ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

PROGETTAZIONE E SVILUPPO DI UN SERVIZIO DI LOCALIZZAZIONE IN AMBITO OSPEDALIERO BASATO SU INFRASTRUTTURA BEACON

Elaborato in
SISTEMI EMBEDDED E IOT

Relatore
Prof. ALESSANDRO RICCI

Co-relatore
Ing. ANGELO CROATTI

Presentata da
GABRIELE GUERRINI

Seconda Sessione di Laurea
Anno Accademico 2017 – 2018

PAROLE CHIAVE

Localizzazione in ambito ospedaliero

TraumaTracker

LocationService

Sistemi embedded

IoT

Indice

Introduzione

Al giorno d'oggi e nei più svariati settori, i sistemi di localizzazione hanno cominciato una rapida diffusione dovuta per la maggiore dal fatto che, in un'infrastruttura complessa, l'organizzazione rappresenta un punto fondamentale per lo svolgimento del lavoro in modo efficace. Questo è stato possibile grazie ad una diffusione pervasiva di tecnologie embedded, creando la possibilità di avere sistemi che riescono ad interagire direttamente con l'ambiente fisico circostante.
Nel nostro caso il dominio considerato si limiterà esclusivamente a quello del settore ospedaliero ma, in ogni caso, il discorso può essere esteso facilmente effettuando considerazioni analoghe.
Si inizierà dunque con il parlare dei sistemi di localizzazione in ambito ospedaliero allo stato dell'arte, illustrandone caratteristiche e benefici, per poi proseguire nello sviluppo di un nuovo sistema di localizzazione basato su infrastruttura beacon. Infine si passerà ad un'integrazione di quest'ultimo nel progetto complessivo "TraumaTracker" e ad una sua validazione mediante valutazioni in merito alle sue prestazioni; questa parte di lavoro si suddividerà in due momenti successivi, effettuando un'analisi teorica a priori in modo sistematico e puntuale e, solo successivamente, una seconda valutazione sul campo in modo più qualitativo, utile ai fini del "deployment" del sistema.

Capitolo 1

Localizzazione in ambito ospedaliero

1.1 Sistemi di localizzazione in ambito ospedaliero

L'ambito ospedaliero rappresenta forse uno dei domini di applicazione per eccellenza quando si parla di sistemi di localizzazione, visti i notevoli scenari di applicazione. L'importanza cresce in modo esponenziale dal momento in cui si considera che entra in gioco anche la salute stessa dei pazienti e dunque supervisione e tempestività risultano di primaria importanza. Strettamente considerando i possibili casi di applicazione, i principali benefici rilevabili risultano essere i seguenti:

- **Snapshot:** la possibilità di applicare un sistema di tracciamento sia a persone che macchine fa in modo che sia possibile ottenere, se attuato a dovere, uno snapshot della situazione corrente per valutare disponibilità di medici, macchinari, utensili e stanze.

- **Infermieristica agevolata:** collegandosi al precedente punto, è possibile contattare direttamente l'operatore di interesse più prossimo al paziente bisognoso; aumento celerità e diminuzione tragitti inutili.

- **Sicurezza dei pazienti:** ci si cali nello senario di avere pazienti affetti da sindromi croniche che ne limitano la lucidità mentale (Alzheimer ecc.); in genere si necessiterebbe una continua stretta sorveglianza. Se si tenesse traccia della posizione di ciascuno, si potrebbe sviluppare un primo sistema di sorveglianza per evitare che escano dalle zone loro dedicate (stanza, reparto ...).

- **Analytics:** fruibile per fini statistici alla volta di monitorare qual si voglia informazione in particolare (tempistiche, medie giornaliere, picchi di arrivi, medici più impegnati ecc.) e, di conseguenza, agire prendendo i giusti accorgimenti per agevolare il lavoro.

Per quanto banali, queste motivazioni di esempio mostrano come una soluzione del genere può entrare in gioco in modo ricorrente, specialmente in infrastrutture di grandi dimensioni.

A dimostrazione del grande interesse del giorno d'oggi, la "Research and Markets", importante compagnia di ricerca di mercato, ha svolto uno studio in merito denominato *"Global Location-based Services Market in the Healthcare Industry 2016-2020"* e, secondo i dati pubblicati, si prevede che il mercato LBS (Location-Based Services) per l'healtcare crescerà con un CAGR [1] del 32.46% nel quadriennio 2016-2020[2].

Allo stato delle cose, nonostante in generale ci sia in alcuni casi ancora un alone di avversione all'introduzione di tecnologie all'avanguardia, sono stati sviluppati diversi sistemi del tipo a testimoniare un settore florido ed in rapida espansione. Si accenneranno ora diversi esempi progettati allo scopo e già in esecuzione (o comunque in fase di sviluppo) e riferimenti a compagnie che si occupano di sviluppo di tecnologie inerenti al settore.

[1] Compound Annual Grow Rate (tasso annuale di crescita composto)[2]. Si tratta di un indice che rappresenta il tasso di crescita di un certo valore in un arco di tempo; la cosa importante da comprendere è che non si tratta di un vero tasso di crescita rispecchiabile nella realtà dei fatti, ma di un numero astratto che stima come un investimento/bene sarebbe cresciuto se fosse stato sottoposto ad una crescita a tasso costante.

[2] Informazioni reperite nell'articolo [1].

Progetto I-Locate

Il progetto "I-Locate" [3], coordinato a livello europeo da "Trilogis" (azienda informatica), ha come obbiettivo quello di creare un'applicazione che prevede più casi d'uso:

- Guida del paziente all'interno della struttura, similmente a quanto avviene all'aperto per mezzo di un navigatore, di "googleMaps" e via dicendo; il paziente di avvarrà di apposita applicazione su telefono per raggiungere la meta desiderata mentre gli addetti in ospedale saranno nel contempo allertati del prossimo arrivo.
- Localizzazione di macchinari in modo da renderli reperibili e fruibili il più velocemente possibile.
- Spostamenti successivi di pazienti costretti in carrozzina a causa di disabilità motoria (permanente o provvisoria che sia) che devono eseguire trattamenti mediante utilizzo di apparecchiature fisse; un'equipe specializzata si occuperà degli spostamenti andata e ritorno.

La sperimentazione ha avuto la durata di 12 mesi (conclusasi nel 2016) nelle sedi ospedaliere "S. Maria" di Rovereto, Utrecht in Olanda, "MITERA" ad Atene, "St James" a Malta e "Alba Iulia" in Romania. Come si può notare, il progetto ha coinvolto molte realtà europee differenti grazie al pieno supporto da parte della UE, la quale crede ampiamente in questi tipi di iniziative e nella loro diffusione negli anni a venire.

Tap my life

"Tap my life"[4] è un progetto sviluppato da un'azienda informatica italiana con sede a Bergamo e specializzata in ambito sanitario. L'applicativo consente di ottenere un meccanismo di tracciamento tramite un'opportuna estensione del ben noto GPS (si ricordi infatti che mentre in ambienti aperti il GPS funziona in modo ottimo, in ambienti "in-door" le cose si complicano causa imprecisione nel rilevamento della posizione).

Le funzionalità che mette a disposizione sono molteplici e includono accoglienza e guida degli utenti, l'analisi dei tempi e delle performance di vari processi (blocco operatorio, sterilizzazione ...), rilevazione dei tempi intercorsi tra interventi successivi, localizzazione di strumentazioni e gestione di trasferimenti di pazienti. Inoltre, tratta anche aspetti più specialistici quali il

monitoraggio del trasporto di emocomponenti, campioni biologici ed organi. Infine, viene annessa una piattaforma di business-intelligence al fine di raccolta statistica dei dati e ottimizzazione dei processi di lavoro.

L'infrastruttura di basso livello si basa sull'utilizzo si di smartphone ma anche di dispositivi "wearable" che stanno prendendo sempre più piede in ottica "IoT".

Una statistica significativa a supporto dell'utilità di questi sistemi è data dal rientro economico previsto in 18 mesi con la sola riduzione del tempo impiegato dal personale sanitario per il rintracciamento di dispositivi medici all'interno della struttura[3].

