ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Scuola di Ingegneria e Architettura Campus di Cesena
Corso di Laurea in Ingegneria Elettronica, Informatica e Telecomunicazioni

DISCOVERY DI DISPOSITIVI EMBEDDED E RACCOLTA DATI CON BLUETOOTH E BLUETOOTH SMART

Elaborato in
FONDAMENTI DI INFORMATICA A

Tesi di Laurea di
GIACOMO RANIERI

Relatore
Prof. MIRKO VIROLI

Anno Accademico 2013 / 2014
2ª Sessione di Laurea

I do not fear computers. I fear the lack of them.
Isaac Asimov

Indice

Introduzione

Con il passare degli anni l'uomo si accorge sempre più di essere circondato da informazioni. Molte di queste sono a lui nascoste. La conoscenza di tutte queste informazioni permette di migliorare la qualità della vita. Sempre più dispositivi ci permettono di ricevere queste informazioni, di renderle vive e disponibili.
È importante non solo avere a disposizione queste informazioni, ma anche riceverle nel momento giusto. Non solo il "cosa" ma anche il "quando". Queste due dimensioni permettono di sfruttare al meglio ciò che otteniamo per velocizzare le attività quotidiane.
Le tecnologie wireless assumono un ruolo di grande importanza in questo campo. Ricevere queste informazioni senza ingombri, in mobilità, rappresenta la chiave per velocizzarne l'utilizzo. Nel mondo odierno la diffusione di queste tecnologie negli smartphone e nei tablet aiuta la visione del progetto. Tutti potremo essere in grado di ricevere informazioni chiave quando ci serviranno.

Questo è Pervasive Computing, noto anche come ubiquitous computing, risultato dello sviluppo esponenziale delle tecnologie, porta qualunque cosa integrabile con un chip, anche il corpo umano e non solo oggetti, ad essere connessa, fino ad arrivare ad una connettività discreta e sempre disponibile. L'ambiente che ci circonda sarà quindi in grado di elaborare informazioni che poi attivamente ci fornirà. Per fare un esempio: supponiamo di arriviamo al supermercato con la lista della spesa sullo smartphone, potremo ricevere sul device informazioni su sconti, prodotti che cerchiamo e dove trovarli all'interno del negozio.

Ogni oggetto quotidiano può integrare queste tecnologie, da un maglia ad un prodotto del banco alimentari, al paio di occhiali da vista che siamo soliti indossare.

Questo è il cosiddetto Internet of Things, Internet delle cose, che incarna a pieno il modello del Pervasive Computing. Chi non ha mai perso

un oggetto in casa, la posizione dell'oggetto è anche questa un informazione importante, e con l'Internet of Things saranno le "cose" a dirci dove si trovano, a comunicare tra loro per aiutarci nella vita quotidiana. È da scenari come quelli appena descritti e altri come il soccorso o le smart cities che si delinea questa tesi.
Verranno introdotte tecnologie per la comunicazione a corto raggio quali Bluetooth e Bluetooth Smart di cui saranno analizzate le caratteristiche peculiari. Si analizzerà inoltre il mondo dei dispositivi embedded, e più in particolare il Raspberry Pi, la nota piattaforma per l'esplorazione del mondo dei computer e dei linguaggi di programmazione. Verrà esposta una panoramica dello stato dell'arte dei sensori per la piattaforma Raspberry Pi, trattando in modo più approfondito la piattaforma sensoristica eHealth v2.0, per il monitoraggio di parametri biometrici. Sarà introdotta inoltre la piattaforma Android per smartphone e tablet come sistema di riferimento per l'ambito mobile.

Lo scopo della tesi è quello poi di produrre un prototipo utilizzando le tecnologie appena descritte, che sia in grado di ottenere dati da un insieme di sensori per poterli poi trasmettere all'utente, in modo che esso sia maggiormente cosciente del mondo che lo circonda. Affronteremo la sfida in uno scenario medico/di soccorso, dove un operatore si avvicinerà ad un gruppo di pazienti con l'intenzione di ottenere i parametri vitali di uno di essi. Ognuno di questi pazienti sarà dotato di un dispositivo in grado di rilevare i parametri e inviarli. Il medico con l'ausilio di uno smartphone o tablet dovrà essere in grado di scegliere di quale paziente nelle vicinanze ricevere le informazioni.
Le sperimentazioni che hanno portato poi alla realizzazione del prototipo hanno relegato al solo ruolo nella localizzazione della tecnologia Bluetooth Smart, lasciando invece la trasmissione dei dati al Bluetooth classico. La piattaforma Android è stata scelta come riferimento per l'operatore per la sua vasta diffusione e le vaste possibilità fornite.

Panoramica dei contenuti

Nel primo capitolo di questa tesi si documenteranno le tecnologie alla base del prototipo in seguito realizzato. Si affronteranno Bluetooth e Bluetooth LE (o Smart) , Raspberry Pi, la piattaforma e-Health v2.0 e Android. All'interno del secondo capitolo analizzeremo invece il progetto. Verranno trattati i requisiti e gli scenari applicativi utili alla comprensione del prototipo che si andrà a realizzare. Nel terzo ed ultimo capitolo

ci occuperemo della descrizione del sistema realizzato, dalla rappresentazione del dominio, alla realizzazione del prototipo e infine alla sua analisi. Motivando le scelte implementative realizzate.

Capitolo 1

Le tecnologie alla base

1.1 Bluetooth

Bluetooh [1] è una tecnologia wireless standard per il trasporto dati a corta distanza, che usa onde corte della banda 2.4 -2.485 Ghz. È uno degli standard maggiormente usati per la creazione di reti personali (PAN: Personal Area Network).
Lo sviluppo di Bluetooth è stato inziato nel 1994 dall'azienda di telecomunicazioni Ericsson, e prosegue ora sotto la guida del Bluetooth SIG (Special Interest Group) di cui fanno parte altre società del settore informatico e delle telecomunicazioni. Il nome Bluetooth è ispirato a Harald Blåtand re di Danimarca il cui nome in inglese è Harold Bluetooth. Il logo è invece composto da rune.
Lo standard Bluetooth è descritto all'interno del documento IEEE 802.15.1, ma non è più mantenuto. Da quando è stato rilasciato nel 1999 ha subito varie modifiche che ne hanno incrementato funzionalità, robustezza e diminuito i consumi energetici. L'ultima versione dello standard è la 4.1, un evoluzione incrementale dello standard 4.0 che vedremo successivamente [2].
I dispositivi che comunicano tramite Bluetooh vengono suddivisi in classi in base alla potenza di trasmissione:

Classe	Potenza Massima		Portata
	(mW)	(dBm)	(m)
1	100	20	100
2	2.5	4	10
3	1	0	1

Per la comunicazione vengono usati diversi protocolli a seconda del profilo Bluetooth utilizzato. A livello Host tra i più importanti troviamo [3]:

RFCOMM (Radio Frequecy Communication) fornisce un emulazione della porta seriale RS-232. Si presenta all'utente sotto forma di flusso di dati, come TCP. È il protocollo maggiormente utilizzato a causa del grande sostegno e della disponibilità di API per molti sistemi operativi.

SDP (Service Discovery Protocol) permette ai dispositivi di scoprire quali servizi supportano l'un l'altro. Ogni servizio è identificato da un UUID (Universal Unique Identifier) in formato 16 bit per i servizi (profili Bluetooth) definiti dal SIG e in formato 128 bit per gli altri.

AVCTP/AVDTP (Audio/Video Control—Data Transport Protocol) usati per controllare e trasmettere flussi audio/video

ATT (Low Energy Attribute Protocol)è usato in Bluetooth 4.0 in modo similare a SDP, ma semplificato e adattato per il low energy. Permette la lettura e la modifica di certi attributi esposti dal server.

OBEX (Object Exchange)che permette un facilitato scambio di oggetti binari tra dispositivi.

L2CAP (Logical Link Control and Adaptation Protocol)si occupa del multiplexing dei protocolli superiori, segmentazione e riassemblaggio pacchetti. Il collegamento che instaura è connectionless. Tutti i protocolli visti sopra hanno come livello sottostante L2CAP.

La definizione delle possibile applicazioni e dei comportamenti che dovrebbero tenere i dispositivi avviene mediante l'uso di profili. La specifica di ogni profilo [4] contiene svariate informazioni quali:

- Dipendenze da altri formati
- Suggerimenti per il formato dell'interfaccia utente
- Le parti dello stack dei protocolli usate e particolari opzioni o parametri per ogni layer dello stack.

Alla base di tutti i profili c'è il Generic Access Profile (GAP) che definisce il modo in cui i dispositivi si trovano e stabiliscono una connessione.
Ci sono profili per una moltitudine di applicazioni.

A2DP/VDP/AVRCP sono i profili usati quando si interagisce con stream audio/video. A2DP (Advanced Audio Distribution Profile) che tratta la parte audio, come può essere per esempio l'ascolto di musica tramite cuffie Bluetooth, VDP (Video Distribution Profile) che tratta la parte video, un esempio è lo streaming da un media center o live da una videocamera, questi due profili si basano su GAVDP (Generic Audio/Video Distribution Profile). AVRCP (Audio Video Remote Control Profile) è invece il profilo per il controllo del riproduzione.

GATT (Generic Attribute Profile)fornisce la funzionalità di scoperta dei profili e la descrizione dei servizi per Bluetooth Low Energy. Definisce come sono raggruppati gli attributi ATT per formare dei servizi.

SPP (Serial Port Profile)è uno dei profili più flessibili, basato su RFCOMM emula una porta seriale RS-232. È base anche di alcuni profili come AVRCP.

L'utilizzo di un profilo però non è legato ad una singola applicazione, ma più che altro ad un famiglia, avente in comune certe caratteristiche che portano alla scelta del determinato profilo. Per differenziare il servizio offerto da un dispositivo rispetto a quello offerto da altri, si utilizza un identificativo, l'UUID. Questo identificatore è codificato in 128 bit, quindi si può suppore che due applicazioni con lo stesso UUID offrano il medesimo servizio.
L'implementazione dello stack dei protocolli Bluetooth e il supporto ai profili è designato agli stack Bluetooth [5]. Ogni sistema può essere supportato da diversi stack Bluetooth, Windows ha il supporto per ben 6 diversi stack. Su Linux abbiamo come riferimento BlueZ che fu usato in Android fino al 2012 prima del passaggio a BlueDroid.

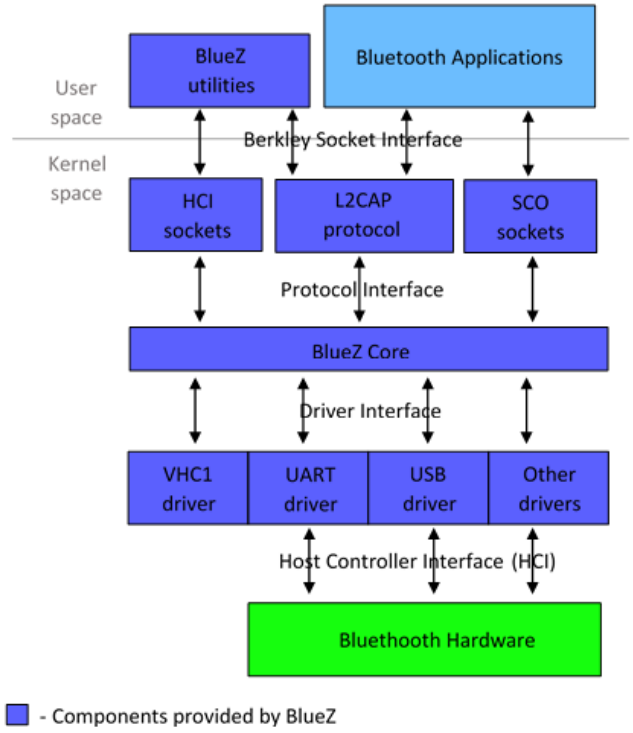


Figura 1.1: Lo stack BlueZ. [1]

1.1.1 Bluetooth 2.0-2.1

Una delle versione di Bluetooth maggiormente usata e diffusa è sicuramente la versione 2, nelle sue due declinazioni 2.0 e 2.1 . Forte del consolidamento della versione 1.2, vengono introdotte importanti novità:

- Introduzione della crittografia per garantire l'anonimato.
- Supporto alle trasmissioni multicast e broadcast.
- Tempi di risposta ridotti.
- Dimezzamento dei consumi grazie all'utilizzo di segnali radio di potenza inferiore.
- Secure Simple Pairing (SPP) che migliora l'accoppiamento dei dispositivi, migliorando uso e livello di sicurezza

[1] fonte:`http://www.stlinux.com/sites/default/files/BlueZ.png`

- Extended Inquiry Response (EIR) che fornisce ulteriori informazioni al fine di filtrare i dispositivi.
- Sniff Subrating per ridurre i consumi in modalità low-power, lasciando la decisione sul timeout per il keepalive ai due interlocutoriciteblu1.
- Enhanced Data Rate (EDR) è una funzionalità, opzionale, che permette un trasferimento dati più veloce, un transfer rate nominale di 3 Mbit/s, che nel pratico diventano 2.1 Mbit/s. EDR può portare anche ad un consumo ridotto.

