ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Scienze e Tecnologie Informatiche

# PROTOTIPO DI SCHERMO ADATTATIVO BASATO SU ANDROID E BLUETOOTH LOW ENERGY

*Relazione finale in*
SISTEMI OPERATIVI

*Relatore*
Prof. MIRKO VIROLI

*Presentata da*
FRANCESCO PARI

*Correlatore*
Ing. ANGELO CROATTI

Sessione I
Anno Accademico 2014/2015

# PAROLE CHIAVE

Bluetooth Low Energy

Mobile computing

Android

Schermi adattativi

Internet of Things

*Alla mia famiglia,*
*ai miei amici*
*e a quanti mi sono stati vicino*
*durante questi anni*

# Indice

# Introduzione

Al giorno d'oggi si assiste ad una vera e propria rivoluzione nel campo dell'Information and Communication Technology. La capacità computazionale infatti non è più soltanto una prerogativa dei "tradizionali" computer ma, a partire dall'avvento degli smartphone, è stata aggiunta ad altre tipologie di dispositivi, il cui primo scopo non è quello di fungere da elaboratori elettronici (per esempio, a un telefono sarebbe richiesto solo di effettuare chiamate e gestire messaggi di testo). Quindi un oggetto con delle funzionalità ben definite è stato dotato di capacità computazionale, allo scopo di migliorare le operazioni già effettuabili e di aggiungerne di nuove.

Due sono le conseguenze, tra loro strettamente correlate. Da un lato, **il mondo ICT si è sempre più spostato verso la mobilità**, favorendo tutte le tecnologie che fanno del funzionamento in contesti di mobile computing il loro punto di forza. Dall'altro, si è sempre più cercato di **dotare gli oggetti della realtà quotidiana, da cui le persone sono circondate, di capacità di elaborare informazioni**, estendendo quindi l'informatica a un sempre maggior numero di elementi della vita di tutti i giorni e rendendola a mano a mano più *pervasiva della realtà*. Questo è il principio fondamentale del nuovo paradigma dell'**Internet of Things**.

Ma, proprio come indica tale nome, la vera utilità e innovazione è che queste "cose" della realtà quotidiana possano essere **interconnesse tra di loro** per creare una "rete". Ecco allora avvenire, dal punto di vista delle tecnologie, sia la riscoperta di alcune già esistenti, ma ritenute di poco interesse o addirittura già date per finite, sia l'introduzione e lo sviluppo di nuove: in ambedue i casi lo scopo è realizzare connettività in ambito mobile tra oggetti delle più svariate tipologie. E, soprattutto, l'aspetto veramente interessante è la possibilità di **integrare queste tecnologie**, di utilizzarle in maniera combinata

per ottenere risultati sempre migliori.

Da queste premesse prende avvio il presente lavoro di tesi. Esso nasce da un'iniziale attività di ricerca approfondita su alcune di queste tecnologie e sulla possibilità di integrarle.

La prima e più importante tra di esse è rappresentata dai **beacon**, dispositivi di ridotte dimensioni utilizzati per la localizzazione. Il loro funzionamento si basa sull'invio di un segnale **Bluetooth Low Energy**, ovvero una nuova specifica di Bluetooth, in rapidissima diffusione, caratterizzata principalmente da una grande riduzione del consumo di energia.

Tuttavia al giorno d'oggi non si può evitare di pensare a come queste queste innovazioni possano interfacciarsi con il mondo degli smartphone e dei tablet, che sta sempre più prendendo il posto dei tradizionali PC. Ecco allora che è stata analizzata la possibilità di interazione con **Android**: esso infatti, oltre ad essere il sistema operativo di gran lunga più diffuso in ambito mobile, è completamente open source, cosa che ha permesso la creazione di una grandissima community di sviluppatori attorno ad esso. Pertanto tale piattaforma viene sempre più frequentemente assunta come riferimento per la costruzione di sistemi basati su dispositivi mobili.

Come conseguenza di questa ricerca, è stato analizzato un caso di studio che permettesse di applicare concretamente i concetti studiati e l'interazione tra di essi, e di valutare quali siano effettivamente le loro attuali potenzialità. Si è quindi pensato a un piccolo **sistema distribuito** per **schermi adattativi**, ovvero capaci di modificare il loro contenuto in relazione a eventi esterni ad essi (concetto rientrante nel già citato paradigma dell'Internet of Things). Partendo da questa idea, si è quindi proceduto alla progettazione e allo sviluppo di un prototipo, servendosi delle tecnologie oggetto di ricerca. Infine si è raggiunto l'obiettivo di verificare le attuali potenzialità di queste innovazioni, e di trarre alcune conclusioni sulle possibilità della loro futura diffusione e di un impiego in contesti differenti.

# Capitolo 1

# Stato dell'arte

In questo primo capitolo si vogliono inquadrare le tecnologie e i paradigmi sui quali, come già spiegato nell'Introduzione, è stata svolta un'iniziale attività di ricerca approfondita. Per ciascuno di essi verrà fornita una descrizione, illustrando in cosa esso consiste e quali sono i suoi meccanismi fondamentali di funzionamento. Inoltre, trattandosi di uno degli obiettivi principali della tesi, sarà sempre tenuta in considerazione la possibilità di interazione con altre tecnologie e dispositivi, soprattutto con il mondo Android.

## 1.1 Bluetooth Low Energy

**Bluetooth Low Energy** (**BLE**), noto anche con il nome commerciale **Bluetooth Smart**, è stato introdotto dal Bluetooth Special Interest Group (SIG)[1] a giugno del 2010 come versione 4.0 della nota tecnologia Bluetooth, e successivamente migliorato nella versione 4.1. Originariamente creato da Nokia nel 2006 con il nome di Wibree, prima che diventasse disponibile sul mercato la compagnia finlandese decise di consegnarlo al BSIG perché lo standardizzasse.

Si tratta di una specifica per la comunicazione wireless su brevi distanze (il segnale può giungere fino a circa 50-60 m in assenza di ostacoli fisici; tuttavia in presenza di questi ultimi tale valore può notevolmente diminuire). Essa mantiene le caratteristiche delle precedenti versioni di Bluetooth, ma è contraddistinta principalmente, come indica il nome, da un **consumo di energia**

[1] Sito ufficiale del Bluetooth Special Interest Group: `https://www.bluetooth.org`

**notevolmente ridotto** (è stato verificato che un device BLE alimentato da una batteria a bottone può durare fino a 3 anni). La motivazione di questa nuova specifica va ricercata nel voler espandere le possibilità di utilizzo del Bluetooth, impiegandolo in device a disponibilità energetica limitata, come per esempio sensori wireless. Questi dispositivi non solo richiedono bassi consumi di energia, ma la quantità di dati trasmessi è relativamente piccola e poco frequente rispetto alle applicazioni Bluetooth tradizionali, come lo streaming audio.

### 1.1.1 Compatibilità tra dispositivi Bluetooth

Bluetooth Smart utilizza le medesime frequenze del Bluetooth classico (2.4 GHz) e, cosa fondamentale, è retrocompatibile con quest'ultimo, e può quindi essere implementato in congiunzione con esso, usando la stessa antenna. Questo però non avviene sempre; a tal proposito, esistono tre categorie per classificare ciascun dispositivo dotato di chip Bluetooth, che quindi deve appartenere a *una e una sola* di esse:

1. **Bluetooth**: compatibile soltanto con il Bluetooth classico.
2. **Bluetooth Smart Ready**: compatibile sia con il Bluetooth classico che con il Bluetooth Low Energy.
3. **Bluetooth Smart**: compatibile soltanto con il Bluetooth Low Energy.

In Figura 1.1 è riportata in dettaglio, per ognuna delle tre categorie, la compatibilità con le altre.

### 1.1.2 Ruoli

Oltre al consumo energetico, vi è un'altra differenza molto importante tra le due versioni: mentre quella classica è caratterizzata da un'architettura di tipo peer-to-peer, nel BLE vi è una **mancanza di simmetria tra dispositivi pari**. Essendo strettamente ottimizzati per bassi consumi di energia, tali device possono supportare, in una singola connessione, soltanto comunicazioni unidirezionali (solo trasmissione o solo ricezione). In altre parole, non sono

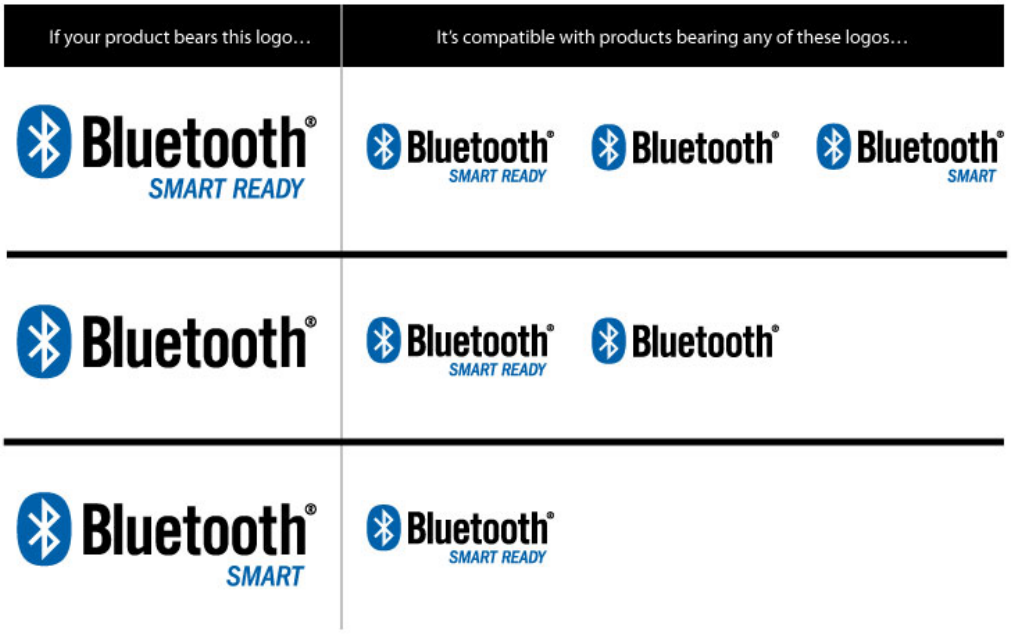



Figura 1.1: Compatibilità tra dispositivi Bluetooth

possibili scambi tra i ruoli master e slave all'interno di una connessione. Questo significa che deve essere chiaro già all'instaurazione del collegamento chi ricoprirà il ruolo di master e chi di slave.

Un'altra differenza, notevole dal punto di vista concettuale, si riscontra nella ricerca dei device. Nel Bluetooth Low Energy, i dispositivi che vogliono essere "trovati" da altri inviano advertisement in broadcast, che verranno ricevuti da coloro che si sono messi in ascolto di questi segnali. Nelle versioni precedenti alla 4.0, la filosofia è opposta: i device che vogliono essere "trovati" si mettono loro stessi in ascolto di quelli che li cercano (e che, per "trovarli", inviano advertisement)!

Si è perciò reso necessario stabilire **quattro ruoli** specifici che un dispositivo può ricoprire, e che rappresentano le possibili combinazioni delle capacità sopracitate:

1. **Broadcaster**: perché un dispositivo possa ricoprire questo ruolo è richiesto soltanto che disponga di un trasmettitore. Esso infatti invia advertisement in broadcast ad altri device, senza tuttavia avere la pos-

sibilità di riceverne. Un esempio può essere un sensore di temperatura, che deve soltanto trasmettere i dati rilevati.

2. **Observer**: è il ruolo complementare al precedente: ad un dispositivo che vuole ricoprire tale ruolo è solo chiesto di avere un ricevitore. La sua attività è infatti mettersi in continuo ascolto degli advertisement, senza poterne inviare. Rimanendo nel contesto dell'esempio precedente, il ruolo di observer può essere ricoperto da un display, che deve solamente ricevere i dati sulla temperatura e visualizzarli sullo schermo.

3. **Peripheral**: in questo caso invece il dispositivo deve possedere sia il trasmettitore che il ricevitore. Infatti bisogna che chi ricopre questo ruolo supporti l'invio di advertisement in broadcast, e abbia la capacità di connettersi con altri device, facendo da slave (ovvero "ricevendo" la connessione). Un semplice esempio può essere una stampante.

