Alma Mater Studiorum
Campus di Cesena

Scuola di Scienze
Corso di laurea in ingegneria e scienze informatiche

Realizzazione di un'interfaccia universale in tecnologia Bluetooth Low Energy per sensori integrati

Prova finale in
PROGRAMMAZIONE A OGGETTI

Relatore:
Prof. MIRKO VIROLI

Autore:
PIERO BIAGINI

Seconda sessione di laurea
Anno Accademico 2014-2015

Parole Chiave

Elements

Bluetooth Low Energy

Android

Interfaccia universale

Trasmissione di dati

Alla mia famiglia,
che mi è sempre stata vicino

Indice

1 Introduzione al Bluetooth Low Energy **1**

1.1 Bluetooth 1

1.1.1 Implementazione 2

1.1.2 Pila protocollare 3

1.1.3 Connessione 4

1.2 Bluetooth Low Energy (*BLE*) 5

1.2.1 Storia 5

1.2.2 Applicazioni 5

1.2.3 Componenti fondamentali 5

1.2.4 Profili, Servizi e Caratteristiche 7

1.2.5 Advertising, connessione e scambio di dati 9

1.2.6 Confronto con altre tecnologie simili 10

2 Introduzione ad Android **13**

2.1 Storia 14

2.2 Architettura 14

2.2.1 Kernel 14

2.2.2 Applicazioni 14

2.3 Versioni di Android 15

2.4 Programmazione 16

2.5 Android e BLE 16

3 Introduzione al modulo Texas Instrument CC2650 **21**

3.1 Programmazione 23

3.1.1 Struttura di un programma tipico 23

4 Analisi dei requisiti **25**

4.1 Descrizione generale 25

4.1.1 Funzioni del prodotto 26

4.1.2 Requisiti espressi dall'azienda 27

4.1.3 Ambiente di utilizzo 27

4.1.4 Linguaggi e piattaforme di sviluppo 27

4.2 Interfacciamento con l'esterno 28

4.2.1 Interfaccia software 28

4.2.2 Interfaccia utente 28

4.3 Funzionalità del sistema 28

4.3.1 Lato lettura e trasmissione dei dati 28

4.3.2 Lato ricezione ed interpretazione dei dati 30

5 Analisi **35**

5.1 Moduli del progetto 35

5.2 Analisi del macro-modulo di lettura e trasmissione 36

5.2.1 Casi d'uso 36

5.2.2 Architettura del macro-modulo 38

5.3 Analisi del macro-modulo di ricezione ed interpretazione 39

5.3.1 Casi d'uso 39

5.3.2 Progettazione concettuale delle classi e della base di dati . . 42

5.4 Sessione tipica di utilizzo 49

5.5 Diagrammi di sequenza 49

5.5.1 Lettura e scambio di dati 49

5.5.2 Connessione 50

5.6 Codifica dei dati 52

5.6.1 Dati trasmessi tramite Bluetooth 52

5.6.2 Dati esportati dall'applicazione Android 52

6 Progettazione **53**

6.1 Progettazione del macro-modulo di lettura e trasmissione 53

6.1.1 Scelta del profilo ricoperto dal dispositivo 53
6.1.2 Configurazione delle caratteristiche utilizzate per l'invio e la ricezione dei dati 54
6.1.3 Organizzazione a task e comunicazione tra essi 54
6.1.4 Lettura dall'ADC 55
6.1.5 Consumo energetico ridotto 55
6.1.6 Interfacciamento con l'utente 56
6.2 Progettazione del macro-modulo di ricezione ed interpretazione . . . 56
6.2.1 Modulo di gestione del Bluetooth 56
6.2.2 Modulo di gestione dei dati 57
6.2.3 Modulo di connessione e visualizzazione in tempo reale . . . 59
6.2.4 Modulo di consultazione dei dati memorizzati 61
6.2.5 Progettazione dell'interfaccia grafica 63

7 Implementazione **67**
7.1 Implementazione del macro-modulo di lettura e trasmissione 67
7.1.1 Configurazione delle caratteristiche 67
7.1.2 Organizzazione a task 69
7.1.3 Comunicazione tra i task 70
7.1.4 Lettura dall'ADC 71
7.1.5 Consumo energetico ridotto 72
7.1.6 Interfacciamento con l'utente 72
7.2 Implementazione del macro-modulo di ricezione ed interpretazione . 73
7.2.1 Modulo di gestione del Bluetooth 73
7.2.2 Modulo di gestione dei dati 75
7.2.3 Modulo di connessione e visualizzazione in tempo reale . . . 77
7.2.4 Modulo di consultazione dei dati memorizzati 78

8 Testing **81**
8.1 Modulo di lettura e trasmissione 81
8.2 Modulo di ricezione ed interpretazione 82
8.3 Soddisfazione dei requisiti 83

Conclusioni **83**

Elenco delle figure **87**

Introduzione

Lo scopo dell'elaborato di tesi è la progettazione e lo sviluppo di un'applicazione per il modulo Bluetooth Low Energy (BLE) Texas Instrument CC2650 in grado di leggere le informazioni da un sensore analogico, tramite un Analog-Digital-Converter (ADC), e di scambiare i dati con uno smartphone Android in tempo reale. L'interfaccia realizzata deve essere universale, ovvero dovrà essere compatibile con sensori di diverso tipo, facilmente estensibile per l'aggiunta di un numero maggiore di periferiche di lettura e utilizzabile con un ampio numero di dispositivi.

Dato lo sviluppo che l'azienda sta avendo e vista la previsione dell'inserimento sul mercato di nuovi dispositivi, di tipo diverso da quelli già prodotti e orientati all'Internet of Things, *Elements* ha espresso l'esigenza di un sistema software in grado di testare e comprendere il funzionamento del modulo BLE prescelto in modo da poterlo, eventualmente, utilizzare come base per i dispositivi futuri.

Il CC2650 dovrà quindi operare come un device BLE in grado di accettare connessioni da parte di uno Smartphone Android con cui dovrà scambiare i dati letti dall'ADC a intervalli di tempo prescelti dall'applicazione stessa. L'applicazione Android dovrà leggere i dati, eseguire delle operazioni su di essi per renderli fruibili e coerenti con la misura realizzata dal sensore collegato al dispositivo e mostrarli a schermo sia in forma tabulare che in forma grafica. Saranno in seguito implementate parti relative alla memorizzazione permanente e alla consultazione dei dati già ricevuti.

Si evidenziano in questo modo i due ruoli di client e server, ricoperti rispettivamente dal modulo BLE CC2650 e dallo smartphone Android.

Essendo il primo approccio della ditta ad applicazioni di questo tipo, il software è da scrivere a partire da zero.

La difficoltà maggiore incontrata durante lo sviluppo di tali applicazioni è l'uso di tecnologie relativamente nuove (il Bluetooth Low Energy è stato presentato nel 2010 ma risulta, nel 2015, ancora relativamente poco diffuso: per esempio su Android è stato reso disponibile solamente dalla seconda metà del 2013) e di dispositivi sul mercato da poco tempo (il CC2650 è uscito nel 2015). Ciò ha reso più difficile la prima parte del lavoro, ovvero quella di studio ed analisi dei sistemi e degli ambienti di sviluppo.

Il progetto è stato realizzato utilizzando due diversi linguaggi. Il microcontrollore ARM contenuto all'interno del modulo CC2650 è infatti programmabile in *C*, mentre l'applicazione Android è stata realizzata in *Java*. Sono state utilizzate gran parte delle conoscenze e delle competenze acquisite durante vari corsi seguiti nel corso della carriera universitaria, tra cui *Programmazione a oggetti*, *Programmazione*, *Programmazione di Sistemi Mobile* e *Programmazione di Sistemi Embedded*.

La tesi sarà divisa in otto parti che ripercorrono le fasi di studio, analisi, progettazione e implementazione che hanno portato alla realizzazione del progetto finito.

Nei primi tre capitoli verranno descritte le tecnologie utilizzate nell'ambito di questo progetto.

I successivi quattro descriveranno in maniera approfondita i requisiti richiesti e le conseguenti scelte progettuali ed implementative che hanno portato alla produzione del sistema finito.

L'ultimo capitolo include le operazioni di testing e valutazione dell'applicazione.

Segue una breve conclusione.

Capitolo 1

Introduzione al Bluetooth e al Bluetooth Low Energy

In questo capitolo verranno descritti il protocollo Bluetooth e la versione Bluetooth Low Energy, tecnologie ampiamente utilizzate per la realizzazione di questo progetto.

1.1 Bluetooth

Il Bluetooth è uno standard di trasmissione dati per *WPAN* (acronimo di *Wireless Personal Area Network*, ovvero Reti Personali Senza Fili). Esso fornisce uno standard economico e sicuro per lo scambio di informazioni tra diversi dispositivi a corto raggio e con un basso consumo di energia.

Ormai tantissimi dispositivi contengono chip Bluetooth: telefoni cellulari, stampanti, tastiere, microfoni, computer, sistemi embedded...

Bluetooth è gestito dalla *Bluetooth Special Interest Group* (*SIG*), un gruppo di compagnie che operano in diverse aree: telecomunicazione, computer, networking ed elettronica di consumo. L'IEEE ha inoltre standardizzato il protocollo con il codice IEEE 802.15.1.

Figura 1.1: Logo del Bluetooth

1.1.1 Implementazione

Bluetooth lavora nelle frequenze libere *ISM* (*Industrial, Scientific and Medical*) a 2,4 GHz. In particolare gli scambi di dati avvengono su frequenze comprese tra 2,402 GHz e 2,480 GHz. Su esse sono stabiliti diversi canali di comunicazione: 79 canali con larghezza di banda di 1 MHz per le prime versioni oppure 40 con larghezze di banda di 2 MHz per la versione 4.0. I dispositivi utilizzano tali canali per inviare dati utilizzando una tecnologia chiamata *Frequency-Hopping Spread Spectrum*. Essa permette di scambiare dati a velocità considerevole cambiando il canale di trasmissione, secondo un ordine pseudo-random condiviso tra trasmettitore e ricevitore, con una frequenza fino a 1600 volte al secondo.

A seconda della classe di dispositivo è permessa una potenza massima di trasmissione diversa che consente, di conseguenza, un certo raggio:

Classe	Potenza massima permessa	Portata
Classe 1	100 mW	circa 100 m
Classe 2	2,5 mW	circa 4 m
Classe 3	1 mW	circa 1 m

È da notare che, a causa della bassa potenza, è necessario che i dispositivi che devono comunicare possano vedersi l'un l'altro: ogni ostacolo che blocchi o rifletta le onde radio trasmesse diminuisce drasticamente il raggio di trasmissione.

Altri fattori che intaccano la distanza a cui due o più dispositivi possono comunicare sono la potenza effettiva di trasmissione, la configurazione dell'antenna, le condizioni della batteria e la sensibilità del dispositivo ricevente.

La maggior parte dei dispositivi attualmente prodotti sono di *Classe 2*, mentre i dispositivi di *Classe 1* sono principalmente utilizzati per applicazioni industriali.

Il protocollo Bluetooth è basato sulla trasmissione a pacchetti con struttura *master-slave*. Un *master* può comunicare con un massimo di 7 *slave*, costruendo una rete detta *piconet*. Il clock viene condiviso tra tutti i dispositivi e ha un periodo di 312,5 µs. La trasmissione avviene per blocchi detti *slot*. Una *slot* è formata da 2 cicli di clock: nella *slot* pari il *master* trasmette mentre lo *slave* riceve, viceversa nella *slot* dispari il *master* riceve mentre lo *slave* trasmette. Ogni pacchetto può essere lungo 1, 3 o 5 slot.

A seconda della versione utilizzata si raggiungono diverse velocità di trasmissione. Nella tabella vengono riportati i dati relativi alle versioni più comuni.[1]

Versione	Velocità di trasferimento teorica
1.2	1M bit/s
2.0 EDR	3 Mbps
3.0 HS	24M bit/s
4.0	24 Mbps

L'acronimo *EDR* significa *Enanched Data Rate*, mentre *HS* indica *High Speed*.

1.1.2 Pila protocollare

Bluetooth ha un'architettura a pila di protocolli. Quelli fondamentali e supportati universalmente sono LMP, L2CAP e SDP. Ad essi si aggiungono una serie di protocolli che potrebbero non essere condivisi da tutti i dispositivi. Di seguito una breve descrizione dei più importanti:

- **LMP:** *Link Management Protocol*, utilizzato per inizializzare e controllare il collegamento radio tra i dispositivi.
- **L2CAP:** *Logical Link Control and Adaptation Protocol*, utilizzato per effettuare il multiplexing tra connessioni multiple per dispositivi che si servono di protocolli diversi.
- **SDP:** *Service Discovery Protocol*, permette ai dispositivi di analizzare i servizi offerti da altri device e di leggere i relativi parametri.

[1] Nell'ambito di questo progetto è stata utilizzata la versione 4.0, in particolare il Bluetooth Low Energy.

1.1.3 Connessione

Ogni dispositivo Bluetooth in modalità *discoverable* (ovvero in grado di essere rilevato da altri dispositivi) trasmette, ad intervalli regolari, pacchetti contenenti:

- Nome del dispositivo
- Classe del dispositivo
- Lista dei servizi offerti
- Altre informazioni (es. marca o altre caratteristiche del dispositivo)

Questa operazione è detta *advertising*.

Altri dispositivi possono effettuare la ricerca per rilevare device in modalità *discoverable* e ottenere informazioni su di essi tramite la ricezione dei messaggi di advertising. Sono definiti due tipi diversi di collegamento:

- **ACL:***Asynchronous ConnectionLess*, servizio asincrono senza connessione. Supporta traffico di tipo dati e fornisce un servizio *best-effort*.
- **SCO:***Synchronous Connection Oriented*, servizio sincrono orientato alla connessione. Utilizzato generalmente per il trasporto della voce.

A seconda del tipo di servizio richiesto può essere necessario utilizzare l'uno o l'altro tipo di collegamento. Potrebbe inoltre essere necessario effettuare *pairing* (in italiano *accoppiamento*). Essa è un'operazione che consente di aumentare la sicurezza del collegamento permettendo la connessione solo a specifici dispositivi. Per eseguire l'accoppiamento è infatti necessario l'intervento dell'utente. Una volta eseguita tale operazione si crea un *bond* tra i due dispositivi che permetterà di effettuare nuovamente la connessione senza l'intervento dell'utente.

1.2 Bluetooth Low Energy (*BLE*)

1.2.1 Storia

Il *Bluetooth Low Energy* (*BLE*), in passato conosciuto come *Bluetooth Smart*, è una tecnologia che il gruppo *Bluetooth SIG* ha prodotto appositamente per applicazioni che necessitano di trasmissione wireless con un consumo di energia ancora minore rispetto al Bluetooth *classico* ma con un bitrate maggiore.

Il progetto è stato ideato nel 2001 e commercializzato nel 2006 da Nokia con il nome di *Wibree*. Nel 2007 il marchio è stato poi incluso all'interno delle specifiche Bluetooth. L'integrazione con la versione 4.0 è avvenuta all'inizio del 2010. I primi dispositivi che implementavano tale caratteristica sono usciti nel 2011. Oggi il Bluetooth Low Energy si basa sulle specifiche del Bluetooth 4.1, rilasciate nel Dicembre 2013.

Figura 1.2: Logo del Bluetooth Smart

1.2.2 Applicazioni

Sono stati definiti alcuni profili, con una serie di caratteristiche che i dispositivi devono possedere per essere compatibili e adatti ad una certa applicazione, tra cui troviamo *Health Care*, *Sport and fitness*, *Internet connectivity*, *Generic sensor*, *HID connectivity*, *Proximity Sensor* e *Alerts and time*.

1.2.3 Componenti fondamentali

Il primo componente fondamentale del BLE è il *GAP* (*Generic Access Profile*). Esso si occupa della gestione della connessione e della fase di *advertising*, rendendo

il device visibile al mondo esterno e determinando quali dispositivi possono o non possono interagire con gli altri.

Il GAP definisce diversi ruoli che possono essere ricoperti dai due dispositivi connessi:

- **Peripheral:** il device è visibile agli altri dispositivi (fa, quindi, *advertising*) e può accettare connessioni in ingresso (*slave*). Non è, però, in grado di iniziare una connessione. Questo ruolo è utilizzato, generalmente, per dispositivi di sensoristica.

- **Central:** il device è in grado di ricercare dispositivi visibili e di iniziare la connessione (*master*). Non è, però, in grado di accettare connessioni in ingresso. Questo ruolo è generalmente ricoperto dagli smartphone, dai tablet o dagli altri dispositivi che si connettono ai sensori.

Due dispositivi che ricoprono lo stesso ruolo non sono in grado di connettersi l'un l'altro. Inoltre un dispositivo *Peripheral* è in grado di connettersi solamente ad un dispositivo *Central* alla volta. È però possibile creare una topologia *broadcast* inviando i dati direttamente nel pacchetto di *advertising*, visibile a tutti gli altri dispositivi.

Nelle figure 1.3 e 1.4 è possibile osservare esempi delle due topologie possibili.

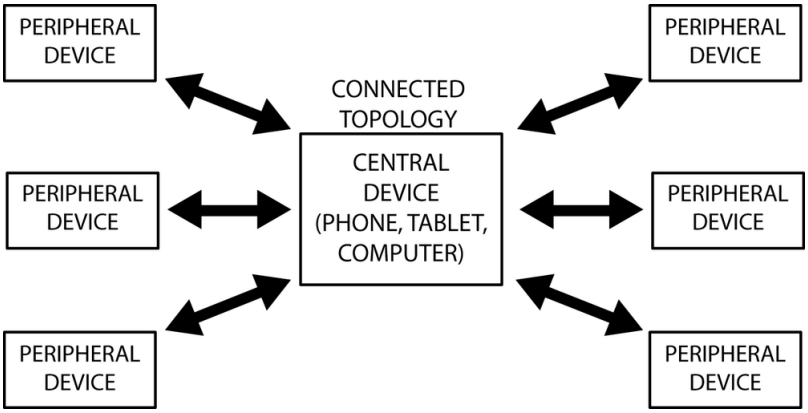

Figura 1.3: Topologia connessa: un dispositivo *Central* è connesso e scambia dati con più dispositivi *Peripheral*.

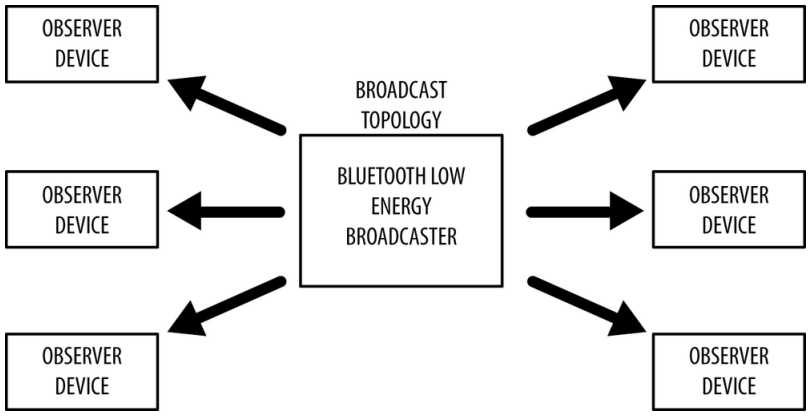

Figura 1.4: Topologia broadcast: un dispositivo *Peripheral* invia dati contemporaneamente a più dispositivi *Central*.