Le comunicazioni non Low Energy avvengono tutte con lo standard Bluetooth 2. Esiste anche Bluetooth 3,che per il trasferimento usa però le reti 802.11 (WiFi).

1.1.2 Bluetooth 4.0 LE

Nel 2010 Bluetooth ha fatto il suo più grande salto, la specifica 4.0 Low Energy [7]. Come suggerisce il nome, il focus di questa nuova versione è quello di poter funzionare su dispositivi con consumi bassissimi. Per identificare i nuovi prodotti LE[2] viene creato un nuovo logo, Bluetooth Smart. Verranno identificati con questo nuovo logo i prodotti solo compatibili con Bluetooth LE mentre con il logo Bluetooth Smart Ready i prodotti compatibili anche con la classica versione di Bluetooth.

[2]Low Energy

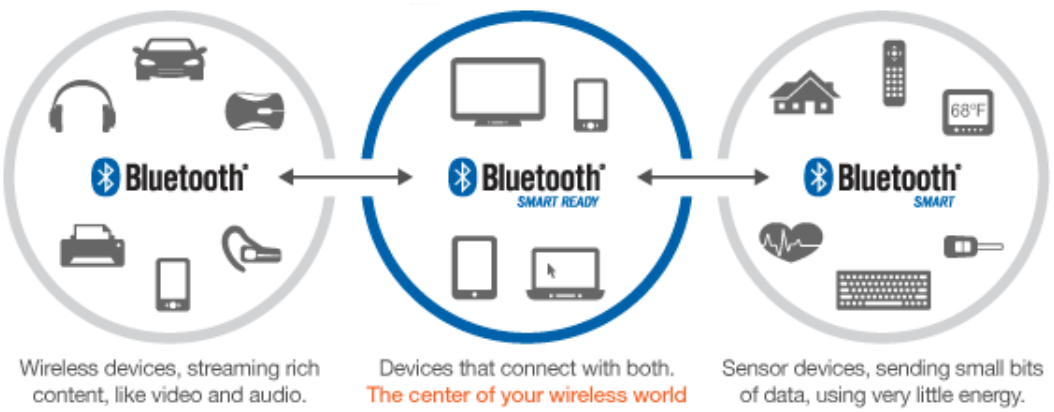


Figura 1.2: I nuovi loghi[3]

Bluetooth smart ha un protocollo interamente nuovo per un collegamento più rapido. È un'alternativa al classico Bluetooth delle versioni dalla 1.0 alla 3.0, che può funzionare su dispositivi alimentati da una batteria a bottone, come gli iBeacon. Questa tecnologia sta avendo una larga diffusione anche nel mondo dei wearable, a partire dalle varie smartband fino agli smartwatch.

Specifiche Tecniche	Bluetooth Classico	Bluetooth Smart
Distanza massima Teorica	100 m	<100 m
Velocità di trasmissione in aria	1-3 Mbit/s	1 Mbit/s
Throughput delle applicazioni	0.7-2.1 Mbit/s	0.27 Mbit/s
Latenza (dallo stato di non-connessione)	100 ms	6 ms
Tempo per l'invio dei dati (consuma la batteria)	100 ms	3 ms
Consumi	1 W (come da reference)	0.01 - 0.5 W (a seconda del caso)

La sua architettura si discosta dall'archiettetura peer to peer, caratterizzata dai tipici ruoli Master-Slave. Vengono quindi quattro ruoli definiti nel Generic Access Profile per BLE [8]:

[3]fonte:`http://mbientlab.com/blog/wp-content/uploads/2014/01/bt40-ecosystem.png`

Broadcaster: è un ruolo ottimizzato per applicazioni di sola trasmissione, e usa l'advertising per l'invio dati.

Observer: è un ruolo ottimizzato per applicazioni di sola ricezione, è complementare al Broadcaster dal quale riceve i dati.

Peripheral: è un ruolo ottimizzato per dispositivi che supportano una sola connessione e sono più semplici di dispositivi Central, funziona in modalità slave.

Central: è un ruolo ottimizzato per dispositivi che supportano connessioni multiple, ed è colui che inizia le connessioni con i dispositivi Peripheral, funziona in modalità master.

Bluetooth Smart usa GATT come base per i suoi profili. Esso è costruito sopra al protcollo ATT (Attribute Protocol). GATT è obbligatorio per BLE ma può essere usato anche sulla versione standard di Bluetooth. GATT definisce due ruoli, client e server, che però non sono necessariamente legati ad uno dei ruoli GAP visti precedentemente.
La struttura di un profilo GATT è molto flessibile. Un profilo formato da una collezione di servizi. Questi servizi possono essere assemblati come più si preferisce, possono essere opzionali e non necessariamente creati da noi. Per esempio per formare il profilo Blood Pressure, vengono usati due servizi: Blood Pressure, Device Information.
Ogni servizio è una collezione di dati e di comportamenti associati. Questi sono rappresentati sotto forma di caratteristiche, contenenti una dichiarazione, delle proprietà e un valore.

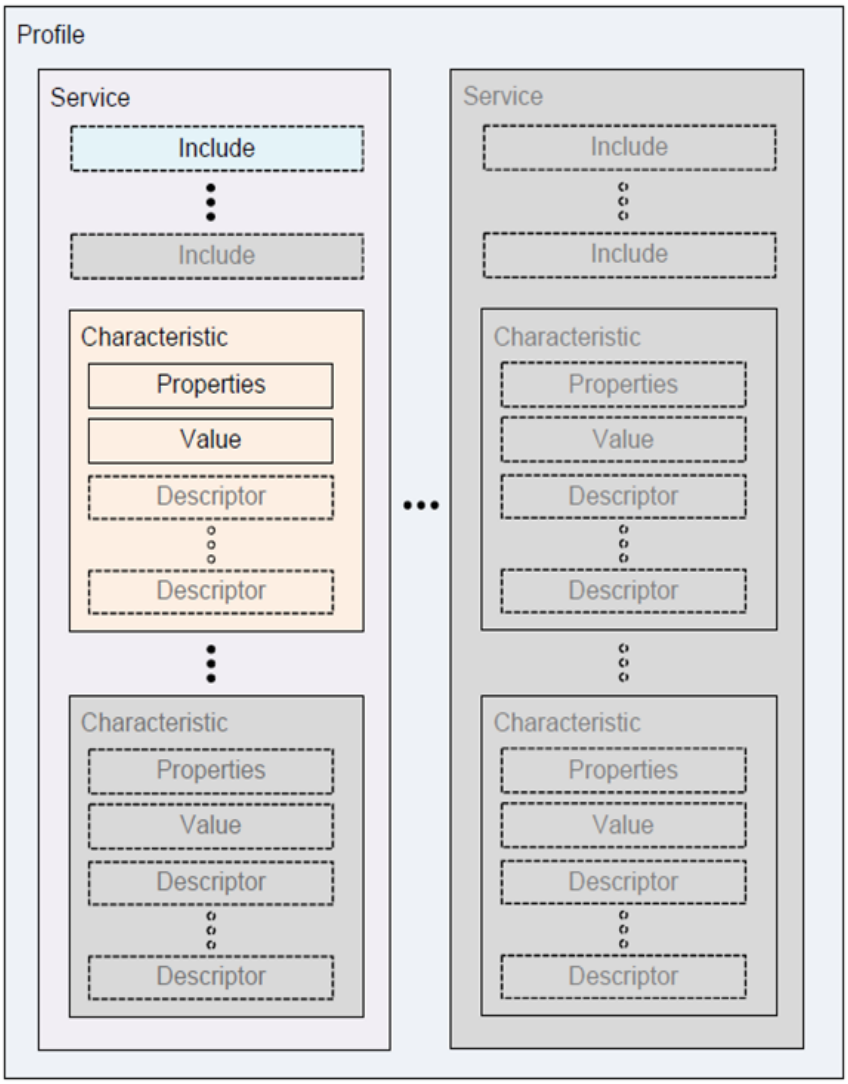


Figura 1.3: Profilo basato su GATT [4]

1.1.3 iBeacon

Tra le dispositivi più interessanti sfruttanti Bluetooth Smart abbiamo iBeacon. iBeacon è un prodotto lanciato da Apple nel 2013 che consiste in un dispositivo Bluetooth LE con la funzionalità di faro (in inglese Beacon). Questo prodotto in accoppiamento ad iOS 7 permetteva di notificare all'utente la presenza del dispositivo nelle vicinanze. Prima produttrice di iBeacon fu Estimote, e da li a poco moltissime società

[4] fonte:`https://developer.bluetooth.org/TechnologyOverview/Documents/GATTprofiles.png`

hanno iniziato la produzione di questi dispositivi.
Questa tecnologia ha molti vantaggi nell'uso indoor, del proximity market o della domotica.
Il dispositivo è formato da una chip BLE, una CPU ARM, e da una batteria a bottone.

Figura 1.4: un Beacon Estimote [5]

Sfrutta sia il ruolo GAP peripheral che quello central per trasmettere le informazioni ai dispositivi nelle vicinanze. Le informazioni mandate nei pacchetti di advertising contengono quattro informazioni rilevanti:

UUID Universal Unique IDentifier, un codice a 128 bit usato per identificare i beacon appartenenti ad una determinata applicazione.

Major un valore a 16 bit usato per distinguere all'interno di un'applicazione a quale sottoinsieme appartengono.

Minor un valore anch'esso a 16 bit per un ulteriore distinzione dei beacon all'interno del sottinsieme associato.

TX Power un dato di 8 bit rappresentante la potenza di trasmissione del beacon, usa la metrica RSSI[6] ed è possibile ottenere la distanza approssimativa grazie ad essa.

La precisione della distanza è influenzata da ostacoli fisici, come muri o persone, e viene classificata in 3 categorie: IMMEDIATE, per distanze inferiori ai 50 cm; NEAR per distanze tra 50 cm e 5 m; FAR per distanze maggiori di 5 m [12].

[5] fonte:`http://cdn-static.zdnet.com/i/r/story/70/00/021354/estimote-beacon-ogrady-475x225.png?hash=MzR0LmNlMT&upscale=1`

[6] Received Signal Strength Indicator espressa in dBm

1.2 Raspberry

Negli ultimi anni c'è stata un esplosione dei dispositivi embedded. Questi dispositivi non più con una specifica funzionalità, ma adattabili a varie situazioni si possono adattare bene al concetto di Internet of Things. Tra i più importanti e famosi abbiamo Arduino e il più recente Raspberry Pi.
Il Raspberry Pi è un computer low-cost della dimensione di una carta di credito, che può essere attaccato ad monitor o TV via HDMI e usare mouse e tastiere USB. È sviluppato nel Regno Unito dalla fondazione Raspberry Pi con l'intenzione di promuovere le basi dell'informatica.
È disponibile in diverse versioni tra i più diffusi c'è il modello B. Questo modello è basato su una CPU ARM11 da 700Mhz con 512 MB di RAM e lo storage è fornito tramite SDCard (solitamente 4GB). Il sistema operativo fornito dalla fondazione è su base Linux. Il linguaggio principale per lo sviluppo sulla piattaforma è Python, ma c'è un ottimo supporto anche per C, C++, Java, etc. [9][10].