4. **Central**: anche per quest'ultimo ruolo è necessario che il dispositivo abbia sia il trasmettitore che il ricevitore. Esso deve supportare l'ascolto di advertisement in broadcast, e essere in grado di iniziare una connessione, facendo da master (ossia "richiedendo" la connessione). Un esempio può essere rappresentato da uno smartphone o da un computer portatile).

È però fondamentale notare come **un dispositivo possa supportare più ruoli** (ovviamente non nella stessa connessione!).

### 1.1.3 Compatibilità con Android

Si vuole ora considerare brevemente l'integrazione tra questa tecnologia e il mondo Android. Innanzitutto, è necessario che il modello di dispositivo Android considerato sia compatibile con il Bluetooth Smart. Soltanto per tali device ha senso considerare i due punti seguenti.

- Android ha introdotto il supporto per il **central role** a partire dalla **versione 4.3** (API Level 18) aggiungendo nuove classi al già esistente

package `android.bluetooth`, fornendo API che le app possono usare per cercare device, richiedere servizi, e leggere/scrivere caratteristiche[2].

- Il supporto per il **peripheral role** è stato invece introdotto solo con l'arrivo di Lollipop (**Android 5.0**, API Level 21), aggiungendo un nuovo package `android.bluetooth.le`[3]. Tuttavia la disponibilità del peripheral mode dipende anche da alcuni fattori legati al chipset del device. Per questo motivo sono ancora pochi i dispositivi che supportano questa modalità: per citare un caso importante, Google ha dovuto rimuovere il supporto al Nexus 5 (dopo averlo inizialmente fornito), tanto che i primi modelli di tale produzione a supportare effettivamente il peripheral mode sono il Nexus 6 e il Nexus 9[4].

## 1.2 Beacon

Un **beacon** (termine inglese che significa *faro*) è un qualunque dispositivo che trasmette un segnale Bluetooth Low Energy per permettere a un altro device di determinarne la prossimità. Il termine **iBeacon**, spesso usato come sinonimo, indica invece la versione Apple di questa tecnologia.

Un beacon è quindi un dispositivo, solitamente di dimensioni molto ridotte, contenente un microcontroller con un radio chip Bluetooth Smart e nella maggioranza dei casi una batteria (tuttavia esistono anche modelli alimentati tramite USB). Questo device non fa altro che **inviare un segnale BLE molto frequentemente** (tipicamente, l'intervallo tra due invii consecutivi può andare da 0.1 a 10 s) permettendo a chi è nelle vicinanze di "accorgersi" della sua presenza, di stimarne la distanza, e di compiere determinate azioni di conseguenza. I beacon sono utilizzati sia per la localizzazione in edifici chiusi (in quanto in ambiente indoor la precisione del GPS cala notevolmente) sia in spazi esterni; questa diffusione deriva soprattutto dal basso consumo di energia

[2]Per maggiori informazioni, si rimanda alla documentazione ufficiale: `https://developer.android.com/guide/topics/connectivity/bluetooth-le.html`.

[3]Si rimanda alla sezione *Bluetooth Low Energy* del seguente link: `https://developer.android.com/about/versions/android-5.0.html#Wireless`.

[4]`https://code.google.com/p/android-developer-preview/issues/detail?id=1570#c52`

(in uno scenario realistico, il consumo di batteria di un telefono causato dalla vicinanza di uno o più beacon dovrebbe essere inferiore all'1%).

(a) Estimote (b) Kontakt (c) StickNFind

Figura 1.2: Alcuni beacon di produttori diversi

Questa tecnologia sta riscontrando molto successo, e si sta diffondendo in maniera crescente in ambienti come centri commerciali, musei, negozi... Il motivo principale è perché si basa su una tecnologia molto popolare come Bluetooth, disponibile ormai su un grandissimo numero di dispositivi. Inoltre il consumo assai ridotto di energia permette sia ai beacon trasmittenti che ai device riceventi di lavorare a piena potenza.

### 1.2.1 Fascio di informazioni

Con iBeacon, Apple ha standardizzato il formato del fascio di informazioni del segnale BLE. Un pacchetto di advertising è formato da **quattro sezioni**:

1. **UUID** (Universally Unique IDentifier): stringa a 128 bit usata per differenziare un largo insieme di beacon (generalmente, tutti quelli appartenenti a un determinato contesto o applicazione).

2. **Major**: stringa a 16 bit usata per distinguere un piccolo sottoinsieme di beacon all'interno dell'insieme.

3. **Minor**: stringa a 16 bit usata per poter distinguere ulteriormente i beacon all'interno del sottoinsieme.

4. **Tx Power**: dato a 8 bit usato per determinare la distanza dal beacon, è definito come la potenza del segnale a 1 m di distanza dal device.

Come si può vedere, questa tecnologia non nasce con l'intento di trasportare della vera e propria "informazione", ma soltanto dei **valori utili unicamente**

**per la localizzazione**. Tuttavia, questo dipende da come i beacon sono realizzati e da cosa offrono da un punto di vista hardware. Infatti, mentre vi sono modelli economici che trasmettono soltanto il protocollo beacon di base, altri possono fornire anche del contenuto informativo, a cui addirittura potrebbero accedere i dispositivi Bluetooth tradizionali. Moltissimi, ad esempio, forniscono una stima del consumo attuale della batteria. Altri possono avere motion control, sensore termico, attivazione di luci e/o suoni, ecc...

Ciascuno di questi parametri è cablato nel firmware di ogni dispositivo e, per la maggior parte dei modelli, può essere impostato a piacere. Oltre ad essi, possono essere modificati l'intervallo di advertising e la potenza del segnale (entrambi sono parametri che hanno un effetto notevole sulla durata della batteria).

### 1.2.2 Precisione della stima della distanza

A parte i valori di UUID, Major e Minor, la cui trasmissione non presenta errori, attualmente la ricezione del segnale non è precisa. **La stima della distanza tra trasmittente e ricevente è spesso instabile** a causa di tanti fattori, come ostacoli fisici e interferenze sullo stesso spettro di onde radio.

Viene allora spontaneo domandarsi per quale motivo i beacon vengano usati per la localizzazione. La risposta è che, **dopo uno studio attento dei luoghi più adatti dove posizionarli, e dopo aver opportunamente calibrato ciascuno di essi, si possono ottenere dei buoni risultati**. Più essi sono vicini l'uno all'altro, e più la potenza del segnale deve essere bassa, per evitare quanto possibile interferenze. È come trovarsi in una stanza con un po' di persone: qualora ciascuna di esse parlasse a voce molto alta, sarebbe impossibile capire cosa ognuno stia dicendo; ma se ogni persona parla a bassa voce, avvicinandosi alla giusta distanza ad una qualunque di esse è possibile comprendere le sue parole.

Ad ogni modo, per quanto riguarda la precisione, **vi è una grossa differenza tra i vari produttori**, i quali utilizzano differenti radio chip e algoritmi per stimare la distanza. Se si hanno a disposizione modelli fra i più precisi, si può fare uso di tecniche di triangolazione, che possono portare a un incremento della precisione ma anche dei costi, sia per l'hardware (occorrono un maggior numero di beacon per realizzare questi algoritmi), sia operativi (lo

studio della locazione migliore per ogni device deve essere più approfondito). Se al contrario i modelli di cui si dispone non sono molto precisi, ricorrere a tecniche di triangolazione non sembra essere un'ottima soluzione, in quanto l'errore da cui sono già affette le stime delle distanze dai singoli dispositivi verrebbe ulteriormente amplificato.

### 1.2.3 Interazione con Android: Android Beacon Library

I device Android di default non sono in grado di interpretare il fascio di informazioni di un beacon. Quest'ultimo infatti, facendo una normale scansione, viene rilevato alla pari di qualunque altro dispositivo Bluetooth Smart: sono dunque individuate le informazioni tipiche di ogni device BLE (nome, MAC address...), ma non quelle specifiche dei beacon (UUID, Major, Minor...).

Per ovviare a questo problema si può ricorrere a una libreria esterna: una delle più note e utilizzate è la **Android Beacon Library**[5]. Di default, essa può interpretare soltanto il fascio di informazioni dei beacon conformi allo standard aperto AltBeacon[6]; tuttavia può essere configurata manualmente per interfacciarsi con qualunque altro modello. Essa presenta infatti una classe `BeaconParser` mediante la quale si può specificare come interpretare il fascio di informazioni contenuto nel segnale BLE di un beacon, a patto che si conosca la suddivisione dei bit del fascio in unità logiche di informazione (cosa non sempre possibile, in quanto alcuni produttori non usano specifiche aperte, e pertanto le funzionalità dei loro modelli sono fruibili appieno soltanto con l'app proprietaria).[7]

Queste API inoltre permettono ai dispositivi Android di utilizzare i beacon in maniera molto simile a come li utilizzano i device iOS, rilevando l'entrata o l'uscita del dispositivo dalla region di uno o più beacon, stimando la distanza tra questi ultimi, ecc... In tal modo è più semplice impostare delle azioni che

[5]Sito del progetto: `http://altbeacon.github.io/android-beacon-library`

[6]Sito del progetto: `http://altbeacon.org`

[7]Per maggiori informazioni, si rimanda alla documentazione ufficiale: `http://altbeacon.github.io/android-beacon-library/javadoc/org/altbeacon/beacon/BeaconParser.html`

dovranno essere eseguite all'entrata o uscita da una region, o a una determinata distanza.

## 1.3 Internet of Things

**Internet of Things** (**IoT**) è un nuovo paradigma in rapida diffusione basato sull'idea centrale, già accennata nell'Introduzione, di **fornire capacità computazionale all'ambiente che circonda le persone**. Le *cose*, gli *oggetti* della realtà quotidiana possono dunque essere **interconnesse tra loro**, mediante varie tecnologie (Wi-Fi, Bluetooth, NFC...) creando a tutti gli effetti una "rete delle cose". Dunque, una rete di telecomunicazioni non è più formata solo da quelli che erano i suoi componenti "tradizionali", ma si estende alle più svariate entità della vita quotidiana (occhiali, orologi, luci di una casa...). Gli oggetti da cui siamo circondati possono dunque interagire gli uni con gli altri e cooperare con i propri vicini per raggiungere obiettivi comuni.

Il grandissimo interesse verso questo paradigma è giustificato soprattutto dal fatto che esso **può riguardare ogni aspetto della vita quotidiana**, anche quelli più lontani dall'Information and Communication Technology. L'US National Intelligence Council prevede che entro il 2025 i nodi Internet potrebbero risiedere nelle cose di tutti i giorni, come confezioni di alimentari, mobili, fogli di carta, ecc... L'impatto sulla realtà quotidiana può essere potenzialmente gigantesco, dal momento che sempre più oggetti possono essere dotati di intelligenza artificiale e quindi agire autonomamente in relazione a eventi esterni ad essi.

# Capitolo 2

# Analisi del caso di studio

Dopo aver inquadrato ed esposto approfonditamente le tecnologie e i paradigmi che fanno da sfondo a tutto il presente lavoro di tesi, in questo secondo capitolo si vuole analizzare il caso di studio che ha permesso la loro applicazione. Oltre ad essere un'opportunità di impiegare concretamente gli argomenti su cui si è svolta l'attività di ricerca, esso è stato scelto anche per il fatto che potrebbe avere, in futuro, una sua utilità pratica.

## 2.1 Definizione ad alto livello del problema

Nei corridoi della sede universitaria sono dislocati alcuni schermi che mostrano, suddivise per ore e per aule, le attività didattiche del Corso di Laurea e Laurea Magistrale previste per quella giornata. Si è riflettuto su come si potesse "personalizzare" in qualche modo il contenuto informativo di questi monitor, e soprattutto su come automatizzare questa personalizzazione. La soluzione a cui si è giunti è che **questi schermi possano modificare dinamicamente ciò che viene visualizzato** in relazione alla vicinanza di un potenziale osservatore (e, soprattutto, di quale osservatore si tratta): cioè, si vuole rendere ognuno di essi uno **schermo adattativo**, ossia in grado di svolgere determinate azioni autonomamente, come risposta ad eventi esterni ad esso.

Si vuole quindi che, se nessuno studente del Corso di Laurea o Laurea Magistrale si trova nei pressi di un monitor (ad una distanza tale da permet-

terne la lettura), quest'ultimo non mostri, come attualmente accade, l'orario "generico" per quella giornata, bensì una schermata di default, che potrebbe semplicemente essere il logo dell'Ateneo, oppure avere un contenuto informativo utile per tutti gli studenti (per esempio avvisi, segnalazioni della segreteria didattica, la pubblicità di un particolare evento di quel periodo che potrebbe suscitare interesse, ecc...).