Tutti i servizi che utilizzano la tecnologia BLE si basano sul *GATT* (*Generic Attribute Profile*). Esso non è altro che un'interfaccia software che definisce come i dispositivi possano inviare e ricevere dati, descrivendo i concetti di *Servizio* e *Caratteristica*. Il GATT sfrutta a sua volta l'*ATT* (*Attribute Protocol*), che viene utilizzato per contenere i dati dei Servizi e delle Caratteristiche che il *GATT* mette a disposizione all'esterno.

I dati relativi a Servizi e Caratteristiche sono memorizzati in un'apposita *look-up-table* usando un identificatore lungo 16 byte chiamato *UUID*. Di questi 16 byte i primi 4 vengono scelti dal programmatore mentre gli altri sono stabiliti dal dispositivo stesso. Poiché nel BLE è importante limitare al massimo la quantità di dati trasmessi, il SIG ha stabilito uno *UUID base* formato dai primi 12 byte dell'*UUID* completo. In questo modo non è necessario trasmettere ogni volta l'intero *UUID*, ma è sufficiente comunicare solamente gli ultimi 4 byte.

1.2.4 Profili, Servizi e Caratteristiche

Lo scambio di dati tra dispositivi BLE è basato su oggetti di alto livello che prendono il nome di Profili, Servizi e Caratteristiche.

Si può accedere ad ogni oggetto solo nelle modalità previste dal progettista (sola lettura, sola scrittura, entrambe o nessuna) e, se necessario, solo dopo aver ottenuto un'autorizzazione.

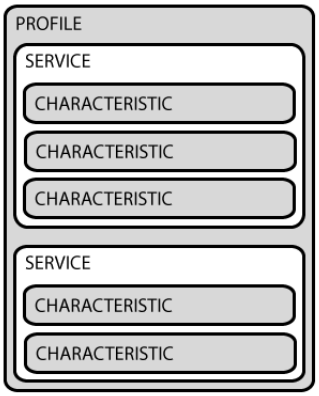

Figura 1.5: Schema dell'interfaccia messa a disposizione dal GATT.

Profili

Collezione di Servizi definita dal *SIG* (es. *Alert Notification Profile*, *Blood Pressure Profile*, *Find Me Profile*) o dal progettista della periferica. Essi combinano Servizi che mettono a disposizione informazioni di vario tipo.

Servizi

Identificati da uno *UUID*, fungono da contenitori per una serie di caratteristiche.

Caratteristiche

Sono gli oggetti di livello più basso. Ognuna di esse incapsula un puntatore ai dati (dati singoli di varia lunghezza, come numeri interi o decimali, oppure array di valori). Esse sono identificate da uno *UUID*.

Oltre ai permessi di lettura e/o scrittura esistono altre proprietà memorizzate nel descrittore della caratteristica che permettono, per esempio, di definire se il valore può essere inviato *broadcast* (ovvero inserito all'interno dei pacchetti di

advertising) oppure di notificare automaticamente il dispositivo connesso della disponibilità di un nuovo dato.

1.2.5 Advertising, connessione e scambio di dati

Advertising

In questa fase il dispositivo BLE con ruolo *Peripheral* invia, ad intervalli regolari, un pacchetto di *advertise*. Essi hanno dimensioni comprese tra 6 e 37 byte e contengono informazioni relative al dispositivo che li ha inviati.

Tali pacchetti sono inviati su 3 dei 40 canali dedicati alla funzione: 37, 38, 39. Se un altro dispositivo è interessato a connettersi con il device che sta facendo *advertising* può effettuare una richiesta di scansione per richiedere dati aggiuntivi.

L'intervallo di *advertising* è impostato dal programmatore e può variare tra 20 ms e 10,24 s in intervalli di 0,625 ms. Viene inoltre aggiunto un intervallo pseudo-random compreso tra 0 e 10 ms per ridurre la possibilità di collisioni tra *advertisement* di differenti device. È importante selezionare con cura la durata dell'intervallo per bilanciare ragionevolmente il consumo di energia e la velocità nel farsi riconoscere da un eventuale altro dispositivo.

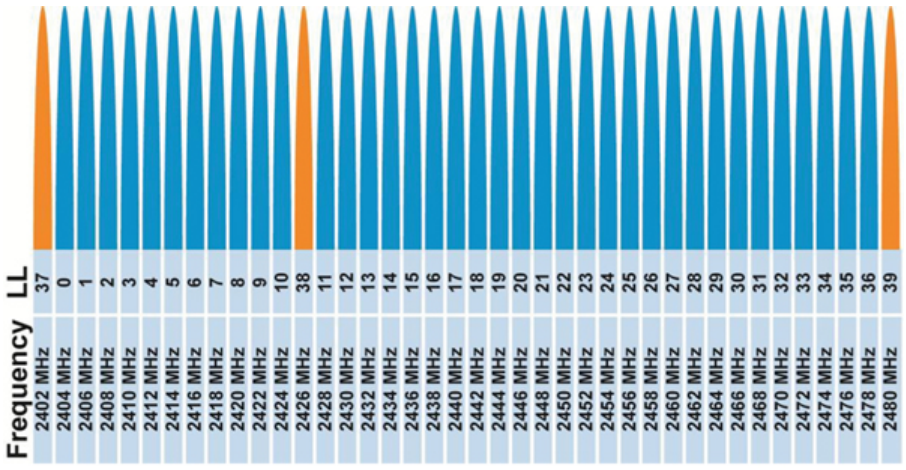

Figura 1.6: Schema dei canali utilizzati dal BLE. In arancione sono evidenziati i canali dedicati all'*advertising*, mentre gli altri vengono utilizzati per la trasmissione di dati.

Connessione e scambio di dati

Una volta che il device *Central* ha richiesto informazioni aggiuntive al device *Peripheral* può stabilire una connessione. Devono essere a questo punto stabiliti alcuni parametri: l'intervallo di connessione e la *Slave Latency*.

I dispositivi mantengono in piedi la connessione scambiando periodicamente dati anche se non c'è nulla da trasmettere: questa operazione è detta *Connection event*. Il periodo di tempo che intercorre tra uno scambio e l'altro è dettato dall'intervallo di connessione. Esso può variare tra 7,5 ms e 4 s ed è scelto indipendentemente.

Per risparmiare ulteriormente energia il dispositivo *Peripheral*, qualora non abbia informazioni da scambiare, può decidere di non rispondere all'evento di connessione. Il numero di eventi di connessione che il device *Peripheral* può ignorare senza far cadere la connessione viene definito dal parametro detto *Slave Latency*.

In questo modo i dispositivi si troveranno a comunicare più frequentemente quando è necessario inviare dati, mentre più raramente quando ciò non è necessario.

1.2.6 Confronto con altre tecnologie simili

Bluetooth Low Energy non è l'unica tecnologia che permette di creare *Wireless Personal Area Network*. Tra esse le più famose sono:

- WiFi Direct
- NFC (Near Field Communication)
- ZigBee

WiFi Direct

WiFi Direct è una tecnologia che permette di connettere più periferiche l'una con l'altra senza necessità di un access point e con una velocità di trasmissione comparabile con quella del WiFi "classico". Solo uno dei dispositivi della rete dev'essere compatibile con tale tecnologia mentre gli altri possono essere normali dispositivi WiFi.

Le principali differenze con il Bluetooth Low Energy sono:[2]

Caratteristica	Bluetooth Low Energy	WiFi Direct
Velocità di trasmissione	25 Mbps	250 Mbps
Distanza di trasmissione	100 m	200 m

Inoltre si è stimato che il BLE permetta di consumare circa il 3% dell'energia consumata dal WiFi Direct per compiere le stesse operazioni.

NFC

NFC (*Near Field Communication*) è una tecnologia che permette di scambiare dati a distanza ravvicinata. Consente, tra le altre cose, di alimentare il dispositivo *slave* direttamente tramite le onde radio.

I difetti principali di questa tecnologia sono la limitatissima portata di trasmissione (massimo 10 cm) e l'assenza di un sistema di autenticazione. Alcuni dei vantaggi sono, invece, il consumo irrisorio di energia e la semplicità di utilizzo (non è necessario inserire PIN o eseguire altre procedure di inizializzazione della connessione).

ZigBee

ZigBee è uno standard di comunicazione curato dalla *ZigBee Alliance* che prevede l'uso di antenne digitali a bassa potenza e basso consumo. Lo standard è progettato per l'utilizzo con sistemi embedded che richiedano un basso *transfer rate* e bassi consumi. A differenza del BLE permette di creare reti più complesse (reti *mesh*, ovvero reti a maglia). È tuttavia molto meno diffuso, in particolare non esiste nessun dispositivo mobile (smartphone o tablet) che lo implementa.

Perché scegliere Bluetooth Low Energy

Per un'applicazione come quella che è stata realizzata in questo elaborato una delle caratteristiche fondamentali è il basso consumo di energia. Il sensore dovrà,

[2]I valori espressi sono i massimi teorici. In realtà la velocità di trasmissione a regime può essere minore.

infatti, essere alimentato a batteria e un limitato consumo energetico permette di aumentarne la durata. È tuttavia necessario avere una portata di trasmissione superiore a quella permessa dall'NFC.

L'interfaccia sviluppata deve essere inoltre universale, sia nel senso che deve permettere l'utilizzo di diversi tipi di sensore, sia nel senso che deve essere compatibile con il più grande numero di dispositivi possibile.

È quindi evidente che la migliore tecnologia tra quelle appena descritte è quella del Bluetooth Low Energy.

Capitolo 2

Introduzione ad Android

Android è un sistema operativo per dispositivi mobile basato sul kernel Linux e sviluppato da Google.

È disegnato in particolare per dispositivi mobili con schermo touchscreen, come smartphone e tablet, ma ne esistono versioni personalizzate per TV (Android TV), auto (Android Auto) e smartwatch (Android Wear).

Android è un progetto Open Source: il codice sorgente viene rilasciato pubblicamente e, in seguito, personalizzato dai vari produttori per farlo funzionare correttamente sul loro hardware. Alcuni produttori, tra cui Google stesso, utilizzano direttamente il codice così come è stato rilasciato (ne è un esempio la serie *Nexus*).

Figura 2.1: Logo di Android

2.1 Storia

La società *Android Inc.* viene fondata a Palo Alto (California) nel 2003 da Andy Rubin, Rich Miner, Nick Sears e Chris White. Inizialmente lo scopo di tale società era di produrre un sistema operativo per fotocamere digitali, ma una volta realizzato che il mercato in questo campo non era sufficientemente ampio il progetto è stato re-indirizzato verso un sistema operativo per dispositivi mobili in grado di competere con *Symbian* (Nokia) e *Windows Mobile* (Microsoft).

La compagnia è stata in seguito acquisita da Google nel 2005 e il team guidato da Andy Rubin ha continuato lo sviluppo del sistema operativo fino al 2013, anno in cui il leader del progetto è stato sostituito da Larry Page.

2.2 Architettura

2.2.1 Kernel

Android è basato sul kernel Linux. Oggi la versione utilizzata è, a seconda della release del sistema operativo, la versione 3.4 o 3.10. Google effettua, tuttavia, alcune modifiche al kernel originale per migliorarne alcune caratteristiche, ad esempio per ridurne il consumo energetico.

Inoltre, a differenza delle altre distribuzioni Linux, all'utente non sono dati i permessi di *root*. Ciò permette, tra le altre cose, di rendere non modificabili alcune partizioni (un esempio è */system*, che contiene i dati del sistema operativo).

2.2.2 Applicazioni

Al di sopra del kernel Linux girano middleware, librerie e API scritte in C. Il software applicativo è in esecuzione su un apposito framework ed è programmabile in Java.

A differenza dell'edizione desktop di Java, Android non utilizza la classica *JVM* (*Java Virtual Machine*), ma una macchina virtuale appositamente creata. In particolare le due *Virtual Machine* utilizzate sono:

- **Dalvik VM:** utilizzata fino alla versione 4.4 Kit Kat, prevede un compilatore di tipo *JIT* (*Just in time*) per eseguire codice detto *dex* (*Dalvik Executable*).
- **ART:** acronimo di *Android RunTime*, utilizzata a partire dalla versione 4.4, a differenza della versione precedente compila il bytecode Java in codice macchina al momento dell'installazione dell'applicazione, permettendo così una maggiore velocità di esecuzione.

2.3 Versioni di Android

Negli anni sono state rilasciate diverse versioni di Android. Ognuna di esse rende disponibili una versione aggiornata di API e funzionalità sempre più avanzate.

Una nota particolare va alla versione Honeycomb, realizzata in maniera specifica per dispositivi con schermo grande (tablet).

Versione	Nome comune	Anno di rilascio
5.0 - 5.1	Lollipop	2014-2015
4.4	Kit Kat	2013-2014
4.1 - 4.2	Jelly Bean	2012-2013
4.0	Ice Cream Sandwich	2011-2012
3.0	Honeycomb	2011-2012
2.3	Gingerbread	2010-2011
2.2	Froyo	2010-2011
2.0 - 2.0.1 - 2.1	Eclair	2009-2010
1.6	Donut	2009
1.5	Cupcake	2009
1	//	2008-2009

2.4 Programmazione

Android viene programmato generalmente in *Java*, utilizzando l'apposito framework fornito dal sistema operativo.

L'IDE ufficiale di Android è *Android Studio*. Esso si basa su *IntelliJ IDEA* prodotto dalla *JetBrains*. Attualmente la versione disponibile è la 14.1.

2.5 Android e BLE

La prima versione di Android a supportare completamente le specifiche del Bluetooth Low Energy è la versione 4.3 Jelly Bean.[1]

Le classi principali da utilizzare nella programmazione di software in grado di interagire con il modulo BLE contenuto all'interno della maggior parte dei dispositivi Android sono:

- **BluetoothGatt:** Fornisce API per interagire con i profili del GATT contenuto all'interno del sistema operativo. Permette, tra le altre cose, di effettuare la connessione, di analizzare i servizi e le caratteristiche di un dispositivo connesso e di leggere o scrivere dati da esse.

- **BluetoothGattService:** Rappresenta un Service (già descritto in precedenza). Presenta metodi atti a ottenere la collezione di Caratteristiche che si trovano al suo interno.

- **BluetoothGattCharacteristic:** Rappresenta una Caratteristica (già descritta in precedenza). Presenta metodi per leggere e/o scrivere i dati (a seconda dei permessi forniti dalla caratteristica stessa) in vari formati, tra cui interi con e senza segno, stringhe, ...

- **BluetoothAdapter:** Fornisce API per interagire con il modulo Bluetooth del device. Per esempio consente di effettuare la ricerca di dispositivi (sia in modalità classica che in modalità BLE).

[1] API Level 18, requisito minimo per il funzionamento dell'applicazione creata in questo progetto

Per poter utilizzare le funzionalità del Bluetooth è necessario fornire all'applicazione alcuni permessi inserendoli nel *manifest*. In particolare:

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
```

Listing 2.1: Permessi necessari all'applicazione per poter utilizzare il Bluetooth.

Inoltre è possibile limitare il funzionamento dell'applicazione stessa ai soli dispositivi in grado di supportare il Bluetooth Low Energy indicando la richiesta dell'apposita caratteristica:

```
<uses-feature android:name="android.hardware.bluetooth_le"
    android:required="true"/>
```

Listing 2.2: Riga di codice che verifica l'esistenza di un modulo BLE integrato nel dispositivo Android su cui esegue l'applicazione.

Concesse tali autorizzazioni e dichiarata la necessità della caratteristica è possibile, tramite il *BluetoothManager*, ottenere un'istanza del *BluetoothAdapter*. Si può fare ciò utilizzando appositi metodi forniti dal framework, in particolare:

```
final BluetoothManager bluetoothManager = (BluetoothManager)
    getSystemService(Context.BLUETOOTH_SERVICE);
BluetoothAdapter bluetoothAdapter = bluetoothManager.getAdapter();
```

Listing 2.3: Ottenimento del BluetoothAdapter, che funge da interfaccia con il GATT.

Una volta ottenuta un'istanza del *BluetoothAdapter* è possibile interagire con esso per avviare la ricerca di altri dispositivi. Poiché i dispositivi su cui funzioneranno queste applicazioni sono generalmente alimentati a batteria (e comunque, date le peculiarità di risparmio energetico del BLE) è consigliabile eseguire la scansione solo per un tempo prefissato. Tale operazione è infatti pesante dal punto di vista del consumo energetico.

Le API messe a disposizione per compiere queste operazioni sono:

```
bluetoothAdapter.startLeScan(LeScanCallback callback); //Avvia la scansione
bluetoothAdapter.stopLeScan(LeScanCallback callback); //Interrompe la scansione
```

Listing 2.4: Righe di codice per l'avvio e la terminazione della scansione.

Ognuna di esse mette a disposizione diversi *callback* che verranno richiamati in caso di verifica di eventi particolari (tra cui il riconoscimento di un dispositivo o l'interruzione imprevista dell'operazione).

È a questo punto possibile stabilire la connessione, ottenendo un'istanza del GATT e, se necessario, il *bonding*. Una volta effettuate tali operazioni si possono ricercare, all'interno dell'oggetto *BluetoothGatt*, i servizi e le caratteristiche che il dispositivo espone all'esterno. Le API messe a disposizione in questo caso sono:

```
BluetoothGatt bluetoothGatt = device.connectGatt(Context context, boolean
    autoConnect, BluetoothGattCallBack callback); //Connessione al dispositivo
bluetoothGatt.close(); //Chiusura della connessione

BluetoothGattService service = device.getServices(UUID serviceUUID); //Ottiene
    il service con un certo UUID
BluetoothGattCharacteristic characteristic = service.getCharacteristic(UUID
    characteristicUUID); //Ottiene la caratteristica con un certo UUID

characteristic.setValue(byte[] value); //Esempio di metodo per scrivere un
    valore sull'attributo prescelto
boolean status = bluetoothGatt.writeCharacteristic(characteristic);

characteristic.getValue(byte[] value); //Esempio di metodo per ottenere un
    valore dall'attributo prescelto
```

Listing 2.5: Creazione della connessione e I/O sul modulo BLE.

Le API mettono inoltre a disposizione metodi utilizzabili per rilevare le notifiche provenienti dal dispositivo connesso. In particolare:

