

**ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA
SCUOLA DI INGEGNERIA E ARCHITETTURA**

**CORSO DI LAUREA MAGISTRALE IN
INGEGNERIA INFORMATICA**

TITOLO DELLA TESI:

**SVILUPPO DI UNA INFRASTRUTTURA
LOCATION-BASED PER L'AUTO-
ORGANIZZAZIONE DI SMART-DEVICES**

Elaborato in:

Linguaggi e Modelli Computazionali L-M

Relatore:

Prof. Mirko Viroli

Presentata da:

Andrea Fortibuoni

Sessione I

Anno Accademico 2013 / 2014

Indice

Introduzione	4
1 Le tecnologie location-based.....	8
1.1 GPS.....	9
1.2 NFC	11
1.3 Bluetooth e BLE.....	13
1.3.1 Beacons	15
2 Domini Applicativi	21
2.1 Caso MBL.....	22
2.2 Caso EduBeacons	24
2.3 Caso Rubens House.....	25
3 L'infrastruttura base.....	26
3.1 Spatial Computing.....	26
3.2 Primo prototipo	28
3.2.1 Architettura Logica	30
3.2.2 Parte progettuale.....	32
3.2.2.1 Server.....	32
3.2.2.2 Client.....	34
3.2.2.3 Architettura di progetto.....	39
3.2.3 Il simulatore Java	40
3.2.4 L'esperienza al MAMbo	41
4 Sperimentazione su BLE e GPS	43
4.1 Applicazione GPS	43
4.2 Applicazione BLE	46
4.2.1 Confronto tra beacons	49
5 Infrastruttura Location-based.....	54
5.1 Re-ingegnerizzazione del primo prototipo	54
5.1.1 Descrizione delle nuove entità introdotte.....	55
5.1.2 Progetto delle nuova infrastruttura.....	56
6 Caso di studio.....	63

6.1	Dettagli implementativi	63
6.2	Test e Demo	71
6.3	Altre applicazioni	81
7	Conclusioni e sviluppi futuri.....	82
	Elenco delle figure	85
	Bibliografia e sitografia	89

Introduzione

La diffusione di dispositivi mobili sempre più avanzati e la parallela evoluzione delle tecnologie wireless, ha rivoluzionato sempre più il mondo della comunicazione e lo stile di vita delle persone, permettendo negli ultimi anni lo sviluppo di un elevato numero di servizi strettamente legati alla mobilità degli utenti al fine di migliorare la loro esperienza nelle esigenze quotidiane. La possibilità di avere a bordo del proprio smartphone una serie di componenti hardware sempre più precisi e potenti, ha allargato l'orizzonte di molti sviluppatori software soprattutto in ambito pervasive-computing. Per fare un esempio: mentre si passeggiava per strada, il nostro smart-device potrebbe raccogliere automaticamente le offerte “radiodiffuse” dai negozi elencandole in ordine di convenienza ed indirizzarci sui negozi più interessanti. Inoltre non è più fantascienza che qualsiasi oggetto di uso quotidiano possa essere integrato con un chip per collegarlo ad una rete infinita di altri dispositivi, con lo scopo di creare un ambiente in cui la connettività è incorporata in modo tale da essere discreta e sempre disponibile, a bassi costi e a lunga durata. Un esempio pratico di questo concetto riguarda il settore della domotica: gli elettrodomestici di casa nostra, come lavatrice, lavastoviglie, frigorifero e forno potrebbero “mettersi d'accordo” tra loro per scambiarsi informazioni al fine di gestire meglio i consumi elettrici.

Ed è proprio in questi nuovi scenari “smart” che si vuole focalizzare l'attenzione per lo sviluppo della tesi: ciò che si vuole realizzare è un'infrastruttura che permetta a dispositivi mobili, come i recenti smartphones e tablet Android, di cooperare e auto-organizzarsi al fine di compiere un certo task definito a livello applicativo, sfruttando le moderne tecnologie location-based che questi hanno a disposizione.

Per tale scopo si parte da un'infrastruttura base sviluppata in ambito accademico e i cui concetti derivano dal campo dello Spatial Computing: ogni nodo del sistema non è direttamente a conoscenza di tutti gli altri presenti, ma interagisce solo coi propri vicini in modo da costruire una mappa che tenga traccia di tutti i nodi presenti e relative distanze da sé stesso, allo scopo di auto-organizzarsi

per svolgere un certo task applicativo. Questa primo prototipo dell’infrastruttura sfrutta la nozione di “vicinato” sulla base delle distanze reali (misurate in centimetri) presenti tra le posizioni di una griglia di tag NFC (Near Field Communication). L’intento che si vuole ottenere è quello di separare la gestione specifica della parte relativa a questa tecnologia, modificando quindi la struttura dell’infrastruttura per renderla “aperta” anche a possibili aggiunte di altre tecnologie location-based senza dover poi stravolgere il comportamento dei componenti già sviluppati.

In particolare, dopo una fase di sperimentazione sulle tecnologie di localizzazione presenti sugli smart-devices in dotazione, si è stabilito di affiancare l’uso del GPS e del BLE (Bluetooth Low Energy) alla parte NFC già esistente, generalizzando il concetto di distanza in modo da renderla univoca per qualsiasi tipo di tecnologia (scegliendo l’uso delle coordinate terrestri). Inoltre si è deciso di gestire in maniera dinamica il passaggio da una tecnologia all’altra, sulla base di ciò che i dispositivi rilevano dai propri sensori al fine di raggiungere nel modo migliore possibile l’obiettivo fissato da un certo task applicativo.

Il goal della tesi è quindi quello di creare una infrastruttura che mantenga il modello tipico dello Spatial Computing e coniungi al meglio le tre tecnologie scelte per la localizzazione.

A dimostrazione dei possibili utilizzi dell’infrastruttura sviluppata si presenterà la realizzazione di un caso applicativo reale, il quale permetterà all’utente di trovare un determinato punto di interesse sfruttando le tre tecnologie location-based scelte: il GPS per la localizzazione outdoor, il BLE per muoversi in ambienti indoor, e l’NFC per spostarsi in un ambiente ancora più circoscritto come una griglia di posizioni posta sopra un tavolo. L’infrastruttura ha anche un altro scopo, quello di facilitare lo sviluppo futuro di nuovi applicativi Android: per questo motivo essa viene strutturata in moduli specifici, mantenendo separata la gestione delle specifiche tecnologie a disposizione sugli smart-device e suddividendo la parte applicativa da quella comunicativa.

Panoramica dei contenuti

Primo capitolo

Nel primo capitolo della tesi vengono descritte le principali tecnologie attualmente disponibili per la localizzazione, con particolare focus su quelle che si decide di utilizzare per lo sviluppo dell'infrastruttura: GPS, NFC e BLE. Proprio in ambito Bluetooth Low Energy verranno presentati anche i "Beacons", piccoli trasmettitori wireless che sfruttano questa tecnologia per emettere periodicamente informazioni nell'ambiente in cui sono collocati e senza il bisogno di stabilire eventuali connessioni per la comunicazione.

Secondo Capitolo

Nel secondo capitolo si mostrano i potenziali domini applicativi in cui queste tecnologie location-based possono trovare utilizzo, e in particolare si illustrano tre esempi di utilizzo pratico dei beacons BLE: un caso inerente alla Major League di Baseball Americana, uno relativo all'ambito educativo (caso EduBeacons), e uno riguardante il caso del museo Rubens House di Anversa.

Terzo Capitolo

Nel terzo capitolo della tesi si descrive l'infrastruttura da cui si parte e che rappresenta la base sulla quale integrare le tecnologie utili alla localizzazione indoor e outdoor, andando a presentare i concetti teorici sulla quale essa si fonda (ovvero lo Spatial Computing), e le caratteristiche architetturali e implementative di questo primo prototipo.

Quarto Capitolo

Nel quarto capitolo viene presentata la parte sperimentale relativa alle nuove tecnologie da integrare nell'infrastruttura base, in modo da avere un reale riscontro su effettive funzionalità e caratteristiche. In particolare si effettua un confronto tra le varie tipologie di beacons di cui si è a disposizione, anche in relazione agli smart-devices che si hanno in dotazione per lo sviluppo della tesi.

Quinto Capitolo

Nel quinto capitolo si illustra come si è deciso di modificare la struttura dell’infrastruttura base per poter ampliare gli scenari di utilizzo e integrare quindi GPS e BLE, andando a descrivere le nuove entità introdotte e mostrando come viene gestita la parte inerente alle tecnologie.

Sesto Capitolo

Nel sesto capitolo viene descritto un caso applicativo reale che mostra le potenzialità della nuova infrastruttura sviluppata. L’applicazione realizzata permetterà all’utente di trovare un determinato punto di interesse sfruttando la tecnologia GPS per la localizzazione outdoor, quella BLE per muoversi all’interno dell’edificio, e infine quella NFC per spostarsi da una posizione all’altra di un particolare tappeto posto su di un tavolo.

Settimo Capitolo

Nel settimo capitolo vengono presentate le conclusioni raggiunte al termine del lavoro svolto e gli eventuali sviluppi futuri riguardanti sia l’infrastruttura, sia il mondo dei beacons.

Capitolo 1

Le tecnologie location-based

Un ambito tecnologico che negli ultimi anni sta riscuotendo un interesse sempre più marcato è quello della localizzazione, in particolare quella indoor, dove occorre avere una tecnologia più precisa rispetto a quella basata sul GPS (Global Positioning System), per guidare le persone verso determinati punti di interesse con un margine d'errore di pochi metri. Sono molte le tecnologie che in questo ambito si sono sviluppate nel corso degli anni distinguendosi per le differenti caratteristiche e peculiarità, fra le quali si annoverano:

- tecniche di Dead Reckoning: stimano la posizione corrente sulla base del calcolo degli spostamenti da un punto iniziale noto, sfruttando direzione e velocità dell'utente acquisite tramite sensori inerziali (come accelerometro e bussola) in uno specifico intervallo di tempo;
- rete cellulare: non fornisce un'informazione sufficientemente accurata per consentire la navigazione in ambienti chiusi e circoscritti;
- sound-based: impiegano gli ultrasuoni per misurare la distanza fra trasmettitore e ricevitore;
- infrarossi: l'utente indossa un “badge” che emette segnali infrarossi e che vengono ricevuti dai sensori presenti nell'ambiente circostante;
- onde radio (come WI-Fi e Bluetooth): localizzano l'utente misurando l'intensità del segnale ricevuto (Received Signal Strength Indicator, o RSSI) dai nodi presenti in un'infrastruttura;
- RFID o NFC: tecniche che consentono il rilevamento di prossimità grazie a particolari tag elettro-magnetici;
- sistemi ottici: si compiono operazioni di analisi su immagini acquisite dal device dell'utente e inviate periodicamente a un webserver centrale, le quali saranno poi confrontate con quelle presenti nel database in modo da restituire la posizione corrente;
- codici QR (Quick Response): sono codici a barre bidimensionali che la maggior parte dei telefoni di ultima generazione è in grado di decodificare.

La scelta delle tecnologie da utilizzare per consentire una localizzazione raffinata e utile allo scopo di questa tesi, si è basata su diversi fattori: accuratezza, stabilità e potenza del segnale, sensibilità ai disturbi, facilità di utilizzo, possibilità e semplicità di sviluppo su smart-devices Android, eventuali costi d'acquisto e installazione. Per questo motivo si è scelto di scartare tecnologie wireless caratterizzate da segnali instabili e poco accurati (come Wi-Fi o rete cellulare), complicate e con un ampio margine d'errore (come le tecniche di Dead Reckoning) o troppo costose (come sistemi ottici).

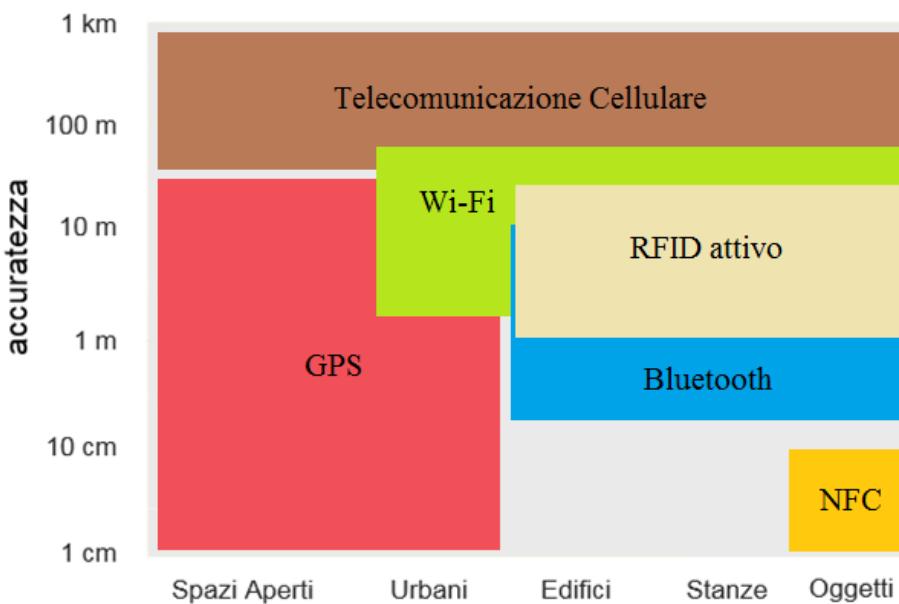


Figura 1.1: Confronto sulla precisione delle principali tecnologie location-based.

Si è dunque deciso di puntare su 3 tecnologie principali: NFC, Bluetooth Low Energy (o BLE) e GPS. In questo modo si è cercato di unire le potenzialità che queste offrono, ricoprendo tutti i “difetti” che hanno se le si utilizza singolarmente per la localizzazione: NFC per avere una precisione al millimetro ma limitata a un raggio corto (2-3 cm), BLE per scenari indoor a medio raggio (50-60 metri) con precisione variabile nell'intorno di pochi metri, e GPS per scenari outdoor su ampio raggio.

1.1 GPS

La tecnologia GPS è stata sviluppata negli anni '70 dal Ministero della Difesa Americano, ma divenne pienamente operativo solo a metà di quelli '90. Essa sfrutta i satelliti in orbita intorno alla Terra e si basa sul metodo della

trilaterazione (o posizionamento sferico): una volta che un dispositivo riceve da almeno 4 satelliti il segnale radio che questi trasmettono, calcola il tempo impiegato da ogni segnale per percorrere la distanza satellite-ricevitore e trovare così le coordinate terrestri precise in cui si trova. Il GPS ha rivoluzionato il mondo della localizzazione, permettendo lo sviluppo di applicazioni sempre più precise come mappe per la navigazione assistita e itinerari outdoor. Nei moderni smart-devices si sfrutta un sistema più complesso di posizionamento che si chiama A-GPS (Assisted Global Positioning System), il quale combina l'uso di reti telefoniche e Wi-Fi al GPS per velocizzare la localizzazione, dato che ogni cella di telefonia mobile presente sul territorio ha una posizione fissa nota e quindi può comunicare ai dispositivi la lista dei satelliti in vista istante per istante ottenuti da un Server.

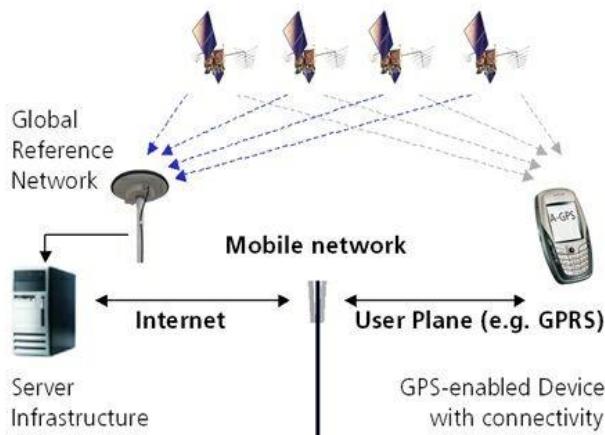


Figura 1.2: Esempio di localizzazione via A-GPS.

Il sistema GPS ha un errore di precisione che rientra nell'ordine dei 5-10 metri, dovuto sia alla precisione dei rilevatori elettronici presenti nei ricevitori dei dispositivi, sia da fattori atmosferici che degradano il segnale prima che raggiunga tali ricevitori. Inoltre non risulta adatto per scenari di localizzazione indoor come interni di edifici, dove non è possibile ottenere il segnale che si riceve direttamente dai satelliti.

Negli ultimi anni tutti gli smart-devices presenti in commercio hanno integrato al loro interno un ricevitore GPS che consente la geolocalizzazione della propria posizione. Addirittura particolari modelli di ricevitori GPS per uso civile in campo ingegneristico hanno ottenuto la possibilità di usufruire del secondo

canale (denominato “L2”) prima riservato solo per scopi militari, permettendo così di raggiungere un margine di precisione centimetrico¹.

1.2 NFC

Il Near Field Communication (o NFC) è una tecnologia che fornisce connettività wireless a corto raggio (2-4 cm), rilasciata nel 2004 dalle aziende Nokia, Philips e Sony. Deriva dalla tecnologia wireless RFID (Radio Frequency Identification), ma con la differenza di consentire una comunicazione bidirezionale: quando due apparecchi NFC vengono posti a contatto (o accostati entro un raggio di 4 cm), viene creata una rete peer-to-peer tra i due in modo da poter inviare e ricevere informazioni. In questo modo non ho più la separazione fra “trasponder” (o tag) e “reader” tipica dell’RFID, e neanche la suddivisione fra tag passivi (che devono essere “letti” da un reader) e attivi (che comunicano direttamente fra loro).



Figura 1.3: A sinistra un tag NFC, a destra un esempio di lettura di tag RFID passivi.

Ogni tag NFC ha un proprio ID univoco, e sul mercato sono presenti diverse categorie di tag che si distinguono per: “capacità complessiva” (ovvero la memoria totale del tag), “memoria disponibile” (ovvero l’effettiva memoria di cui si può disporre per scrivere dati sul tag), numero massimo di caratteri che può contenere un link URL memorizzato sul tag, numero massimo di caratteri di cui si può disporre per scrivere un messaggio testuale sul tag, crittografia e compatibilità.

Le tre categorie principali presenti attualmente sul mercato e le relative caratteristiche possono essere mostrate nella seguente tabella:

¹ http://it.wikipedia.org/wiki/Sistema_di_Posizionamento_Globale

	Tag Mifare	Tag Ultralight	Tag NTAG203
Capacità complessiva	1024 byte	64 byte	168 byte
Memoria disponibile	716 byte	46 byte	137 byte
Lunghezza URL	256 caratteri	41 caratteri	132 caratteri
Lunghezza testo	709 caratteri	39 caratteri	130 caratteri
Crittografia	Crypto-1	No	No
Perché sceglierlo	Necessità di alto storage o per V-Card (tessere, biglietti da visita).	URL brevi, affissioni Smart, campagne NFC.	Ideale per qualsiasi uso; consigliato soprattutto per campagne di marketing.

Un'altra caratteristica importante è la possibilità di poter modificare in qualsiasi momento cosa è scritto sul tag NFC, a meno di non rendere il tag read-only (operazione irreversibile). In genere questa funzionalità è fornita solo nei tag del tipo Ultralight. Il costo d'acquisto dei tag NFC varia quindi in base alle caratteristiche appena descritte, e oscilla attorno ai 2 euro.

Campi di applicazione emergenti NFC sono:

- p2p payment, ovvero gli utenti possono effettuare pagamenti semplicemente avvicinando i propri smartphones e inserendo la cifra da trasferire;
- identificazione personale in cantieri edili o aziende, e timbratura del turno di lavoro;
- chiavi elettroniche per aprire le porte di casa e della macchina;
- ubiquitous information applications, come guide audio per musei, librerie, noleggio cd e dvd, ecc.;
- health and safety applications, come monitoraggio di acqua, diete, pressione del sangue, ecc.;
- social networking e marketing, mediante scambio dei propri contatti o altre informazioni memorizzate sui tag;
- smart mobility, come l'accesso a biciclette o servizi specifici in città, e mobile ticketing per il trasporto pubblico (treni, metro, autobus), cinema, concerti ed eventi sportivi.

Una annotazione riguarda i devices di Apple: la compagnia ha deciso di non implementare questa tecnologia all'interno dei propri smartphones, puntando invece sul Bluetooth Low Energy per ricoprire le stesse funzionalità dell'NFC.

1.3 Bluetooth e BLE

Il Bluetooth è uno standard tecnologico nato a metà degli anni '90 per agevolare la trasmissione wireless di dati fra dispositivi mobili attraverso radio-frequenze a corto raggio. La specifica Bluetooth è stata sviluppata in primis da Ericsson e in seguito formalizzata dalla Bluetooth Special Interest Group (SIG), ovvero un insieme di società fondata nel 1999 dalla stessa azienda svedese e altre come IBM, Intel, Toshiba, Nokia, Motorola.

Nel corso degli anni si è sempre più raffinato nelle varie specifiche rilasciate, sia dal punto di vista della potenza e della velocità di trasmissione, sia nell'affidabilità, robustezza e consumo energetico: dalla versione 1.0 del 1999 si sono susseguite la 2.0 nel 2004, la 2.1 nel 2007 e la 3.0 nel 2009, fino ad arrivare a quella attuale 4.0 rilasciata nel luglio del 2010. Proprio verso la fine dell'anno scorso (dicembre 2013) il Bluetooth SIG ha annunciato il rilascio a breve di una nuova specifica 4.1 che sostanzialmente estenderà le capacità della precedente.

Rispetto alla tecnologia NFC che permette un raggio d'azione di pochi centimetri, quella Bluetooth consente un raggio d'azione più ampio (50-70 metri effettivi), a discapito di una peggiore precisione (nell'ordine di un paio di metri). Approfondendo la versione di nostro interesse, ovvero quella 4.0, essa presenta un'importante novità rispetto allo standard classico detto anche Bluetooth Basic Rate utilizzato fino ad allora: dentro di sé infatti racchiude una nuova tecnologia a basso consumo energetico chiamata Bluetooth Low Energy (BLE) e commercializzata col nome di "Bluetooth Smart", incompatibile con il precedente standard. Questa tecnologia fu progettata adottando le specifiche dello standard Nokia denominato Wibree e rilasciato già nel lontano 2006, il quale consentiva di operare alla stessa frequenza della banda caratteristica del Bluetooth Basic Rate (ovvero 2.4 GHz ISM) ma con una bit-rate inferiore (un massimo di 1 Mbps a fronte dei 3 Mbps dello standard classico). Il vantaggio principale introdotto da questa tecnologia risiedeva nei consumi energetici,

ridotti fino a 100 volte meno rispetto allo standard classico (0.01 mW), permettendo quindi a dispositivi dotati anche solo di semplici batterie coin-cell al Litio di durare per mesi o addirittura anni.

Come già detto precedentemente il Bluetooth Smart fu poi incorporato nella versione 4.0 del Core-Specifications e divenne standard nel 2010. I primi dispositivi commerciali (computer e smartphones) dotati della versione 4.0 furono immessi sul mercato solo dalla seconda metà del 2011, primi fra tutti Apple con il suo iPhone 4S. Solo di recente gli altri sistemi operativi come Android (dalla versione 4.3 in poi), Windows Phone (dalla versione 8), e BlackBerry (dalla versione 10) hanno introdotto nativamente sui loro dispositivi il supporto Low Energy.

Il BLE può considerarsi quindi un vero e proprio nuovo standard che presenta un diverso protocollo e una diversa struttura di base rispetto al Bluetooth Basic Rate: l'architettura infatti non è più simmetrica, ma consente di avere ruoli diversi (Central vs Peripheral) e quindi di avere periferiche semplici, poco costose e con bassissimo consumo di energia. Se con lo standard classico si potevano avere solamente i ruoli di Master e Slave, con il BLE si possono distinguere 4 ruoli principali:

- Broadcaster: si trasmettono segnali periodici, senza la possibilità di ricezione;
- Observer: si ricevono segnali provenienti da un Broadcaster, senza possibilità di trasmettere;
- Peripheral: permette il broadcasting e può anche ricevere, consentendo la connessione a altri dispositivi ma solo in modalità Slave;
- Central: riceve i segnali dei Broadcaster e in più può trasmettere, permettendo di connettersi verso uno o più Peripheral iniziando lui la connessione e quindi operando in modalità Master.