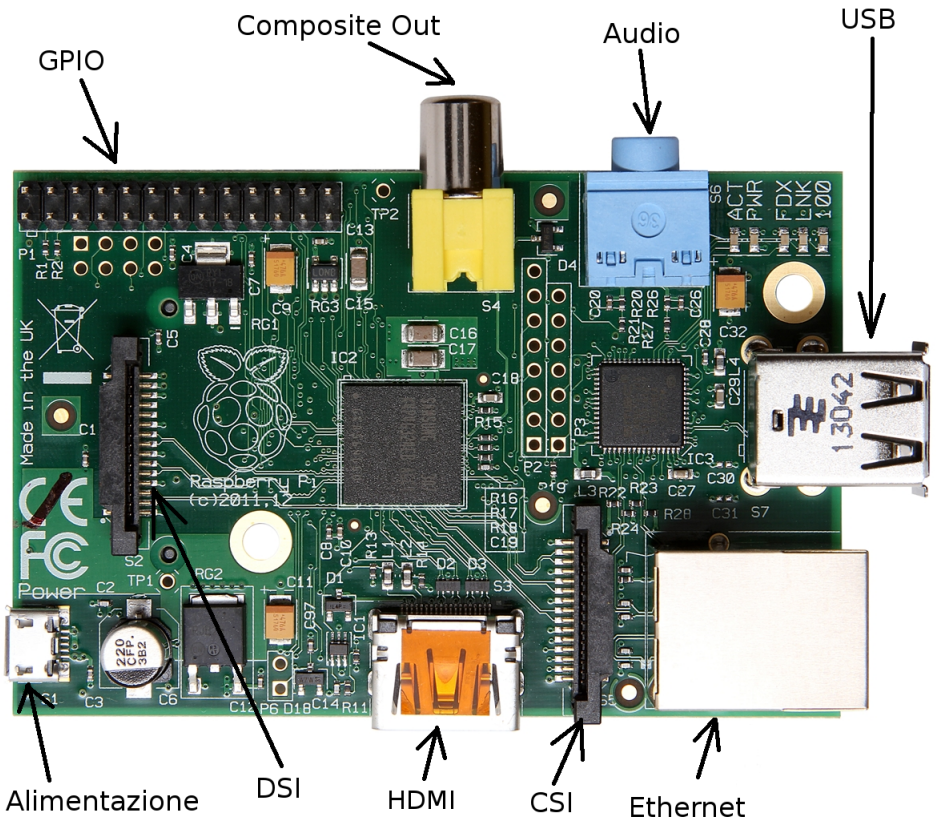


Figura 1.5: Raspberry Pi Modello B[7]

Il Raspberry Pi è stato accolto con entusiasmo dalla community per il suo basso costo, solo 25 $, e le tante potenzialità fornite dalla piattaforma Linux che gira sopra. Sono stati creati dal più semplice media server, a supercomputer con Raspberry Pi in cluster, passando per stazioni meteo e robot [11].
Molte applicazioni sono state realizzate sfruttando una delle più importanti funzionalità fornite dal Rasberry Pi, la presenza dei GPIO. I General Purpose Input/Output sono pin digitali senza una specifica funzionalità, ma che possono essere programmati a seconda delle esigenze. Questa possibilità ha portato alla creazione di svariate schede di espansione per Raspberry.

1.3 Sensori

Nel panorama delle varie espansioni del Raspberry Pi mediante l'uso dei GPIO una delle più importanti possibilità sono quelle fornite dai sensori.
La vasta gamma di sensori che possiamo trovare aprono la strada a una moltitudine di possibilità. Abbiamo sensori di prossimità, di temperatura, di posizione, che permettono una vasta gamma di applicazioni nella domotica, nelle smart cities, nell'intrattenimento e la cultura, etc..

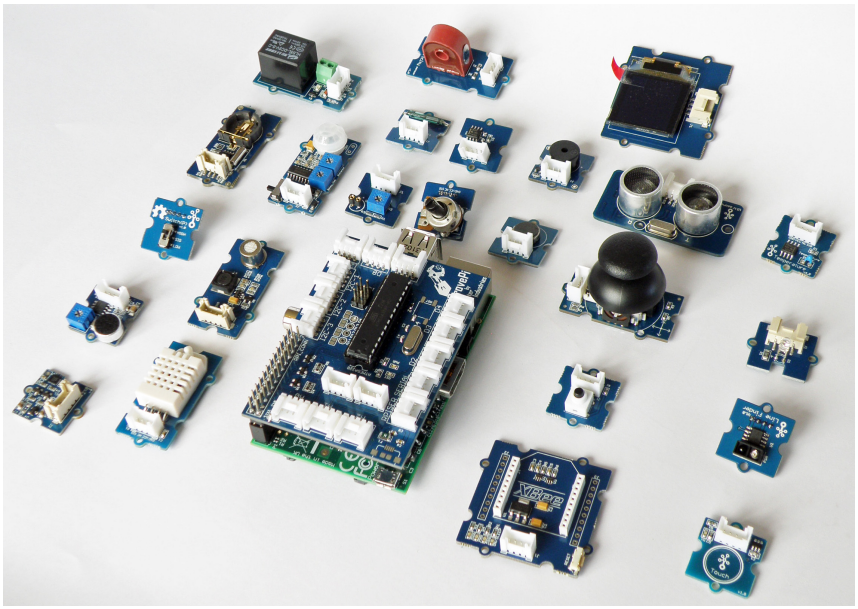


Figura 1.6: Sensori per Rapberry Pi Grove Pi[8]

[7]fonte:`http://www.nexuscyber.com/content/images/thumbs/0000011_raspberry-pi-model-b-rev-20-512mb.jpeg`

Tramite schede convertitrici di segnali analogici, si possono poi adattare altri sensori non specificatemente progettati per Raspberry.

1.3.1 e-Health 2

Tra i vari sensori disponibili, una piattaforma interessante è e-Health Sensor Platform v2 prodotta da Libelium nel progetto Cooking Hacks. Nata come piattaforma sensoristica per Arduino, è utilizzabile tramite un adattatore su Raspberry Pi. Fornisce una vasta gamma di sensori biomedicali quali: sensori di pressione del sangue, per elettrocardiogrammi, per le pulsazioni e il livello di ossigeno, per elettromiografie, per la temperatura, il glucosio nel sangue, la posizione del corpo e la conduttività della pelle.

Figura 1.7: I sensori della piattaforma e-Health v2[9]

Questa sarà la piattaforma che verrà usata poi per il nostro prototipo.

1.4 Android

Il mondo degli smartphone e dei tablet è in continua espansione e, tra i sistemi operativi diffusi, a farla da padrone è Android, l'OS di Google.

[8] fonte:`http://www.dexterindustries.com/GrovePi/wp-content/uploads/2013/10/GrovePi-Grove-for-the-Raspberry-Pi-Grove-Sensors-5.jpg`

[9] fonte:`http://skin.cdn-libelium.com/frontend/default/cooking/images/catalog/documentation/e_health_v2/e_health_sensors_big.png`

Android è un sistema operativo mobile basato sul kernel Linux, progettato per l'utilizzo su dispositivi touchscreen. Nel Q2 2014 Android detiene l'84,7% delle quote di mercato tra i sistemi mobile, seguito da iOS e Windows Phone [13]. Android è distribuito sotto licenza open source, ciò ha portato la community alla creazione di numerose personalizzazioni.
Android è stato sviluppato inizialmente da Android Inc., sovvenzionata da Google, che nel 2005 ha poi rilevato la società. Solo nel 2007 è stata rilasciata la prima release di Android, con il nome in codice Cupcake. Caratteristica particolare di Android è la nomenclatura usata per le versioni del sistema operativo, che usa nomi di dolci, infatti a seguito di Cupcake sono arrivati: Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich, Jelly Bean, KitKat e la più recente Lollipop, con numero di versione 5.0 [14].
Per via della sua natura open source molte società produttrici di smartphone hanno deciso di usare Android per i loro dispositivi portando il sistema operativo alla diffusione su una vastissima gamma di dispositivi. Al contempo la cosa ha creato una grande frammentazione, e la versione più recente del SO spesso tarda ad arrivare o non viene rilasciata. La scelta di questo sistema operativo per quella che sarà poi la parte utente è dovuta all'elevata diffusione.
Una delle caratteristiche principali, che hanno portato alla scelta di Android è il linguaggio di programmazione utilizzato per la creazione delle applicazioni, Java. Ad eseguire il codice Java su Android non è però la normale JVM (Java Virtual Machine), ma una versione appositamente creata da Google, la Dalvik Virtual Machine [15].

Figura 1.8: Architettura di Android con Dalvik[10]

A seguito della compilazione di una applicazione si vengono a creare i file .dex contenti il bytecode. Questi file vengono ottimizzati in fase di installazione con un processo chiamato DEXOPT, ad ogni elaborazione i file ottimizzati vengono trasformati in codice nativo con il processo di compilazione JIT (Just-In-Time). Questo permette l'utilizzo dello stesso APK su tutte le piattaforme supportate, in quanto non dipendente dall'hardware
A Novembre 2013 Google introduce, in Android 4.4 KitKat, la preview del nuovo runtime che andrà a sostituire Dalvik nelle future release di Android: ART (Android Runtime). Le novità introdotte sono varie e si riflettono in un notevole aumento di prestazioni e un completo supporto multi-piattaforma (ARM,x86/MIPS).
Viene usato un misto di JIT, AOT (Ahead Of Time) e interpretato per la compilazione e l'esecuzione. L'introduzione della compilazione AOT, è una delle novità più importanti. Il codice del file .dex invece di essere solo ottimizzato all'installazione, viene compilato con un processo chiamato DES2OAT, ottenendo codice nativo. Questo riduce drasticamente i tempi di caricamento delle applicazioni, che non devono compilarsi completamente ogni avvio. Vengono migliorate inoltre le prestazioni del Garbage Collector, riducendo ad un pausa sola il lavoro del GC [16][17].

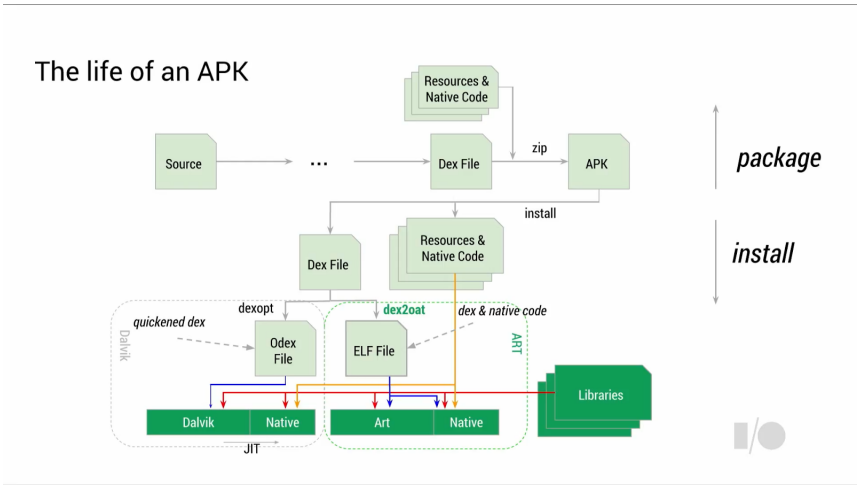


Figura 1.9: Confronto tra Dalvik e ART nella vita di un APK[11]

[10] fonte:https://14b1424d-a-62cb3a1a-s-sites.googlegroups.com/site/io/dalvik-vm-internals/2008-05-29-Presentation-Of-Dalvik-VM-Internals.pdf

[11] fonte:https://www.youtube.com/watch?v=EBlTzQsUoOw

1.5 Le librerie

Per la creazione del sistema abbiamo scelto Java come linguaggio di programmazione, in quanto cross-platform.

1.5.1 Bluetooth sul Raspberry Pi

Le "Java Apis for Bluetooth Wireless Technology" (JABWT) sono una specifica per J2ME (Java 2 Micro Edition) definita nel documento JSR-82 [19]. Lo stile di programmazione adottato è quello tipico delle socket[18].
Si apre un server su un certo profilo, esempio "btspp", e con un determinato UUID corrispondente al nostro servizio fornito, si attende un client, e ad avvenuta connessione si ottengono gli stream con il quale poter comunicare con il client come una normale socket. C'è il supporto alle connessioni multiple, con il limite di 7 dispositivi Bluetooth. Le specifiche della connessione, quali autenticazione, criptazione, possono essere specificate nell'url di avvio della socket.
Quattro sono i componenti di maggior utilizzo:

LocalDevice Questa classe fornisce accesso e controllo del dispositivo Bluetooth locale. È progettata per soddisfare i requisiti definiti dal Generic Attribute Profile nelle specifiche Bluetooth.

RemoteDevice Questa classe rappresenta un dispositivo Bluetooth remoto. Fornisce informazioni base su questo dispositivo, come il suo indirizzo Bluetooth e il nome del dispositivo.

StreamConnectionNotifier L'interfaccia per gestire la socket. Permette l'attesa di un client e la chiusura della socket.

StreamConnection L'interfaccia per la gestione della connessione con un `RemoteDevice`. Consente l'accesso agli stream di lettura e scrittura e la chiusura della connessione.

Su piattaforma J2SE (Java 2 Standard Edition) esistono due implementazioni di JSR-82, AvetanaBluetooth e BlueCove. La mia scelta è ricaduta su BlueCove. BluCove, appoggiandosi alle librerie native JNI, fornisce il supporto a tutti i sistemi operativi più diffusi. Per via della presenza di BlueZ come stack Bluetooth su Linux è però necessaria l'inclusione di una libreria aggiuntiva per l'interfacciamento con esso. Un ulteriore punto da notare è la necessità dei permessi di root per modificare la modalità di discover del dispositivo [20].