Altri

Diverse ulteriori aziende sul mercato offrono servizi analoghi, tra le quali citiamo "Eximia"[5] e "IDEST"[6]. Si basano principalmente sull'utilizzo di tecnologie RFId in modo più o meno articolato, dai classici tag RFId con apposito lettore a sistemi "spider" per il controlo real-time su ambienti affollati. Risulta interessante considerare anche la "Kontakt.io"[7] poiché, oltre a disporre di dispositivi RFId, fa utilizzo anche, contrariamente ai precedenti, di dispositivi beacon; l'azienda mette a disposizione anche sistemi gestionali (mobile e non) da annettere al sistema.

[3]Fonte ufficiale "Tap my life", provvista di certificato di eccellenza, "Seal of Excellence", conferito da parte della commissione europea.

1.2 Progetto "TraumaTracker"

Il progetto "TraumaTracker" nasce all'interno della collaborazione tra l'università di Bologna e l'ospedale "Bufalini" di Cesena al fine di impiantare all'interno di quest'ultimo sistemi all'avanguardia nel campo dell'informatica. L'obiettivo principale prevede la supervisione passo passo del trauma riportato del paziente in modo da facilitare l'operato dell'équipe di lavoro. Questo può avvenire, ad esempio, dando la possibilità di tener aggiornato lo stato del trattamento in tempo reale tra le varie fasi di lavoro (dosaggi, tempistiche, parametri vitali ecc.).
Tra le funzionalità richieste risulta inoltre importante monitorare lo sviluppo del trauma, dall'arrivo in ospedale del paziente fino alla conclusione del trattamento, e, a questo fine, si rende dunque necessario un micro-servizio per seguire l'evoluzione del trattamento, articolato su varie fasi e in più stanze nell'ala dell'ospedale interessata. Il "LocationService" si occuperà dunque di adempiere a questo compito.

Capitolo 2

LocationService

2.1 Analisi dei requisiti

L'obiettivo è quello di realizzare un micro-servizio che permetta di effettuare localizzazione al fine di tener traccia degli spostamenti all'interno del settore coperto dal servizio, i.e. ala pronto soccorso dell'ospedale "Bufalini" di Cesena.

Il processo di trattamento sanitario si articola in diverse fasi e stanze. Il tracciamento degli spostamenti deve iniziare all'arrivo del paziente in ospedale e deve durare ininterrottamente fino a fine trattamento, ovvero all'arrivo nelle stanze dedicate di terapia intensiva. Il medico (o chi per lui) dispone di un tablet da cui deve poter dare inizio al processo di cure con conseguente inizio della localizzazione e deve essere notificato ogni qual volta ci sia un cambio di stanza. Alla fine, sempre tramite tablet, conclude il caso clinico associato al trattamento effettuato e di conseguenza anche la localizzazione cessa la sua attività.

Analizzando il problema si deduce che non sia necessario fare in modo di rilevare la posizione precisa al metro ma è sufficiente un servizio di localizzazione ad alto livello. Fatta questa assunzione, si è deciso di basare l'infrastruttura su due concetti fondamentali:

- **Tag:** identificatore necessario per determinare in modo univoco il trauma.
- **Gateway:** entità che raccoglie informazioni sui tag.

Inoltre, a supporto, è stato introdotto un terzo concetto, i.e *regione*, per aumentare ulteriormente il livello astrazione e realizzare il sistema nel modo migliore.

Regioni

Il concetto di regione è stato introdotto per fare in modo in primis, di stabilire a piacere un'area massima entro cui ciascun gateway debba rilevare i tag presenti e, inoltre, per poter associare una stessa regione a più gateway.

La gestione delle regioni è a carico esclusivamente dell'applicazione "LocationService" che la gestisce in modo autonomo, senza coinvolgere altri dispositivi del sistema. In questo modo si dissocia dal dispositivo fisico in sé (gateway) il concetto di regione, permettendo di associarne molteplici ad una stessa. Questo permette di modellare nel modo migliore il mondo fisico sottostante; un esempio è riportato nell'immagine 2.1.

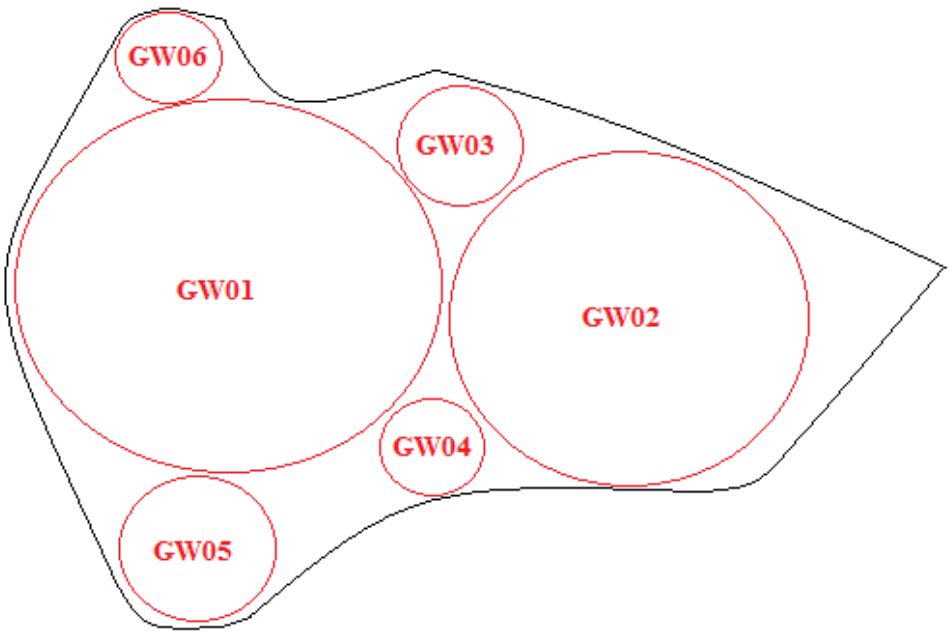

Figura 2.1: *Esempio di regione in cui operano più gateway in modo da mappare nel modo migliore l'area di interesse.*

A basso livello, questo non consiste altro che nel considerare la potenza dei segnali ricevuti e stabilire se essi siano o meno sopra una certa soglia; questo è fattibile poiché ovviamente tra potenza di segnale ricevuta e distanza dal tag vige una proporzionalità inversa.

Definito il processo da seguire e i concetti da utilizzare, si può definire ora l'algoritmo operativo che viene eseguito.
All'arrivo di un paziente in ospedale, il medico curante prende possesso di un tag libero e, tramite tablet, lo assegna al paziente in modo da poter definire trauma e trattamento in modo univoco. In ogni stanza di interesse è posizionato un gateway che rileva i tag presenti in sua prossimità. Il "LocationService" provvede a reperire le informazioni gateway per gateway in modo da integrarle ed elaborarle. Se viene rilevato un cambio di posizione in uno dei tag attualmente attivi, una notifica verrà inviata. A fine trattamento, sarà sufficiente indicare tramite tablet che il processo si è concluso e il tag sarà reso nuovamente libero e a disposizione.
Una strutturazione di questo tipo comporta i seguenti vantaggi:

- **Manutenzione facilitata:** i gateway sono posizionati in modo stabile e dunque è possibile collegarli direttamente all'impianto elettrico, fornendo potenzialmente alimentazione a tempo indeterminato; inoltre i tag, a contatto diretto con gli utilizzatori, permettono una sostituzione molto semplice delle batterie da parte del personale addetto.

- **Connessioni più stabili:** i gateway, oltre ad essere alimentati continuamente, possono essere connessi tramite cavo ethernet alla rete internet ospedaliera; una connessione cablata ovviamente risulta in condizioni normali più veloce e stabile rispetto ad una wireless.

- **Ingombro minimo:** i tag sono da intendersi come oggetti di dimensione veramente ridotta, equiparabile per intendere ad un portachiavi; dunque un suo utilizzo non crea ingombri in quanto può essere agganciato alla barella o al collo tramite l'utilizzo di un opportuno laccio.

Dunque, tirando le somme, il servizio "LocationService", intermediario tra il front-end (tablet, dashboard browser ecc.) e il back-end (sistemi embedded vari), si deve occupare delle seguenti funzioni:

- Rilevazione di informazioni aggiornate sui tag.

- Salvataggio di informazioni in modo persistente su database.

- Esposizione di un'interfaccia API verso l'utente al fine di permettere la registrazione ad un determinato tag e la ricezione di notifiche.

- Interfacciamento per la gestione di tag e gateway da remoto (classiche operazioni di inserimento, cancellazione, ecc. che implicano tuttavia lo sviluppo di meccanismi necessari per mantenere la consistenza dei dati e per la gestione di chiamate REST).