Mentre i dispositivi che implementano il Bluetooth Basic Rate possono essere suddivisi in tre classi a seconda della loro potenza in uscita (100, 2.5, e 1 mW), quelli Low Energy non hanno classi ma solo specifiche sui valori di potenza massima e minima del trasmettitore (da 10 a 0.01 mW).

La specifica 4.0 permette comunque ai dispositivi di implementare sia sistemi “classici” sia low-energy, e quelli che li implementano entrambi sono noti come dispositivi “dual-mode” o Smart Ready. Quelli che implementano solo il Bluetooth Smart e quindi solo la tecnologia low-energy sono noti come dispositivi “single-mode”: essi hanno il vantaggio di avere costi e consumi ridotti, ma lo svantaggio di non poter comunicare con i dispositivi che adottano il Bluetooth Basic Rate.

Per questo motivo i chip single-mode sono spesso utilizzati per dispositivi low-cost alimentati a batteria e che fungono principalmente da Broadcaster o Peripheral, mentre quelli dual-mode sono impiegati sempre più in smartphones, tablets, e laptops.

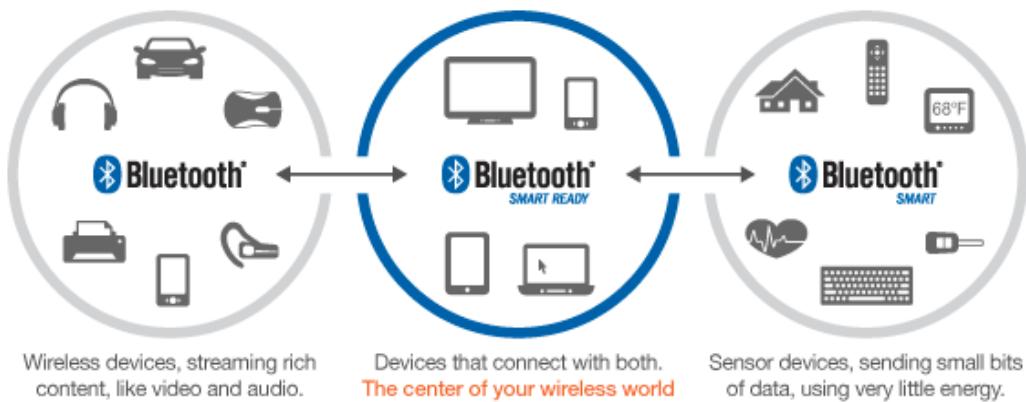


Figura 1.4: Compatibilità tra le diverse tipologie di Bluetooth.

La lista dei dispositivi che sfruttano la tecnologia Smart e Smart Ready è in continuo aumento, tanto che il Bluetooth SIG prevede una copertura Low Energy sul 90% degli smartphones in circolazione entro l’anno 2018².

1.3.1 Beacons

Come visto nel paragrafo precedente, i campi di applicazione del BLE sono in continuo aumento e molte aziende hanno esaminato la possibilità di utilizzarlo per scopi commerciali, visualizzando ad esempio informazioni (come offerte, promozioni, avvisi ad hoc) sugli smartphones dei clienti quando essi entrano nel raggio di un trasmettitore BLE fisso. L’inconveniente che si aveva con il Bluetooth classico era il non poter trasmettere messaggi a dispositivi

² Bluetooth Special Interest Group, "Mobile Telephony Market", January 16, 2014.

sconosciuti, a causa di dover identificare prima la presenza di ogni singolo device e poi inviare un messaggio mirato. Col Bluetooth Smart invece si sfrutta il modello di advertising caratteristico del ruolo di Broadcaster, che permette a un dispositivo di inviare ripetutamente informazioni racchiuse in piccoli pacchetti e senza preoccuparsi di stabilire una connessione: è così che nasce l’idea di “beacon”.

Inizialmente fu Apple a lanciare nel giugno 2013 un primo prodotto chiamato “iBeacons”, che permetteva agli smartphones dotati di iOS 7 di notificare la loro presenza ai dispositivi BLE nelle vicinanze (con possibilità quindi di agire sia da Central, sia da Peripheral). Di lì a poco altre aziende incominciarono a produrre dispositivi semplici e a basso costo che potessero fungere da Peripheral, prima fra tutte Estimote³. Altre aziende che hanno deciso di seguire questa strada sono state: Stick N Find, Qualcomm, BlueCats, BlueSense Networks, Accent Advanced Systems, Kontakt, Radius Networks, Sensorberg, Roximity, Gelo, Glimworm, Swirl. Il costo d’acquisto attuale dei beacons forniti da queste aziende è più o meno lo stesso e si è assestato a circa 25 euro l’uno, anche se in genere vengono venduti non singolarmente ma in confezioni da tre o quattro elementi.

La lista delle aziende produttori di beacons è in continua crescita, e da come si può notare la maggior parte sono americane mentre poche quelle in Europa e addirittura nessuna in Italia. Visto le richieste sempre più crescenti da parte di developers e clienti interessati, non è difficile pensare che questo tipo di tecnologia si diffonderà a breve anche sul nostro territorio nazionale. Soprattutto per i vantaggi che se ne potrebbero trarre in scenari come localizzazione indoor, proximity marketing, smart-mobility, domotica e eventi sportivi. Scopo di questa tesi è anche quello di mostrare l’utilità e le potenzialità di questa nuova tecnologia.

Avendo essenzialmente il compito di ricoprire il ruolo di Peripheral, un beacon deve occuparsi solo di emettere informazioni ad intervalli di tempo prefissati e può quindi essere facilmente costruito utilizzando un chip BLE e relativa antenna, un processore integrato (in genere un 32-bit ARM) con relativa RAM,

³ Sito ufficiale: <http://estimote.com>.

e una batteria che gli consenta di funzionare per un lasso di tempo accettabile (i fattori che incidono sulla durata della batteria verranno poi spiegati nel corso del capitolo).



Figura 1.5: Composizione di un beacon Estimote.

In questo modo si possono produrre beacons a basso costo e dalle dimensioni ridotte, quindi poco ingombranti e molto adatti a essere collocati in spazi poco visibili (ad esempio nei negozi possono essere fissati su muri, nascosti sotto tavoli, lampade o altri oggetti). Nella figura 1.6 vengono mostrate meglio le effettive dimensioni dei beacons attualmente in commercio e prodotti dalle principali aziende citate prima.



Figura 1.6: Dimensioni di alcuni beacons delle principali aziende.

Una annotazione riguarda i dispositivi Android, che sappiamo fornire compatibilità al BLE dalla versione 4.3: al momento non viene supportata la modalità Peripheral ma si ricopre solo il ruolo Central, dunque non è possibile fare advertising o utilizzare tali smart-devices per simulare beacons “virtuali” (cosa invece possibile su iOS 7). Per ovviare a questa mancanza esiste anche la

possibilità di creare semplici beacons “fai-da-te” a basso costo, utilizzando una scheda Raspberry Pi e un dongle BLE USB: sfruttando il sistema operativo Linux e lo stack software BlueZ, si può collegare il Raspberry munito di dongle e da linea di comando abilitare l’advertising. Un’altra soluzione simile utilizza una scheda Arduino con chip BLE integrato.

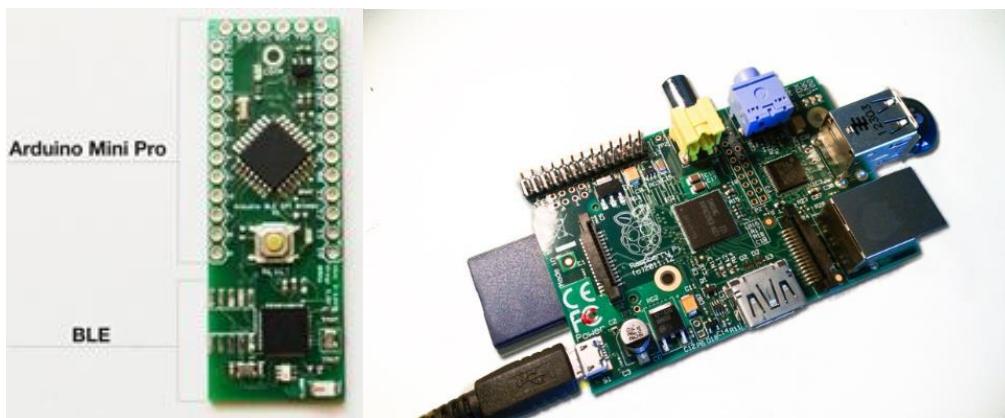


Figura 1.7: A sinistra una scheda Arduino BLE, e a destra un beacon fai-da-te costruito con scheda Raspberry Pi.

Un’alternativa per riuscire a fare advertising BLE riguarda l’uso di OS X Mavericks come beacon “virtuale”, sia su dispositivi Apple recenti (e quindi con chip BLE già a bordo), sia su Mac non dotati di chip BLE ma a cui è possibile collegare un dongle BLE USB: attraverso una semplice applicazione facilmente reperibile in rete⁴ si può infatti simulare un vero e proprio beacon. Inoltre è possibile percorrere tale strada anche su altri sistemi operativi come Windows, semplicemente utilizzando una Virtual Machine e l’immagine virtuale di Mavericks (sempre attraverso l’uso dell’immancabile dongle BLE e dell’applicazione citata in precedenza).

⁴ <https://github.com/mttrb/BeaconOSX>, Matthew Robinson, November 2013.

The left side of the image shows the Xcode interface with the 'BeaconOSX' project selected. The 'BLCAAppDelegate.m' file is open, displaying code for initializing a beacon with proximity UUID, major, minor, and measured power. The right side shows a window titled 'BeaconOSX' with fields for proximity UUID (12345678-1234-1234-1234-123456789000), major (5), minor (5000), and measured pwr (-50). A 'Start Broadcasting iBeacon' button is at the bottom.

```

// BLCBeaconAdvertisementData.m
// BeaconOSX
// Created by Matthew Robinson on 1/11/2013.
#import "BLCBeaconAdvertisementData.h"

@implementation BLCBeaconAdvertisementData

- (id)initWithProximityUUID:(NSUUID *)proximityUUID major:(uint16_t)major minor:(uint16_t)minor measuredPower:(int8_t)pwr {
    self = [super init];
    if (self) {
        self.proximityUUID = proximityUUID;
        self.major = major;
        self.minor = minor;
        self.measuredPower = pwr;
    }
    return self;
}

- (NSDictionary *)beaconAdvertisement {
    NSString *beaconKey = @“kCBAAdvDataAppleBeaconKey”;
    unsigned char advertisementBytes[21] = {0};

    [self.proximityUUID getUUIDBytes:(unsigned char *)advertisementBytes];
    advertisementBytes[16] = (unsigned char)(self.major >> 8);
    advertisementBytes[17] = (unsigned char)(self.major & 255);

    advertisementBytes[18] = (unsigned char)(self.minor >> 8);
    advertisementBytes[19] = (unsigned char)(self.minor & 255);

    advertisementBytes[20] = self.measuredPower;

    NSMutableData *advertisement = [NSMutableData dataWithBytes:advertisementBytes length:21];
    return [NSDictionary dictionaryWithObject:advertisement forKey:beaconKey];
}

@end

```

Figura 1.8: Come si presenta l'applicazione utilizzata per simulare un beacon su OSX Mavericks.



Figura 1.9: Esempi di dongle BLE USB.

Passiamo ad approfondire ciò che effettivamente viene trasmesso in questi pacchetti di advertising. Le informazioni rilevanti sono quattro:

- UUID: un codice univoco a 128 bit utile per esempio a differenziare i beacons di un certo negozio da quelli degli altri negozi;
- Major: un valore di 16 bit, utile per esempio a differenziare i beacons delle diverse aree in cui è suddiviso il negozio;
- Minor: un valore di 16 bit, utile per esempio a differenziare i singoli beacons di una stessa area del negozio;
- TX Power: un valore di 16 bit che rappresenta la potenza di trasmissione, utile per capire a che distanza ci si trova dal beacon (conforme alla metrica Received Signal Strength Indication, detta anche RSSI e misurata in dBm).

La precisione del segnale BLE dipende da vari fattori, fra cui le attenuazioni dovute a ostacoli fisici come ad esempio mura e soprattutto elementi contenenti acqua, come il corpo umano.

I parametri che incidono sulla durata della batteria dei beacons, sono potenza del segnale e intervallo di advertising. Riguardo la prima (detta anche “broadcasting power”), una maggiore potenza del segnale significa una maggiore energia irradiata attraverso l’antenna del beacon e una maggiore distanza entro cui gli smart-devices saranno in grado di captare il segnale, oltre al conseguente consumo maggiore della batteria. In genere una potenza pari a -30 dBm (circa 0.001 mW) consente di avere un range pari a circa 2-3 metri, mentre una potenza pari a +4 dBm (circa 2.5 mW) estende il range fino a circa 70 metri (il massimo impostabile). Per quanto riguarda l’intervallo di advertising, più frequentemente si broadcastano le informazioni maggiore è la possibilità che il segnale venga captato e maggiore è la quantità di dati raccolti dal ricevitore: in questo modo più dati si hanno a disposizione, più sarà accurata la distanza stimata dal beacon (particolarmente utile per l’indoor positioning). Su tali aspetti i beacons prodotti delle varie aziende adottano diverse configurazioni, ma in genere i parametri di default si assestano intorno a una potenza pari a -12 dBm e un intervallo di advertising pari a 200 ms, consentendo alla batteria di durare per molti mesi. Di seguito una figura che mostra tali aspetti⁵:

		Broadcasting power		
		-30 dBm [low]	-4 dBm	+4 dBm [high]
Advertising interval	2000 ms [long]	3.3 years	3 years	2.3 years
	1000 ms	1.9 years	1.7 years	1.3 years
	600 ms	1.2 years	1 year	300 days
	200 ms	160 days	140 days	104 days
	50 ms [short]	40 days	35 days	26 days

⁵ Dati tratti dal Blog ufficiale Estimote (<http://blog.estimote.com/>).

Capitolo 2

Domini Applicativi

I domini applicativi su cui ci si può incentrare racchiudono tutti i contesti visti prima per le tecnologie scelte. In particolare annoveriamo quelli in maggiore ascesa nel mondo di oggi:

- Eventi sportivi o musicali: ad esempio schermi pervasivi usati per effetti luminosi, localizzazione di servizi (bagni, ristorazione, ecc.) o di persone (amici o famiglia).
- Negozi o supermercati: localizzazione di prodotti o di persone, pagamenti (un esempio già in uso è rappresentato dal PayPal beacon⁶), e informazioni su eventuali casse o camerini liberi, scale inagibili, ecc.
- Musei o librerie: localizzazione di determinate opere, tour e guide dinamiche.
- Smart-city mobility: verificare eventuali ostacoli vicini, assenza/presenza di servizi, ecc.

Un esempio pratico è il seguente: un negozio vuole fornire la possibilità alle persone che posseggono uno smart-device BLE-compatibile di informarli sulle nuove offerte proposte o sui nuovi arrivi, quando queste si avvicinano entro un certo range (20-30 metri dal negozio); inoltre, una volta all'interno del negozio, si vuole fornire la possibilità di guidarli verso la locazione dell'oggetto di interesse e in caso d'acquisto poter effettuare pagamenti anche contact-less, ovvero completati con transazioni automatiche dal proprio smartphone (in combinazione con piattaforme di pagamento online come PayPal) e senza l'uso di monete, banconote o carte di credito.

Tutto questo viene reso possibile senza troppi costi aggiuntivi solo con l'utilizzo di pochi beacons ben disposti nella struttura del negozio e di una applicazione che gli utenti possono installare sui propri smart-devices. Il risultato ottenuto è quello di rendere più “smart” l'esperienza commerciale, sia dal punto di vista

⁶ <https://www.paypal.com/us/webapps/mpp/beacon>

del cliente sia da quello del venditore, aumentando la qualità dei servizi forniti e il valore aggiunto.

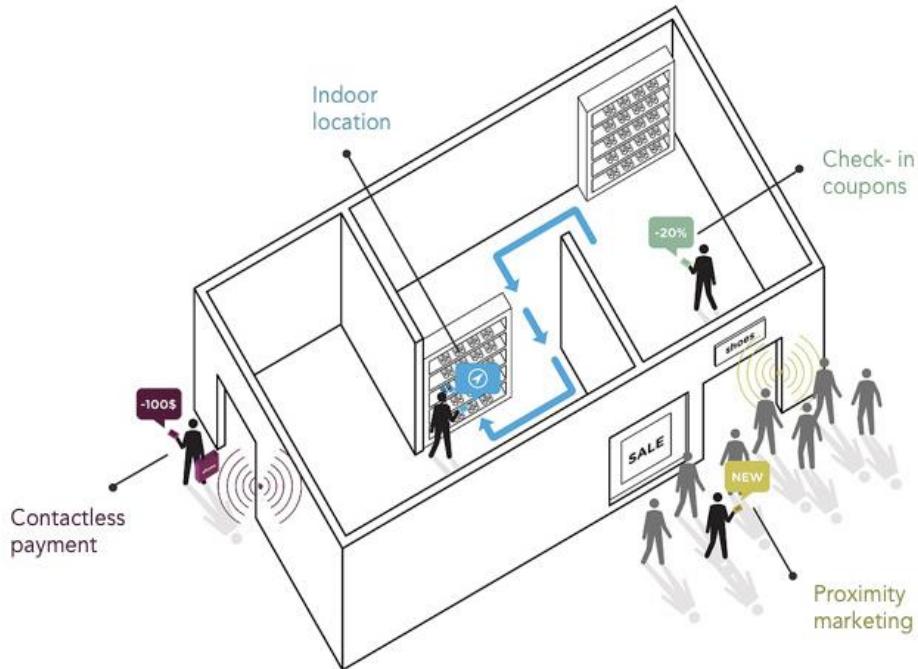


Figura 2.1: Esempio di utilizzo applicativo dei beacons.

Decidendo quindi di focalizzarci in particolar modo sulla tecnologia BLE e sull'utilizzo pratico di questi beacons, si possono mostrare vari casi reali caratteristici dei differenti domini applicativi presentati prima.

2.1 Caso MBL

Nella Major League di Baseball Americana (MLB) si è recentemente deciso di introdurre la tecnologia BLE nello stadio Citi Field del quartiere Queens a New York, la famosa casa dei Mets, allo scopo di fornire servizi location-based per i possessori di smartphone Apple che montano iOS 7. In particolare sono stati installati molteplici beacons in diverse zone dello stadio e si è sviluppata una apposita applicazione fruibile gratuitamente dall'AppStore. In questo modo, assicurandosi di avere tale applicazione lanciata, avvicinandosi all'ingresso dello stadio si riceve subito un messaggio di benvenuto e proseguendo verso il museo "Home Run Apple" dei Mets si può visualizzare sullo schermo un video che ritrae la storia gloriosa di questa squadra.

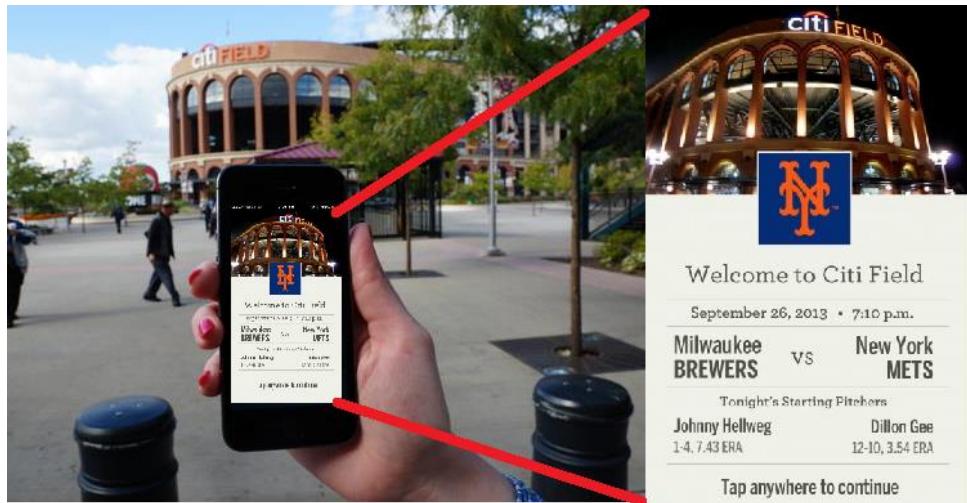


Figura 2.2: Esempio che mostra la posizione in cui l'utente riceve le informazioni di benvenuto al Citi Field Stadium di New York.

Una volta entrati dentro la struttura all'interno dello stadio, si viene avvisati con un altro messaggio su eventuali offerte relative ai negozi interni alla struttura, tenendo conto anche della possibilità di memorizzare quante volte l'utente è stato allo stadio e offrire quindi sconti mirati come coupon speciali. Inoltre si può interagire con l'applicazione per indirizzare gli utenti verso i loro posti, scegliendo il percorso più vicino in base alla loro attuale posizione.

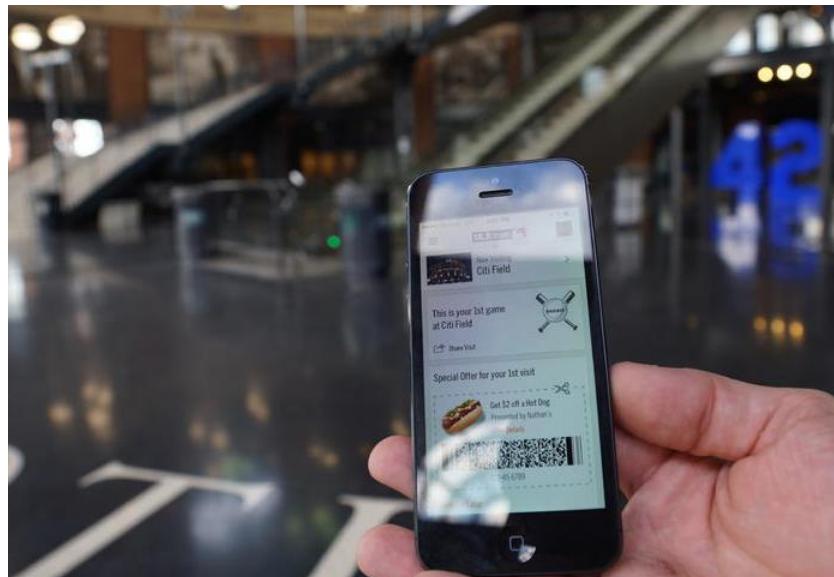


Figura 2.3: Esempio di promozione per nuovi visitatori visualizzata all'ingresso.



Figura 2.4: Esempio di localizzazione del proprio posto a sedere.

2.2 Caso EduBeacons

EduBeacons è un'applicazione sviluppata da un'azienda americana denominata “Vektor Digital” insieme a un insegnante esperto di tecnologie di nome Paul Hamilton⁷, per permettere agli studenti più piccoli (bambini di 5-6 anni di età) di visualizzare il programma di una determinata lezione su un iPad Apple. Quando questi studenti entrano in una classe appositamente adibita con vari beacons posti nelle diverse zone della classe, ricevono su tale tablet dapprima un messaggio di benvenuto e successivamente quando si avvicinano a pochi metri dal singolo beacon compare un piano della specifica lezione interattiva quotidiana.



Figura 2.5: Esempio che mostra l'utilizzo dell'applicazione EduBeacons.

⁷ Sito ufficiale: <http://www.appsbypaulhamilton.com>.

Nel caso particolare della classe di Paul, egli ha predisposto 3 differenti beacons marcati Estimote nell'aula, in corrispondenza di tre diverse aree di apprendimento: arte, tecnologia e librerie⁸.