1.5.2 Android

Sulla piattaforma Android l'avvio delle applicazioni avviene ad opera del demone Zygote, che è quindi padre di ogni applicazione, e la sua terminazione porterà alla chiusura di tutte le applicazioni e a quello che viene detto in gergo "soft reset" [21].
Un'applicazione Android presenta l'interfaccia grafica mediante le Activity. Ogni singola schermata a sè stante di un'applicazione è un'Activity. Di tutte le Activity dichiarate all'interno del MANIFEST di un'applicazione (il documento xml che presenta le informazioni base dell'applicazione al sistema) una deve essere marcata come iniziale, e sarà il punto di partenza dell'app.
Ogni Activity è poi in grado di lanciarne altre mediante un intent, mano a mano che vengono aperte Activity queste si dispongono a formare uno stack, in cui ogni istanza di Activity è considerata autonomamente [22].
È importante capire il ciclo di vita di un'Activity per poter programmare su Android. Sette sono i metodi che vengono forniti per interagire con il ciclo di vita di un'Activity:

onCreate() Chiamato alla prima creazione dell'Activity. Qui è consigliato di creare il setup statico dell'applicazione (creazione views, bind dati-liste, etc). È inoltre possibile recuperare qui dati di un'istanza dell'activity precedentemente salvati.

onRestart() Chiamato dopo lo stop di un'applicazione, quando essa viene ricominciata a seguito di una navigazione verso di essa.

onStart() Chiamato subito dopo `onCreate()` o `onRestart()`. È seguito da `onResume()` se l'Activity va in primo piano e da `onStop()` se essa va in background.

onResume() Chiamato appena prima dell'inizio dell'interazione dell'Activity con l'utente. Qui l'Activity è in cima allo stack.

onPause() Chiamato quando un'altra Activity passa in foreground. È solito utilizzare questo metodo per fare il commit di dati ancora non salvati e fermare attività intensive per la CPU come le animazioni. Se l'applicazione torna in primo piano è seguito da `onResume()` altrimenti prosegue verso `onStop()`.

onStop() Chiamato quando un'Activity non è più visibile all'utente, perché un'Activity è stata avviata sopra o perché è stata terminata.

Se questa Activity proseguirà verso la terminazione verrà successivamente chiamato il metodo `onDestroy()` altrimenti il prossimo a seguire sarà `onRestart()`.

onDestroy() Chiamato appena prima della distruzione dell'Activity. È l'ultima chiamata che riceverà, può essere a causa di una terminazione voluta o di una richiesta da parte del sistema per necessità di spazio. La distinzione tra i due scenari avviene tramite il metodo `isFinishing()`.

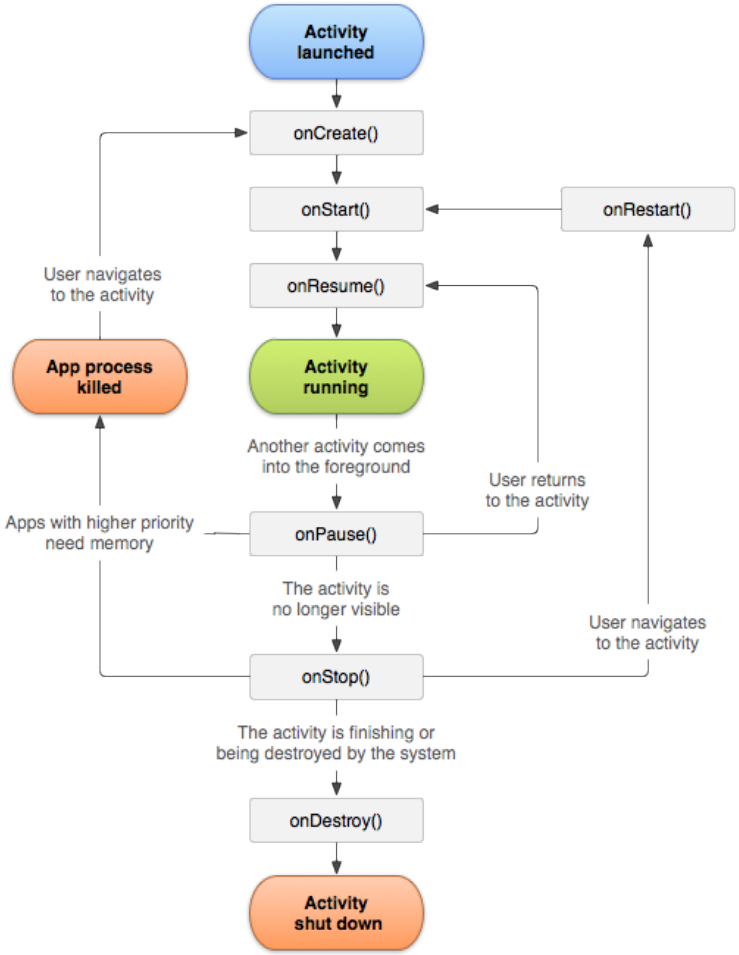


Figura 1.10: Ciclo di vita di un'Activity[12]

[12]fonte:`http://developer.android.com/images/activity_lifecycle.png`

1.5.3 Bluetooth su Android

La piattaforma Android include il supporto allo stack Bluetooth, e fornisce l'accesso alle sue funzionalità tramite le "Android Bluetooth APIs". Permettono la comunicazione punto-a-punto e multi-punto su RFCOMM.
Per l'utilizzare Bluetooth l'applicazione deve richiedere i permessi `BLUETOOTH` e `BLUETOOTH_ADMIN`. Quest'ultimo permette di modificare i settaggi ed avviare azioni di discovery. Il modello su cui si basano le API Bluetooth di Android è quello delle callback.
Possiamo differenziare due set di API, uno per il Bluetooth standard e uno per il Bluetooth Smart.

Bluetooth

Il componente base è il `BluetoothAdapter`, valido anche per BLE, che rappresenta l'adattatore Bluetooth del nostro dispositivo (in modo simile a quello del `LocalDevice` in JSR-82). Per ottenere i dispositivi Bluetooth con cui interagire si hanno due strade: richiedere i dispositivi accoppiati o effettuare una scansione. Gli eventi di una scansione Bluetooth standard deve essere catturati mediante l'uso di un `BroadcastReceiver`, accoppiato con l'azione `ACTION_FOUND`.
Una volta scelto un dispositivo trovato, ottenuto in forma di `BluetoothDevice`. Si possono avviare con esso connessioni RFCOMM sicure ed insicure, a seconda della necessità di criptazione e autenticazione. La connessione va effettuate verso uno specifico UUID, in quanto possono esserci più servizi su uno stesso dispositivo. La comunicazione avviene tramite una `BluetoothSocket`, in modo similare a quello di usato in JSR-82, con i due stream di lettura e scrittura [23].

Bluetooth Low Energy

Le principali funzionalità fornite da questo set di API sono di discovery dei dispositivi BLE e la connessione come client GATT. La scansione avviene mediante le funzionalità fornire dalla classe `BluetoothLeScanner`, presente però anche all'interno del `BluetoothAdapter`. Alla scansione è associata una callback eseguita ogni volta che viene trovato un dispositivo. Scelto un dispositivo Low Energy del quale si vogliono ottenere le informazioni periodicamente, ci si può connettere al suo GATT e gestire tramite una callback gli eventi scatenati dall'interazione con il device. Con l'introduzione delle API 21, rilasciate insieme ad Android 5.0, oltre alla possibilità di ricevere informazioni da un server GATT, è stata introdotta la possibilità di fare advertise [24].

Capitolo 2

Il Progetto

In un mondo dove siamo costantemente circondati da informazioni, le tecnologie e i dispositivi sopra descritti possono trovare un ruolo fondamentale. L'idea dietro questa tesi è la costruzione di sistema comprendente un dispositivo embedded, che per semplicità chiameremo TAG, in grado di inviare dati tramite bluetooth e uno smartphone controllato da un persona che definiremo operatore, in grado di riceverli ed interpretarli.
Il TAG viene depositato nell'ambiente dall'operatore, al fine di poter ricevere le informazioni su ciò che lo circonda. Possono coesistere più TAG all'interno dello stesso ambiente, e uno dei focus è stato quello di permettere la scelta del TAG dal quale prelevare le informazioni.

2.1 Requisiti

Ciò che si vuole studiare è come tramite uno smartphone o tablet si possano individuare dispositivi (TAG) fonti di informazioni e successivamente ricevere da essi i dati. Si vuole in particolare guardare alla modalità di rilevazione e trasmissione di queste informazioni. Per raggiungere questo obbiettivo si vogliono combinare le funzionalità offerte da:

- Bluetooth Smart, il nuovo protocollo Bluetooth orientato al basso consumo, nella veste del iBeacon e della sua funzionalità di sensore di prossimità.
- Bluetooth, il protocollo standard di comunicazione senza fili a corto raggio, per la trasmissione effettiva dei dati.

In definitiva, lo scopo è di aiutare la persona, operatore o cittadino, a ricevere agevolmente le informazioni delle fonti nelle immediate vicinanze.

2.2 Analisi dei requisiti

2.2.1 Requisiti Espliciti

Il sistema che si andrà a realizzare si occuperà di tre cose:

- individuare i TAG nell'ambiente;
- fornire all'utente la scelta di quale TAG ricevere le informazioni;
- presentare queste informazioni all'utente.

L'individuazione del TAG è approssimativa, l'utente non è conscio della posizione effettiva del TAG ma solo della sua presenza nella zona d'interesse. Ai fini del progetto la sola vicinanza o meno al dispositivo è sufficiente, e l'uso della funzionalità beacon permette l'implementazione questa funzionalità con semplicità.

2.2.2 Requisiti Impliciti

Il sistema in questione ovviamente non può che vedere due entità, TAG e operatore, necessariamente separate. Tra i requisiti impliciti ovviamente si devono considerare la presenza delle tecnologie Bluetooth e Bluetooth Smart nei due attori del sistema, nello specifico il supporto all'operatore dovrà implementare entrambe le tecnologie per poter percepire e comunicare con il TAG. Quest'ultimo necessiterà anche lui delle tecnologia Bluetooth standard e Smart.

2.3 Terminologia

Nella descrizione dei requisiti e nella loro analisi ricorrono alcuni termini e loro sinonimi, e nel corso della descrizione del progetto altri compariranno. Per chiarificare il significato e evitare ambiguità, a seguito ne spiegherò l'utilizzo e il significato ai fini del progetto.

TAG (o dispositivo) termine per indicare l'oggetto piazzato sul territorio, che provvede alla raccolta e all'invio delle informazioni. Comprende una parte embedded e un beacon.

Operatore (o utente) l'attore del sistema che è interessato alla ricezione delle informazioni e posa nell'ambiente il TAG, e in estensione il device usato per ottenere tali informazioni.

Beacon (o iBeacon) supporto al TAG per l'individuazione da parte dell'operatore della posizione approssimativa.

Territorio (o ambiente) è lo spazio fisico dove possono essere posati i TAG e che quindi contiene informazioni potenzialmente utili all'operatore.

Zona d'interesse è una porzione del territorio che racchiude uno o più TAG. All'interno di questa porzione, l'operatore ha scelta su quali informazioni vuole ricevere.

Raggio di rilevazione (o di azione) è la porzione del territorio, centrata nell'utente entro il quale lui può rilevare i TAG.

2.4 Scenari

Esistono una vasta gamma di scenari dove si può applicare questa idea, dove può essere utile percepire dati rapidamente.

- Smart City, una realtà in ampio sviluppo, una città intelligente che aiuta chi ci vive. Ci possiamo aspettare un operatore, ad esempio un vigile, che può posare il TAG nel territorio per monitorare il traffico, oppure misurare la qualità dell'aria, dell'acqua e in seguito ottenere queste informazioni più facilmente.

- Un cittadino può avvelersi delle informazioni di questi TAG, velocizzando le routine quotidiane. Un semplice esempio può essere la cosiddetta Smart Mobility, dove i veicoli, dotati di TAG, cooperano tra loro per fornirsi informazioni sul traffico, o dove all'entrata di un parcheggio il parcheggio ti informerà della disponibilità e del prezzo in modo agevole. Possiamo anche immaginare un utilizzo ludico di queste informazioni, con videogame che usino le informazioni dell'ambiente per caratterizzare l'esperienza, o di cultura per ottenere informazioni storiche sui luoghi che si visitano.

- In casi di disastri come terremoti, alluvioni o persino zone di guerra, la possibilità di depositare uno di questi dispositivi su un ferito permette ai medici, e in generale agli operatori sul campo, di ottenere informazioni sullo stato di una persona, solo avvicinandosi, senza doverle misurare.

