca dei *Neighbors*. Intanto il dispositivo rimane in ascolto per accettare eventuali tentativi di connessione mentre può lui stesso inoltrarne altri.
Post-condizione	La ricerca dei dispositivi deve ripetersi automaticamente.

ID	Gestione
Descrizione	Instaura la connessione fra due dispositivi
Attori	*Neighbors*, *Utente*
Precondizione	Deve essere avvenuto prima lo scenario di Attivazione
Scenario principale	Un *Utente* decide di connettersi a uno dei *Neighbors* comparsi nello scenario di Attivazione. Una volta connesso, può avvenire lo scambio di messaggi
Scenario alternativo	Un *Neighbor* decide di connettersi al dispositivo dell'*Utente*. Una volta accettata la connessione, può avvenire lo scambio di messaggi

ID	Configurazione
Descrizione	Controlla se il Bluetooth è supportato, attiva il Bluetooth se disattivo e imposta i parametri che caratterizzerano la comunicazione Bluetooth
Attori	*Utente*
Scenario principale	L'utente necessita prima di tutto di avere a disposizione i parametri settati prima di incominciare con le altre possibili operazioni.

2.3.4 Modello del dominio

Viene illustrato di seguito il modello del dominio della libreria.

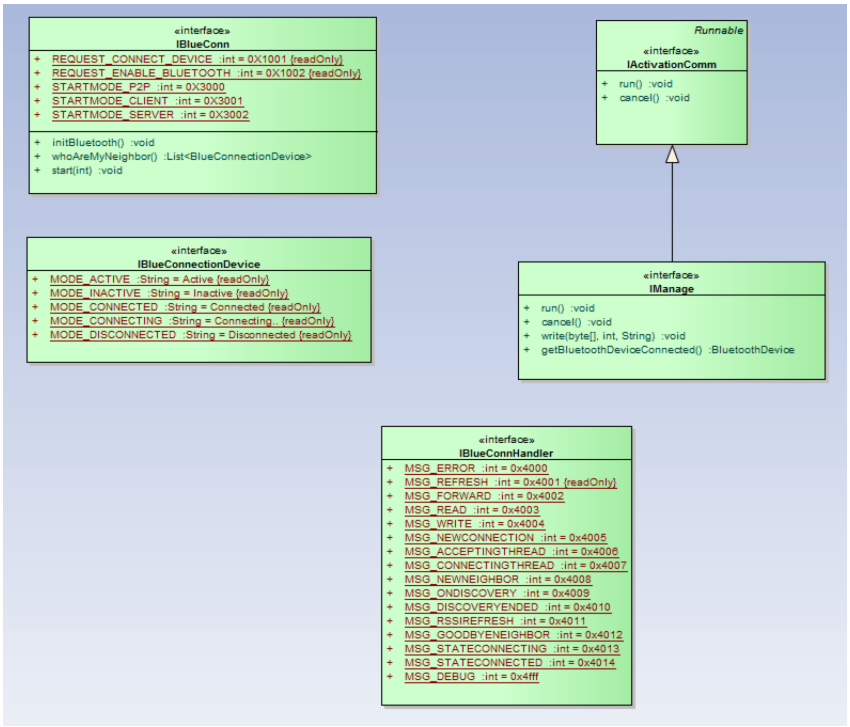


Figura 2.2: Domain Model

Il modello del dominio si basa essenzialmente su 4 punti cardine:

- *Handler*
- *Device*
- *Thread*
- *System*

Questi 4 sottosistemi uniti danno vita alla libreria stessa che comunicherà con le applicazione che dipendono da lei attraverso l'*Handler*, modellerà i *Neighbor* come *IBlueConnectioneDevice*, gestire le connessioni usando il sottosistema dei *Thread* prendendo quindi forma nel *System*.

2.4 Analisi del problema

In questa fase andremo ad analizzare il problema. Come da requisiti, si ricorda che il sistema sarà ambientato su device Android e che alcune classi e/o operazioni mostrate e non spiegate saranno dovute al linguaggio stesso.

2.4.1 Architettura logica

Interazione

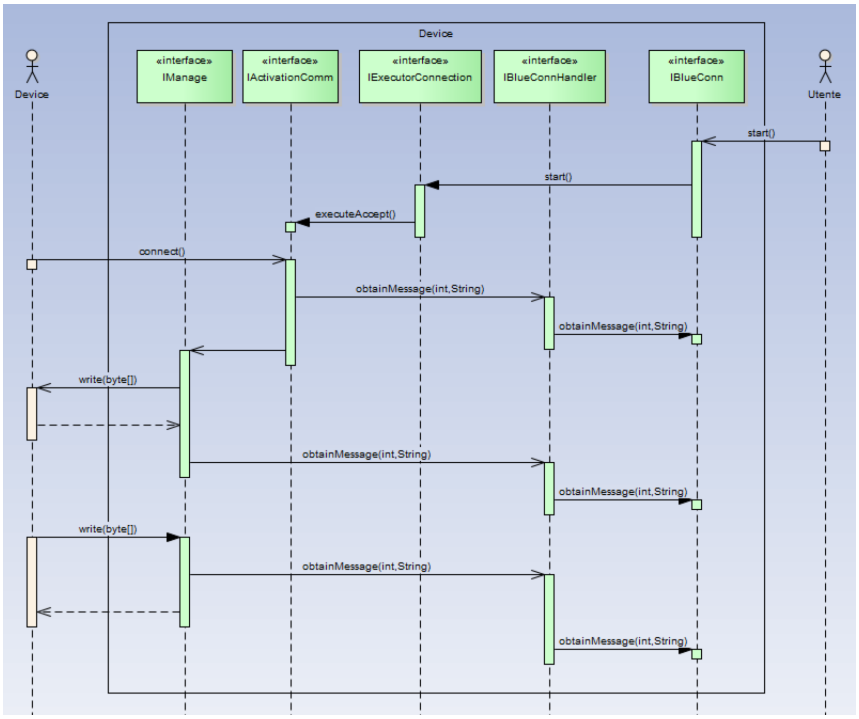


Figura 2.3: Interazioni in caso di attesa connessioni

Si assume nell'immagine seguente che l'utente abbia già inviato lo *start()*.

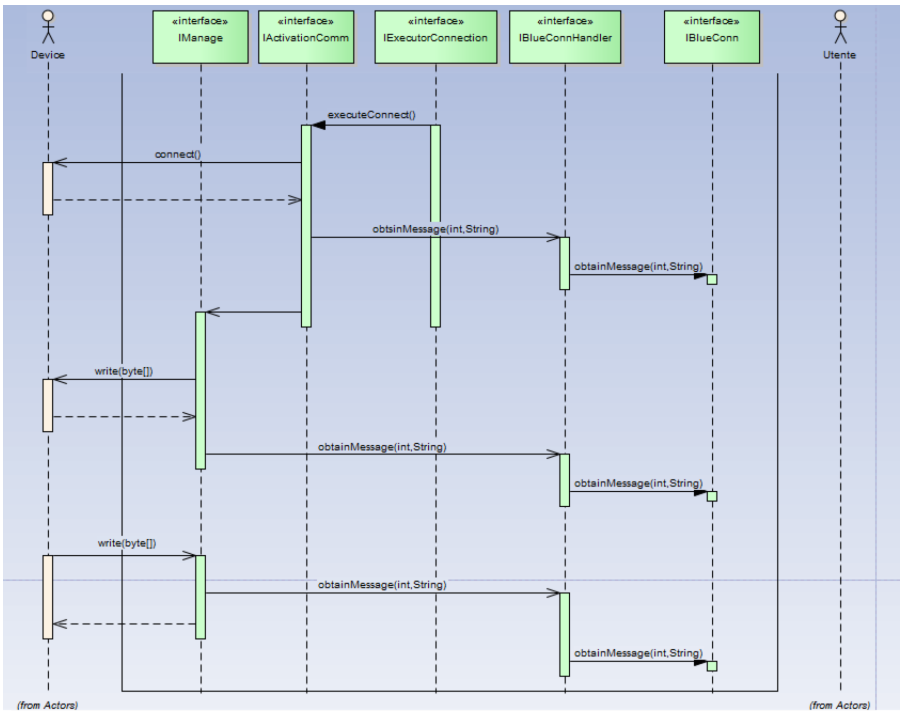


Figura 2.4: Interazioni in caso di invio richieste di connessione

Comportamento

Di seguito è riportato il comportamento del sistema in caso sia in attesa di connessioni e in caso di invio richieste di connessione.