2.3 Caso Rubens House

Rubens House è la casa del pittore fiammingo dallo stile barocco Pieter Paul Rubens, situata nella città di Anversa e adibita a museo dal 1946. Recentemente il museo ha introdotto l'uso dei beacons Estimote per unire arte e tecnologia al fine di fornire un servizio innovativo ai propri visitatori, che sia al tempo stesso coinvolgente ed educativo. L'artefice di questa innovazione è un'azienda denominata Prophets⁹, la quale ha sviluppato per conto del museo un'applicazione mobile che sfrutta diversi di questi beacons e la tecnologia iBeacon di Apple per interagire coi visitatori, fornendo varie features tra cui:

- prossimità: ad esempio entrando nel cortile viene illustrata l'evoluzione architettonica della veranda, mentre stando in piedi o camminando vicino a un ritratto di famiglia Rubens viene lanciato un puzzle semplice che permette di ricostruire l'albero genealogico;
- distanza: localizzazione di specifiche parti o zone della collezione d'arte, e grazie alle mappe indoor precaricate è possibile avere una guida turistica virtuale del museo;
- notifica: vengono forniti contenuti educativi in base alla posizione del visitatore all'interno del museo, compresi alcuni fatti divertenti e una splendida grafica interattiva.



Figura 2.6: Esempio che mostra alcune delle caratteristiche dell'applicazione sviluppata per il museo Rubens House.

⁸ Un video più approfondito che mostra una demo su tale caso lo si può trovare online al seguente link: <https://www.youtube.com/watch?v=S04viOYnSg4>.

⁹ <http://www.prophets.be/>

Capitolo 3

L’infrastruttura base

Per lo sviluppo della tesi si è partiti da un primo prototipo funzionante di una infrastruttura che consentiva a vari smart-devices Android posizionati su diversi tag NFC di auto-organizzarsi e collaborare al fine di svolgere un certo compito applicativo. Il concetto teorico alla base di tale infrastruttura deriva dal campo dello Spatial Computing, in cui si sfrutta il concetto di “vicinato”: ogni nodo del sistema non è direttamente a conoscenza di tutti gli altri nodi presenti, ma attraverso lo scambio di messaggi coi propri vicini riesce a costruirsi una “mappa” che tiene traccia di tutti i nodi e relative distanze, in modo da organizzarsi di conseguenza per la propria computazione. Il motivo che ha portato alla nascita dell’infrastruttura sta nel bisogno di fornire agli smart-devices (che rappresentano i nodi del sistema) una specie di middleware sul quale “appoggiarsi” per poter scambiare le informazioni utili alla computazione, ovvero inviando i propri dati provenienti dalla parte “sensoristica” (quindi l’ID del tag NFC che individua una certa posizione nello spazio) e ricevendo la lista degli altri nodi presenti nel proprio vicinato in modo da fare un broadcasting di tali informazioni e favorire la collaborazione tra devices. In questa maniera, analogamente a quanto accade nei modelli di Spatial Computing, ogni dispositivo è situato in modo preciso nello spazio ed è sempre a conoscenza di tutti gli eventuali vicini e relativa distanza, potendo quindi sfruttare queste informazioni per auto-organizzarsi al fine di svolgere un determinato task definito a livello applicativo.

3.1 Spatial Computing

Lo Spatial Computing è un campo emergente della ricerca in cui si fa un uso esplicito del concetto di spazio nella computazione. I calcolatori che effettuano computazioni di questo tipo vengono detti “spatial computers”, ovvero una serie di dispositivi computazionali distribuiti in uno spazio fisico, dove la distanza fra i dispositivi ha una forte incidenza sulle comunicazioni e gli obiettivi funzionali del sistema sono definiti sulla base della sua struttura spaziale. Le applicazioni

in questo campo coprono molti ambiti diversi tra cui monitoraggio ambientale, pervasive-computing, reti di sensori, reti ad-hoc, smart-environment, ecc. Il tema comune tra tutti questi domini rimane comunque l'enfasi sulla rappresentazione e gestione esplicita dello spazio.

Per facilitare l'organizzazione della computazione distribuita in tali sistemi, possiamo modellare il sistema stesso come uno spazio continuo piuttosto che come una rete, agendo quindi su regioni dello spazio geometrico invece che sui singoli dispositivi con conseguenti vantaggi in termini di scalabilità, robustezza e adattabilità.¹⁰

Se i dispositivi comunicano direttamente tra loro su distanze brevi (ad esempio ai vicini confinanti via wireless) allora la struttura globale della rete di comunicazione forma una approssimazione discreta della struttura dello spazio di interesse, che possiamo definire come un “amorphous medium”. Un amorphous medium può essere considerato come una “Riemannian manifold”¹¹, in cui si ha un device in ogni punto e dove ognuno di questi conosce l'ultimo stato di tutti gli altri devices del suo vicinato locale (o neighborhood), ovvero quelli a cui è collegato. Una rete di dispositivi che comunicano localmente può essere quindi vista come una approssimazione discreta di un amorphous medium, dove ogni device rappresenta una piccola regione di spazio circostante e dove i messaggi inviati tra devices adiacenti costituiscono il flusso informativo scambiato nel vicinato.

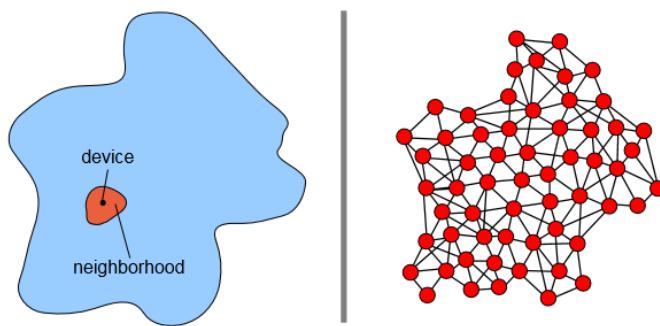


Figura 3.1: A sinistra un esempio di amorphous medium, a destra una rete di dispositivi che lo approssimano.

¹⁰ J. Beal, R. Schantz, “A Spatial Computing Approach to Distributed Algorithms”, 45th Asilomar Conference on Signals, Systems, and Computers, November 2010.

¹¹ Una “Riemannian manifold” (o varietà riemanniana) è una generalizzazione del concetto di curva e di superficie differenziabile in dimensioni arbitrarie, su cui sono definite le nozioni di distanza, lunghezza, geodetica, area (o volume) e curvatura. È utile a modellizzare spazi “curvi” di dimensione arbitraria.

Attraverso determinate primitive computazionali accuratamente scelte e combinate fra loro, come quelle fornite dal linguaggio domain-specific Proto¹², è possibile scrivere programmi per questi spatial computers utilizzando proprio tale astrazione dell'amorphous medium; così facendo il programmatore non agisce più a livello di singoli dispositivi (livello locale), ma può direttamente descrivere il comportamento globale delle varie regioni dello spazio. Tali programmi saranno poi trasformati automaticamente in azioni locali che verranno eseguite dai devices della rete reale, al fine di raggiungere un'approssimazione del comportamento aggregato desiderato.

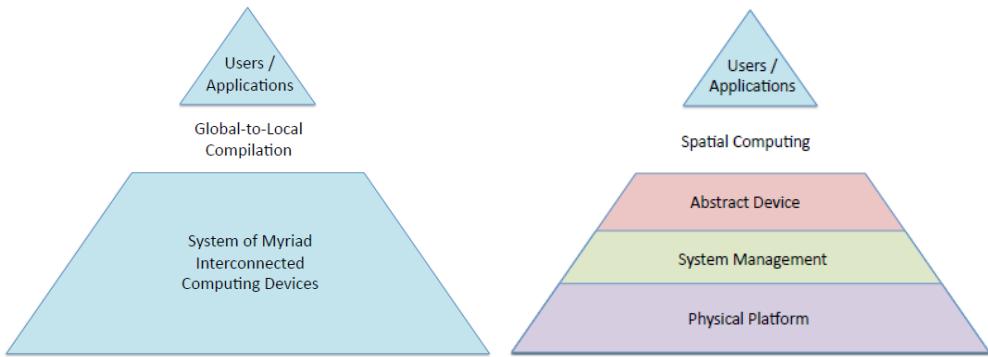


Figura 3.2: Gap tra computazione a livello locale e quella globale colmato dallo Spatial Computing.¹³

In questo modo è possibile colmare quel gap che esiste tra la computazione a livello di singoli dispositivi e la possibilità di controllare il loro comportamento aggregato, e che tuttora costituisce una sfida nel mondo della programmazione.

3.2 Primo prototipo

Come già detto in precedenza, il primo prototipo dell'infrastruttura teneva conto solo della tecnologia NFC: l'idea era quella di sfruttare un "tappeto" in cui erano presenti vari tag passivi identificativi di una certa posizione all'interno del tappeto stesso, in modo che appoggiandoci sopra gli smart-devices e muovendoli da una posizione all'altra si potessero scatenare eventi in maniera molto fluida e naturale. Gli eventi così generati potevano essere comunicati all'infrastruttura, la quale in risposta forniva le informazioni sui devices vicini

¹² M. Viroli, J. Beal, K. Usbeck, "Operational semantics of Proto", *Science of Computer Programming* (2012).

¹³ J. Beal, S. Dulman, K. Usbeck, M. Viroli, N. Correll, "Organizing the Aggregate: Languages for Spatial Computing", April 2nd, 2012.

in modo tale che ognuno di questi potesse di conseguenza auto-organizzarsi e cooperare al meglio per lo scopo applicativo. Il modello computazionale consisteva praticamente in un ciclo d’ascolto infinito che ad ogni intervallo di tempo prestabilito permetteva ai devices del sistema di percepire gli eventi scatenati dall’utente, elaborarli per auto-organizzarsi al meglio, e fare un broadcasting delle nuove informazioni.

L’occasione per mostrare le potenzialità di questo “Magic Carpet” in combinazione con l’uso di molteplici dispositivi Android in nostro possesso (quali tablet Nexus 7 e smartphones Nexus 5) è stata l’evento dello scorso 14-15 marzo al Museo d’Arte Moderna di Bologna (MAMbo), e a tale proposito furono sviluppate diverse applicazioni che potessero avere uno scopo ludico e un certo fattore “wow” per attrarre l’attenzione del pubblico. La lista delle varie applicazioni realizzate per tale occasione comprende le seguenti:

- Channel: si sfruttano i dispositivi che si trovano tra una sorgente e una destinazione per visualizzare il canale più breve che li unisce.
- Partition: date diverse sorgenti differenziate per colore, i dispositivi che non lo sono assumono il colore della sorgente più vicina.
- TicTacToe: è il classico gioco del tris, si sfruttano due griglie già prestabilite sul tappeto.
- OpenTicTacToe: gioco del tris senza una griglia prefissata, ma “a tutto campo”.
- EightPuzzle: forma breve del “gioco del quindici”, si ha una griglia (non prestabilita) di nove posizioni (3x3) in cui i dispositivi possono scorrere in orizzontale o verticale in corrispondenza dello spazio vuoto, con l’obiettivo di ordinarli dall’1 all’8 (partendo dalla casella in alto a sinistra e lasciando lo spazio vuoto in basso a destra); si sono creati tre livelli di difficoltà diversi.
- Phuzzle: data una foto (che può essere anche scattata sul momento), essa viene visualizzata sui dispositivi posizionati nel tappeto e, in base alla loro disposizione, riadattata per visualizzarla nel modo più esteso possibile (suddividendola in parti diverse, una per ogni device).

Per realizzare l’astrazione tipica dello Spatial Computing, in cui ogni nodo del sistema ottiene dinamicamente le informazioni sui vicini presenti nel suo

intorno, si è sfruttata una parte Server per gestire la comunicazione di tali informazioni agli smart-devices. Inoltre, è stata racchiusa in essa anche la possibilità di comunicare ai devices quale applicazione mandare in esecuzione tra le varie descritte prima (permettendo il passaggio da una all'altra in qualsiasi momento). Riguardo alle modalità di comunicazione in rete sono stati connessi Server e smart-devices sotto una stessa rete Wi-Fi e si sono utilizzate socket TCP, in modo da evitare problemi di recupero degli indirizzi pubblici di ogni nodo e al tempo stesso avere una certa stabilità, affidabilità e reattività nello scambio dei messaggi.

3.2.1 Architettura Logica

Analizzando questo primo prototipo dell'infrastruttura emerge una suddivisione in due blocchi principali: una che rappresenta la parte Server e una che rappresenta quella Client.

La parte Server è costituita da un unico sottosistema denominato “GeoServer” che si occupa sostanzialmente di specificare quale applicazione mandare in esecuzione fra quelle sviluppate comunicandolo ai nodi client (e quindi agli smart-devices) che ne fanno richiesta, e fornendo loro le informazioni relative al proprio vicinato in caso di spostamenti. La parte Client è costituita invece da diversi sottosistemi, differenziati in base alle loro funzionalità specifiche:

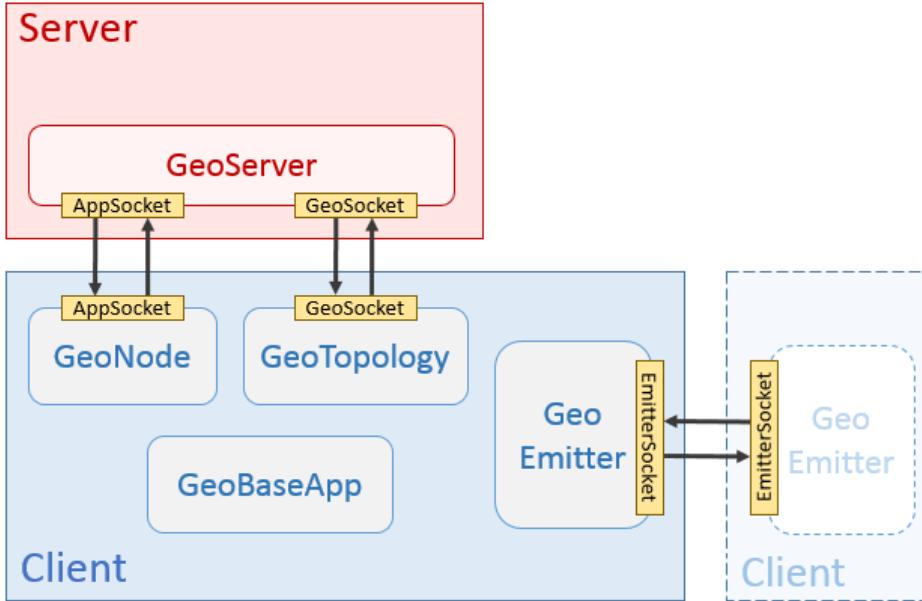
- un sottosistema denominato “GeoNode” che si occupa di eseguire sullo smart-device l'applicazione scelta dal GeoServer, specificare tutta la parte computazionale relativa alle varie applicazioni sviluppate, e interagire con la tecnologia NFC per il riconoscimento dei vari tag e degli eventi generati dall'utente;
- un sottosistema denominato “GeoTopology” che si occupa di scambiare con il GeoServer i messaggi contenenti le informazioni aggiornate sugli spostamenti dei nodi e relativi vicinati;
- un sottosistema “GeoEmitter” che si occupa di ricevere e aggregare tutte le informazioni di stato aggiornate relative ad ogni nodo, e inviarle agli altri nodi dello stesso vicinato;
- un sottosistema “GeoBaseApp” per implementare la base di una applicazione che utilizza il sistema, eseguendo il ciclo computazionale

che consente ad ogni intervallo di tempo di notificare gli aggiornamenti di stato relativi ai nodi, generare la tabella del vicinato con le informazioni aggiornate e segnalare al GeoEmitter di comunicarla ai nodi interessati.

Tutte le comunicazioni di rete avvengono tramite lo scambio di diversi tipi di messaggi su socket TCP, con l'utilizzo di un convertitore JSON per serializzare e deserializzare gli oggetti in stringhe. È inoltre presente una parte protocollare che specifica i vari tipi di messaggi scambiati, separando ciò che è attinente alle applicazioni da ciò che riguarda le informazioni su spostamenti dei nodi e tabelle dei vicinati. Per questo si ha:

- una parte denominata “AppProtocol” contenente la definizione dei messaggi relativi alla sola parte applicativa, ovvero: APP_JOIN (richiesta di quale applicazione lanciare), APP_WELCOME (in risposta alla APP_JOIN), APP_RESET (per segnalare un cambio di applicazione) e APP_SHUTDOWN (per segnalare la terminazione).
- una parte denominata “GeoProtocol” contenente la definizione dei messaggi relativi ai nodi, ovvero: GEO_HELLO (ingresso di un nodo nel sistema con una certa posizione sul tappeto), GEO_MOVING (uscita di un nodo dal sistema), GEO_NEWPOS (un nodo ha cambiato di posizione sul tappeto), GEO_NHOOD (vicinato corrispondente a un nodo posizionato), GEO_NEIGH (informazioni su un nodo che sta entrando in nuovo vicinato), GEO_INVAL (informazioni su un nodo che sta uscendo da un vicinato).
- una parte denominata “EmitterProtocol” per la specifica dei messaggi scambiati tra i devices Client, ovvero le tabelle dei vicinati aggiornate (EMITTER_NBRTABLE).

L’architettura logica di base relativa a questo primo prototipo dell’infrastruttura può quindi essere rappresentata dalla seguente figura:



3.2.2 Parte progettuale

Una volta definita l'architettura logica complessiva del sistema passiamo a descrivere a grandi linee come è progettato ogni singolo sottosistema. Si sceglie di non mostrare in maniera approfondita come viene implementata la parte comunicativa e protocollare.

3.2.2.1 Server

Per quanto riguarda la parte Server, il sottosistema GeoServer è implementato mediante un thread Java che alla creazione si occupa di:

- costruire le socket per la comunicazione coi nodi Client;
- lanciare un thread che si occupi di ricevere e immagazzinare le informazioni sui nodi Client che entrano nel sistema (messaggi di tipo GEO_HELLO) o che si spostano da una posizione all'altra del tappeto (messaggi di tipo GEO_MOVING);
- lanciare un thread che si occupi di ricevere messaggi di tipo APP_JOIN da parte dei nodi Client, salvare le informazioni di questi (per poterli poi avvisare di eventuali cambi di applicazione o terminazione del sistema), e rispondere a loro tramite messaggi di tipo APP_WELCOME;
- lanciare un thread che si occupi di gestire la terminazione del sistema, segnalandola ai nodi Client (tramite messaggi di tipo APP_SHUTDOWN).

Quando il GeoServer verrà eseguito, questi creerà a sua volta un nuovo thread che si occuperà di eseguire un ciclo infinito in cui:

- si recupereranno i messaggi precedentemente immagazzinati relativi ai nodi Client (GEO_HELLO e GEO_MOVING);
- se il messaggio è di tipo GEO_HELLO si memorizzeranno le informazioni (coordinate e indirizzo IP) del nodo;
- se il messaggio è di tipo GEO_MOVING si notificherà ai vecchi vicini del nodo la sua uscita dal vicinato (messaggi di tipo GEO_INVAL), poi si notificherà ai nuovi vicini l'entrata del nodo nel loro vicinato (messaggi di tipo GEO_NEIGH) e infine si notificherà al nodo stesso il nuovo vicinato (messaggio di GEO_NHOOD).

Per creare ed eseguire il thread del GeoServer sono possibili due modalità: una versione che sfrutta la piattaforma Android, e una che sfrutta Java. Per la modalità Android è stato creato un progetto a parte denominato “GeoAndroidServer”, in cui si gestisce il tutto tramite una semplice “MainActivity” che si occupa anche di creare tutte le componenti grafiche per la selezione dell’applicazione da lanciare. Per la modalità Java si ha invece una semplice classe denominata “ServerLauncher” che funge da main per la creazione e l’avvio del thread GeoServer, e una classe JFrame denominata “ServerMonitor” che fornisce un’interfaccia grafica per la scelta delle diverse applicazioni da eseguire.

Un’annotazione riguarda l’applicazione “Phuzzle”: essa può essere selezionata solo su piattaforma Android, visto che la possibilità di scattare foto o sceglierne una già presente dalla galleria dello smart-device è stata implementata solo all’interno della MainActivity.

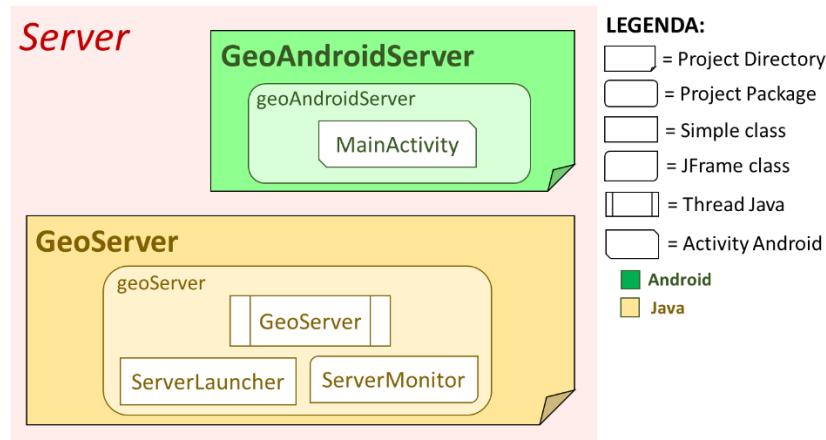


Figura 3.3: Progettazione della parte Server.

3.2.2.2 Client

Per quanto riguarda la parte Client ogni sottosistema è stato implementato con un progetto Java, ad eccezione del sottosistema GeoNode che dovendo essere installato sugli smart-devices è stato realizzato tramite un progetto Android.

3.2.2.2.1 *Il sottosistema GeoTopology*

Per il sottosistema GeoTopology sono state create due entità distinte, in base alle funzionalità che svolge questo sottosistema:

- un’entità denominata “GeoClient” che si occupa di gestire tutta la comunicazione con il GeoServer, ovvero l’invio e la ricezione dei messaggi definiti nel GeoProtocol;
- un’entità denominata “NeighborhoodManager” per la gestione dei vicini nuovi/persi (comunicati dal GeoClient e notificati a loro volta al sottosistema GeoEmitter) e degli spostamenti dei nodi sul tappeto (notificati al GeoClient);

Queste due entità sono implementate tramite due differenti classi Java, e quando viene creata un’istanza del NeighborhoodManager esso si occupa di:

- creare un’istanza del GeoClient passandogli le informazioni riguardanti un nodo Client, e l’indirizzo IP e porta del GeoServer;
- registrarsi come ascoltatore degli eventi generati dal GeoClient;
- comunicare al GeoClient di inviare un messaggio di tipo GEO_HELLO al GeoServer, per segnalare l’entrata nel sistema del nodo in questione.

Il NeighborhoodManager ha anche la funzionalità di avvisare il GeoClient dello spostamento del nodo da un certo tag NFC a un altro (in modo che questi possa poi inviare un messaggio di tipo GEO_MOVING al GeoServer), e in conseguenza di ciò notificare al GeoEmitter la tabella del vicinato aggiornata.

Per quanto riguarda il dettaglio delle funzionalità del GeoClient, esso si occuperà in primo luogo di creare le socket per la comunicazione col GeoServer dato, e in secondo luogo si metterà sempre in ascolto dei messaggi di risposta inviategli da quest'ultimo (GEO_NHOOD, GEO_NEWNEIGH, e GEO_INVAL). In caso di ricezione di messaggi che indicano l'entrata o l'uscita del nodo da un certo vicinato, il GeoClient andrà a notificare il NeighborhoodManager di tali eventi, il quale a sua volta lo notificherà al sottosistema GeoEmitter.

In pratica gli eventi generati possono essere suddivisi in due tipi diversi:

- “esterni” quando un altro nodo Client entra o esce dal vicinato del nodo in questione;
- “interni” se generati dai movimenti del nodo in questione (mi posiziono sopra a un tag NFC o mi tolgo da uno di questi).

Per questo motivo sono state implementate due interfacce Java differenti per gli ascoltatori di questi due diversi tipi di eventi, denominate appunto “IGeoExternalEventListener” e “IGeoInternalEventListener”.

3.2.2.2 Il sottosistema GeoEmitter

Per il sottosistema GeoEmitter è stata realizzata una classe Java “EmitterManager” che si occupa di:

- creare una socket per la comunicazione con gli altri nodi Client (quindi con gli altri EmitterManager);
- lanciare un thread che si occupa di ricevere e immagazzinare i messaggi di tipo EMITTER_NBRTABLE provenienti dagli altri nodi Client;
- lanciare un thread che esegue un ciclo infinito in cui prima si vanno a recuperare i messaggi immagazzinati in precedenza, e poi si va ad aggiornare la tabella del vicinato con le informazioni aggiornate riguardanti i nodi vicini;

- fornire un metodo per poter trasmettere a tutti i nodi del vicinato la nuova tabella con le informazioni aggiornate (messaggi di tipo Emitter_NBRTABLE).