2.2 Localizzazione mediante tecnologie BlueTooth

Sin dalla sua introduzione, lo standard "Bluetooth 4.0" è stato considerato come la tecnologia più adatta per supportare infrastrutture in ottica "Internet of Things" (IoT). Il protocollo Bluetooth 4.0 prevede infatti una nuova modalità di funzionamento a basso consumo e questa differenza risulta essere sostanziale nel momento in cui si vanno a considerare dispositivi esclusivamente alimentati a batteria, aumentandone la durata di funzionamento.
Partendo da questo presupposto, nel 2014 è stata resa disponibile una nuova tecnologia denominata "iBeacon Bluetooth Low Energy" (BLE) firmata Apple. A seguito di questa, altre tecnologie analoghe sono state sviluppate seguendo la scia "iBeacon" nel giro di alcuni mesi, tra cui troviamo su tutte "Eddystone" di Google.
I beacon BLE in questione sono piccoli dispositivi hardware a basso costo che possono trasmettere svariate informazioni sfruttando connessioni "Bluetooth 4.0". L'uso di connessione bluetooth fa intendere però come la portata di utilizzo sia limitata nell'ordine di qualche metro.
L'utilizzo di questi nuovi beacon BLE è diventato sempre più pervasivo poiché impiegabili in modo agevole e in molteplici scenari. Un tipico utilizzo di beacon BLE si ha in infrastrutture adibite alla localizzazione poiché, tra le varie informazioni trasmesse dai beacon, troviamo anche un identificativo univoco che risulta essere estremamente pratico; inoltre si riescono ad evitare i classici problemi riscontrabili con un sistema basato su GPS in scenari di localizzazione indoor.
Infine, come mostrato in figura 2.2, aderire allo standard "Bluetooth 4.0", permette di creare un sistema aperto in cui il dispositivo situato all'altro estremo della comunicazione non risulta strettamente vincolato.

Figura 2.2: *Si mostra un'interazione tra un beacon su tecnologia "Eddystone" e diversi dispositivi eterogenei; questo dimostra come sia possibile sviluppare un sistema aperto ed integrare svariate componenti.*[1]

Le specifiche del progetto vincolano all'utilizzo di un'infrastruttura basata su tecnologie beacon BLE. Sul mercato sono oramai presenti numerose aziende che mettono a disposizione questi dispositivi ma la preferenza è ricaduta nell'utilizzo di tecnologie di marchio "BlueUp".

[1] Immagine presa da `http://g03.s.alicdn.com/kf/HTB1yTuNJpXXXXaiXVXXq6xXFXXXW/220518698/HTB1yTuNJpXXXXaiXVXXq6xXFXXXW.jpg`

2.3 Tecnologie BlueUp

La scelta della "BlueUp" è stata guidata dalla possibilità di poter creare un'infrastruttura complessa e personalizzabile poiché i dispositivi forniti risultano essere "aperti" e l'interfacciamento avviene tramite un'API esposta pubblicamente. Si passerà ora ad una descrizione di queste al fine di capirne funzionamento e caratteristiche.
Scendendo ad una vista di basso livello, i tag e gateway non sono altro che sistemi embedded, rispettivamente:

- **Tag:** sono dei beacon, degli "emettitori" di segnali che possono lanciare per l'appunto un segnale contenente informazioni varie; il segnale ovviamente si diffonde entro una certa distanza massima.

- **Gateway:** sono dei "ricevitori" che possono catturare i segnali emessi dai tag e leggere le informazioni ricevute di volta in volta; anche qui c'è un limite fisico massimo alla distanza entro la quale i gateway possono rilevare i messaggi inviati dai tag.

Questi tipi di dispositivi sono molto comuni nel mondo embedded e dunque sul mercato numerose compagnie se ne occupano.

2.3.1 Beacon BlueUp

I beacon "BlueUp", come del resto anche tutti gli altri tipi di dispositivi beacon, utilizzano un segnale bluetooth per emettere i propri messaggi e comunicare con i gateway. Le informazioni trasmesse sono veramente molte e includono uuid, numero seriale, RSSI, major e minor number, batteria residua, timestamp invio messaggio ecc. Inoltre, la "BlueUp" permette anche l'invio di frame personalizzati, mettendo a disposizione più slot per l'invio messaggi e differenti tecnologie di comunicazione, "IBeacon" e "Eddystone" su tutte ma anche "Quuppa" e altre. Nel nostro caso si sono mantenute le impostazioni di default per l'invio di messaggi e "IBeacon" come protocollo di comunicazione.

"BlueUp" mette a disposizione un'applicazione per smartphone che permette di collegarsi al beacon, sempre utilizzando bluetooth, e di configurarlo a piacimento; tra i parametri configurabili troviamo uuid, password di accesso, potenza del segnale emesso e vari meccanismi per il risparmio energetico.

Figura 2.3: *Esempio di beacon "BlueUp".*[2]

2.3.2 Gateway BlueUp

Ogni gateway "BlueUp" è semplicemente un dispositivo "RaspberryPi3" a cui è stata aggiunta una serie di software interni per permettere la scansione periodica dei beacon in propria vicinanza ed esporre un'interfaccia verso l'esterno in modo tale di poter recuperare le informazioni che il gateway ha raccolto; nel nostro caso sarà l'applicazione "LocationService" ad interagire con quest'ultimi. Infine sono dotati di un involucro rigido in plastica bianca per fini estetici ma anche di protezione.

Si può interagire facilmente con il dispositivo per effettuare una configurazione manuale utilizzando una dashboard browser, mostrata in figura 2.4,

[2]Immagine presa da `https://www.blueupbeacons.com/images/shopimg/BlueBeaconGateway2.jpg`

facilmente raggiungibile una volta scoperto almeno un indirizzo IP assegnato al gateway. Questa funzionalità risulta di fondamentale importanza per diverse ragioni, tra cui:

- **Connettività:** permette la modifica di alcuni parametri base come l'attivazione dell'interfaccia internet wireless, l'assegnamento di un IP statico, di un server Proxy ecc.
- **Monitoraggio:** permette di vedere real-time i vari beacon rilevati ("BlueUp" e non).
- **Sicurezza:** possibilità di impostare una password per l'accesso alla dashboard.
- **Testing:** si possono effettuare facilmente test empirici per verificare graficamente la risposta in potenza di segnale dei beacon in vicinanza al gateway.
- **Cloud:** si può far in modo di effettuare periodicamente un backup dei dati su cloud.
- **Webhook:** possibile impostare un approccio "push" per il recupero delle informazioni dal gateway; dispone tuttavia di alcuni limiti nell'invio delle informazioni poiché non assicura a priori il numero di messaggi inviati.

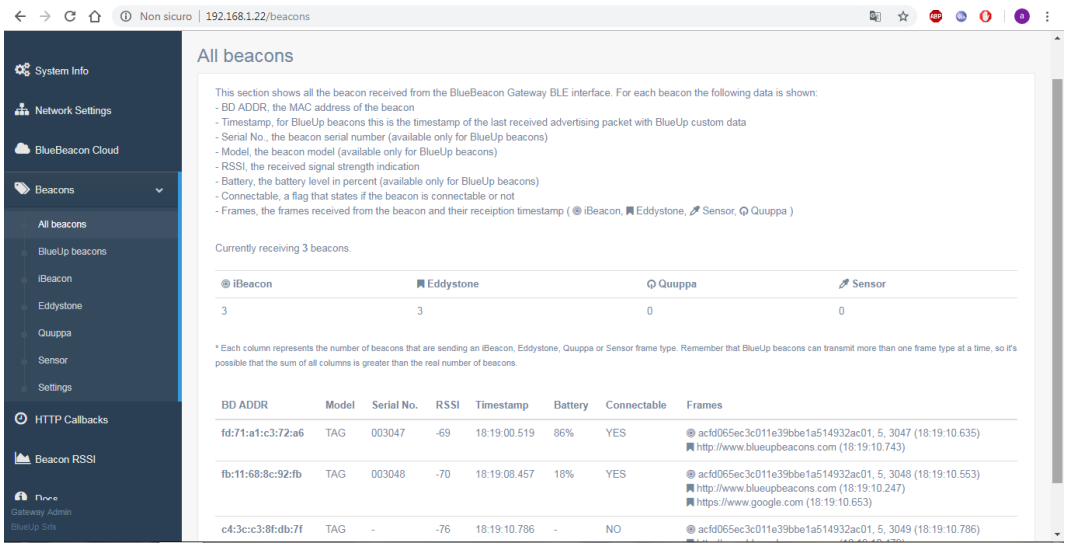

Figura 2.4: *Esempio di dashboard fornita dalla "BlueUp" mediante la quale è possibile interagire con un gateway; nell'immagine è mostrata una schermata di visualizzazione dei tag rilevati dal dispositivo.*

Figura 2.5: *Esempio di gateway "BlueUp" modello "D".*[3]

[3]Immagine presa da `https://www.blueupbeacons.com/images/shopimg/BlueBeaconGateway2.jpg`

2.4 Progettazione

Vista l'elevata complessità, l'applicazione è stata sviluppata secondo un paradigma ad attori, ciascuno operante su un modello a event-loop. L'architettura principale dell'applicazione è mostrata in figura 2.6.

