Ingegneria dei Sistemi Software Adattativi Complessi

Stima della distanza utilizzando tecnologie BLE

Federico Torsello - Matr. 702619

6 ottobre 2016

Indice

Sommario

Capitolo 1 In questo capitolo si viene introdotti al progetto, quindi vengono descritti la vision e i goals.

Capitolo 2 In questo capitolo si definiscono ed analizzano i requisiti funzionali, i casi d'uso e i scenari relativi al progetto.

Capitolo 3 In questo capitolo si analizza brevemente la tecnologia iBeacon ed in generale il BLE (Bluetooth Low Energy). Sono inoltre forniti alcuni esempi di utilizzo reali e caratteristiche del protocollo di comunicazione.

Capitolo 4 In questo capitolo si definisce cos'è RSSI, come calcolarlo e come sfruttarlo per stimare la distanza utente-iBeacon target.

Capitolo 5 In questo capitolo si parla della stima della distanza utilizzando Arduino, prima con un progetto di test con questo connesso al PC e poi con l'implementazione legata all'app. Nel dettaglio si parlerà di come è possibile connettere direttamente l'Arduino allo smartphone per ricevere dati sensoristici.

Capitolo 6 In questo capitolo si struttura il progetto facendo l'analisi delle classi più alcuni codici sorgente di esempio.

Capitolo 7 In questo capitolo si passa all'implementazione vera e propria dell'app, riportando immagini e descrizioni per del suo utilizzo.

Capitolo 1

Introduzione

Stimare con precisione la distanza che intercorre tra un utente ed un punto target utilizzando poco hardware, ad un costo accessibile, è tutt'ora una sfida tecnologica a cui si cerca una soluzione valida. Questa informazione è diventata via via sempre più importante in quanto vari sistemi (mobile e non) ne necessitano per poter funzionare in modo corretto.

Per stimare la distanza esistono diverse tecnologie più o meno precise e costose. In questo senso l'avvento delle tecnologie IoT (*Internet of things*) ha influito positivamente abbassando il costo dell'hardware necessario, creando community di hobbisti e professionisti; quindi ampliando il numero di librerie software (spesso open source) disponibili e progetti da cui prendere spunto.

D'altro canto, vista la disponibilità pressoché capillare degli smarphone, sembra ovvio cercare di utilizzarli come risorsa per trovare una soluzione al problema citato. Tali dispositivi mobili implementano sensoristica, capacità di calcolo e software (inteso come OS o come librerie) sfruttabile direttamente dall'utente mediante un'app creata ad-hoc.

In questo senso i dispositivi iBeacon (grazie alla tecnologia BLE) possono essere utilizzati in relazione agli smartphone, aiutando a stimare la distanza utente-iBeacon con un costo limitato ed una buona efficienza.

1.1 Vision

- Realizzare un sistema software mobile per stimare al meglio la distanza che intercorre tra l'utente e gli iBeacon disposti in un ambiante indoor.
- Utilizzare solo tecnologie open source per realizzare il tutto, ribadendone l'utilità e l'efficienza.

1.2 Goals

1.2.1 Goals principali

1. Sviluppare un'app Android in grado di interagire con degli iBeacon disposti in una stanza.
2. Realizzare un'app compatibile con tutte le API Android 18 e superiori.
3. L'app deve utilizzare i valori RSSI dei iBeacon per determinare la distanza trasmettitore-ricevitore.
4. Implementare diversi filtri per ridurre gli effetti indesiderati del *multi-path fading* sulla stima della distanza.
5. Realizzare e testare un **filtro di Kalman**, un **filtro ARMA** (*Auto Regressive Moving Average*) e un **filtro RunningAverageRssi** per limitare gli effetti indesiderati sopracitati.
6. Visualizzare dei grafici sull'app che in tempo reale descrivano l'andamento della distanza stimata.
7. Curare la *User Experience* realizzando una GUI chiara e responsive utilizzando librerie *com.android.support*.
8. Sviluppare l'intero sistema utilizzando GNU/Linux e FOSS (*Free and open-source software*).

1.2.2 Goals secondari

1. Realizzare un programma C++/Wiring in grado di determinare la distanza percepita da un sensore ultrasonico collegato ad una board Arduino.
2. Realizzare un mini progetto per testare il sensore ultrasonico. Nello specifico si considera un Arduino connesso al PC attraverso la porta USB e una view di feedback per visualizzare a schermo la distanza "reale" (o meglio, di riferimento) stimata.
3. Abilitare la comunicazione seriale mediante tecnologia **USB OTG**, facendo diventare lo smartphone un host USB.
4. Implementare la visualizzazione della distanza percepita dall'Arduino direttamente nell'app. L'obiettivo è dare un feedback della distanza di rifermento iBeacon-utente per poterla confrontare con quella stimata utilizzando gli RSSI.

   In questo caso si considera un **Arduino connesso allo smartphone** attraverso la porta USB.

Capitolo 2

Requisiti

2.1 Requisiti funzionali

Per la realizzare del sistema si necessita:

- di un utente in possesso di uno smartphone con sistema operativo Android versione 4.3 o superiore;
- uno o più iBeacon;
- un ambiente chiuso (come una stanza) in cui disporre gli iBeacon.

2.2 Requisiti funzionali secondari

Ai fini di un confronto tra la distanza stimata grazie agli iBeacon e quella "reale" si vuole utilizzare una board Arduino ed un sensore ultrasonico; quindi creare un feedback a schermo. Nello specifico si necessita di:

- una board Arduino UNO (o superiore);
- un sensore ultrasonico;
- cavi di collegamento maschio-femmina;
- un cavo USB da stampante;
- uno smartphone che supporti la tecnologia **USB OTG (*On-The-Go*)**[1]
- un cavo USB OTG.

[1] USB OTG - https://it.wikipedia.org/wiki/USB_On-The-Go

2.3 Requisiti non funzionali

I requisiti funzionali di questo progetto sono molteplici:

- ottenere una GUI fluida e di facile utilizzo.
- realizzare dei filtri che non incidano sulle prestazioni del sistema.
- avere dei feedback reali di quello che succede, con numeri ben visibili e grafici in realtime con colori distinguibili.
- poter registrare i dati (sotto forma di immagine o testo) per analisi a posteriori.
- ottenere un'app che non vada in crash.

2.4 Analisi dei requisiti

2.4.1 Casi d'uso

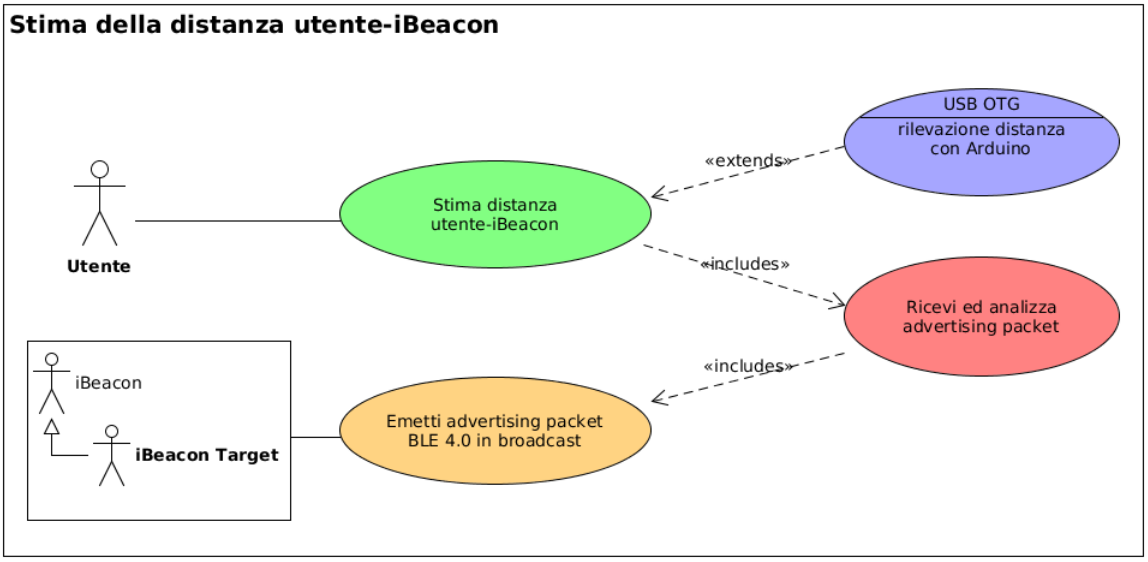

Figura 2.1: Use case - Stima della distanza utente-iBeacon

2.4.2 Scenari

Scenario classico

Nello scenario classico l'utente utilizza l'app per scansionare l'ambiente alla ricerca di iBeacon e leggere delle informazioni a schermo.

Nello specifico in questo scenario:

- Si considera un utente in una stanza con uno smartphone Android in mano.
- La stanza in questione può presentare uno o più iBeacon disposti sui muri ad un'altezza di circa 1 metro dal suolo.
- Sullo smartphone viene avviata l'app.
- Nel caso in cui la radio Bluetooth dello smarphone fosse spenta o si dovesse spegnere, l'app stessa la accenderà/riaccenderà indicando all'utente l'avvenuto switch.
- L'utente preme su un bottone per avviare la scansione degli iBeacon presenti.
- Se vengono rilevati iBeacon corrispondenti con la lista di indirizzi MAC presente nell'app, allora viene caricata una lista di informazioni a schermo (*friendly name*, immagine, distanza stimata, ecc...).

Scenario classico avanzato

- Una volta che l'utente ha visualizzato l'iBeacon di suo interesse (detto *target*), lo seleziona dalla lista facendo click sull'item corrispondente.
- A questo punto all'utente vengono proposti solo i dettagli relativi al target, isolandoli dagli altri.
- In questa sezione l'utente potrebbe visualizzare un feedback della distanza reale (se tutte le condizioni a contorno sono soddisfatte) in metri.
- Trascinando il dito da destra a sinistra all'utente viene proposta un'altra area ancora più dettagliata dove è possibile mettere a confronto le varie stime della distanza (filtrata, non filtrata e reale se presente), in dei grafici realizzati in tempo reale.

Capitolo 3

Breve introduzione alla tecnologia iBeacon

Gli iBeacon (anche detti BLE Beacon) sono dispositivi portatili, spesso alimentati a batteria, che integrano una radio Bluetooth. Il loro scopo è definire **regioni** di spazio delimitato dall'estensione del segnale che emettono in broadcast ripetutamente con una frequenza predefinita. Si rendono utili per *micro-location system* in quanto offrendo una precisione molto senza forti ripercussioni sulla loro autonomia.

Spesso sono utilizzati come *trigger* per App che integrano *custom action*, reagendo alla prossimità, creando delle reazioni visibili sullo smartphone che utente appaiono come **interazioni col mondo fisico**.

La tecnologia chiave che permette agli iBeacon di funzionare è la **proximity specification BLE** (Bluetooth Low Energy), spesso indicata come **Bluetooth Smart**. BLE è un miglioramento della specifica Bluetooth che permette un consumo energetico ridotto che si ripercuote positivamente sull'autonomia dei dispositivi a batteria.

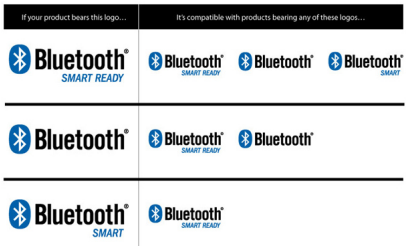

Figura 3.1: Compatibilità delle diverse tecnologie Bluetooth

Il limitato consumo energetico degli iBeacon si deve alla ridotta potenza di trasmissione di questa tecnologia che permette lunghi periodi di utilizzo (mesi o addirittura anni).

3.1 Il protocollo iBeacon

Il protocollo di comunicazione iBeacon è semplice e lineare. Si suddivide tra **broadcaster** (trasmettitore) e **receiver** (ricevitore).

3.1.1 Broadcaster

Nella specifica Bluetooth, un iBeacon è per definizione un *broadcaster*. I broadcaster trasmettono **advertising packet** periodici che contengono informazioni utilizzabili dai *receiver*.

3.1.2 Receiver

In questo protocollo i receiver non devono rispondere ai pacchetti che ricevono ed in generale i broadcaster non sono abilitati alla ricezione. La comunicazione degli advertising packet avviene sempre e solo attraverso una trasmissione unidirezionale.

3.1.3 Advertising packet

Essenzialmente gli advertising packet sono dispersi dai broadcaster nell'aria e i receiver possono scegliere se agire o no in base al loro contenuto.

La ricezione di un advertising packet da parte di un receiver provoca la creazione di un **advertisement event**. Per questo gli advertising packet e gli advertisement event genericamente sono denominati come **advertisement**.

Figura 3.2: Logo che identifica i dispositivi iBeacon

3.2 Frame iBeacon

Il formato del frame iBeacon è abbastanza semplice. Sono presenti solo un due parametri variabili che sono comunque sufficienti per supportare applicazioni complesse.

Le informazioni che un iBeacon emette si suddividono in tre identificativi:

- Universal Unique Identifier
- Major number
- Minor number

3.2.1 UUID (128 bits)

L'UUID identifica univocamente la società di cui l'iBeacon fa parte. A differenza di altri protocolli di rete come 802.11, l'UUID non è gestito centralmente per evitare conflitti. Il protocollo Bluetooth presuppone che UUID siano unici.