L'EmitterManager si occupa anche di mettere/togliere dalla mappa dei vicinati le informazioni relative a un certo nodo che ha cambiato vicinato o si è tolto da un tag NFC (eventi segnalati dal NeighborhoodManager), in modo da mantenere una tabella del vicinato sempre aggiornata e consistente.

All'interno di questo sottosistema vengono create anche tutte le classi e le interfacce Java necessarie all'implementazione dell'EmitterProtocol per la comunicazione fra EmitterManager di Client diversi.

3.2.2.3 Il sottosistema GeoBaseApp

Per il sottosistema GeoBaseApp è stata creata una classe Java “BaseApp” che si occupa di svolgere ad ogni intervallo di tempo (fissato a piacere, ad esempio ogni 250 millisecondi) il ciclo computazionale infinito in cui:

- si recuperano dall'EmitterManager le informazioni del nodo Client in considerazione e relativo vicinato;
- si trasmettono queste informazioni sui vicini (con un metodo astratto che tutte le diverse applicazioni specifiche andranno a ridefinire per i loro scopi applicativi) per consentire l'auto-organizzazione del nodo e ottenere successivamente l'aggiornamento sul suo nuovo stato;
- si notificano questi nuovi aggiornamenti di stato del nodo appena computati agli ascoltatori interessati (ovvero al sottosistema GeoNode);
- si segnala all'EmitterManager di trasmettere a tutti i vicini del nodo la nuova tabella del vicinato con le informazioni di stato aggiornate.

È stata implementata anche un'interfaccia Java denominata “IAppUpdateListener” per gli ascoltatori interessati alle notifiche sugli aggiornamenti di stato del nodo.

3.2.2.4 Il sottosistema GeoNode

Per il sottosistema GeoNode è stato realizzato un progetto Android installabile su ogni smart-device che funge da nodo Client. Le diverse Activity Android presenti nel progetto sono:

- una “LauncherActivity” che si occupa di stabilire una connessione alla stessa rete Wi-Fi del Server (se è la prima volta che viene lanciata) e di comunicare con il GeoServer (creando le relative socket) per richiedere quale applicazione specifica lanciare, mandandogli quindi un messaggio di tipo APP_JOIN e aspettando la sua risposta (messaggio di tipo APP_WELCOME). Una volta ottenuta la risposta su quale applicazione eseguire, si lancia l’Activity specifica relativa a quest’ultima e si termina la LauncherActivity stessa;
- una “BaseActivity” in cui si racchiude il comportamento comune a tutte le varie Activity delle specifiche applicazioni, ovvero si occupa di: creare le socket per la ricezione dei messaggi di tipo APP_RESET e APP_SHUTDOWN inviati dal GeoServer (con conseguente lancio della LauncherActivity e terminazione della BaseActivity stessa); notificare gli eventi scatenati dall’interazione tra utente e smart-devices (ovvero il posizionamento sopra un certo tag NFC, la rimozione del device da sopra un certo tag, e il doppio-tocco sullo schermo); inizializzare le applicazioni con le informazioni base sulle caratteristiche del tappeto (dimensioni e distanze), fornendo loro un metodo astratto che ogni specifica Activity di queste applicazioni andrà poi a ridefinire per i propri scopi applicativi;
- diverse specifiche Activity relative alle varie applicazioni sviluppate, quindi: ChannelActivity, EightPuzzleActivity, OpenTictacToeActivity, PartitionActivity, PhuzzleActivity e TicTacToeActivity. Ognuna di queste avrà il proprio specifico layout e andrà a ridefinire il metodo astratto descritto prima occupandosi di visualizzare sullo schermo del device le informazioni applicative aggiornate a fronte del nuovo stato del nodo segnalatogli dal sottosistema GeoBaseApp.

Per quanto riguarda la parte computazionale delle specifiche applicazioni, si sono create diverse classi Java che estendono la classe BaseApp descritta in precedenza, occupandosi quindi di ridefinire il metodo astratto caratteristico di tale classe per consentire l’auto-organizzazione del nodo in caso di nuove informazioni sui vicini e fornire dunque l’aggiornamento del suo nuovo stato. Allo stesso modo delle Activity, tali classi vendono denominate come:

ChannelApp, EightPuzzleApp, OpenTictacToeApp, PartitionApp, PhuzzleApp e TicTacToeApp.

Inoltre, dentro al sottosistema è presente anche un'altra classe Java che funge da “collegamento” tra le specifiche Activity Android appena descritte e i sottosistemi implementati in Java descritti nei paragrafi precedenti, inclusa anche la parte computazionale data dalla BaseApp. In questo modo si è cercato di disaccoppiare “view” e “control”, andando cioè a separare la parte relativa al controllo da quella relativa alla rappresentazione. Questa classe è stata denominata “GenericAppExecutor”, e si occupa sostanzialmente di:

- creare l'EmitterManager per una specifica applicazione;
- creare il NeighborhoodManager collegandolo all'EmitterManager appena creato e all'indirizzo IP e porta del GeoServer (che poi comunicherà al GeoClient);
- registrare l'EmitterManager come ascoltatore degli eventi generati dal NeighborhoodManager (esterni e interni);
- registrarsi come ascoltatore degli eventi generati dalla specifica applicazione e avvisare di conseguenza la relativa Activity interessata.

Anche in questo sottosistema si possono dunque rilevare due diversi tipi di eventi, dovuti alle modalità con cui l'utente interagisce con lo smart-device:

- l'utente effettua un doppio-tocco sullo schermo dei device;
- l'utente alza il device da una certa posizione del tappeto e volendo lo può spostare verso un'altra posizione.

Per questo motivo sono state implementate due interfacce Java differenti per gli ascoltatori di questi due diversi tipi di eventi, denominate “IGestureSensorListener” e “IPositionSensorListener”.

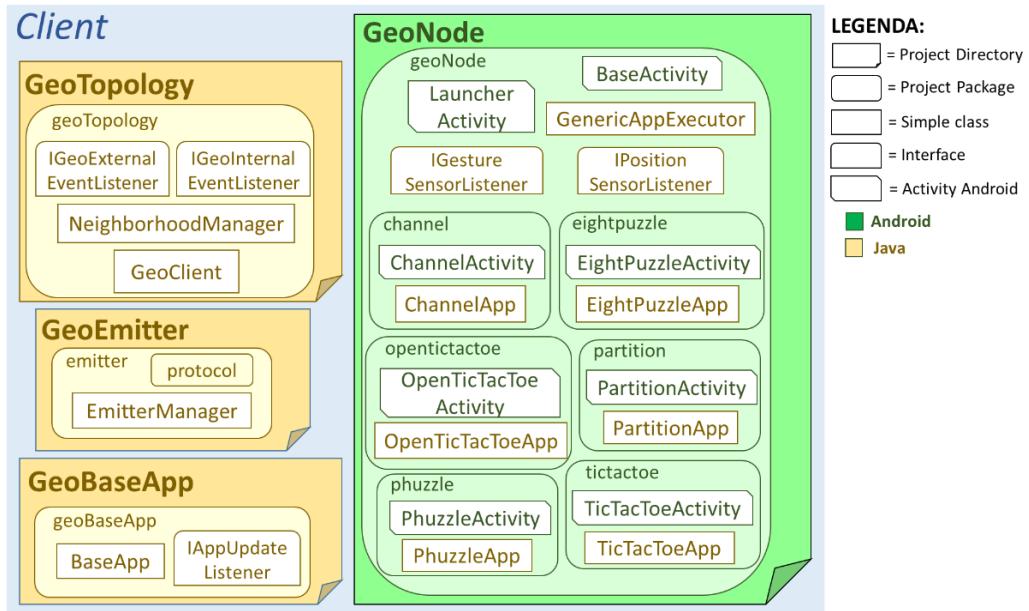


Figura 3.4: Progettazione della parte Client.

3.2.2.3 Architettura di progetto

L’architettura di progetto esemplificativa del primo prototipo dell’infrastruttura, che mostra le macro-interazioni tra i vari sottosistemi descritti in precedenza, viene rappresentata in figura 3.5.

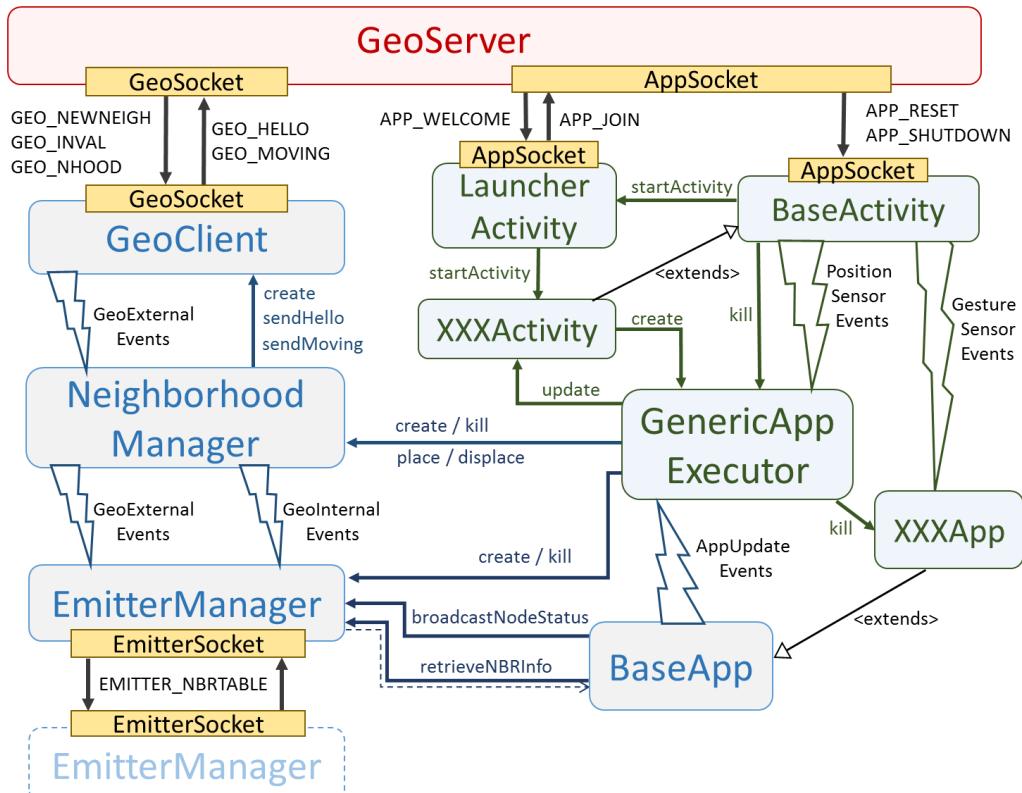


Figura 3.5: Architettura di progetto relativa al primo prototipo dell’infrastruttura.

3.2.3 Il simulatore Java

Per poter avere uno strumento che permetta di sperimentare e verificare il comportamento delle varie applicazioni sviluppate, è stato creato anche uno specifico simulatore Java (senza dunque dover installarle ogni volta su tutti gli smart-devices in possesso).

Questo simulatore si basa su diversi componenti grafici JFrame:

- una classe “NFCcarpet” che consente di simulare il tappeto NFC “disegnando” tutte le posizioni in una griglia (5x6 come nel tappeto reale) che si adatta alle dimensioni dello schermo;
- una classe “BaseNode” che permette di simulare un generico nodo del sistema rappresentandolo come una finestra grafica delle stesse dimensioni di una posizione del tappeto simulato, e che racchiude dentro di sé tutte le funzionalità comuni ai nodi delle varie applicazioni specifiche, ovvero la possibilità di connettersi all’indirizzo IP e porta del Server (tramite una JDialog a parte), e di simulare (grazie a componenti grafici come JButton e JComboBox): un piazzamento sopra una certa posizione del tappeto, una rimozione da una certa posizione su cui si era piazzati, e il doppio-tocco sul nodo;
- varie classi rappresentanti i nodi simulati delle specifiche applicazioni, dentro le quali si va a implementare la rispettiva logica applicativa a fronte degli aggiornamenti dati dalla BaseApp;
- una classe “NodeLauncher” che si occupa di lanciare una serie di nodi simulati relativi a una specifica applicazione.

Nella figura 3.6 si mostra un esempio di simulazione della applicazione Partition, con 10 nodi simulati piazzati in maniera sparsa nel tappeto simulato (di cui 3 sorgenti di colore diverso fra loro).

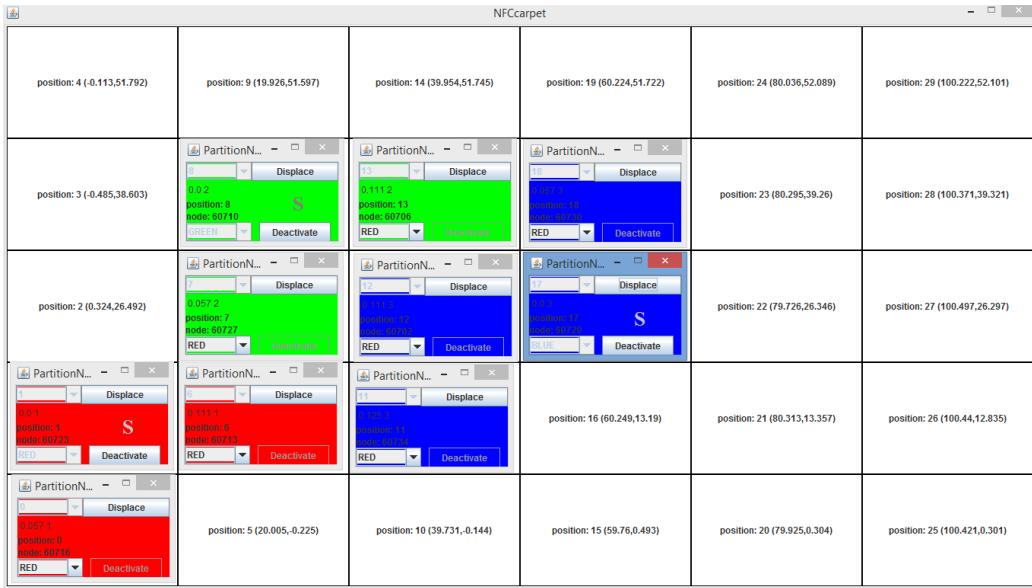


Figura 3.6: Un esempio di simulazione della applicazione Partition.

3.2.4 L’esperienza al MAMbo

Durante l’evento dello scorso 14-15 marzo 2014 il dipartimento del DISI si è presentato al Museo d’Arte Moderna di Bologna (MAMbo) per mostrare una concreta rappresentazione dei prodotti creativi nati da un processo tecnologico innovativo. In tale occasione si è esposto al pubblico questo primo prototipo e si è ottenuto un feedback sul lavoro svolto. Le impressioni che si sono avute sono state molto positive, soprattutto l’interesse dei più piccoli per le applicazioni di stampo ludico (come il tris, l’8-puzzle e la partition). Questo fa capire come la strada intrapresa possa essere di successo anche dal punto di vista commerciale, con l’unica limitazione data dall’esigenza di impiegare molteplici smart-devices Android dal costo non indifferente (a fronte invece del basso costo dei tag NFC), anche se si potrebbe pensare di fornire una base per “giochi di società” in scenari dove molti utenti (ognuno col proprio smart-device) si incontrano per “giocare” sul tappeto, e quindi commercializzare solo tappetino e relative applicazioni Android.

Pensando invece a futuri sviluppi su larga scala, si può astrarre dalla singola tecnologia NFC per permettere alle applicazioni di avere un utilizzo anche in spazi più ampi come l’interno di edifici o spazi aperti. Ad esempio in scenari crowd (come all’interno di stadi, o concerti), l’applicazione Channel può guidare l’utente dal punto in cui si trova a quello indicato come destinazione

auto-organizzandosi dinamicamente per indicare il percorso più breve (in base alla presenza di altri smart-device tra l'utente e la destinazione), oppure l'applicazione Partition può essere utilizzata nei concerti per particolari effetti luminosi che possono variare dinamicamente e differenziarsi in base alla locazione degli utenti.

È per questo motivo che si è deciso di passare allo sviluppo di una nuova infrastruttura, attraverso la reingegnerizzazione di questo primo prototipo e astraendo dalla singola tecnologia location-based utilizzata, modificando la struttura in modo da renderla modulare e permettendo così la possibilità di aggiungere nuove tecnologie all'occorrenza senza stravolgere l'architettura sviluppata.

Capitolo 4

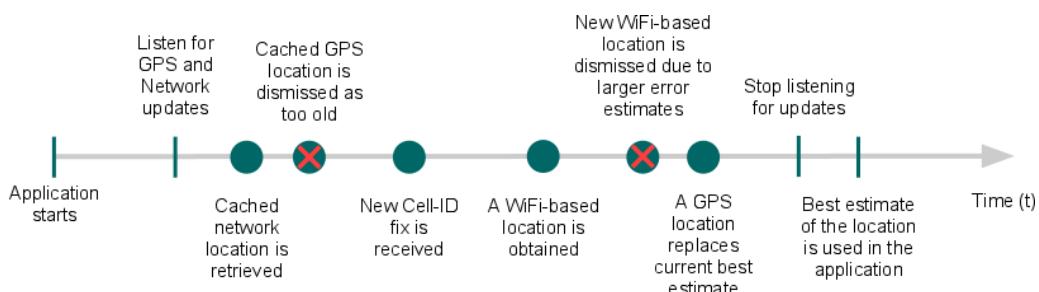
Sperimentazione su BLE e GPS

Per verificare l'effettiva possibilità di integrazione delle nuove tecnologie con l'infrastruttura base, e al tempo stesso avere una stima precisa sulle diverse caratteristiche che queste possiedono, si è deciso di creare due diversi applicativi Android sfruttando gli smart-devices in dotazione: tablet Google Nexus 7, smartphones Google Nexus 5 e il personale Samsung Galaxy S3.

Come si vedrà nel seguito, sia relativamente alla parte GPS che a quella BLE non si sono riscontrate differenze comportamentali degne di nota tra i diversi smart-devices in dotazione.

4.1 Applicazione GPS

Per quanto riguarda il GPS, si è sviluppata una applicazione che si basa su una semplice Activity Android che permette di ricevere le notifiche relative agli aggiornamenti periodici sulla posizione geografica del dispositivo (implementando l'interfaccia “LocationListener”). A questo scopo si utilizza un “LocationManager” che permette l'accesso ai servizi di localizzazione presenti sullo smart-device, e si decide di combinare il GPS con il Network Location Provider di Android: in questo modo si determina la posizione dell'utente utilizzando anche i ripetitori per cellulari e segnali Wi-Fi, e si riescono a ottenere le informazioni sulla posizione del dispositivo più velocemente rispetto al solo GPS, anche in ambienti chiusi e consumando meno la batteria. Nella seguente figura si mostra come questo modello combina le informazioni ricavate dai due differenti providers per ottenere la più precisa posizione possibile:



Per ricevere gli aggiornamenti di posizione di entrambi tali providers, occorre definire tra i vari permessi descritti nel file Manifest di Android quello relativo a “ACCESS_FINE_LOCATION”, come mostrato nella seguente figura:

```
<manifest ... >
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    ...
</manifest>
```

Relativamente all’Activity invece, si decide di definire un’oggetto di classe “Criteria” che stabilisce i criteri di applicazione per la selezione del miglior provider di localizzazione in base a accuratezza, consumo di energia, velocità di acquisizione della posizione, e anche la capacità di segnalare l’altitudine. In questo modo verrà scelto in automatico il provider migliore e si otterrà da esso l’ultima posizione nota dell’utente. Ciò viene mostrato nella seguente figura:

```
// Get the location manager
locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
// Define the criteria how to select the location provider -> use default
Criteria criteria = new Criteria();
provider = locationManager.getBestProvider(criteria, false);
Location location = locationManager.getLastKnownLocation(provider);
```

Per questa applicazione che possiamo definire “di prova” si decidono di visualizzare solo latitudine e longitudine come coordinate terrestri per la localizzazione e si sfruttano anche le mappe Google, come si può vedere in figura 4.1.

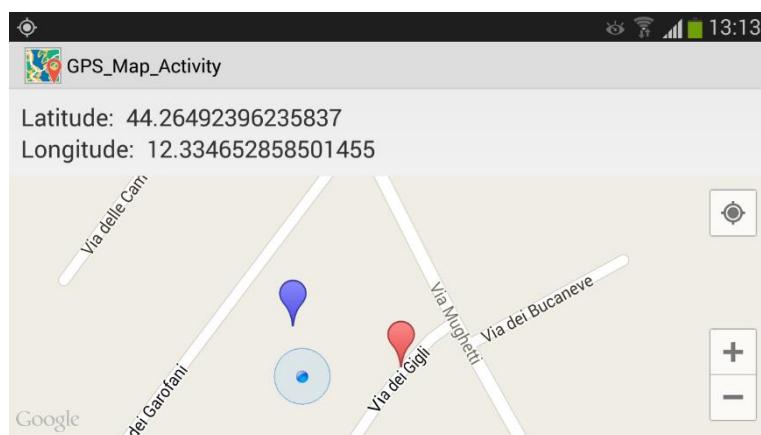


Figura 4.1: Un esempio che mostra l’applicazione GPS sviluppata per la sperimentazione.

Un’annotazione riguarda la precisione delle coordinate terrestri: latitudine e longitudine sono grandezze angolari e hanno il massimo errore all’equatore essendo il raggio della Terra massimo. L’accuratezza della longitudine aumenta

allontanandosi dall'equatore (il raggio della Terra varia da 6378.160 a 0 km), mentre l'accuratezza della latitudine cambia dello 0.335% (il raggio della Terra varia da 6378.160 a 6356.778 km). Essendo quindi la circonferenza della Terra pari a $2\pi r$, essa risulta pari a 40075.1612 km, e quindi il valore di un singolo "grado" si ottiene dividendo questo valore per 360 (pari a 111.319 km).

I gradi possono essere rappresentati in vari formati, nella applicazione sviluppata si sfrutta quello decimale. Di seguito viene mostrata una tabella che rappresenta la precisione delle coordinate in base alle cifre decimali¹⁴:

Posti	Gradi decimali	Gradi	Distanza
0	1.0	1°0'0"	111.319 km
1	0.1	0°6'0"	11.132 km
2	0.01	0°0'36"	1.113 km
3	0.001	0°0'3.6"	111.3 m
4	0.0001	0°0'0.36"	11.13 m
5	0.00001	0°0'0.036"	1.11 m
6	0.000001	0°0'0.0036"	11.1 cm
7	0.0000001	0°0'0.00036"	1.11 cm

Dopo aver effettuato varie prove su tutti i dispositivi a disposizione, si è riscontrata un'ottima precisione e velocità nell'acquisizione e aggiornamento della posizione del device in un'ambiente outdoor: al lancio dell'applicazione trascorrono circa 4-5 secondi prima di ottenere le coordinate terrestri relative a longitudine a latitudine, con una precisione nell'ordine del centimetro. Relativamente ad ambienti indoor invece, si notano peggioramenti sia dal punto di vista delle tempistiche (occorrono dai 10 ai 15 secondi per ricavare la prima posizione), sia in termini di precisione in quanto si ha una grande instabilità: ad esempio, anche se il device rimane fermo, si rilevano cambi di posizione nell'ordine di una decina di metri, oppure se si è in movimento occorrono molti secondi prima di ottenere una stima (comunque imprecisa) sulla posizione aggiornata.

Con questa applicazione si è avuto modo di verificare concretamente tutti i limiti del GPS, anche se la possibilità di sfruttare il Network Provider ha comunque contribuito al miglioramento nell'utilizzo di tale tecnologia permettendo una

¹⁴ Dati ottenuti dal sito: www.sunearthtools.com.

localizzazione geografica all'interno degli edifici, a discapito però di un considerevole margine d'instabilità nell'accuratezza. Questo è uno dei motivi principali per cui questa tecnologia non viene impiegata per applicazioni che richiedono una buona precisione in ambienti indoor, mentre risulta essere lo "standard" utilizzato per la navigazione stradale e l'orientamento outdoor.