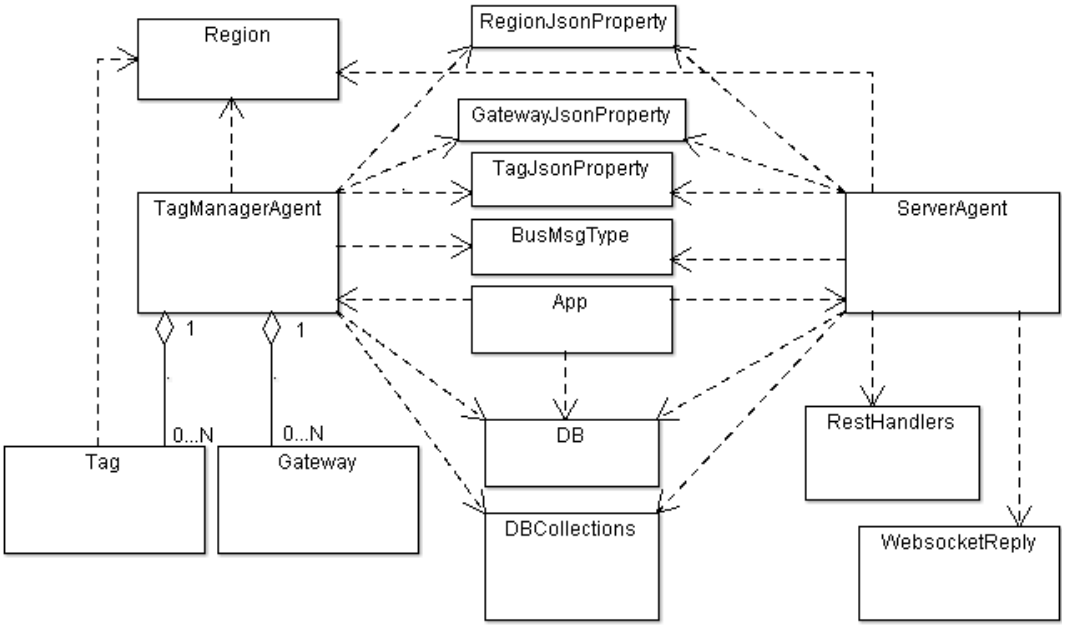

Figura 2.6: *Diagramma UML delle classi raffigurante l'architettura dell'applicazione*

Le classi principali e colonne portanti dell'applicazione sono "TagManagerAgent" e "ServerAgent", ovvero i due agenti in esecuzione; il primo si occupa del reperimento delle informazioni sui tag, mentre il secondo è il responsabile per l'interfacciamento verso l'utente e tutto ciò che ne comporta (web-socket per le notifiche e chiamate REST su tutte). Molte sono le classi di utilità condivise, utilizzate da entrambi gli agenti.

Per adempiere in modo congruo alla filosofia del pattern ad attori, gli agenti comunicano esclusivamente tramite scambio di messaggi su un bus di comunicazione messo a disposizione dalla libreria "Vert.x" e utilizzando un modello publish/subscribe interno all'applicazione. Una descrizione dettagliata dei due agenti nella sezione 2.5.

Nella struttura mostrata in immagine 2.6 si fa riferimento ad una classe "RestHandlers". Questo per non complicare oltremodo la raffigurazione già non banale ma, nel dettaglio, nasconde una complessa gerarchia e articolazione

di “handlers” a sua volta; l’immagine 2.7 sviluppa suddetta classe e mostra ciò in modo esaustivo.

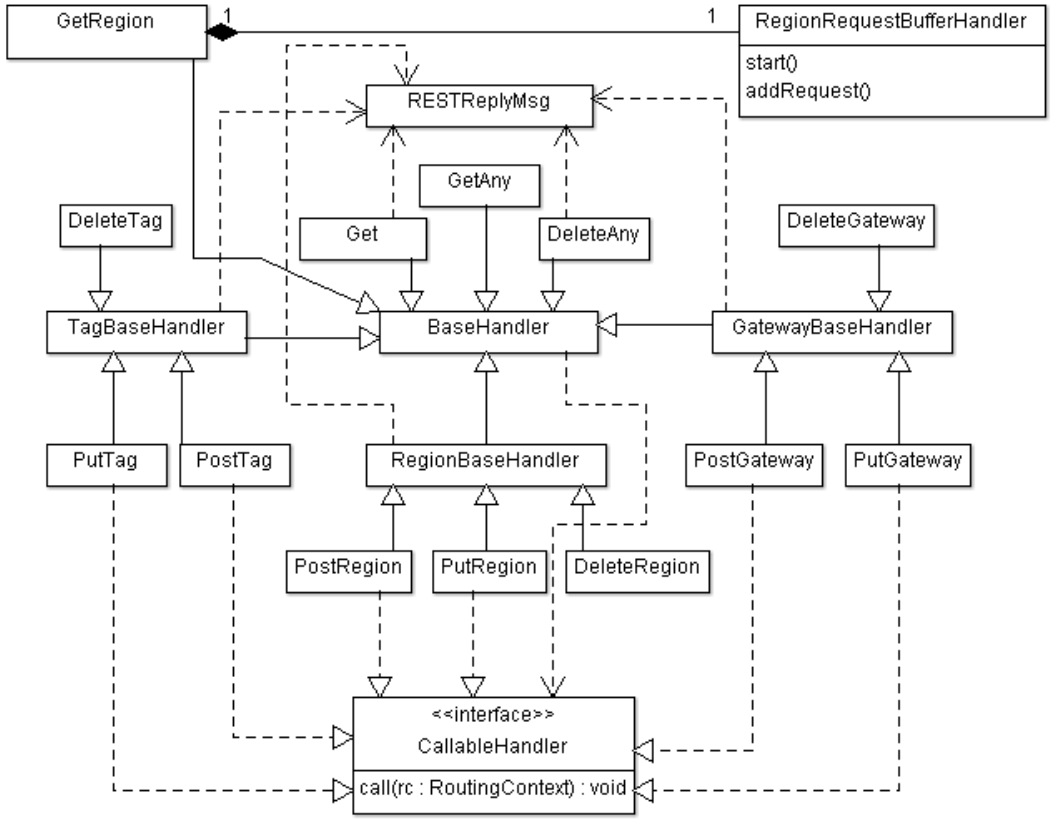

Figura 2.7: *Diagramma UML delle classi raffigurante l’architettura gerarchica degli “handler” usati per assolvere alle richieste REST in arrivo all’applicazione.*

L’agente “ServerAgent”, tramite un sistema di routing interno, riesce a selezionare di volta in volta l’handler corretto da utilizzare per la risoluzione della richiesta. La gerarchia si sviluppa su tre livelli in base al grado di astrazione che si considera, variando secondo tipo di richiesta (get, post ecc.) e tipo di soggetto della richiesta (tag, gateway, regione). Vi è innanzi tutto un handler base (non instanziabile) per ogni tipo di richiesta, degli handler specifici per soggetto (anch’essi non instanziabili) e degli handler specifici per soggetto e tipo di richiesta.
Al fine di gestire alcune situazioni circostanziali che si potrebbero verificare e portare ad errori, diversi handler implementano l’interfaccia “CallableHandler”.
Un discorso a parte va fatto per le chiamate “GET” sulle regioni poiché è

l'unico caso in cui l'handler deve usare il bus anche per la ricezione di messaggi e dunque si avvale di una classe usiliaria ("RegionRequestBufferHandler") internamente operante su event-loop e provvista di un buffer per lo "storage" delle richieste in sospeso.