**Nel momento in cui uno studente arriva nelle vicinanze di uno schermo, esso deve invece mostrare lo specifico orario dell'anno di corso desiderato dall'osservatore** (quindi tipicamente quello a cui è iscritto). Questa visualizzazione deve essere mantenuta finché egli non si allontana di nuovo (caso in cui si ritorna alla schermata di default) o non modifica l'anno selezionato (qualora, per esempio, avesse commesso un errore nella scelta, o nel caso fosse interessato a seguire anche una o più lezioni di anni precedenti, o semplicemente perché ha iniziato a frequentare l'anno successivo). In questo secondo caso deve essere visualizzato l'orario corrispondente alla nuova selezione.

Attualmente infatti uno studente (che potrebbe essere di fretta, per esempio perché la lezione che deve seguire sta per iniziare) deve cercare tra tutte le attività didattiche della giornata - che sono solitamente molte - soltanto quelle di suo interesse. In questo modo invece potrebbe avere subito a disposizione il proprio orario, e solo per il tempo della sua permanenza nei pressi dello schermo.

## 2.2 Analisi dei requisiti

Dopo aver fornito una definizione iniziale, ad alto livello, della problematica considerata, si vuole ora approfondirne l'analisi, ponendosi le seguenti domande: più in dettaglio, "cosa deve fare" il sistema? O meglio, qual è il comportamento concreto e osservabile che da esso ci si attende? Questi interrogativi portano a riflettere sulle funzionalità che il sistema deve mettere a disposizione, ma in maniera indipendente dalle scelte progettuali e implementative che si potrebbero attuare.

Poiché uno schermo deve compiere determinate azioni a causa della vicinanza di uno studente, si possono suddividere i requisiti in due gruppi: ciò

che è richiesto sia fatto *da uno studente* e ciò che è richiesto sia fatto *da uno schermo*.

Per quanto riguarda quest'ultimo, bisogna inoltre notare che attualmente i monitor sono tutti collegati a un unico computer del quale visualizzano i contenuti. È quindi necessario innanzitutto che **ciascuno di essi sia reso autonomo, cioè capace di mostrare un proprio contenuto indipendentemente dagli altri**.

### 2.2.1 Requisiti per lo studente

Un generico studente deve essere in grado di compiere le seguenti attività:

- "Registrarsi" se desidera usufruire del servizio, ovvero se vuole che il monitor mostri l'orario di suo interesse quando egli si trova nelle vicinanze. Questa attività include necessariamente la comunicazione iniziale dell'anno di cui vuole che sia visualizzato l'orario.
- Modificare la scelta effettuata relativamente all'anno.
- Annullare la sua "registrazione", se non desidera più usufruire del servizio.

### 2.2.2 Requisiti per lo schermo

Un generico schermo deve essere in grado di compiere le seguenti attività:

- Mostrare la schermata di default se non vi è nessuno studente nelle vicinanze.
- Se uno studente arriva nelle vicinanze, visualizzare l'orario dell'anno di corso che egli ha selezionato.
- Se uno studente, che già si trova nelle vicinanze, modifica la scelta relativa all'anno, visualizzare l'orario corrispondente alla nuova selezione.
- Se uno studente, che già si trova nelle vicinanze, si allontana, ripristinare la visualizzazione della schermata di default.

## 2.3 Casi d'uso

In Figura 2.1 si può vedere una rappresentazione dei possibili casi d'uso conforme allo standard UML.

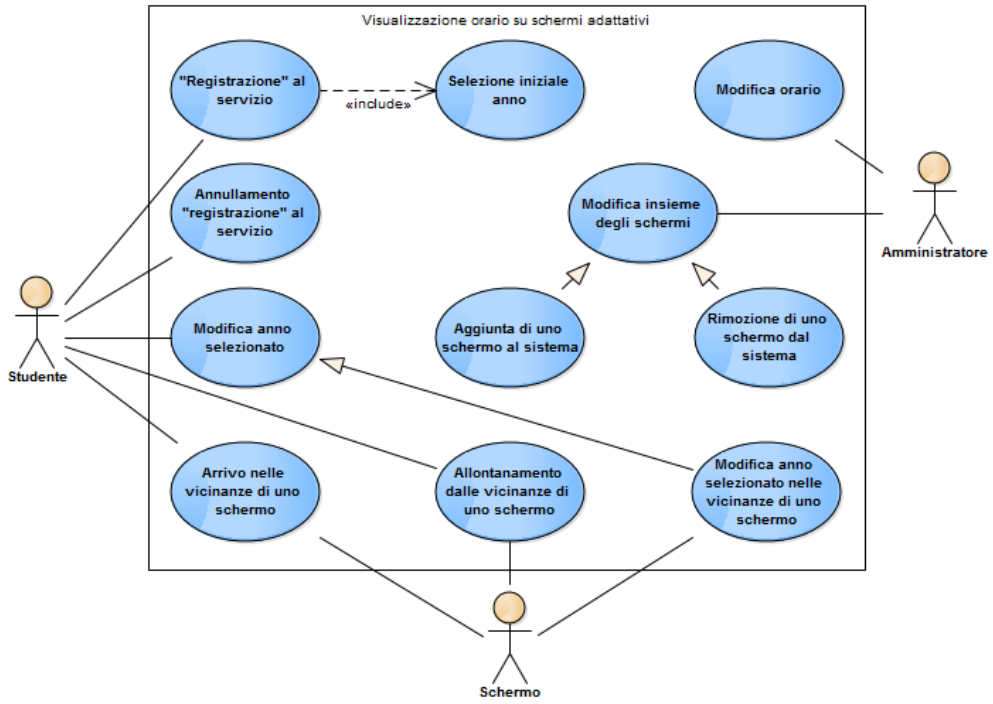



Figura 2.1: Diagramma dei casi d'uso

Come è evidente dal diagramma, i possibili utilizzatori del sistema sono di tre tipologie: le prime due sono fondamentali, mentre la terza svolge un ruolo comunque importante, ma di manutenzione, e quindi secondario ai fini della comprensione dello stesso.

1. **Studente**: è il fruitore del servizio erogato dal sistema: si interfaccia con esso nelle seguenti modalità.

   - **"Registrazione" al servizio**: lo studente decide di utilizzare il servizio che visualizza l'orario scelto su qualunque monitor nelle cui vicinanze egli si trovi.
     - **Selezione iniziale anno**: per potersi "registrare" bisogna necessariamente selezionare un anno di corso. Senza questa ope-

razione infatti il sistema non può sapere quale tra i vari orari visualizzare.

- **Annullamento "registrazione" al servizio**: lo studente decide di non utilizzare più il servizio.
- **Modifica anno selezionato**: lo studente modifica la scelta precedentemente effettuata.
  - **Modifica anno selezionato nelle vicinanze di uno schermo**: si tratta di una versione particolare del caso precedente, poiché se la modifica dell'anno avviene nelle vicinanze di un monitor, ciò implica che quest'ultimo deve immediatamente cambiare il contenuto visualizzato e mostrare l'orario corrispondente alla nuova scelta.
- **Arrivo nelle vicinanze di uno schermo**: lo studente arriva nei pressi di un monitor (a una distanza tale da renderne possibile la lettura).
- **Allontanamento dalle vicinanze di uno schermo**: lo studente si allontana dalle vicinanze di un monitor (ovvero non si trova più a una distanza tale da consentirne la lettura).

2. **Schermo**: è il "centro", sia dal punto di vista teorico che pratico, del sistema, il luogo in cui l'attività di quest'ultimo è concretamente percepibile dall'osservatore esterno. Esso si interfaccia con il sistema nelle modalità elencate di seguito:

   - **Arrivo nelle vicinanze di uno schermo**: quando uno studente arriva nei pressi del monitor, quest'ultimo deve visualizzare l'orario dell'anno che è stato precedentemente selezionato.
   - **Allontanamento dalle vicinanze di uno schermo**: quando uno studente si allontana dalle vicinanze del monitor, esso ritorna a mostrare la schermata di default.
   - **Modifica anno selezionato nelle vicinanze di uno schermo**: quando uno studente seleziona un nuovo anno nel momento in cui si trova già nei pressi del monitor, quest'ultimo deve visualizzare l'orario corrispondente alla scelta appena effettuata.

3. **Amministratore**: è colui a cui è affidata la manutenzione del sistema. Il suo interfacciarsi con quest'ultimo si riassume nelle due modalità seguenti:

   - **Modifica orario**: l'orario delle lezioni può subire modifiche frequentemente, anche all'interno di uno stesso anno accademico. Al verificarsi di tali cambiamenti, l'amministratore inserisce tempestivamente nel sistema la versione aggiornata.

   - **Modifica insieme degli schermi**: se il numero di monitor presenti nella sede del Corso di Laurea e Laurea Magistrale dovesse variare, in positivo o in negativo, bisogna effettuare rispettivamente una delle due operazioni descritte qui di seguito.

     – **Aggiunta di uno schermo al sistema**: nel caso si disponga di un monitor in più, esso viene opportunamente inserito all'interno del sistema.

     – **Rimozione di uno schermo dal sistema**: qualora venga tolto un monitor, quest'ultimo deve anche essere cancellato dal sistema.

### 2.3.1 Scenari dei più significativi casi d'uso

Dopo aver brevemente illustrato, mediante una semplice descrizione di ciascun caso d'uso, il contenuto del diagramma precedente, si procede ora ad un'analisi più approfondita di esso. Infatti, per ognuno dei casi d'uso vi possono essere delle precondizioni, senza le quali non possono essere eseguiti, e diverse sequenze possibili degli eventi, che possono portare al successo o al fallimento dello stesso. Con la seguente rigorosa rappresentazione schematica si vuole quindi analizzare gli scenari dei casi d'uso più significativi per il funzionamento del sistema.

| *Identificatore* | **"Registrazione" al servizio** |
|---|---|
| *Breve descrizione* | Lo studente decide di utilizzare il servizio |
| *Attori primari* | Studente |
| *Precondizioni* | Lo studente non sia già registrato al servizio |
| *Sequenza principale degli eventi* | Lo studente si registra al servizio (attività che comprende la selezione di un anno di cui visualizzare l'orario). Da questo momento può visualizzare l'orario scelto su ogni schermo nelle cui vicinanze si trovi |
| *Condizioni di successo* | La registrazione, comprensiva della selezione dell'anno, avviene correttamente |
| *Condizioni di fallimento* | La registrazione, comprensiva della selezione dell'anno, non avviene |

| *Identificatore* | **Annullamento "registrazione" al servizio** |
|---|---|
| *Breve descrizione* | Lo studente decide di non utilizzare più il servizio |
| *Attori primari* | Studente |
| *Precondizioni* | Lo studente sia già registrato al servizio |
| *Sequenza principale degli eventi* | Lo studente annulla la registrazione al servizio. Da questo momento non visualizza più l'orario scelto su ogni schermo nelle cui vicinanze si trovi |
| *Condizioni di successo* | L'annullamento della registrazione avviene correttamente |
| *Condizioni di fallimento* | L'annullamento della registrazione non avviene |

| *Identificatore* | **Modifica anno selezionato** |
|---|---|
| *Breve descrizione* | Lo studente modifica la scelta dell'anno di corso precedentemente effettuata |
| *Attori primari* | Studente |
| *Precondizioni* | Lo studente sia già registrato al servizio |
| *Sequenza principale degli eventi* | Lo studente seleziona un nuovo anno di corso. Da questo momento può visualizzare il nuovo orario scelto su ogni schermo nelle cui vicinanze si trovi |
| *Condizioni di successo* | La modifica avviene correttamente |
| *Condizioni di fallimento* | La modifica non avviene |

| *Identificatore* | **Modifica anno selezionato nelle vicinanze di uno schermo** |
|---|---|
| *Breve descrizione* | Lo studente, nel momento in cui si trova nelle vicinanze di uno schermo, modifica la scelta dell'anno di corso precedentemente effettuata |
| *Attori primari* | Studente, schermo |
| *Precondizioni* | Lo studente sia già registrato al servizio e si trovi nelle vicinanze di uno schermo |
| *Sequenza principale degli eventi* | Lo studente seleziona un nuovo anno di corso. Immediatamente lo schermo nelle vicinanze visualizza il nuovo orario scelto (che lo studente può ovviamente visualizzare anche su ogni altro schermo nelle cui vicinanze si trovi) |
| *Condizioni di successo* | La modifica avviene correttamente e immediatamente lo schermo visualizza il nuovo orario scelto |
| *Condizioni di fallimento* | La modifica non avviene e lo schermo non visualizza il nuovo orario scelto |

| *Identificatore* | **Arrivo nelle vicinanze di uno schermo** |
|---|---|
| *Breve descrizione* | Lo studente arriva nelle vicinanze di uno schermo e quest'ultimo visualizza l'orario dell'anno che è stato precedentemente selezionato |
| *Attori primari* | Studente, schermo |
| *Precondizioni* | Lo studente sia già registrato al servizio e non si trovi nelle vicinanze di uno schermo |
| *Sequenza principale degli eventi* | Lo studente arriva nelle vicinanze di uno schermo (a una distanza tale da renderne possibile la lettura) e quest'ultimo visualizza l'orario dell'anno che è stato precedentemente selezionato |
| *Condizioni di successo* | La visualizzazione dell'orario avviene correttamente |
| *Condizioni di fallimento* | La visualizzazione dell'orario non avviene |

| *Identificatore* | **Allontanamento dalle vicinanze di uno schermo** |
|---|---|
| *Breve descrizione* | Lo studente si allontana dalle vicinanze di uno schermo e quest'ultimo visualizza la schermata di default |
| *Attori primari* | Studente, schermo |
| *Precondizioni* | Lo studente sia già registrato al servizio e si trovi nelle vicinanze di uno schermo |
| *Sequenza principale degli eventi* | Lo studente si allontana dalle vicinanze di uno schermo (ovvero non si trova più a una distanza tale da consentirne la lettura) e quest'ultimo ritorna a visualizzare la schermata di default |
| *Condizioni di successo* | La visualizzazione della schermata di default avviene correttamente |
| *Condizioni di fallimento* | La visualizzazione della schermata di default non avviene |