L'UUID è il numero più facile da elaborare perché dovrebbe essere unico. Un sistema che supporta più brand name può utilizzare diversi UUID.

La maggior parte degli UUID sono creati da generatori di numeri casuali, spesso integrando l'ora corrente e un identificatore del generatore (ad esempio l'indirizzo MAC). Molte applicazioni per la configurazione di iBeacon commerciali hanno un pulsante per generare un UUID pseudocasuale.

3.2.2 Major number (16 bits)

Le specifiche Bluetooth e iBeacon non dispongono di una struttura per l'uso dei major o minor numers.

Il Major number viene utilizzato per identificare i Major groups di iBeacon di proprietà di un'unica entità. Nell'esempio della catena di negozi, il Major number tipicamente è utilizzato dai iBeacon che si trovano all'interno del negozio, identificando gruppi di proximity area in modo logico. Il campo a 16 bit permette di avere 65.000 possibilità.

3.2.3 Minor number (16 bits)

Il Minor number viene utilizzato per identificare il livello più basso della gerarchia all'interno di un insieme di iBeacon. Tornando all'esempio di una catena di negozi, il Minor number sarà utilizzato per i singoli iBeacon all'interno di una singola posizione del negozio, per esempio potrebbe identificare un prodotto esposto.

Il Minor number è una ulteriore suddivisione del raggruppamento definito dal Major number. I Minor number di solito si riferiscono ad una posizione geografica o POI all'interno di una posizione.

3.3 RSSI - Received Signal Strength Indication

La **proximity estimation** (stima di prossimità) dipende dal RSSI. RSSI non è trasmesso nel advertising packet, ma viene percepito dal receiver come il livello di potenza del segnale ricevuto.

3.3.1 Advertising Interval

Anche se le specifiche Bluetooth permettono di definire più *advertising interval*, la specifica iBeacon fissa l'advertising interval a 100 ms.

L'impostazione dell'advertising interval è bilanciato in modo da preservare la vita della batteria, ma comunque essere abbastanza lungo per consentire ai servizi basati sugli iBeacon di funzionare.

3.4 iBeacon Advertising Packet Contents

Tutti i advertising packet hanno sempre la stessa lunghezza e sono composti da una serie di campi fissi. L'ultima parte del frame contiene informazioni specifiche del costruttore definito da Apple. È comunque possibile definire un formato pacchetto personalizzato in modo da migliorare le capacità di puntamento un iBeacon.

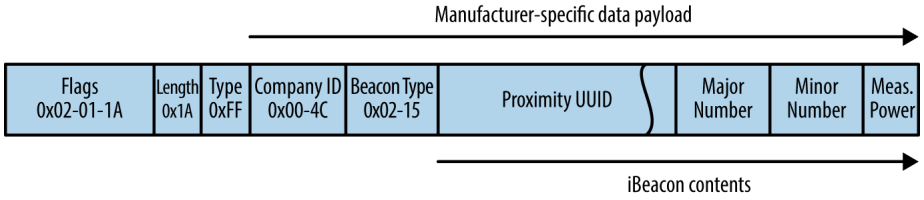

Figura 3.3: Formato di un iBeacon advertising packet format

3.4.1 Measured power

L'idea di **ranging** è implicita all'interno del protocollo iBeacon. Il *ranging* indica la distanza tra il receiver e l'iBeacon analizzando le variazioni di potenza della trasmissione ricevuta. La distanza da un iBeacon viene stimata in base alla costante di calibrazione adeguata, misurando il tempo che un segnale impiega per poter essere ricevuto e la sua potenza.

Il *Measured power* è un parametro impostato tenendo un receiver ad un metro dall'iBeacon e calcolando la media di RSSI. Questo campo contiene la potenza misurata come complemento a due.

Esempio: un valore di C5 indica che la potenza misurata a un metro è di -59 dBm.

Dati:

- $\texttt{0xC5}_{\texttt{hex}} = \texttt{197}_{\texttt{dec}}$
- $\texttt{0x100}_{\texttt{hex}} = \texttt{256}_{\texttt{dec}}$

```
(256 - 197) = -59 dBm
```

Capitolo 4

Stima della distanza con RSSI

4.1 Attenuazione dei segnali elettromagnetici

Il fenomeno dell'**attenuazione** dei segnali elettromagnetici si manifesta nella decrescita della potenza di segnale ricevuto dal ricevitore in relazione all'aumentare della distanza dalla sorgente emittente di tale segnale.

Nota questa relazione, se si conosce la potenza di segnale del trasmettitore **P** è possibile creare un modello per legare l'***attenuazione di segnale*** **A** e la distanza col ricevitore al fine di stimare la ***distanza relativa*** trasmettitore-ricevitore **d**.

$$P = f(d) \tag{4.1}$$

4.2 Received Signal Strength Indicator - RSSI

Per stimare la distanza relativa tra trasmettitore e ricevitore si può sfruttare l'attenuazione di segnale indicata dal valore RSSI il cui calcolo è poco costoso e non necessita di hardware aggiuntivo.

RSSI è utile per smartphone che implementano radio Bluetooth in quanto permette di sviluppare proximity app o localizzazione indoor. Il suo valore viene calcolato in automatico dal sistema operativo dello smartphone.

Purtroppo questo approccio non è particolarmente preciso in quanto l'ambiente influisce sulla potenza del segnale ricevuto e quindi sull'attenuazione reale che rende il valore RSSI oscillante.

4.3 Calcolo di RSSI

Per il calcolo di RSSI (in dBm) si assume:

1. che un segnale radio emesso in nello spazio libero da ostacoli e rumore decade di un fattore d^{-2}, dove **d** è la distanza relativa trasmettitore-ricevitore

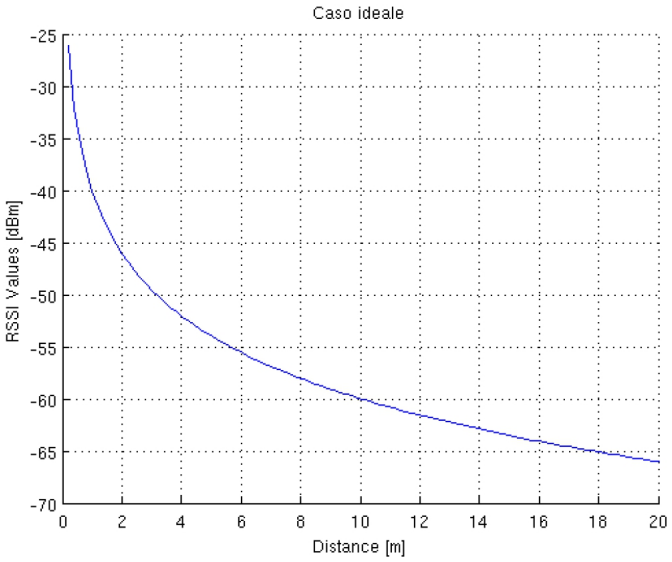

Figura 4.1: RSSI - Modello dell'andamento di RSSI in funzione della distanza

2. che la potenza media ricevuta attraverso un canale reale decade proporzionalmente a $n = d^{-n}$, dove **n** è detto **l'esponente path-loss**. Tipicamente n è compreso tra 2 e 4.

4.3.1 Equazione di trasmissione di Friis

La distanza dal trasmettitore viene valutata utilizzando l'**equazione di trasmissione di Friis**:

$$P_R = P_T \frac{G_T G_R \lambda^2}{(4\pi)^2 \mathbf{d}^n} \tag{4.2}$$

dove:

- P_R : potenza del segnale ricevuto (espressa in Watt)
- P_T : potenza del segnale trasmesso (espressa in Watt)
- G_R : guadagno dell'antenna ricevente
- G_T : guadagno dell'antenna trasmittente
- $\lambda = \frac{v}{f}$: lunghezza d'onda
  v : velocità di propagazione
  f : frequenza dell'onda
- d : distanza in metri

- n : constante di propagazione del segnale che dipende dall'ambiente

L'equazione (4.2) calcola il rapporto tra la potenza ricevuta da un'antenna e la potenza trasmessa, in condizioni ideali.

4.3.2 Conversione della potenza

Conversione dalla potenza espressa in watt alla potenza espressa in dBm

$$1_{[dBm]} = 0.001258925_{[W]} \tag{4.3}$$

$$1_{[W]} = 30_{[dBm]} \tag{4.4}$$

$$P_{[dBm]} = 10 \log_{10}(10^3 P_{[W]}/1_{[W]}) \tag{4.5}$$

4.3.3 Potenza media a distanza di riferimento d_0

$$P(d)_{[dBm]} = P_{0\,[dBm]} \left(\frac{d}{d_0}\right)^{-n} \tag{4.6}$$

dove P_0 è la potenza ricevuta (dBm) a una piccola distanza di riferimento d_0.

4.3.4 Equazione di RSSI

Combinando la (4.2) e la (4.5), applicando le proprietà dei logaritmi si ottiene:

$$RSSI = -(10\, n \log_{10} \mathbf{d} - A) \tag{4.7}$$

dove A è la potenza del segnale ricevuto (dBm) a distanza di un metro considerando una costante di propagazione n.

4.4 Calcolo della distanza

La seguente equazione permette di stimare la distanza tra l'utente ed un target conoscendo il valore RSSI ed i parametri A ed n:

$$RSSI = P{-}10{*}n{*}\log_1 0(d){*}n = 2(infreespace){*}d = 10^(TxPower{-}RSSI)/(10{*}n)) \tag{4.8}$$

$$\mathbf{d} = 10 \left(\frac{A - RSSI}{10\, n}\right) \tag{4.9}$$

Con questa formula si può stimare la distanza .

4.5 Problematiche della stima della distanza con RSSI

I principali fenomeni negativi che inficiano l'utilizzo di RSSI come approccio per determinare la distanza tra due punti sono:

- **Riflessione:** il segnale si propaga anche attraverso un percorso riflesso, provocando un **multi-path fading**. Al ricevitore giungono segnali con ampiezze e fasi differenti che vanno a sommarsi o sottrarsi in funzione della frequenza, causando un fading selettivo. Può essere causato da metalli e altri materiali riflettenti.
- **Intralcio:** shadowing che altera il normale decadimento dell'intensità che si avrebbe in spazio libero. L'attenuazione improvvisa del segnale è causata degli ostacoli (mobili, muri, alberi, edifici, ecc.) nel cammino trasmettitore-ricevitore
- **Assorbimento:** oggetti, come elementi liquidi o corpi umani, che assorbono la potenza del segnale.
- **Altezza:** la differenza di altezza può falsare la stima
- **Orientamento relativo:** il segnale decade se il ricevitore non è direzionato verso l'emettitore

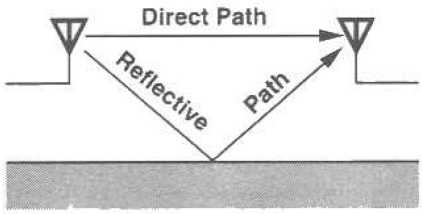

Figura 4.2: RSSI - Problemi di riflessione

Capitolo 5

Stima della distanza con Arduino

5.1 Progetto per stimare la distanza con un sensore ultrasonico ed un Arduino

Per poter stimare la distanza con Arduino è stato utilizzato un sensore ultrasonico. Questo sensore è stato connesso direttamente alla board attraverso cavi maschio-femmina.

Per

- Implementare un progetto Arduino che faccia uso della libreria **NewPing**[1];
- Sfruttare la libreria **usb-serial-for-android**[2] per connettere l'Arduino ad Android e ricevere i dati seriali in tempo reale.
- Sviluppare un mini progetto su **NetBeans**[3] ed impiegare la libreria **jSSC**[4] (*java-simple-serial-connector*) per testare il sensore di prossimità collegando l'Arduino al PC e visualizzando la distanza a schermo.

NewPing

Caratteristiche

- Compatibile con diversi modelli di sensori ad ultrasuoni: SR04, SRF05, SRF06, DYP-ME007 e Parallax Ping™.
- Non soffre di **lag** di un secondo se non si riceve un ping di eco.

[1] **NewPing** - http://playground.arduino.cc/Code/NewPing
[2] **usb-serial-for-android** - https://github.com/mik3y/usb-serial-for-android
[3] **NetBeans** - https://netbeans.org/
[4] **jSSC** - https://github.com/scream3r/java-simple-serial-connector

- Produce un ping coerente e affidabile fino a 30 volte al secondo.
- Timer interrupt method per sketch event-driven.
- Metodo di filtro digitale built-in `ping_median()` per facilitare la correzione degli errori.
- Utilizzo dei registri delle porte durante l'accesso ai pin per avere un'esecuzione più veloce e dimensioni del codice ridotte.
- Consente l'impostazione di una massima distanza di lettura del ping "in chiaro".
- Facilita l'utilizzo di più sensori.
- Calcolo distanza preciso, in centimetri, pollici e uS.
- Non fa uso di `pulseIn`, metodo che risulterebbe lento e che con alcuni modelli di sensore a ultrasuoni restituisce risultati errati.
- Attualmente in sviluppo, con caratteristiche che vengono aggiunte e bug/issues affrontati.