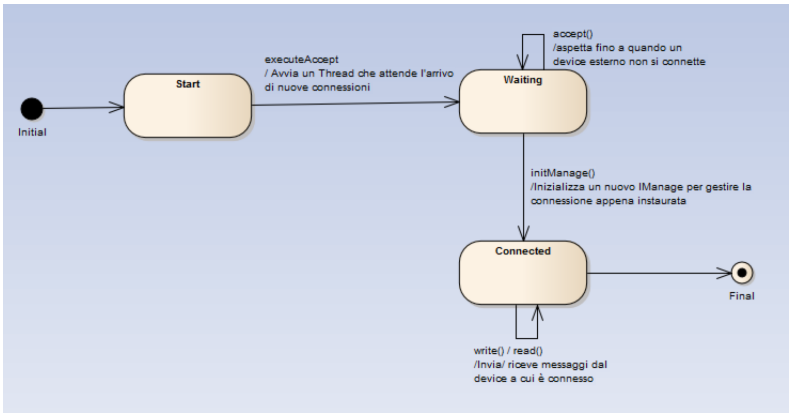


Figura 2.5: Comportamento in caso di attesa connessioni

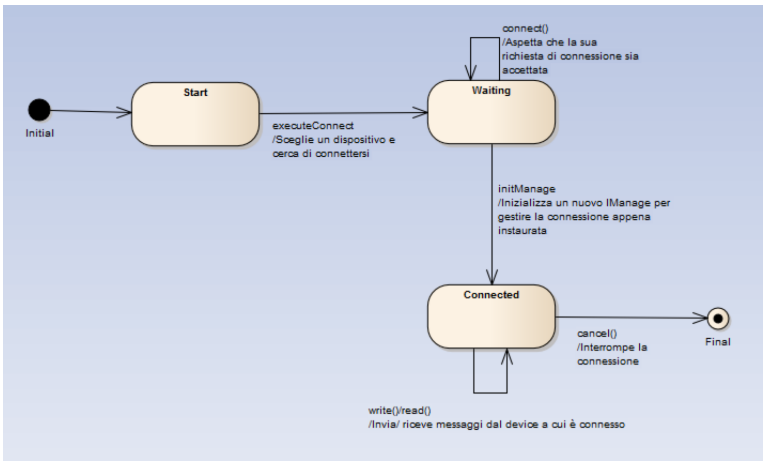


Figura 2.6: Comportamento in caso di invio richieste di connessione

Struttura

- **Handler**

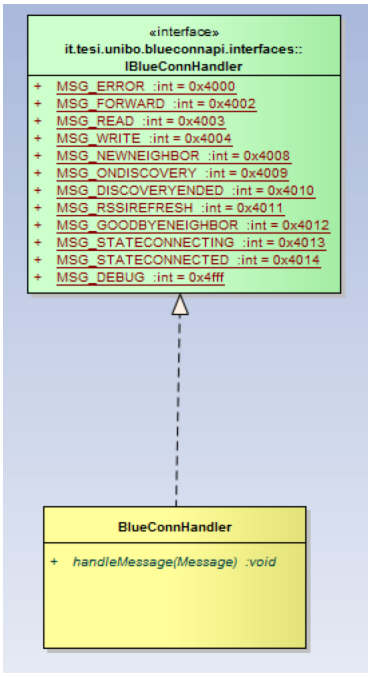


Figura 2.7: Struttura di Handler

L'**Handler** è la parte della libreria che ha il compito di trasmettere tutti i messaggi che la libreria invia a una qualsiasi applicazione che ne vuole usufruire. Per tale motivo è pensata come *abstract class* in quanto si vuole ogni sviluppatore che voglia usufruire della libreria possa modellare a proprio piacimento la gestione dei messaggi provenienti dalla libreria. La serie di costanti aggiunte sono i tipi di messaggi che la libreria invierà.

- **Device**

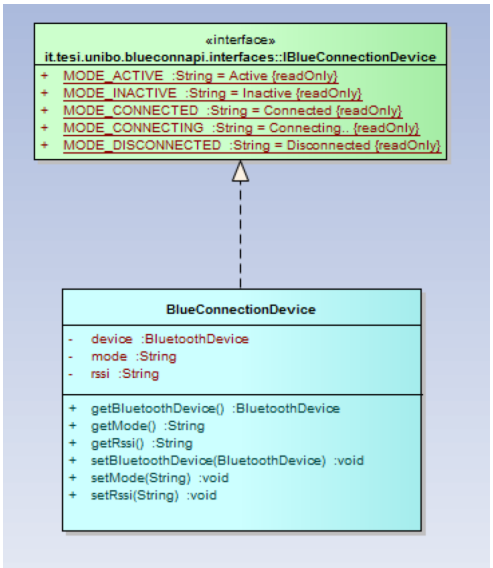


Figura 2.8: Struttura di Device

Il **BlueConnectionDevice** è l'istanziazione del *Neighbor* descritto nei requisiti. Si è pensato infatti di avere campi riferiti al device remoto stesso(*device*), il campo dedicato alla distanza (*rssi*) e un campo dedicato allo stato della connessione (*mode*). RSSI (**Received Signal Strength Indicator**), per quanto riguarda il Bluetooth, è un dato che misura la potenza ricevuta in un segnale ricevuto. E' un dato che è strettamente correlata alla potenza di trasmissione di un modulo Bluetooth presente all'interno di ogni device. Tenendo conto di ciò, se RSSI è basso allora l'altro device sarà distante altrimenti sarà vicino. In genere nei dispositivi mobili varia tra i -100dBm (molto lontano) e -10dBm solitamente(molto vicino). Essendo strettamente correlato con tanti fattori, questo parametro non è sicurissimo ma è l'unico reso a disposizione da Bluetooth stesso. Usando questo parametro è possibile valutare la qualità del segnale ricevuto ed è un parametro importantissimo ai fini della comunicazione. Le modalità

descritte nell'interfaccia sono le classiche sei modalità di connessione che caratterizzano un dispositivo nelle fasi di comunicazione.

- **Thread**

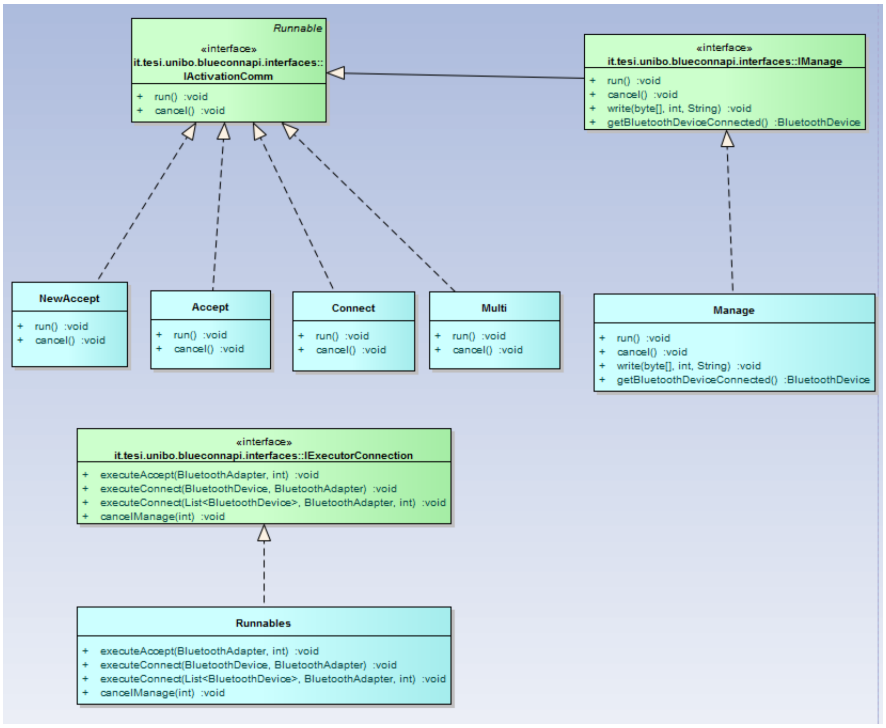


Figura 2.9: Struttura di Thread

I **Thread** devono rappresentare le fasi di comunicazione. *IExecutorConnection* rappresenta il metodo con cui il *System* esegue uno degli *IActivationComm*. Questi rappresentano le 4 modalità con cui la libreria può attivare la connessione.