2.5 Sviluppo

Lo sviluppo è avvenuto in linguaggio Java e con il supporto della libreria "Vert.x", cosa che ha permesso la gestione in modo semplice di diversi aspetti (approccio ad attori, websocket, interfacciamento al DB ...).
La scelta del DB è ricaduta su "MongoDB" (DB non relazionale).

2.5.1 TagManagerAgent

La classe "TagManagerAgent" istanzia uno dei due agenti dell'applicazione e, internamente, si basa su un funzionamento ad "event-loop". I compiti assegnategli sono due:

- Recuperare informazioni sui tag da ogni gateway presente nel sistema.
- Valutare i cambi di posizione di ogni tag in modo più tempestivo possibile e, nel caso, notificare tramite bus la nuova locazione del tag.

Un problema fondamentale consiste nelle locazioni che il tag può assumere poiché, oltre a trovarsi in una determinata stanza provvista di un gateway, può spesso essere in zone non coperte (corridoio, magazzino, stanza senza gateway ecc.). Si deve dunque prevedere una locazione speciale che prenderà il nome di "no-place"; questo creerà complicazioni non indifferenti nella creazione di un algoritmo efficace per l'individuazione di tutti i tag.

L'agente, nella sua versione base, per adempiere ai suoi compiti si avvale di un insieme di IP e di una mappa per gestire i tag. La mappa, riempita inizialmente con i dati trovati su DB, mantiene essenzialmente tre informazioni sui tag: "id", un flag per capire se il tag è stato rilevato o meno da un gateway nell'ultimo ciclo di richieste "GET" e l'ultimo gateway che lo ha rilevato. Per quanto concerne i gateway, sono necessarie due informazioni: indirizzo IP e una soglia (threshold) che permette di stabilire un limite alla *regione* associata al gateway.

Algoritmo di scanning

Un timer, continuamente ad intervalli regolari, richiama un handler che provvede ad inviare una richiesta GET ad ogni gateway, secondo l'API specificata nella documentazione fornita dalla "BlueUp". Ogni risposta ricevuta contiene come payload un JSON in cui sono specificati i tag che suddetto gateway ha trovato nell'ultimo campionamento. In una prima fase si filtrano i tag in base al threshold del gateway per eliminare quelli con distanza presumibilmente troppo elevata e dunque esterni alla regione. Nei rimanenti si va a confrontare se la corrente locazione di ogni tag (campo "place" associato al gateway da cui

proviene la risposta) è coerente con quella salvata nella mappa dei tag: in caso positivo non si effettua nulla poiché il tag non si è mosso, altrimenti si aggiorna la locazione. In ogni caso si pone comunque a "true" il flag per segnare che il tag è stato rilevato. Infine si aggiornano batteria e timestamp se il valore di carica della batteria è differente da quello precedentemente salvato.
All'arrivo dell'ultima risposta (un contatore se ne occupa), si va a porre locazione *"no-place"* in tutti i tag non rilevati; questo può avvenire solamente dopo ricezione di tutte le rispsote in modo da avere una vista globale sulla situazione ed avere la certezza di mantenere i dati consistenti. Nel mentre vengono effettuati questi passaggi, ogni qualvolta si vada a cambiare il valore di locazione del tag, si invia sul bus di comunicazione condiviso da tutti gli agenti un messaggio il cui payload specifica "id" del tag e nuova regione in cui è stato rilevato; il "ServerAgent", in ascolto su questo tipo di messaggi, agirà di conseguenza.
Lo stub di codice nell'illustrazione 2.1 mostra uno scheletro del meccanismo descritto per l'elaborazione dei dati ricevuti dai gateway.

```
private void processBuffer(final Buffer b, final String ip, final String
    regionId) throws DecodeException {
        final JsonArray tagList = b.toJsonObject().getJsonArray("beacons");
        tagList.stream()
                .map(JsonObject.class::cast)
                .forEach(tag -> {
                    final int serial =
                        tag.getInteger(TagJsonProperty.BLUEUP_SERIAL);
                    final int battery =
                        tag.getInteger(TagJsonProperty.BATTERY);
                    final int rssi = tag.getInteger(TagJsonProperty.RSSI);
                   //Update battery
                   .
                   .
                   .
                   //Update place
                   .
                   .
                   .
                });
    }
```

Listato 2.1: Sezione di codice che illustra ad alto livello uno scheletro della funzione che ha il compito di elaborare le informazioni ricevute dai gateway.

A conclusione di queste operazioni, si ritorna al punto iniziale e si può dire

che un ciclo di campionamento può considerarsi concluso.

Contromisure per cadute di rete e malfunzionamenti

Nella pratica non è possibile avere sempre uno scenario di funzionamento perfetto in modo continuativo, dunque è giusto prendere contromisure per casi particolari in cui la corretta esecuzione deve avvenire a fronte di problematiche varie. I principali avvenimenti che possono modificare lo scenario di funzionamento sono:

- Ritardi nelle comunicazioni di rete.
- Caduta della rete.
- Malfunzionamento di uno o più gateway.

Si estendono dunque le informazioni sui gateway mantenute in memoria, aggiungendo un flag ausiliario in modo da tener conto, per ogni gateway, se una risposta sia arrivata o meno da esso nel ciclo di campionamento corrente. Ovviamente, ad ogni inizio ciclo il valore è impostato a "false" per ognuno e all'arrivo di una risposta, si cambia il valore corrispondente.

Se un gateway fosse guasto o se la rete fosse congestionata e dunque le comunicazioni dovessero impiegare troppo tempo, dopo un certo periodo, avviene il "fire" di un timer che segnala un errore, facendo si che l'applicazione consideri il gateway che ha scatenato l'errore come se avesse risposto in modo congruo con nessun tag rilevato.

Nel caso invece in cui arrivi una risposta duplicata poiché si sta aspettando ancora uno o più gateway dal ciclo precedente e intanto si è iniziato con il successivo, essa verrà processata ma in contatore non verrà aggiornato (poiché in quel momento una risposta da quel gateway è già segnata come arrivata).

Se la rete dovesse completamente cadere, si avrebbe una serie continua di errori e tutto resterebbe immutato; è vero che senza rete l'applicazione non potrebbe inviare notifiche agli utenti, rendendo il tutto non utilizzabile, ma nel momento in cui fosse ripristinata il tutto riprenderebbe automaticamente a funzionare in modo corretto.
La funzione mostrata nell'illustrazione 2.2 mostra concretamente l'impiego delle contromisure descritte.

```
this.client.get(DEFAULT_REQUEST_PORT, ip, RELATIVE_URL, resp -> {
            resp.bodyHandler(buf -> {
                if (this.currGwMap.containsKey(ip)) {
                //Set the gw to online if offline
                this.updateGwStatus(ip, true, p -> !p);
```

```
                    final JsonObject selectQuery = new JsonObject()
.put(GatewayJsonProperty.ADDRESS_PROPERTY, ip);
                    //Get gw region
                    DB.get().getConnection().find(DBCollections.GATEWAYS,
                        selectQuery, res-> {
                        if (res.result().size() > 0) {
                            final String regionId = res.result().get(0)
.getString(GatewayJsonProperty.PLACE_PROPERTY);
                            try {
                                this.processBuffer(buf, ip, regionId);
                            } catch (DecodeException e) {
                                //Gw not ready or src is not a gw
                                this.updateGwStatus(ip, false, p -> p);
                                System.out.println("DEGUG: Catch decode
                                    exception...!");
                            }
                        }
                        //Duplied reply
                        if (!this.currGwMap.get(ip).isRcvReply()) {
                            this.currGwMap.get(ip).setRcvReply(true);
                            this.incCounter();
                        }
                    });
                }
            });
        }).exceptionHandler(exc -> {
            if (this.currGwMap.containsKey(ip)) { //The (failed request) reply
                must be from a current existing gateway!
                System.out.println("DEBUG: Reply timeout from GW: " + ip);
                if (!this.currGwMap.get(ip).isRcvReply()) {
                    this.currGwMap.get(ip).setRcvReply(true);
                    this.incCounter();
                }
                //Set the gw to offline if online
                this.updateGwStatus(ip, false, p -> p);
            }
```

Listato 2.2: Sezione di codice che illustra la funzione che processa le risposte provenienti dai gateway, con tutte le precauzioni e contromisure del caso.