5.2 Progetto di test della comunicazione seriale PC - Arduino

Il metodo più importante di questo mini progetto è `updateDistance()`. Questo metodo è di tipo **void**. Il suo scopo è quello di riceve in input i *byte* dalla porta seriale e convertirli in stringhe da visualizzare a schermo su una *jLabel*.

Per poter fare I/O sulla porta seriale si deve istanziare e configurare l'oggetto SerialPort, abilitando la comunicazione via USB con una board Arduino.

Per avviare la comunicazione:

- si settano i parametri di ingaggio (baund rate, numero di bit dei pacchetti, numero dei bit di stop e se è presente un controllo di parità)
- si imposta l'event mask in modo da controllare se sul canale sono prenti *char*
- si registra l'istanza di SerialPort in un listener di eventi di I/O della seriale per poi considerare solo i dati di tipo *char* e scartare tutti gli altri.

Metodo updateDistance()

```
private void updateDistance() {
    SerialPort serialPort = new SerialPort("/dev/ttyACM0");
    try {
        serialPort.openPort();

        serialPort.setParams(
            SerialPort.BAUDRATE_115200,
            SerialPort.DATABITS_8,
            SerialPort.STOPBITS_1,
            SerialPort.PARITY_NONE);

        serialPort.setEventsMask(SerialPort.MASK_RXCHAR);

        serialPort.addEventListener((SerialPortEvent
            serialPortEvent) -> {
            if (serialPortEvent.isRXCHAR()) {
                try {
                    Thread.sleep(20);
                    String distance =
                        serialPort.readString();
                    jLabel1.setText(distance);
                } catch (SerialPortException |
                    InterruptedException ex) {
                }
            }
        });
    } catch (SerialPortException ex) {
        System.out.println("SerialPortException: " +
            ex.toString());
    }
}
```

Capitolo 6

Project

- Interagire con gli iBeacon sfruttando la libreria **AltBeacon**[1].
- Implementare un **filtro di Kalman** per ridurre gli effetti indesiderati del *multipath fading* sulla stima della distanza.
- Implementare un **filtro ARMA** (*Auto Regressive Moving Average*) per stimare al meglio il valore RSSI.
- Implementare un **filtro RunningAverageRssi** da confrontare con i precedenti.
- Visualizzare grafici in tempo reale utilizzando la libreria **MPAndroidChart**[2]

6.1 ApplicationActivity

[1] **AltBeacon** - https://github.com/AltBeacon/android-beacon-library
[2] **MPAndroidChart** - https://github.com/PhilJay/MPAndroidChart

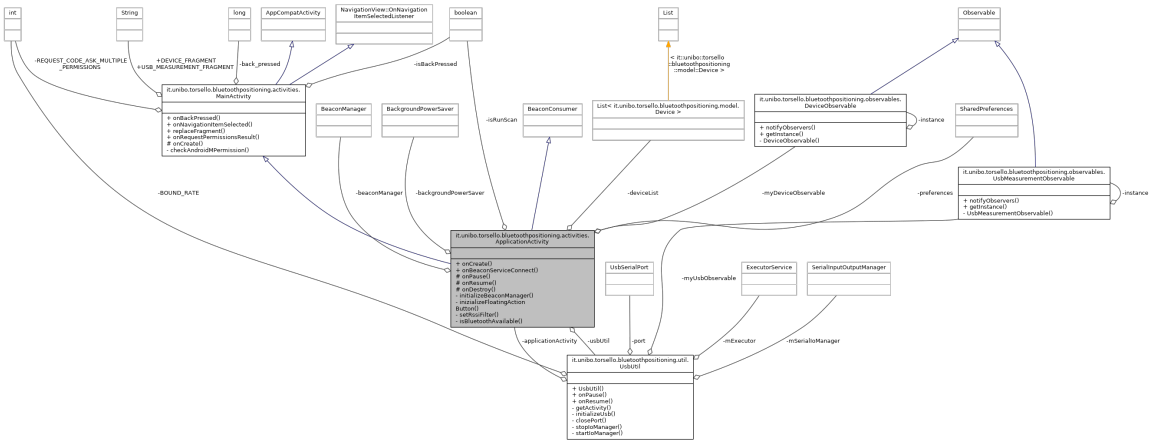

Figura 6.1

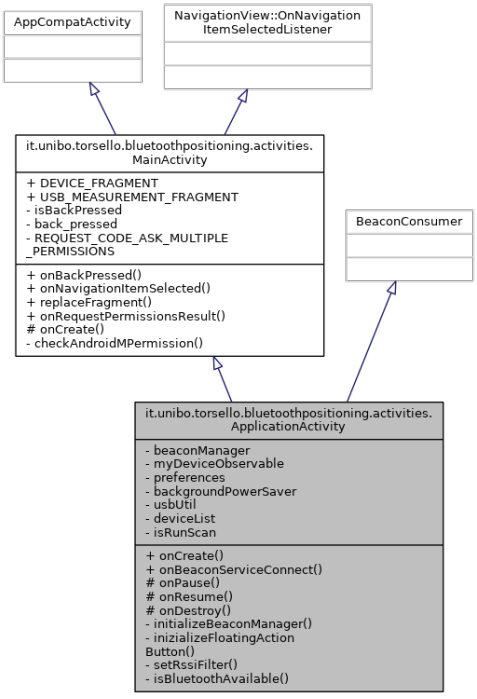

Figura 6.2

6.2 MainActivity

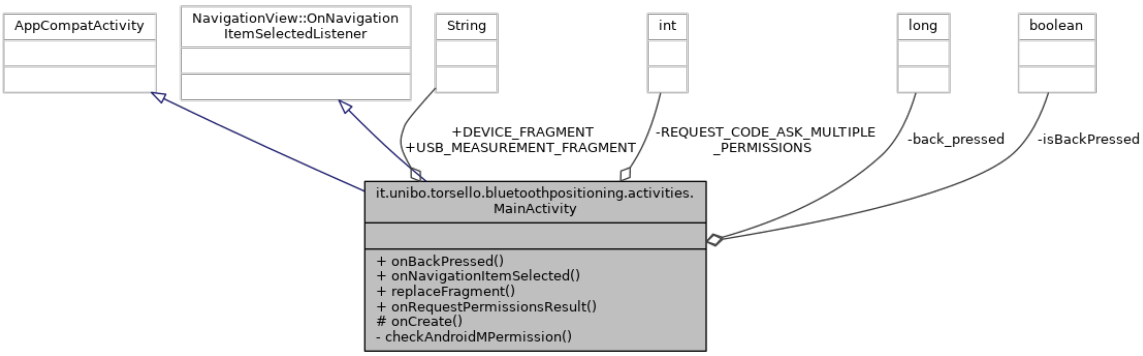

Figura 6.3

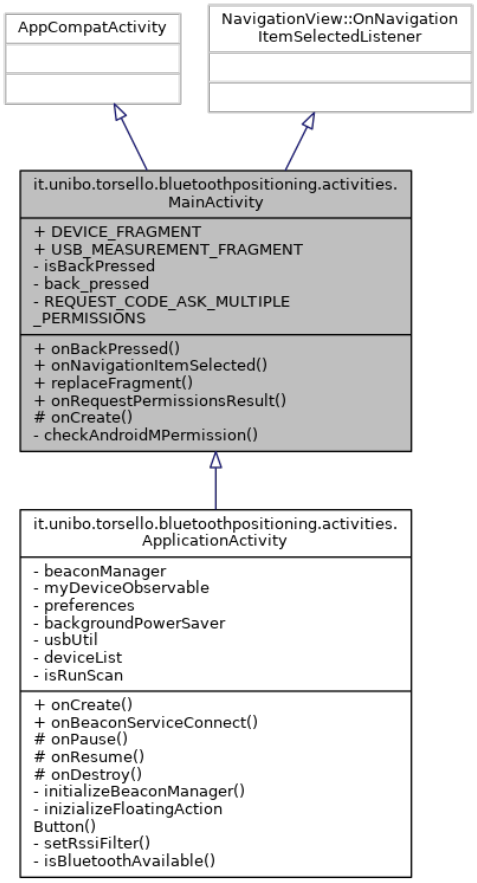
AppCompatActivity
NavigationView::OnNavigation
ItemSelectedListener
it.unibo.torsello.bluetoothpositioning.activities.
MainActivity
+ DEVICE_FRAGMENT
+ USB_MEASUREMENT_FRAGMENT
- isBackPressed
- back_pressed
- REQUEST_CODE_ASK_MULTIPLE
_PERMISSIONS
+ onBackPressed()
+ onNavigationItemSelected()
+ replaceFragment()
+ onRequestPermissionsResult()
# onCreate()
- checkAndroidMPermission()
it.unibo.torsello.bluetoothpositioning.activities.
ApplicationActivity
- beaconManager
- myDeviceObservable
- preferences
- backgroundPowerSaver
- usbUtil
- deviceList
- isRunScan
+ onCreate()
+ onBeaconServiceConnect()
# onPause()
# onResume()
# onDestroy()
- initializeBeaconManager()
- inizializeFloatingAction
Button()
- setRssiFilter()
- isBluetoothAvailable()

Figura 6.4

6.3 DeviceViewHolder

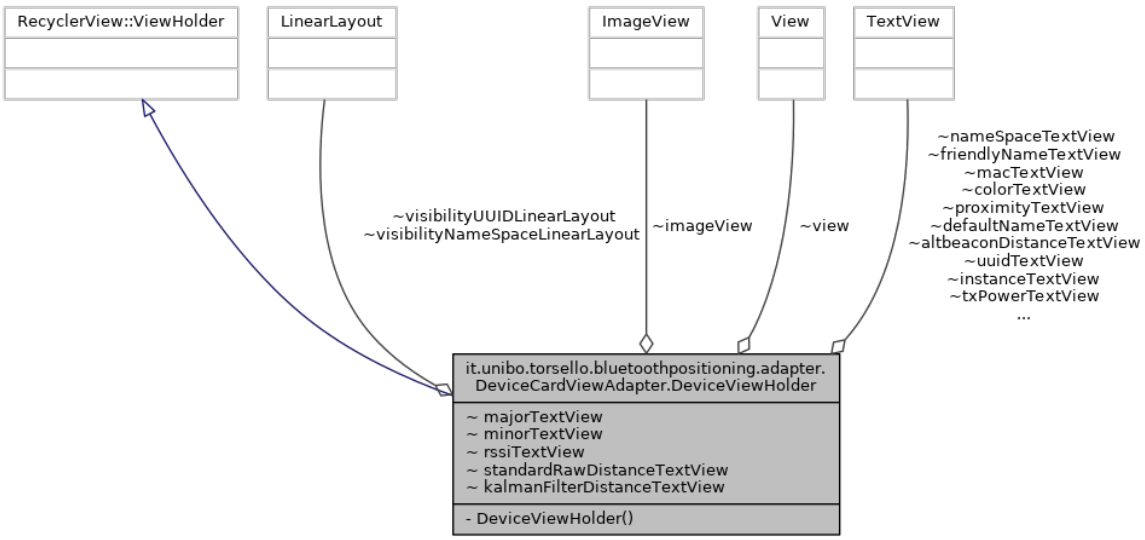

Figura 6.5

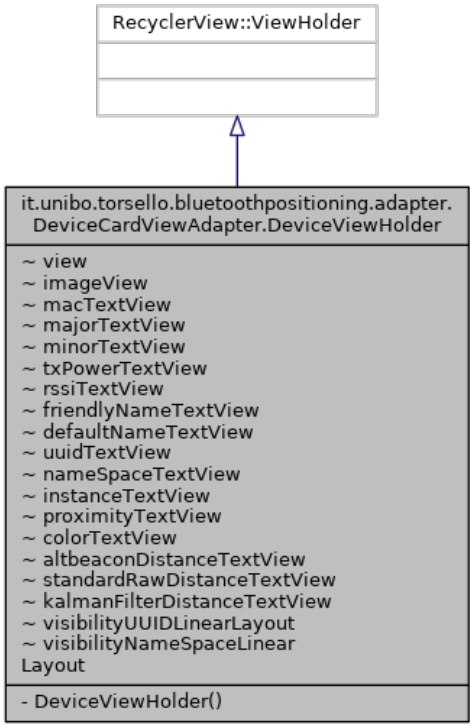
RecyclerView::ViewHolder
it.unibo.torsello.bluetoothpositioning.adapter.
DeviceCardViewAdapter.DeviceViewHolder
~ view
~ imageView
~ macTextView
~ majorTextView
~ minorTextView
~ txPowerTextView
~ rssiTextView
~ friendlyNameTextView
~ defaultNameTextView
~ uuidTextView
~ nameSpaceTextView
~ instanceTextView
~ proximityTextView
~ colorTextView
~ altbeaconDistanceTextView
~ standardRawDistanceTextView
~ kalmanFilterDistanceTextView
~ visibilityUUIDLinearLayout
~ visibilityNameSpaceLinear
Layout
- DeviceViewHolder()