- *Accept*: attesa di una singola connessione
- *NewAccept*: attesa di tutte le possibili connessioni
- *Connect*: connessione a un singolo *Neighbor*
- *Multi*: connessione a più *Neighbors*

Infine abbiamo *IManage* che specializza gli *IActivationComm* e serve una volta garantita la fase di connessione o accettazione l'instaurazione del canale di comunicazione. Per ogni device connesso si avrà un *IManage*. Si fornisce un metodo per comunicare che consiste

- **System**

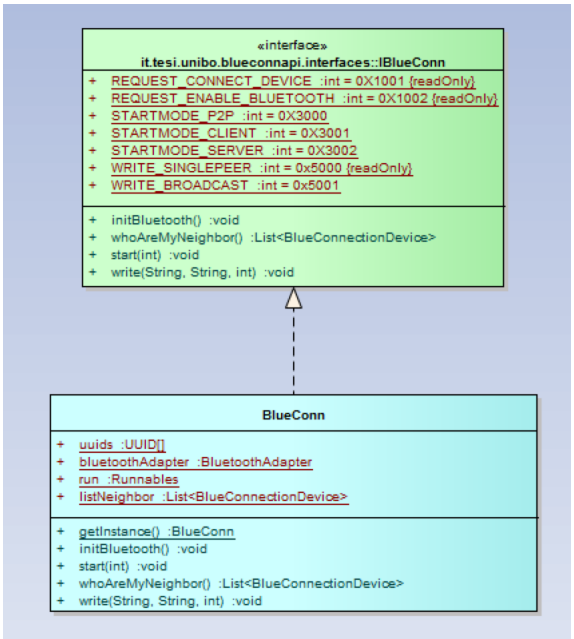


Figura 2.10: Struttura di System

Con **System** si intende il vero e proprio corpo centrale del sistema con i compiti richiesti dal committente. Esso dovrà essere configurabile con un costruttore adatto. Avrà il compito di inizializzare *Bluetooth* nel *Device* in modo di renderlo pronto alle fasi di *start*. Si è pensato che uno sviluppatore che usa questa libreria abbia l'opportunità di usarla in uno dei 3 metodi forniti ossia *Peer2Peer*, *Master* o *Slave*. In questo modo si lascia più libertà e la libreria non avrà problemi comunque a funzionare correttamente. Infine con il metodo *whoAreMyNeighbors()* si fornisce la possibilità di ottenere la lista aggiornata dei device. *IBlueConn*, secondo requisiti, deve essere in

grado di inviare messaggi sia a peer singoli che broadcast. Per cui si sono definiti delle costanti che aggiunte in *mode* nella definizione della funzione, forniscono il modo di poter avere entrambi i tipi di comunicazione nella stessa funzione. Inoltre, secondo requisiti stessi, si deve definire un metodo per il quale il *Device* è in continua scoperta di nuovi *Neighbors*. Lascio questa necessità di definizione del metodo *startReapeatingTask()* al progettista.

A questo punto si può definire l'intero sistema della libreria.

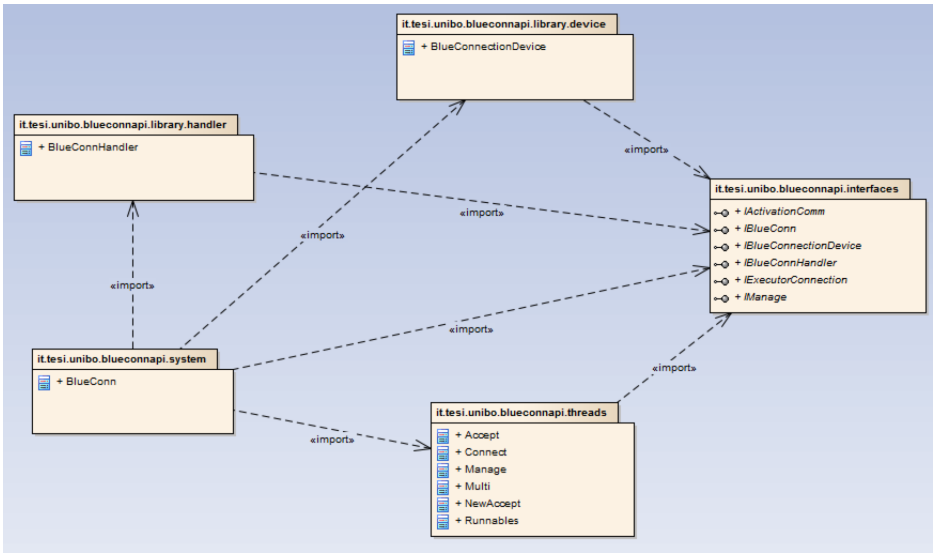


Figura 2.11: Struttura dei package

2.4.2 Abstraction Gap

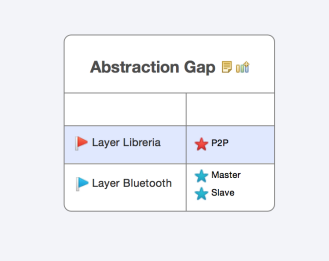


Figura 2.12: Il layer della libreria deve mascherare quello del Bluetooth per il P2P

Il gap maggiore da colmare con questa libreria è quello dovuto alla tecnologia di trasmissione scelta dai requisiti. Bluetooth, infatti, è un tipo di protocollo **Master-Slave** in quanto c'è un device detto *Client* che si connette a un *Server* e garantisce una comunicazione fra i due. Questo protocollo di comunicazione è strettamente in contrasto con i requisti forniti in quanto bisogna creare un tipo di protocollo che sia di tipo **P2P**. Infatti si vuole che un device sia in grado sia di connettersi a un altro device ma anche di accettare le connessioni da altri. Per colmare questo gap bisognerà garantire la trasparenza agli utilizzatori del fatto che Bluetooth è intrinsecamente un protocollo Master-Slave e fare in modo che oltre a connettersi, accetti anche le connessioni provenienti da altri device.

2.4.3 Analisi dei rischi

Dopo aver realizzato l'architettura logica del sistema, saranno identificate le **parti critiche** per il progetto e sviluppo.
Per identificare le parti che richiedono più attenzioni e impegno durante il progetto, è creata una tabella che mostra il livello di rischio di ogni parte del sistema.
Con livello di rischio si intende l'**impegno** e il **tempo** richiesto per il progetto di ogni sottosistema identificate nell'analisi.

Riconoscendo le **parti critiche** con maggiori probabilità di fallimento, può essere definito un miglior piano di lavoro.

Analisi dei rischi

	Livello di rischio
Device	Basso
Handler	Basso
Threads	Alto
System	Alto

Figura 2.13: Analisi dei rischi

2.5 Piano di lavoro

Tenendo conto della fase di analisi e quella dell'analisi dei rischi, può essere diviso ora il lavoro progettando e sviluppando le entità identificate nell'architettura logica.
Estimated Time To Complete (ETC) Definizione del termine: ETC è una proiezione del tempo e/o impegno richiesto a completare l'attività del progetto. ETC è un valore che è espresso in giorni di lavoro richiesti a completare un task o progetto

ETC è definito come il tempo richiesto per la costruzione di un primo prototipo.

Work Plan

	Project Lead (who)	Difficoltà	Priorità	ECT	Progress
Device	Luca	Facile	Minima	1 giorno	Inizio
Handler	Luca	Facile	Media	1 giorno	Inizio
Threads	Luca	Difficile	Massima	5 giorni	Inizio
System	Luca	Media	Massima	3 giorni	Inizio

Figura 2.14: Piano di lavoro

2.6 Progetto

2.6.1 Struttura

Device

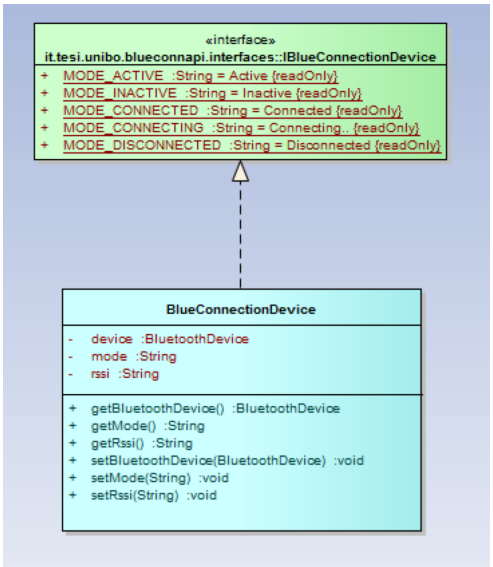


Figura 2.15: Progettazione di Device

BlueConnectionDevice è la classe grazie al quale possiamo ottenere tutti i parametri utili alla comunicazione e connessione di dispositivi. Il primo sarà il parametro che identificherà in Android il device con le proprietà che lo caratterizzano. Purtroppo tra queste non è presente RSSI in quanto coloro che hanno costruito tale libreria han pensato che tenere questo dato potesse essere troppo dispendioso a livello energetico e quindi un problema su dispositvi mobili. Il campo RSSI perciò dovrà essere ottenuto utilizzando un filtro apposta come spiegato di seguito nella sezione **Comportamento**. Infine, l'ultimo parametro sarà quello dedicato allo stato della comunicazione. Ci serve per capire se il *Device* è connesso o meno a un altro *Neighbor*. Questo parametro quindi è l'istanziazione astratta di un *Neighbor*, per cui nel System, avremo bisogno di una lista di BlueConnectionDevice per ottenere tutto ciò che ci serve da tutti i *neighbors*.