Sincronizzazione con DB

Il primo e più facile approccio è quello di intervenire direttamente nella modifica delle informazioni desiderate, agendo a mano sul DB e riavviando l'applicazione. Un approccio di questo tipo, oltre che estremamente scomodo e sconsigliato per utenti non competenti del settore, non sarebbe applicabile poiché, al fine di aggiornare le informazioni, si dovrebbe procedere con il riavvio dell'applicazione e ciò comporterebbe seri problemi come la chiusura delle web-socket e conseguente perdita di ogni sottoscrizione.

L'applicazione allora introduce la possibilità per l'utente di interagire con il sistema per la modifica delle informazioni su tag e gateway (da una dashboard browser nel nostro caso, ma non necessariamente) e nel contempo effettuare una sincronizzazione inline dei dati quando necessario.

Il "TagManagerAgent", usando molte delle informazioni salvate su DB, necessita di aggiornare le proprie in memoria quando queste venissero modificate. Questo tipo di sincronizzazione può essere effettuata in un unico lasso di tempo, ovvero nel momento che intercorre tra l'elaborazione dell'ultima risposta di un ciclo di campionamento e l'inizio del successivo ciclo. Questo perché una variazione dei dati a metà di un ciclo potrebbe portare a situazioni di inconsistenza dei dati. Le informazioni di supporto supplementari per implementare un meccanismo robusto e funzionante sono due per tipo di dispositivo (tag, gateway): flag booleano per indicare se sia o meno necessario effettuare una sincronizzazione e un flag che funge da barriera per effettuare la sincronizzazione. Fissato dunque il punto in cui effettuare la sincronizzazione, il metodo da attuare per effettuarla è banale: l'agente resta in ascolto su due tipi di messaggi per l'aggiornamento di tag e gateway; quando un altro agente modifica il DB invia un messaggio sul bus per segnalare che ci sono nuovi dati aggiornati. Consideriamo il caso dei tag.
All'arrivo di un messaggio di aggiornamento tag, l'agente pone il flag booleano corrispettivo a "true". Alla ricezione dell'ultima risposta del ciclo si controlla se vi è una richiesta di aggiornamento tag in sospeso e, in caso, si procede all'aggiornamento dei facendo uso del flag barriera per evitare che si cominci nel contempo un ulteriore ciclo. Analogamente avviene per i gateway.

L'illustrazione 2.3 mostra l'implementazione di una funzione usata per sincronizzare le informazioni dei tag, reperendole dal DB.

```
private void syncTagData() {
        this.pendingTagUpdate = false;
        this.doingTagDataSync = Future.future(); //It prevents from sending
            request while the data sync has not finished yet
        DB.get().getConnection().find(DBCollections.TAGS, new JsonObject(),
            res -> {
            if (res.succeeded()) {
                final Map<Integer, Tag> tmpMap = new
                    HashMap<>(this.tagMap);
                this.tagMap.clear();
                res.result().stream()
                            .forEach(t -> {
                                //Fill again map
                                final int id =
                                    t.getInteger(TagJsonProperty.SERIAL);
                                this.tagMap.put(id, new Tag(
                                  t.getString(DB.GENERIC_OBJECT_ID_KEY),
                                  t.getInteger(TagJsonProperty.SERIAL),
                                  t.getInteger(TagJsonProperty.BATTERY)));
                                //Keep previous location info
                                if (tmpMap.containsKey(id)) {
                                  this.tagMap.get(id)
                                    .setLocation(tmpMap.get(id).getLocation());
                                }
                            });
                this.doingTagDataSync.complete(); //Get requests can be sent
                    again
            } else {
                System.out.println("[BeaconManagerAgent] DEBUG: Failed to
                    read tags' set from the DB!");
            }
        });
    }
```

Listato 2.3: Sezione di codice che illustra l'algoritmo per l'aggiornamento dei tag.

Il diagramma di attività in figura 2.8 illustra l'algoritmo completo.

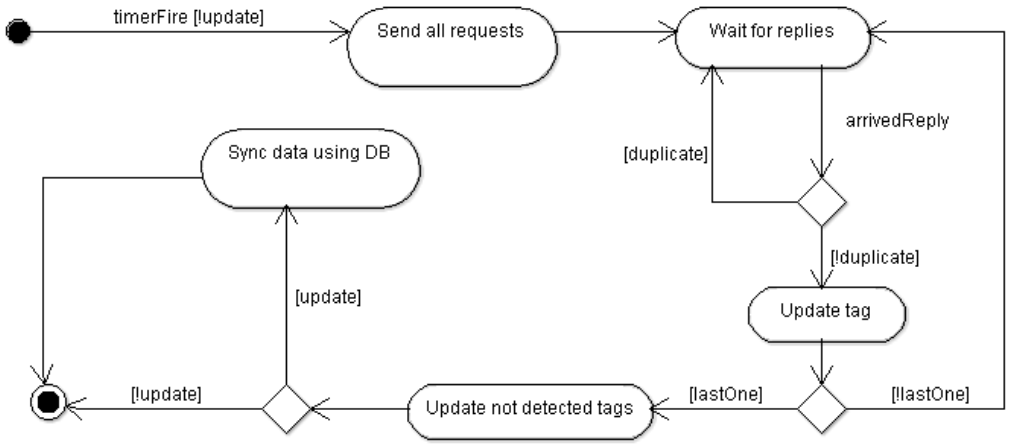

Figura 2.8: *Diagramma UML di attività raffigurante l'algoritmo per la rilevazione delle informazioni sui tag dai gateway; sono illustrate le varianti di funzionamento possibili.*

2.5.2 ServerAgent

"ServerAgent" [4] corrisponde al secondo agente in esecuzione nell'applicazione. Anche qui si ha un funzionamento interno basato su event-loop. I compiti che assolve sono principalmente di due tipi:

- Gestione delle web-socket per i meccanismi di registrazione e notifica.
- Gestione delle chiamate REST.

Tag: sottoscrizione e sistemi di notifica

Probabilmente la funzionalità portante dell'intero sistema, su cui è incentrata la base del progetto. Si vuole mettere a disposizione un modo per potersi registrare su un determinato tag e ricevere notifiche qualora questo cambi la propria posizione; l'operazione di registrazione verrà effettuata da apposita applicazione in esecuzione su tablet, scegliendo il tag corrispondente a quello usato. Uno stesso tag non può essere usato in contemporanea da più medici.

Il "ServerAgent" aspetta richieste di apertura di web-socket su un determinato URL (si consulti la sezione 3.1 per i dettagli approfonditi). Qualora la richiesta andasse a buon fine, un web-socket con il client è aperta l'invio delle notifiche ha inizio. L'applicazione provvede immediatamente a recuperare la posizione attuale del tag interessato e inviare una prima notifica. Successivamente, ogni qual volta che il tag cambia posizione.
Le notifiche sono inviate in tre casi possibili:

- **Evento di entrata:** il tag entra in una regione e di conseguenza è rilevato da un gateway; passaggio "no-place"/regione.
- **Evento di uscita:** il tag esce da una regione e non viene rilevato da alcun gateway; passaggio regione/"no-place".
- **Evento cambio diretto di regione** inserito per completezza, rappresenta il passaggio regione/regione, in cui il tag tra due campionamenti successivi cambia regione; più rari poiché il dominio considerato è molto ampio (intero settore ospedaliero) e un passaggio intermedio in situazione di "no-place" (passaggio per il corridoio, stanze di transito ecc.) è solitamente necessario.

[4] Il nome "ServerAgent" deriva dal tipo di interazione effettuata con l'utente, comportandosi per l'appunto da server in un classico schema client-server e rimanendo dunque in attesa di richieste da parte dei client (utenti); per massima precisione un appunto andrebbe fatto per la questione di notifica sulle web-socket (approccio "push").

A fine trattamento, sempre tramite tablet, la sottoscrizione al tag viene annullata: comporta nella pratica alla chiusura delle web-socket; a questo punto il tag può essere di nuovo utilizzato per seguire un nuovo trauma.

Il diagramma di sequenza in figura 2.9 mostra le interazioni tra operatori e servizio di "LocationService", esponendo la mediazione effettuata dalla web-socket e lo scambio di messaggi tra gli agenti.