4.2 Applicazione BLE

Per quanto riguarda il Bluetooth Low Energy, vista l'impossibilità di simulare il comportamento dei beacons con i dispositivi Android in dotazione si è deciso di contattare tutte le maggiori aziende produttrici di beacons in modo da ottenere maggiori informazioni su: prezzi, tempi di spedizioni effettivi, SDK compatibile con Android, caratteristiche del beacon (potenza di trasmissione e intervallo di advertising impostati di default) e ricambio batteria. Per tutte queste considerazioni, si è deciso di puntare sui beacons di Stick N Find e Accent Advanced System.

Dal momento che i tempi di spedizione risultavano comunque elevati, si è inizialmente deciso di intraprendere la strada del beacon fai-da-te sfruttando i dongle BLE USB (acquistabili agevolmente online su piattaforme e-commerce come Amazon e Ebay, e quindi ottenibili in pochi giorni). Essendo già in possesso di un computer fisso Apple e un notebook con sistema operativo Windows, si è optato per la soluzione che prevede l'uso di OS X Mavericks come beacon "virtuale". Visto che il Mac in dotazione risulta datato 2008 e quindi oltre a non avere il chip BLE ha anche un sistema operativo precedente alla versione richiesta, si è dovuto prima di tutto eseguire un upgrade a Mavericks. Per quanto riguarda il notebook con Windows (anch'esso non dotato di chip BLE) si è utilizzata un'immagine virtuale di Mavericks ottenibile facilmente dalla rete ed eseguibile su VMware Player¹⁵. Il passo successivo per poter simulare il comportamento di un beacon è stata l'installazione della applicazione di Matthew Robinson citata nel primo capitolo della tesi e di cui si decide di non esporre i dettagli implementativi; collegando quindi un dongle BLE alla macchina utilizzata e lanciando tale applicazione, si ottiene un beacon

¹⁵ Guida completa al seguente link: <http://www.sysprobs.com/working-os-x-10-9-mavericks-vmware-image-for-windows-os-intel-processor>

virtuale che effettua l’advertising tanto desiderato (UUID, potenza di trasmissione, Major e Minor modificabili lato utente).

Per lo sviluppo dell’applicazione Android che permette di verificare se gli smart-devices in dotazione riescono a rilevare correttamente le informazioni dei beacons e a valutarne la stabilità a fronte di utilizzi in scenari indoor, si è creata una semplice Activity che effettua uno “scan” dei dispositivi BLE che stanno trasmettendo informazioni nell’ambiente circostante, visualizzandole sullo schermo e segnalando visivamente quando ci si avvicina entro una certa soglia a uno di questi. A tale scopo si è deciso di sfruttare una libreria open source fornita dal team di Radius Networks e facilmente scaricabile dalla rete¹⁶, che permette in sostanza di interagire con qualsiasi marca e tipologia di beacons (simulati e non) mediante specifiche API. Si è scelto di proseguire in tale modo visto l’impossibilità di rilevare l’UUID di uno specifico dispositivo BLE tramite le API Android: infatti, attualmente non si ha ancora un supporto completo per ottenere le informazioni caratteristiche dei beacons, ma si possono soltanto ottenere gli identificativi univoci dei servizi che il dispositivo BLE offre (come il supporto audio, consumo batteria, termometro, cardiofrequenzimetro, ecc.) e il nome non univoco.

Prima di tutto, bisogna dichiarare nel file Manifest dell’applicazione Android i permessi relativi alla compatibilità BLE, come mostrato nella seguente figura:

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

Nella Activity invece basterà implementare l’interfaccia “IBeaconConsumer” della libreria utilizzata, in cui si sfrutta un oggetto di classe “IBeaconManager” per connettersi al servizio che consente di ottenere le notifiche sui beacons rilevati nell’ambiente circostante. In figura 4.2 si mostra parte del codice relativo al metodo specifico che permette questa funzionalità e si mostra l’utilizzo di alcune delle API della libreria.

¹⁶ <http://developer.radiusnetworks.com/ibeacon/android/>

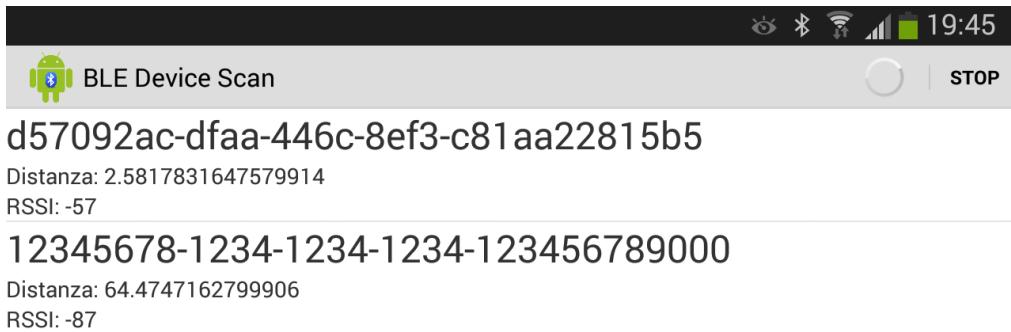
```

@Override
public void onIBeaconServiceConnect() {
    iBeaconManager.setRangeNotifier(new RangeNotifier() {
        @Override
        public void didRangeBeaconsInRegion(Collection<IBeacon> iBeacons, Region region) {
            if(iBeacons.size() > 0){
                beacons = new IBeacon[iBeacons.size()];
                int i = 0;
                for(Iterator<IBeacon> it = iBeacons.iterator(); it.hasNext(); i++){
                    beacons[i] = it.next();
                    it.remove();
                }
                // Memorizza le informazioni relative di ogni singolo beacons
                for(int j=0; j<beacons.length; j++){
                    beacons[j].getProximityUuid(); // UUID
                    beacons[j].getRssi(); // RSSI
                    beacons[j].getAccuracy(); // Distanza stimata (in metri)
                }
            }
        });
        try {
            iBeaconManager.startRangingBeaconsInRegion(
                new Region("myRegionId", null, null, null));
        } catch (RemoteException e) {
        }
    }
}

```

Figura 4.2: Codice relativo al riconoscimento dei beacons rilevati nell'ambiente.

Nella figura seguente invece si mostra un esempio di come l'applicazione sviluppata mostra le varie informazioni memorizzate, in particolare nel caso in cui si rilevano due beacons nell'ambiente circostante:



Un'annotazione riguarda la distanza stimata dal beacon (in metri): purtroppo l'algoritmo utilizzato da Radius Networks per il calcolo della cosiddetta “accuracy” non ha la stessa stabilità di quello impiegato da Apple (e non noto al pubblico), e fluttua considerevolmente in base al rumore sull'intensità del segnale trasmesso dal beacon. Questa misura è ancora più disturbata in presenza di ostacoli che si frappongono fra il device e il beacon (soprattutto liquidi, incluso quindi il corpo umano). Tale algoritmo tiene in considerazione i valori di RSSI e potenza di trasmissione, ma per le caratteristiche intrinseche di Android può essere effettuata solo una misurazione ogni 1.1 secondi, mentre su iOS è possibile ottenere una misura diversa per ogni advertisement trasmesso.

Quindi se si ha un beacon che trasmette 30 volte al secondo, iOS sarà in grado di ottenere 300 campioni in un periodo di 10 secondi mentre Android solo 9. Questo spiega perché la stima sulla distanza ha un rumore più alto su Android. Anche se non si ha la possibilità di verificare questa differenza per mancanza di smartphones Apple, utilizzando quelli Android in dotazione si decide di raccogliere una serie di dati per dimostrare l'entità delle oscillazioni sul calcolo della distanza, confrontando i risultati ottenuti anche in base al tipo di beacon utilizzato e analizzare se ci sono sostanziali differenze.

4.2.1 Confronto tra beacons

Come presentato in precedenza, i primi dongle BLE USB acquistati sono stati quelli marcati Inateck, aventi la caratteristica di montare un chipset CSR (Cambridge Silicon Radio) compatibile con qualsiasi sistema operativo. Successivamente si sono acquistati dongle con chipset Broadcom marcati IOGear, anche in previsione di futuri utilizzi mediante schede Raspberry PI (che non si è avuto poi il tempo di sperimentare), e i beacons veri e propri di Stick N Find e Accent Advanced System (iBKS).

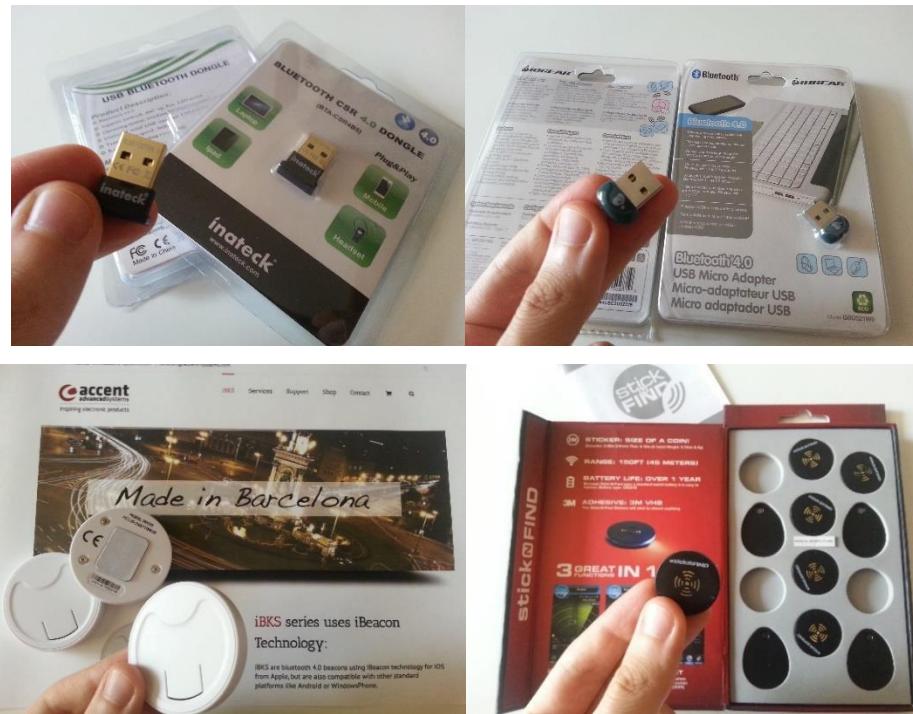


Figura 4.3: Materiale acquistato per sperimentare la tecnologia BLE.

Visto che è l'accuratezza l'elemento fondamentale ai fini della localizzazione indoor, risulta interessante verificare come varia la stabilità del segnale BLE

rilevato in base al tipo di beacons utilizzato, anche per dimostrare gli eventuali limiti dati da Android. Per questo si decide di posizionarsi a varie distanze reali (1, 5 e 10 metri dal beacon, senza ostacoli interposti) e senza spostarsi si prende nota di quanto ci si discosta dal valore effettivo, facendo una media delle prime dieci misurazioni rilevate. Oltre al valore corrispondente alla distanza stimata, si tiene conto anche di quello relativo all’RSSI.

Un’annotazione riguarda i beacons Stick N Find e iBKS: essi necessitano di essere “attivati” inizialmente per poterli rilevare, scaricando le relative applicazioni di configurazione dal Google Play Store, come mostrato nelle figure 4.4 e 4.5.



Figura 4.4: L’applicazione di configurazione per i beacons di Stick N Find.

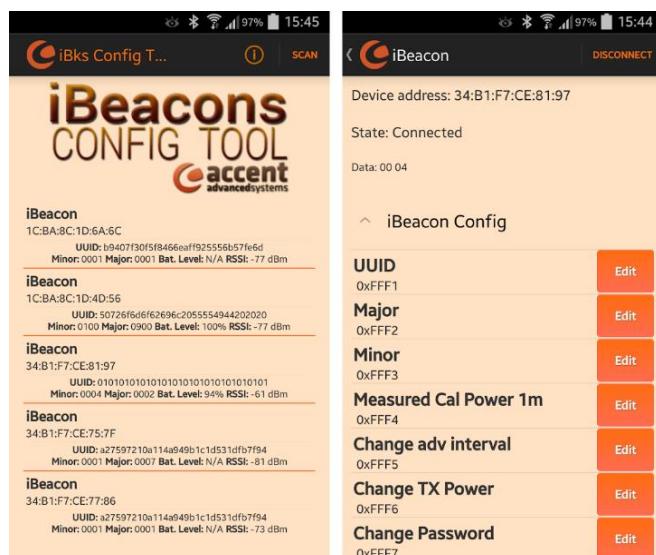
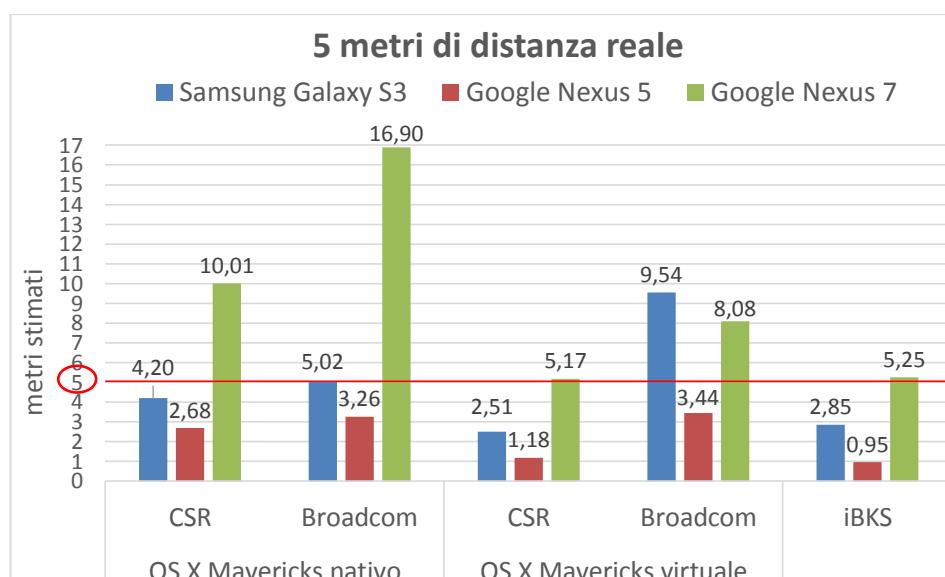
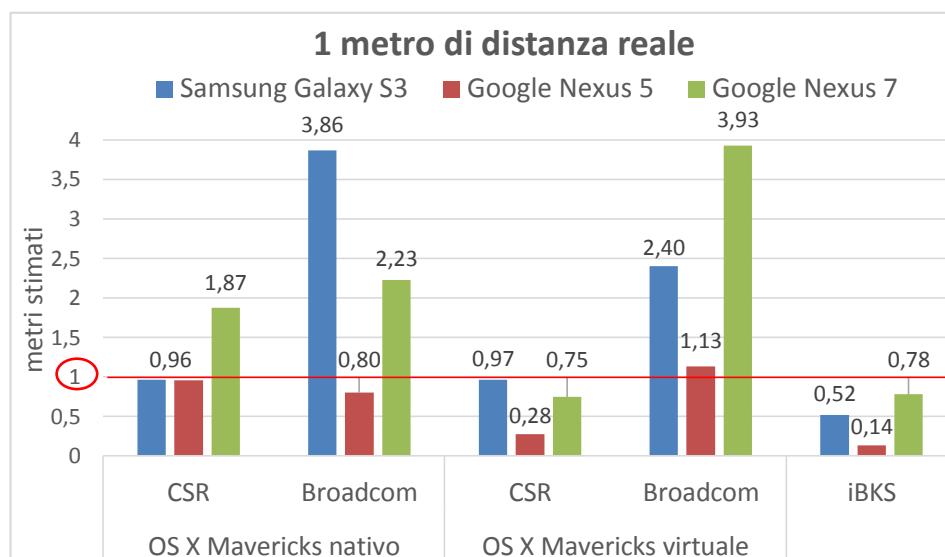
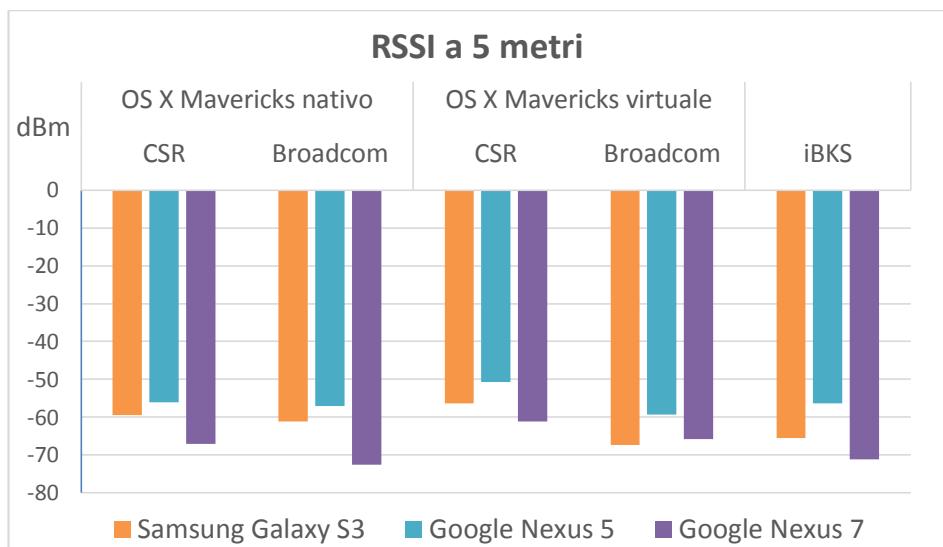
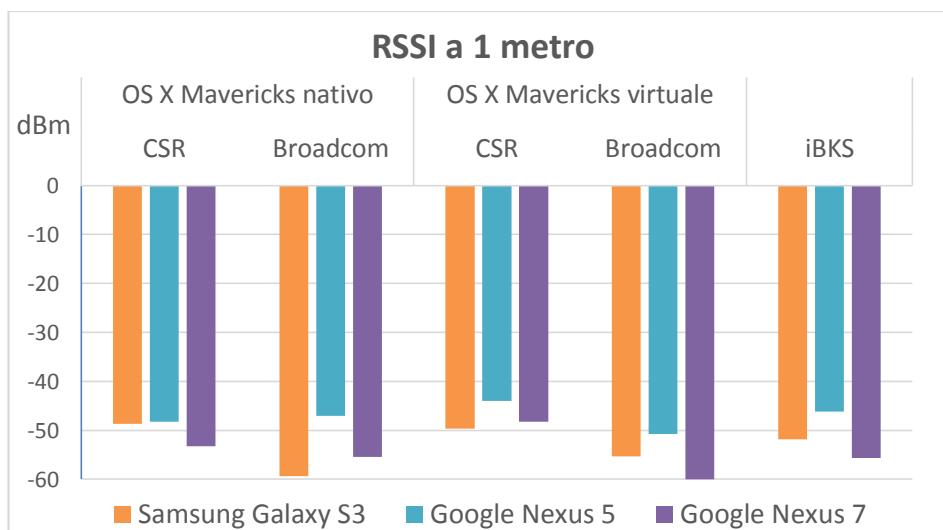
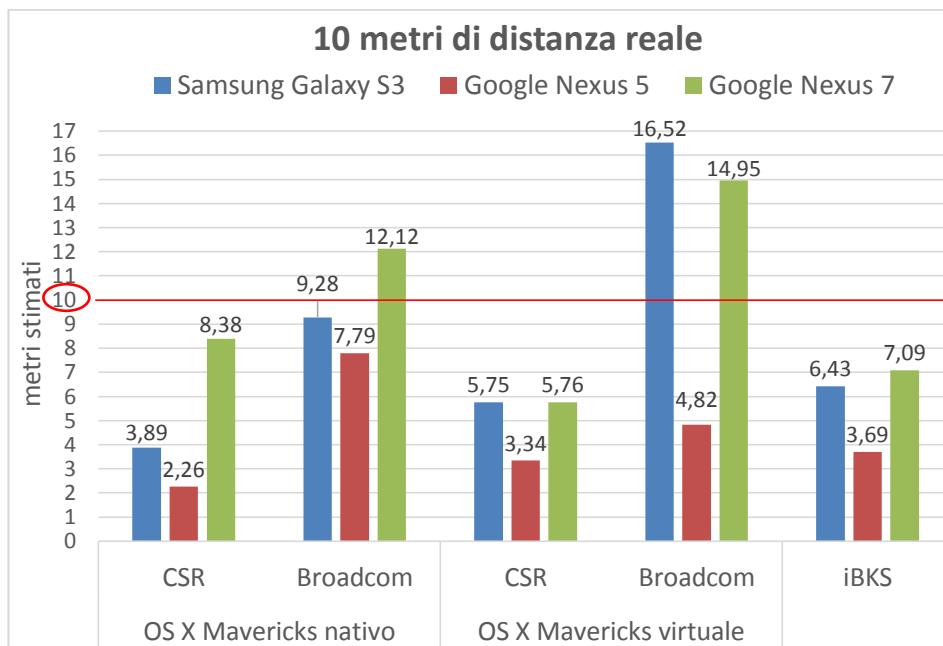


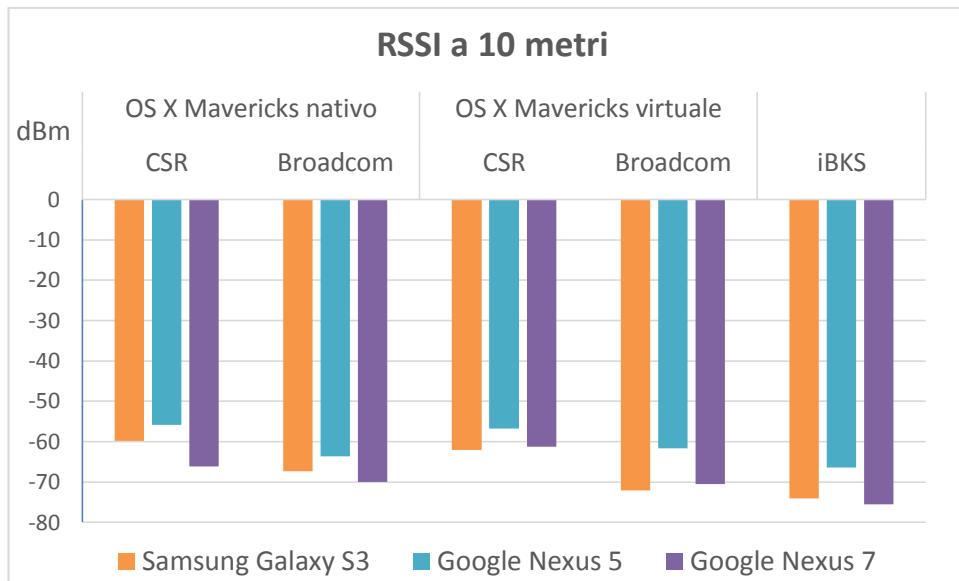
Figura 4.5: L’applicazione di configurazione per i beacons iBKS.

Una volta configurati tali beacons si è passati a verificare se l'applicazione sviluppata (che ricordiamo sfrutta le API fornite da Radius Network) li poteva rilevare senza problemi. Per quanto riguarda quelli di iBKS non si sono avute particolari difficoltà, mentre quelli di Stick N Find sono risultati impossibili da rilevare e pertanto si è deciso di non utilizzarli per sviluppi futuri: a quanto pare essi possono essere rilevati solo mediante API proprietarie. A conferma di questo si è provato a rilevarli anche con l'applicazione ufficiale di Radius Network, ma i risultati sono stati negativi.

Successivamente si è proceduto alla raccolta delle misurazioni effettuate, sintetizzando i risultati ottenuti nei seguenti grafici riepilogativi:







Analizzando i risultati conseguiti si è giunti alla conclusione che l'accuratezza peggiora con l'aumentare della distanza dal beacon, e che in genere tutti i devices sono imprecisi più o meno alla stessa maniera. Inoltre, sembra che i risultati non dipendano da un fattore in particolare, ma la loro instabilità può essere dovuta a varie cause legate al particolare momento in cui si stanno prendendo i dati: cali di tensione sui beacons, disturbi elettromagnetici nell'ambiente, rumore transitorio sul sensore degli smart-device, ecc.