Thread

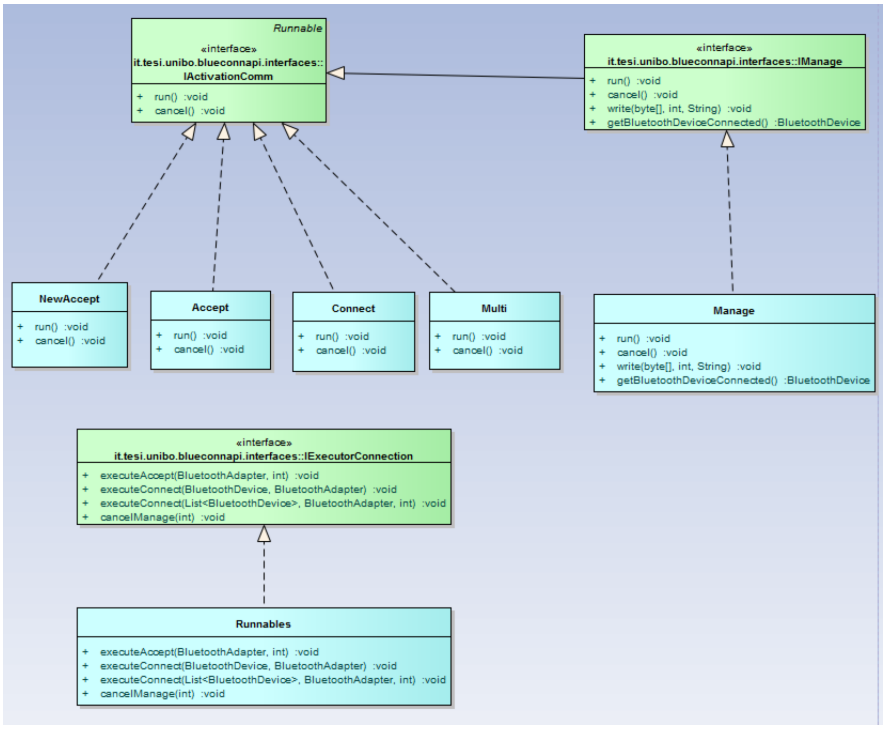


Figura 2.16: Progettazione di Thread

Questa parte di sottosistema serve esclusivamente per la parte di comunicazione. In un qualsiasi tipo di comunicazione ci sarà sempre qualcuno che si connette e qualcuno che accetta la connessione. Nel nostro caso la parte che si dedica alla connessione è riservata a:

- **Connect**: serve per connettersi a uno specifico BluetoothDevice remoto.
- **Multi**: serve per connettersi a più dispositivi remoti, forniti grazie a una lista di BluetoothDevice.

Per quanto riguarda Connect(), abbiamo bisogno di un altro costruttore dedicato solo per l'invio di messaggi broadcast. In questo costruttore si aggiuge un campo t*text* dedicato al messaggio broadcast che si vuole inviare.

Questo nuovo campo ci permette di effettuare una connessione singola a un dispositivo. Una volta istanziata un *thread Manage* dedicato questo provvederà a inviarlo e in quanto settato il campo di text dovrà automaticamente cancellarsi. Naturalmente verrà usato se e solo se non è già stata istanziata un thread Manage dedicato alla comunicazione con quel *neighbor*. Invece la parte che si dedica alla accettazione delle connessioni è riservata a:

- **Accept**: serve per accettare una singola connessione.
- **NewAccept**: serve per avviare un processo di ascolto per il numero massimo di dispositivi con cui si può tenere una connessione.

Una volta instaurata la connessione, questi *Thread* sono inutili, e si passa il controllo a **Manage** che ha il compito di mantener la connessione attendendo in caso di ricezione di qualche messaggio nel processo di *run()* che caratterizza la classe Java *Runnable*. Mentre la connessione è mantenuta il *Device* può mandare messaggi ai *neighbors*. Infine abbiamo **Runnables**, ossia una classe che si occupa di tenere delle funzioni per permettere al *System* di avviare i processi di inizializzazione e instaurazione della connessione.

Handler

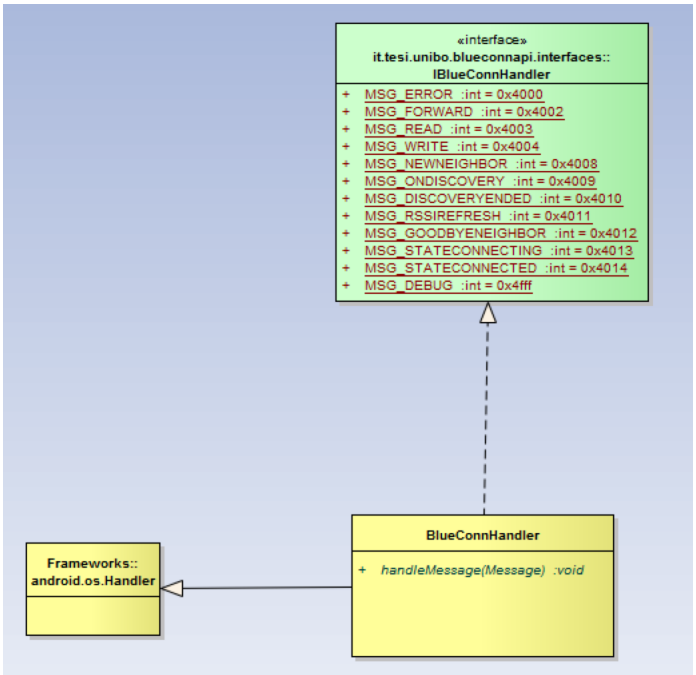


Figura 2.17: Progettazione di Handler

In questa fase analizzando ciò che è stato realizzato nella fase di analisi si è valutato che ai fini della progettazione siccome la libreria è basata su Java per Android, si ritiene che BlueConnHandler debba estendere la classe fornita dalle librerie definita come *android.os.Handler*. Questa classe si ritiene sia la più giusta in quanto fornisce l'esatta concezione di cosa inteso nella fase di analisi. Nella fase di *Interazione* verranno spiegati in dettaglio i significati singoli di ognuno delle costanti.

System

Per quanto riguarda **System**, si è pensato invece all'aggiunta di alcuni metodi che caratterizzino a fondo i passi di creazione della comunicazione attraverso la libreria per rendere più comprensibile ciò che si esegue.

- *checkEnabled()*: serve per controllare se *Bluetooth* è attivo o meno.

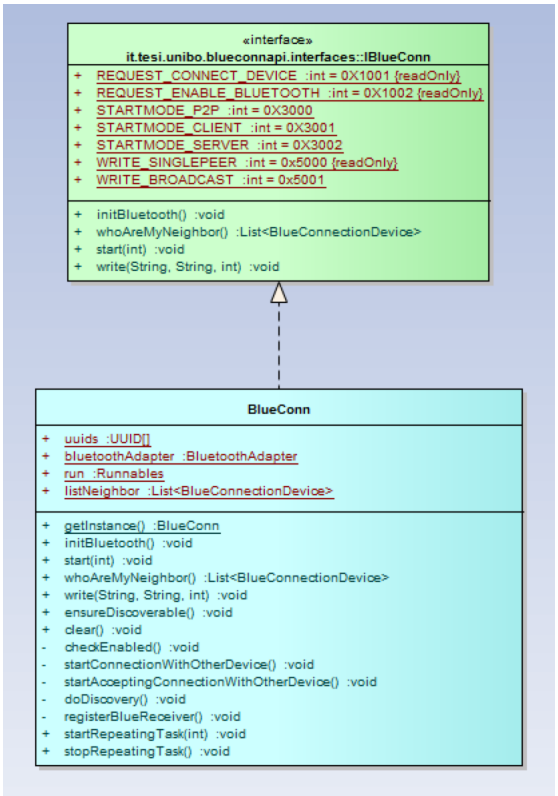


Figura 2.18: Progettazione di System

- *startConnectionWithOtherDevice()*/ *startAcceptingConnectionWithOtherDevice()*: sono metodi che servono per separare la parte client da quella server. La chiamata di questi metodi deve dipende strettamente dalla modalità con cui si chiama il metodo *start()*.