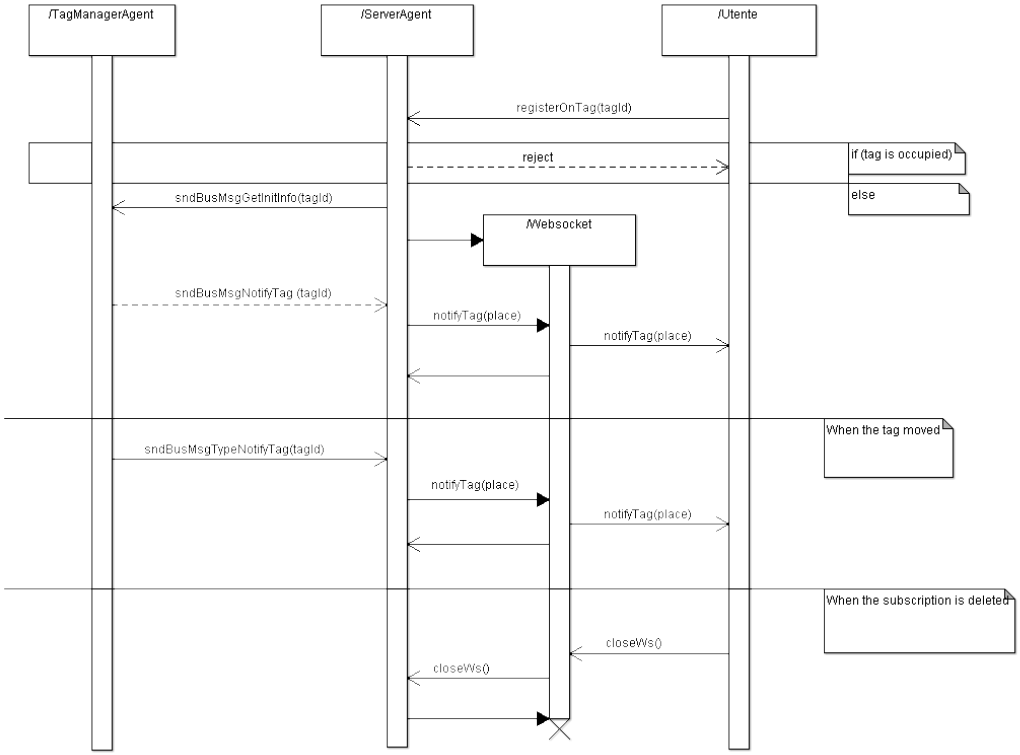

Figura 2.9: *Diagramma di sequenza che illustra come un operatore possa interagire con l'applicazione: inizio della sottoscrizione, ricezione di una notifica, chiusura della sottoscrizione.*

Interfacciamento a richieste REST

Richieste

L'agente resta in attesa di chiamate conformi alle norme REST, differenziate a seconda della semantica dell'operazione da compiere; nello specifico si hanno i seguenti tipi di richiesta accettati:

- **GET:** richiesta informazioni di tag, gateway o regioni.
- **PUT:** aggiornamento di un tag, gateway o regioni già esistenti in precedenza.
- **POST:** inserimento di un tag, gateway o regioni.
- **DELETE:** eliminazione di un tag, gateway o regioni.

Ogni altro tipo di richiesta sarà ignorato.
A seconda del tipo di informazioni richieste, si avranno i seguenti URL a cui aderire:

- `gt2/locationservice/api/tags/:id`
- `gt2/locationservice/api/gateways/:id`
- `gt2/locationservice/api/regions/:id`

L'unica eccezione è fatta dalle richieste che coinvolgono tutti i dati a disposizione su tag gateway o regioni e, in tal caso, gli URL a cui aderire sono i seguenti:

- `gt2/locationservice/api/tags/any`
- `gt2/locationservice/api/gateways/any`
- `gt2/locationservice/api/regions/any`

Ogni richiesta non conforme stabilito sarà ignorata.
Infine, a conclusione del protocollo di comunicazione, si è stabilito che il payload del messaggio, quando presente, debba consistere in un file JSON contente:

- **Inserimento:** tutti i campi necessari all'istanziare un nuovo oggetto, quindi [id, serialNumber] per i tag, [id, place, ipAddress, threshold] per i gateway e [id, name] per le regioni.

- **Aggiornamento:** id e tutti i campi aggiornabili, dunque [serialNumber] per i tag, [place, ipAddress, threshold] per i gateway e [name] per le regioni.

Una verifica è effettuata sulla corretta sintassi del file JSON, sulla conformità dei tipi di dati (string, int, bool ...) e sulla loro consistenza (chiavi esterne); un qualsiasi errore provoca il rigetto della richiesta.

Ovviamente, è verificato anche il fatto che l'operazione sia possibile (chiave già in uso, eliminazione di oggetti non esistenti ecc); si noti la questione con attenzione nei casi di inserimento e aggiornamento poiché alcuni campi possono fungere da chiavi secondarie, in particolare per il campo "serialNumber" dei tag e "ipAddress" dei gateway.

Risposte

Secondo il protocollo stabilito, il payload di ogni risposta inviata all'utente consiste in un file JSON contenente:

- **code:** number, codice di stato.
- **msg:** string, espressione dello stato sotto forma di stringa.

I possibili valori sono i seguenti:

- **0:** "Resquest satisfied correctly!"
- **1:** "Request rejected, it's not compliant to the protocol!"
- **2:** "An error occured while doing the selected operation, please try again!
- **3:** "The tag is currently occupied!"
- **4:** "At least one tag is occupied!"

Per massima precisione, è giusto sottolineare che nelle richieste GET andate a buon fine, il payload ovviamente consiste nelle informazioni richieste (e non in un JSON di codice "0").

2.6 Test e deployment

La prima fase di testing, assumibile di fatto come un *alfa-test*, è stata effettuata in parallelo allo sviluppo dell'applicazione per tutta la durata del progetto. Lo sviluppo, anche se non in modo prettamente canonico, si può riassumere in un approccio "agile" in quanto, causa deadline parziali da dover rispettare, si sono avute delle scadenza non prorogabili entro le quali sviluppare certi aspetti e, di conseguenza, testare quanto fatto. Vista la natura dell'applicazione, effettuare test sistematici nel modo convenzionale non è stato possibile e ci si è basati per la maggiore su test empirici ragionati in modo tale da valutare ogni possibile reale scenario di funzionamento.

La prima fase di deploy ha riguardato il posizionamento dei gateway nelle locazioni stabilite a priori. come mostrato in figura 2.10; questa operazione è stata effettuata del personale competente dell'ospedale. La parte che ha riguardato direttamente il team di sviluppo è stata piuttosto la fase successiva di dispiegamento del software e configurazione del sistema. Per prima cosa, tramite supporto del settore informatico dell'ospedale, si è fatto uso di un server DHCP per creare connettività e raggiungere i gateway tramite dashboard[5] in modo di assegnare loro un indirizzo IP statico ad ogni gateway. A questo punto si è configurato il sistema inserendo le informazioni necessarie su database e si sono configurate le varie regioni dei gateway (discussione dettagliata in sezione 3.3.2).

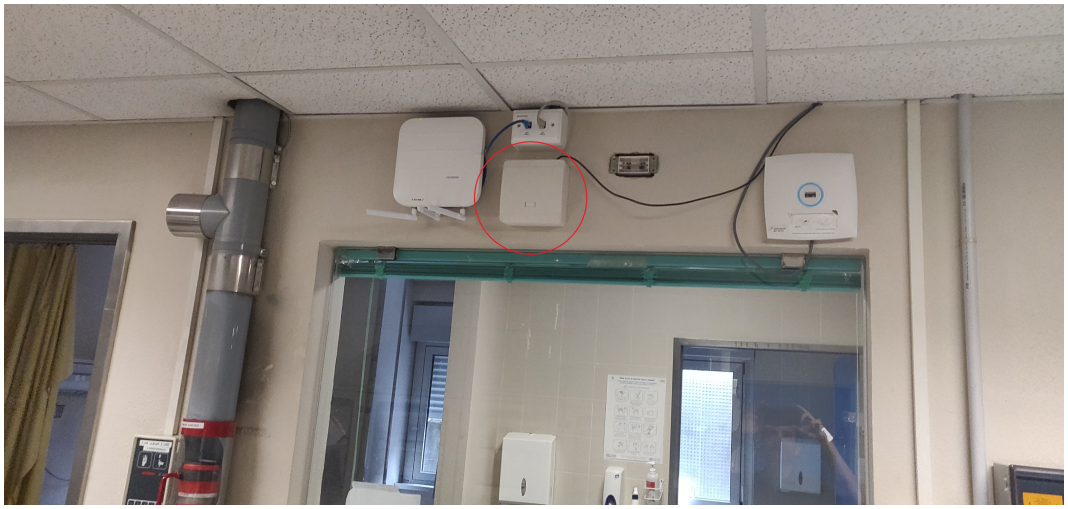

Figura 2.10: *Esempio di gateway (cerchiato in rosso) posizionato nella locazione stabilita; alla sua sinistra un "access point" alla rete ospedaliera.*

[5]Si intende la dashboard propria di ogni gateway, provvista dalla "BlueUp" e non di quella associata al "LocationService" presentata in sezione 2.7.