```
bluetoothGatt.setCharacteristicNotification(
   BluetoothCharacteristic characteristic, boolean enabled);
```

Listing 2.6: Registrazione per l'ottenimento di notifiche dal dispositivo.

Diventa quindi possibile collegare un apposito descrittore alla caratteristica per poter ottenere dei *callback* al momento della ricezione della notifica.

Ulteriori informazioni riguardanti questo tema possono essere reperite in rete sul sito *developer.android.com*, dove è presente la documentazione completa delle classi e dei metodi descritti in precedenza.

Capitolo 3

Introduzione al modulo Texas Instrument CC2650

Figura 3.1: Immagine della base di debug SmartRF06 su cui è montato il modulo BLE CC2650

Texas Instrument CC2650 è una piattaforma *MCU* (*Micro Controller Unit*) basata su un SoC ARM. Esso contiene all'interno svariate periferiche tra cui bus di vario tipo (*I2C* e *UART*), un'antenna Bluetooth, due *ADC* (*Analog to Digital Converter*, ...). Il dispositivo è stato progettato per diversi tipi di applicazioni, tra cui accessori per smartphone, equipaggiamento per automazione e controllo casalingo, sport o fitness e controllo remoto.

Per essere programmato e per potersi interfacciare con il modulo Bluetooth viene utilizzata la scheda di debug *SmartRF06*. Essa fornisce un'interfaccia seriale o USB che permette di caricare il software scritto sul PC nella memoria del microcontrollore. Contiene inoltre un display LCD, una serie di pulsanti e l'alimentazione dedicata per il CC2650.

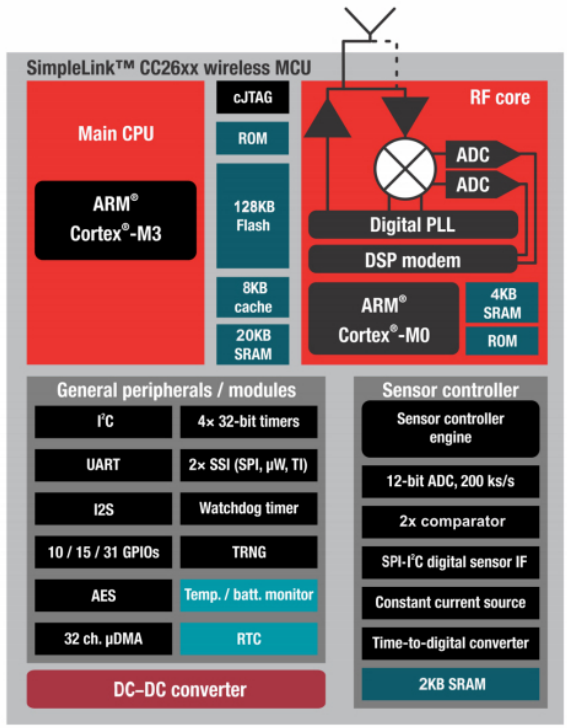

Figura 3.2: Diagramma a blocchi del microcontrollore.

Di seguito alcune delle caratteristiche più importanti:

- **Processore:** CC2650 monta un processore ARM Cortex-M3 (Architettura Harvard con bus dati e programma separati) con una frequenza massima di clock pari a 48 MHz.

- **Memoria:** sono disponibili 8 Kb di memoria flash e RAM.

- **Alimentazione:** l'alimentazione può essere fornita tramite USB (5 V) o tramite batterie (3 V). Il dispositivo è progettato per avere il minor consumo energetico possibile.

- **Sensor Controller:** il dispositivo contiene un'interfaccia apposita in grado di controllare i sensori indipendentemente dalla CPU. Contiene inoltre un ADC, un comparatore analogico e BUS *SPI* e I^2C.

- **Timer:** all'interno del CC2650 si trova un timer di sistema a 24 bit con una frequenza di clock programmabile e una serie di timer general purpose da 16 o 32 bit utilizzabili per applicazioni varie (ad esempio generazione di segnale PWM).

- **Pin di I/O:** il modulo mette a disposizione 31 pin GPIO in grado di erogare fino a 8 mA di corrente. Essi possono essere configurati come trigger per interrupt e il loro stato può essere mantenuto anche in modalità di risparmio energetico.

- **Analog-Digital-Converter:** CC2650 integra un ADC 12 bit a 8 canali in grado di campionare con una frequenza fino a 200 kHz.

Il modulo CC2650 comunica con la base SmartRF06 tramite protocollo JTag. La base viene poi collegata al computer tramite un semplice cavo USB.

3.1 Programmazione

Il sistema viene programmato in *C*.

L'ambiente di sviluppo fornito dalla Texas Instrument è *Code Composer Studio* (*CCS*), basato su *Eclipse C++*. Il compilaltore utilizzato è *gcc*.

3.1.1 Struttura di un programma tipico

Un tipico programma per CC2650 è formato da due parti principali:

- **Bluetooth Stack:** comprende la parte di configurazione che permette la comunicazione via BLE, in particolare GATT e GAP. Fornisce le API che il programma potrà utilizzare. Tra i valori principali che possono essere settati all'interno dello stack possiamo trovare[1]
  - **Connection Interval:** tempo che trascorre tra due eventi di connessione.
  - **Slave latency:** numero di eventi di connessione che la periferica può ignorare senza far decadere la connessione.
  - **Supervision timeout:** tempo massimo che può trascorrere tra due connection events senza che la connessione venga considerata interrotta.
- **Applicazione:** è il programma vero e proprio che consente di eseguire le operazioni richieste. Utilizza i servizi offerti dallo stack per comunicare tramite il Bluetooth attraverso un modulo chiamato *ICall*. Le parti fondamentali del programma sono:
  - **Main:** Inizializza il programma e crea lo scheduler.
  - **Scheduler:** Gestisce l'allocazione della CPU ai vari task. Lo scheduler è di tipo *preemptive*.
  - **Task:** Parti di programma che eseguono indipendentemente l'uno dall'altro secondo l'ordine dettato dallo scheduler. Ogni task ha un certo valore di priorità e una sua area di memoria a disposizione per memorizzare le variabili locali. I task possono comunicare tra loro scambiando messaggi su apposite code, utilizzando dei semafori per la sincronizzazione.

    Esiste un task particolare che si occupa della comunicazione BLE che deve avere priorità massima.

[1] È possibile trovare una spiegazione più approfondita di questi termini nel capitolo 1

Capitolo 4

Analisi dei requisiti

In questo capitolo verranno descritti i requisiti che l'applicazione deve rispettare e le funzionalità che deve implementare.

4.1 Descrizione generale

Il software progettato e sviluppato è composto da due applicazioni in grado di interagire scambiando dati provenienti da un sensore analogico e di mostrarli all'utente.

L'azienda che ha commissionato questo lavoro prevede di utilizzare l'interfaccia così creata per realizzare sensori ambientali in grado di misurare grandezze fisiche di vario tipo (ad esempio luminosità, pressione e temperatura) tramite uno o più sensori. È quindi importante mantenere l'applicazione il più universale possibile, in modo che sia utilizzabile con apparecchiature diverse senza rendere necessaria una modifica complessa al software, ma semplicemente sostituendo l'hardware del sensore ed, eventualmente, aggiungendo task che si occupino della lettura.

La parte di trasmissione e l'interfaccia da essa esposta ad un eventuale dispositivo in grado di leggere i dati dovrà essere realizzata su TI CC2650 ed è da considerarsi prioritaria rispetto alla parte di acquisizione e memorizzazione dei dati, da realizzare su piattaforma Android.

4.1.1 Funzioni del prodotto

Data la duplice natura dell'applicazione le specifiche verranno descritte separatamente per i due moduli richiesti.

Lato lettura e trasmissione dei dati

- Lettura dei dati da un sensore analogico tramite ADC.
- Trasmissione dei dati ad intervalli di tempo personalizzabili.
- Impostazione dell'intervallo di trasmissione da remoto.
- Possibilità di capire lo stato del dispositivo (*advertising*, *connesso* o *disattivo* dall'esterno).

Lato ricezione ed interpretazione dei dati

- Connessione al dispositivo e ricezione dei dati dal modulo di lettura e trasmissione.
- Impostazione a distanza dell'intervallo di tempo di trasmissione.
- Visualizzazione dei dati su schermo in forma numerica.
- Memorizzazione dei dati ricevuti.
- Consultazione dei dati memorizzati in forma tabulare.
- Correzione dei dati in ricezione impostando un valore di scostamento (*offset*) e di guadagno.

Di seguito verranno elencati alcuni requisiti opzionali, ma comunque realizzati all'interno del progetto finale:

- Visualizzazione dei dati su schermo in forma grafica (su piano cartesiano).
- Consultazione dei dati memorizzati in forma grafica.

- Esportazione dei dati memorizzati in un formato comune e leggibile da altri dispositivi e applicazioni.

4.1.2 Requisiti espressi dall'azienda

Il software progettato deve rispettare alcune caratteristiche aggiuntive molto importanti per il campo in cui verrà, in futuro, utilizzato, ovvero:

- **Consumo energetico ridotto:** in vista di un'alimentazione a batteria, è importante massimizzare la sua durata per evitare di doverla sostituire frequentemente.

- **Universalità:** poiché il campo di utilizzo non è ancora molto specifico è importante che l'applicazione possa essere utilizzata per leggere dati provenienti da uno o più sensori di diverso tipo (l'applicazione deve risultare modulare per permettere l'aggiunta di ulteriori componenti).

4.1.3 Ambiente di utilizzo

Il modulo di lettura e trasmissione dei dati dovrà poter essere eseguito sul dispositivo CC2650 prodotto da Texas Instrument.

Il modulo di ricezione ed interpretazione dei dati dovrà funzionare su un qualunque dispositivo con Sistema Operativo Android in grado di supportare la tecnologia Bluetooth Low Energy. Non si esclude, in futuro, la realizzazione di un sistema di lettura ed analisi per dispositivi con Sistema Operativo iOS.

4.1.4 Linguaggi e piattaforme di sviluppo

La scelta dei linguaggi e delle piattaforme di sviluppo è vincolata al tipo di dispositivo scelto per la realizzazione dei moduli:

- **C** per il modulo di lettura e trasmissione (capitolo 3, sezione 1)

- **Java** per il modulo di ricezione ed interpretazione (capitolo 2, sezione 4)

4.2 Interfacciamento con l'esterno

4.2.1 Interfaccia software

L'interfaccia software esposta dal modulo Bluetooth è il cuore di questo progetto. Essa deve risultare quanto più universale e modulare. Dovrà anche garantire prestazioni sufficienti.

4.2.2 Interfaccia utente

Il software deve fornire un'interfaccia utente intuitiva e funzionale, anch'essa universale e scalabile come l'interfaccia software.

4.3 Funzionalità del sistema

Di seguito verranno descritte in maniera più dettagliata le funzionalità del sistema e i requisiti funzionali che esse implicano.

4.3.1 Lato lettura e trasmissione dei dati

Advertising e connessione

Il software deve essere in grado di inviare segnali di *advertising* tramite il modulo BLE per rendersi rilevabile da dispositivi esterni. Deve inoltre essere in grado di stabilire una connessione ed effettuare l'accoppiamento con essi per permettere lo scambio di dati.

Per mantenere limitato il consumo energetico è necessario che, dopo un certo tempo, l'*advertising* venga automaticamente disabilitato.

Requisiti funzionali derivanti:

- Interazione con il modulo BLE.
- Invio di segnali di *advertising*.

- Disabilitazione automatica dell'*advertising* tramite apposito controllo qualora dopo un certo intervallo di tempo non sia stata stabilita una connessione.
- Riabilitazione dell'*advertising* tramite apposito controllo hardware.
- Stabilimento di una connessione.
- Effettuazione del pairing.

Scambio di dati su Bluetooth Low Energy

Il software deve essere in grado di inviare i dati generici letti dal sensore tramite il modulo BLE. Deve inoltre rendere possibile la ricezione di dati per la selezione della frequenza di trasmissione.

Requisiti funzionali derivanti:

- Interazione con il modulo BLE.
- Invio di dati via BLE.
- Ricezione di dati tramite BLE.

Lettura di dati dal sensore

Il software deve essere in grado di leggere i dati provenienti dal sensore collegato all'ADC a 12 bit integrato nel TI CC2650 (dettagli aggiuntivi descritti nel capitolo 3).

Requisiti funzionali derivanti:

- Interazione con i pin GPIO.
- Lettura e campionamento di dati dall'ADC.
- Preparazione dei dati per la trasmissione (codifica in una modalità compatibile con dati di diverso tipo).

4.3.2 Lato ricezione ed interpretazione dei dati

Rilevamento dei dispositivi

Il software deve essere in grado di effettuare una scansione BLE per verificare la presenza di dispositivi in fase di *advertising*.

Per mantenere limitato il consumo energetico è necessario che la scansione venga automaticamente terminata se entro un certo tempo limite, non è stata stabilita una connessione.

Requisiti funzionali derivanti:

- Ricerca automatica di dispositivi connessi.
- Disabilitazione della scansione dopo un certo intervallo di tempo.
- Riabilitazione della scansione tramite apposito controllo.

Selezione del dispositivo

L'utente deve essere in grado di selezionare un dispositivo tra quelli rilevati durante la fase di scansione.

Requisiti funzionali derivanti:

- Visualizzazione della lista dei dispositivi connessi.
- Visualizzazione dell'identificatore e delle caratteristiche del dispositivo connesso.
- Selezione di un dispositivo tra quelli presenti nella lista.

Connessione al dispositivo e pairing

Il software deve essere in grado di stabilire una connessione con il dispositivo selezionato dall'utente. Deve inoltre permettere di effettuare il pairing tra i due dispositivi per permettere lo scambio dei dati.

Requisiti funzionali derivanti:

- Connessione al dispositivo selezionato dall'utente.
- Se non già effettuato, pairing con il dispositivo connesso.

Ricezione di dati dal dispositivo

Il software deve essere in grado di ricevere dati dal dispositivo ad intervalli regolari.

Requisiti funzionali derivanti:

- Mantenimento della connessione.
- Ricezione di dati provenienti dal dispositivo.

Elaborazione dei dati ricevuti

Il software deve essere in grado di eseguire semplici elaborazioni sui dati ricevuti prima di mostrarli a schermo, in particolare l'applicazione di un valore di scostamento e di guadagno.

Requisiti funzionali derivanti:

- Esecuzione di semplici operazioni di elaborazione in tempo reale sui dati ricevuti.

Invio dei dati di configurazione

Il software deve essere in grado di inviare dati di configurazione al dispositivo, in particolare la selezione dell'intervallo di invio dei dati. L'utente deve poter scegliere tale valore. Esso deve essere mantenuto nella memoria del device Android in modo che possa essere re-impostato in una sessione successiva.

Requisiti funzionali derivanti:

- Possibilità di scelta dell'intervallo di trasmissione.
- Invio del dato prescelto.
- Memorizzazione dell'intervallo di trasmissione.

Visualizzazione in tempo reale

L'utente deve essere in grado di visualizzare, in tempo reale, i dati ricevuti dal dispositivo. In particolare devono essere mostrati in forma numerica e in forma grafica.

Requisiti funzionali derivanti:

- Visualizzazione dei dati su schermo in forma numerica.
- Visualizzazione dei dati su schermo in forma grafica.

Memorizzazione dei dati

Il software deve gestire la memorizzazione dei dati ricevuti per poterli consultare in un secondo momento. Non vi sono specifiche aggiuntive sul formato di salvataggio.

Requisiti funzionali derivanti:

- Integrazione di un sistema per la memorizzazione persistente delle informazioni.
- Salvataggio dei dati in modo sicuro e senza minare le prestazioni dell'applicazione.

Consultazione dei dati memorizzati

L'utente deve poter consultare i dati memorizzati visualizzandoli, suddivisi per sessione, nelle due forme, grafica e tabulare. Deve inoltre poter eliminare dati di sessioni non più interessanti.

Requisiti funzionali derivanti:

- Lettura dei dati dal supporto di memorizzazione persistente.
- Visualizzazione dell'elenco delle sessioni memorizzate.
- Visualizzazione dei dati di una sessione in forma grafica.
- Visualizzazione dei dati di una sessione in forma numerica.
- Possibilità di cancellare i dati relativi ad una sessione.

Capitolo 5

Analisi

In questo capitolo verrà esaminata la suddivisione in moduli del progetto nel suo complesso. Verranno in seguito descritte, in modo più dettagliato, la struttura e le funzionalità di ogni modulo.

5.1 Moduli del progetto

Il progetto è stato diviso in due macro-moduli, a loro volta divisi in sotto-moduli:

- **Macro-modulo di lettura e trasmissione**
  - Trasmissione dei dati
  - Lettura dei dati dall'ADC
- **Macro-modulo di ricezione ed interpretazione**
  - Modulo di gestione del Bluetooth
  - Modulo di gestione dei dati
  - Modulo di connessione e visualizzazione in tempo reale
  - Modulo di consultazione dei dati memorizzati

5.2 Analisi del macro-modulo di lettura e trasmissione

Il modulo si occupa della lettura e della trasmissione dei dati letti tramite BLE. Gestisce, inoltre, lo stabilimento della connessione e del pairing e controlla l'intervallo a cui il dispositivo invierà i dati al device connesso.

Dovrà essere implementato su TI CC2650 e realizzato in C.

5.2.1 Casi d'uso

In figura 5.1 è possibile vedere il diagramma dei casi d'uso di questa parte di progetto.

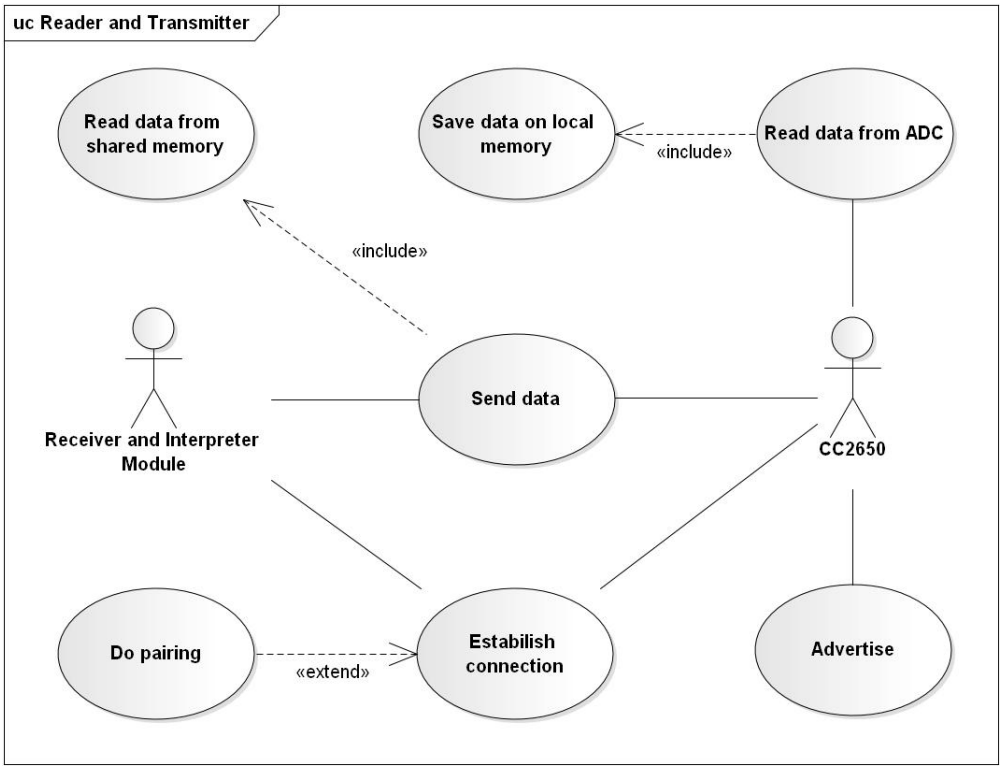

Figura 5.1: Diagramma dei casi d'uso del macro-modulo di lettura e trasmissione.

Gli attori rappresentano i due macro-moduli, in particolare:

- Macro-modulo di lettura e trasmissione: *CC2650*
- Macro-modulo di ricezione ed interpretazione: *Receiver and Interpeter Module*

Di seguito una breve descrizione dei singoli casi d'uso.

Send data* e *Read data from shared memory

Una volta trascorso l'intervallo di trasmissione impostato dall'utente il dispositivo deve inviare i dati al device Android (*Send data).* Tale operazione include la lettura di dati dalla memoria condivisa tra il modulo di trasmissione e quello di lettura (*Read data from shared memory).* Potrebbe essere necessario attendere la disponibilità dei dati.

Read data from ADC* e *Save data on shared memory

Il dispositivo deve leggere il valore proveniente dal sensore (*Read data from ADC*) e salvare il dato letto all'interno della memoria condivisa (*Save data on local memory).* Per evitare problemi di concorrenza è necessario fare in modo che durante la scrittura sulla memoria nessun altro flusso di controllo possa accedere a tale area.

Advertise

Il device è in fase di *Advertising* (ulteriori informazioni possono essere trovate nella sezione 1.2.5), ovvero è in grado di essere rilevato da altri dispositivi in fase di ricerca.

Estabilish connection* e *Do pairing

A seguito di una richiesta di connessione proveniente dall'applicazione Android, il dispositivo risponde (*Connection)* e, se necessario, avvia la procedura per effettuare l'accoppiamento (*Pairing*).

5.2.2 Architettura del macro-modulo

La struttura più adatta per la modellazione di questo modulo è quella della macchina a stati finiti asincrona (*FSM, Finished State Machine*). È infatti possibile individuare una serie di condizioni che si susseguono durante l'esecuzione del programma.

Nella figura 5.2 è possibile osservare una rappresentazione grafica di tali stati.

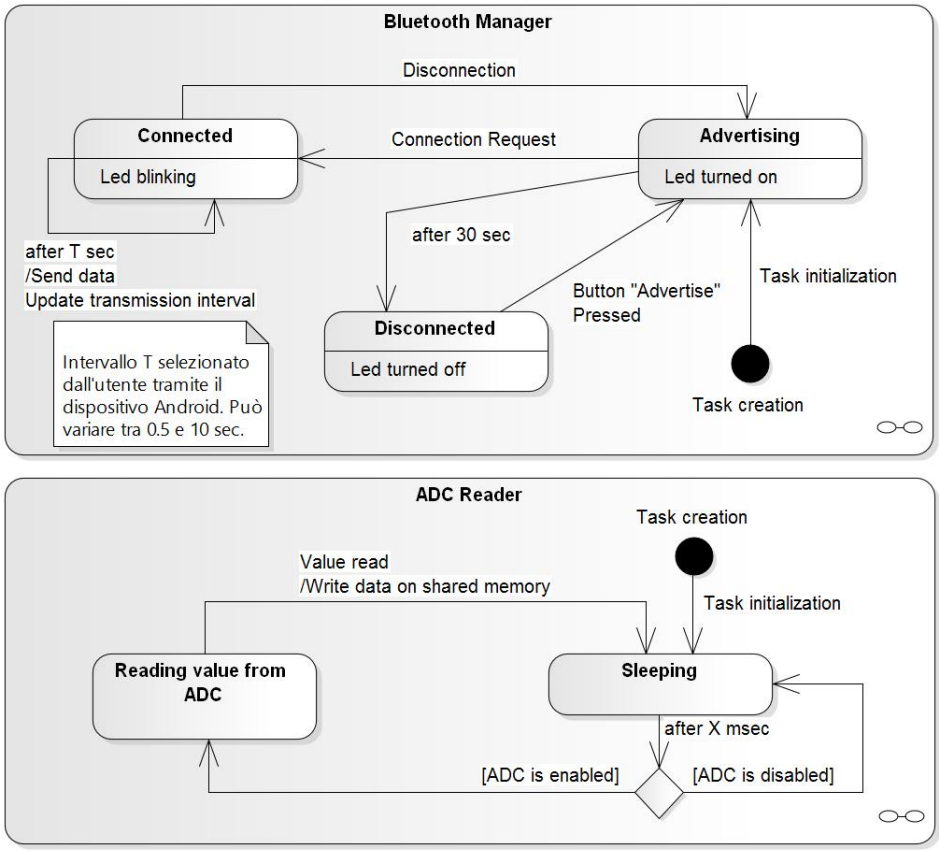

Figura 5.2: Macchina a stati finiti del macro-modulo di lettura e trasmissione.

Come è possibile notare, sono presenti due macchine a stati finiti concorrenti: la prima riguarda lo stato del modulo Bluetooth, mentre la seconda rappresenta il task dell'invio dei dati e del controllo dell'intervallo di trasmissione, svolto ciclicamente. Esse vengono eseguite in contemporanea e in maniera indipendente l'una

dall'altra, ma hanno la necessità di comunicare tra loro per permettere che i dati letti dal sensore possano poi essere trasmessi.

5.3 Analisi del macro-modulo di ricezione ed interpretazione

Il modulo si occupa della ricezione dei dati dal dispositivo BLE e della loro memorizzazione. Permette, inoltre, di consultare i valori memorizzati e di impostare l'intervallo di trasmissione desiderato. Dovrà essere implementato in modo da funzionare sui dispositivi Android compatibili con la tecnologia Bluetooth Low Energy.[1]

5.3.1 Casi d'uso

In figura 5.3 è possibile osservare il diagramma dei casi d'uso di questa parte di progetto. Gli attori rappresentano i due macro-moduli e l'utente utilizzatore del software, in particolare:

- Utilizzatore del software (interagisce con il dispositivo Android): *User*
- Macro-modulo di ricezione ed interpretazione: *Receiver and interpreter module*
- Macro-modulo di lettura e trasmissione: *Reader and transmitter module*

[1]Come già detto nel capitolo 3, i dispositivi Android compatibili con la tecnologia di comunicazione utilizzata in questo progetto sono quelli su cui gira un sistema operativo con API di livello maggiore o uguale a 18 (Android 4.3 Jelly Bean).

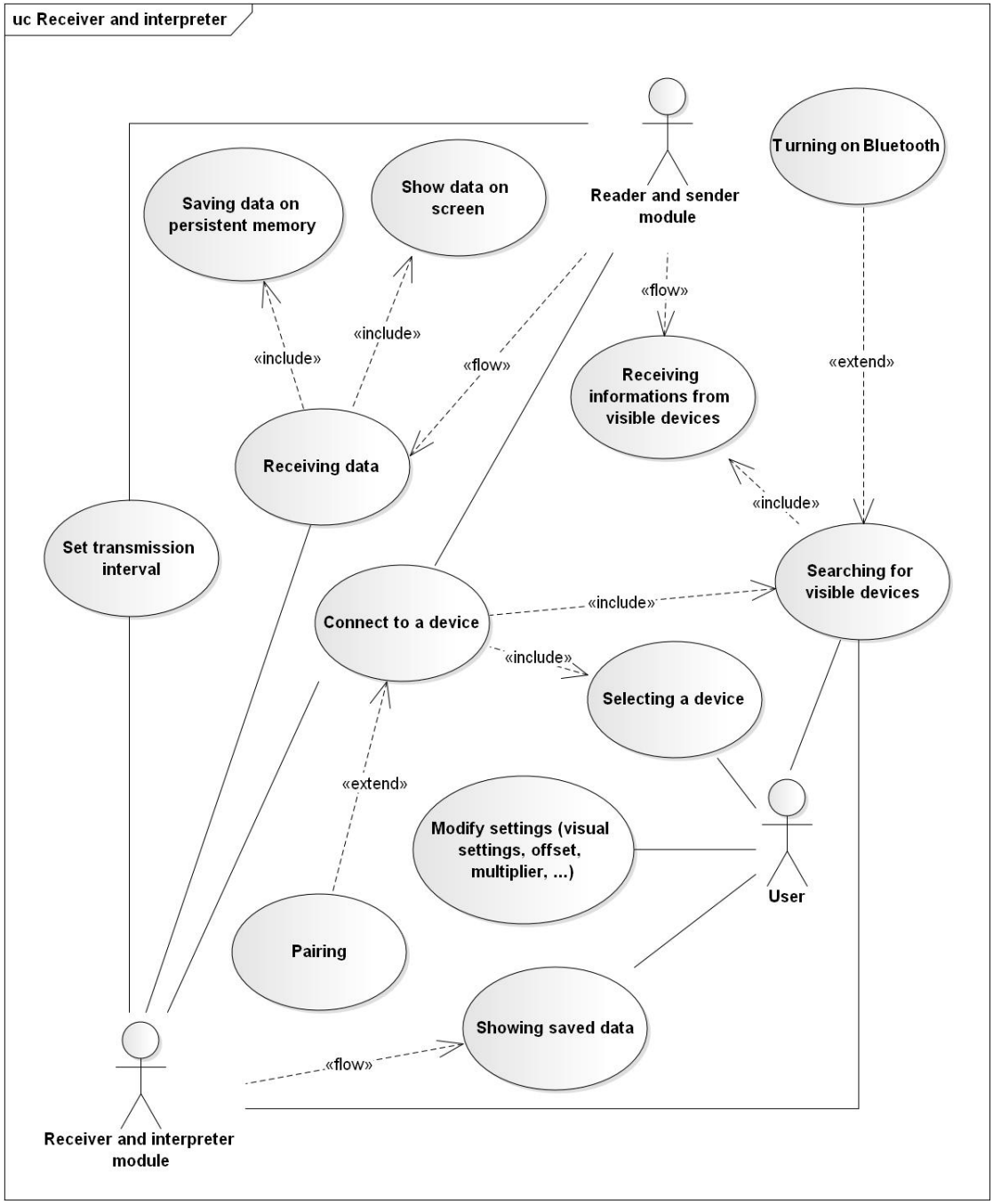

Figura 5.3: Diagramma dei casi d'uso del macro-modulo di ricezione ed interpretazione.

Di seguito una breve descrizione dei singoli casi d'uso:

Searching for visible devices* e *Turning on Bluetooth

L'utente è in grado di effettuare una scansione per ricercare dispositivi visibili all'interno del range di rilevazione. In caso il Bluetooth sia disattivato è necessario attivarlo prima di poter effettuare la ricerca.

Receiving informations from visible devices* e *Selecting a device

Qualora vengano trovati dispositivi, devono essere lette, interpretate e visualizzate su schermo le informazioni contenute all'interno dei pacchetti di *advertising*. La rappresentazione sul display dei dati letti permette di selezionare il dispositivo a cui si desidera connettersi.

Connect to a device* e *Pairing

Il device Android e il dispositivo selezionato dall'utente devono stabilire una connessione. Per rendere possibile lo scambio dei dati è necessario, se non già fatto in passato, effettuare l'accoppiamento (*pairing*).

Receiving data*, *Show data on screen*, *Saving data on persistent memory* e *Set transmission interval

Una volta stabilita la connessione il device Android inizierà a ricevere dati. I valori così ricevuti dovranno essere visualizzati su schermo e salvati all'interno di un'area di memoria persistente. È inoltre possibile impostare l'intervallo di trasmissione del modulo BLE.

Modify settings

L'utente deve essere in grado di modificare alcune impostazioni.

Showing saved data

L'utente può scegliere di visualizzare i dati memorizzati nelle precedenti sessioni di connessione.

5.3.2 Progettazione concettuale delle classi e della base di dati

In questa sezione verranno presentati i diagrammi delle classi dei quattro moduli descritti nella sezione 5.1. Per indicare che una classe viene fornita direttamente dalle API di Android è stato utilizzato lo stereotipo $\ll Android \gg$.

Per motivi di spazio verranno riportati solamente gli attributi, le operazioni e le classi più significative.

Modulo di gestione del Bluetooth

Questo modulo si occupa di gestire la connessione Bluetooth e il relativo scambio di dati. Il diagramma delle classi è riportato in figura 5.4.

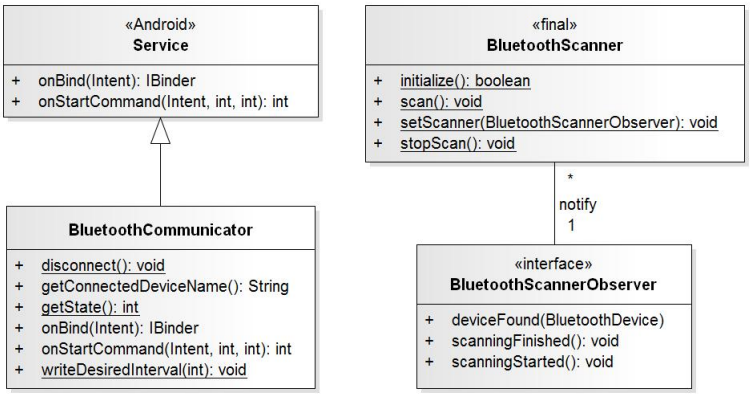

Figura 5.4: Diagramma delle classi del modulo di gestione del Bluetooth.

Modulo di gestione dei dati