- Come estensione del caso di soccorso possiamo vederne un utilizzo all'interno di una struttura ospedaliera, dove ogni paziente è

accompagnato dal dispositivo che permette al medico di ottenere velocemente il quadro della situazione.

Quello che è in comune tra questi scenari è un operatore che posiziona il TAG e tutti gli operatori possono percepirne i dati. Questo rientra quello che viene chiamato pervasive computing, dove l'elaborazione dei dati è ovunque, in ogni oggetto che ci circonda e usiamo tutti i giorni, come il frigorifero o un paio di occhiali, rendendo tutto più vivo.

2.4.1 Scenario di riferimento

Lo scenario che andremo a considerare per questo progetto tra quelli descritti è quello ospedaliero o di soccorso, molto simili tra loro. Le differenze che li caratterizzano non influenzeranno la parte analizzata in questa tesi.
Possiamo immaginare di disporre di questo TAG, aventi le funzionalità di beacon e di invio informazioni. Il beacon sarà il "faro" che segnalerà la posizione approssimativa all'utente che si avvicina. Quando l'utente è nel range potrà sfruttare questo beacon, da un elenco dei disponibili nelle sue vicinanze, e connettersi al TAG. A connessione avvenuta il dispositivo inizia la trasmissione delle informazioni verso l'utente.
Dobbiamo quindi fare delle ipotesi perché questo scenario si realizzi:

1. Ogni TAG deve essere in grado di lavorare su Bluetooth Smart per fornire la funzionalità di faro.

2. Il TAG deve essere in grado di inviare i dati usando uno degli standard Bluetooth.

3. Se un beacon è nel raggio di rilevazione, allora il TAG è nel raggio di azione, e quindi mi posso connettere ad esso.

4. L'operatore deve essere in grado di percepire e comunicare con il TAG, quindi il supporto di cui sarà fornito dovrà essere in grado sfruttare sia la tecnologia Bluetooth Standard che quella Smart.

Capitolo 3

Il sistema realizzato

Fino ad ora abbiamo solo parlato del sistema in modo generico. Ora però analizzeremo il sistema che è stato effettivamente realizzato per questa tesi.
Il sistema prototipo creato è formato come già detto da due componenti fondamentali: il TAG, e l'operatore. Il TAG è stato realizzato mediante l'uso di un Raspberry Pi. I dati che esso preleva sono rilevati tramite la sensor platform e-Health v2, dotata di sensori biometrici. Lo scenario che quindi andremo ad esplorare sarà quello ospedaliero / di soccorso, dove l'operatore vorrà raccogliere i dati di un paziente in particolare, in una stanza con più pazienti.

3.1 Dominio

Qui a seguito avverrà una descrizione del dominio dei dati e dei sensori, con interfacce e implementazioni.
Il linguaggio utilizzato è Java, quindi ci si atterrà alle convenzioni tipiche nella realizzazione del progetto, senza però legare la modellazione del dominio a caratteristiche peculiari del linguaggio.

3.1.1 Dati

Il Modello dei dati è basato principalmente sull'interfaccia [25]:
it.unibo.eHealth.tag.domain.interfaces.IData

`IData` rappresenterà il generico dato fornito da un sensore, e potrò quindi essere usato per implementazioni in scenari diversi da quello del caso di studio.

it.unibo.eHealth.tag.domain.interfaces.IDataSet

Il `IDataSet` sarà invece una collezione di dati, utile per raggruppare i dati al fine di inviarli in massa.

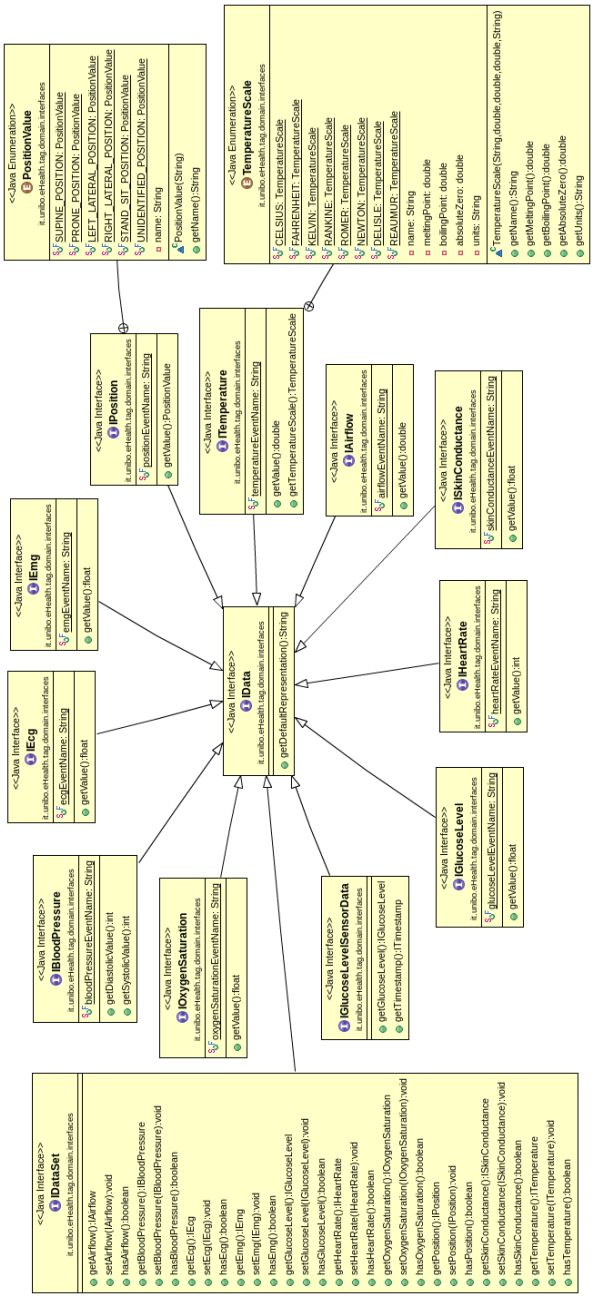


Figura 3.1: Diagramma delle Classi del dominio rappresentante le interfacce dei dati.

3.1.2 Sensori

Per la gestione dei sensori si è usato il pattern Bridge. Avremo quindi due interfacce principali, che sono complementari [25]:

it.unibo.eHealth.domain.interfaces.ISensor

ISensor rappresenterà il generico sensore, dal cui metodo `getInput()` si potrà ottenere il dato, sfruttando il pattern Implementor.

it.unibo.eHealth.domain.interfaces.ISensorImpl

ISensorImpl rappresenterà l'implementazione del sensore, `getInput()` conterrà l'implementazione del sensore.

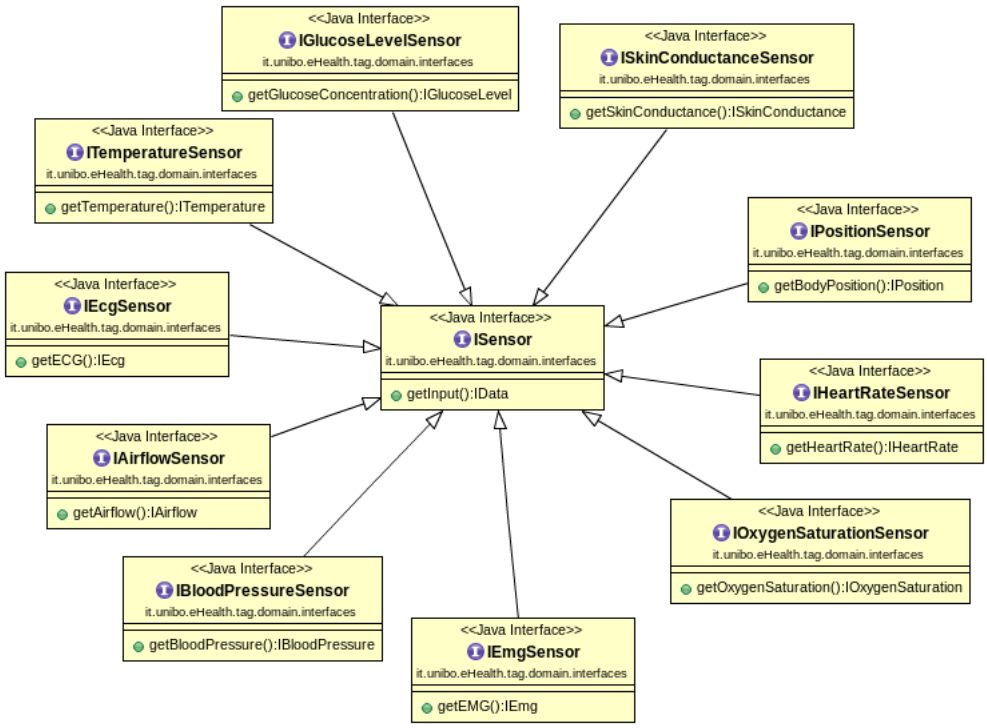


Figura 3.2: Diagramma delle Classi del dominio rappresentante le interfacce dei sensori.

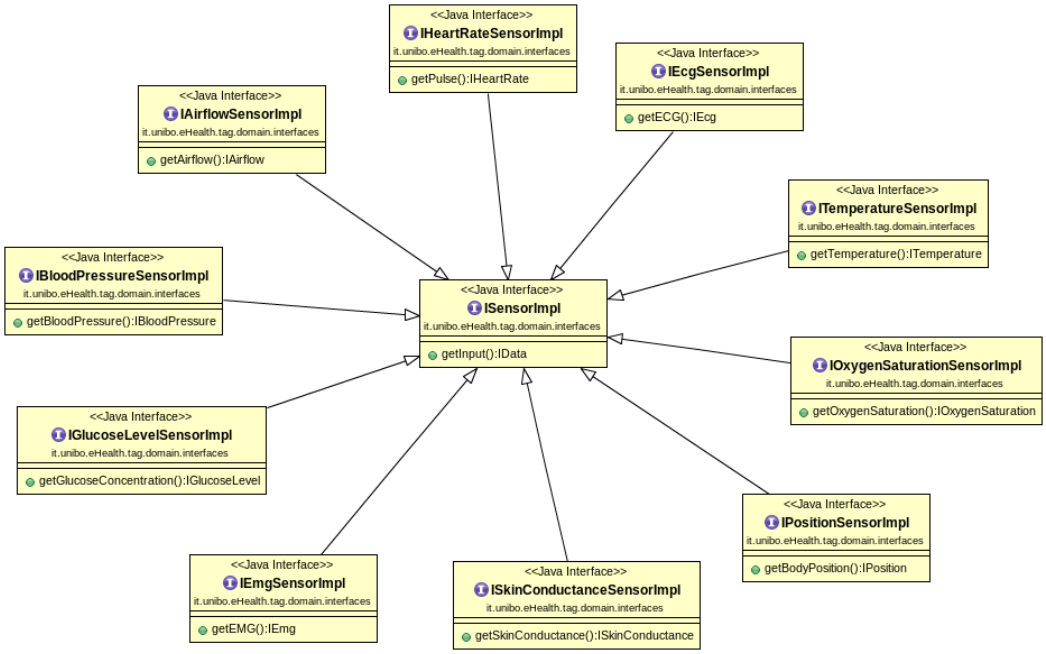


Figura 3.3: Diagramma delle Classi del dominio rappresentante le interfacce dei sensori.

3.1.3 Piattaforma

Scendendo più nello specifico del nostro caso di studio, ora analizziamo brevemente la nostra piattaforma. Per la creazione dei sensori useremo il sistema delle factory, e per esporre i sensori invece il pattern Facade [25].

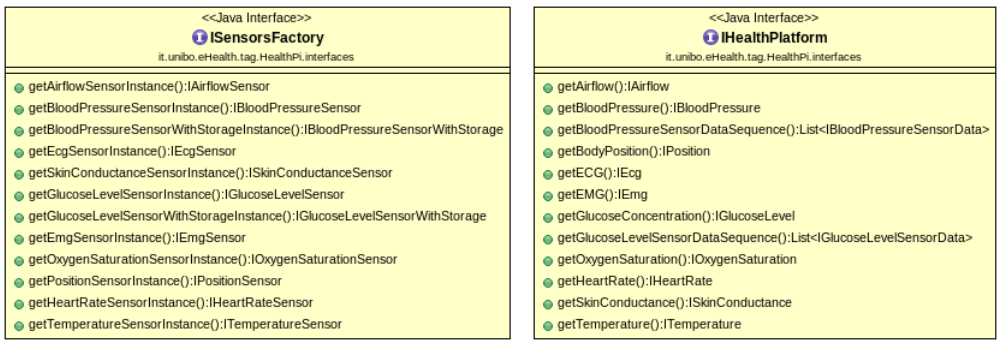


Figura 3.4: Diagramma delle Classi del dominio rappresentante le interfacce dei sensori.