In sostanza non si riesce a trovare una correlazione che sappia spiegare in modo accurato a cosa sono dovute le oscillazioni misurate, ma visto che comunque l'errore è contenuto in pochi metri si decide di mantenere in considerazione l'utilizzo della tecnologia BLE per la parte relativa alla localizzazione indoor. Grazie alle informazioni ricavate sul valore di RSSI si riesce inoltre ad avere un'idea del valore "medio" che si potrà poi utilizzare nel futuro sviluppo della applicazione relativa al caso di studio, in modo da definire la soglia entro cui si sceglie di essere nelle immediate vicinanze del beacon rilevato. Questo valore sarà pari a -60 dBm.

Capitolo 5

Infrastruttura Location-based

In questo capitolo verrà illustrato come si è deciso di modificare il primo prototipo dell’infrastruttura basata sulla singola tecnologia NFC, in modo da ampliare gli scenari di utilizzo e gli ambiti applicativi al fine di una localizzazione “completa”, che sfrutta quindi la tecnologia BLE per gli scenari indoor e il GPS per quelli outdoor. Lo scopo è quello di modellare il sistema in una struttura modulare che consenta di separare le parti specifiche relative ad ogni singola tecnologia, e permettere l’aggiunta dinamica di nuove all’occorrenza (come se si potesse fare una sorta di “plug-in”).

Inoltre si è deciso di gestire dinamicamente la scelta di quale tecnologia implementata adottare, a seconda delle rilevazioni provenienti dai sensori degli smart-devices.

5.1 Re-ingegnerizzazione del primo prototipo

Come visto nel capitolo precedente, il primo prototipo non prevede moduli specifici per la gestione della tecnologia NFC, ma il tutto è integrato all’interno del sottosistema GeoNode. In particolare è la BaseActivity che si occupa della parte sensoristica in cui viene avviato e gestito il processo di rilevazione di un particolare tag NFC identificativo di una determinata posizione del tappeto. Avendo stabilito di introdurre all’occorrenza nuove tecnologie (come il BLE e il GPS) e di mantenere le funzionalità di base comune ad ogni Activity specifica (racchiuse appunto nella BaseActivity), si decide di delegare la gestione del riconoscimento dei tag e in generale delle varie tecnologie a delle entità a parte specializzate nel recupero di tali informazioni provenienti dai sensori degli smart-devices. Queste entità saranno denominate “TechNFC”, “TechBLE” e “TechGPS”. Inoltre, visto che si vuole avere la possibilità di gestire autonomamente quale di queste tecnologie utilizzare in base a ciò che lo smart-device rileva dai propri sensori, si decide di creare un’entità che si occupa di definire una priorità tra le varie tecnologie e di gestire il “passaggio” da una all’altra; questa entità sarà denominata “GeoTechManager”.

Un’importante modifica riguarda anche ciò che viene comunicato quando si rileva uno spostamento del nodo nello spazio: ora non ho più soltanto l’id di un certo tag NFC corrispondente a una determinata posizione nel tappeto, ma ho pure la lista dei beacons BLE se mi trovo all’interno dell’edificio e le coordinate GPS se mi trovo all’esterno. Per questo motivo si decide di creare un’entità “GeoTech” che racchiude al suo interno tutte e tre queste informazioni. Inoltre, non si ha più un riferimento sulla distanza effettiva (in centimetri) tra le diverse posizioni del tappeto, ma si vuole generalizzare il concetto di distanza in modo che sia la stessa per tutti i tipi di tecnologia utilizzati: si decide pertanto di ricondurre il tutto alle coordinate terrestri, in modo da essere localizzati in una specifica posizione del globo.

Per la modifica dell’infrastruttura si decide di agire anche lato Server, andando a creare un file di configurazione in cui si stabiliscono a priori tutte le informazioni relative alle posizioni (in coordinate terrestri) fissate per i tag NFC e i beacons presenti nell’ambiente. Inoltre, per la gestione della nuova posizione del nodo a fronte di una segnalazione dal GeoClient, si decide di adottare comportamenti diversi in base alle tecnologie utilizzate, creando quindi tre entità separate: “NFCNeighbors”, “BLENeighbors” e “GPSNeighbors”.

5.1.1 Descrizione delle nuove entità introdotte

Passiamo ora a definire nel dettaglio tutte le entità nuove introdotte.

Per quanto riguarda il sottosistema GeoServer, è stata creata:

- un’entità “ConfigFileReader” che si occupa di prelevare dal file di configurazione gli identificativi e le coordinate terrestri del tappeto NFC e dei beacons presenti nell’ambiente;
- un’entità “NFCNeighbors” che si occupa di fornire le coordinate terrestri precise corrispondenti a un certo tag NFC sopra al quale si trova un nodo dato;
- un’entità “BLENeighbors” che si occupa di fornire le coordinate terrestri di un nodo dato a fronte di una lista di beacons rilevati;
- un’entità “GPSNeighbors” che si occupa di fornire le coordinate terrestri corrispondenti alla posizione di un nodo nello spazio.

Per quanto riguarda la parte Client, si è agito dentro il sottosistema GeoNode decidendo di creare:

- un’entità “GeoTechManager” che si occupa di notificare un nuovo cambiamento di posizione di un nodo in base alle tecnologie utilizzate, e anche di definire gli intervalli di tempo che consentono il passaggio dinamico da una tecnologia a un’altra sulla base di ciò che si rileva dai sensori del nodo Client;
- un’entità “TechNFC” che si occupa di rilevare (e segnalare) se il nodo è sopra un certo tag NFC identificativo di una determinata posizione sul tappeto, oppure non lo è più;
- un’entità “TechBLE” che si occupa di rilevare (e segnalare) quali beacons BLE sono percepiti dal nodo;
- un’entità “TechGPS” che si occupa di rilevare (e segnalare) la locazione (in coordinate terrestri) del nodo, sfruttando il segnale GPS.

Come già preannunciato prima, si è deciso anche di creare un’entità “GeoTech” all’interno della parte protocollare che modella cosa viene scambiato nelle comunicazioni fra le varie entità del sistema. Questa entità si occupa quindi di fornire la possibilità di ottenere e impostare le informazioni relative a: ID del tag NFC, lista dei beacons BLE, e coordinate terrestri date dal GPS.

5.1.2 Progetto delle nuova infrastruttura

Ora passiamo a “collocare” le entità descritte in precedenza nella nuova architettura di progetto dell’infrastruttura. Tutte le varie entità sono state implementate mediante semplici classi Java, mentre per quanto riguarda il file di configurazione esso viene realizzato in formato testuale e collocato all’interno della directory di progetto “GeoServer”.

Riguardo il file di configurazione, per ottenere in modo automatico tutte le coordinate terrestri (latitudine e longitudine) dei tag presenti sul tappeto NFC si sono utilizzati i calcoli matematici relativi alla lunghezza longitudinale equivalente a una certa latitudine ricavati dal portale di Wikipedia¹⁷. In Italia, essendo la latitudine di 45 gradi circa, la variazione di longitudine è diversa

¹⁷ http://en.wikipedia.org/wiki/Geographic_coordinate_system

rispetto a quella che si ha all'equatore o ai poli, ed è pari a 7.89 metri se si prendono in considerazioni spostamenti di 0.0001 gradi. La variazione corrispondente alla latitudine invece rimane quella indicata nella tabella utilizzata durante la sperimentazione sul GPS per definire l'accuratezza, ovvero pari a 11.13 metri (sempre per spostamenti di 0.0001 gradi). Si riporta di seguito la tabella completa relativa a tali variazioni data una certa latitudine “ ϕ ”¹⁸:

ϕ	Δ_{LAT}^1	Δ_{LONG}^1
0°	110.574 km	111.320 km
15°	110.649 km	107.550 km
30°	110.852 km	96.486 km
45°	111.132 km	78.847 km
60°	111.412 km	55.800 km
75°	111.618 km	28.902 km
90°	111.694 km	0.000 km

Riassumendo, la variazione sull'asse x (relativa alla longitudine) corrispondente a 1 metro è pari a 1.26e-5, mentre quella sull'asse y (relativa alla latitudine) risulta essere pari a 8.985e-6. Siccome sul tappetino NFC le posizioni sono a una distanza orizzontale pari a circa 21 centimetri e verticale di circa 12 centimetri l'una dall'altra, il passo da utilizzare per la creazione delle coordinate viene ottenuto con dalle seguenti formule:

$$\text{Asse } x = 0.21 \text{ m} * 1.26e-5 \text{ m} = 2.646e-4 \text{ m}$$

$$\text{Asse } y = 0.12 \text{ m} * 8.985e-6 \text{ m} = 1.0782e-6 \text{ m}$$

Sperimentando sui valori ottenuti durante le effettive misurazioni, ci si accorge che in realtà le distanze non sono proprio 21 e 12 centimetri: infatti, la variazione in orizzontale risulta essere di circa 30 centimetri, mentre quella verticale è pari a circa 6 centimetri. Per questo si introduce un piccolo fattore di “fix” per rendere tali distanze uguali a quelle reali, pari a 0.15 per le variazioni orizzontali e 0.24 per quelle verticali. I rispettivi passi diventano quindi i seguenti:

$$\text{Asse } x = 0.21 \text{ m} * 1.26e-5 \text{ m} * 0.15 = 3.969e-5 \text{ m}$$

$$\text{Asse } y = 0.12 \text{ m} * 8.985e-6 \text{ m} * 0.24 = 2.58768e-7 \text{ m}$$

¹⁸ Dati tratti dal portale Wikipedia e presenti al seguente link:

http://en.wikipedia.org/wiki/Length_of_a_degree_of_latitude#Length_of_a_degree_of_latitude

Ora basterà solo scegliere un punto in coordinate terrestri dove posizionare il tappeto NFC e definirne l'orientazione rispetto al centro della Terra (ovvero rispetto alle coordinate nulle) per impostare in maniera opportuna le variazioni di latitudine e longitudine. Il codice relativo al metodo che gestisce la creazioni di tali coordinate del tappeto NFC è ovviamente posto all'interno del ConfigFileReader, è riportato di seguito:

```
private void generateAndAddCarpet(Cartesian2dCoordinates origin, double orientation) {
    for (int i = 0; i < carpetSizeX; i++) {
        for (int j = 0; j < carpetSizeY; j++) {
            int s = i * carpetSizeY + j;
            tagIDs[s] = "" + (i + j);
            double x = origin.getX() + i * 0.15 * 1.26e-5;
            double y = origin.getY() + j * 0.24 * 8.985e-6;
            tagsLat[s] = x * Math.cos(orientation) - y * Math.sin(orientation);
            tagsLong[s] = x * Math.sin(orientation) + y * Math.cos(orientation);
            tagsCoords[s] = new Cartesian2dCoordinates(tagsLat[s], tagsLong[s], 0);
        }
    }
}
```

Una volta definite tutte le coordinate sarà come già detto il ConfigFileReader a leggerle e memorizzarle per ottenere la posizione precisa di ogni tag NFC e beacons BLE, le quali saranno poi richieste dai relativi tecnology-Neighbors. In particolare il BLENeighbors calcola la posizione del nodo in base a due condizioni: se entra nelle immediate vicinanze di un beacon assume le coordinate di quest'ultimo, altrimenti fa una media semplice delle posizioni di tutti i beacons rilevati e si pone in corrispondenza delle coordinate ottenute. Il valore di RSSI scelto per definire la soglia sopra la quale si è all'interno di questo “range” è pari a quello medio ottenuto dalle rilevazioni in fase di sperimentazione, ovvero -60 dBm. Il codice relativo a tale implementazione è mostrato in figura 5.1.

Per ora questo è solo un primo prototipo, in futuro sarebbe più accurato fare una media ponderata su tutti i vari valori di RSSI e costruire un algoritmo di calcolo più complesso, anche sulla base delle informazioni caratteristiche dell'ambiente in cui si decide di posizionare i beacons.

La parte progettuale relativa al Server viene mostrata in figura 5.2.

```

public Cartesian2dCoordinates getMyPosition(GeoTech geotech) {
    Cartesian2dCoordinates mycoords = new Cartesian2dCoordinates(0,0,0);
    beacons = geotech.getBeacons();
    allBeacons = file.getBeaconsNumber();
    for(int i=0; i<beacons.length; i++){
        for(int j=0; j<allBeacons; j++){
            if(beacons[i].getProximityUuid().equals(file.getBeaconUUID(j))){
                if(beacons[i].getRssi()>-60){
                    mycoords = file.getBeaconCoords(j);
                    countBeacons = 0;
                    break;
                }else {
                    countBeacons++;
                    mycoords = new Cartesian2dCoordinates(
                        mycoords.getX()+file.getBeaconCoords(j).getX(),
                        mycoords.getY()+file.getBeaconCoords(j).getY(),
                        mycoords.getZ()+file.getBeaconCoords(j).getZ());
                }
            }
        }
        if(countBeacons==0)
            break;
    }
    if(countBeacons>0){
        mycoords = new Cartesian2dCoordinates(mycoords.getX()/countBeacons,
                                              mycoords.getY()/countBeacons,
                                              mycoords.getZ()/countBeacons);
        countBeacons = 0;
    }
    return mycoords;
}

```

Figura 5.1: Codice relativo al calcolo della posizione attuale in base ai beacons rilevati.

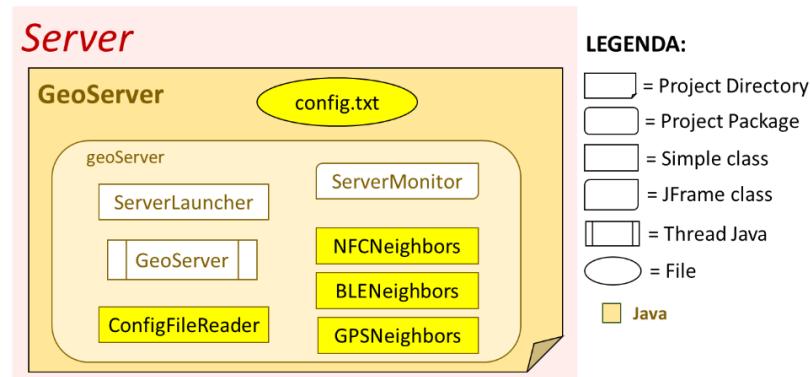


Figura 5.2: Rappresentazione parte Server della nuova infrastruttura (in giallo le novità).

Relativamente alla parte del Client, si agisce come detto in precedenza solo sul progetto Android corrispondente al sottosistema GeoNode; per quanto riguarda la classe Java “TechBLE” che si occupa di rilevare quali beacons sono percepiti dal nodo, si decide di riusare parte dell’applicazione BLE sperimentale descritta nel capitolo precedente (e quindi sfruttare le API fornite dalla libreria open source di Radius Networks). Stessa cosa per quanto riguarda la classe Java “TechGPS”, in cui ci si avvale della stessa modalità di localizzazione utilizzata

per l'applicazione GPS sperimentata. Riguardo il GeoTechManager, per la gestione dinamica del cambio di tecnologia si imposta un semplice controllo prima di notificare l'aggiornamento di posizione: si decide di definire delle “classi di priorità” in ordine di accuratezza, ovvero prima si verifica se si è sopra un tag NFC, altrimenti si passa a controllare se rilevo qualche beacon BLE e infine se ho una certa coordinata data dal GPS. Il codice relativo a questa gestione è mostrato nella seguente figura:

```

private void update(int entryClass) {
    int computedPosClass ;
    if (entryClass == CLASS_NULL) {
        if (state.getTag() != -1) {
            computedPosClass = CLASS_NFC;
        } else if (state.getBeacons() != null) {
            computedPosClass = CLASS_BLE;
        } else if (state.getGPSCoords() != null) {
            computedPosClass = CLASS_GPS;
        } else
            computedPosClass = CLASS_NULL;
    } else
        computedPosClass = entryClass;
    if (computedPosClass <= oldPositionClass) {
        oldPositionClass = computedPosClass;
        notifyPositionListeners();
    }
}

```

Una rappresentazione che mostra come vengono collocate queste nuove entità relative al lato Client è ritratta in figura 5.3 (sono state omesse e abbreviate con i tre puntini di sospensione quelle relative ai progetti Java, visto che rimangono invariati rispetto al vecchio prototipo).

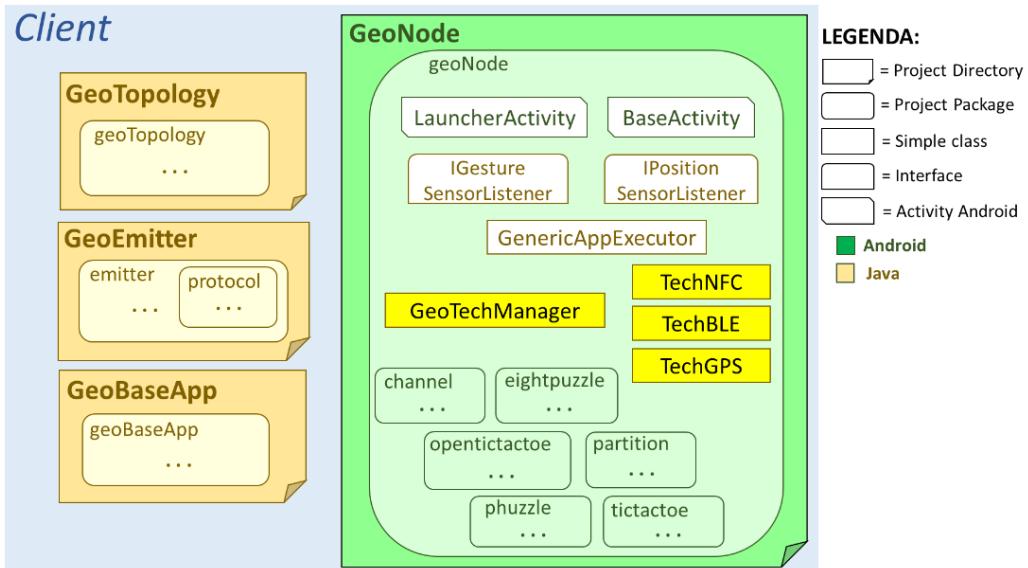


Figura 5.3: Rappresentazione delle novità introdotte nell’infrastruttura per la parte Client.

In questo modo si è riusciti a rendere modulare l’infrastruttura introducendo apposite classi specifiche per la gestione di ogni singola tecnologia location-based, e scegliendo di affiancare BLE e GPS all’NFC si sono ricoperte tutte le esigenze riguardanti la localizzazione seppure con i limiti riscontrati durante la fase di sperimentazione.

In figura 5.4 si riporta lo schema complessivo che mette in evidenza le interazioni tra le varie parti dell’infrastruttura, andando quindi a delineare la nuova architettura di progetto.

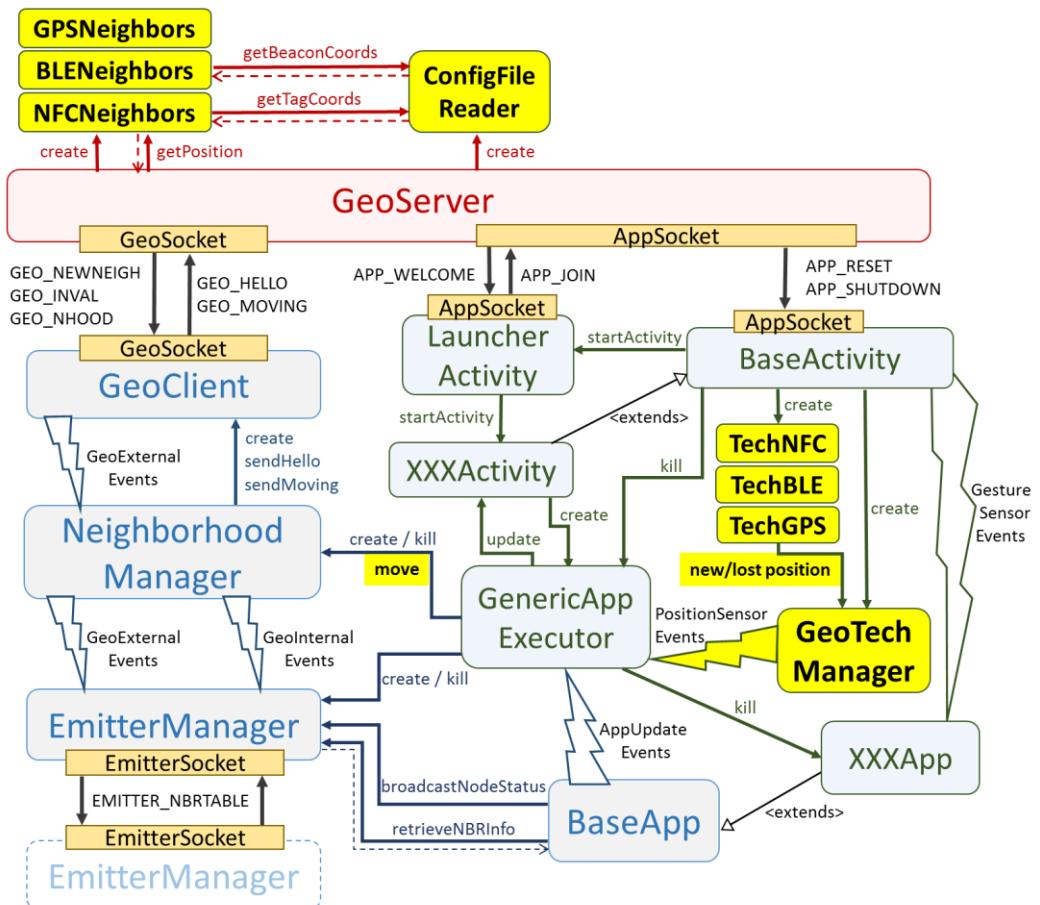


Figura 5.4: Architettura di progetto della nuova infrastruttura.

Capitolo 6

Caso di studio

Per dimostrare le funzionalità della nuova infrastruttura sviluppata si decide di realizzare un reale caso applicativo che sarà denominato “Flag Finder”. Come si intuisce dal nome, tale applicazione Android permetterà all’utente di trovare un determinato punto di interesse rappresentato da uno smart-device “bandiera” collocato in una delle posizioni del tappetino NFC utilizzato per il primo prototipo dell’infrastruttura, e situato all’interno di una stanza dell’edificio sede dei corsi di laurea in Ingegneria e Scienze Informatiche (Cesena, via Sacchi n.3). Partendo quindi da un ambiente outdoor esterno alla facoltà e sfruttando la tecnologia GPS si guida l’utente verso la localizzazione dell’edificio di interesse; una volta arrivati a destinazione si sfrutta il segnale BLE emesso dai vari beacons posizionati all’interno della facoltà per muoversi verso l’aula in cui è presente il dispositivo bandiera. All’interno di tale stanza sarà presente l’ormai noto tappeto NFC sul quale appoggiare lo smart-device dell’utente, e sfruttando tale tecnologia verrà indicato il percorso che porta alla destinazione.

6.1 Dettagli implementativi

Per realizzare questa applicazione si è dunque partiti dallo schema implementativo caratteristico delle applicazioni già sviluppate nel primo prototipo dell’infrastruttura, ovvero si è creata:

- una classe “FlagFinderApp” che estende la classe BaseApp e si occupa di calcolare la distanza del nodo da tutti gli eventuali vicini presenti nel sistema, e impostare la giusta direzione verso cui “puntare”;
- una classe “FlagFinderActivity” che estende la classe BaseActivity e si occupa di visualizzare sullo schermo del device le informazioni dell’applicazione, in particolare la possibilità di scegliere se essere la destinazione e quindi visualizzare la bandiera (figura 6.1), o essere il “Finder” e visualizzare una freccia orientata verso la destinazione con relativo colore dello sfondo sulla base della distanza calcolata e

comunicata ad ogni aggiornamento di stato dal GenericAppExecutor (come mostrato nella figura 6.2).