Questo modulo si occupa di gestire il salvataggio e l'ottenimento dei dati dal database. Il diagramma delle classi è riportato in figura 5.5.

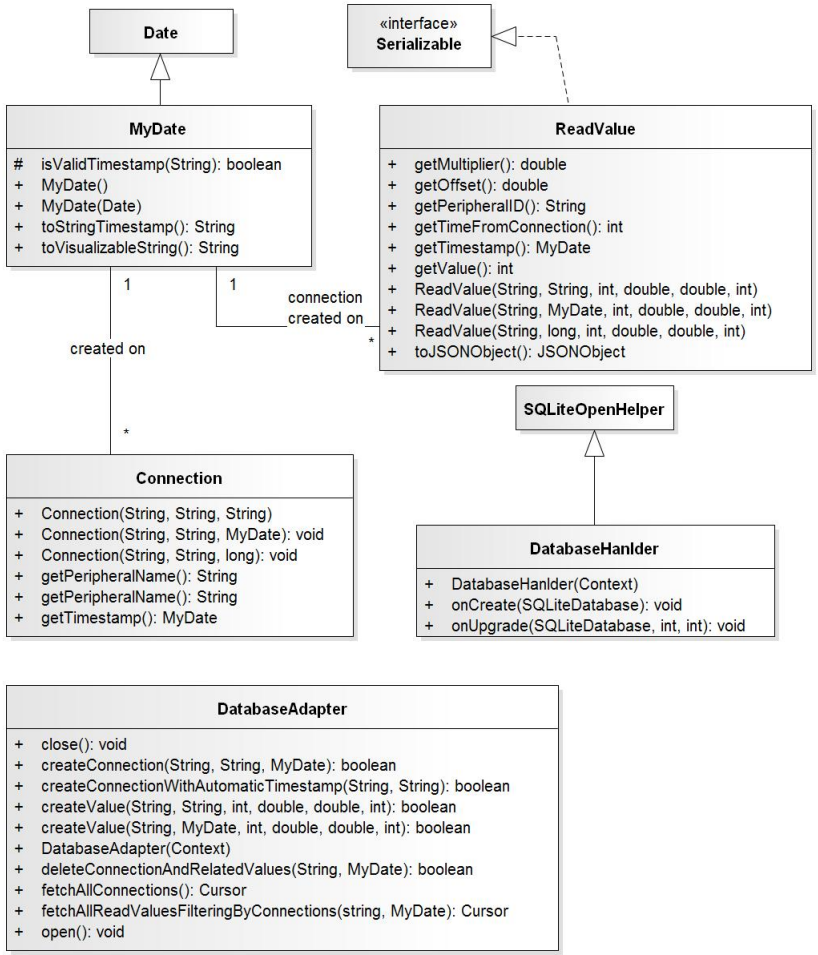

Figura 5.5: Diagramma delle classi del modulo di gestione dei dati.

La memorizzazione dei dati è affidata a un Database *SQLite*. Esso permette, infatti, di memorizzare informazioni in modo strutturato così che possano essere ottenute e aggregate più velocemente rispetto ad altre strutture dati (per esempio un file testuale con rappresentazione *JSON* o *CSV* o un file binario in cui possono essere serializzati gli oggetti Java rappresentanti i dati raccolti). Inoltre *SQLite* risulta sufficientemente leggero per funzionare senza problemi su un dispositivo mobile e Android fornisce librerie per interfacciarsi con esso.

In figura 5.6 si può vedere la rappresentazione sotto forma di schema *Entity Relationship* (*ER*) del database che verrà utilizzato per la memorizzazione dei dati.

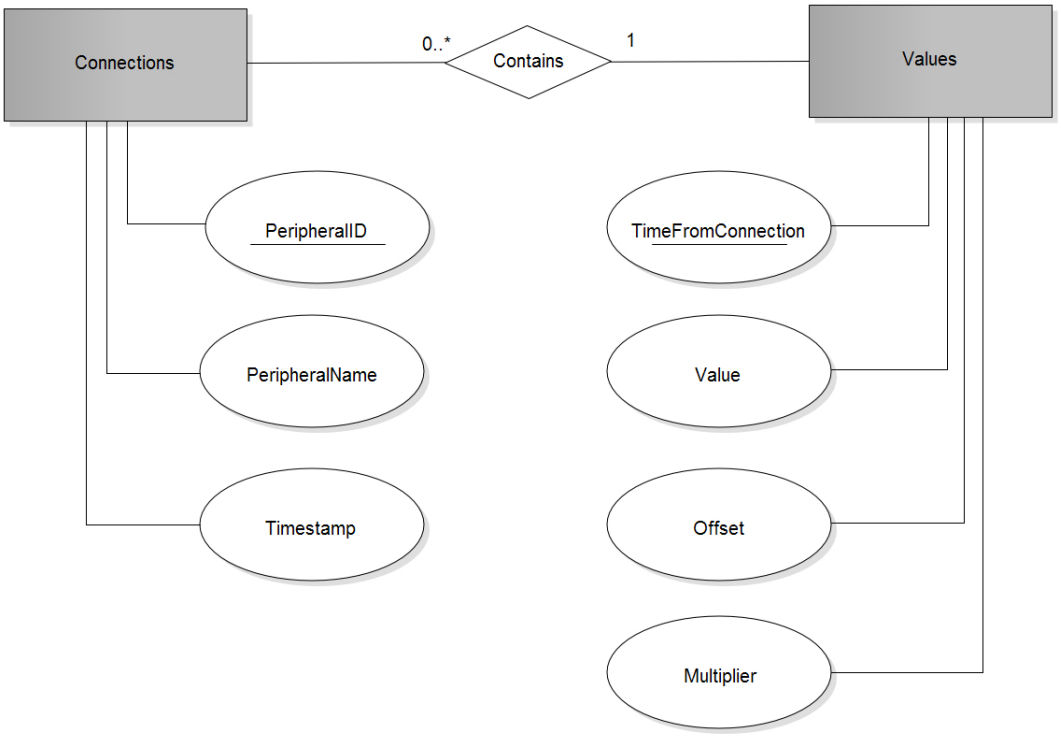

Figura 5.6: Diagramma di sequenza della procedura di lettura e scambio dati.

Modulo di connessione e visualizzazione in tempo reale

Questo modulo si appoggia a quello di gestione di Bluetooth per effettuare la connessione e rileva i dati provenienti da esso mostrandoli su schermo. Contiene quindi tre *view* (nell'ambiente di Android chiamate *Activity*) che permettono di:

- Eseguire la scansione e selezionare il dispositivo a cui connettersi.
- Visualizzare i dati in forma numerica e in forma grafica.
- Visualizzare e modificare le impostazioni.[2]

Il diagramma delle classi è riportato in figura 5.7.

[2]Questa *view* non è rappresentata nel diagramma delle classi poiché viene generata in maniera semi-automatica dalle API di Android.

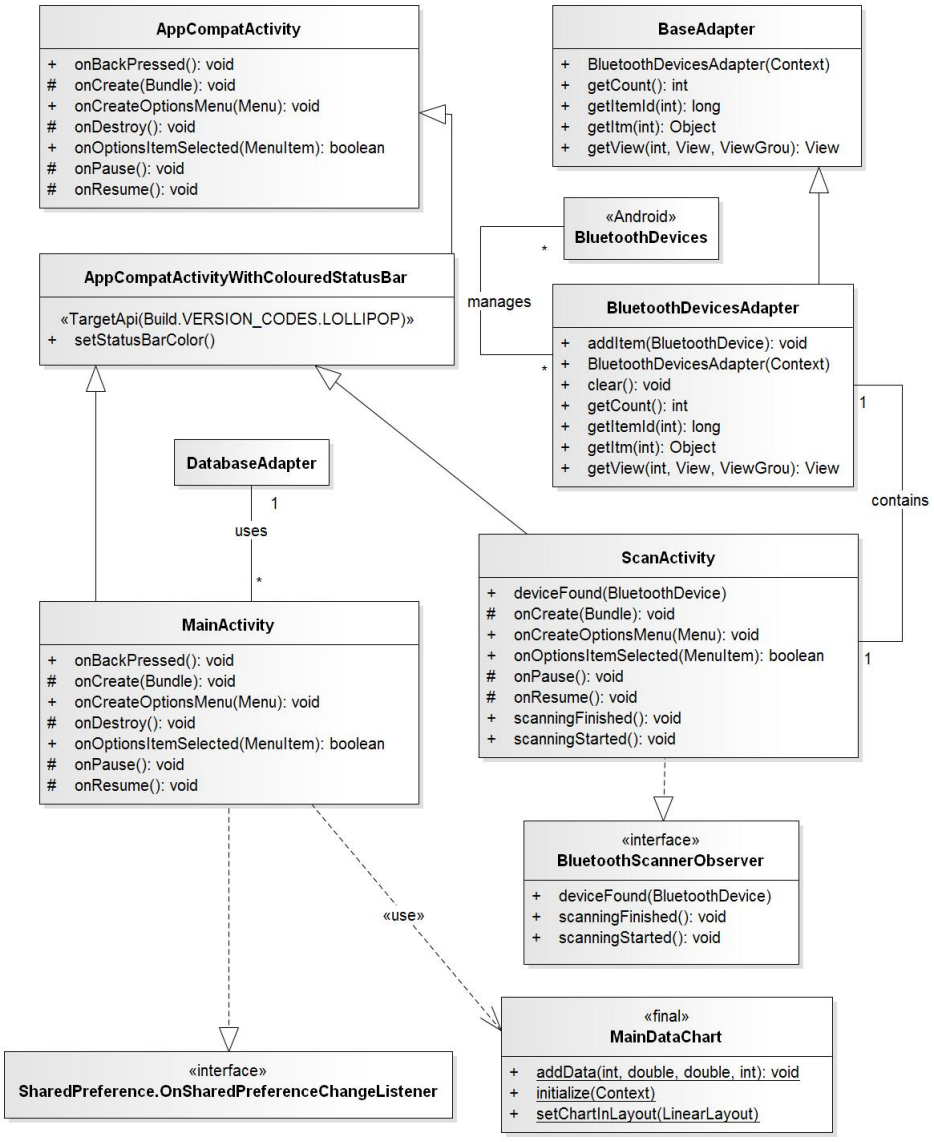

Figura 5.7: Diagramma delle classi del modulo di connessione e visualizzazione in tempo reale.

Modulo di consultazione dei dati memorizzati

Questo modulo si occupa di ottenere i dati memorizzati e mostrarli a schermo sfruttando le tre *query* messe a disposizione dal modulo di gestione dei dati: ottenimento dell'elenco delle connessioni, ottenimento dei valori letti durante una sessione e cancellazione di tutti i dati relativi ad una connessione.

Il modulo comprende due *Activity* diverse:

- Visualizzazione dell'elenco delle sessioni: in questa *view* è possibile visualizzare tutte le sessioni memorizzate e selezionarle per visualizzare i valori in essa contenuti o cancellarle.
- Visualizzazione dei dati contenuti all'interno della sessione selezionata nella *view* precedente. Questa *Activity* è divisa in due diverse *tab*:[3]
  - Visualizzazione grafica: permette di visualizzare i dati in un grafico analogo a quello presente nella *view* principale.
  - Visualizzazione tabulare: permette di visualizzare i dati in una tabella formata dalle stesse colonne della tabella *ReadValues* del database (descritta nella figura 5.6). L'utente può selezionare quali colonne mostrare e quali nascondere.

Il diagramma delle classi è riportato in figura 5.8.[4]

[3]Le *view* in Android possono essere organizzate in *Tab*, ovvero un'*Activity* può contenere all'interno più finestre che possono essere visualizzate alternativamente selezionandole tramite un apposito selettore.

[4]Per ragioni di spazio sono state omesse alcune classi che si occupano della gestione del layout grafico.

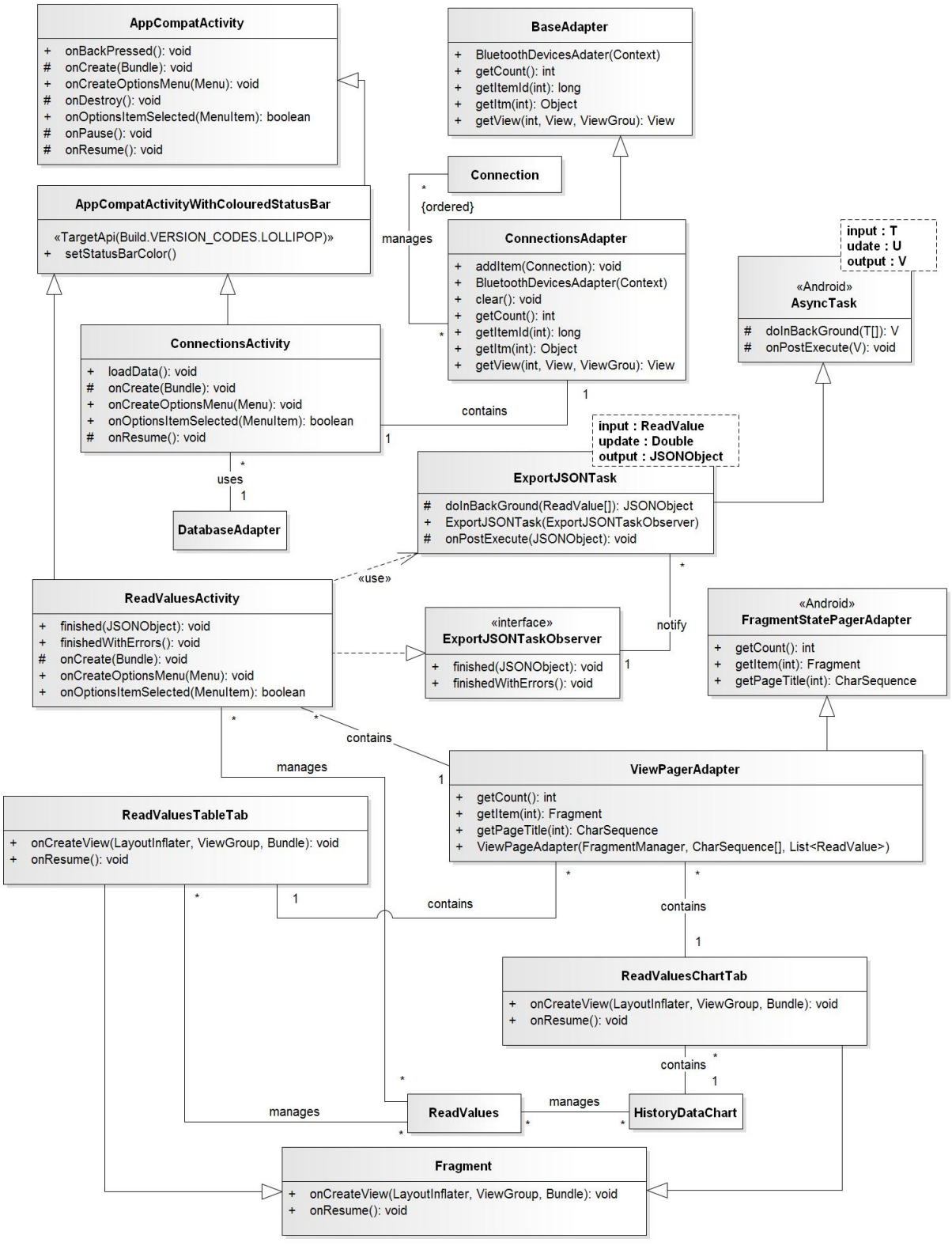

Figura 5.8: Diagramma delle classi del modulo di connessione e consultazione dei dati memorizzati.

5.4 Sessione tipica di utilizzo

Una sessione tipica di utilizzo si svolge in questo modo:

- Il modulo di lettura e trasmissione viene acceso e impostato in modalità *advertising*.
- L'applicazione su Android viene lanciata e, dopo aver attivato il Bluetooth, è avviata la scansione.
- La scansione rileva alcuni dispositivi in range e l'utente sceglie quello corrispondente all'altro modulo dell'applicazione.
- Viene stabilita la connessione e, in caso non fosse già stato fatto in una precedente sezione, viene effettuato l'accoppiamento dei dispositivi.
- Il dispositivo BLE inizia la trasmissione dei dati. L'utente visualizza i dati su schermo ed eventualmente può decidere di modificare l'intervallo di trasmissione.
- Viene effettuata la disconnessione.
- L'utente visualizza i dati registrati durante tale sessione.

5.5 Diagrammi di sequenza

In questa sezione si possono osservare i diagrammi di sequenza di alcune delle operazioni più importanti svolte dal sistema: **Lettura e scambio di dati** e **Connessione**.

5.5.1 Lettura e scambio di dati

Il diagramma in figura 5.9 rappresenta la sequenza di lettura e scambio di dati.

Tale operazione può essere effettuata solo dopo lo stabilimento di una connessione. Il task BLE segnalerà al task ADC di effettuare la lettura. Una volta

completata tale operazione il task ADC porrà il valore letto all'interno della memoria condivisa e il task BLE potrà, in qualunque momento, leggerlo e inviarlo al modulo di ricezione ed interpretazione.

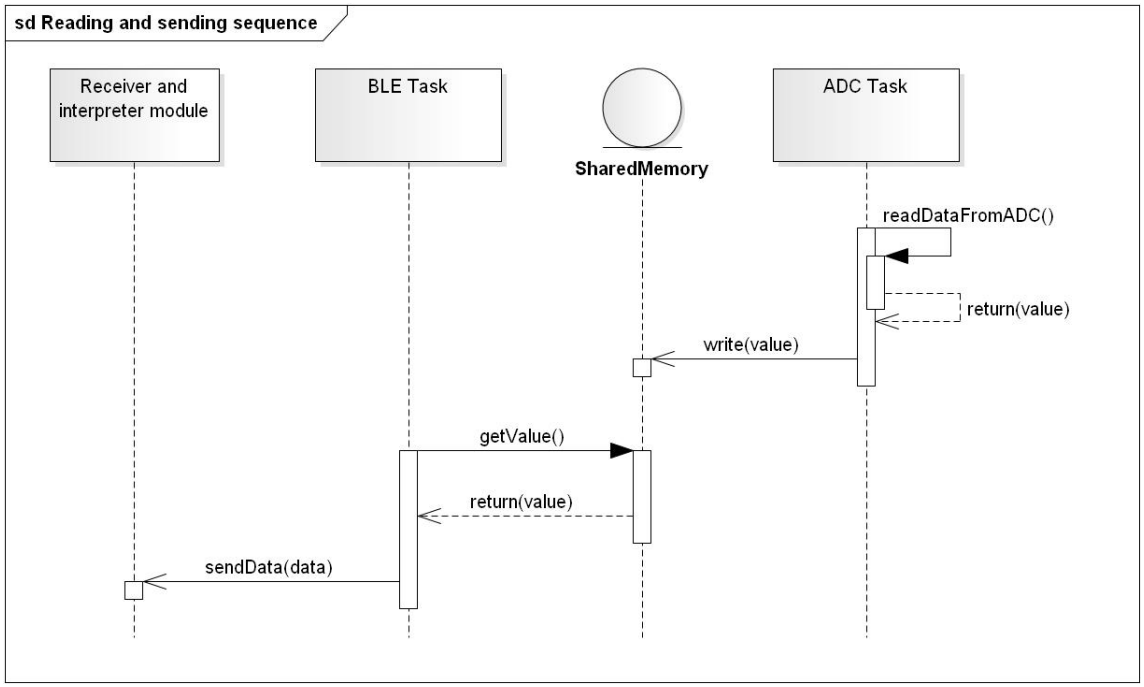

Figura 5.9: Diagramma di sequenza della procedura di lettura e scambio dati.

5.5.2 Connessione

Il diagramma in figura 5.10 rappresenta la sequenza di connessione.

Tale operazione viene avviata nel momento in cui l'utente decide di effettuare la scansione dei dispositivi. Il comando viene impartito tramite interfaccia grafica e viene passato al gestore del Bluetooth. Esso si occuperà di effettuare la ricerca e di restituire i riferimenti ai dispositivi trovati all'interfaccia grafica, che a sua volta la mostrerà all'utente. L'utilizzatore del sistema può, a questo punto, scegliere uno dei device rilevati ed effettuare la connessione. Qualora non sia già stato effettuato il pairing verrà avviata la procedura, al termine della quale i dispositivi risulteranno connessi.

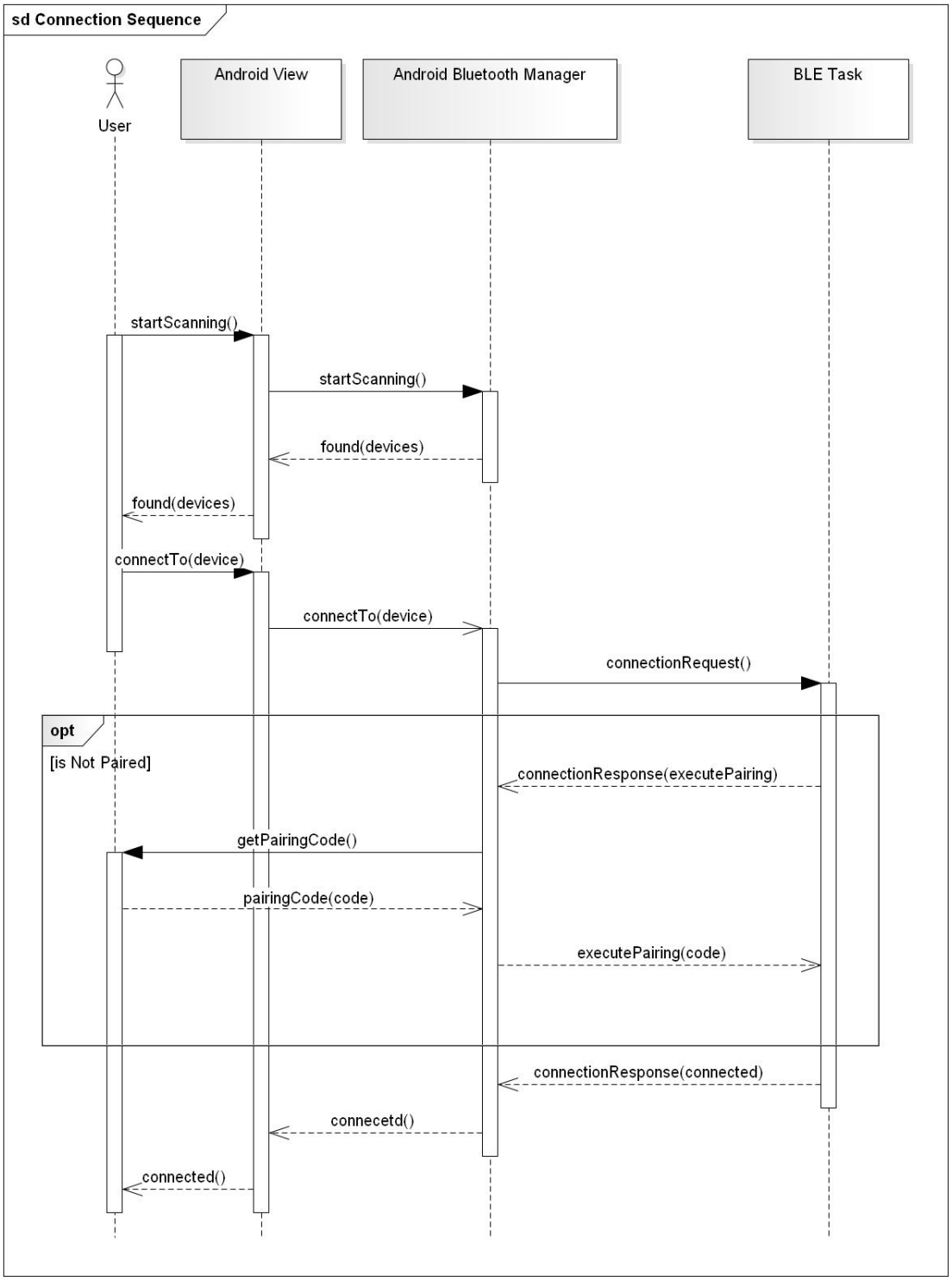

Figura 5.10: Diagramma di sequenza della procedura di connessione.

5.6 Codifica dei dati

5.6.1 Dati trasmessi tramite Bluetooth

Per mantenere la codifica il più semplice, leggera e universale possibile si è scelto di codificare il dato trasmesso come un numero intero a 32 bit. In questo modo la larghezza di banda del Bluetooth è sufficiente per inviare dati con una frequenza elevata (caratteristica utile qualora si decida di espandere il sistema aggiungendo ulteriori sensori) e allo stesso tempo risulta utilizzabile da praticamente qualsiasi dispositivo compatibile con lo standard Bluetooth Low Energy.

5.6.2 Dati esportati dall'applicazione Android

L'esportazione dei dati dall'applicazione Android dovrebbe essere effettuata in modo che sia il più possibile interoperabile con altre applicazioni. Uno dei formati più adatti all'esportazione di dati poco complessi è JSON. Esistono, infatti, librerie per l'interpretazione e la creazione di JSON per quasi tutti i sistemi operativi e i linguaggi esistenti.

Un altro tipo di codifica utilizzabile in contesti simili a questi è CSV. Essa ha tuttavia lo svantaggio di non contenere nomi o riferimenti al tipo di dato contenuto e, di conseguenza, risulta di più difficile interpretazione.

Capitolo 6

Progettazione

In questo capitolo verrà descritta in maniera più dettagliata l'architettura dei due moduli e le scelte che hanno portato alla loro realizzazione.

Come per il precedente capitolo, verranno trattati in maniera separata i due macro-moduli in quanto largamente indipendenti l'uno dall'altro.

6.1 Progettazione del macro-modulo di lettura e trasmissione

6.1.1 Scelta del profilo ricoperto dal dispositivo

Come descritto nella sezione 1.2, ogni dispositivo Bluetooth Low Energy ricopre un profilo che determina le azioni che esso può compiere. In particolare la scelta va effettuata tra *Peripheral* e *Central*. Poiché il sensore deve funzionare come *slave*, ovvero deve accettare connessioni in ingresso ma non ha necessità di iniziare connessioni, il profilo più adeguato per questa applicazione è *Peripheral*. In questo modo non è in grado di richiedere una connessione ma solo di accettare richieste provenienti da altri dispositivi. Una volta effettuato il *binding* può tuttavia inviare e ricevere dati in qualunque momento.

6.1.2 Configurazione delle caratteristiche utilizzate per l'invio e la ricezione dei dati

I dispositivi Bluetooth Low Energy mettono a disposizione delle caratteristiche che incapsulano un puntatore ad uno o più dati e ne permettono quindi la lettura e la scrittura (capitolo 1, sezione 2). Ad ogni caratteristica possono essere assegnati diversi permessi per consentire o meno alcune operazioni.

In questa applicazione le caratteristiche usate sono due: una si occupa di mettere a disposizione i dati letti dal sensore mentre l'altra di ricevere il valore dell'intervallo di trasmissione. Per la prima è stato scelto di utilizzare un approccio a notifiche (il modulo BLE invia una notifica al dispositivo connesso contenente i dati; non è in questo modo necessario, per l'altro dispositivo, effettuare letture *on demand*) mentre per la seconda sono stati forniti permessi in scrittura.

Un'alternativa possibile per la caratteristica di invio dati era quella di inviare i dati in maniera *broadcast*. Tale approccio è stato tuttavia scartato perché, sebbene più semplice, risulta meno versatile. Non permette, infatti, di rendere la trasmissione privata e risulta, inoltre, più dispendioso dal punto di vista energetico: con un sistema *connection-oriented* è possibile abilitare l'*advertising* su richiesta ed effettuare l'invio dei dati solo fintanto che la connessione viene mantenuta, mentre in un sistema *connectionless* è invece necessario inviare dati continuamente per permetterne la lettura in ogni momento.

6.1.3 Organizzazione a task e comunicazione tra essi

Uno dei punti di forza dell'interfaccia richiesta dall'azienda è la sua universalità. Per questo motivo l'approccio utilizzato per la gestione delle operazioni di trasmissione e lettura è stato quello di creare due task separati. In questo modo il sistema risulta scalabile ed espandibile: volendo aggiungere ulteriori sensori, è sufficiente creare un ulteriore task che si occupa della lettura del secondo dato e un'ulteriore caratteristica BLE che si occupa della sua trasmissione.

Questo approccio consente, inoltre, di rendere il sistema più reattivo a stimoli esterni. Eseguendo tutte le operazioni all'interno dello stesso task, infatti, l'ope-

razione di lettura dall'ADC avrebbe interrotto momentaneamente le operazioni di invio e ricezione dati, rischiando in questo modo di perdere dei pacchetti in arrivo. Per motivi fisici del processore (single core) non è possibile eseguire contemporaneamente i due task ma, utilizzando questo sistema, si riesce a migliorarne le prestazioni.

I due task in esecuzione sul modulo hanno necessità di comunicare tra loro (il task di lettura deve inviare al task di trasmissione il dato campionato dall'ADC). Il metodo utilizzato per permettere lo scambio di dati è la creazione di un'area di memoria condivisa, similmente al problema *Readers and writers*.[1]

6.1.4 Lettura dall'ADC

I dati da inviare devono essere letti dall'ADC a 12 bit integrato nel CC2650. L'approccio utilizzato è quello di leggere spesso il valore (con un tempo più breve di quello del periodo minimo di trasmissione) per fare in modo di avere il dato sempre il più aggiornato possibile.

6.1.5 Consumo energetico ridotto

Per mantenere il consumo energetico ridotto è necessario prevedere alcuni accorgimenti:

- Disattivazione della modalità di *advertising* se, entro un certo tempo, nessun dispositivo si è connesso.
- Disattivazione della lettura dall'ADC negli stati di disconnessione e *advertising*.
- Impostazione di adeguati parametri del GATT (intervallo di *advertising*, intervallo di connessione e *slave latency*, descritti nella sezione 1.2.5).

[1] *Readers and writers* è un problema comune nella programmazione concorrente. Si crea quando esistono uno o più "produttori" che scrivono su un buffer e allo stesso tempo uno o più "lettori" devono leggere lo stesso dato. La concorrenza sulla variabile comune può essere risolta con l'utilizzo di un semplice semaforo *mutex*.

6.1.6 Interfacciamento con l'utente

L'utente deve poter osservare lo stato del dispositivo leggendo i dati scritti sul display (se presente) oppure tramite un apposito LED di segnalazione.

6.2 Progettazione del macro-modulo di ricezione ed interpretazione

6.2.1 Modulo di gestione del Bluetooth

Il modulo si occupa di eseguire le operazioni legate al Bluetooth (scansione, connessione e ricezione/trasmissione di dati) e deve operare senza minare la reattività e la fluidità dell'interfaccia grafica. Le operazioni più complesse devono quindi essere svolte su un thread separato.

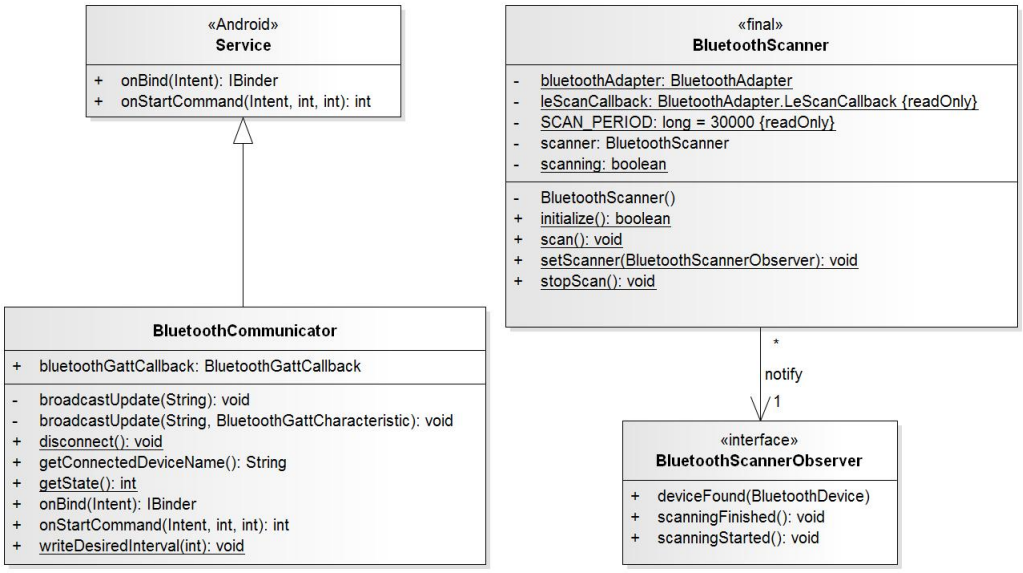

Figura 6.1: Progettazione del modulo di gestione del Bluetooth.

Di seguito una descrizione delle classi principali:

- **BluetoothCommunicator:** è la classe chiave di questo modulo. Si occupa di effettuare e mantenere la connessione, della ricezione delle notifiche con i dati letti e dell'invio del valore relativo all'intervallo di trasmissione.

- **BluetoothScanner:** si occupa di eseguire la scansione alla ricerca di dispositivi rilevabili compatibili con la tecnologia BLE. Permette di registrare un *Observer*[2] che verrà notificato al verificarsi di eventi quali il rilevamento di un dispositivo e l'inizio o terminazione della scansione.

In figura 6.1 è possibile osservare il progetto delle associazioni di questo modulo.

6.2.2 Modulo di gestione dei dati

Il modulo si occupa dell'esecuzione delle operazioni sul database. Come già descritto nella sezione 5.3.2 verrà utilizzato un database SQLite.

Di seguito una descrizione delle classi principali:

- **ReadValue** e **Connection:** modellazione a oggetti dei record appartenenti alle rispettive tabelle del database.

- **MyDate:** specializzazione della classe di libreria *Date* in cui è possibile ottenere rappresentazioni della data compatibili con il database o visualizzabili sull'interfaccia grafica.

- **DatabaseHandler:** si occupa della creazione e della connessione al database.

- **DatabaseAdapter:** classe che funge da ponte tra il *DatabaseHandler* e gli utilizzatori, mette a disposizione metodi per aprire e chiudere la connessione e per effettuare le *query* necessarie.

In figura 6.2 è possibile osservare il progetto delle associazioni di questo modulo.

[2]Il pattern *observer* è un design pattern utilizzato per tenere sotto controllo lo stato di diversi oggetti senza necessità di dover fare operazioni di *polling*. Basa il suo funzionamento sul meccanismo del *callback*, ovvero funzioni che vengono richiamate al verificarsi di certi eventi.

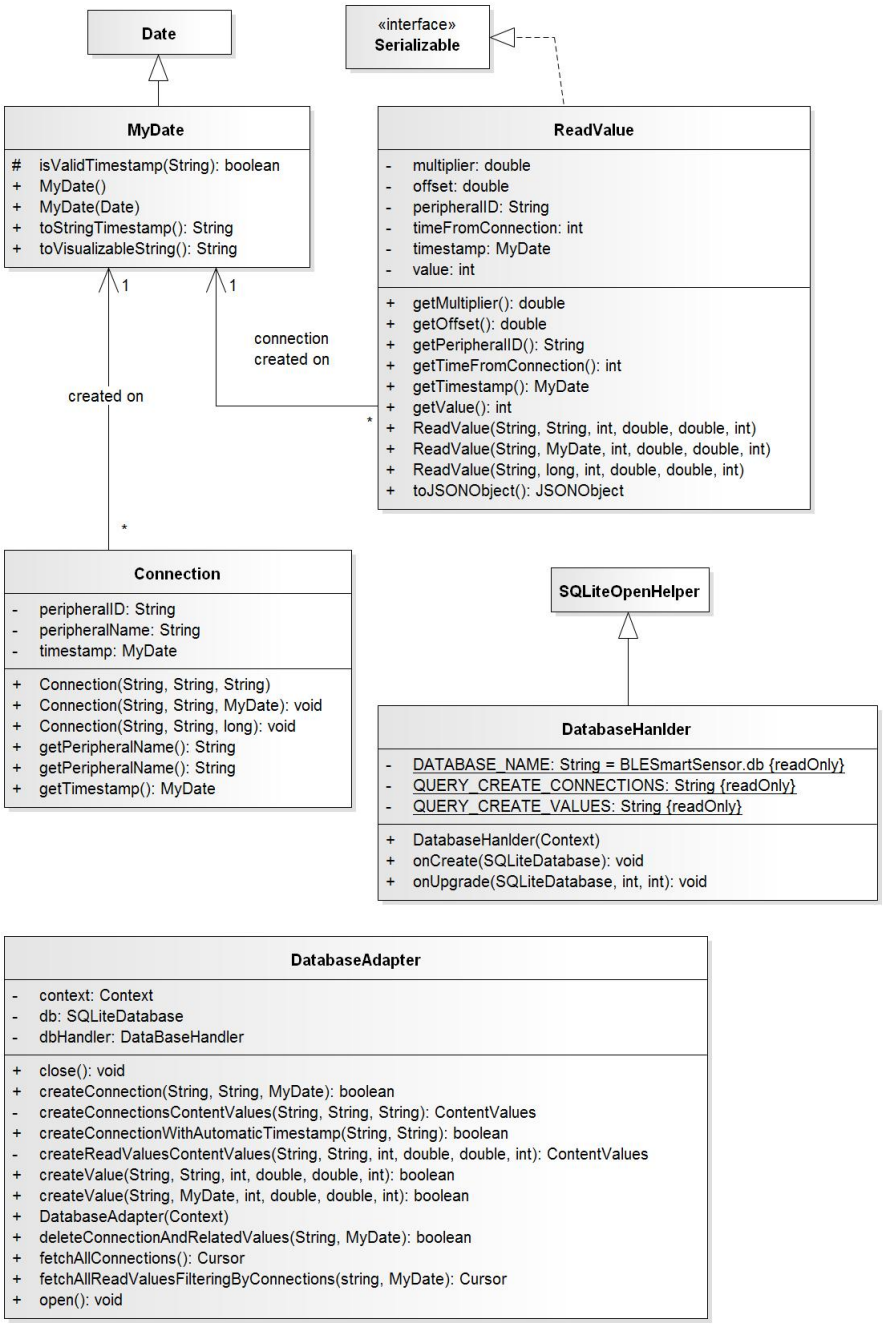

Figura 6.2: Progettazione del modulo di gestione dei dati.

6.2.3 Modulo di connessione e visualizzazione in tempo reale

Il modulo mette a disposizione dell'utente le funzioni offerte dalle classi per la gestione del Bluetooth. Come descritto nel paragrafo 5.3.2 esso è formato da tre *view*.

Di seguito una descrizione delle classi principali:

- **MainActivity:** controller della *view* principale, permette di visualizzare su schermo i dati ricevuti nelle due forme richieste dai requisiti (grafica, tramite *MainDataChart* e testuale). Questa *view* implementa *SharedPreference.OnSharedPreferenceChangeListener*. Ciò le permette di ottenere delle notifiche qualora i valori delle impostazioni cambino.
- **ScanActivity:** controller della *view* che si occupa della scansione. Visualizza la lista di dispositivi rilevati utilizzando l'apposito *BluetoothDeviceAdapter*. Questa *view* implementa *BluetoothScannerObserver* per ottenere notifiche riguardo gli eventi generati dalla parte di modulo di gestione del Bluetooth che si occupa della scansione.

In figura 6.3 è possibile osservare il progetto delle associazioni di questo modulo.

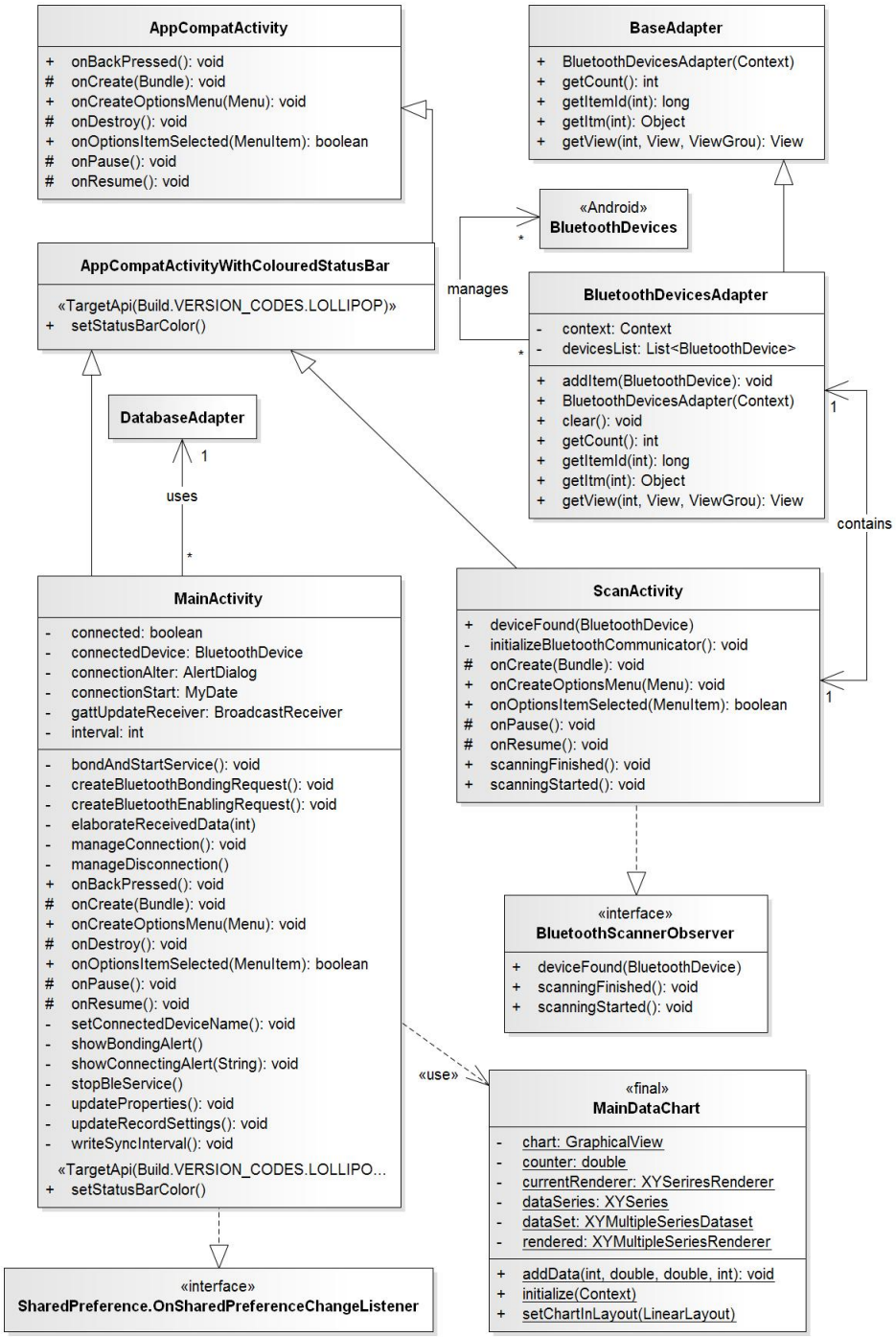

Figura 6.3: Progettazione del modulo di connessione e visualizzazione in tempo reale.

6.2.4 Modulo di consultazione dei dati memorizzati

Il modulo utilizza le classi per la gestione dei dati per visualizzare in apposite *view* i valori salvati in memoria. Come descritto nel paragrafo 5.3.2 è formato da due *view*.

Di seguito una descrizione delle classi principali:

- **ConnectionsActiviy:** controller della *view* che permette di mostrare l'elenco delle sessioni memorizzate. Utilizza un apposito *ConnectionAdapter* per visualizzare i dati su schermo e si serve del *DatabaseAdapter* per ricavare i dati dalla memoria.

- **ReadValuesActivity:** controller della *view* che mostra i dati raccolti all'interno di una sessione. Si serve di *ViewPagerAdapter* per mostrare alternativamente una delle due schede che contengono la visualizzazione grafica (*ReadValuesTableTab*) o tabulare (*ReadValuesChartTab*) dei dati ottenuti dal database. Implementa inoltre *ExportJSONTaskObserver* per ricevere aggiornamenti sullo stato dell'esportazione.

- **ExportJSONTask:** task realizzato estendendo la classe di libreria di Android *AsyncTask*. Essa mette a disposizione un oggetto in grado di svolgere operazioni su un thread separato. In questo caso si occupa della conversione in *JSON* dei dati mostrati su schermo.

In figura 6.4 è possibile osservare il progetto delle associazioni di questo modulo.

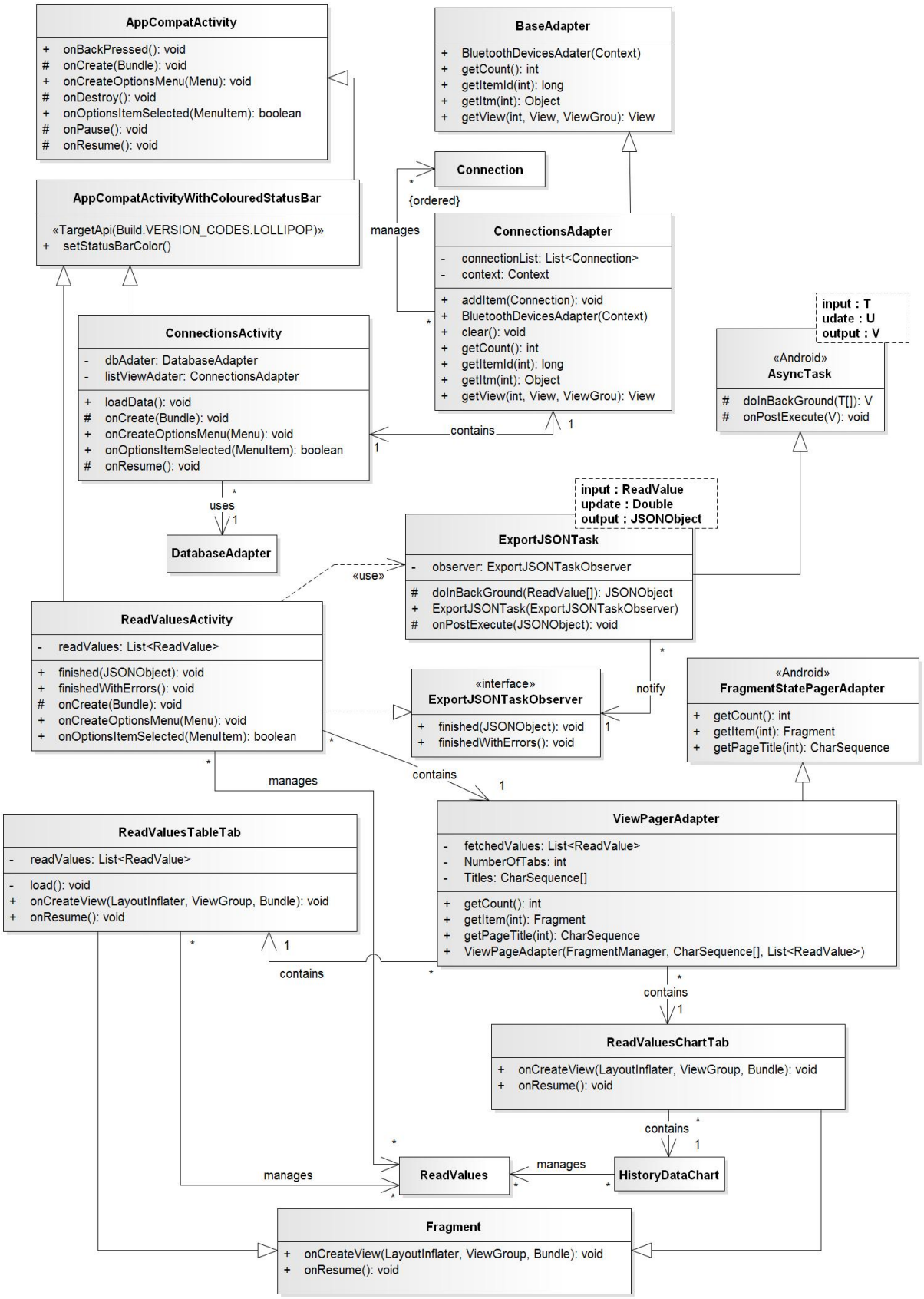

Figura 6.4: Progettazione del modulo di connessione e consultazione dei dati memorizzati.

6.2.5 Progettazione dell'interfaccia grafica

Buona parte del modulo di ricezione ed interpretazione è formato dalle *view* che mostrano su schermo i dati appena ricevuti o quelli memorizzati all'interno del database. È quindi necessario definire come l'interfaccia grafica dovrà apparire. Nella figura 6.5 è possibile osservare un *wireframe*[3] delle *view* dell'applicazione.

La grafica è stata realizzata seguendo il più possibile i dettami del *Material Design*.[4]

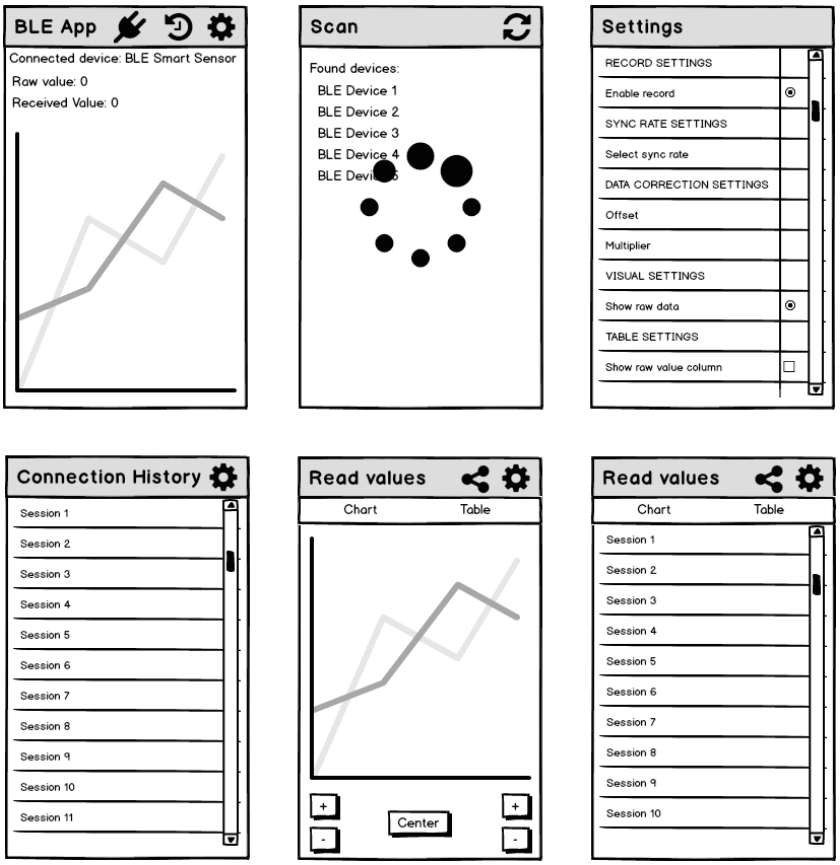

Figura 6.5: In senso orario, partendo dall'alto a sinistra, *Main View*, *Scan*, *Settings*, *Connection History* e le due tab di *Read Values*.

[3]Con *wireframe* si intende un modello strutturale del layout dell'interfaccia grafica senza indicare stili, colori e altri dettagli grafici.

[4]Il *Material Design* è un insieme di regole che Google propone per la realizzazione della grafica di applicazioni e siti web.

Activity *Main View*

Pagina principale dell'applicazione. Si possono notare tre campi di testo con cui è possibile visualizzare rispettivamente il nome del dispositivo connesso (*Connected device)*, l'ultimo valore indicato così come ricevuto dal dispositivo (*Raw value)* e la sua correzione (*Received value*). Nella *Toolbar*[5] sono presenti tre bottoni che permettono di visualizzare la schermata di scansione, visualizzare le sessioni salvate o avviare la *view* delle impostazioni.

Questa *view* può mostrare alcuni messaggi sotto forma di *Alert*:[6]

- Attivazione del Bluetooth
- Guida per l'effettuazione del pairing

Activity *Scan*

Pagina di scansione. I dettagli principali di questa *Activity* sono il pulsante per ri-avviare la scansione (in alto a destra), la lista di dispositivi rilevati (cliccabili per selezionarli) e l'indicatore della scansione in corso (che scomparirà al termine della scansione). Questa *view* può mostrare alcuni messaggi sotto forma di *Alert*:

- Attivazione del Bluetooth
- Connessione in corso

Activity *Settings*

Pagina delle impostazioni. Questa *Activity* permette di gestire le impostazioni, tra cui la scelta di registrare la sessione, la visualizzazione del valore non corretto e l'impostazione dei parametri di offset e moltiplicatore.

[5] In un'*Activity* Android la *Toolbar* è la barra posizionata nella parte alta dello schermo in cui sono presenti il titolo della *view* ed, eventualmente, alcuni bottoni.

[6] Un *Alert* è una finestra modale che compare al centro della schermata. Può contenere diversi elementi grafici, come un campo di testo o dei bottoni.

Activity *Connection History*

Pagina per la visualizzazione dell'elenco delle sessioni salvate. Si può notare la lista nella parte centrale e il pulsante per richiamare le impostazioni in alto a destra, nella *Toolbar*.

Activity *Read Values*

Pagina per la visualizzazione dei dati salvati nel contesto di una sessione. Questa schermata è divisa in due *tab*: la visualizzazione grafica (*Chart*) e la visualizzazione tabulare (*Table*). Si può accedere all'una o all'altra visualizzazione utilizzando il selettore posto immediatamente sotto la *Toolbar*. Altri controlli presenti all'interno della schermata sono i pulsanti per la condivisione e le impostazioni (in alto a destra, nella *Toolbar*) e i pulsanti per il controllo dello zoom nel grafico (nella parte bassa della tab per la visualizzazione grafica).

Altre view

Esistono alcune *view* che vengono utilizzate in questa applicazione ma che non sono state create all'interno di questo progetto. Esse fanno parte di applicazioni preinstallate all'interno del sistema operativo Android e possono essere richiamate per svolgere alcune operazioni. In particolare esse sono:

- **View per effettuare il pairing**
- **View per effettuare la condivisione**

Capitolo 7

Implementazione

In questo capitolo verranno descritte alcune parti dell'implementazione del progetto, riportando anche porzioni di codice esemplificative. In particolare verranno riportate le scelte implementative fatte per soddisfare i vincoli progettuali descritti nel capitolo 6.

7.1 Implementazione del macro-modulo di lettura e trasmissione

7.1.1 Configurazione delle caratteristiche

Come già descritto parzialmente nella sezione 6.1.2, è stato necessario utilizzare due caratteristiche: una per l'invio dei dati e una per la ricezione del valore di intervallo prescelto. I permessi forniti alle caratteristiche sono, rispettivamente, in lettura (con notifica) e in scrittura. Nel blocco di codice 7.1 è presente la dichiarazione della caratteristica per la lettura dell'intervallo di trasmissione.