3.1.4 Messaggi

I messaggi che vengono scambiati tra i due componenti dovranno essere serializzabili. Estenderanno l' interfaccia `IMessage`.

it.unibo.eHealth.bluetooth.interfaces.IMessage

`IMessage` rappresenta un generico messaggio che viene scambiato tra due componenti del sistema.

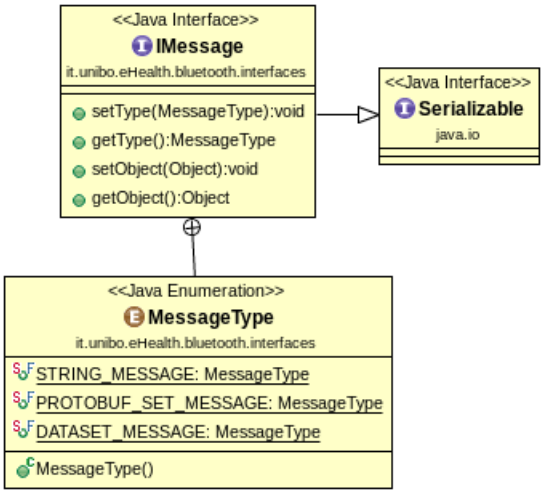


Figura 3.5: Diagramma delle Classi del dominio rappresentante le interfacce dei messaggi.

3.2 Prototipo Raspberry-Android

Ora tratteremo di come è stata affrontata la creazione del prototipo. Abbiamo già detto in precedenza che due saranno le parti fondamentali: il TAG e l'operatore. Il primo è realizzato con un Raspberry Pi, il secondo tramite un applicazione Android.
Per l'implementazione del sistema sono state pensate diverse soluzioni. Il primo approccio è stato un tentativo di rendere il TAG completamente Low Energy, utilizzando quindi la tecnologia Bluetooth Smart per la localizzazione e l'invio delle informazioni sensoriali. Dopo un attenta analisi delle possibilità fornite da Linux all'interno di BlueZ per quando concerne il Bluetooth Low Energy, e la sua implementazione in Java, si è convenuto che il tempo a disposizione non era sufficiente per creare un prototipo mediante l'uso di questa tecnologia, in quanto richiedeva

un modifica dello stack BlueZ di cui non sono chiaramente documentati i meccanismi interni necessari all'introduzione di un profilo GATT personalizzato. Si è quindi deciso di cambiare direzione e di affrontare il problema in modo diverso.
Considerando quindi l'eliminazione di Bluetooth Low Energy dal Raspberry, si viene a creare un problema per la localizzazione del TAG. Per sopperire a questo problema si affianca quindi un iBeacon. Questo può essere usato dall'applicazione operatore per segnalare la presenza del TAG nel raggio di azione.
Per la comunicazione dei dati sensoriali si può invece usare il Bluetooth Standard dal Raspberry Pi.
Si presenta a noi quindi un ulteriore problema. Essendo l'iBeacon e il Raspberry due dispositivi che lavorano su due tecnologie differenti, pur avendo esse la medesima base, come si può associare creare di essi un unico dispositivo TAG? Essendo l'UUID un identificativo universale possiamo presupporre che due TAG usino due diversi UUID, quindi per la definizione del servizi offerto dall'iBeacon e dal Raspberry possiamo usare il medesimo UUID e questo identificherà quella specifica coppia iBeacon-Raspberry.

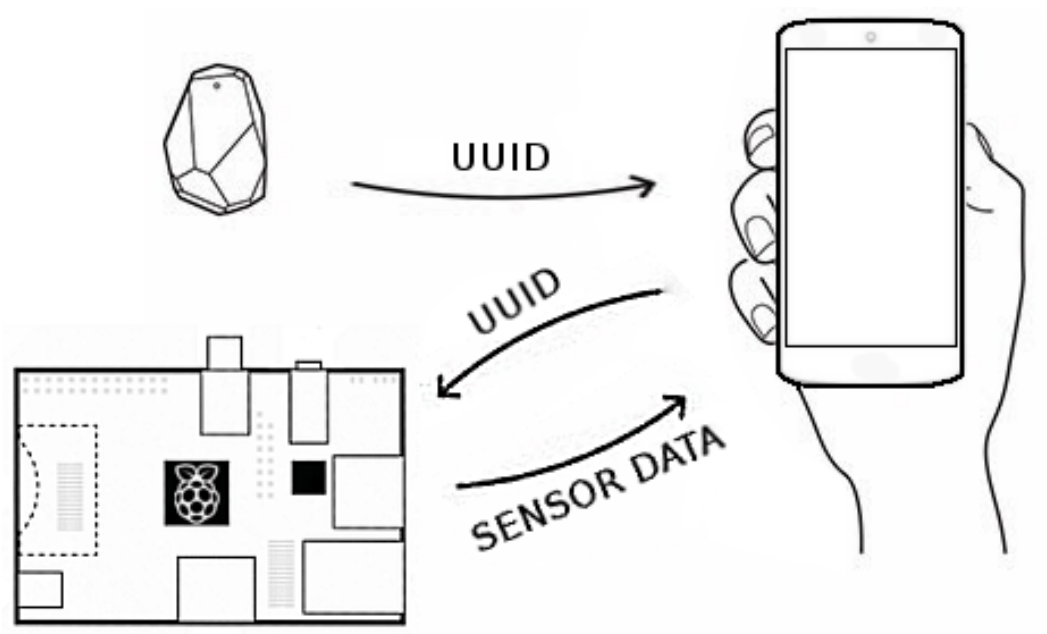


Figura 3.6: Il funzionamento.

3.2.1 TAG

A fronte delle difficoltà trovate nell'implementazione di un profilo GATT personalizzato. Si è quindi preferito l'utilizzo della tecnologia Bluetooth Standard. Per utilizzare questa tecnologia in Java è stata quindi utilizzata l'implementazione BluCove di JSR-82.

BluetoothServer

L'applicazione Java avente la funzione di server Bluetooth, è un semplice socket server. Rimane in attesa di una richiesta di connessione da parte del client. Alla connessione si avviano gli handler, accomunati dall'interfaccia `IHandler`. Si occuperanno di gestire la comunicazione. In modo che il server possa tornare ad attendere un nuovo client.

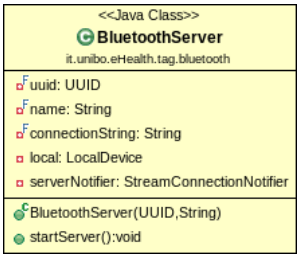


Figura 3.7: Diagramma delle Classi rappresentante il Server Bluetooth.

Listing 3.1: Il server Bluetooth

```
while (true) {
   try {
      System.out.println("\nWaiting for clients to connect...\n");

      StreamConnection connection = serverNotifier.acceptAndOpen();
      System.out.println("Client Connected");

      RemoteDevice dev = RemoteDevice.getRemoteDevice(connection);
      System.out.println("Remote device: address: "
            + dev.getBluetoothAddress() + ";name: "
            + dev.getFriendlyName(true));

      OutputStream out = connection.openOutputStream();

      BluetoothSenderHandler sendHandler = new BluetoothSenderHandler(
      dev.getFriendlyName(true), connection, out);
      sendHandler.start();
   } catch (Exception ex) {
      serverNotifier.close();
      throw ex;
   }
}
```

BluetoohSenderHandler

Per il prototipo in questione è stato scelto di utilizzare un solo handler, che si occuperà dell'invio periodico dei dati.

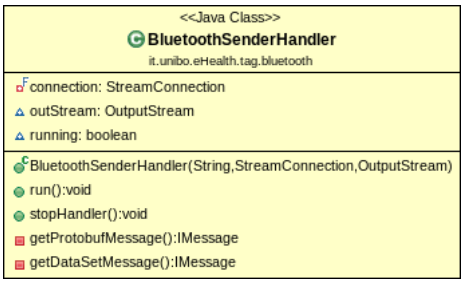


Figura 3.8: Diagramma delle Classi rappresentante il gestore dell'invio delle informazioni.

Listing 3.2: L'Handler di invio delle informazioni.

```
while (running) {
   IMessage msg;
   if (SysConf.USE_PROTOBUFFER) {
      msg = getProtobufMessage();
   } else {
      msg = getDataSetMessage();
   }

   outStream.write(BluetoothMessage.serialize(msg));

   try {
      Thread.sleep(SysConf.SAMPLIG_TIME);
   } catch (InterruptedException e) {
      e.printStackTrace();
   }
}
```

Per raccogliere i dati dai sensori, è stata necessaria la creazione di un wrapper JNI (Java Native Interface) come ponte per la comunicazione con la libreria C++ fornita dal produttore della scheda per il prelevamento dei dati. L'implementazione poi della libreria dei sensori in Java non fa altro che richiamare questo wrapper.

Listing 3.3: L'istanziazione del wrapper

```
public static String LIBRARY_NAME = "eHealth2-simulated";
public static String LIBRARY_PATH = "./";

private EHealthJNI() {
   System.out.println("[Debug] Loading Library :" + LIBRARY_PATH
         + LIBRARY_NAME);
   System.load(LIBRARY_PATH + LIBRARY_NAME);
}
```

A seconda della configurazione del server, specificabile con parametri all'avvio dell'applicazione è possibile scegliere quali sensori utilizzare, e se usare una simulazione invece dei dati reali. Quest'ultima scelta può essere fatta cambiando la libreria che verrà caricata dal wrapper.
Si può notare che nell'handler si fa riferimento a due possibilità per l'invio dei messaggi, a seconda che si voglia. La prima possibilità prevede l'uso di Protocol Buffer [26], un formato semplice ed estensibile per l'invio di dati ideato da Google, mentre la seconda prevede la serializzazione dei un oggetto implementante l'interfaccia `IDataSet`. In entrambe le implementazioni vengono inviati solo le informazioni dei sensori effettivamente rilevati, secondo la configurazione iniziale.

3.2.2 Operatore

Lato Operatore, come già detto, avremo una applicazione Android. Lo smartphone, per sfruttare Bluetooth Low Energy dovrà essere dotato di Android 4.3+, e avere un modulo Bluetooth, BLE compatibile.

MainScanActivity

La `MainScanActivity` è l'Activity principale dell'applicazione, quella eseguita all'avvio, perciò i controlli sulla presenza delle funzionalità Bluetooth e Bluetooth Low Energy vengono eseguiti alla crezione di questa activity.

Listing 3.4: Controllo del supporto Bluetooth

```
final BluetoothManager bluetoothManager =
   (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
mBluetoothAdapter = bluetoothManager.getAdapter();
if (mBluetoothAdapter == null) {
   Toast.makeText(this, R.string.error_bluetooth_not_supported,
       Toast.LENGTH_SHORT).show();
   finish();
   return;
}
// controllo se e' supportato BLE
if
    (!getPackageManager().hasSystemFeature(PackageManager.FEATURE_BLUETOOTH_LE))
    {
   Toast.makeText(this, R.string.ble_not_supported,
       Toast.LENGTH_SHORT).show();
   finish();
}
```

Dopo aver controllato lo stato del Bluetooth e richiesta se necessaria l'attivazione, viene avviata la scansione. L'avvio della scansione avviene invocando sul `BluetoothAdapter` la funzione `startLeScan()`. Come parametro avremo una callback. Di questa callback viene eseguito ogni vol-

ta che viene trovato un dispostivo il metodo `onLeScan()`. Di ogni dispositivo siamo interessati a tenere da parte una coppia dispositivo-UUID, per avere un associazione al momento della scelta dalla lista. L'UUID viene preso eseguendo una conversione dal risultato in byte della scansione, in quanto all'interno del BluetoothDevice le informazioni sull'UUID risultano mancanti. Viene infine ovviamente dato in pasto all'adapter per essere visualizzato.