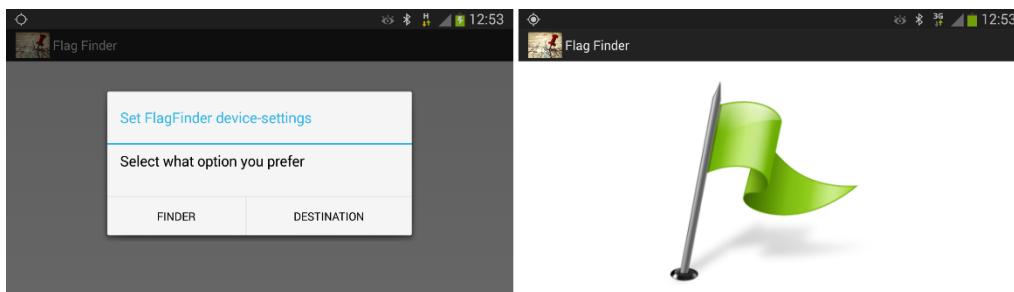


Figura 6.1: A sinistra la schermata di selezione dell'applicazione FlagFinder, a destra ciò che viene visualizzato se si sceglie di essere una destinazione.

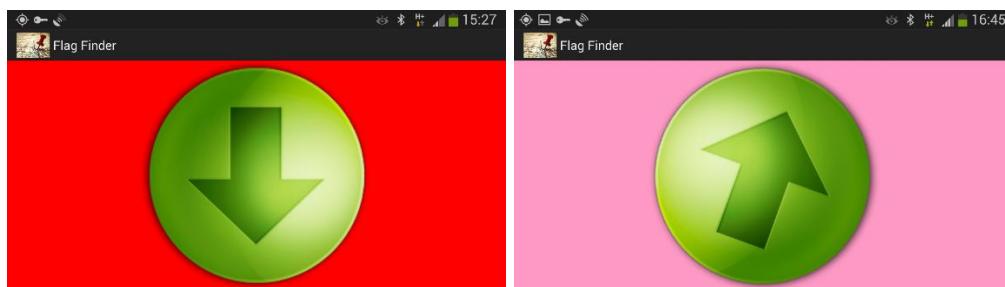


Figura 6.2: Esempi di orientazione freccia e colore sfondo per il Finder (a sinistra se si è a una distanza minore di 24.4 centimetri dalla destinazione, a destra se oltre gli 80 metri).

Per il calcolo della distanza tra coordinate diverse si è sfruttata una libreria open-source di Google¹⁹ che permette di ottenere la distanza geodetica tra due punti secondo il modello ellissoide WGS84.

Per semplicità, nel caso in esame si decide di definire nel file di configurazione un orientamento nullo in corrispondenza delle coordinate del tappeto NFC, in modo da far coincidere gli spostamenti in verticale (asse y) alla variazione di latitudine e quelli in orizzontale (asse x) alla variazione di longitudine. Esso viene collocato all'interno dell'aula 20, posta al primo piano dell'edificio.

Posizione e coordinate terrestri relative a tale tappetino sono indicate nella figura 6.3.

¹⁹ <https://code.google.com/p/amazingapis/>



Figura 6.3: Posizionamento nell’edificio del tappeto NFC.

Per il caso in esame si decide di posizionare i nostri beacons a una certa distanza tra loro (circa 15 metri) in modo da evitare situazioni in cui uno smart-device rilevi più beacons con valore di RSSI maggiore di -60 dBm, considerata anche la forte instabilità riscontrata in fase di sperimentazione. I primi cinque beacons sono posizionati al primo piano dell’edificio come per il tappeto NFC, mentre l’ultimo beacon è posto vicino all’ingresso della sede e in particolare vicino le scale che portano proprio verso il primo piano. La loro collocazione è mostrata in figura 6.4.



Beacon 1 coordinate:	44.1399077, 12.2426333	Beacon 4 coordinate:	44.1399712, 12.2431697
Beacon 2 coordinate:	44.1400425, 12.2427593	Beacon 5 coordinate:	44.1398326, 12.2432341
Beacon 3 coordinate:	44.1400540, 12.2429846	Beacon 6 coordinate:	44.1396998, 12.2431026
località: Via Sacchi, 3, 47521 Cesena FC, Italia		Altitudine: 42 meters	

Figura 6.4: Posizionamento nell’edificio dei beacons.

Siccome la struttura dell’edificio è un quadrato (come si può vedere dalla figura 6.4), non è possibile muoversi in diagonale come giustamente sarà indicato dalla freccia se si è posizionati vicino a tali beacons. Per questo motivo si sceglie di adattarsi all’ambiente e modificare le coordinate di questi per seguire correttamente scale e corridoi della facoltà, sempre cercando di mantenere una distanza dalla destinazione in linea con quella reale. Essendo il beacon 6 posto vicino alle scale a sinistra dell’entrata della sede, si decide di impostare le sue coordinate in modo che la freccia punti verso di esse quando oltrepasso l’ingresso; una volta salite le scale si impostano le coordinate del beacon 5 in maniera tale da proseguire dritto lungo il corridoio, quelle del beacon 4 per indicare una svolta a sinistra e infine quelle del beacon 3 per tornare a proseguire dritto. Le coordinate dei beacon 1 e 2 possono rimanere le stesse, visto che sono già in linea con il percorso da seguire. In figura 6.5 si mostra come vengono ottenute le nuove coordinate da assegnare a questi beacons.



Figura 6.5: Modifica delle posizioni dei beacons in base alla struttura dell’ambiente indoor.

In pratica si è cercato di mantenere le distanze reali dal punto di destinazione in modo da avere un colore di sfondo congruo, ma si sono cambiate latitudine e longitudine per lo scopo desiderato.

Per la gestione di ciò che verrà visualizzato sullo schermo dei nodi Client che fungono da “Finder”, decido di attribuire un certo colore allo sfondo dello smart-device sulla base della reale distanza ricavata dalle coordinate terrestri, e in base alla differenza tra i campi magnetici di questo e la destinazione calcolo anche l’orientazione della freccia. Per questo motivo nella FlagFinderApp devo impostare correttamente le informazioni di stato: ogni nodo memorizza le proprie coordinate e quelle del device più vicino, poi imposta la corretta direzione della freccia per gli spostamenti sul tappeto NFC.

Parte del codice che implementa questa funzionalità è riportato nella figura 6.6.

```

myPos = me.getPosition().getCoordinates();
myLocation = new Location("reverseGeocoded");
myLocation.setLatitude(myPos.getX());
myLocation.setLongitude(myPos.getY());
if (myPos != null && nextNode!=null){
    destLocation = new Location("reverseGeocoded");
    destLocation.setLatitude(nextNode.getX());
    destLocation.setLongitude(nextNode.getY());
    if (nextNode.getY()>(myPos.getY())){
        if (nextNode.getX()>(myPos.getX())){
            arrowDir = 45;
        } else if (myPos.getX()>(nextNode.getX())){
            arrowDir = 315;
        } else {
            arrowDir = 0;
        }
    } else if (myPos.getY()>(nextNode.getY())){
        if (nextNode.getX()>(myPos.getX())){
            arrowDir = 135;
        } else if (myPos.getX()>(nextNode.getX())){
            arrowDir = 225;
        } else {
            arrowDir = 180;
        }
    } else if (nextNode.getX()>(myPos.getX())){
        arrowDir = 90;
    } else {
        arrowDir = 270;
    }
}

```

Figura 6.6: Codice relativo all'impostazione delle nuove informazioni di stato per l'applicazione FlagFinder.

Riguardo la FlagFinderActivity invece, ogni volta che arriva un aggiornamento di stato si vanno a recuperare tali informazioni e si agisce di conseguenza sulla base di ciò che si è scelto di essere (destinazione o Finder) e se ci si trova sul tappeto NFC o meno. Per impostare correttamente la direzione della freccia quando sono in movimento e permettere quindi di puntare sempre verso la destinazione qualunque sia l'orientamento dello smart-device assunto, sfrutta gli eventi scatenati dai sensori dell'accelerometro e del campo magnetico.

Parte del codice che mostra come è stata gestita questa parte viene illustrato nella figura 6.7.

```

currentDegree = state.getArrowDir();
myDistance = state.getDistDst();
if (myDistance == 0) {
    getWindow().getDecorView().setBackgroundColor(Color.WHITE);
    mybitmap = flagbtmp;
    mCustomDrawableView.invalidate();
} else {
    if(myDistance!=-1){
        chooseColor(myDistance);
        if(myDistance<1.25){
            onNFC = true;
            mybitmap = arrowbtmp;
            mCustomDrawableView.invalidate();
        }else{
            onNFC = false;
            myLocation = state.getMyLocation();
            destLocation = state.getDestLocation();
            mybitmap = arrowbtmp;
            azimut = orientation[0] +90;
            azimut = azimut * 180 / (float)Math.PI;
            geoField = new GeomagneticField(
                (float) myLocation.getLatitude(),
                (float) myLocation.getLongitude(),
                (float) myLocation.getAltitude(),
                System.currentTimeMillis());
            azimut -= geoField.getDeclination();
            bearing = myLocation.bearingTo(destLocation);
            direction = azimut - bearing;
            matrix.setRotate(direction);
            mCustomDrawableView.invalidate();
        }
    }
}

```

Figura 6.7: Codice relativo alla computazione che fa seguito ad ogni aggiornamento di stato.

Ogni volta che viene richiamato l'aggiornamento della grafica, sullo schermo del device si disegna la relativa immagine impostata. Il codice relativo a tale funzionalità è mostrato in figura 6.8.

Come già detto in precedenza al variare della distanza varia anche il colore dello sfondo. Per effettuare questa discriminazione, tengo conto delle caratteristiche del tappeto NFC e delle distanze indicative tra i beacons: per gli spostamenti sul tappetino sfrutto come limite le circonferenze di raggio pari alla distanza limite data dai passi in diagonale (pari a 0.244 metri), mentre per quelli indoor considero distanze di 10 e 20 metri (ovvero la distanza che si ha circa tra un beacon e l'altro). Il codice riassuntivo che chiarisce meglio questo concetto viene mostrato in figura 6.9.

```

protected void onDraw(Canvas canvas) {
    int width = getWidth();
    int height = getHeight();
    int centerx = width/2;
    int centery = height/2;
    if(mybitmap.sameAs(flagbtmp))
        canvas.drawBitmap(mybitmap, centerx-(mybitmap.getWidth()/2),
                          centery-(mybitmap.getHeight()/2), paint);
    else{
        if(!onNFC){
            if (azimut != null)
                canvas.rotate(-direction, centerx, centery);
            canvas.drawBitmap(mybitmap, centerx-(mybitmap.getWidth()/2),
                              centery-(mybitmap.getHeight()/2), paint);
        }else{
            canvas.rotate(currentDegree, centerx, centery);
            canvas.drawBitmap(mybitmap, centerx-(mybitmap.getWidth()/2),
                              centery-(mybitmap.getHeight()/2), paint);
        }
    }
}

```

Figura 6.8: Codice relativo agli aggiornamenti grafici sullo schermo del device.

```

private void chooseColor(double distance) {
    if (distance<0.244){
        getWindow().getDecorView().setBackgroundColor(Color.RED);
    } else if (distance>=0.244 && distance<0.488){
        getWindow().getDecorView().setBackgroundColor(Color.rgb(255, 175, 5)); //orange
    } else if (distance>=0.488 && distance<0.732){
        getWindow().getDecorView().setBackgroundColor(Color.YELLOW);
    } else if (distance>=0.732 && distance<0.976){
        getWindow().getDecorView().setBackgroundColor(Color.GREEN);
    } else if (distance>=0.976 && distance<1.22){
        getWindow().getDecorView().setBackgroundColor(Color.rgb(15, 120, 50)); //dark green
    } else if (distance>=1.22 && distance<10){
        getWindow().getDecorView().setBackgroundColor(Color.CYAN);
    } else if (distance>=10 && distance<20.0){
        getWindow().getDecorView().setBackgroundColor(Color.rgb(0, 115, 200)); //light blue
    } else if (distance>=20.0 && distance<40.0){
        getWindow().getDecorView().setBackgroundColor(Color.BLUE);
    } else if (distance>=40.0 && distance<60.0){
        getWindow().getDecorView().setBackgroundColor(Color.rgb(95, 0, 200)); //violet
    } else if (distance>=60.0 && distance<80.0){
        getWindow().getDecorView().setBackgroundColor(Color.MAGENTA);
    } else if (distance>=80.0){
        getWindow().getDecorView().setBackgroundColor(Color.rgb(253, 153, 196)); //pink
    }
}

```

Figura 6.9: Codice relativo alla scelta del colore di sfondo sulla base della distanza misurata dalla destinazione.

6.2 Test e Demo

Terminata la fase di implementazione si è passati a testare quanto sviluppato per verificare l’effettivo funzionamento dell’applicazione.

Innanzitutto si necessita di instaurare una connessione per far comunicare i nodi Client tra loro e con il Server. Visto che gli indirizzi IP di Server e Client devono poter essere raggiungibili dai nodi del sistema per permettere le relative comunicazioni, si è deciso di creare una rete virtuale privata (VPN) appoggiandosi su AlmaWifi. Si è dovuto procedere in questo modo a causa della policy interna della rete di Ateneo, la quale non permette comunicazioni dirette fra dispositivi Client all’interno della rete. Inizialmente si era testato il sistema con una connessione tramite hotspot creato su uno dei device in dotazione, ma la potenza del segnale ricopre un’area di circa 20 metri scarsi (visto anche le mura massicce che dividono aule e corridoi). Siccome AlmaWifi ricopre solamente il perimetro dell’edificio, si è deciso di sfruttare uno smart-device con possibilità di connessione alla rete 3G (tramite operatore di rete) per lo scenario outdoor. Per fare in modo di poter comunicare con la rete VPN dall’esterno ci si è procurati un indirizzo IP pubblico con traffico aperto in entrata e uscita.

Una volta impostata la rete, i passi che portano alla realizzazione di una demo utile alla fase di testing sono i seguenti:

- si avvia il thread del GeoServer da una macchina contenente il file di configurazione che specifica le coordinate di tappeto e beacons;
- si manda in esecuzione l’applicazione FlagFinder su un nodo Client e si sceglie di essere destinazione, collocando tale smart-device su una delle posizioni del tappeto NFC;
- si manda in esecuzione l’applicazione FlagFinder su un altro nodo Client in qualsiasi posizione dell’ambiente e si sceglie di essere “Finder”: per il caso outdoor il nodo dovrà essere uno smart-device con connettività 3G (e quindi che abbia al suo interno una scheda telefonica con operatore di rete mobile), mentre per il solo indoor si sfrutta la rete AlmaWifi e quindi si può utilizzare qualsiasi dispositivo in dotazione.

A questo punto sul device dell’utente che funge da Finder verrà visualizzata una freccia orientata verso il percorso che porta alla destinazione, con relativo colore di sfondo in base alla distanza da quest’ultima (come già descritto nel paragrafo precedente). Di seguito si riportano alcune figure che mostrano le varie situazioni particolari in cui ci si può trovare nel corso della demo.



Figura 6.10: Esempio di scenario outdoor quando ci si trova a una distanza superiore a 80 metri dalla destinazione (colore sfondo rosa).



Figura 6.11: Esempio di scenario outdoor quando ci si trova a una distanza compresa tra i 60 e gli 80 metri dalla destinazione (colore sfondo magenta).



Figura 6.12: Esempio di scenario outdoor quando ci si trova a una distanza compresa tra i 40 e i 60 metri dalla destinazione (colore sfondo viola).



Figura 6.13: Esempio di scenario outdoor quando ci si trova a una distanza compresa tra i 20 e i 40 metri dalla destinazione (colore sfondo blu).



Figura 6.14: Esempio di scenario indoor quando si è in prossimità del beacon 6 (freccia verso le scale e sfondo blu).



Figura 6.15: Esempio di scenario indoor quando si è in prossimità del beacon 5 (freccia lungo il corridoio e sfondo blu).



Figura 6.16: Esempio di scenario indoor quando si è in prossimità del beacon 4 (freccia che indica la svolta al prossimo corridoio e sfondo blu).



Figura 6.17: Esempio di scenario indoor quando si è in prossimità del beacon 3 (freccia lungo il corridoio e sfondo blu).



Figura 6.18: Esempio di scenario indooring quando si è in prossimità del beacon 2 (freccia che indica la svolta al prossimo corridoio e sfondo blu chiaro).



Figura 6.19: Esempio di scenario indooring quando si è lungo il corridoio che collega il beacon 2 con il beacon 1 (sfondo azzurro chiaro).

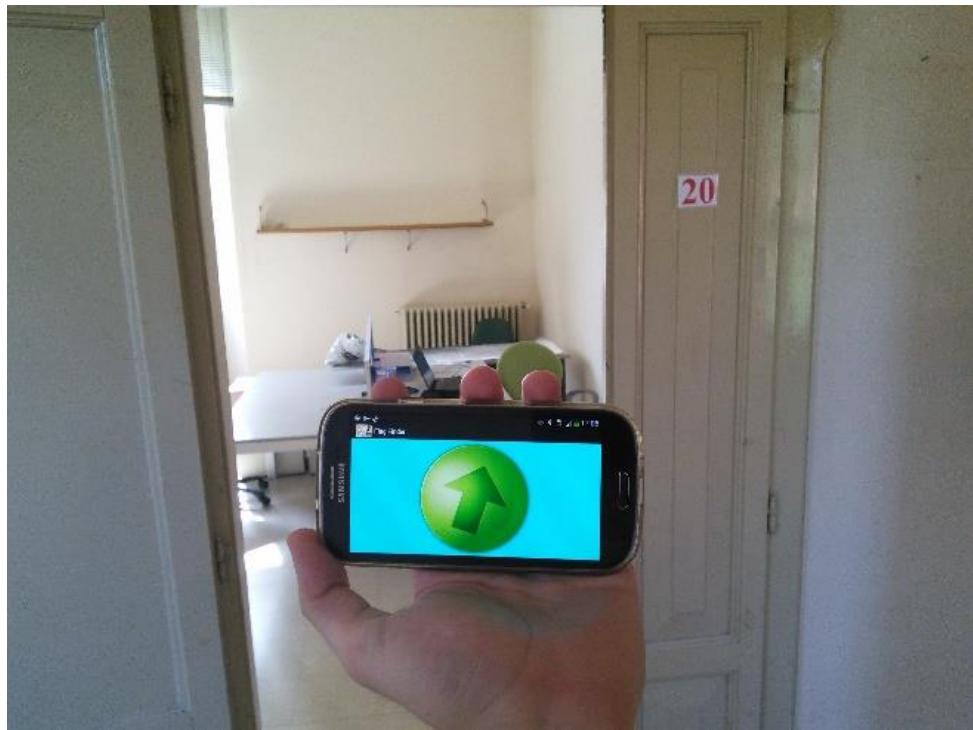


Figura 6.20: Esempio di scenario indoор quando si è in prossimità del beacon 1 (stanza contenente la destinazione).



Figura 6.21: Esempio di scenario in cui ci si trova sul tappetino NFC e a una distanza compresa tra 97.6 e 122 centimetri dalla destinazione (sfondo verde scuro).

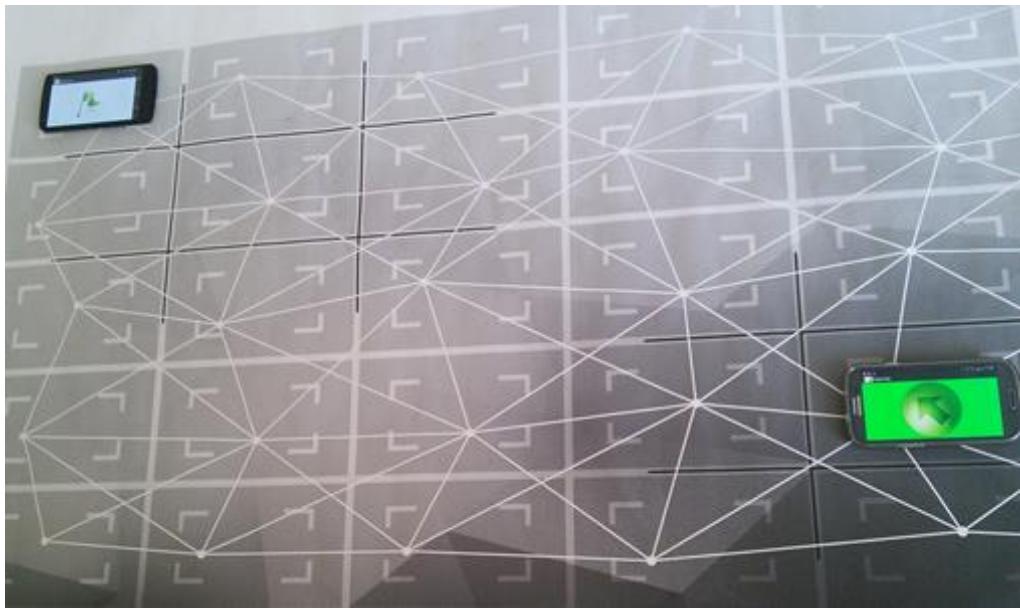


Figura 6.22: Esempio di scenario in cui ci si trova sul tappetino NFC e a una distanza compresa tra 73.2 e 97.6 centimetri dalla destinazione (sfondo verde chiaro).

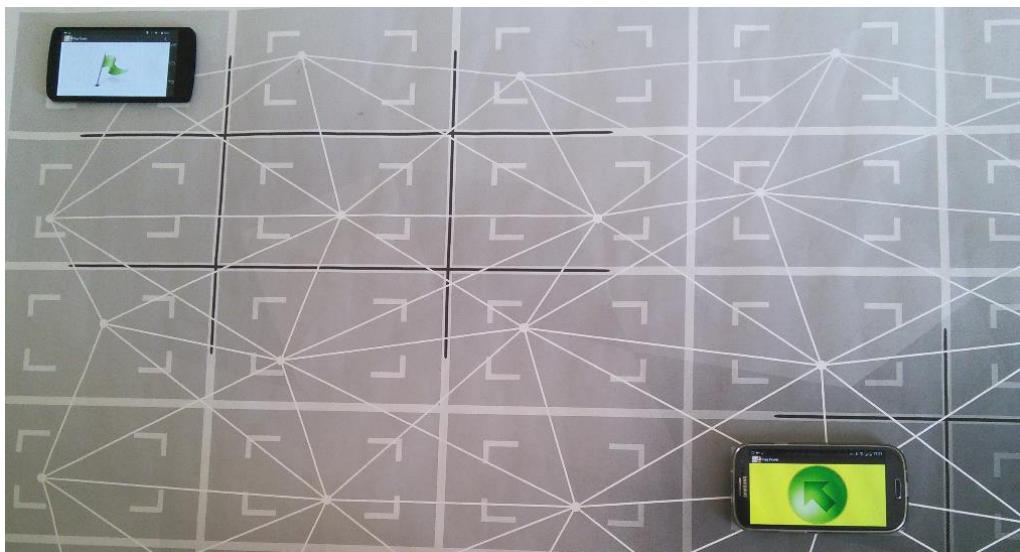


Figura 6.23: Esempio di scenario in cui ci si trova sul tappetino NFC e a una distanza compresa tra 48.8 e 73.2 centimetri dalla destinazione (sfondo giallo).