Figura 6.6

6.4 DeviceCardViewAdapter

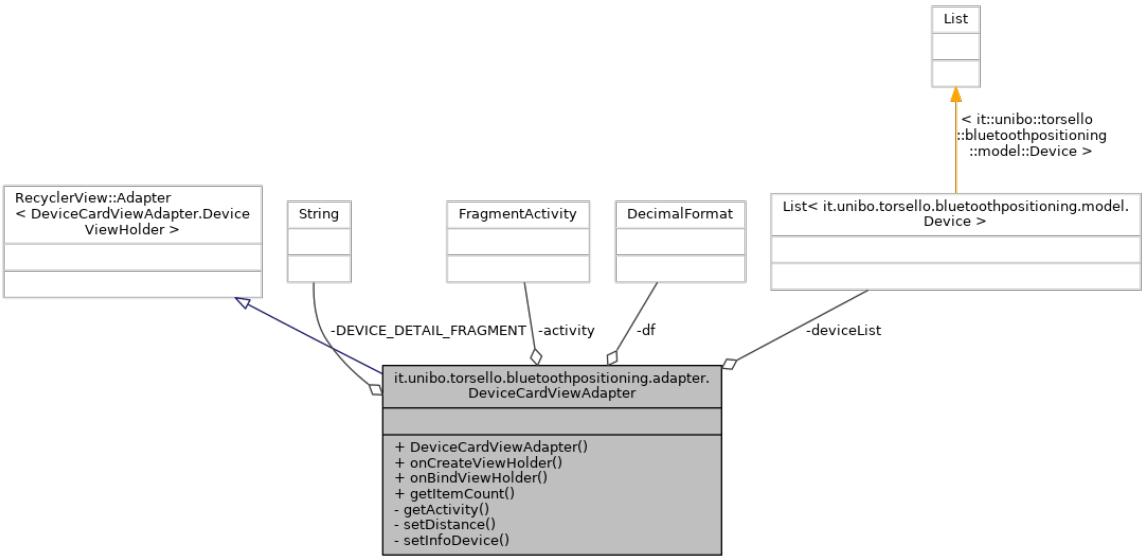

Figura 6.7

6.5 StatePagerAdapter

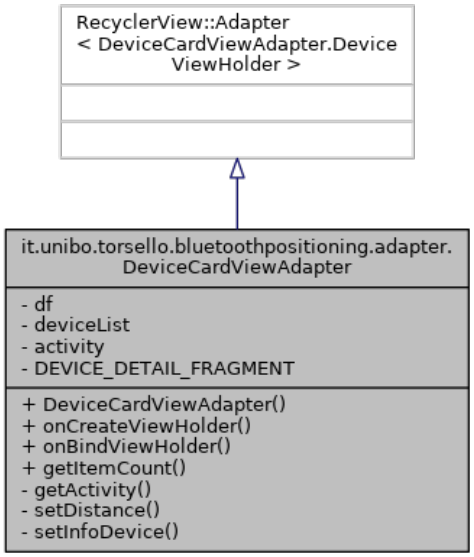

Figura 6.8

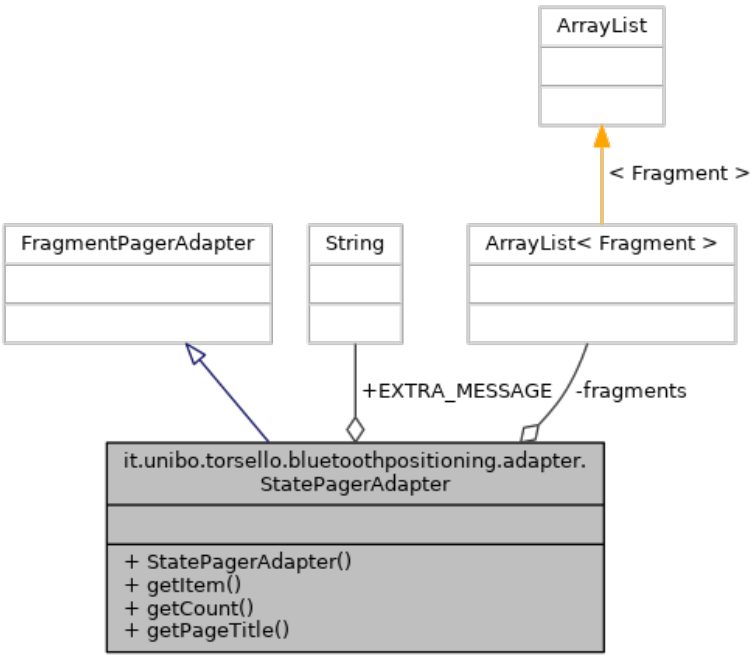

Figura 6.9

6.6 MyArmaRssiFilter

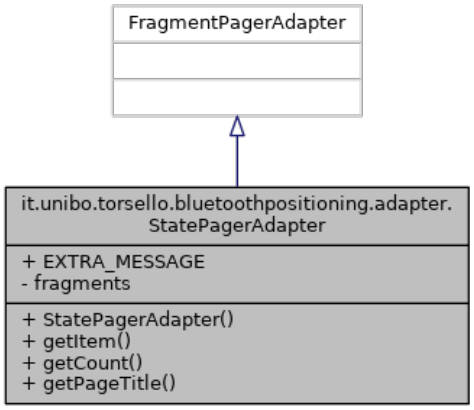

Figura 6.10

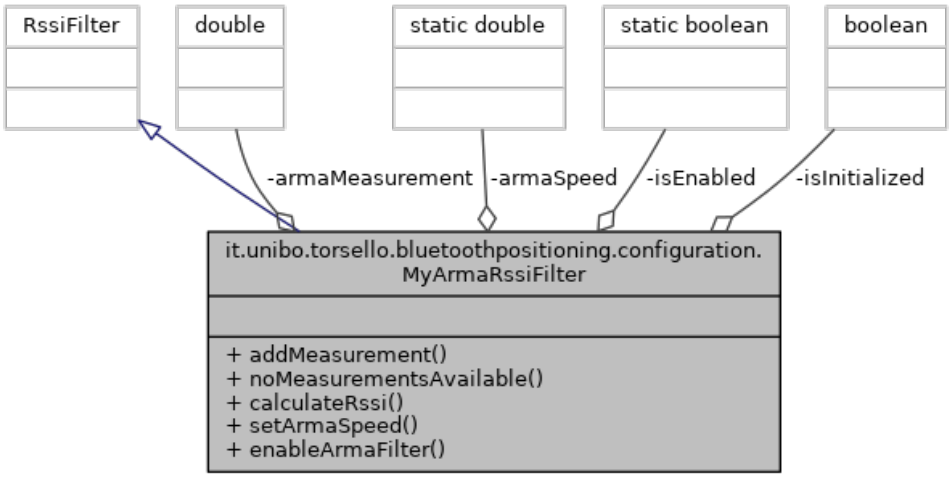

Figura 6.11

6.7 DeviceConstants

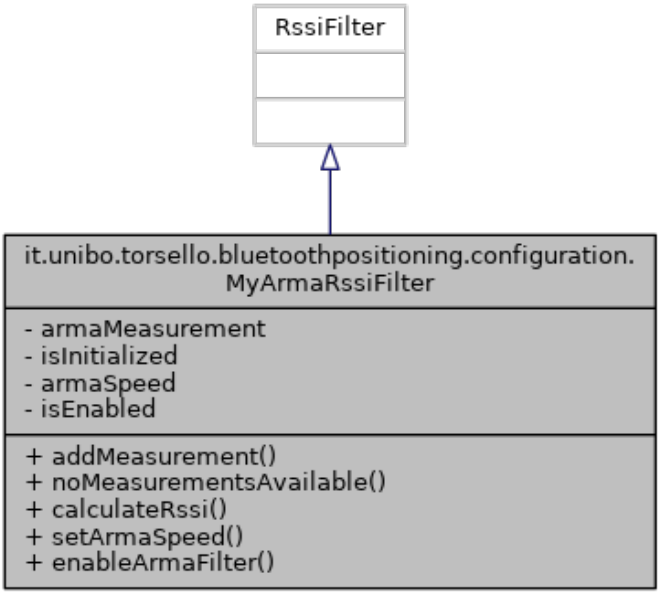

Figura 6.12

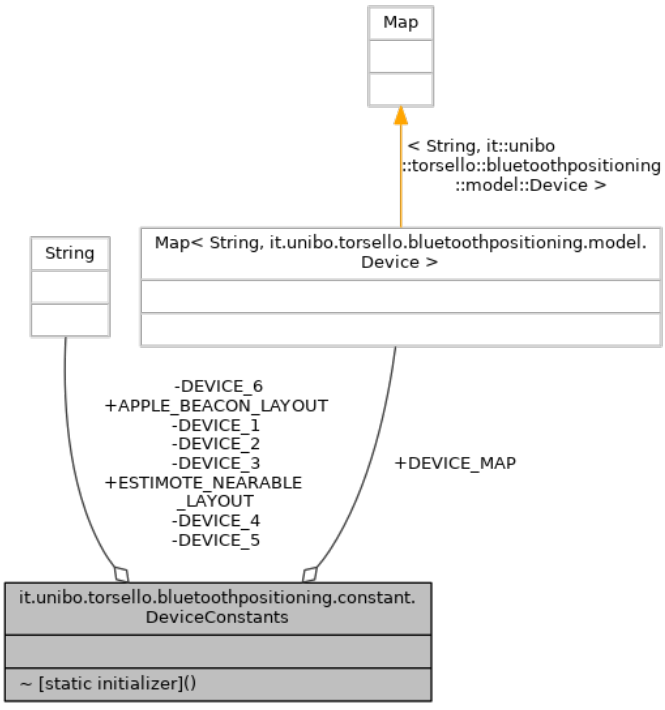

Figura 6.13

6.8 KFilterConstants

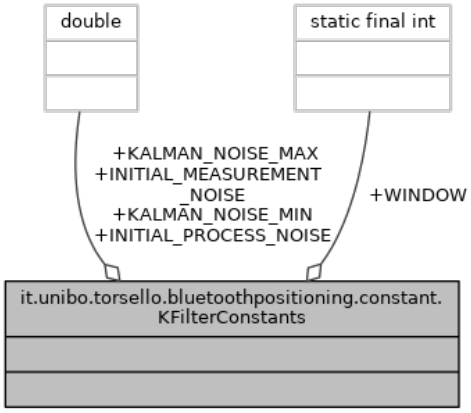

Figura 6.14

6.9 SettingConstants

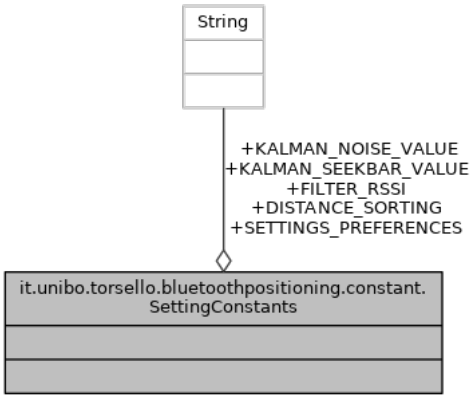

Figura 6.15

6.10 Estimation

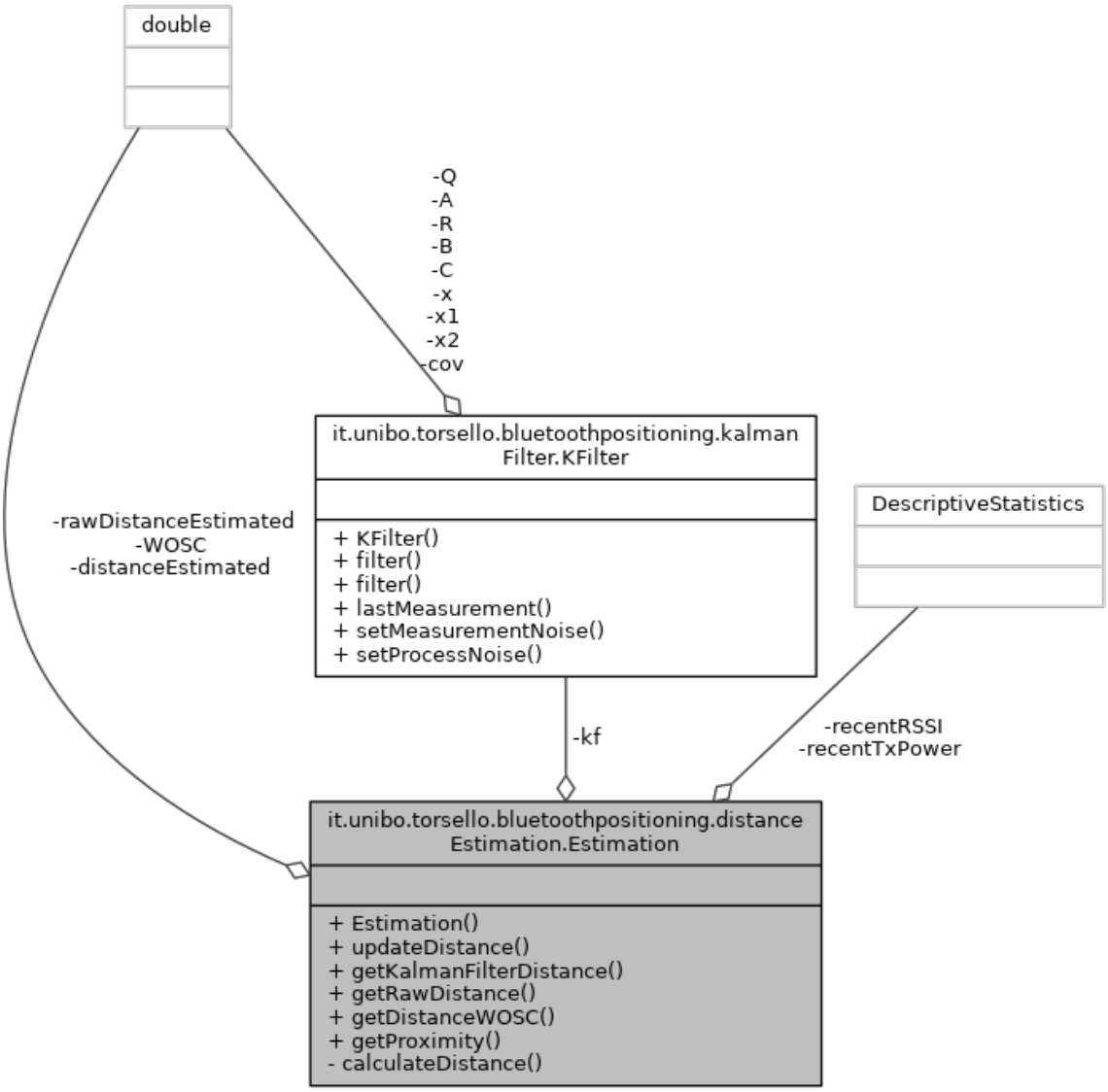
double
-Q
-A
-R
-B
-C
-x
-x1
-x2
-cov
it.unibo.torsello.bluetoothpositioning.kalman
Filter.KFilter
+ KFilter()
+ filter()
+ filter()
+ lastMeasurement()
+ setMeasurementNoise()
+ setProcessNoise()
DescriptiveStatistics
-rawDistanceEstimated
-WOSC
-distanceEstimated
-kf
-recentRSSI
-recentTxPower
it.unibo.torsello.bluetoothpositioning.distance
Estimation.Estimation
+ Estimation()
+ updateDistance()
+ getKalmanFilterDistance()
+ getRawDistance()
+ getDistanceWOSC()
+ getProximity()
- calculateDistance()