# Capitolo 3

# Progettazione del sistema

Nel capitolo precedente si è definita la problematica del caso di studio in esame e sono stati analizzati approfonditamente i requisiti del sistema, indipendentemente da scelte progettuali e implementative. Ora invece si vogliono interpretare tali requisiti in una soluzione architetturale e specificare quali tecnologie si sono scelte, così da avere un progetto completo del sistema che possa fungere da input per la successiva fase di sviluppo.

## 3.1 Utilizzo dei beacon per rilevare la prossimità

Come è emerso chiaramente, l'aspetto fondamentale, oltre che innovativo, del sistema è che quest'ultimo abbia la capacità di **"accorgersi" della presenza di una particolare persona nei pressi di uno schermo**. Questo è il criterio in base al quale il monitor agisce, modificando le informazioni visualizzate. Perciò la prima cosa da fare in questa fase di progettazione è pensare a un meccanismo grazie al quale il sistema possa rilevare la prossimità di uno studente a uno schermo.

Tenendo conto delle tecnologie approfondite nel Capitolo 1, si è quindi progettato di **posizionare un beacon accanto ad ogni schermo**, cosicché il segnale BLE di ciascuno di essi sia rilevabile solo ad una certa distanza dal monitor corrispondente.

Dopo alcune valutazioni, si è deciso di utilizzare dei modelli iBKS102 della Accent Systems[1]. Questi dispositivi presentano sulla carta numerosi vantaggi:

- Prima di tutto, è nota la suddivisione in unità logiche di informazione dei bit del segnale Bluetooth Low Energy. Perciò questi device, pur non essendo conformi allo standard AltBeacon, emettono un fascio di informazioni che può essere interpretato mediante la Android Beacon Library.
- I parametri UUID, Major, Minor, intervallo di advertising e potenza del segnale sono tutti modificabili: per rendere tale operazione il più possibile semplice ed intuitiva esiste un'app sviluppata dal produttore, e disponibile per Android e iOS. Inoltre è possibile impostare una password per impedire ad altri di modificare il valore di ciascun parametro.
- Messi a confronto con modelli di altri produttori, a parità di caratteristiche offerte, il loro costo è abbastanza contenuto.

### 3.1.1 Valutazione della stima della distanza

Prima di procedere con le successive fasi della progettazione, sono stati svolti sui modelli scelti alcuni test preliminari per valutare la precisione della stima della distanza, utilizzando un device Nexus 5 (con Android aggiornato alla versione 5.1 Lollipop) e la Android Beacon Library. Si noti anche che, prima di svolgere tali verifiche, tutti i beacon sono stati opportunamente calibrati.

Purtroppo dai test è emerso come, per il primo segnale rilevato, la stima della distanza non sia eccessivamente precisa (si è registrato, in media, un errore di circa 3-4 m), ma soprattutto come, camminando nell'area raggiunta dal segnale, l'aggiornamento della stima al momento della ricezione di nuovi segnali sia oltremodo lento, anche impostando al massimo la frequenza di advertising (ossia scegliendo un intervallo di 0.1 s tra due segnali consecutivi). Quest'ultimo aspetto è in buona parte dovuto alla scelta da parte della Android Beacon Library di calcolare la distanza facendo a intervalli regolari la media dei valori ricevuti, la cui diretta conseguenza è una variazione molto più

[1]Sito ufficiale della Accent Systems: `http://www.accent-systems.com`

lenta della stima della distanza. Oltretutto, i beacon sono stati calibrati in condizioni ideali, ovvero senza ostacoli fisici e grosse interferenze tra essi e il dispositivo Android usato per la calibrazione. In situazioni di normale utilizzo del sistema è però probabile che le condizioni non siano ideali, e non è possibile darne una previsione a priori valida per tutti i casi (per esempio, non si può sapere se vi saranno grosse interferenze dovute ad altre onde radio).

In conclusione, si è verificato quindi come **il criterio per la modifica della visualizzazione di uno schermo non si possa basare su tale stima della distanza**. Per prima cosa, infatti, si è ritenuto eccessivo l'errore sul primo segnale rilevato: ma se anche si fosse deciso di tollerare tale errore, ciò che è inaccettabile per le finalità del sistema è la lentezza nell'aggiornamento della stima. Solitamente infatti uno studente si muove in maniera rapida nei corridoi e nei pressi di uno schermo: se il criterio per visualizzare l'orario scelto fosse trovarsi entro una certa distanza dal monitor, nel migliore dei casi egli dovrebbe aspettare sostare per almeno 20 secondi davanti allo schermo, in attesa dell'aggiornamento della stima della distanza. Nel peggiore dei casi, invece, lo schermo potrebbe addirittura non modificare mai il contenuto visualizzato, se l'errore fosse superiore a una certa soglia e/o se le condizioni non fossero ideali.

Inoltre, viste le premesse, **non si è ritenuto opportuno tentare l'utilizzo di algoritmi di triangolazione** poiché, data la poca precisione nella stima della distanza, si è previsto che tali tecniche avrebbero potuto amplificare l'errore iniziale, o comunque, nel migliore dei casi, non avrebbero portato a risultati ottimali. Allo stesso tempo però esse avrebbero determinato un grande aumento dei costi sia in termini di hardware (sarebbe stato necessario aggiungere almeno altri due beacon nei pressi di ogni schermo) sia soprattutto di tempo (sarebbe stato necessario approfondire tali algoritmi, studiare il posizionamento più adatto per ciascun beacon, effettuare una nuova calibrazione per ognuno di essi, ecc...). In conclusione, tale scelta non è sembrata un investimento conveniente.

Si è quindi pensato di **abbassare la potenza di trasmissione del segnale al minimo** (-23 dBm), così che esso fosse percepibile solo ad una distanza non eccessiva dallo schermo, e di usare quindi la semplice ricezione o meno

del segnale BLE come criterio per la modifica dei contenuti mostrati da un monitor.

## 3.2 Architettura logica

Come osservato durante la fase di analisi, le entità, gli attori fondamentali del sistema sono due: lo *studente* e lo *schermo*. Pertanto, dal punto di vista architetturale, ciascuno di essi deve essere considerato come un'unità modulare a sé stante, un componente software separato dal resto del sistema, ma allo stesso tempo in grado di interagire facilmente con altre parti di esso. Si vuole quindi ora descrivere ad alto livello l'architettura software di entrambi e motivare le scelte effettuate, tenendo continuamente in considerazione i requisiti precedentemente esposti, che rappresentano una guida per poter avanzare nella fase di progettazione.

### 3.2.1 Architettura software per lo studente

Per permettere allo studente di interfacciarsi con il sistema si è progettata un'**app Android**. I vantaggi di questa decisione sono molteplici.

- Dopo i test già effettuati sui beacon e le conclusioni tratte, si tratta di una scelta quasi "obbligata".
- Come già detto, Android è assunto sempre più di frequente come riferimento per la costruzione di sistemi basati su dispositivi mobili.
- Essendo il sistema operativo di gran lunga più diffuso in ambito mobile, il sistema in futuro potrebbe avere una maggiore diffusione. Inoltre, dal momento che i beacon agiscono in peripheral role, al device Android è richiesto di supportare solo il central role (che, come si è visto, è disponibile su un numero di dispositivi molto maggiore rispetto al peripheral).
- Da ultimo, usando Android, si possono sviluppare in maniera molto semplice i requisiti richiesti per lo studente:
  - Se si progetta l'app in modo che, dopo il primo avvio, resti sempre attiva in background, i requisiti di "registrazione" al servizio e

di annullamento della stessa vengono realizzati semplicemente con l'installazione e la disinstallazione dell'app sul dispositivo Android di proprietà dello studente.

- Dopo l'installazione, al primo avvio dell'app, è possibile comunicare l'anno di interesse, completando così la propria "registrazione" al servizio.
- La scelta relativa all'anno di corso è ovviamente modificabile in ogni momento direttamente dall'interfaccia utente dell'app.

Inoltre, tramite le funzionalità offerte dalla Android Beacon Library, si può configurare l'app per rilevare solamente i beacon aventi un certo UUID (che deve corrispondere a quello scelto per identificare tutti i beacon appartenenti al sistema).

### 3.2.2 Architettura software per lo schermo

I monitor presenti nella sede del Corso di Laurea e Laurea Magistrale non sono dotati di capacità computazionale. Per poter costruire attorno ad essi il sistema, dunque, bisogna che **ciascuno sia collegato a un dispositivo in grado di elaborare informazioni**.

La scelta del tipo di device deve essere innanzitutto guidata dal meccanismo che si vuole utilizzare per l'interazione con i dispositivi Android degli studenti.[2] Mentre, una volta scelte le tecnologie beacon e Android, la comunicazione tra di esse può avvenire solo tramite Bluetooth Low Energy, quella tra i device Android e i dispositivi collegati agli schermi può anche utilizzare altre forme di interazione. Dal momento che tutta la sede universitaria è ben coperta dalla rete wireless di Ateneo, a cui ciascuno studente può accedere tramite le proprie credenziali, e considerato inoltre che su moltissimi dispositivi Android (soprattutto smartphone) è attiva una connessione dati fornita da un operatore di telefonia mobile, si è ritenuto che la forma di comunicazione più immediata, affidabile e semplice da gestire fosse rappresentata da **internet** e

[2]Si noti che l'argomento dell'interazione tra le componenti del sistema verrà affrontato per esteso nel paragrafo dedicato. Tuttavia è ora necessario fare qualche considerazione preliminare a riguardo.

dal suo protocollo **HTTP**. Per questo motivo il requisito fondamentale per i device collegati ai monitor è che essi siano dotati di una scheda di rete in grado di ricevere richieste HTTP.

Una volta rispettata questa caratteristica essenziale, si vuole progettare questa componente del sistema nel modo più generico possibile, rendendola per quanto si può indipendente dal particolare dispositivo su cui viene eseguita. Ad ogni modo, la scelta ideale potrebbe essere usare un **sistema embedded** (dato il costo generalmente non elevato, la facilità di programmazione e le dimensioni estremamente ridotte). Quindi a partire da ora, per brevità, nel testo si indicherà qualunque device collegato a un monitor come sistema embedded.

Per quanto riguarda i requisiti per lo schermo, il loro sviluppo sarà quindi basato sul meccanismo di comunicazione scelto, ovvero internet: infatti il sistema embedded modificherà il contenuto da visualizzare (uno specifico orario oppure la schermata di default) in seguito alla ricezione di richieste HTTP.

### 3.2.3 Architettura software per l'interazione tra studente e schermo

Anche se l'interazione tra studente e schermo verrà affrontata più avanti in questo capitolo, è ora necessario fare qualche considerazione a riguardo, per giustificare l'introduzione nel sistema di un'ulteriore componente, e poterne quindi descrivere l'architettura software.