- *doDiscovery()*: serve per cominciare a scoprire i *Neighbors*.

- *registerBlueReceiver()*: si è scoperto che per gestire il Bluetooth è necessario creare un BroadcastReceiver definito nella libreria *android.content*. Utilizzando il metodo, si deve istanziare un BroadcastReceiver che si attivi alla ricezione di un certo tipo di broadcast.

- *startRepeatingTask(int) / stopRepeatingTask()*: grazie a queste due funzioni si aziona/ferma la *discovery* dei *neighbors* ogni certo numero di secondi definito dall'intero in ingresso a startRepeatingTask.

- *ensureDiscoverable()* : serve per abilità che rende il *Device* **discoverable** per gli altri device. In Android, un dispositivo è *discoverable* per pochi minuti se non impostato in maniera diversa nelle impostazioni del Bluetooth. Con questa modalità il *Device* è visibile ai *Neighbor*.

2.6.2 Interazione

L'*interazione* fra i vari sottosistemi avviene sia tramite l'esecuzione da parte del sistema dei comandi forniti da *IExecutorConnection* e sia tramite *IBlueConnHandler*. Il primo serve per far partire i *IActivationComm*, i quali si occupano dei diversi tipi di connessione. Una volta connessi il device e il *neighbor*, allora si deve passare a *IManage*. Una volta trovati in questo punto ci sono due strade:

- Il *device* riceve un array di byte

- Il *device* invia un array di byte

Nel primo caso si avviserà tramite il *IBlueConnHandler* che il dispositivo ha ricevuto un array di byte inserendo come tipo di messaggio il valore costante **MSG_READ**. Nell'altro caso viene notificato solo l'avvenuto avvio dell'array di byte con un messaggio di tipo **MSG_WRITE**. Le altre costanti invece servono per:

- **MSG_ERROR**: In una delle parti del sistema è stato riscontrato un errore e viene segnalato.

- **MSG_FORWARD**: Notifica di inoltro di un messaggio

- **MSG_NEWNEIGHBOR**: Notifica l'entrata in lista di un nuovo *Neighbor*

- **MSG_ONDISCOVERY**: Notifica che il *Device* è in discovery mode

- **MSG_DISCOVERYENDED**: Notifica la fine della discovery mode.

- **MSG_RSSIREFRESH**: Notifica e aggiorna l'attuale RSSI dei *neighbor*.
- **MSG_GOODBYENEIGHBOR**: Notifica se un *neighbor* si è disconnesso.
- **MSG_CONNECTING**: Notifica che il *Device* sta provando a connettersi a un *neighbor*.
- **MSG_CONNECTED**: Notifica o che una connessione è stata accettata o che una connessione è avvenuta con successo.
- **MSG_DEBUG**: Notifica tutti i messaggi di debug per rendere più chiaro il processo della libreria.

2.6.3 Comportamento

Il **comportamento** fra i vari sottosistemi avviene in maniera simile a quanto visto nella fase di analisi del problema. Si è riscontrata però una certa limitazione che ogni dispositivo Bluetooth possiede. Infatti ogni *Device* può essere connesso a un massimo di 7 dispositivi formando così una *piconet*. Per evitare casi limite, si consiglia che il dispositvo se ha un numero uguale a 7 connessioni e un altro *Neighbor* si vuole connettere o ci si vuole connettere a un altro *Neighbor*, è consigliabile che si disconnetta una delle connessioni più vecchie per dar spazio a quella nuova. In questo modo, se viene inviato un messaggio broadcast o deve essere ricevuto un messaggio, le connessioni più vecchie vengono tolte per lasciar spazio a nuove connessioni, evitando di generere errori dovuti a Bluetooth stessi.
Inoltre bisogna evidenziare che il *Device* deve essere costantemente in Discovery di altri *Neighbors*. A tale proposito si deve registrare un *Broadcast receiver* per gestire dinamicamente se per caso un nuovo *Neighbor* è entrato nel range. Questo Broadcast receiver deve essere registrato con 4 filtri chiamati in Android, *IntentFilter*, che sono:

- **BluetoothDevice.ACTION_FOUND:** serve per trovare i dispositivi che hanno attivi Bluetooth nel range del *Device*
- **BluetoothDevice.ACTION_UUID**: serve per ottenere i servizi offerti da un *device remoto*. Se non sono gli stessi impostati per la libreria allora viene scartato

- **BluetoothAdapter.ACTION_DISCOVERY_FINISHED**: la discovery mode è finita
- **BluetoothAdapter.ACTION_DISCOVERY_STARTED**: la discovery mode è iniziata

Il problema in Android è che la *Discovery mode* dura massimo 12 secondi. Per cui servirà un *Handler* che comunichi al dispositivo di tornare in *Discovery mode* ogni certo numero di secondi, che deve essere più grande di 12 altrimenti sarebbe sempre inutile e non troppo grande per evitare lunghe attese per aggiornare il *device* sul arrivo di un nuovo *neighbor*. Altro problema inoltre è che la risposta ricevuta dal *Broadcast Receiver* per i servizi offerti da un dispositivo remoto arriva solo quando la discovery è finita. Per ovviare a ciò si può caratterizzare un tempo adeguato che permetta al dispositivo di fare la discovery e nel tempo rimanente fare il resto.

2.7 Implementazione

Work Plan

	Project Lead (who)	Difficoltà	Priorità	ECT	Progress
Device	Luca	Facile	Minima	1 giorno	Fine
Handler	Luca	Facile	Media	1 giorno	Fine
Threads	Luca	Difficile	Massima	5 giorni	Fine
System	Luca	Media	Massima	3 giorni	Fine

Figura 2.19: Completamento piano di lavoro

L'implementazione è stata rispettata nei tempi indicati nel **Piano di Lavoro** (vedi paragrafo 2.5). Per implementare è stato necessario utilizzare librerie Java e Android fornite dal tool di sviluppo ADT (Android Development Tools). Seguendo passo passo le istruzioni lasciate dal progettista si è arrivati a definire un primo prototipo della libreria per comunicazioni Bluetooth tramite Android.

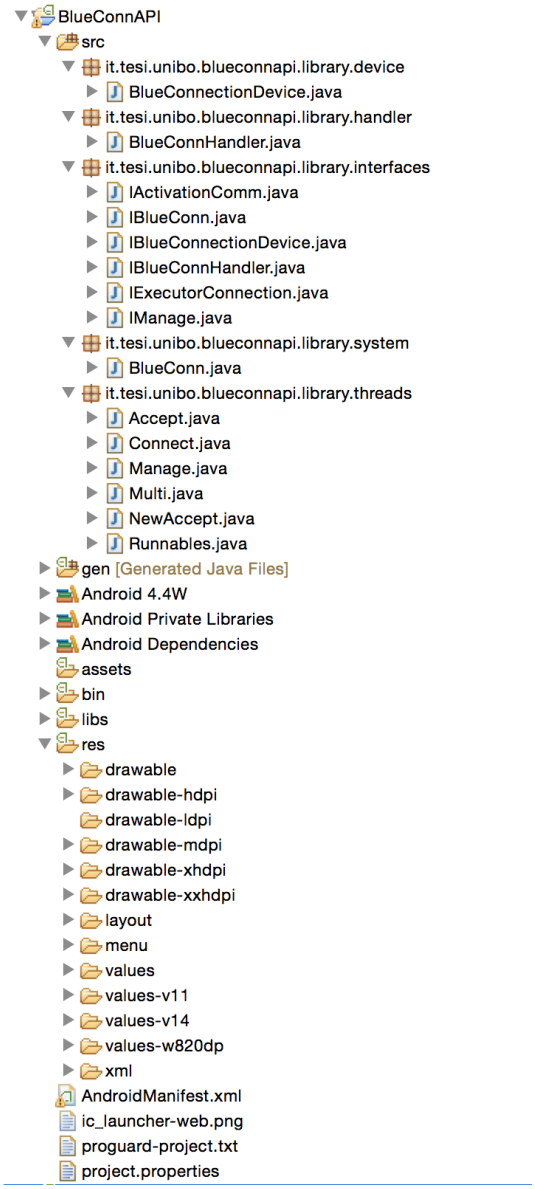


Figura 2.20: BlueConnAPI Eclipse Android Project Workspace

In seguito verranno elencate le aggiunte fatte a livello di programmazione per rendere il lavoro fatto nella parte di **Analisi del problema** e nella parte di **Progetto** più efficace per il linguaggio Java per Android. Sono state aggiunte soprattutto per necessità di coloro che vorranno cimentarsi con questa libreria. L'intento di questa libreria è quello di lasciare un sistema in grado di essere flessibile ad ogni tipologia di problema coinvolta con le comunicazioni Bluetooth.