Figura 6.16

6.11 FABBehavior

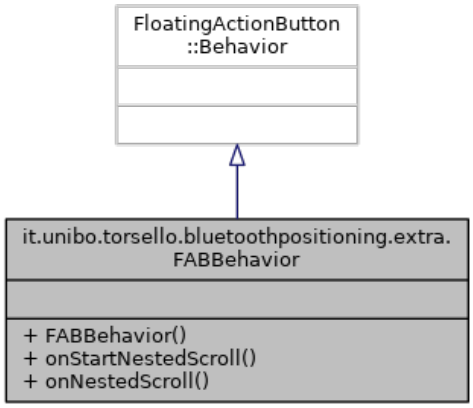

Figura 6.17

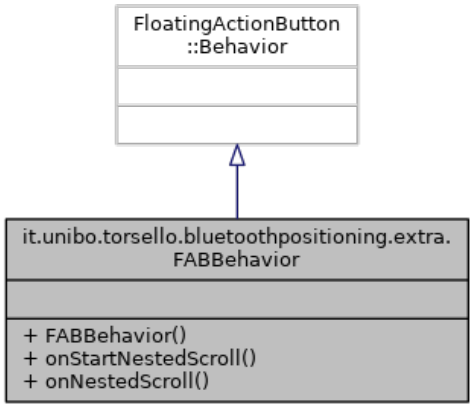

Figura 6.18

6.12 CameraFragment

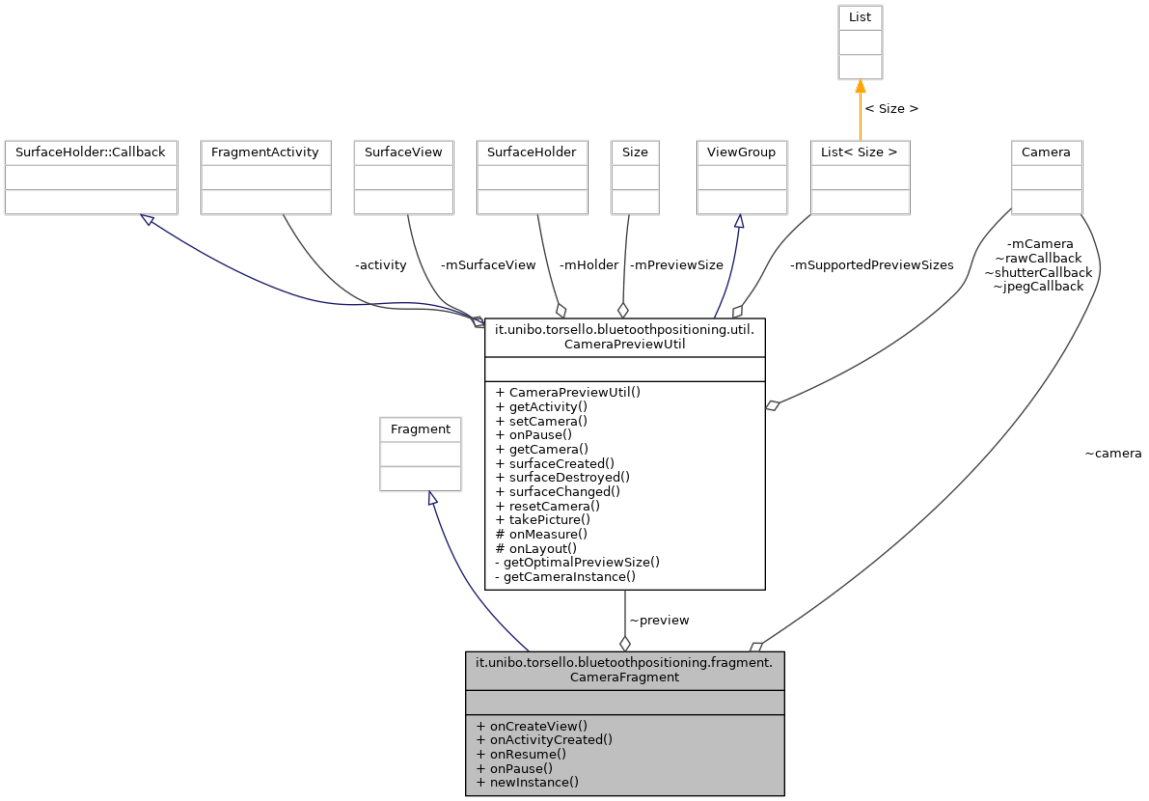

Figura 6.19

6.13 DeviceDetailFragment

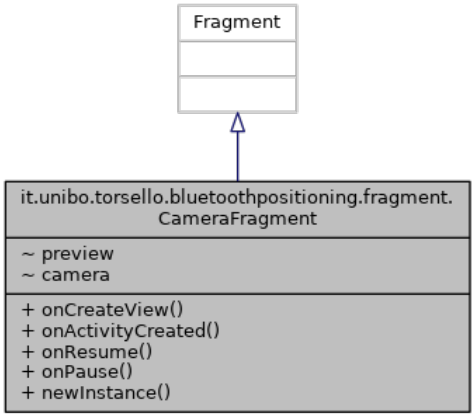

Figura 6.20

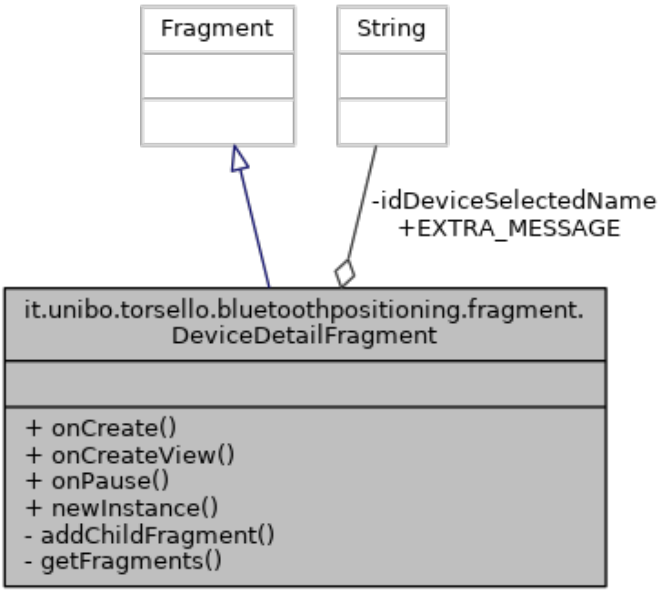

Figura 6.21

6.14 DeviceDetailInner2Fragment

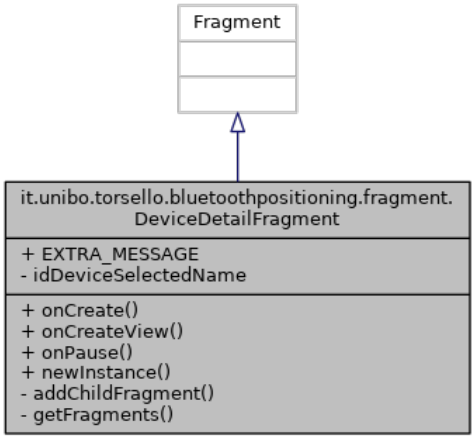

Figura 6.22

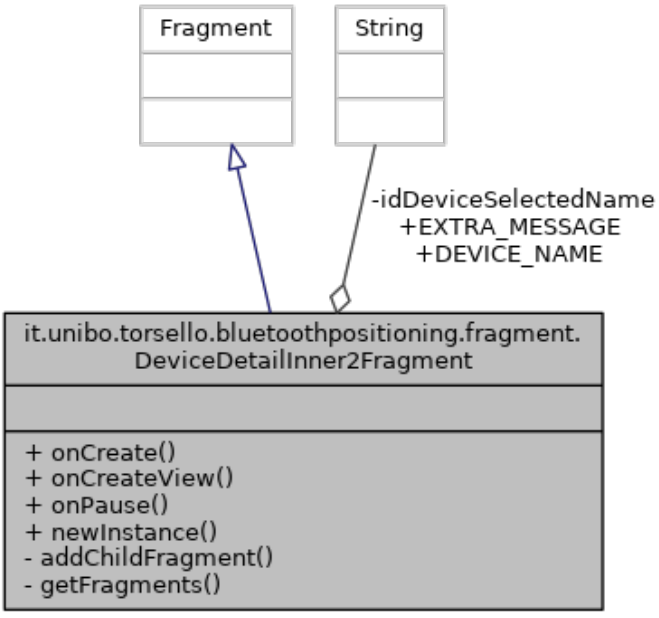

Figura 6.23

6.15 SettingsFragment

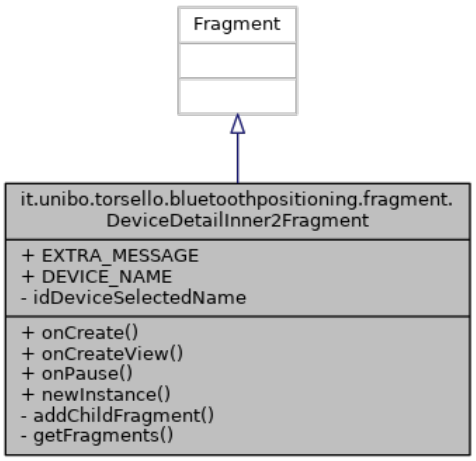

Figura 6.24

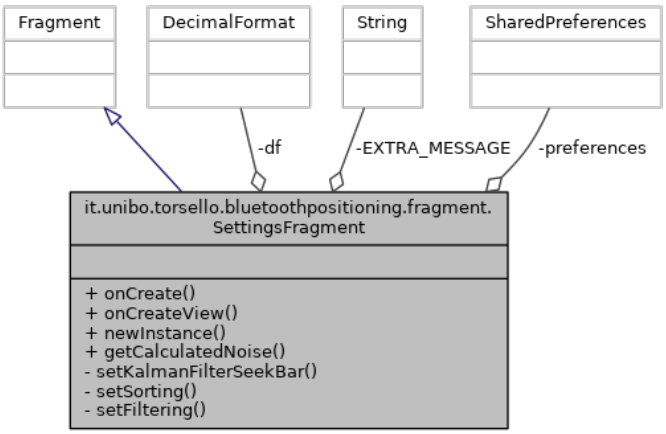

Figura 6.25

6.16 DeviceChartFragment

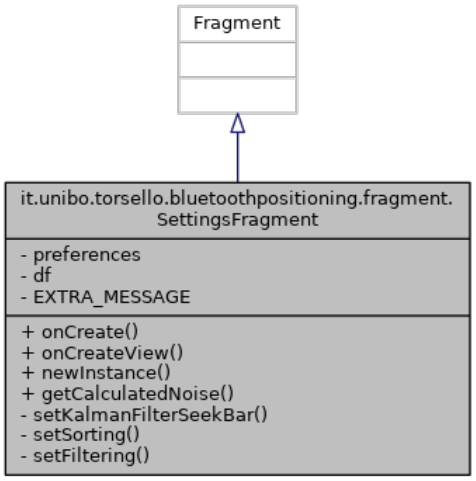

Figura 6.26

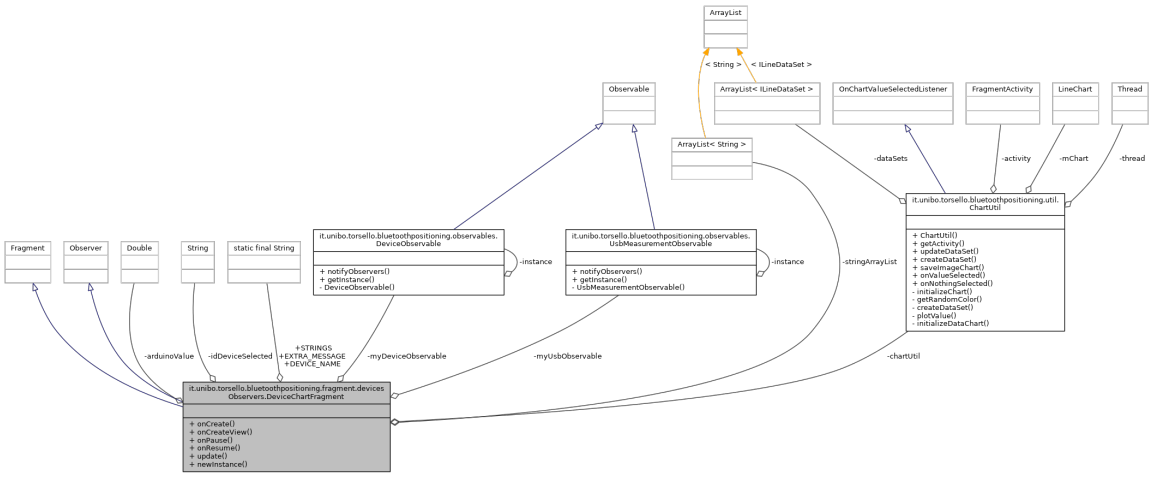

Figura 6.27

6.17 DeviceDetailInner1Fragment

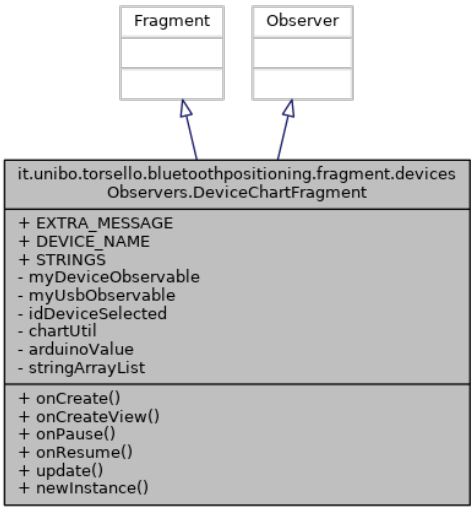

Figura 6.28

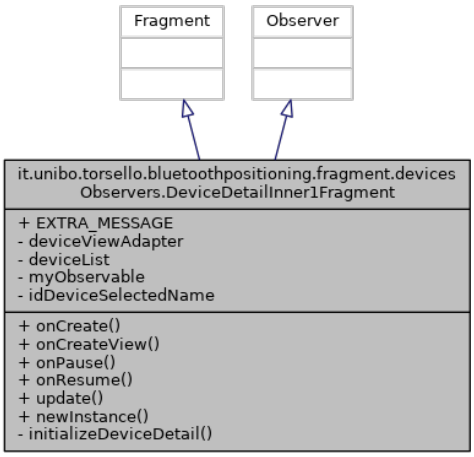

Figura 6.29

6.18 DeviceListFragment

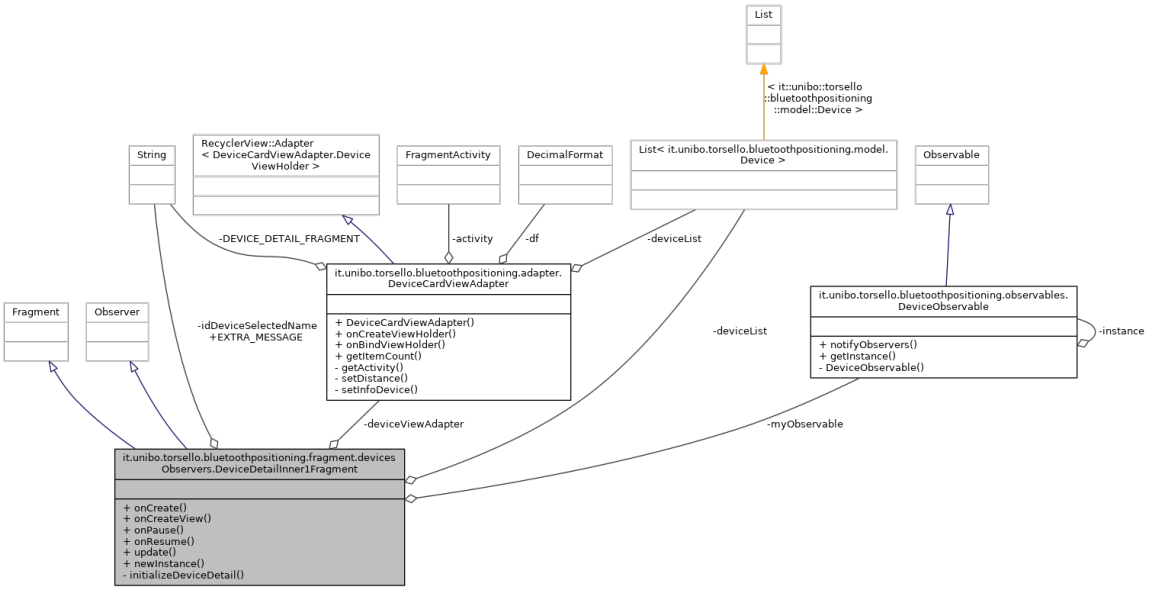

Figura 6.30

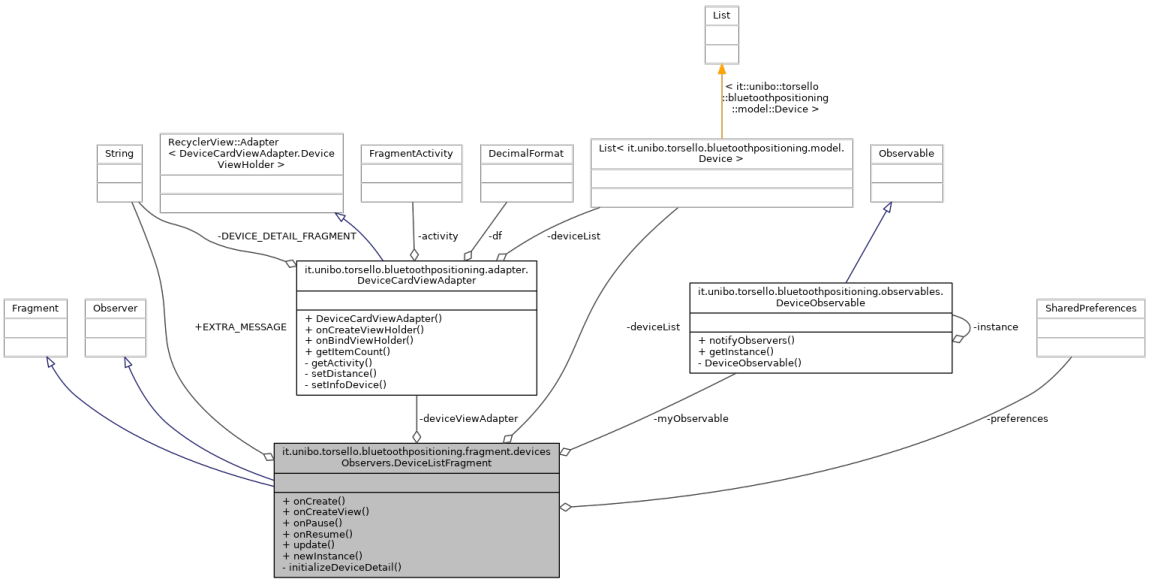

Figura 6.31

6.19 UsbMeasurementFragment

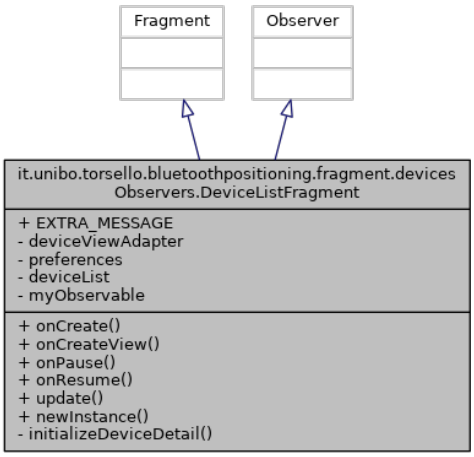