Listing 3.5: Callback per la scansione Low Energy

```
mBluetoothAdapter.startLeScan(mLeScanCallback);

// Device scan callback.
private BluetoothAdapter.LeScanCallback mLeScanCallback =
   new BluetoothAdapter.LeScanCallback() {

      @Override
      public void onLeScan(final BluetoothDevice device, int rssi, byte[]
          scanRecord) {
      mLeDeviceUUID.put(device, Utility.uuidFromScanRecord(scanRecord));

      runOnUiThread(new Runnable() {
         @Override
         public void run() {
            mLeDeviceListAdapter.addDevice(device);
            mLeDeviceListAdapter.notifyDataSetChanged();
         }
      });
   }
};
```

Il fulcro di tutta l'applicazione è l'associazione iBeacon-Raspberry. Quindi eseguita la scansione, l'utente sceglie il dispositivo al quale collegarsi. Questa selezione è gestita nel metodo `onListItemClick()`. Recuperato l'UUID tramite la mappa precedentemente riempita, si deve ora eseguire una scansione per i dispositivi Bluetooth non LE.
Per fare ciò, si esegue una `startDiscovery()` sul `BluetoothAdapter`. Il sistema ad ogni rilevazione di un dispositivo lancia un broadcast Intent, la cui azione è `ACTION_FOUND`. Per intercettare questa azione bisogna assegnare un `BroadcastReceiver` alla ricezione dello specifico Intent. Ad ogni ricezione salveremo informazioni sul dispositivo scoperto. Al termine della scansione un'altro Intent sarà lanciato, `ACTION_DISCOVERY_FINISHED`.
La sola scansione non rende però disponibili informazioni sui servizi forniti dal dispositivo, non possiamo quindi ancora associarli tramite UUID. Per ottenere gli identificativi dei servizi forniti da ogni `BluetoothDevice` si usa il metodo `fetchUuidsWithSdp()` , che tramite il Service Discovery Profile recupererà queste informazioni. Anche per il recupero degli UUID è usato il meccanismo degli Intent, con l'azione `ACTION_UUID`.
Alla ricezione della lista degli UUID si fa un controllo per la presenza di

quello inizialmente recuperato dal dispositivo LE. Quando viene identificato il Raspberry associato si avvia l'Activity che si occupa del dialogo con il TAG per la ricezione delle informazioni.

Listing 3.6: Ricerca del TAG con un determinato beacon

```
final BroadcastReceiver mReceiver = new BroadcastReceiver() {
   public void onReceive(Context context, Intent intent) {
      String action = intent.getAction();
      // Quando la discovery trova un device
      if (BluetoothDevice.ACTION_FOUND.equals(action)) {
         BluetoothDevice device =
             intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
         mDeviceList.add(device);
      }
      if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action)) {
         for (BluetoothDevice d : mDeviceList) {
            d.fetchUuidsWithSdp();
         }
      }
      if (BluetoothDevice.ACTION_UUID.equals(action)) {
         BluetoothDevice d =
             intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
         Parcelable[] uuidExtra =
             intent.getParcelableArrayExtra(BluetoothDevice.EXTRA_UUID);
         if (d != null && uuidExtra != null && mDeviceList.contains(d)) {
            for (Parcelable uuid : uuidExtra) {
               if (UUID.fromString(uuid.toString()).equals(bleuuid)) {
                  try {
                     final Intent tagIntent = new Intent(getContext(),
                         TagDataActivity.class);
                     tagIntent.putExtra("Device", d);
                     tagIntent.putExtra("UUID", bleuuid);
                     dialog.cancel();
                     startActivity(tagIntent);
                  } catch (Exception ex) {
                     ex.printStackTrace();
                  }
               }
            }
            mDeviceList.remove(d);
         }
      }
   }
};
// Registro il BroadcastReceiver
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
filter.addAction(BluetoothDevice.ACTION_UUID);
registerReceiver(mReceiver, filter);

mBluetoothAdapter.startDiscovery();
```

TagDataActivity

La vera e propria comunicazione con il TAG avviene all'interno di questa Activity. All'avvio vengono recuperati il dispositivo associato e l'UUID del servizio, forniti al lancio dell'Activity.

Questi vengono usati per la creazione del `BluetoothClient`, un entità che si occupa della creazione e dell'avvio della socket Bluetooth. L'ultimo parametro è il `BluetoothMessageHandler`, che gestirà i messaggi ricevuti e inviati da parte del client.

Listing 3.7: Istanziazione del BluetoothClient

```
mBtHandler = new BluetoothMessageHandler();
try {
   mBtClient = new BluetoothClient(mTag, mServiceUUID, mBtHandler);
   mBtClient.setName("BtClient for " + mDeviceName);
} catch (IOException e) {
   finish();
}
```

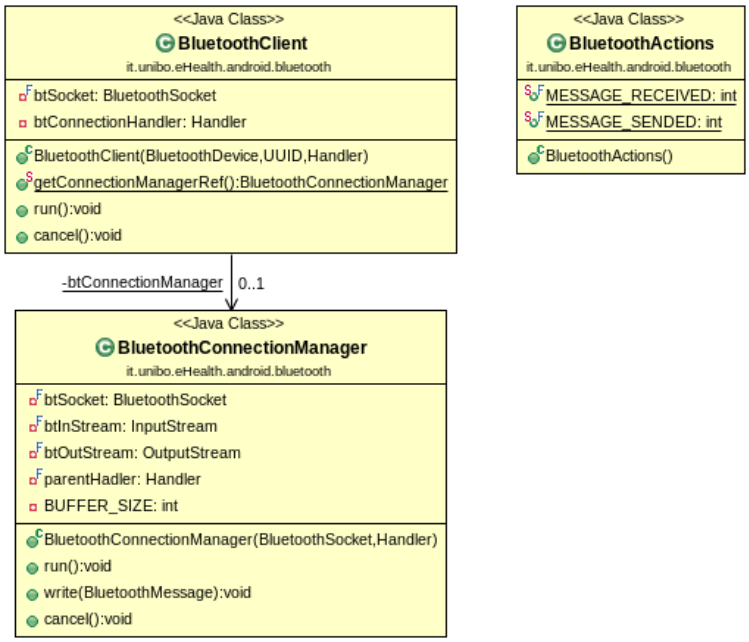


Figura 3.9: Diagramma delle Classi rappresentante i gestori della connessione Bluetooth.

Il `BluetoothClient` è un thread. All'istanziazione crea una socket al servizio con l'UUID fornito, sul device scelto. All'esecuzione del thread si connette alla socket precedentemente creata, e delega il lavoro di invio e ricezione dalla socket al `BluetoothConnectionManager`.

```
\\Costruttore
btSocket = device.createRfcommSocketToServiceRecord(appUUID);

...

\\run()
```

```
try {
   btSocket.connect();
} catch (IOException e) {
   Log.e("Bluetooth Client", e.toString());
   try {
      btSocket.close();
   } catch (IOException e1) {
      Log.e("Bluetooth Client", e1.toString());
   }
   return;
}
btConnectionManager = new BluetoothConnectionManager(btSocket,
   btConnectionHandler);
btConnectionManager.setName("Bluetooth Connection Manager");
btConnectionManager.start();
```

Il `BluetoothConnectionManager` è anch'esso un thread, alla sua creazione richiede gli stream input e output dalla socket. L'esecuzione del thread è una continua lettura sull'input stream. Ogni messaggio ricevuto è deserializzato in dai byte letti, e passato all'handler come `MESSAGE_RECEIVED`. L'invio di un messaggio avviene sempre tramite il `BluetoothConnectionManager`, serializzando il messaggio e scrivendolo nello stream. Viene anche passato all'handler per segnalare il corretto invio come `MESSAGE_SENDED`.

```
\\Costruttore
try {
   btInStream = socket.getInputStream();
   btOutStream = socket.getOutputStream();
} catch (IOException e) {
   Log.e("Bluetooth Connection Manager", e.toString());
}

...

\\run()
byte[] buffer = new byte[BUFFER_SIZE];

while (true) {
   try {
      btInStream.read(buffer);
      BluetoothMessage btMsg = BluetoothMessage.deserialize(buffer);
      parentHadler.obtainMessage(BluetoothActions.MESSAGE_RECEIVED, -1, -1,
          btMsg)
         .sendToTarget();

   } catch (Exception e) {
      Log.e("Bluetooth Connection Manager", e.toString());
      break;
   }
}

...

\\write()
try {
   byte[] buffer = BluetoothMessage.serialize(btMsg);

   btOutStream.write(buffer);
   parentHadler.obtainMessage(BluetoothActions.MESSAGE_SENDED, -1, -1, btMsg)
```

```
      .sendToTarget();
} catch (Exception e) {
   Log.e("Bluetooth Connection Manager", e.getMessage());
}
```

Il `BluetoothMessageHandler` per ricevere i messaggi che gli vengono passati esegue l'override del metodo `handleMessage()`. Qui il messaggio viene copiato e a seconda della `BluetoothAction` si avvia il task asincrono per la sua gestione.
`ManageReceivedMessageTask` si occupa dei messaggi ricevuti, quindi i messaggi contenenti i dati dei sensori,e rende le informazioni disponibili all'utente. `ManageSendedMessageTask` gestice i messaggi inviati, mandando un feedback all'utente per l'avvenuto invio.

```
\\BluetoothMessageHandler

Message copiedMessage = new Message();
copiedMessage.copyFrom(msg);

switch (msg.what) {
   case BluetoothActions.MESSAGE_RECEIVED:
      new ManageReceivedMessageTask().execute(copiedMessage);
      break;

   case BluetoothActions.MESSAGE_SENDED:
      new ManageSendedMessageTask().execute(copiedMessage);
   break;
}

\\ManageReceivedMessageTask

switch (btMsg.getType()) {
   case STRING_MESSAGE:
      break;
   case PROTOBUF_SET_MESSAGE:
      manageProtobufMessage(btMsg);
      break;
   case DATASET_MESSAGE:
      manageDataSetMessage(btMsg);
      break;
   default:
      break;
}

\\ManageSendedMessageTask

switch (btMsg.getType()) {
   case STRING_MESSAGE:
      Toast notification = Toast.makeText(getApplicationContext(), "Sended
          message to server", Toast.LENGTH_SHORT);
      notification.show();
      break;
   default:
      break;
}
```

3.3 Analisi del prototipo

Analizziamo ora il comportamento del prototipo realizzato.

3.3.1 TAG

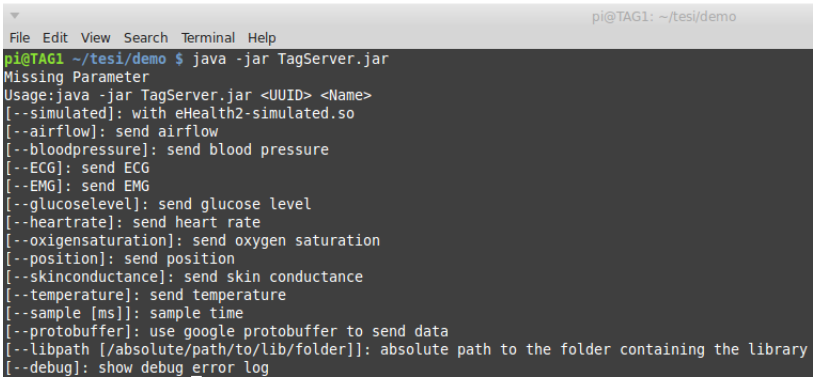
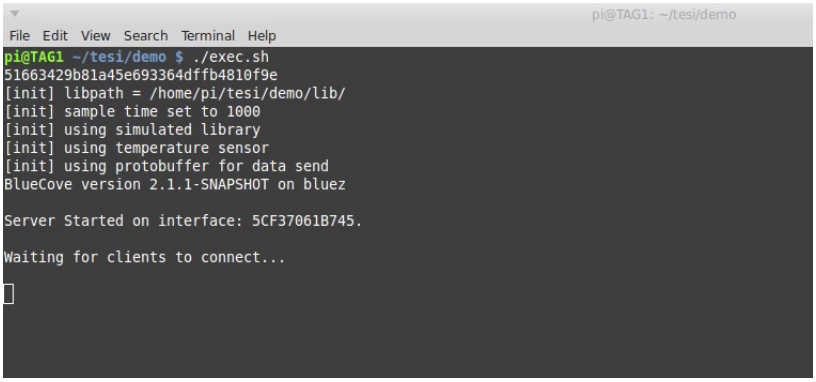
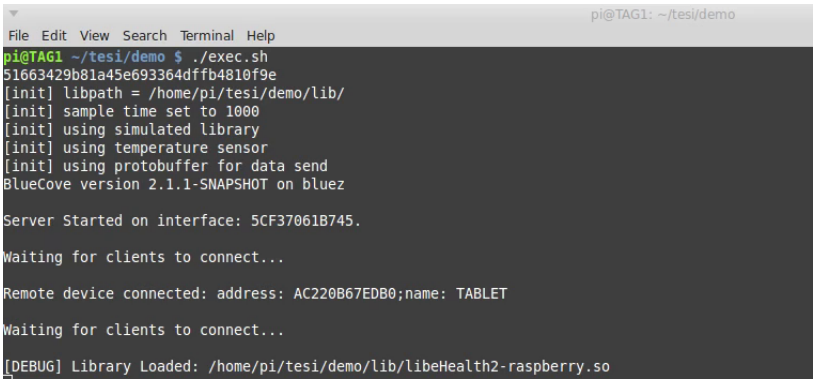


Figura 3.10: I parametri dell'applicazione TAG.