La comunicazione tra le due parti del sistema finora analizzate potrebbe, sulla carta, avvenire in maniera diretta: nel momento in cui uno studente arriva nelle vicinanze di un monitor o si allontana, l'app Android invia una richiesta HTTP al sistema embedded collegato allo schermo, cosicché esso modifichi il contenuto visualizzato. Ma proprio qui risiede il problema fondamentale: la **corrispondenza tra beacon e sistemi embedded**. L'app infatti deve sapere, per ogni beacon, qual è il corrispondente sistema a cui inviare le proprie richieste. È necessaria pertanto una struttura dati in cui memorizzare il mapping tra beacon e sistemi embedded. A livello teorico essa potrebbe essere parte dell'app Android, ma ad ogni modifica avente effetto sull'insieme degli schermi (aggiunta o rimozione di un monitor, modifica di un URL di un

sistema embedded, ecc...), andrebbe rilasciato un aggiornamento per tutti gli utenti! E, cosa ancor più grave e considerata inaccettabile, senza aggiornare la propria applicazione uno studente potrebbe non aver più accesso al sistema o a parte di esso.

Per questi motivi è evidente come non sia sufficiente progettare soltanto dei meccanismi di interazione tra le due parti del sistema finora descritte, ma **occorra necessariamente un'ulteriore componente software "centrale"** (accessibile solo da un amministratore) che contenga il mapping tra beacon e sistemi embedded, e abbia inoltre la funzione di gestire e regolare la comunicazione fra studente e schermo. E, dal momento che tale comunicazione avviene tramite internet, è necessario che anche questa parte software sia dotata di una scheda di rete, poiché deve essere in grado di ricevere ed effettuare richieste HTTP. Pertanto la più naturale e immediata conseguenza di quanto detto è progettare questa componente del sistema come un **server** centrale in esecuzione su un terminale collocato nella sede del Corso di Laurea e Laurea Magistrale.

## 3.3 Struttura

Terminata la descrizione ad alto livello delle tre componenti del sistema, si vuole ora scendere ad un livello più basso, progettando per ognuna i campi e i metodi necessari e descrivendone scopo e modalità di funzionamento, cosicché la successiva fase di implementazione possa essere guidata in ogni passo dalla struttura qui descritta.

### 3.3.1 Struttura dell'app Android

L'app Android, la cui struttura è rappresentata in Figura 3.1 conformemente allo standard UML, è composta da due activity.

1. `MainActivity`: come indica il nome è l'activity principale. Essa implementa l'interfaccia `BeaconConsumer` della Android Beacon Library, operazione indispensabile per fruire dei servizi offerti dalla libreria. È la prima activity ad essere lanciata all'avvio dell'applicazione, e quella

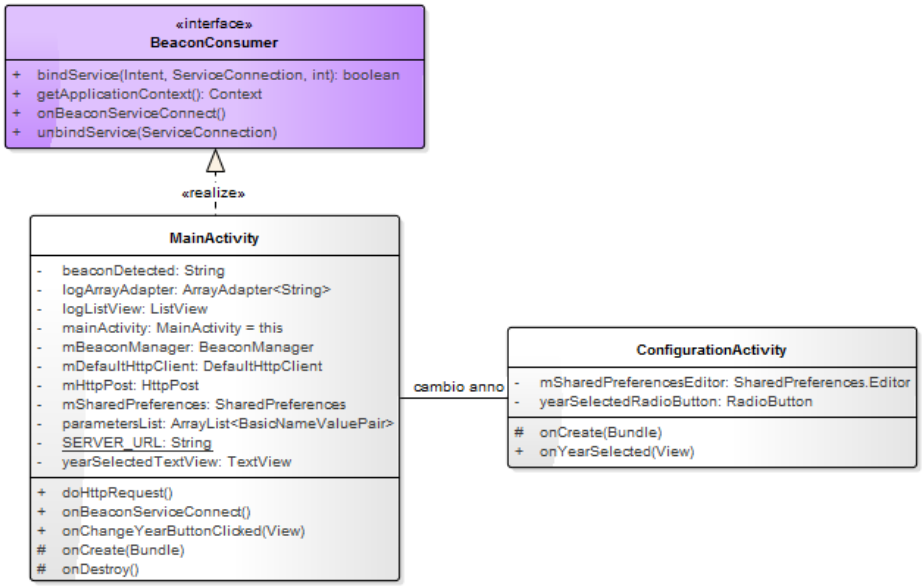



Figura 3.1: Diagramma delle classi per l'app Android

che realizza la quasi totalità delle operazioni necessarie. La sua interfaccia grafica contiene un'indicazione dell'anno selezionato, un pulsante per cambiare la scelta, e una finestra di log per tenere traccia degli eventi di avvicinamento e allontanamento da uno schermo. I metodi implementati all'interno della classe sono i seguenti:

- `onCreate()`: si tratta dell'override del corrispondente metodo della classe `Activity`. Invocato alla creazione dell'activity, la inizializza preparando gli elementi dell'interfaccia grafica (indicatore dell'anno, finestra di log...) e tutti i componenti necessari per gestire le richieste HTTP e l'interazione con i beacon.
- `onDestroy()`: è l'override del metodo corrispondente della classe `Activity`. Esso viene eseguito alla distruzione dell'activity e si occupa soprattutto di "rilasciare" l'oggetto di classe `BeaconManager` che gestisce la comunicazione con i beacon.
- `onBeaconServiceConnect()`: si tratta dell'override del corrispondente metodo dell'interfaccia `BeaconConsumer` (di cui è l'unico a dover essere obbligatoriamente implementato) ed è in assoluto il metodo più importante dell'app Android, e addirittura dell'intero

sistema. È di un metodo di callback che viene invocato appena è stato istanziato l'oggetto di classe `BeaconManager`: si occupa di avviare la ricerca dei beacon, e innesca altri componenti e metodi di callback per gestire l'interazione con i beacon, e le operazioni da effettuare all'avvicinamento o allontanamento da uno schermo.

- `doHttpRequest()`: gestisce le richieste HTTP da effettuare quando si arriva o ci si allontana dalle vicinanze di un monitor.
- `onChangeYearButtonClicked()`: invocato al momento del click sul pulsante per modificare l'anno selezionato.

2. `ConfigurationActivity`: si occupa della gestione della selezione dell'anno di corso, che viene memorizzato nelle SharedPreferences, in modo tale da non doverlo inserire ad ogni nuovo avvio dell'app. La sua interfaccia grafica contiene semplicemente un `RadioGroup` contenente un `RadioButton` per ogni anno di corso, così da consentirne la scelta di uno e uno solo. I metodi implementati al suo interno sono i seguenti:

   - `onCreate()`: override del metodo corrispondente contenuto nella classe `Activity`, viene eseguito alla creazione dell'activity, e si occupa soprattutto di preparare la sua struttura grafica.
   - `onYearSelected()`: invocato al momento della selezione di uno degli anni di corso.

### 3.3.2 Struttura dell'applicazione per il sistema collegato allo schermo

Ogni sistema embedded collegato a un monitor deve essere sempre pronto, in ogni momento, a ricevere richieste HTTP. Si è quindi progettato un **web service** denominato `MonitorManager` (la cui struttura è rappresentata graficamente in Figura 3.2) affinché stia in esecuzione continua. Esso è composto da una semplice **servlet** Java, di nome `MonitorServlet`: i metodi implementati al suo interno sono i seguenti.

- `doGet()`: override del corrispondente metodo della classe `HttpServlet`, viene invocato all'arrivo di una richiesta HTTP di tipo GET alla servlet.

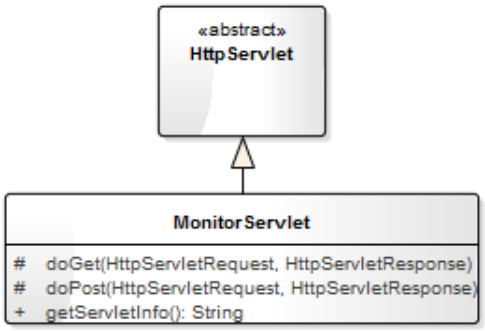



Figura 3.2: Diagramma delle classi per il web service `MonitorManager`

In questo metodo è racchiuso tutto il funzionamento del web service, in quanto tutte le richieste in arrivo dal server centrale sono di tipo GET. Ricevuta una richiesta, il metodo si occupa di visualizzare effettivamente il contenuto desiderato (uno specifico orario oppure la schermata di default).

- `doPost()`: override del corrispondente metodo della classe `HttpServlet`, viene invocato all'arrivo di una richiesta HTTP di tipo POST alla servlet. Ai fini del sistema non è richiesto che svolga alcuna operazione, in quanto il server centrale non invia ad alcun monitor richieste POST.

- `getServletInfo()`: override del corrispondente metodo contenuto nella classe `GenericServlet` (superclasse di `HttpServlet`), fornisce una stringa con una breve descrizione della servlet.

### 3.3.3 Struttura del server centrale

È necessario che anche il server centrale, come è ovvio, sia in esecuzione continua, poiché in ogni momento deve essere pronto sia a ricevere che a effettuare richieste HTTP. Anche in questo caso la soluzione migliore è apparsa quella di progettare un **web service**, denominato `CentralServer`, che presenta la struttura indicata in Figura 3.3. Esso è formato da una **servlet** Java denominata `CentralServlet`. Essa contiene i seguenti campi statici, di fondamentale importanza, in quanto memorizzano valori di utilità generale per

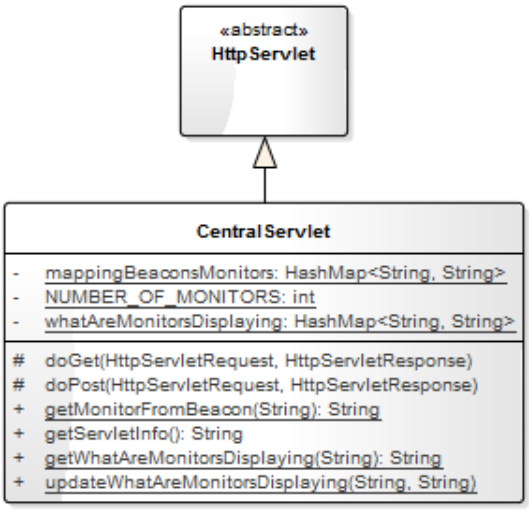



Figura 3.3: Diagramma delle classi per il server centrale

il sistema. Data la criticità di questi dati, essi sono tutti implementati come privati.

- `NUMBER_OF_MONITORS`: valore intero che tiene traccia di quanti schermi fanno attualmente parte del sistema.

- `mappingBeaconsMonitors`: è un campo fondamentale, senza il quale il funzionamento del sistema non sarebbe possibile. Si tratta di una struttura dati di tipo HashMap contenente il mapping tra il MAC address di ogni beacon e l'URL della `MonitorServlet` in esecuzione sul sistema collegato allo schermo corrispondente.

- `whatAreMonitorsDisplaying`: campo di tipo HashMap che tiene traccia di cosa ogni schermo sta attualmente mostrando. Da notare che la chiave della mappa è sempre il MAC address del beacon, per uniformità con la struttura dati precedente: tramite quest'ultima, partendo dal MAC address si otterrà lo schermo corrispondente.

La servlet al suo interno implementa inoltre questi metodi:

- `doGet()`: override del corrispondente metodo della classe `HttpServlet`, viene invocato all'arrivo di una richiesta HTTP di tipo GET alla servlet.

Ai fini del sistema non è richiesto che svolga alcuna operazione, in quanto il server centrale non riceve da alcun dispositivo Android richieste GET.

- `doPost()`: override del corrispondente metodo della classe `HttpServlet`, viene invocato all'arrivo di una richiesta HTTP di tipo POST alla servlet. Questo metodo implementa al suo interno le operazioni fondamentali del server centrale, poiché tutte le richieste in arrivo dai device Android sono di tipo POST. Ricevuta una richiesta da uno studente nelle vicinanze di uno schermo, il metodo si occupa di inviare una richiesta HTTP di tipo GET al sistema embedded collegato a tale monitor.

- `getServletInfo()`: override del corrispondente metodo contenuto nella classe `GenericServlet` (superclasse di `HttpServlet`), fornisce una stringa con una breve descrizione della servlet.