```
#define INTERVAL_RECEIVE_CHARACTERISTIC          0
#define SERVICE_UUID                             0xFFF0
#define INTERVAL_RECEIVE_CHARACTERISTIC_UUID     0xFFF1
[...]
CONST uint8 simpleProfileServUUID[ATT_BT_UUID_SIZE] = {
    LO_UINT16(SERVICE_UUID),
      HI_UINT16(SERVICE_UUID) };
[...]
CONST uint8 simpleProfilechar1UUID[ATT_BT_UUID_SIZE] = { LO_UINT16(
      INTERVAL_RECEIVE_CHARACTERISTIC_UUID), HI_UINT16(
      INTERVAL_RECEIVE_CHARACTERISTIC_UUID) };
static uint8 intervalReceiveCharacteristicProperties = GATT_PROP_READ |
    GATT_PROP_WRITE;
static uint8 intervalReceiveCharacteristicValue = 0;
static uint8 intervalReceiveCharacteristicDescription[9] = "RW Value";
[...]
static gattAttribute_t simpleProfileAttrTbl[SERVAPP_NUM_ATTR_SUPPORTED] =
{
   {
      {ATT_BT_UUID_SIZE, primaryServiceUUID},
      GATT_PERMIT_READ,
      0, (uint8 *)&simpleProfileService
   },
   {
      {ATT_BT_UUID_SIZE, characterUUID},
      GATT_PERMIT_READ,
      0, &intervalReceiveCharacteristicProperties
   },
   {
     {ATT_BT_UUID_SIZE, intervalReceiveCharacteristicUUID},
     GATT_PERMIT_READ | GATT_PERMIT_WRITE,
     0, &intervalReceiveCharacteristicValue
   },
   {
      {ATT_BT_UUID_SIZE, charUserDescUUID},
      GATT_PERMIT_READ,
      0, intervalReceiveCharacteristicDescription
   },
   [...]
};
```

Listing 7.1: Dichiarazione della caratteristica per la lettura dell'intervallo di trasmissione. Tratto da *simpleGATTprofile.h* e *simpleGATTprofile.c*

7.1.2 Organizzazione a task

Per implementare l'approccio a task come descritto nel capitolo 6 (sezione 6.1.3) è stato scelto di utilizzare le omonime strutture messe a disposizione dalle librerie fornite in bundle con il CC2650. Esse mettono a disposizione funzioni per creare flussi di controllo autonomi che possono eseguire contemporaneamente.[1] Ogni task ha a disposizione un proprio stack e viene schedulato, tramite l'utilizzo di appositi semafori, in base alla priorità impostata al momento della sua creazione.

Un esempio di codice che effettua l'inizializzazione dal task che gestisce la lettura del valore rilevato dall'ADC si trova nel listato 7.2. Si può notare l'inizializzazione dei parametri e, più in basso, la costruzione del task stesso.