- Il doppio costruttore di **Connect** è stato realizzato come segue:

```
private void registerBlueReceiver() {
    // Register for broadcasts when a device is discovered
    IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
    filter.addAction(BluetoothDevice.ACTION_UUID);
    filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
    filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_STARTED);
    context.registerReceiver(mReceiver, filter);
}

public void unregisterBlueReceiver(){
    context.unregisterReceiver(mReceiver);
    //context.unregisterReceiver(mRssiUpdateReceiver);
}
```

Figura 2.21: Doppio costruttore per Connect

- Per quanto riguarda il discorso del numero massimo di connessioni, in *Runnables* è stato realizzato un semplice algoritmo che prende la connessione più vecchia e la scarta.

```
@Override
public void executeConnect(BluetoothDevice d,BluetoothAdapter a) {
    //executor.execute(new Connect(d, value));
    if(manageThreads.size()>=7){
        manageThreads.get(0).cancel();
    }
    connectSThread = new Connect(d,a,0,handler,executor);
    executor.execute(connectSThread);
}
```

Figura 2.22: Esempio algoritmo per eliminazione connessione più vecchia

- All'interno del *BroadcastReceiver* si è aggiunto un Comparator per ordinare la lista in ordine decrescente in base al RSSI:

```
Comparator<BlueConnectionDevice> c= new Comparator<BlueConnectionDevice>() {

    @Override
    public int compare(BlueConnectionDevice lhs,BlueConnectionDevice rhs) {
        // TODO Auto-generated method stub
        int rssi1= Integer.parseInt(lhs.getRssi());
        int rssi2=Integer.parseInt(rhs.getRssi());

        return rssi2-rssi1;
    }

};

Collections.sort(listNeighbor,c);
handler.obtainMessage(BlueConnHandler.MSG_RSSIREFRESH).sendToTarget();
```

Figura 2.23: Compator di RSSI per ordine decrescente

- Come indicato nella progettazione, il BroadcastReceiver registra gli IntentFilter. E' stato aggiunto anche una funzione *unregisterBlueReceiver()* per evitare di incorrere in errori per la mancata deregistrazione.

```
private void registerBlueReceiver() {
    // Register for broadcasts when a device is discovered
    IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
    filter.addAction(BluetoothDevice.ACTION_UUID);
    filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
    filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_STARTED);
    context.registerReceiver(mReceiver, filter);
}

public void unregisterBlueReceiver(){
    context.unregisterReceiver(mReceiver);
    //context.unregisterReceiver(mRssiUpdateReceiver);
}
```

Figura 2.24: IntentFilter aggiunti prima della registrazione

- Testando la libreria, si è notato che il tempo migliore per permettere al dispositivo di fare tutto è 15 secondi.

  Queste ultime due caratteristiche aggiunte rendono il nostro sistema dinamico e capace di adattarsi a qualsiasi situazione in quanto se un dispositivo si spegne o esce dal nostro range diventa *Inactive* e posto in fondo alla lista. Inoltre è possibile grazie all'uso di un

```
void startRepeatingTask(int mInterval) {
    this.mInterval= mInterval*1000;
    mStatusChecker.run();
}

void stopRepeatingTask() {
    mHandler.removeCallbacks(mStatusChecker);
}
```

Figura 2.25: Si è settato il refreshTime a 15

2.8 Rilascio

La libreria viene rilasciata come file .jar o come progetto Android. Per l'utilizzo di questa libreria, basterà importarla nel progetto. Una volta importata, si istanzierà un oggetto di tipo **BlueConn**. Eseguendo il metodo start(), la libreria comincerà a cercare *Neighbors*. Naturalmente viene fornita anche la documentazione con cui consultare opportunamente i metodi.

Capitolo 3

Scenari applicativi

In questa sezione vengono trattati tutti i possibili utilizzi della libreria di cui nel capitolo precedente ho mostrato il processo di sviluppo. Anticipo che per questo tipo di applicazione, gli scenari sono pressochè infiniti in quanto le comunicazioni sono alla base della società moderna in cui viviamo. Il primo scenario è dedicato a un'applicazione che ho personalmente creato basandomi su tale libreria. La seconda è riferita a un'applicazione che comunica sempre attraverso la libreria ma questa volta con un dispositivo Android. Il resto sono possibili sviluppi futuri che si possono realizzare basandosi sulla libreria stessa.

3.1 Chat bluetooth

Questa applicazione si basa sulla libreria creata nel capitolo precedente. E' una semplice applicazione usata come test per verificare il corretto funzionamento di ogni aspetto della libreria. Iniziamo quindi installando l'APK.

3.1.1 Attivazione bluetooth e discovery

Una volta installato e aperto, potrebbe capitare che il Bluetooth è disattivato. In tal caso come previsto nella libreria si attiverà la funzione *checkEnabled()*. Se è attivo andrà avanti altrimenti automaticamente ci chiederà di voler attivare il Bluetooth. Una volta abilitato il Bluetooth, come prima azione clicchiamo sul bottone azzurro **Start** che si trova in basso. Incominceremo a vedere una *ProgressBar* che incomincia a girare in alto a destra come mostrato in figura: Questo indica che il nostro *Device* è entrato in

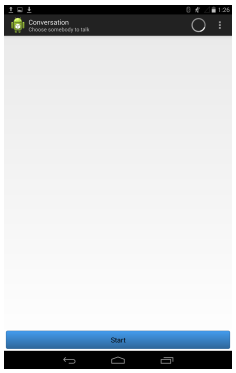


Figura 3.1: MessageActivity

modalità *discovery*. Secondo *Android*, il tempo che impiega a trovare un dispositivo è minimo 2 cicli di discovery in quanto alla prima identifica l'indirizzo e nella seconda riempe una *Hashmap* con il nome del dispositivo. Quindi si aspetterà una quantità finita di tempo prima che il dispositivo ci mostrerà che ha trovato alcuni dispositivi come mostrato in figura:

I 3 dispositivi fanno parte di una lista la quale è stata rappresentata all'interno della *ListView* usando un *Adapter* personalizzato. Il valore in alto a sinistra di ogni elemento rappresenta il nome del device. Sotto troveremo il MAC address del Bluetooth. In alto a destra avremo lo stato della

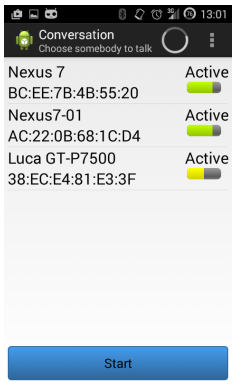


Figura 3.2: Abbiamo trovato 3 dispositivi nel range

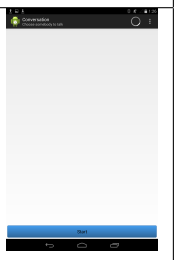
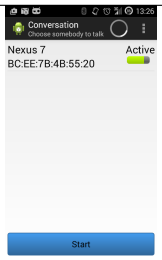
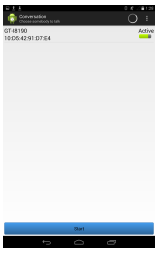
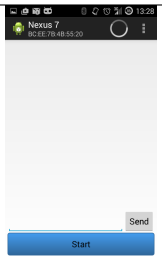
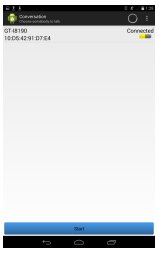
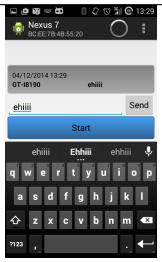
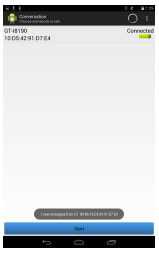
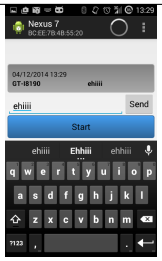
comunicazione fra i due dispositivi e infine sotto di questo avremo una *ProgressBar* orizzontale personalizzata che ci dà il livello della distanza a cui ci troviamo dall'altro dispositivo. Questa può avere diversi colori partendo dal verde, passando per il giallo e finendo col rosso per indicare rispettivamente se ci si trova vicini, a media distanza o lontani dall'altro device. Una volta trovati, possiamo pensare di voler inviare un messaggio. Come pensato nella libreria, i messaggi possono essere inviati:

- **BROADCAST**: cioè vengono inviati a tutti i dispositivi vicini
- **DIRETTI**: cioè viene inviato a un *neighbor* vicino.