Figura 6.32

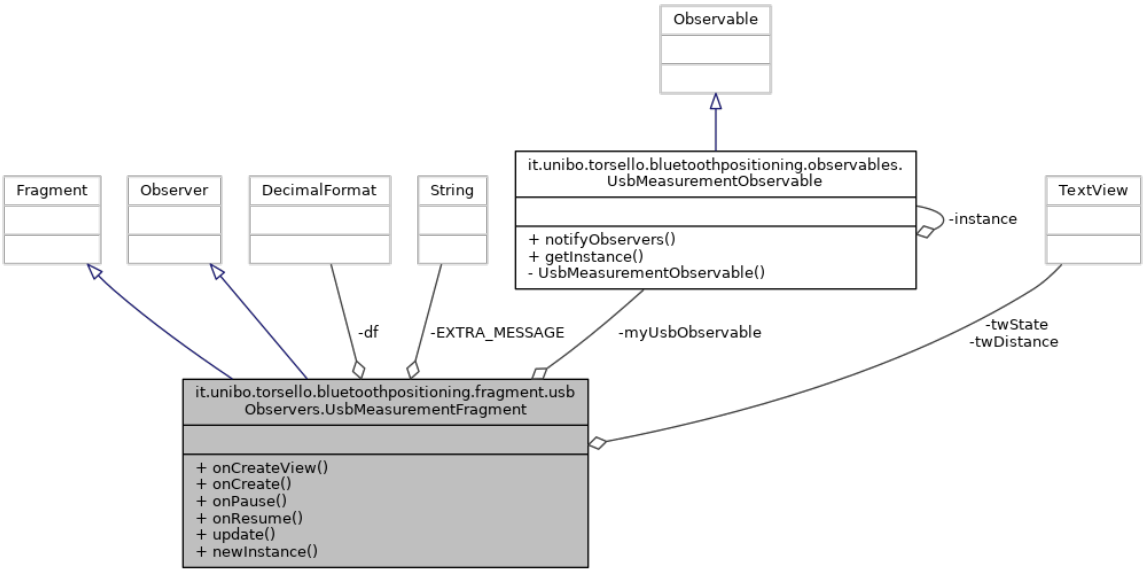

Figura 6.33

6.20 KFilterBuildertFragment

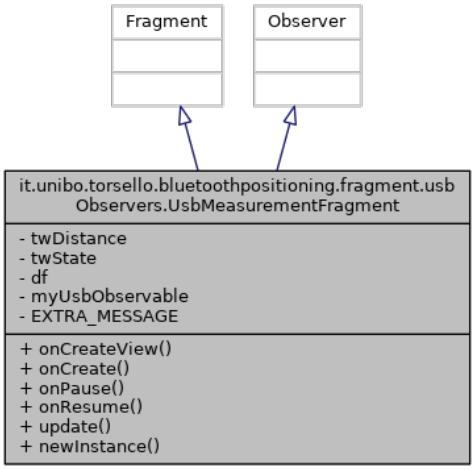

Figura 6.34

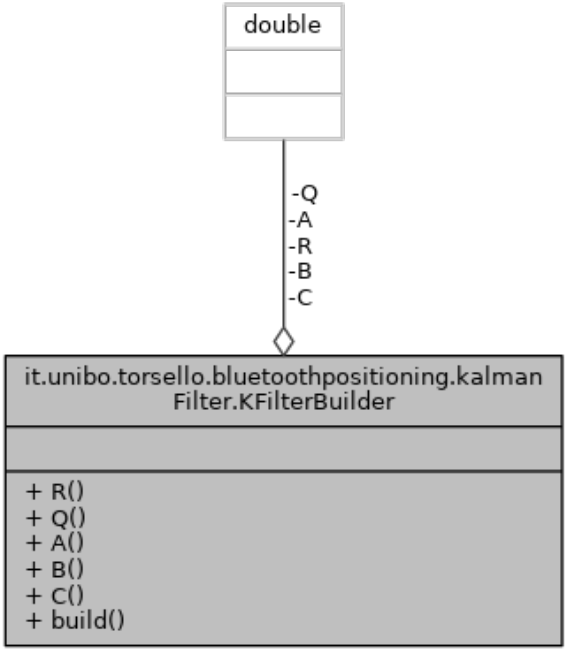

Figura 6.35

6.21 KFilter

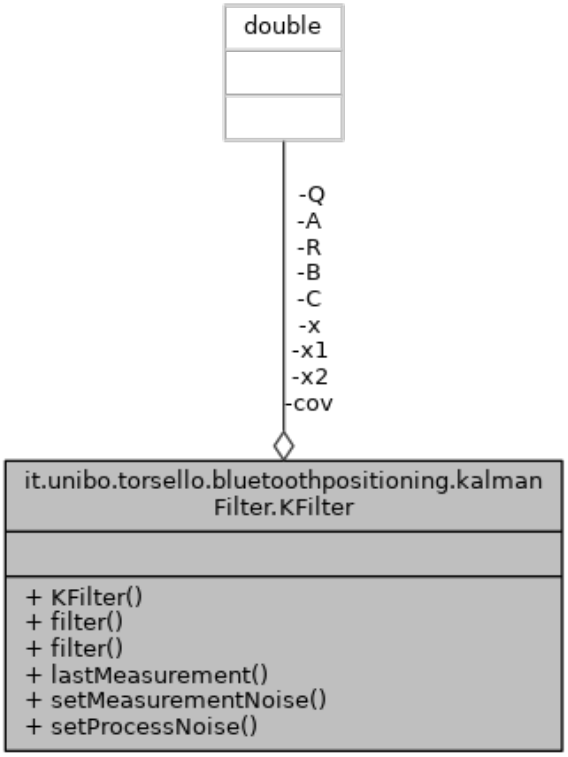

Figura 6.36

6.22 Device

6.23 DeviceObservable

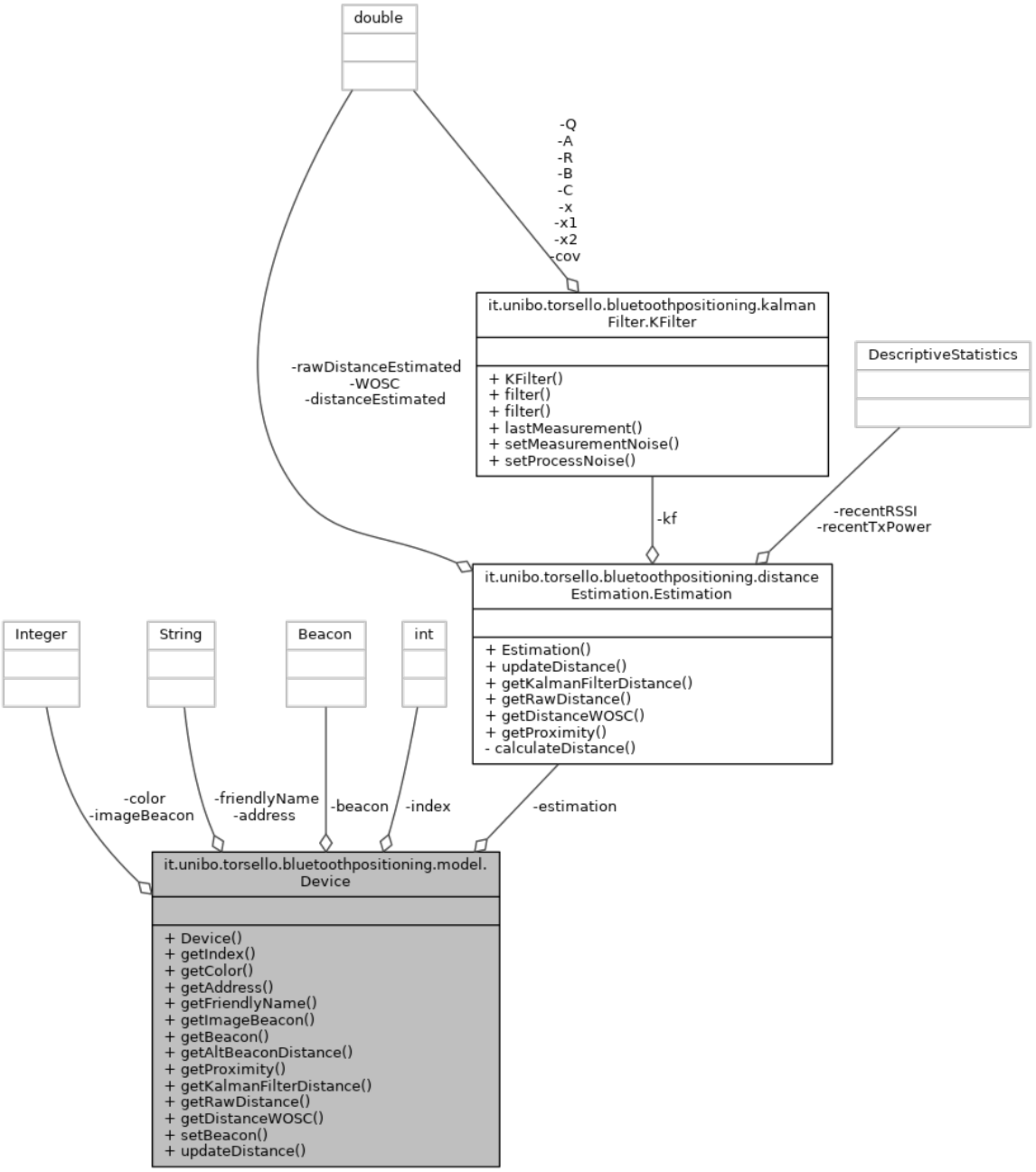
double
-Q
-A
-R
-B
-C
-x
-x1
-x2
-cov
it.unibo.torsello.bluetoothpositioning.kalman Filter.KFilter
+ KFilter()
+ filter()
+ filter()
+ lastMeasurement()
+ setMeasurementNoise()
+ setProcessNoise()
DescriptiveStatistics
-rawDistanceEstimated
-WOSC
-distanceEstimated
-kf
-recentRSSI
-recentTxPower
it.unibo.torsello.bluetoothpositioning.distance Estimation.Estimation
+ Estimation()
+ updateDistance()
+ getKalmanFilterDistance()
+ getRawDistance()
+ getDistanceWOSC()
+ getProximity()
- calculateDistance()
Integer
String
Beacon
int
-color
-imageBeacon
-friendlyName
-address
-beacon
-index
-estimation
it.unibo.torsello.bluetoothpositioning.model. Device
+ Device()
+ getIndex()
+ getColor()
+ getAddress()
+ getFriendlyName()
+ getImageBeacon()
+ getBeacon()