- `getMonitorFromBeacon()`: metodo che, fornito in input l'indirizzo MAC di un beacon, restituisce in output lo schermo corrispondente (o meglio, l'URL della `MonitorServlet` in esecuzione sul sistema collegato allo schermo corrispondente).

- `getWhatAreMonitorsDisplaying()`: metodo che, fornito in input l'indirizzo MAC di un beacon, restituisce in output cosa lo schermo corrispondente sta mostrando in quell'istante.

- `updateWhatAreMonitorsDisplaying()`: metodo che, fornito in input l'indirizzo MAC di un beacon e un contenuto visualizzato da uno schermo, aggiorna la struttura dati `whatAreMonitorsDisplaying`, associando il beacon (e quindi il monitor) al contenuto.

## 3.4 Interazione

Dopo aver descritto in maniera dettagliata la struttura della componente hardware (i beacon) e delle tre componenti software che, nel loro insieme, danno vita al sistema, per terminare questa fase non resta altro che progettare le modalità che permettono il formarsi di questo insieme: ovvero, come queste parti interagiscono tra loro per poter effettivamente offrire le funzionalità del

sistema. Come infatti si è visto nessuna delle componenti è autosufficiente, nessuna da sola è in grado di offrire funzionalità utili, ma ciascuna è pensata per poter interagire con una o più delle rimanenti. D'altro canto, per ogni sistema distribuito la comunicazione tra le componenti è uno degli aspetti più importanti, ma specialmente lo è in questo caso, in cui le singole parti non svolgono operazioni complesse, e quindi **l'aspetto più significativo del sistema consiste proprio nell'interazione fra di esse**.

Tra i vari casi d'uso (definiti durante la fase di analisi), due sono quelli che avviano l'interazione tra tutte le parti del sistema: si tratta dell'*arrivo nelle vicinanze di uno schermo* e dell'*allontanamento dalle vicinanze di uno schermo*. In Figura 3.4 si può vedere una rappresentazione schematica valida per entrambi: infatti l'unica cosa che differenzia le due interazioni sono i parametri delle richieste HTTP.

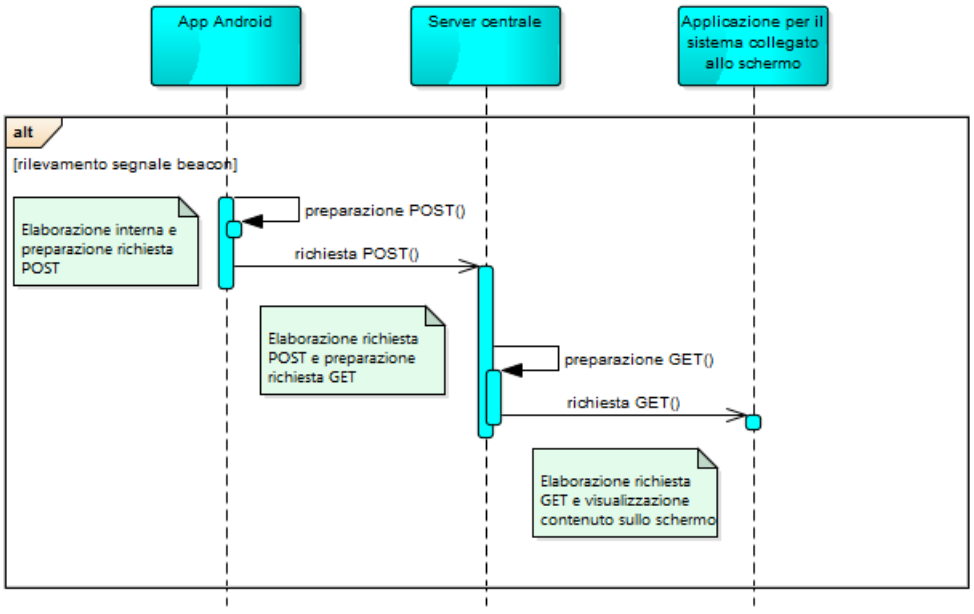



Figura 3.4: Diagramma di sequenza

### 3.4.1 Interazione avviata dall'arrivo nelle vicinanze di uno schermo

Una volta avviata, l'app Android fa una continua scansione alla ricerca dei beacon del sistema. Quando comprende di trovarsi nelle vicinanze di uno di essi (in una cosiddetta `region`), immediatamente informa l'utente tramite la finestra di log, e invia una richiesta HTTP, di tipo POST, alla servlet in esecuzione sul server centrale. L'aspetto fondamentale di questa richiesta sono i tre parametri:

1. `requestType`: valore intero usato per distinguere le richieste di visualizzazione di uno specifico orario da quelle di visualizzazione della schermata di default.

2. `year`: l'anno di corso precedentemente selezionato dallo studente (del quale si vuole dunque che sia mostrato l'orario).

3. `beacon`: l'indirizzo MAC del beacon che ha emesso il segnale rilevato.

Il server centrale, appena ricevuta una richiesta di tipo POST, valuta il parametro `requestType` e capisce che si tratta di una richiesta di visualizzazione di uno specifico orario. Quindi, utilizzando il parametro `beacon`, risale allo schermo corrispondente tramite il metodo `getMonitorFromBeacon()`, e invia alla servlet in esecuzione sul sistema ad esso collegato una nuova richiesta HTTP (questa volta di tipo GET), avente come parametro il valore contenuto nel parametro `year` della POST. Inoltre, tramite il metodo `updateWhatAreMonitorsDisplaying()`, il server centrale aggiorna l'informazione relativa a cosa il monitor sta visualizzando.

Il sistema collegato allo schermo, non appena riceve una richiesta di tipo GET, verifica se essa contenga un parametro o no: in questo caso la risposta è affermativa, e ciò significa che si tratta di una richiesta di visualizzazione di uno specifico orario. Infine si valuta il parametro, per risalire all'anno e mostrare sullo schermo l'orario corrispondente.

### 3.4.2 Interazione avviata dall'allontanamento dalle vicinanze di uno schermo

Quando l'app Android comprende di non trovarsi più in una `region` (ovvero nelle vicinanze di uno schermo), lo notifica allo studente attraverso la finestra di log e invia una richiesta HTTP, di tipo POST, alla servlet che è in esecuzione sul server centrale. Ancora una volta, l'aspetto fondamentale di questa richiesta sono i parametri: è attraverso di essi che avviene la differenziazione tra una richiesta causata dall'arrivo nelle vicinanze di un monitor, e una causata dall'allontanamento dalle vicinanze di un monitor. In questo caso, non dovendo visualizzare uno specifico orario, non è necessario inoltrare l'informazione relativa all'anno selezionato; pertanto i parametri sono soltanto due.

1. `requestType`: valore intero usato per distinguere le richieste di visualizzazione di uno specifico orario da quelle di visualizzazione della schermata di default.

2. `beacon`: l'indirizzo MAC del beacon che ha emesso il segnale rilevato.

Non appena il server centrale riceve una POST, esso comprende, tramite il parametro `requestType`, che si tratta di una richiesta di visualizzazione della schermata di default. Dunque, mediante il parametro `beacon`, risale al monitor corrispondente ad esso per mezzo del metodo `getMonitorFromBeacon()`, e invia alla servlet in esecuzione sul sistema ad esso collegato una nuova richiesta HTTP (questa volta di tipo GET). In questo caso, tale richiesta non ha bisogno di parametri, perché non è necessario comunicare uno specifico anno di cui mostrare l'orario. Oltre a ciò, il server centrale, attraverso il metodo `updateWhatAreMonitorsDisplaying()`, aggiorna il dato relativo a cosa il monitor sta visualizzando.

Appena riceve una richiesta di tipo GET, il sistema collegato allo schermo verifica se essa contenga o meno un parametro. In questo caso la risposta è no, e questo significa che si tratta di una richiesta di visualizzazione della schermata di default. Quindi al sistema non rimane altro da fare che mostrare sullo schermo ad esso collegato la schermata di default.

# Capitolo 4

# Sviluppo del prototipo

Dopo aver affrontato, per il caso di studio scelto, le fasi di analisi e di progettazione, in quest'ultimo capitolo si vuole procedere allo sviluppo del prototipo e alla valutazione dei risultati ottenuti. Più in dettaglio, dapprima ci si occuperà dell'implementazione delle tre componenti software del sistema e del loro deployment; successivamente verranno brevemente descritti i risultati dei test, tramite i quali si potrà verificare direttamente le potenzialità attuali delle tecnologie utilizzate.

## 4.1 Implementazione

### 4.1.1 Implementazione dell'app Android

Come già evidenziato in fase di progettazione, la `MainActivity` è l'activity più importante, poiché contiene la quasi totalità delle operazioni richieste a tutta l'app. Pertanto, per potere avere una visione più chiara e completa del suo funzionamento, sono qui riportate le parti più importanti della sua implementazione.

Per prima cosa si può osservare la dichiarazione di tutti i suoi campi, così da conoscere la classe di cui sono oggetto e comprendere meglio i punti successivi del codice in cui saranno presenti. Successivamente è possibile vedere l'implementazione completa del metodo `onCreate()`: in esso, dopo un iniziale controllo delle SharedPreferences (per sapere se sia necessario avviare la `ConfigurationActivity`), vengono inizializzati quasi tutti i campi della clas-

se, e infine viene istanziato l'oggetto di classe `BeaconManager`, fondamentale per l'interazione tra l'app e i beacon. È anche riportata l'implementazione completa del metodo `onDestroy()`, che è incaricato soprattutto di "rilasciare" l'oggetto di classe `BeaconManager`.

```
private static final String SERVER_URL = /*URL della servlet in
    esecuzione sul server centrale*/;
private final MainActivity mainActivity = this;
private SharedPreferences mSharedPreferences;
private TextView yearSelectedTextView;
private ArrayAdapter<String> logArrayAdapter;
private ListView logListView;
private ArrayList<BasicNameValuePair> parametersList;
private HttpPost mHttpPost;
private DefaultHttpClient mDefaultHttpClient;
private BeaconManager mBeaconManager;
private String beaconDetected;

@Override
protected void onCreate(Bundle savedInstanceState) {

   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);

   mSharedPreferences = getSharedPreferences("MyPreferences",
       MODE_PRIVATE);

   if (!mSharedPreferences.contains("year")) {

      Intent intent = new Intent(this, ConfigurationActivity.class);
      startActivity(intent);
      finish();

   } else {

      yearSelectedTextView = (TextView)
```

```
        findViewById(R.id.yearSelectedTextView);
    yearSelectedTextView.setText("L'anno selezionato e' il " +
        mSharedPreferences.getString("year", "Errore: anno non
        trovato"));

    logArrayAdapter = new ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_1);
    logListView = (ListView) findViewById(R.id.logListView);
    logListView.setAdapter(logArrayAdapter);

    parametersList = new ArrayList<BasicNameValuePair>(3);
    mHttpPost = new HttpPost(SERVER_URL);
    mDefaultHttpClient = new DefaultHttpClient();

    mBeaconManager = BeaconManager.getInstanceForApplication(this);
    mBeaconManager.getBeaconParsers().add(new
        BeaconParser().setBeaconLayout(
          "m:2-3=0215,i:4-19,i:20-21,i:22-23,p:24-24"));
    mBeaconManager.bind(this);

  }

}

@Override
protected void onDestroy() {
  super.onDestroy();
  if (mBeaconManager != null) {
    mBeaconManager.unbind(this);
  }
}
```