Analizzeremo di seguito i vari casi.

3.1.2 Invio Messaggi Diretti

L'invio di messaggi diretti è il più semplice in quanto usa le regole base della comunicazione. Una volta ottenuta la lista dei vicini si deciderà quale dei vicini è quello a cui desideriamo inviare il messaggio. Nela tabella sottostante vengono rappresentate le azioni in simultanea di due dispositivi che usano la chat per messaggi singoli. S3 Mini è colui che si connette mentre Nexus 7 è quello che accetta. Il pulsante *start* è stato premuto in tutti e due.

Azione	S3 Mini	Nexus7
1) Ricerca dispositivo	Uguale a Nexus	
2) Dispositivo trovato		
3) Connessione		
4) Invio messaggio		
5) Lettura messaggio		

3.1.3 Invio Messaggi Broadcast

Ci troviamo in questa situazione: abbiamo effettuato la discovery e aggiornato di continuo gli RSSI dei dispositivi. Decidiamo perciò di inviare un

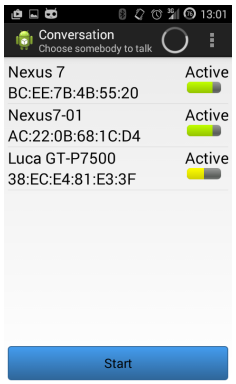


Figura 3.3: Inizio

messaggio broadcast a tutti i vicini. Clicco sul menu a tendina e in seguito su *Send All*. Una volta fatto ciò comparirà questo:

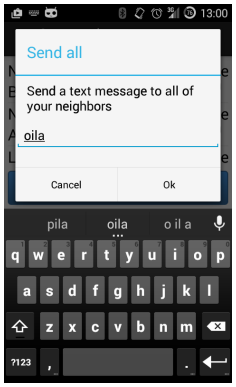


Figura 3.4: Scrivi il messaggio da inviare a tutti

Inviamo e noteremo che il nostro dispositivo si connetterà uno alla volta a ogni vicino:

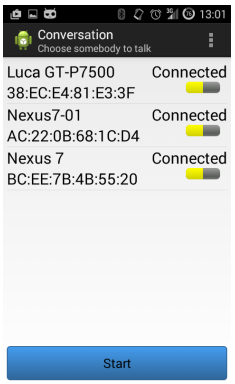


Figura 3.5: Connessione e invio messaggio effettuaro

Intanto se controlliamo nei dispositivi vicini notiamo che è arrivato il messaggio come dimostrano le immagini di seguito.

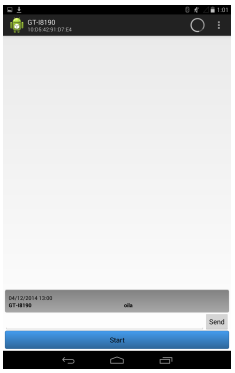


Figura 3.6: Asus Nexus

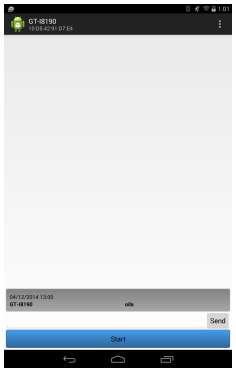


Figura 3.7: Asus Nexus 7-01

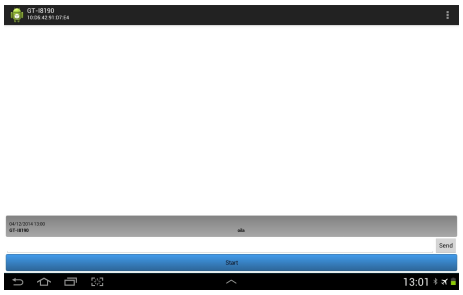


Figura 3.8: Samsung Galaxy Tab 10.1

3.2 Altri scenari

3.2.1 FireChat

Firechat è un caso reale di possibile scenario per l'uso di questa libreria. Agli inizi di ottobre a Hong Kong ci sono state proteste contro le restrizioni imposte dal governo cinese. I manifestanti per evitare queste restrizioni e controlli della rete Internet ha usufruito di FireChat, l'app di messaggistica istantanea e anonima, spesso confusa con una delle tante alternative a WhatsApp o WeChat, che permette ai ragazzi in piazza di comunicare senza troppi problemi e in assenza di copertura mobile o connessione internet. L'app non è nuova: arriva da San Francisco, è nata qualche mese fa su iOS ed è da poco presente anche in Google Play. Si tratta di un servizio che sfrutta il Multipeer Connectivity Framework – ovvero un sistema che è stato introdotto da Apple con iOS 7 – che permette di creare diversi ponti di comunicazione tra dispositivi, usando il Bluetooth e il Wi-Fi. In pratica, il Multipeer Connectivity Framework usa il sistema mesh network, ovvero una rete "a maglia" con cui i messaggi possono viaggiare da un telefono all'altro tramite il supporto a Bluetooth e Wi-Fi, permettendo di creare una catena formata da diversi device connessi tra loro: ovunque ci sia gente che usa FireChat, si viene quindi a creare una sorta di rete "locale" dove chiunque può funzionare da server o da client, in modo da far arrivare i messaggi a destinazione. Un'altra particolarità di questo sistema è che si espande ogni volta che un nuovo utente accede al servizio. Attualmente, sono più di 100mila gli attivisti che hanno scaricato l'applicazione in una sola notte su iOS e Android, ed almeno 33mila quelli che la usano con una certa frequenza. Un caso analogo si era registrato negli scorsi mesi in alcune province dell'Iraq, dove gli abitanti avevano cominciato a usare FireChat per sfuggire ai limiti imposti dal governo, che aveva chiuso le connessioni internet e bloccato gli accessi ai social. Offre tutti i servizi delle classiche chat di messaggistica istantanea con il surplus del funzionamento anche senza connessione ad Internet. Il sistema, infatti, si adatta perfettamente allo spirito dei manifestanti, dal momento che, come un movimento, non viene imposto dall'alto, ma trova la sua forza dalle migliaia di piccole connessioni che vengono dal basso.

3.2.2 Ospedali

Si pensi a questo scenario in maniera tale che per ogni stanza ci sono dei device abilitati che monitorano le condizioni dei pazienti. Il dottore entra e con il suo tablet riesce a controllare i dati relativi del paziente semplicemente avvicinandosi valutando i dati attuali. Questo scenario è semplice realizzarlo con la libreria sviluppata. Semplicemente, il device del dottore è sempre in discovery mentre i device dei pazienti sono in attesa sempre visibili. Quando il tablet del dottore si avvicina a una certa distanza inferiore a un parametro definito allora si connette automaticamente al dispositivo del paziente e visualizza i dati in tempo reale. Questa soluzione eviterebbe la stampa di fogli per le cartelle cliniche e una più semplice consultazione dei dati.

3.2.3 Sistemi di domotica

Voler spegnere la luce dal divano con il proprio smartphone è il sogno di tutti dopo una lunga e faticosa giornata. Usando la libreria per scoprire tutti i dispositivi nel raggio di azione del Bluetooth è possibile fare tutto ciò. Si ha a disposizione ad esempio una serie di moduli Bluetooth che controllano le luci della casa. Infatti semplicemente connettendosi a questi moduli e inviando un segnale di attivazione, le luci si accenderanno. Oppure per controllare la temperatura di casa e accendere il riscaldamento se troppo freddo. Riassumendo, qualsiasi cosa che si può attivare o che deve restituire dati piccoli, rende Bluetooth molto utile in quanto non consuma molto e per il raggio di segnale che ha riesce a coprire gran parte della casa.