+ getAltBeaconDistance()
+ getProximity()
+ getKalmanFilterDistance()
+ getRawDistance()
+ getDistanceWOSC()
+ setBeacon()
+ updateDistance()

Figura 6.37

6.24 UsbMeasurementObservable

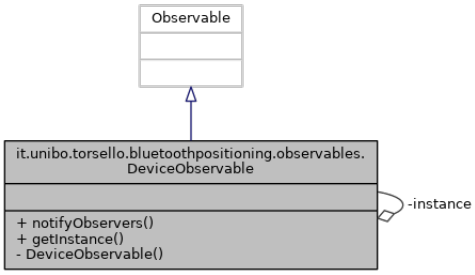

Figura 6.38

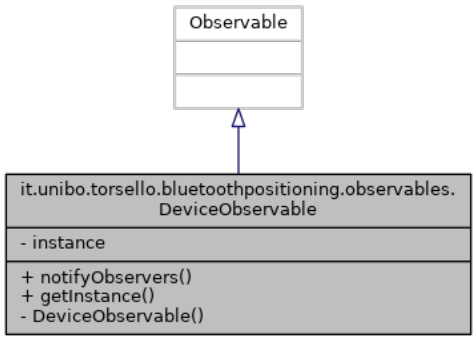

Figura 6.39

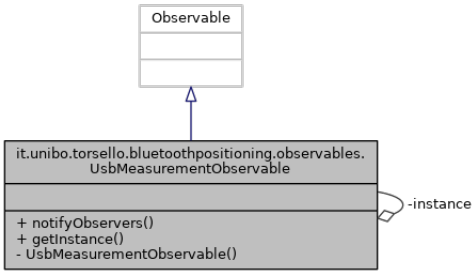

Figura 6.40

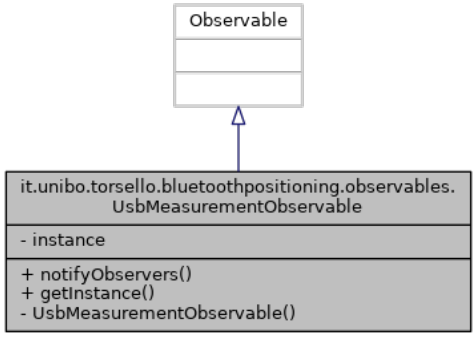

Figura 6.41

6.25 SaveImageTask

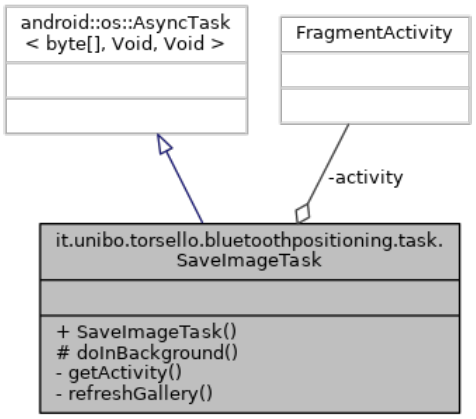

Figura 6.42

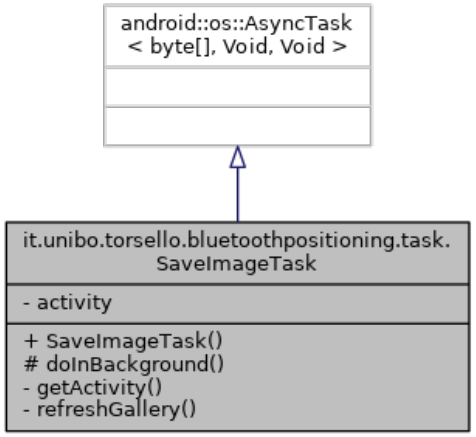

Figura 6.43

6.26 CameraPreviewUtil

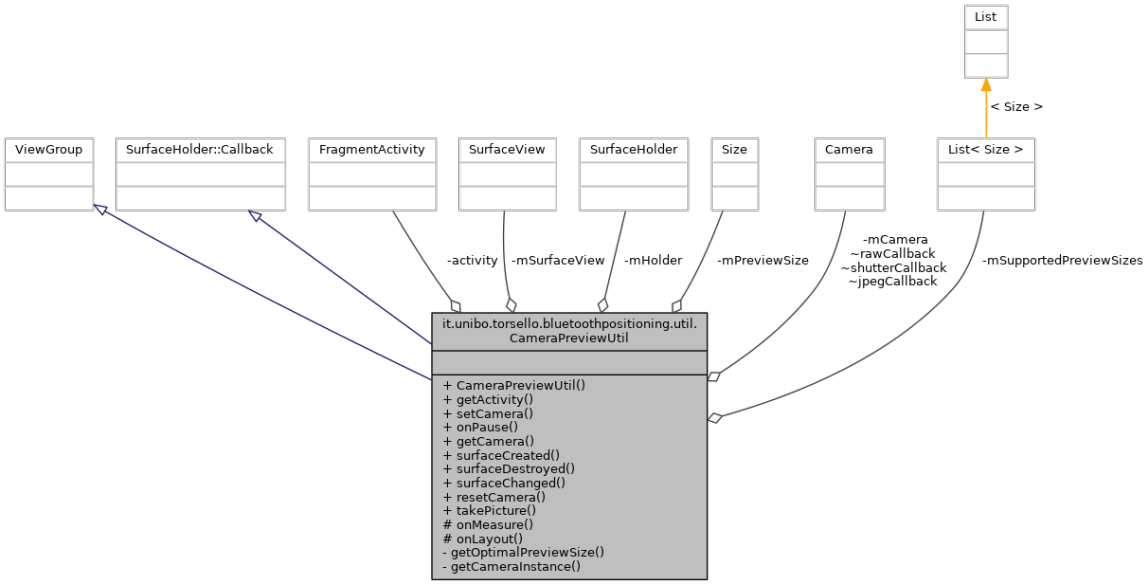

Figura 6.44

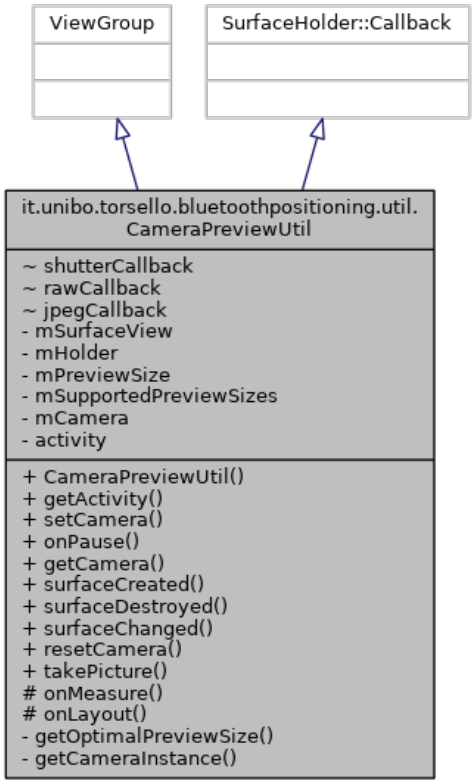
ViewGroup
SurfaceHolder::Callback
it.unibo.torsello.bluetoothpositioning.util.
CameraPreviewUtil
~ shutterCallback
~ rawCallback
~ jpegCallback
- mSurfaceView
- mHolder
- mPreviewSize
- mSupportedPreviewSizes
- mCamera
- activity
+ CameraPreviewUtil()
+ getActivity()
+ setCamera()
+ onPause()
+ getCamera()
+ surfaceCreated()
+ surfaceDestroyed()
+ surfaceChanged()
+ resetCamera()
+ takePicture()
# onMeasure()
# onLayout()
- getOptimalPreviewSize()
- getCameraInstance()