```
1  #define TASK_STACK_SIZE 512
2  #define TASK_PRI      1
3
4  void ADC_createTask() {
5     Task_Params params;
6     Task_Params_init(&params);
7     params.priority = TASK_PRI;
8     params.stackSize = TASK_STACK_SIZE;
9     params.stack = taskStack;
10
11    Task_construct(&taskStruct, taskFxn, &params, NULL);
12
13    [...]
14 }
```

Listing 7.2: Inizializzazione di un task. Tratto da *ADC.c*

[1] La contemporaneità è, ovviamente, apparente: il singolo processore del CC2650 può infatti eseguire un solo processo alla volta. Il software permette, tuttavia, di portare avanti in parallelo entrambi i flussi di controllo.

7.1.3 Comunicazione tra i task

Per permettere la comunicazione tra i task è stata creata un'apposita parte di codice che contiene una variabile condivisa a cui il task dell'ADC può accedere in scrittura e il task del BLE in lettura. Per la sincronizzazione è stato utilizzato un semaforo binario la cui struttura è dichiarata all'interno delle librerie fornite da Texas Instrument. Nel listato 7.3 è possibile trovare il codice corrispondente.

Con lo stesso principio è stato organizzato il sistema per attivare e disattivare la lettura dall'ADC rispettivamente alla connessione e alla disconnessione della periferica (per ottimizzare il risparmio energetico).