3.2.4 Veicoli

Ogni dispositivo Android ha un MAC Address unico per il Bluetooth che lo identifica. Se abbiamo a disposizione un veicolo che registra tale MAC Address, allora possiamo fare in modo che quando compare nel suo range di azione e il MAC Address combacia allora se si avvicina a una certa distanza, la macchina si apre altrimenti se si allontana, la macchina si chiude. Sempre con questo metodo si possono verificare le distanze dagli altri veicoli e assistire la frenata in caso di disattenzione o in caso una macchina si avvicini velocemente segnalare al conducente che una macchina si appresta a superare e ti prestare attenzione. Questo è molto utile in caso di scarsa visibilità. Essendo poi in campo aperto, Bluetooth non soffre di eccessivi

disturbi. Oppure si immagini un trasmettitore Bluetooth installato a ogni semaforo che invia i dati sul tempo rimanente da aspettare prima di partire e sul suo stato attuale. Un altro scenario possibile è quello dei parcheggi. Vogliamo parcheggiare in centro anche se sappiamo che a quell'ora non si trova un posto. Prima di infilarci senza motivo in un parcheggio, magari questo ha un trasmettitore che riconosce la macchina e gli invia un messaggio con la notifica dell'esaurimento dei posti, evitando così di girare invano nel parcheggio e risparmiando tempo.

3.2.5 Emergenze

Se ci si trova in piazza e abbiamo installata un'applicazione che tramite Bluetooth si rende visibile e nella stessa piazza si trova qualcuno con problemi cardiaci con un monitor collegato in Bluetooth allo smartphone. Se per caso succede qualcosa e il programma che oltre ad avere algoritmi che capiscono in tempo reale da monitoraggio se il paziente ha un attacco cardiaco è in fase di discovery in quanto cerca dispositivi *Helper*, ossia disponibili per aiuti, è facile capire come ci metta un attimo a notificare il pericolo a tutti gli*Helper* circostanti con un messaggio broadcast e chiamare direttamente il 118.

Capitolo 4

Conclusione

4.1 Conclusione

L'applicazione *ChatBluetooth*, basata sulla libreria creata, funziona correttamente e, come mostrano gli screenshot nel *capitolo 3.1*, rispetta i vincoli imposti, ossia:

- Crea una lista dinamica ordinata in base la distanza dei dispositivi che si aggiorna a ogni discovery effettuata
- Crea un layer P2P che maschera la natura Master-Slave delle connessioni Bluetooth
- Invia messaggi sia singoli che broadcast

Tuttavia anche se rispetta tali vincoli, bisogna evidenziare alcune problematiche che a volte capitano durante l'esecuzione dell'applicazione *ChatBluetooth*. In particolare:

- se il Bluetooth viene lasciato attivo per più di 10 minuti, il processo interno di Android che gestisce Bluetooth(*com.android.bluetooth*) e la Condivisione Bluetooth riporta un errore e, anche se non fa chiudere l'applicazione, fa cadere le connessioni attive in quel momento.

- si è trovata come una limitazione quella che durante la fase di discovery dei dispositivi, il processo di connessione o è lento o non viene effettuato, mentre se la fase di discovery è completata questa avviene in maniera relativamente rapida. Android spiega sul sito ufficiale che questo è dovuto al fatto che la discovery mode utilizza in maniera ampia la banda del segnale Bluetooth, e che quindi qualsiasi azione compiuta che usi in maniera altrettanto ampia la banda come la connessione a un altro dispositivo o non viene eseguita o impiega molto tempo. Per risolvere questo problema, la fase di discovery, che per dispositivi Android dura 12 secondi, viene ripetuta attraverso un handler ogni 15 secondi lasciando uno spazio giusto tra la terminazione di una discovery e l'inizio di un'altra per permettere alle connessioni di andare a buon fine.

- si è riscontrato che due dispositivi diversi impiegano tempi diversi per vedersi come vicini. Testando su più dispositivi, si è notato come alcuni siano molto lenti a vedere dispositivi con stessi servizi rispetto ad altri. Probabilmente questo può essere dovuto al fatto che alcune case produttrici risparmiano sull'utilizzo dei moduli Bluetooth da usare per i dispositivi mobili, puntando a valorizzare di più su altri aspetti. Generalmente nel mio caso, mentre alcuni dispositivi impiegavano uno/due cicli di discovery per vedere tutti i dispositivi vicini, altri impiegavano più cicli con un tempo di risposta di circa 1-2 minuti superiore rispetto altri dispositivi.

4.2 Ringraziamenti

Ringrazio i miei genitori che in questi anni difficili mi hanno dato la forza di continuare a credere nei miei sogni e hanno continuato a sostenere i miei studi facendo enormi sacrifici. Anche se non totalmente spero di averli ripagati con questo risultato a sufficienza per la fiducia e la convinzione che mi hanno dato.
Ringrazio mia sorella Sonia, che anche se nell'ultimo anno si è trasferita, mi ha fatto sempre capire il bene che mi vuole e anche se ogni tanto durante questi anni mi ha fatto arrabbiare, le ho sempre voluto molto molto bene.
Ringrazio Diana che mi ha fatto capire quanto vale questa vita e quanto preziosa può essere. Mi ha fatto capire ciò che conta veramente e nonostante non ci sia più credo che in questo momento mi stia aiutando ad andare avanti e a continuare a credere in me stesso. E' stato difficile andare avanti senza di lei ma non smetterò mai di ringraziarla per tutto ciò che ha fatto per me.
Ringrazio i miei parenti a cui voglio un gran bene che mi sono stati sempre vicini e che immagino saranno contentissimi di questo risultato. Ringrazio i miei tantissimi cugini per i bellissimi momenti passati insieme. In modo particolare voglio ringraziare Enzo che durante questi anni mi ha ispirato e aiutato a intraprendere la strada di Ingegneria Informatica e Sara, che per me è come una sorella per i tantissimi momenti passati insieme.
Ringrazio il mio amico storico Marco per gli ormai più di 8 anni di allenamenti, partite, FIFA, PES, COD, LoL, serate, estati e viaggi leggendari sempre insieme come due fratelli.
Ringrazio Mara per tutto quello che ha fatto per me in tutti questi anni che ci conosciamo e per ogni parola che mi ha fatto sempre star meglio. E' una ragazza importante e lo sarà sempre per me.
Ringrazio Giorgia e Cristina che insieme a Marco mi hanno accompagnato in università per avermi sopportato ogni mattina e ogni sera negli ultimi anni.
Ringrazio sempre Marco ma anche Erick, Federico e Nicola per le serate devasto che mi sono servite a smaltire le fatiche e le sofferenze degli ultimi anni in cui tra problemi personali, esami, corsi e infine tesi avevo il cervello fuso e mi hanno aiutato a recuperare le batterie.
Ringrazio tutti i miei compagni di corso e persone conosciute in facoltà, alcuni che già conoscevo dal liceo come Alessandro e Matteo, e altri che

invece ho conosciuto e con cui ho stretto un legame di amicizia e rispetto che considero molto importante.
Ringrazio inoltre tutte quelle persone che ho conosciuto durante la mia vita, che ancora ci tengono a sentirmi e che hanno sempre creduto in me.
Infine ci tengo a ringraziare il mio relatore, il prof. Mirko Viroli, che mi ha aiutato in ogni fase di progetto oltre che sopportandomi per due mesi ogni giovedì a ricevimento, anche dandomi in custodia mezzi e risorse per lo sviluppo della tesi.
Ci tengo a condividere il merito di questa tesi e di questo momento felice con tutte queste persone perchè come si dice:
"Qualsiasi cosa farai nella vita, non sarà davvero leggendaria, se non ci saranno i tuoi amici o i tuoi cari a viverla con te".

Bibliografia

[1] Bluetooth SIG, Sito ufficiale Bluetooth, `http://www.bluetooth.com/Pages/Bluetooth-Home.aspx`

[2] Bluetooth SIG, CoreSpecification Bluetooth, `https://developer.bluetooth.org/TechnologyOverview/Pages/core-specification.aspx`

[3] Wikipedia, `it.wikipedia.com`

[4] AOSP, Sito ufficiale Android Developers, `developer.android.com/`

[5] Wikipedia, `it.wikipedia.com`

[6] Wikipedia, `it.wikipedia.com`

[7] Wikipedia, `it.wikipedia.com`

[8] Antonio Natali Ambra Molesini, *Costruire sistemi software: dai modelli al codice*, II edizione, Ottobre 2009, Progetto Leonardo