Figura 6.45

6.27 ChartUtil

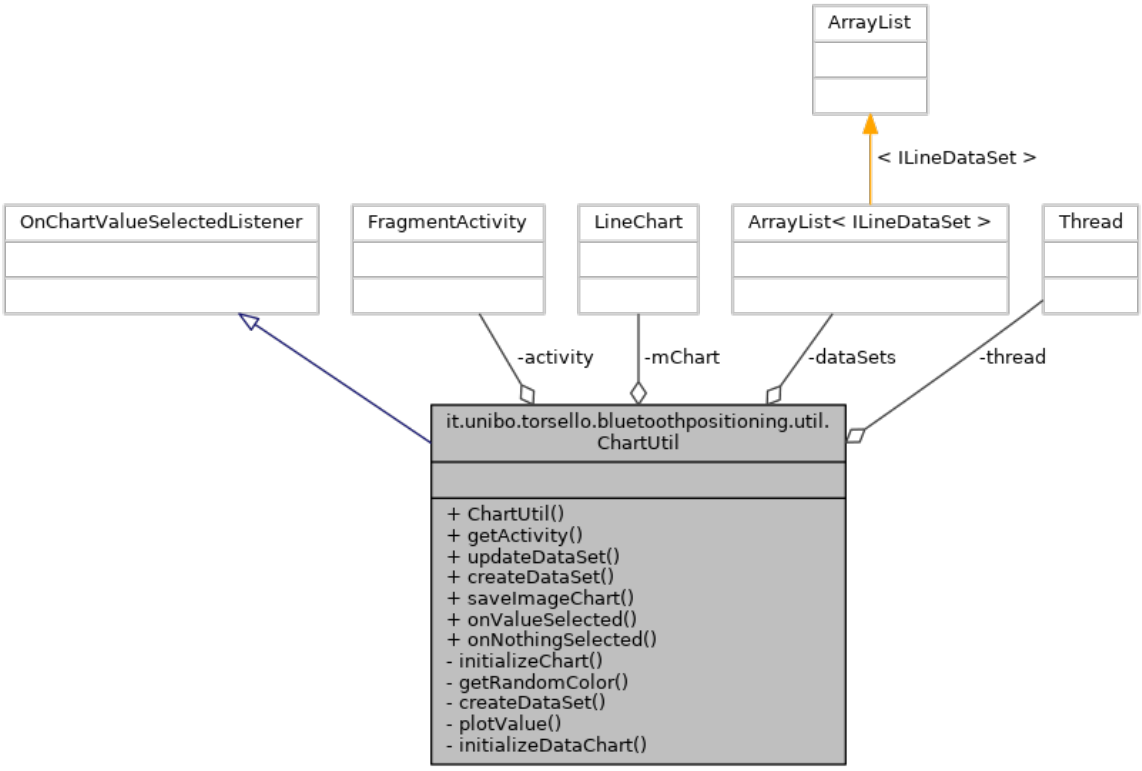

Figura 6.46

6.28 UsbUtil

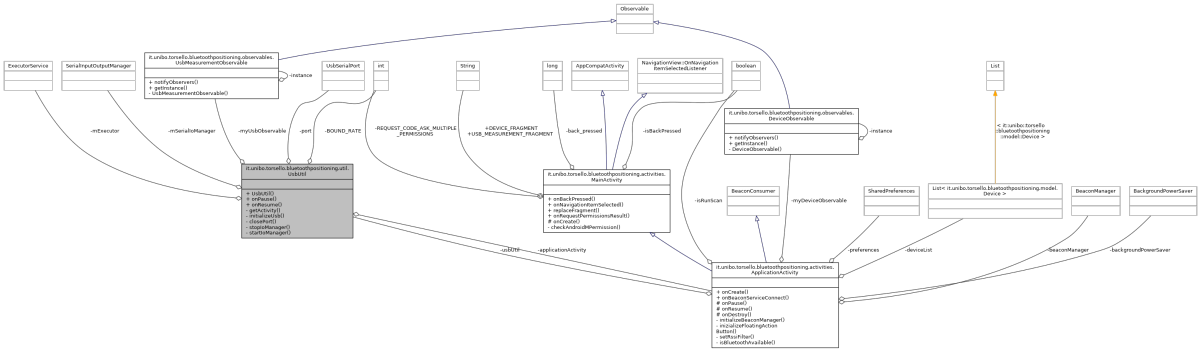

Figura 6.47

Capitolo 7

Implementazione

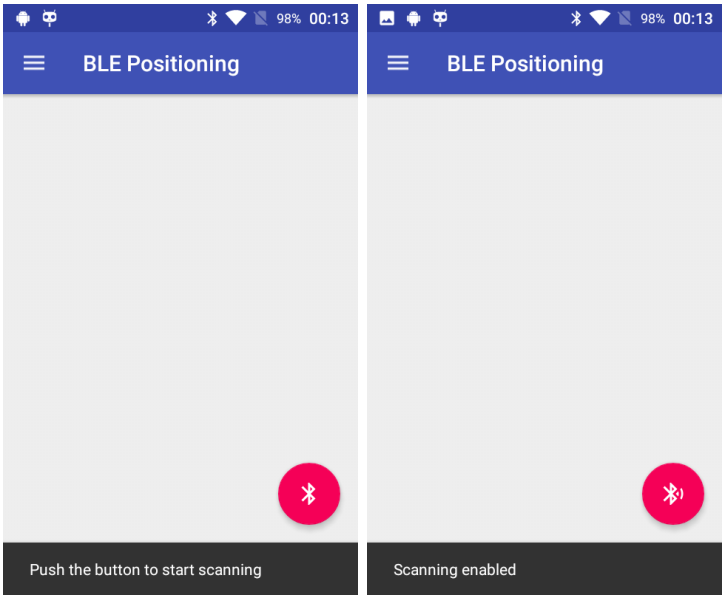

Figura 7.1

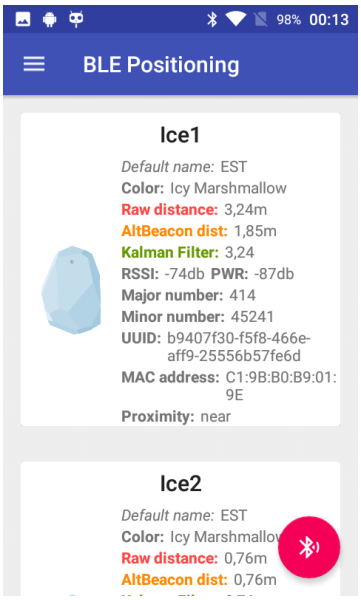

Figura 7.2

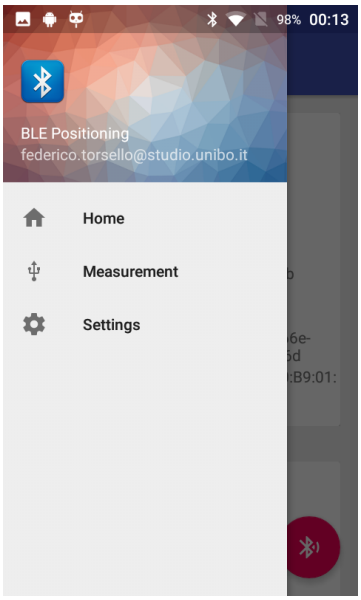

Figura 7.3

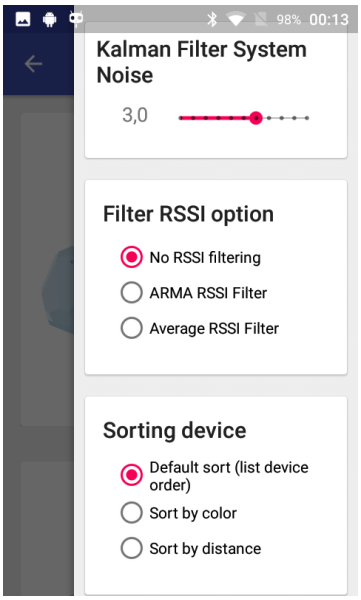

Figura 7.4

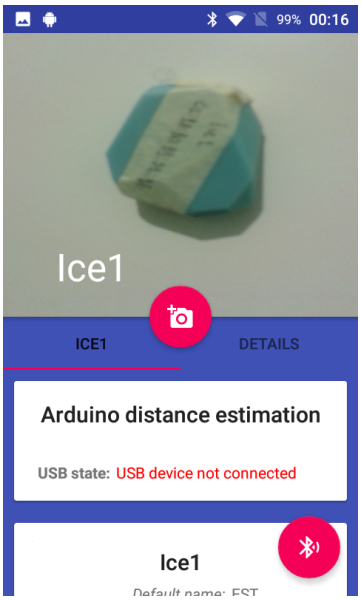

Figura 7.5

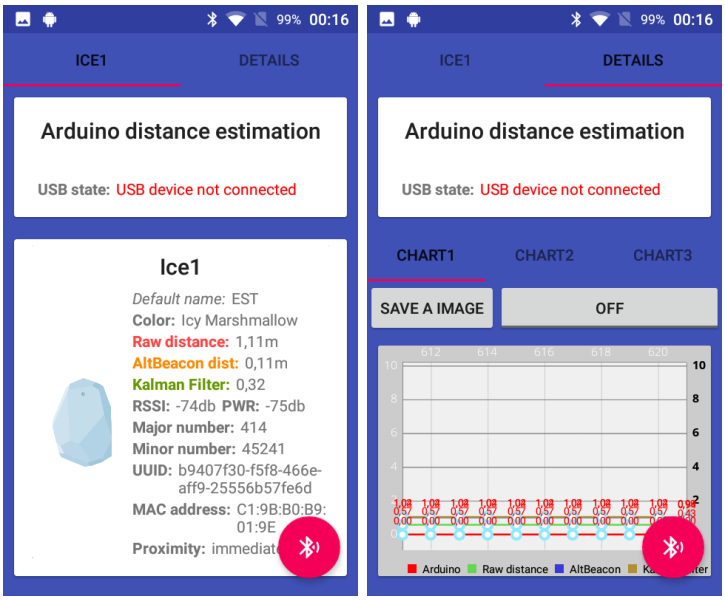

Figura 7.6

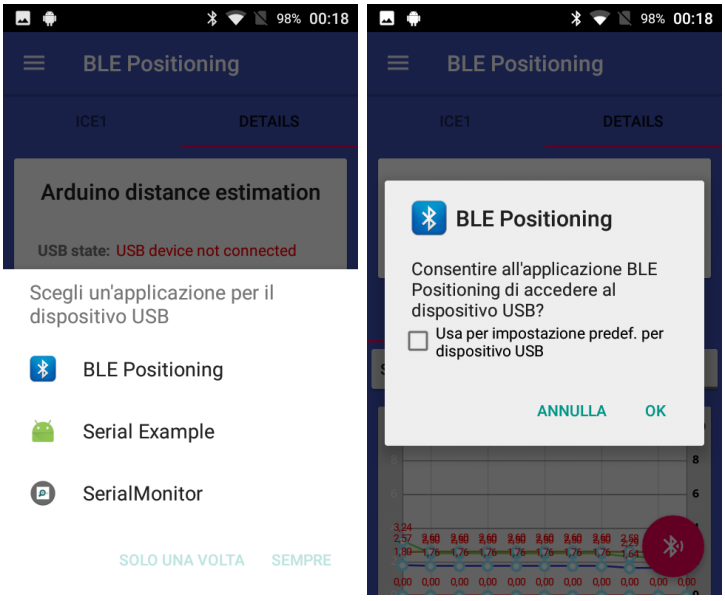

Figura 7.7

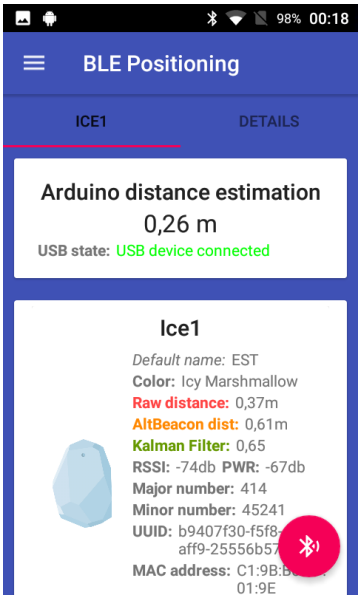
98% 00:18
BLE Positioning
ICE1
DETAILS
Arduino distance estimation
0,26 m
USB state: USB device connected
Ice1
Default name: EST
Color: Icy Marshmallow
Raw distance: 0,37m
AltBeacon dist: 0,61m
Kalman Filter: 0,65
RSSI: -74db PWR: -67db
Major number: 414
Minor number: 45241
UUID: b9407f30-f5f8-
aff9-25556b57
MAC address: C1:9B:
01:9E

Figura 7.8

Capitolo 8

Testing

8.0.1 Indicazioni preliminari

Nello specifico, per non alterare la ricezione dei segnali inviati dagli iBeacon, questi devono essere disposti:

- su pareti regolari a circa 1 metro da terra;
- lontano da apparecchi che emettono onde elettromagnetiche (ad esempio router wifi);
- al sicuro da raffiche di vento;
- in aree in cui non si frappongano oggetti o persone tra lo smartphone e l'iBeacon target.