Listato 4.1: `MainActivity`: campi, metodo `onCreate()` e metodo `onDestroy()`

Infine è sembrato d'obbligo riportare l'implementazione completa del metodo `onBeaconServiceConnect()` in quanto, come già detto, si tratta del metodo

più importante dell'intera applicazione. **In esso viene gestito ogni aspetto relativo all'interazione con i beacon**: viene avviata la scansione alla ricerca di questi ultimi, sono rilevate l'entrata o l'uscita da una `region` e vengono svolte alcune operazioni preliminari per preparare le richieste POST (che saranno poi effettivamente gestite e inviate tramite il metodo `doHttpRequest()`).

```
@Override
public void onBeaconServiceConnect() {
   mBeaconManager.setMonitorNotifier(new MonitorNotifier() {

      @Override
      public void didDetermineStateForRegion(int arg0, Region arg1) {
      }

      @Override
      public void didEnterRegion(Region arg0) {
         mainActivity.runOnUiThread(new Runnable() {
            @Override
            public void run() {
               logArrayAdapter.add("Sei arrivato nelle vicinanze di
                   uno schermo");
            }
         });
         mBeaconManager.setRangeNotifier(new RangeNotifier() {

            @Override
            public void didRangeBeaconsInRegion(Collection<Beacon>
                arg0, Region arg1) {
               if (!arg0.isEmpty()) {
                  try {
                     mBeaconManager.stopRangingBeaconsInRegion(arg1);
                  } catch (RemoteException e) {}
                  beaconDetected =
                      arg0.iterator().next().getBluetoothAddress();
                  parametersList.clear();
                  parametersList.add(new
```

```
                BasicNameValuePair("requestType", "1"));
            parametersList.add(new BasicNameValuePair("year",
                mSharedPreferences.getString("year", "[Errore:
                anno non trovato]")));
            doHttpRequest();
          }
        }

      });
      try {
        mBeaconManager.startRangingBeaconsInRegion(new
            Region("Ranging schermo adattativo",
            Identifier.parse("12345678123456781234567812345678"),
            null, null));
      } catch (RemoteException e) {}
    }

    @Override
    public void didExitRegion(Region arg0) {
      mainActivity.runOnUiThread(new Runnable() {
        @Override
        public void run() {
          logArrayAdapter.add("Ti sei allontanato dalle
              vicinanze di uno schermo");
        }
      });
      parametersList.clear();
      parametersList.add(new BasicNameValuePair("requestType",
          "0"));
      doHttpRequest();
    }

});
try {
  mBeaconManager.startMonitoringBeaconsInRegion(new
      Region("Monitoring schermo adattativo",
```

```
            Identifier.parse("12345678123456781234567812345678"), null,
            null));
    } catch (RemoteException e) {}
}
```

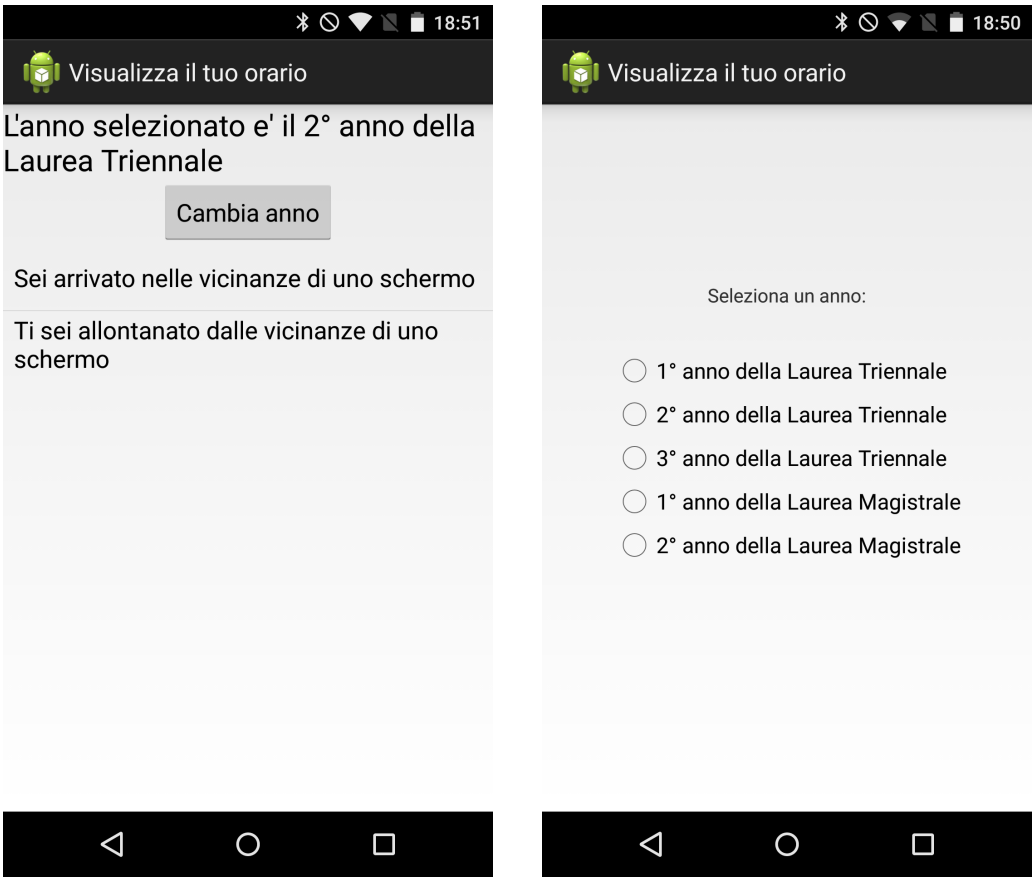
Listato 4.2: `MainActivity`: metodo `onBeaconServiceConnect()`

Per quanto riguarda la `ConfigurationActivity`, è utile spendere qualche parola per illustrare brevemente i criteri secondo i quali viene attivata.

Come si è visto, nell'`onCreate()` della `MainActivity`, dopo aver svolto le operazioni del metodo di cui si è fatto override e aver indicato qual è il layout dell'activity, la prima operazione svolta è controllare se nelle SharedPreferences è memorizzato un valore `year`. Se la risposta è affermativa, significa che è già stato selezionato un anno, e che di conseguenza l'app può funzionare correttamente, visualizzando sugli schermi l'orario di quell'anno: in questo caso il metodo `onCreate()` continua a svolgere le sue operazioni. È comunque possibile modificare in ogni momento la scelta effettuata tramite la pressione del pulsante "Cambia anno" (che attiva la `ConfigurationActivity`). Se invece tale valore non è memorizzato, questo vuol dire che non è mai stato scelto un anno, e che di conseguenza è impossibile un corretto funzionamento dell'app. Quindi, per permettere questa scelta, viene avviata immediatamente la `ConfigurationActivity`, e subito dopo è terminata la `MainActivity`.

La `ConfigurationActivity` consente la selezione di un solo anno di corso. La scelta effettuata è immediatamente memorizzata tramite il valore `year` delle SharedPreferences, e subito dopo è avviata la `MainActivity` e terminata la `ConfigurationActivity`. A questo punto può iniziare la ricerca dei beacon.

In Figura 4.1 si può vedere l'aspetto delle due activity. È molto importante notare che, una volta che l'app è stata avviata (e che è memorizzata la selezione di un anno) essa, anche se posta in background, continua ad essere in esecuzione (e quindi i contenuti visualizzati dai monitor continuano a venire modificati). Ad ogni modo lo studente può, in qualunque momento lo desideri, consultare la finestra di log che riporta gli eventi di arrivo nelle vicinanze di uno schermo e di allontanamento dalle vicinanze di uno schermo.




(a) `MainActivity` (b) `ConfigurationActivity`

Figura 4.1: Aspetto delle due activity

### 4.1.2 Implementazione dell'applicazione per il sistema collegato allo schermo

L'implementazione dell'applicazione per il sistema collegato allo schermo non presenta particolarità o aspetti significativi all'interno del codice rispetto a quanto già enunciato durante la fase di progettazione. All'arrivo di una richiesta HTTP GET, il metodo `doGet()` controlla se tale richiesta contiene parametri: se la risposta è affermativa, viene visualizzato l'orario dell'anno indicato dal parametro, altrimenti è mostrata la schermata di default.

### 4.1.3 Implementazione del server centrale

Nell'implementazione del server centrale svolge un ruolo molto importante il costruttore della servlet: esso infatti, all'avvio della servlet (che deve avvenire obbligatoriamente per mano di un amministratore) popola le due HashMap `mappingBeaconsMonitors` e `whatAreMonitorsDisplaying`.

Data la grande importanza, già evidenziata durante la progettazione, del metodo `doPost()`, **che realizza il compito di fare da congiunzione tra studente e schermo**, è qui riportata la sua implementazione completa.

```
@Override
protected void doPost(HttpServletRequest request,
    HttpServletResponse response)
        throws ServletException, IOException {
    String beacon = request.getParameter("beacon");
    URLConnection connection;
    InputStream answer;
    switch
        (Integer.parseInt(request.getParameter("requestType"))) {
        case 1:
            String year = request.getParameter("year");
            String query = String.format("param1=%s",
                URLEncoder.encode(year, "UTF-8"));
            connection = new URL(getMonitorFromBeacon(beacon) +
                "?" + query).openConnection();
            connection.setRequestProperty("Accept-Charset",
                "UTF-8");
            answer = connection.getInputStream();
            updateWhatAreMonitorsDisplaying(beacon, year);
            return;
        case 0:
            connection = new
                URL(getMonitorFromBeacon(beacon)).openConnection();
            connection.setRequestProperty("Accept-Charset",
                "UTF-8");
            answer = connection.getInputStream();
```

```
            updateWhatAreMonitorsDisplaying(beacon, null);
        }
    }
```

Listato 4.3: `CentralServlet`: metodo `doPost()`

## 4.2 Deployment

Per quanto riguarda l'app Android, si sono utilizzati per il deployment un device Nexus 5 con la versione di Android 5.1 Lollipop e un Nexus 7 con Android aggiornato alla versione 5.1.1 Lollipop (quindi tutti e due con API Level 22). Entrambe le tipologie di modelli sono compatibili con il Bluetooth Low Energy e supportano il central role (caratteristiche necessarie per poter realizzare le funzionalità del sistema).

Per il deployment dei due web service (quello dell'applicazione per il sistema collegato allo schermo e quello del server centrale) si è fatto uso di Apache Tomcat per poter agevolmente eseguire le due applicazioni web.

## 4.3 Test e valutazione

Terminata la fase di implementazione si è proceduto al testing del sistema sviluppato. Come prevedibile, l'aspetto più importante è a quale distanza dello studente dal monitor avviene la modifica dei contenuti visualizzati.

Innanzitutto, per quanto riguarda le due comunicazioni HTTP (ovvero la richiesta POST dell'app Android al server centrale, e la richiesta GET di quest'ultimo all'applicazione per il sistema collegato allo schermo) si è rilevato che **impiegano in media un tempo complessivo di poco superiore a 1 s, quindi ampiamente accettabile**. Questo risultato era d'altro canto prevedibile data l'ottima copertura in tutta la sede universitaria della rete wireless di Ateneo.

È quindi evidente, come ci si aspettava, che l'aspetto più critico riguarda la comunicazione tra beacon e device Android (che è quella che avvia l'interazione tra tutte le componenti del sistema), e, quindi, **a quale distanza dallo**

**schermo l'app rileva il segnale del beacon corrispondente**.

Considerando *l'arrivo nelle vicinanze di uno schermo*, **nella maggior parte dei casi il primo segnale è rilevato dal dispositivo Android a circa 6 m dal monitor, che è un buon risultato**, considerato che a tale distanza, se anche non fosse possibile la lettura, lo studente può accorgersi che la modifica della visualizzazione è dovuta alla sua presenza, e facendo pochi passi è in grado di leggere i contenuti dello schermo. Inoltre, lo scenario più probabile è che l'utente interessato a consultare il proprio orario stia camminando verso il monitor, e nel tempo dovuto alle comunicazioni HTTP sia giunto a una distanza tale da rendere possibile la lettura.

Tuttavia il limite più grosso che è stato rilevato dai test è che, trattandosi di onde radio, la distanza a cui possono arrivare i segnali è estremamente variabile e, oltre un certo limite, non prevedibile a priori. Se da un lato è raro il caso in cui esso venga percepito quando lo studente è già abbastanza vicino allo schermo da poterne leggere i contenuti (e quindi debba attendere), si verifica più spesso la situazione opposta, ovvero che **il segnale venga percepito dal device Android a una distanza troppo elevata** (circa 10 m, o talvolta poco di più). Di conseguenza avviene la spiacevole situazione in cui il monitor mostra l'orario di uno studente quando egli è ancora troppo lontano, o addirittura non interessato a consultare le informazioni visualizzate.

Per quanto riguarda invece *l'allontanamento dalle vicinanze di uno schermo*, si è verificato che **il rilevamento dell'uscita dalla `region` del monitor avviene quasi sempre quando l'utente si trova a una distanza superiore a quella a cui mediamente è rilevata l'entrata**. Con ogni probabilità si presume che questo sia dovuto alla Android Beacon Library, la quale, mentre necessita di un solo segnale ricevuto per determinare l'ingresso in una `region`, ha invece bisogno di non rilevare un segnale per un certo periodo di tempo per poter concludere che l'uscita dalla `region` è effettivamente avvenuta.