La parte TAG del prototipo è rappresentata da un Raspberry Pi accessoriata con la sensor board eHealth v2.0. Il sistema di base è Linux Debian nella sua forma adattata e ridotta per Raspberry, Raspbian. La versione di Java utilizzata per la compilazione e l'esecuzione dell'applicativo è 1.7.0r65. Si era pensato anche un approccio mediante l'uso di Java 1.8, ma l'idea è stata scartata causa della mancanza del supporto su Android. Gli stack Bluetooth utilizzati per la realizzazione del prototipo sono Bluez 4.99 per quanto riguarda la parte nativa Linux, e Bluecove 2.1.1 per quanto riguarda l'implementazione Java.
L'applicazione è vastamente configurabile: l'ambiente può essere simulato se la sensor board non è presente, e si può scegliere quali sensori attivare, ogni quanto inviare le informazioni, e con quale mezzo di codifica. È inoltre importante specificare il percorso della libreria che accompagnerà l'applicazione nell'interfacciamento ai sensori.



(a) TAG in attesa del client



(b) TAG in fase di comunicazione

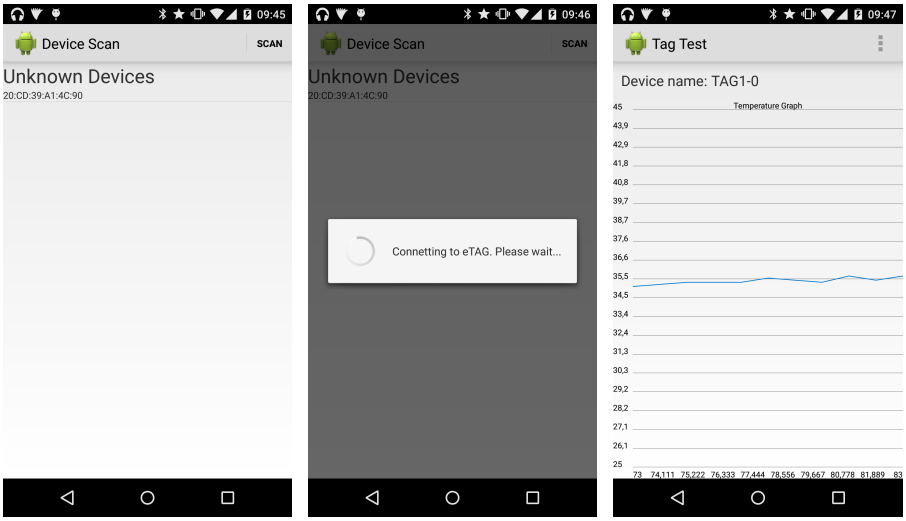
Figura 3.11: TAG durante la demo.

Per la dimostrazione l'esecuzione avviene tramite uno script preconfigurato, con invio della temperatura in formato protobuffer ogni secondo. La libreria viene caricata al primo utilizzo di un sensore. Questo provoca un ritardo nella prima comunicazione dall'avvio del server, dell'ordine di un paio di secondi. Per le successive connessioni questo ritardo non è presente.

Listing 3.8: Script lancio demo

```
#!/bin/bash
sudo java -jar TagServer.jar 51663429B81A45E693364DFFB4810F9E TAG1
    --temperature --libpath /home/pi/tesi/demo/lib --protobuffer --sample 1000
```

3.3.2 Operatore



(a) Operatore in fase selezione del beacon (b) Operatore in attesa di collegamento al TAG (c) Operatore in fase di comunicazione

Figura 3.12: Operatore durante la demo.

La parte Operatore del prototipo è invece rappresentata nelle varie prove da un Nexus 7 avente come versione di Android la 4.4.4 o da un Nexus 5 avente la versione 5.0. Le API usate per la compilazione sono le 19 (Android 4.4.2).
Per la dimostrazione è stato possibile allestire un solo TAG, e non è stato possibile dimostrare la scansione di più Beacon in quanto solo funzionava. L'operatore può eseguire manualmente la scansione dei dispositivi LE. Selezionando una voce dall'elenco viene iniziata la ricerca.
Questo processo richiede circa 10 secondi, al termine dei quali vengono richiesti gli UUID. Tutto ciò avviene in background durante la permanenza del box di caricamento. I dispositivi devono essere accoppiati per poter comunicare, la mancanza di un accoppiamento impedirà l'apertura delle comunicazioni verso il TAG.
La comunicazione dei dati inizia immediatamente, ci sono ritardi apprezzabili solo nel caso della prima connessione del TAG dal suo avvio (vedi sopra). Le informazioni recuperate vengono visualizzate in una `ListView` caricata dinamicamente in base ai dati ricevuti.

Conclusioni

A sistema realizzato possiamo quindi tirare le conclusioni.
Le tecnologie analizzate sono state pienamente sfruttate per la creazione del prototipo. La tecnologia Bluetooth Low Energy, o Smart, a causa della difficoltà incontrata nell'implementazione di un profilo GATT personalizzato sulla piattaforma Linux del Raspberry Pi è stata utilizzata solo per l'uso con il beacon. In futuro, e con più tempo a disposizione sarà sicuramente possibile implementare tutta la parte BLE inizialmente pensata. A causa di questa complicazione la parte di invio dati del TAG, che potenzialmente poteva essere nel profilo personalizzato, è stata realizzata mediante il Bluetooth standard.
Seppur concettualmente un unica entità, i dispositivi di cui è formato un TAG sono due, iBeacon e Raspberry Pi, e si occupano ognuno dell'implementazione di una delle due tecnologie Bluetooth. Questo crea un overhead in fase di scelta di un dispositivo. Dovrà essere cercata infatti l'altra parte del TAG tra i device Bluetooth disponibili nell'area di rilevazione, prima di iniziare il trasferimento. L'implementazione tramite Bluetooth Smart avrebbe di fatto eliminato questo overhead, ma considerate le premesse questa è l'implementazione più funzionale.
Lato operatore le conclusioni da trarre riguardano solo l'accoppiamento, in quanto un implementazione completamente BLE era facilmente realizzabile con le API fornite dal sistema Android. Sono invece stati rilevati comportamenti anomali per quanto riguarda l'accoppiamento. Anche con l'utilizzo di una socket RFCOMM insecure, senza autenticazione e crittografia, la mancanza di un accoppiamento tra i due dispositivi impossibilitava all'apertura del canale di comunicazione. Per velocizzare quindi il tutto è consigliabile pre-accoppiare i dispositivi, o il primo tentativo di collegamento fallirà, creando però l'accoppiamento.
Per sviluppi futuri a partire dal prototipo realizzato, vedo un implementazione BLE di tutto il TAG direttamente su Raspbery Pi, appena raggiunta la maturità delle API Linux e Java per questa tecnologia.
Prevedo in futuro un utilizzo sempre più esteso del concetto che si è cercato di esprimere con questa tesi. Un mondo dove le informazioni

che descrivono ciò che ci circonda potranno essere a nostra disposizione in modo immediato, direttamente a noi su smartphone o dispositivi di realtà aumentata, a bordo dei nostri veicoli o comodamente nelle nostre case.

Ringraziamenti

Si conclude qua la mia laurea triennale. Tante sono le persone che dovrei ringraziare, oserei dire troppe, mi limiterò un po'. Ringrazio la mia famiglia, genitori e nonni, zii e cugini, per il supporto che mi hanno dato in questi tre anni. Ringrazio inoltre Andrea, Gianluca e tutti gli altri compagni di questa avventura che hanno reso indimenticabili questi anni passati insieme. È doveroso ringraziare il professore Viroli per l'opportunità concessami con questa tesi. Un altro ringraziamento va ai ragazzi del PSLab, Edoardo, Pietro e soprattutto Angelo che ha sopportato i miei continui dilemmi e divagamenti. Grazie a tutti i compagni di studio ingegneri e non ingegneri. E un ultimo grazie a tutti gli amici che mi hanno sostenuto e aiutato a superare i momenti difficili lungo il mio percorso.
Grazie!

Bibliografia

[1] Wikipedia. *Bluetooth* - wikipedia, the free encyclpedia. , 2014. http://en.wikipedia.org/wiki/Bluetooth [Online; in data 19-Novembre-2014]

[2] *Updated Bluetooth® 4.1 Extends the Foundation of Bluetooth Technology for the Internet of Things* ,http://www.bluetooth.com/Pages/Press-Releases-Detail.aspx?ItemID=197 [Online; in data 19-Novembre-2014]

[3] Wikipedia. *List of Bluetooth protocols* - wikipedia, the free encyclpedia. , 2014. http://en.wikipedia.org/wiki/List_of_Bluetooth_protocols [Online; in data 19-Novembre-2014]

[4] Wikipedia. *List of Bluetooth profiles* - wikipedia, the free encyclpedia. , 2014. http://en.wikipedia.org/wiki/List_of_Bluetooth_profiles [Online; in data 19-Novembre-2014]

[5] Wikipedia. *Bluetooth Stack* - wikipedia, the free encyclopedia. , 2014. http://en.wikipedia.org/wiki/Bluetooth_stack [Online: in data 20-Novembre-2014]

[6] *Sniff Subrating*, http://searchmobilecomputing.techtarget.com/definition/sniff-subrating [Online: in data 20-Novembre-2014]

[7] Wikipedia. *Bluetooth Low Energy* - wikipedia, the free encyclopedia. , http://en.wikipedia.org/wiki/Bluetooth_low_energy [Online; in data 20-Novembre-2014]

[8] Bluetooth SIG. *Bluetooth Core Specification v4.0 Vol 1* [30-Giugno-2010]

[9] Raspberry Pi Foundation *What is a Raspberry Pi?* http://www.raspberrypi.org/help/what-is-a-raspberry-pi/ [Online: in data 21-Novembre-2014]

[10] Wikipedia. *Raspberry Pi* - wikipedia, the free encyclopedia. ,http://en.wikipedia.org/wiki/Raspberry_Pi [Online: in data 21-Novembre-2012]

[11] Network World. *20 cool things you can do with a Raspberry Pi*, www.networkworld.com/article/2290609/computers/153240-20-cool-things-you-can-do-with-a-Raspberry-Pi.html[Online: in data 24 Novembre]

[12] Andrea Fortibuoni. *Sviluppo di una infrastruttura location-based per l'auto-organizzazione di smart-devices*, 2013/2014.

[13] IDC *Smartphone OS Market Share, Q2 2014* , http://www.idc.com/prodserv/smartphone-os-market-share.jsp [Online: in data 25-Novembre-2014]

[14] Android Official Blog *Android: Be together. Not the same.*, http://officialandroid.blogspot.it/2014/10/android-be-together-not-same.html[Online: in data 25-Novembre-2014]

[15] Dan Bornstein *Dalvik VM Internals* Google Inc. https://sites.google.com/site/io/dalvik-vm-internals/2008-05-29-Presentation-Of-Dalvik-VM-Internals.pdf?attredirects=1[Online: in data 3-Dicembre-2014]

[16] Google Inc *Google I/O 2014 - Keynote* https://www.youtube.com/watch?v=wtLJPvx7-ys [Online: in data 3-Dicembre-2014]

[17] AnandTech *A Closer Look at Android RunTime (ART) in Android L* http://www.anandtech.com/show/8231/a-closer-look-at-android-runtime-art-in-android-l

[18] Wikipedia. *Java APIs for Bluetooth* - wikipedia, the free encyclopedia. ,http://en.wikipedia.org/wiki/Java_APIs_for_Bluetooth [Online: in data 25-Novembre-2014]

[19] Motorola *Java APIs for Bluetooth Wireless Technology (JSR 82) Specification Version 1.1.1* [29-Luglio-2008]

[20] BluCove *BluCove Linux Module*, http://www.bluecove.org/bluecove-gpl/ [Online: in data 25-Novembre-2014]

[21] Anatomy of Android *Zygote* http://anatomyofandroid.com/2013/10/15/zygote/ [Online: in data 3-Dicembre-2014]

[22] Google Inc *Activities*`http://developer.android.com/guide/components/activities.html` [Online: in data 3-Dicembre-2014]

[23] Google Inc *Bluetooth*`http://developer.android.com/guide/topics/connectivity/bluetooth.html` [Online: in data 25-Novembre-2014]

[24] Google Inc *Bluetooth Low Energy*`http://developer.android.com/guide/topics/connectivity/bluetooth-le.html` [Online: in data 25-Novembre-2014]

[25] Fabio Vasini*HealthPi library process report* [20-Settembre-2014]

[26] Google Inc *Protocol Buffers - Google's data interchange format*, `https://code.google.com/p/protobuf/` [Online: in data 27-Novembre-2014