```
1  Semaphore_Struct exchangeDataSem;
2  Semaphore_Handle exchangeDataSemHandle;
3  volatile uint16_t sharedADCSample;
4
5  void init() {
6     Semaphore_Params params;
7     Semaphore_Params_init(&params);
8     params.mode = Semaphore_Mode_BINARY;
9     Semaphore_construct(&exchangeDataSem, 1, &params);
10    exchangeDataSemHandle = Semaphore_handle(&exchangeDataSem);
11 }
12
13 void setData(uint16_t value) {
14    Semaphore_pend(exchangeDataSemHandle, 500);
15    sharedADCSample = value;
16    Semaphore_post(exchangeDataSemHandle);
17 }
18
19 uint16_t getData() {
20    return sharedADCSample;
21 }
```

Listing 7.3: Area di memoria condivisa e relative strutture di sincronizzazione.

7.1.4 Lettura dall'ADC

La lettura dall'ADC è eseguita dall'apposito task. Esso contiene una parte di inizializzazione dei pin di GPIO e una parte relativa alla lettura. La lettura viene effettuata all'interno di un ciclo infinito per essere eseguita ripetutamente durante lo stato di connessione.

```
for(;;)
{
   Task_sleep(100 * 1000 / Clock_tickPeriod);
   AUXADCGenManualTrigger();
   Semaphore_pend(hSem, BIOS_WAIT_FOREVER );
   setData(singleSample);
}
```

Listing 7.4: Lettura di dati dall'ADC.

La lettura dei dati viene avviata dalla funzione *AUXADCGenManualTrigger()*. Al termine dell'operazione viene generato un interrupt gestito da un'apposita *Interrupt service routine* (*ISR*) che funge da *callback*.

```
void adcIsr(UArg a0) {
  singleSample = AUXADCReadFifo();
  HWREGBITW(AUX_EVCTL_BASE + AUX_EVCTL_O_EVTOMCUFLAGSCLR,
      AUX_EVCTL_EVTOMCUFLAGSCLR_ADC_IRQ_BITN) = 1;

  Semaphore_post(hSem);

}
```

Listing 7.5: *Callback* dell'ADC

7.1.5 Consumo energetico ridotto

Le librerie del CC2650 mettono a disposizione vari metodi per ridurre il consumo energetico:

- Impostazione dei valori richiesti[2] di *Connection interval*, *Slave latencty* e *Advertising interval* (nel caso specifico, rispettivamente 100 ms - 800 ms, 0 e 100 ms).
- Impostazione della potenza della trasmissione (nel caso specifico, 0 dBm).
- Attivazione della modalità di risparmio energetico quando il dispositivo è in stato di *idle* (ovvero non sta eseguendo operazioni).
- Disattivazione automatica della modalità di *advertising*.

7.1.6 Interfacciamento con l'utente

Per l'interfacciamento con l'utente è stato scelto, inizialmente, di utilizzare due diversi metodi: un LED e il display, due periferiche integrate sulla basetta di debug del CC2650. Inoltre è stato necessario utilizzare un pulsante per permettere la riattivazione della modalità di *advertising*. Di seguito tratti di codice che realizzano queste caratteristiche. Nella versione successiva per ragioni di ottimizzazione di spazio e di consumo energetico si è scelto di eliminare il display.

```
1 static PIN_Config SBP_configTable[] =
2 {
3  Board_LED1   | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL |
       PIN_DRVSTR_MAX,
4  Board_KEY_UP | PIN_INPUT_EN | PIN_PULLUP | PIN_HYSTERESIS,
5  PIN_TERMINATE
6 };
```

Listing 7.6: Inizializzazione dei PIN di GPIO

[2]Come descritto nel capitolo 1, il dispositivo che ha il ruolo di periferica non è in grado di scegliere tali valori per la connessione, ma può fornire delle indicazioni sugli intervalli che dovrebbero essere utilizzati dal master.