**Ciò tuttavia non pregiudica l'usabilità del sistema**, poiché la tempistica fondamentale, anche dal punto di vista della *user experience*, è quella della visualizzazione di uno specifico orario. Se, dopo che lo studente ha consultato il contenuto informativo dello schermo, quest'ultimo ripristina la schermata di

default quando egli è già lontano, probabilmente egli non farà nemmeno caso al problema, e soprattutto questo fatto non influisce sullo scopo del sistema.

### 4.3.1 Sviluppi futuri

La conseguenza logica dei risultati e soprattutto dei limiti evidenziati durante il testing è prevedere dei possibili sviluppi per il sistema, che possano in primis migliorare le funzionalità già implementate, e in secondo luogo aggiungerne di nuove.

- Il primo e più importante miglioramento necessario riguarda sicuramente il **raffinamento della distanza a cui lo schermo modifica i contenuti visualizzati**.

    1. Una prima possibilità può essere utilizzare altri modelli di beacon che permettano di abbassare ulteriormente la potenza di trasmissione del segnale. Questo impedirebbe che il monitor mostri l'orario di uno studente quando egli è troppo lontano e diminuirebbe la distanza a cui viene ripristinata la schermata di default: tuttavia potrebbe aumentare il rischio che il segnale, essendo più debole, non venga rilevato tempestivamente quando l'utente è vicino al monitor, causando un'attesa.
    2. Una seconda ipotesi riguarda l'uso di altri modelli di beacon e/o di un'altra libreria tali da fornire una stima della distanza molto più precisa. In questo modo si potrebbe quantomeno, alla prima ricezione di un segnale, verificare che la distanza dal dispositivo trasmittente sia inferiore a un certo valore. Se inoltre la libreria scelta fosse di gran lunga più veloce nell'aggiornamento della stima, allora si potrebbe veramente pensare non solo che lo schermo mostri un orario quando lo studente è sufficientemente vicino ad esso, ma anche che **tale visualizzazione sia mantenuta finché egli rimane entro una certa distanza**.
    3. La terza possibilità, e forse la più interessante, consiste nell'utilizzo integrato di ulteriori tecnologie. Tra tutte si evidenzia l'**Activity Recognition**, funzionalità aggiunta da un paio d'anni ai Google

Play Services: essa permette di rilevare se l'utente è fermo, o se sta camminando, o correndo... Si potrebbe quindi richiedere la visualizzazione di un certo orario se l'utente si trova nelle vicinanze dello schermo ed è fermo. Non sembra invece utilizzabile il GPS in quanto in ambienti interni la sua precisione diminuisce notevolmente.

**Si noti comunque che nessuna delle possibilità elencate esclude le altre, e che un loro uso combinato può portare a grandi miglioramenti.**

- L'altro aspetto critico riguarda **una più robusta gestione dello scenario in cui più studenti sono contemporaneamente nelle vicinanze di uno stesso schermo**. Assumendo di voler utilizzare una logica FIFO, ovvero dare la precedenza a chi è giunto prima nei pressi del monitor, sul server centrale si potrebbe tenere traccia degli utenti in attesa, e appena lo studente di turno si allontana dallo schermo, visualizzare direttamente l'orario scelto dal primo della fila, senza passare dalla schermata di default. Non sembra invece fattibile che se l'orario mostrato dallo schermo corrisponde a quello selezionato da uno studente in arrivo in quel momento, quest'ultimo non venga messo in attesa, in quanto i due orari corrispondono: infatti chi è arrivato per primo non potrebbe scegliere un altro orario da visualizzare (perché magari non interessa all'altro). È quindi più intelligente porre comunque in attesa chiunque arriva in un momento in cui il monitor sta già mostrando un orario: se ad uno studente in attesa interessa proprio quell'orario, lo può comunque consultare pur non essendo il suo turno, e se si allontana prima che tocchi a lui viene rimosso dalla fila.

- Infine tra le varie funzionalità che si potrebbero aggiungere si citano le **ulteriori personalizzazioni delle visualizzazioni**: ciascuno studente potrebbe inserire tramite l'app il proprio orario, comprensivo anche di appelli d'esame, ricevimenti dei docenti... Inoltre si potrebbe dare la possibilità ai docenti di visualizzare sullo schermo l'orario delle lezioni che devono tenere.

# Conclusioni

Fin dall'Introduzione era stato evidenziato come, nel panorama attuale del mondo ICT (e dunque tenendo sempre presente l'ambito dell'Internet of Things), l'aspetto veramente interessante fosse la possibilità di integrare diverse tecnologie. Il caso di studio trattato in questa tesi, ovvero lo sviluppo di un prototipo di schermo adattativo per la visualizzazione dinamica degli orari delle lezioni universitarie, ha consentito di esplorare l'attuale panorama tecnologico in ambito mobile e web, e di coniugare tra loro diverse tecnologie innovative e all'avanguardia.

**Per quanto riguarda Android, i risultati sono stati totalmente positivi**: il sistema operativo ha confermato la sua nota versatilità e la capacità, anche grazie all'aiuto di librerie esterne, di interfacciarsi e dialogare con dispositivi e tecnologie differenti, adattandosi ai più svariati contesti applicativi.

**A livello teorico, il Bluetooth Low Energy è apparsa una tecnologia dalle grandi potenzialità**: i suoi punti di forza, oltre al consumo di energia assai ridotto, sono la grandissima diffusione di cui già gode e, appunto, le ottime possibilità di dialogo con l'ambiente Android.

A livello pratico, si è vista un'applicazione del Bluetooth Smart nel caso dei beacon, che sono stati il fulcro del sistema progettato, ciò che di fatto ha permesso la realizzazione degli aspetti più importanti e innovativi di esso. Riguardo ai beacon, si è visto innanzitutto come abbiano **diverse possibilità di utilizzo e di successo in vari ambiti, ma sempre dipendentemente dal contesto**. Per esempio, il prototipo realizzato è basato su una semplice rilevazione del segnale, poiché come detto non è stato possibile ottenere risultati soddisfacenti stimando la distanza. In altri contesti, tuttavia, si potrebbe per

esempio necessitare di una stima della distanza più precisa, o di altri requisiti.

Allo stato attuale dell'arte, quindi, **i beacon non possono essere considerati come una tecnologia "universale", applicabile allo stesso modo in qualunque situazione**. È logico e giusto sperare nel progredire di questa tecnologia, che è ancora in stato poco più che embrionale: bisogna tenere però presente che essa rimane una tecnologia wireless, e quindi soggetta alle limitazione che caratterizzano le onde radio.

L'opinione di chi scrive è che in futuro si potranno avere significativi miglioramenti per la tecnologia beacon, che ne permetteranno un maggiore impiego. Tuttavia, a prescindere da tali sviluppi, si è potuta consolidare, sia mediante l'attività di ricerca che tramite una "verifica sul campo", l'idea che, attualmente come in futuro, **la chiave di volta, la grande possibilità attuale per l'ICT risieda nell'integrazione di varie tecnologie**. Ciò è vero per i beacon (e lo si è evidenziato non solo nel prototipo, ma anche nelle ipotesi avanzate per gli sviluppi futuri), ma in generale per la maggior parte delle tecnologie. Anzi si potrebbe affermare che al giorno d'oggi la stessa possibilità di successo di ogni tecnologia è in buona parte legata alla capacità di interagire e integrarsi con altre.

# Ringraziamenti

Giunto al termine della tesi, e soprattutto del percorso che mi ha portato alla Laurea, è evidente - già lo era durante questi anni, ora anche più - come non avrei mai potuto raggiungere questo traguardo da solo: tantissime persone mi hanno aiutato sotto ogni punto di vista, umano e professionale. Soprattutto nei momenti più difficili e faticosi ho potuto sperimentare di non essere mai da solo.

Io non amo troppo i ringraziamenti "pubblici", è impossibile dire tutto quello che si prova e per cui si è grati davanti a tutti e in poche righe. Tuttavia queste persone (e mi scuso già se dovessi dimenticare qualcuno) hanno avuto un ruolo talmente importante in questi anni da meritare - oltre ai ringraziamenti in forma personale che certamente avverranno - anche questa modalità più "ufficiale".

Il primo ringraziamento va al relatore della tesi, il Prof. Mirko Viroli. Non è semplicemente una formalità, una cosa "dovuta". A lui sono veramente riconoscente per avermi seguito e consigliato durante il tirocinio e la scrittura della tesi, per la grandissima disponibilità dimostrata durante questi mesi e per avermi permesso di svolgere tirocinio e tesi nel Pervasive Software Lab - ambiente in cui, come dirò tra poco, mi sono trovato benissimo.

Un altro immenso ringraziamento va ad Angelo, correlatore della tesi: per avermi seguito fin dall'inizio in tutte le fasi di scrittura della tesi e di realizzazione del prototipo, per i tanti consigli, per la grandissima disponibilità a chiarire subito ogni dubbio, e per avermi supportato e sopportato, rispondendo sempre alle mie mille domande.

Ringrazio tantissimo Pietro, anche lui mi ha fornito aiuto e preziosi con-

sigli durante il tirocinio e la scrittura della tesi. Un ringraziamento al Prof. Alessandro Ricci e a tutte le altre persone incontrate nel Pervasive Software Lab: le tante ore trascorse insieme sono state per me una grande fonte di arricchimento dal punto di vista non solo professionale, ma anche relazionale e umano. Ringrazio anche tutti quei professori che in questi anni, con il loro impegno ed entusiasmo, mi hanno fatto appassionare alla loro materia.

Un grazie a tutti quelli che mi sono stati vicino in università, in particolare i miei compagni di corso Mattia e Luca con cui ho condiviso tanti momenti tra lezioni, pranzi, studio e preparazione di esami. Sono veramente riconoscente a loro per la compagnia in università, l'aiuto nello studio, e il sostegno soprattutto nelle vicinanze degli esami.

Un immenso ringraziamento va a mio fratello, alle mie sorelle e soprattutto ai miei genitori, per quanto mi sono stati vicino, per tutto l'aiuto dato, per avermi sempre fornito preziosi consigli lasciandomi allo stesso tempo libero di scegliere, per non avermi mai fatto pesare gli insuccessi, e per avermi sostenuto (anche dal punto di vista economico).

Ringrazio tutti coloro che mi sono stati vicino in qualunque modo, anche solo con un messaggio per sapere come stavo o per il tempo di un caffè insieme.

E infine, *last but not least*, un ringraziamento grandissimo e speciale a tutti i miei amici. È impossibile nominarli uno ad uno, ma ci tengo a ringraziare in particolare Lorenzo, Ilaria, Luca, Marco e Samuele, perché sono stati quelli che ho sentito più vicini, che ci sono sempre stati e che mi hanno fatto compagnia in tutte le circostanze: chi è venuto a farmi una sorpresa a casa mentre ero immerso nella tesi, chi mi ha spronato a credere in me quando io stesso ci credevo poco, chi ha condiviso lo studio e ogni altra circostanza quotidiana, bella o brutta... Come già detto, però, la mia gratitudine si estende anche agli altri amici che non ho potuto nominare uno per uno, ma che ho tutti ben presente, e ai quali avrò modo di portare comunque di persona i miei ringraziamenti.

*Francesco*

# Bibliografia


[1] *Android Developers*, https://developer.android.com

[2] *StackOverflow*, http://stackoverflow.com

[3] *Progetto Android Beacon Library*, http://altbeacon.github.io/android-beacon-library

[4] Kuor-Hsin Chang *Bluetooth: A Viable Solution for IoT?*, in IEEE Wireless Communications, Dicembre 2014

[5] Michael Foley *Getting Familiar with Bluetooth 4.0 Low Energy*, in RTC Magazine, Novembre 2014 (http://rtcmagazine.com/articles/view/102266)

[6] *What is iBeacon?*, dal sito beaconsandwich (http://www.beaconsandwich.com/what-is-ibeacon.html)

[7] *What is iBeacon? A guide to Beacons*, dal sito ibeaconinsider (http://www.ibeacon.com/what-is-ibeacon-a-guide-to-beacons)

[8] *The Hitchhikers Guide to iBeacon Hardware: A Comprehensive Report by Aislelabs (2015)*, dal sito Aislelabs, Maggio 2015 (http://www.aislelabs.com/reports/beacon-guide)

[9] Daniel Giusto, Antonio Iera, Giacomo Morabito, Luigi Atzori *The Internet of Things*, Ed. Springer, 2010

[10] Luigi Atzori, Antonio Iera, Giacomo Morabito *The Internet of Things: A survey*, in Computer Networks, 2010