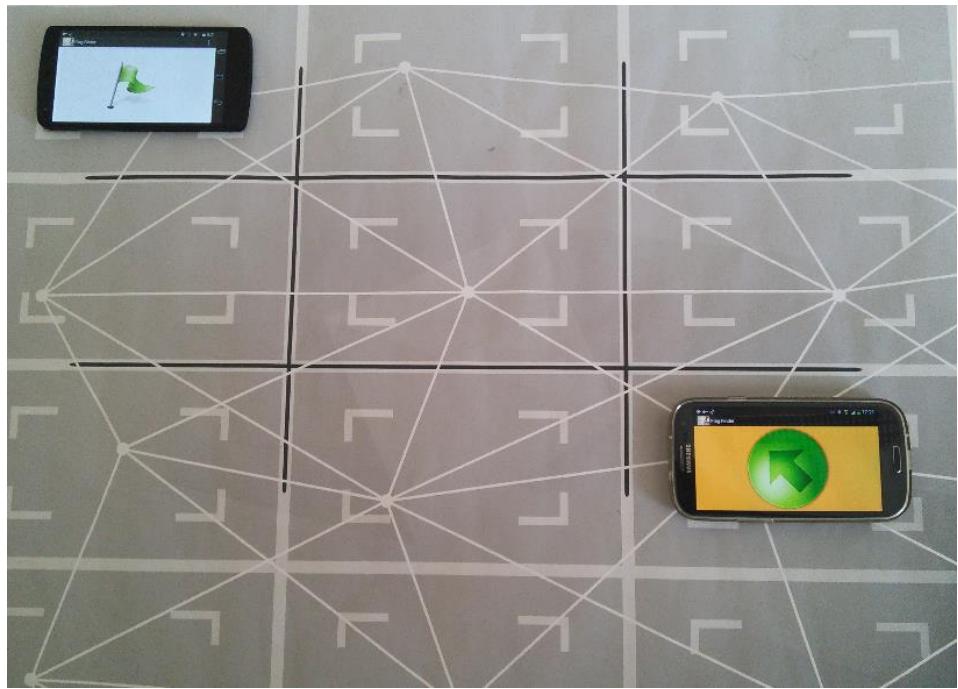


Figura 6.24: Esempio di scenario in cui ci si trova sul tappetino NFC e a una distanza compresa tra 24.4 e 48.8 centimetri dalla destinazione (sfondo arancione).

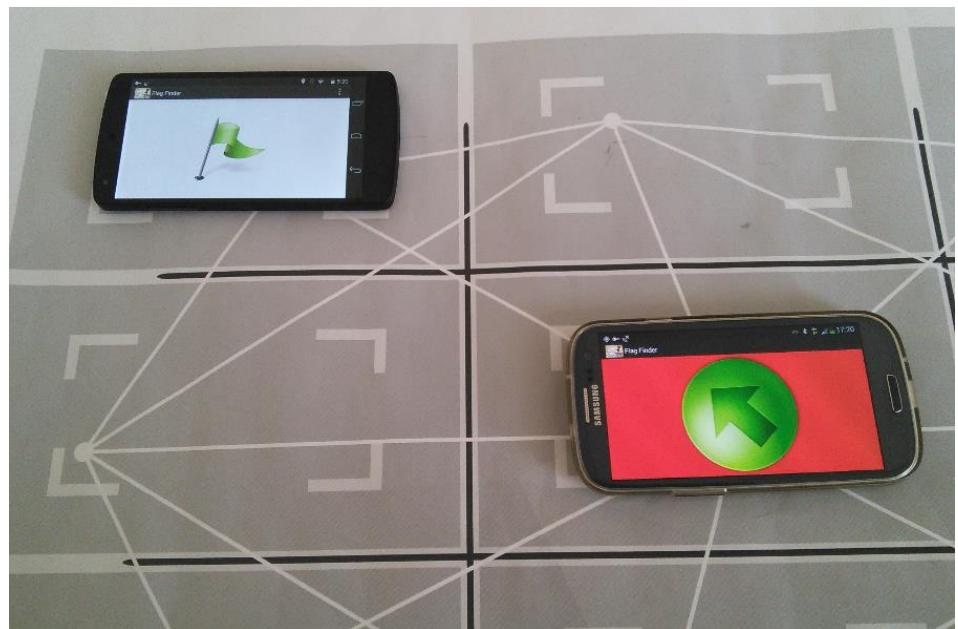


Figura 6.25: Esempio di scenario in cui ci si trova sul tappetino NFC e a una distanza inferiore a 24.4 centimetri dalla destinazione (sfondo rosso).

Nel paragrafo precedente è stato descritto il motivo per cui si adattano le coordinate dei beacons in base alla struttura dell’edificio, in modo da riuscire a ottenere un percorso orientato che segua effettivamente i corridoi della struttura. Un’alternativa a tale impostazione è data dall’utilizzo di altri smart-devices posti lungo il percorso, in special modo uno vicino ad ogni beacon: così facendo

sullo schermo dell’utente si indicherebbe la direzione che porta al beacon più vicino, e di beacon in beacon si giungerebbe fino a destinazione. Purtroppo non è stato possibile testare tale metodo per motivi legati alla rete: infatti con l’uso di molteplici devices aumenta il carico di rete, e le impostazioni attuali del Server non garantiscono sufficiente reattività nella comunicazione coi Client. Molto spesso infatti si sono verificati momenti di stallo in cui il Server si bloccava sulla ricezione di un messaggio di “moving” inviato da un certo nodo Client, oppure non riusciva più a contattarlo per l’invio delle nuove informazioni sul vicinato riscontrando quindi errori di connessione. Allo stato attuale la rete si basa su protocollo TCP e quindi il Server tenta comunque ad ogni intervallo di tempo di riconnettersi al nodo Client che non riesce a raggiungere, e questo causa un accodamento dei messaggi che intanto giungono al Server.

Queste problematiche di rete sono comunque presenti in modo saltuario anche se i devices presenti nel sistema sono pochi, e probabilmente possono essere dovute all’instabilità della rete WiFi che non garantisce sempre una potenza del segnale stabile a buoni livelli. Così risulta facile avere intervalli di tempo in cui un device per un attimo perde la connessione, e il Server si blocca in attesa di contattarlo. Si è cercato comunque di agire lato infrastruttura per gestire meglio questi problemi, visto che rispetto al primo prototipo dell’infrastruttura i messaggi scambiati sono molto più frequenti (a causa dell’instabilità del segnale BLE e il fatto di essere sempre in movimento sia indoor che outdoor) e anche di dimensioni maggiori (non scambio più solo l’id del tag, ma un oggetto di tipo GeoTech che contiene la lista dei beacons percepiti e le coordinate GPS): si è dunque agito sul GeoTechManager andando a inserire un tempo di attesa pari a tre secondi prima di ogni notifica di “moving”. Tale metodo però non ha ovviato al problema, e allora si è cercato di prolungare a cinque i secondi ma con risultati peggiori: aumentando il tempo di attesa infatti diminuisce la reattività nella localizzazione della nuova posizione, senza risolvere le frequenti problematiche di rete e errori di connessione. Per questo motivo si è giunti a un compromesso impostando un tempo di attesa pari a 1 secondo.

Un’altra soluzione che non si è avuto modo di testare in maniera approfondita riguarda l’impostazione lato GeoServer: attualmente non c’è alcun tipo di

aggregazione dei messaggi ricevuti, essi vengono tutti accodati in una lista (bloccante) che il GeoServer ciclicamente processa. Un’idea potrebbe essere il miglioramento di queste dinamiche di processamento, introducendo meccanismi di concorrenza e aggregando l’invio delle risposte ad intervalli di tempo prefissati.

6.3 Altre applicazioni

Ovviamente per motivi di tempistiche ristrette non si sono potute sviluppare altre applicazioni di interesse, ma tante sono le possibilità di utilizzo dell’infrastruttura sviluppata nei vari domini applicativi descritti in precedenza. Alcuni esempi di applicazioni possono essere i seguenti:

- Guida turistica: a partire da un qualsiasi punto outdoor cittadino, si può guidare l’utente verso una certa opera di un museo o attrazione, e una volta raggiunta si possono collocare dei tag nfc o dei beacons nelle immediate vicinanze per fornire determinate informazioni e/o applicazioni ludiche;
- Proximity Marketing: partendo da uno scenario outdoor o indoor se all’interno di un negozio o di un supermercato, si può informare il cliente su un determinato tipo di servizio o offerta, indicando anche il percorso più breve per raggiungerlo;
- Ricerca di persone: attraverso il possesso di beacons si può sviluppare un’applicazione che consente di sapere la distanza e il percorso che porta dalla posizione attuale a un amico o familiare in corrispondenza di determinati scenari crowd (come in stadi o supermercati), oppure se si vuole monitorare la posizione degli operai (e magari anche gli orari di lavoro) all’interno di un cantiere o dell’azienda.
- Servizi ospedalieri: tracciamento di pazienti, controllo degli accessi, e visualizzazione del percorso più breve a una certa stanza.

Capitolo 7

Conclusioni e sviluppi futuri

Allo stato attuale l’infrastruttura sviluppata permette di gestire al meglio le tre tecnologie presentate in questa tesi e utili per una localizzazione outdoor e indoor fino a spostamenti con precisione nell’ordine del millimetro. Inoltre si è riusciti a generalizzare il concetto di distanza basandosi sulle coordinate terrestri, riuscendo così a rendere dinamico e flessibile il cambiamento di tecnologia location-based da utilizzare in base a ciò che i sensori degli smart-devices rilevano.

Tanti rimangono comunque gli sviluppi futuri su cui ci si può focalizzare. In primo luogo ciò che riguarda scalabilità e comunicazioni di rete: la possibilità di utilizzare molteplici nodi Client che concorrentemente scambiano informazioni con il Server e reattivamente ottengono gli aggiornamenti di stato per la particolare applicazione in esecuzione, richiede un’analisi approfondita delle dinamiche interne all’infrastruttura ma non è una sfida impossibile da affrontare. Si pensa che il lavoro svolto sia un importante punto di partenza per poter in futuro costruire qualcosa di stabile e utile anche per lo sviluppo di applicazioni che possono trovare impiego in casi reali, soprattutto per quanto riguarda la tecnologia BLE: al momento per la localizzazione dell’utente sulla base dei beacons rilevati si utilizza un algoritmo molto semplice ed efficace, tuttavia si potrebbe di certo affinarlo per renderlo ancora più accurato, anche in base alle esigenze applicative.

Un altro possibile sviluppo futuro sul lavoro svolto è la definizione di un linguaggio Domain Specific (con relative API) che si ispiri al dominio dello Spatial Computing (ad esempio Proto-like), per poter riprogrammare in maniera semplificata l’infrastruttura sviluppata. Inoltre, si potrebbe definire un nuovo IDE con possibile integrazione alla piattaforma di sviluppo utilizzata (ovvero Eclipse), sotto forma di plug-in: in questo modo si faciliterebbe il lavoro di futuri programmatore per la creazione di nuove applicazioni dell’infrastruttura, e anche modifiche alla sua struttura e alle sue funzionalità. Un primo esempio può essere l’integrazione del simulatore Java creato (dopo averlo perfezionato

rispetto allo stato attuale) dentro una libreria che può essere parte di un plug-in Eclipse.

Per quanto riguarda i beacons, essi sono ancora nella loro fase “primordiale” e quindi non è sbagliato pensare che nell’immediato futuro questi possano avvalersi anche di altri componenti hardware in modo da ampliare le loro possibilità di utilizzo. Alcune aziende stanno già lavorando sul potenziamento dei beacons, andando ad aggiungere nuove funzionalità date dall’utilizzo delle più svariate tipologie di sensori: accesso Wi-Fi, temperatura, accelerometro, umidità, luce, prossimità, infrarossi, rumorosità, remote-control, ecc. Il caso più interessante è il progetto open-source “WunderBar” di un’azienda tedesca che si occupa di piattaforme per l’estensione di Internet al mondo degli oggetti e dei luoghi concreti (il famoso “Internet of Things”), e che si chiama Relayr²⁰. Questa WunderBar (mostrata nella figura 7.1) è un insieme di 6 beacons BLE che possono essere collegati insieme o staccati per utilizzarli dove si vuole, e che possono essere monitorati e controllati dal proprio smartphone o da applicazioni web.



Figura 7.1: La WunderBar di Relayr.

Questi beacons sono dotati ognuno di diversi sensori:

- accelerometro e giroscopio;
- sensore di temperatura e umidità;
- sensore di rumorosità;

²⁰ Sito ufficiale di Relayr: <http://relayr.io/>

- sensore di luce, colore e prossimità;
- sensore ad infrarossi per il remote-control;
- un sensore a scelta richiesto dall’utente.

Oltre a questi, è presente un modulo “Master” dotato di scheda Wi-Fi e chip BLE a cui ogni beacons comunica i propri dati, come mostrato nella figura 7.2. Infatti il loro uso principale non è quello “classico” visto nella tesi e che permette di fare advertising dei dati ai fini di una localizzazione, ma è soprattutto quello di inviare i propri dati rilevati a tale modulo Master, in modo che quest’ultimo trasmetta tali informazioni sul cloud tramite rete Wi-Fi.

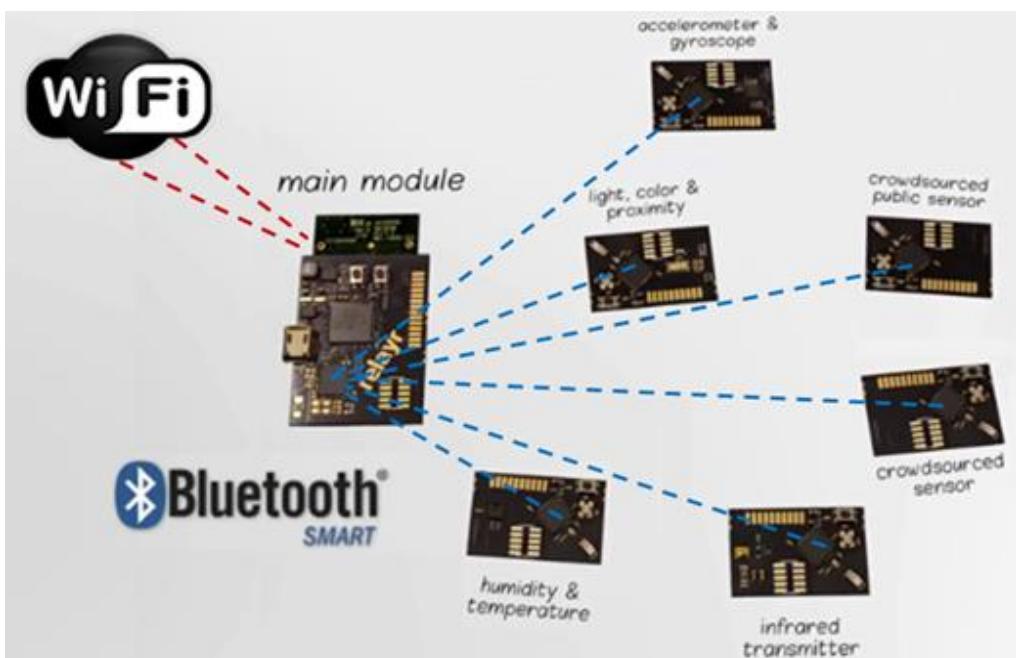


Figura 7.2: Come interagiscono i vari moduli della WunderBar.

La WunderBar è distribuita con l’SDK open-source fornito dall’azienda stessa e compatibile con iOS o Android, in modo che gli sviluppatori possano facilmente interagire con essa per creare le più svariate applicazioni.

Quanto presentato in questa tesi vuole mettere in evidenza il passo in avanti che avverrà nell’immediato futuro, una “rivoluzione” tecnologica verso una nuova forma di interazione “smart” con tutto ciò che ci circonda e caratterizza la nostra vita quotidiana. Il mondo in cui viviamo sarà fortemente influenzato da queste nuove tecnologie.

Elenco delle figure

Figura 1.1: Confronto sulla precisione delle principali tecnologie location-based	9
Figura 1.2: Esempio di localizzazione via A-GPS.....	10
Figura 1.3: A sinistra un tag NFC, a destra un esempio di lettura di tag RFID passivi.	11
Figura 1.4: Compatibilità tra le diverse tipologie di Bluetooth.	15
Figura 1.5: Composizione di un beacon Estimote.	17
Figura 1.6: Dimensioni di alcuni beacons delle principali aziende.	17
Figura 1.7: A sinistra una scheda Arduino BLE, e a destra un beacon fai-da-te costruito con scheda Raspberry Pi.	18
Figura 1.8: Come si presenta l'applicazione utilizzata per simulare un beacon su OSX Mavericks.	19
Figura 1.9: Esempi di dongle BLE USB.....	19
Figura 2.1: Esempio di utilizzo applicativo dei beacons.	22
Figura 2.2: Esempio che mostra la posizione in cui l'utente riceve le informazioni di benvenuto al Citi Field Stadium di New York.....	23
Figura 2.3: Esempio di promozione per nuovi visitatori visualizzata all'ingresso.....	23
Figura 2.4: Esempio di localizzazione del proprio posto a sedere.....	24
Figura 2.5: Esempio che mostra l'utilizzo dell'applicazione EduBeacons.....	24
Figura 2.6: Esempio che mostra alcune delle caratteristiche dell'applicazione sviluppata per il museo Rubens House.	25
Figura 3.1: A sinistra un esempio di amorphous medium, a destra una rete di dispositivi che lo approssimano.	27
Figura 3.2: Gap tra computazione a livello locale e quella globale colmato dallo Spatial Computing.	28
Figura 3.3: Progettazione della parte Server.....	34
Figura 3.4: Progettazione della parte Client.	39
Figura 3.5: Architettura di progetto relativa al primo prototipo dell'infrastruttura.	39
Figura 3.6: Un esempio di simulazione della applicazione Partition.....	41

Figura 4.1: Un esempio che mostra l'applicazione GPS sviluppata per la sperimentazione.	44
Figura 4.2: Codice relativo al riconoscimento dei beacons rilevati nell'ambiente.....	48
Figura 4.3: Materiale acquistato per sperimentare la tecnologia BLE.....	49
Figura 4.4: L'applicazione di configurazione per i beacons di Stick N Find. .	50
Figura 4.5: L'applicazione di configurazione per i beacons iBKS.	50
Figura 5.1: Codice relativo al calcolo della posizione attuale in base ai beacons rilevati.	59
Figura 5.2: Rappresentazione parte Server della nuova infrastruttura (in giallo le novità).	59
Figura 5.3: Rappresentazione delle novità introdotte nell'infrastruttura per la parte Client.....	61
Figura 5.4: Architettura di progetto della nuova infrastruttura.	62
Figura 6.1: A sinistra la schermata di selezione dell'applicazione FlagFinder, a destra ciò che viene visualizzato se si sceglie di essere una destinazione.	64
Figura 6.2: Esempi di orientazione freccia e colore sfondo per il Finder (a sinistra se si è a una distanza minore di 24.4 centimetri dalla destinazione, a destra se oltre gli 80 metri).	64
Figura 6.3: Posizionamento nell'edificio del tappeto NFC.	65
Figura 6.4: Posizionamento nell'edificio dei beacons.	66
Figura 6.5: Modifica delle posizioni dei beacons in base alla struttura dell'ambiente indoor.	67
Figura 6.6: Codice relativo all'impostazione delle nuove informazioni di stato per l'applicazione FlagFinder.	68
Figura 6.7: Codice relativo alla computazione che fa seguito ad ogni aggiornamento di stato.	69
Figura 6.8: Codice relativo agli aggiornamenti grafici sullo schermo del device.	70
Figura 6.9: Codice relativo alla scelta del colore di sfondo sulla base della distanza misurata dalla destinazione.	70
Figura 6.10: Esempio di scenario outdoor quando ci si trova a una distanza superiore a 80 metri dalla destinazione (colore sfondo rosa).	72

Figura 6.11: Esempio di scenario outdoor quando ci si trova a una distanza compresa tra i 60 e gli 80 metri dalla destinazione (colore sfondo magenta)..	72
Figura 6.12: Esempio di scenario outdoor quando ci si trova a una distanza compresa tra i 40 e i 60 metri dalla destinazione (colore sfondo viola).	73
Figura 6.13: Esempio di scenario outdoor quando ci si trova a una distanza compresa tra i 20 e i 40 metri dalla destinazione (colore sfondo blu).	73
Figura 6.14: Esempio di scenario indoor quando si è in prossimità del beacon 6 (freccia verso le scale e sfondo blu).....	74
Figura 6.15: Esempio di scenario indoor quando si è in prossimità del beacon 5 (freccia lungo il corridoio e sfondo blu).	74
Figura 6.16: Esempio di scenario indoor quando si è in prossimità del beacon 4 (freccia che indica la svolta al prossimo corridoio e sfondo blu).	75
Figura 6.17: Esempio di scenario indoor quando si è in prossimità del beacon 3 (freccia lungo il corridoio e sfondo blu).	75
Figura 6.18: Esempio di scenario indoor quando si è in prossimità del beacon 2 (freccia che indica la svolta al prossimo corridoio e sfondo blu chiaro).	76
Figura 6.19: Esempio di scenario indoor quando si è lungo il corridoio che collega il beacon 2 con il beacon 1 (sfondo azzurro chiaro).....	76
Figura 6.20: Esempio di scenario indoor quando si è in prossimità del beacon 1 (stanza contenente la destinazione).....	77
Figura 6.21: Esempio di scenario in cui ci si trova sul tappetino NFC e a una distanza compresa tra 97.6 e 122 centimetri dalla destinazione (sfondo verde scuro).....	77
Figura 6.22: Esempio di scenario in cui ci si trova sul tappetino NFC e a una distanza compresa tra 73.2 e 97.6 centimetri dalla destinazione (sfondo verde chiaro).	78
Figura 6.23: Esempio di scenario in cui ci si trova sul tappetino NFC e a una distanza compresa tra 48.8 e 73.2 centimetri dalla destinazione (sfondo giallo).	
.....	78
Figura 6.24: Esempio di scenario in cui ci si trova sul tappetino NFC e a una distanza compresa tra 24.4 e 48.8 centimetri dalla destinazione (sfondo arancione).....	79

Figura 6.25: Esempio di scenario in cui ci si trova sul tappetino NFC e a una distanza inferiore a 24.4 centimetri dalla destinazione (sfondo rosso)	79
Figura 7.1: La WunderBar di Relayr	83
Figura 7.2: Come interagiscono i vari moduli della WunderBar.....	84

Bibliografia e sitografia

Robinson, M. (s.d.). <https://github.com/mtrrb/BeaconOSX>.

Bluetooth Special Interest Group, "Mobile Telephony Market", January 16, 2014.

J. Beal, R. Schantz, "A Spatial Computing Approach to Distributed Algorithms", 45th Asilomar Conference on Signals, Systems, and Computers, November 2010.

M. Viroli, J. Beal, K. Usbeck, "Operational semantics of Proto", Science of Computer Programming, 2012.

J. Beal, S. Dulman, K. Usbeck, M. Viroli, N. Correll, "Organizing the Aggregate: Languages for Spatial Computing", April 2nd, 2012.

Erik Vlugt, "Bluetooth Low Energy, Beacons and Retail", October 23, 2013.

Documentazione Bluetooth, <http://www.bluetooth.com/Pages/Bluetooth-Smart.aspx>.

Articolo su tag NFC, <http://tagnfc.net/it/>

Wikipedia, <http://it.wikipedia.org/>.

Sito ufficiale Estimote, <http://estimote.com>.

Blog ufficiale Estimote, <http://blog.estimote.com/>.

Sito ufficiale Stick N Find, <https://www.sticknfind.com>.

Sito ufficiale Accent Systems, <http://ibeacon.accent-systems.com>.

Guida online per impostare OSX Mavericks come beacon simulato, <http://www.blendedcocoa.com/blog>.

Guida alle varie tipologie di Beacon, <http://beekn.net/guide-to-ibeacons/>, Dicembre 2013.

Guide ufficiali per applicazioni Android, <http://developer.android.com>.

Articolo sul caso MBL Citi Field, <http://techcrunch.com/2013/09/29/mlbs-ibeacon-experiment-may-signal-a-whole-new-ball-game-for-location-tracking/>.

Sito ufficiale di Paul Hamilton (caso EduBeacons), <http://www.appsbypaulhamilton.com>.

Sito ufficiale per ottenere l'Android Beacon Library, <http://developer.radiusnetworks.com/ibeacon/android/>.

Sito ufficiale di Relayr, <http://relayr.io>.

Ringraziamenti

Desidero ringraziare innanzitutto il professor Viroli per avermi dato l'opportunità di lavorare su ciò che a me piace di più, ovvero sperimentare le più moderne tecnologie in ambito smart. Inoltre, ringrazio Davide per aver condiviso con me i momenti più difficili di questo lavoro e per il prezioso aiuto in fase di sviluppo e testing. Vorrei infine ringraziare Maria Giorgia per i consigli e il sostegno morale in questi mesi di duro lavoro, e i miei genitori per avermi supportato e sopportato in tutti questi anni di università.