```
LCD_WRITE_STRING("BLESmartSensor", LCD_PAGE0);
```

Listing 7.7: Esempio di scrittura del nome del dispositivo sul display

```
PIN_setOutputValue(hSbpPins, Board_LED1, state);
```

Listing 7.8: Esempio di accensione o spegnimento di un LED

7.2 Implementazione del macro-modulo di ricezione ed interpretazione

7.2.1 Modulo di gestione del Bluetooth

Per realizzare il requisito di reattività (descritto nella sezione 6.2.1) è stata utilizzata la classe *Service* messa a disposizione da Android. Essa consente di creare un nuovo flusso di controllo su cui possono essere eseguite alcune operazioni.

Per gestire la conclusione delle operazioni di lunga durata è stato implementato un apposito sistema a *callback*.

Di seguito alcuni tratti di codice provenienti dalle classi contenute in questo modulo.

```
1  @Override
2  public void onConnectionStateChange(final BluetoothGatt gatt, final int
       status, final int newState) {
3      if(newState == BluetoothProfile.STATE_CONNECTED) {
4          connectionState = STATE_CONNECTED;
5          broadcastUpdate(ACTION_GATT_CONNECTED);
6          bluetoothGatt.discoverServices();
7      } else if(newState == BluetoothProfile.STATE_DISCONNECTED) {
8          String intentAction = ACTION_GATT_DISCONNECTED;
9          gatt.close();
10         connectionState = STATE_DISCONNECTED;
11         broadcastUpdate(intentAction);
12     }
13 }
```

Listing 7.9: *Callback* per gestire la connessione o disconnessione di un device

```
        @Override
        public void onCharacteristicChanged(BluetoothGatt gatt,
            BluetoothGattCharacteristic characteristic) {
            super.onCharacteristicChanged(gatt, characteristic);
            broadcastUpdate(ACTION_DATA_AVAILABLE, characteristic);
        }
    };
```

Listing 7.10: *Callback* per gestire la ricezione di una notifica

7.2.2 Modulo di gestione dei dati

Il cuore di questo modulo è formato dalle *query* che vengono eseguite per la creazione e per la modifica del database e dalle classi che permettono di effettuare la trasposizione in oggetto dei record del database. Nei blocchi di codice 7.11, 7.12, 7.13 e 7.14 si possono vedere le funzioni che permettono di eseguirle.

Per la gestione dei valori sono state usate le classi di libreria di Android *ContentValues* e *Cursor*, che permettono di creare oggetti e array di oggetti senza una specifica struttura definita a priori.

```
private static final String QUERY_CREATE_CONNECTIONS = "create table
    Connections (" +
            "    PeripheralID varchar(12) not null, " +
            "    PeripheralName varchar(50) not null, " +
            "    Timestamp varchar(10) not null, " +
            "    constraint IDConnection primary key (PeripheralID,
                Timestamp));";

private static final String QUERY_CREATE_VALUES = "create table ReadValues (" +
            "    PeripheralID varchar(12) not null, " +
            "    Timestamp varchar(10) not null, " +
            "    Value numeric(10) not null, " +
            "    Offset numeric(10,5) not null, " +
            "    Multiplier numeric(10,5) not null, " +
            "    TimeFromConnection numeric(10) not null, " +
            "    constraint FKContains primary key (PeripheralID, Timestamp,
                TimeFromConnection), " +
            "    foreign key (PeripheralID, Timestamp) references
                Connessione);";
```

Listing 7.11: Query di creazione del database.

```
1 public Cursor fetchAllConnections() {
2       return db.query(TABLE_CONNECTIONS, new
            String[]{CONNECTION_PERIPHERAL_ID, CONNECTION_PERIPHERAL_NAME,
            CONNECTION_TIMESTAMP}, null, null, null, null, null);
3   }
```

Listing 7.12: Query per l'ottenimento della lista di sessioni salvate

```
1 public Cursor fetchAllReadValuesFileringByConnections(String peripheralID,
      MyDate timestamp) {
2       return db.query(TABLE_VALUES, new String[]{VALUES_PERIPHERAL_ID,
            VALUES_TIMESTAMP, VALUES_VALUE, VALUES_OFFSET, VALUES_MULTIPLIER,
            VALUES_TIME_FROM_CONNECTION},
3               VALUES_PERIPHERAL_ID + " LIKE '" + peripheralID + "' AND " +
                    VALUES_TIMESTAMP + " LIKE '" + timestamp.toStringTimestamp()
                    + "'", null, null, null, null, null);
4
5   }
```

Listing 7.13: Query per l'ottenimento della lista di valori appartenenti ad una certa sessione.

```
1 public boolean deleteConnetionAndRelatedValues(String peripheralID, MyDate
      timestamp) {
2       boolean result = false;
3       Cursor c = db.query(TABLE_CONNECTIONS, new
            String[]{CONNECTION_PERIPHERAL_ID, CONNECTION_TIMESTAMP},
            CONNECTION_PERIPHERAL_ID + " LIKE '" + peripheralID + "'", null,
            null, null, null);
4       c.moveToNext();
5       result = db.delete(TABLE_VALUES, VALUES_PERIPHERAL_ID + " LIKE '" +
            peripheralID + "' AND "+ VALUES_TIMESTAMP + " LIKE '" +
            timestamp.toStringTimestamp() + "'", null) > 0;
6       result = result || db.delete(TABLE_CONNECTIONS,
            CONNECTION_PERIPHERAL_ID + " LIKE '" + peripheralID + "' AND "+
            CONNECTION_TIMESTAMP + " LIKE '" + timestamp.toStringTimestamp() +
            "'", null) > 0;
7       return result;
8   }
```

Listing 7.14: Query per la cancellazione dei record correlati ad una certa sessione (quello rappresentante la sessione e quelli rappresentanti i dati salvati).

7.2.3 Modulo di connessione e visualizzazione in tempo reale

Questo modulo si compone di due *Activity* (*view*) e di classi correlate che utilizzano il modulo di gestione del Bluetooth. Nel listato 7.15 si può vedere il dettaglio della registrazione al *Broadcast Receiver* e l'avvio del service corrispondente alla classe *Bluetoothcommunicator*.

```
1  Intent startCommunication = new Intent(this, BluetoothCommunicator.class);
2         startCommunication.putExtra(getString(R.string.device_extra_intent),
              connectedDevice);
3         startService(startCommunication);
4
5         registerReceiver(gattUpdaterReceiver, new
              IntentFilter(BluetoothCommunicator.ACTION_GATT_CONNECTED));
6         registerReceiver(gattUpdaterReceiver, new
              IntentFilter(BluetoothCommunicator.ACTION_GATT_DISCONNECTED));
7         registerReceiver(gattUpdaterReceiver, new
              IntentFilter(BluetoothCommunicator.ACTION_GATT_SERVICE_DISCOVERED));
8         registerReceiver(gattUpdaterReceiver, new
              IntentFilter(BluetoothCommunicator.ACTION_DATA_AVAILABLE));
9
10 private final BroadcastReceiver gattUpdaterReceiver = new BroadcastReceiver() {
11        @Override
12        public void onReceive(Context context, Intent intent) {
13            final String action = intent.getAction();
14            if(BluetoothCommunicator.ACTION_GATT_CONNECTED.equals(action)) {
15                manageConnection();
16
17            } else
                  if(BluetoothCommunicator.ACTION_GATT_DISCONNECTED.equals(action))
                  {
18                manageDisconnection();
19            } else
                  if(BluetoothCommunicator.ACTION_DATA_AVAILABLE.equals(action)) {
20                int value = (int)
                      intent.getExtras().get(BluetoothCommunicator.EXTRA_DATA);
21                elaborateReceivedData(value);
22            }
23        }
24    };
```

Listing 7.15: Avvio del service e registrazione al *Broadcast Receiver*.

La parte più caratteristica è, probabilmente, la visualizzazione dei dati su grafico. La libreria utilizzata per permettere tale vista è *AChartEngine*.[3] Essa permette di creare grafici di diverso tipo, tra cui il piano cartesiano richiesto nelle specifiche, creando un oggetto di tipo *View* che può essere sostituito ad un *LinearLayout* presente all'interno dell'*Activity* (dimostrazione nel blocco di codice 7.16).

```
1 if(chart == null) {
2           MainDataChart.initialize(layout.getContext());
3           chart = ChartFactory.getLineChartView(layout.getContext(), dataSet,
                renderer);
4           layout.addView(chart);
5       } else {
6           chart.repaint();
7       }
```

Listing 7.16: Sostituzione dell'oggetto *grafico* all'interno del *LinearLayout*

7.2.4 Modulo di consultazione dei dati memorizzati

Come il precedente, anche questo modulo concerne prevalentemente aspetti grafici. Esso si basa sui servizi offerti dal *DatabaseAdapter* (modulo di gestione dei dati) per ottenere i dati memorizzati e mostrarli a schermo. Anche in questo modulo è stato utilizzato un grafico creato con la libreria *AChartEngine*. La peculiarità dell'*Activity* di visualizzazione dati è l'organizzazione in forma di tab. Il modulo contiene, inoltre, funzioni che permettono di esportare i dati in formato JSON. Nel listato 7.17 si può vedere il codice dell'*AsyncTask* che si occupa di eseguire questa operazione.

[3]Disponibile all'indirizzo code.google.com/p/achartengine/

```
1  public class ExportJSONTask extends AsyncTask<ReadValue, Double, JSONObject> {
2      public static final String VALUE_COUNT = "valueCount";
3      public static final String VALUES = "values";
4
5      private final ExportJSONTaskObserver observer;
6
7      public ExportJSONTask(ExportJSONTaskObserver observer) {
8          this.observer = observer;
9      }
10
11     @Override
12     protected JSONObject doInBackground(ReadValue... readValues) {
13         JSONArray valuesJSONArray = new JSONArray();
14         for (ReadValue readValue : readValues) {
15             try {
16                 valuesJSONArray.put(readValue.toJSONObject());
17             } catch (JSONException e) {
18                 return null;
19             }
20         }
21         try {
22             return new JSONObject()
23                     .put(ReadValue.PERIPHERAL_ID,
                           readValues[0].getPeripheralID())
24                     .put(ReadValue.TIMESTAMP, readValues[0].getTimestamp())
25                     .put(VALUE_COUNT, readValues.length)
26                     .put(VALUES, valuesJSONArray);
27         } catch (JSONException e) {
28             return null;
29         }
30     }
31
32     @Override
33     protected void onPostExecute(JSONObject jsonObject) {
34         super.onPostExecute(jsonObject);
35         if (jsonObject == null) {
36             observer.finishedWithErrors();
37         } else {
38             observer.finished(jsonObject);
39         }
40     }
41 }
```

Listing 7.17: Esportazione dei dati in formato JSON

Capitolo 8

Testing

In questo capitolo verranno descritti i test effettuati sull'applicazione. Verranno inoltre riportati alcuni dati relativi al consumo energetico.

8.1 Modulo di lettura e trasmissione

L'applicazione per CC2650 è stata testata sul dispositivo fornito da *Elements* in diverse condizioni:

- Connesso a un PC e alimentato dal PC.
- Connesso a un PC e alimentato a batteria.
- Alimentato a batteria.

Nonostante le diverse configurazioni di alimentazione e le differenti impostazioni del debugger non sono stati riscontrati problemi di alcun tipo.

Sono state fatte alcune misurazioni riguardanti la corrente consumata dal dispositivo. I dati sono stati rilevati alimentando il dispositivo con una tensione pari a 3.2 V (generata da due batterie di tipo AAA) con parametri impostati nel seguente modo:[1]

[1]Non è possibile stabilire con precisione il valore di tali parametri poiché, come descritto nel capitolo 1, essi vengono scelti dal dispositivo master. Lo slave può solo proporre dei valori.

- **Advertising interval:** 100 ms
- **Desired connection interval:** $100ms < x < 1000ms$
- **Desired connection timeout:** 10 s
- **Desired slave latency:** 0

I dati raccolti sono i seguenti:

- **Fase di advertising:** 2.76 mA
- **Fase di trasmissione (ogni 0.5 s):** 2.48 mA
- **Fase di trasmissione (ogni 10 s):** 2.33 mA
- **Inattivo:** 2.03 mA

È da notare che l'accensione del LED implica un aumento di consumo di circa 1.5 mA. Un modo per risparmiare ulteriore energia è quindi quello di disattivare completamente il LED. Inoltre si può pensare di eliminare il display LCD (di cui, però, non è stato possibile misurare il consumo effettivo).

8.2 Modulo di ricezione ed interpretazione

L'applicazione Android è stata testata su due diversi dispositivi, appartenenti a fasce di mercato diverse. Di seguito verranno elencati i modelli, le caratteristiche hardware più significative e le versioni del sistema operativo Android[2] su cui sono stati eseguiti i test.

- **OnePlus One:**
  - **Processore:** Qualcomm Snapdragon 801 (2.5 GHz, Quad core, 32 bit)
  - **Memoria RAM:** 3 GB
  - **Risoluzione del display:** FullHD (1920×1080)

[2]Nel capitolo 2 si possono trovare informazioni più dettagliate riguardanti le varie versioni.

  - **Versione Bluetooth:** 4.1
  - **Versioni del Sistema Operativo:**
    * *Lollipop* 5.0.1 *CyanogenMod* (*API Level*: 21)
    * *Lollipop* 5.1.1 *CyanogenMod* (*API Level*: 22)

- **Motorola Moto G 2013:**
  - **Processore:** Qualcomm Snapdragon 400 (1.2 GHz, Quad core, 32 bit)
  - **Memoria RAM:** 1 GB
  - **Risoluzione del display:** HD (1280×720)
  - **Versione Bluetooth:** 4.0
  - **Versioni del Sistema Operativo:**
    * *KitKat* 4.4.4 *Stock* (*API Level*: 19)
    * *Lollipop* 5.1 *Stock* (*API Level*: 22)

A dispetto delle notevoli differenze hardware e della diversità di sistema operativo installato non sono stati riscontrati problemi legati né alla perdita di dati, né alla difficoltà di connessione o comunicazione.

Dispositivi meno potenti tendono, tuttavia, ad avere problemi con intervalli di trasmissione inferiore al secondo. In tal caso i dati non sempre vengono mostrati su schermo al momento della trasmissione, ma l'interfaccia grafica si aggiorna ogni secondo (circa). Questo non è comunque un problema in quanto nessun dato viene perso (vengono semplicemente bufferizzati dal sistema operativo stesso) e sono in ogni caso visibili nell'area dedicata alla consultazione dei dati memorizzati.

8.3 Soddisfazione dei requisiti

Come dimostrato dai test e come descritto nelle precedenti sezioni, i requisiti sono stati soddisfatti: il sistema è, infatti, in grado di scambiare dati in tempo reale senza problemi di perdita degli stessi nonostante la frequenza di trasmissione particolarmente elevata.

Conclusioni

Le attività di studio, progettazione e implementazione per la realizzazione di questo progetto sono state davvero interessanti e motivanti. Hanno infatti permesso l'approfondimento di un gran numero di argomenti che precedentemente erano stati affrontati in maniera più superficiale, per esempio la programmazione di sistemi embedded diversi da Arduino o la realizzazione di applicazioni per Android. È stato molto interessante, inoltre, approfondire le conoscenze riguardo alle tecnologie utilizzate, in particolare il Bluetooth Low Energy, sia a livello teorico ma anche potendo mettere le mani su sistemi che le utilizzano.

Il processo di realizzazione si è svolto come pianificato implementando dapprima le parti più importanti, ovvero lo scheletro della comunicazione tramite i due dispositivi e, in seguito, le parti di secondaria importanza ma comunque utili ai fini della dimostrazione delle potenzialità del microcontrollore Texas Instrument CC2650 e della tecnologia Bluetooth Low Energy.

Viste le innumerevoli possibilità che questi dispositivi mettono a disposizione, il risultato è stato assolutamente positivo.

Data la modularità e scalabilità del sistema è inoltre possibile utilizzare lo stesso come base per future realizzazioni che l'azienda vorrà sviluppare nei prossimi anni.

Per la realizzazione di questo progetto sono stati di fondamentale importanza l'appoggio e la fiducia riposta in me dall'intero team di Elements, che si è impegnato prima nella ricerca del dispositivo più adatto e, in seguito, nella realizzazione di un'interfaccia hardware realizzata ad hoc in grado di mostrare le funzionalità del sistema.

Elenco delle figure

1.1 Logo del Bluetooth 2
1.2 Logo del Bluetooth Smart 5
1.3 Topologia connessa: un dispositivo *Central* è connesso e scambia dati con più dispositivi *Peripheral*. 6
1.4 Topologia broadcast: un dispositivo *Peripheral* invia dati contemporaneamente a più dispositivi *Central*. 7
1.5 Schema dell'interfaccia messa a disposizione dal GATT. 8
1.6 Schema dei canali utilizzati dal BLE. In arancione sono evidenziati i canali dedicati all'*advertising*, mentre gli altri vengono utilizzati per la trasmissione di dati. 9

2.1 Logo di Android 13

3.1 Immagine della base di debug SmartRF06 su cui è montato il modulo BLE CC2650 21
3.2 Diagramma a blocchi del microcontrollore. 22

5.1 Diagramma dei casi d'uso del macro-modulo di lettura e trasmissione. 36
5.2 Macchina a stati finiti del macro-modulo di lettura e trasmissione. . 38
5.3 Diagramma dei casi d'uso del macro-modulo di ricezione ed interpretazione. 40
5.4 Diagramma delle classi del modulo di gestione del Bluetooth. 42
5.5 Diagramma delle classi del modulo di gestione dei dati. 43
5.6 Diagramma di sequenza della procedura di lettura e scambio dati. . 44

5.7 Diagramma delle classi del modulo di connessione e visualizzazione in tempo reale. 46
5.8 Diagramma delle classi del modulo di connessione e consultazione dei dati memorizzati. 48
5.9 Diagramma di sequenza della procedura di lettura e scambio dati. . 50
5.10 Diagramma di sequenza della procedura di connessione. 51

6.1 Progettazione del modulo di gestione del Bluetooth. 56
6.2 Progettazione del modulo di gestione dei dati. 58
6.3 Progettazione del modulo di connessione e visualizzazione in tempo reale. 60
6.4 Progettazione del modulo di connessione e consultazione dei dati memorizzati. 62
6.5 In senso orario, partendo dall'alto a sinistra, *Main View*, *Scan*, *Settings*, *Connection History* e le due tab di *Read Values*. 63

Bibliografia

Siti web

- Elements SRL: *elements-ic.com/*
- Bluetooth: *developer.bluetooth.com*
- Texas Instrument: *www.ti.com*
- Android Developers: *android.developers.com*
- Adafruit: *learn.adafruit.com*
- Cplusplus.com: *www.cplusplus.com*
- Javadoc: *http://docs.oracle.com/javase/7/docs/api/overview-summary.html*

Libri

- *Getting Started with Bluetooth Low Energy: Tools and Techniques for Low-Power-Networking*, Kevin Townsend, Carles Cufi, Akiba, Robert Davidson, O'Reilly, 2014
- *Pro Android 4*, Satya Komatineni, Dave MacLean, Apress, 2012

Articoli

- *Monitoraggio wireless nei sistemi medicali portatili*, Redattori Europei, Publitek, 09/07/2015