2.7 Dashboard

La dashboard, accessibile tramite browser, rappresenta il tradizionale metodo con cui l'utente medio può interfacciarsi al servizio "LocationService" per effettuare monitoraggio del servizio o qual si voglia operazione di gestione del sistema. Essa sfrutta l'API pubblica messa a disposizione dal "LocationService" e dunque rappresenta solamente una delle possibili soluzioni per assolvere ai suddetti compiti; in vista di possibili futuri sviluppi non è escluso che, se necessario, non possano essere sviluppate altre applicazioni, anche da terze parti.

2.7.1 Analisi dei requisiti

L'applicazione deve consentire la visualizzazione e modifica delle informazioni memorizzate riguardanti tag, gateway e regioni; con il termine modifica si intenda funzionalità di inserimento, aggiornamento e cancellazione. Le informazioni da dover elaborare sono id, seriale, batteria residua e timestamp per i tag; si parla invece di id, indirizzo IP, stato, regione e threshold per i gateway; infine si ha id e nome per le regioni. Si ricordi inoltre che nonostante numero seriale e indirizzo IP siano chiavi secondarie, rispettivamente per tag e gateway, sono comunque modificabili al contrario della chiave primaria (ovvero l'"id").

2.7.2 Progettazione

L'applicazione si compone di un componente principale che modella una home-page da cui è possibile monitorare il sistema nel complesso e di tre moduli secondari, uno per ciascun tipo di concetto da gestire, che permettono visualizzazione dettagliata e modifica delle informazioni. Infine, per completezza, è presente un'ulteriore componente che ha il solo compito di visualizzare un messaggio di errore qualora la risorsa richiesta dall'utente non sia reperibile.

2.7.3 Sviluppo

Lo sviluppo è avvenuto tramite uso di Angular v6.0.8 in modo tale da poter gestire con semplicità applicazioni complesse con un approccio di programmazione ad oggetti fornito da "TypeScript".

Lo scheletro base consiste di una parte fissa in cui è inserita una barra di navigazione e una parte gestita dal modulo di routing per visualizzazione delle componenti opportune in modo dinamico. Difatti, le componenti di gestione di tag, gateway e regioni si suddividono ciascuna in 3 sotto-componenti principali:

una per la visualizzazione, una per l'inserimento di nuovi elementi e una per modifica/eliminazione di elementi esistenti. Queste sotto-componenti non sono disgiunte, bensì mantengono rapporti di parentela padre-figlio come illustrato in figura 2.11.

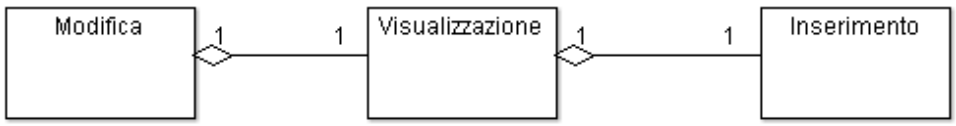

Figura 2.11: *Pseudo-diagramma UML delle classi raffigurante la struttura delle componenti prese in esame.*

Esempi della GUI realizzata sono riportati nelle immagini 2.12 e 2.13.

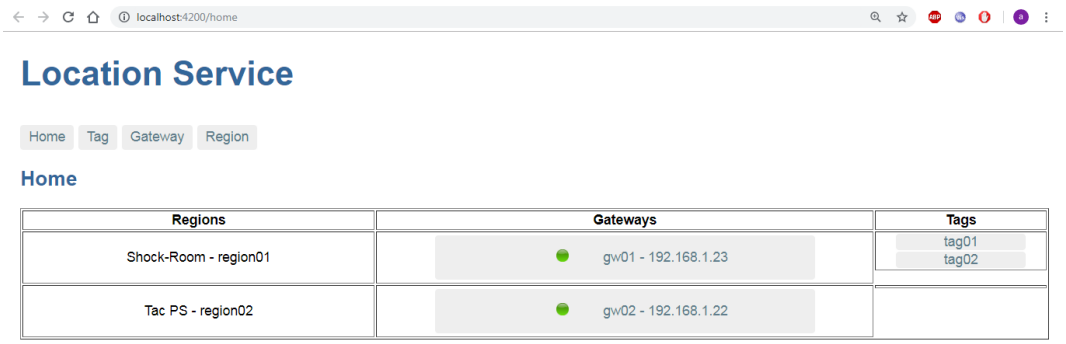

Figura 2.12: *Home-page presente nell'applicazione che comprende le informazioni base per la supervisione del sistema.*

Figura 2.13: *Pagina per la gestione dei gateway.*

Si riporta infine un esempio di codice sorgente nell'illustrazione 2.4 riguardo la pagina di home mostrata nella figura 2.12.

```
<h2> Home </h2>
<table border="1">
  <thead>
  <tr>
    <th>Regions</th>
    <th>Gateways</th>
    <th>Tags</th>
  </tr>
  </thead>
  <tbody>
  <tr *ngFor="let region of regionList">
    <td class="reg-td"> {{region.name}} - {{region.id}} </td>
    <td class="gw-td">
      <div *ngFor="let gw of gatewayList" >
        <a *ngIf="gw.place === region.id"
           href="http://{{gw.ipAddress}}/network" target="_blank">
          <figure>
            <img src="{{gw.online ? onlineImg : offlineImg}}" alt="{{gw.online
                ? 'Online' : 'Offline'}}" width="15" height="15"/>
          </figure>
          {{gw.id}} - {{gw.ipAddress}}
        </a>
      </div>
```

```
    </td>
    <td class="tag-td">
      <div *ngFor="let tag of tagMap">
        <a *ngIf="tag.region === region.id" href="tags/{{tag.id}}">
            {{tag.id}} </a>
      </div>
    </td>
  </tr>
  </tbody>
</table>
```

Listato 2.4: Codice sorgente che realizza la pagina di home della dashboard.

Capitolo 3

Integrazione nel progetto "TraumaTracker"

Il capitolo tratta in primis l'integrazione del micro-servizio all'interno del progetto "TraumaTracker". Successivamente sono riportati ed argomentati i dati rilevati nelle fasi di deploying e testing in modo tale da poter documentare caratteristiche (potenza di segnale, soglie delle regioni) e performance del servizio, sia in ambiente isolato sia sul campo.

3.1 Interfacciamento "Personal Assistant Agent"

Il progetto "TraumaTracker" non comprende esclusivamente il servizio di localizzazione e questo pone dunque il problema dell'interfacciamento verso gli altri moduli del sistema. L'unico punto di contatto verso l'esterno[1] realizza una funzionalità che permette ad un utente di registrarsi e venire notificati sul cambiamento di posizione dei tag. Il servizio consente di usufruire di un approccio "publish/subscribe" al fine di permettere la registrazione su un tag e venir notificati con approccio "pull". Il servizio espone un'API pubblica e raggiungibile ad un URL del tipo:

```
gt2/traumatracker/api/tags/:tagId/place
```

Il server in esecuzione nell'applicativo è in ascolto costantemente e, ad ogni richiesta conforme, apre una connessione persistente verso il client (PersonalAssistantAgent, in esecuzione sul dispositivo dell'utente) se possibile. Nel caso il tag desiderato sia già occupato, la richiesta sarà rigettata (non è possibile difatti monitorare uno stesso tag da più utenti in contemporanea).
Mentre non vi è invio di dati nel verso client-server (chiusura web-socket esclusa), le notifiche inviate all'utente assumono un formato JSON con il seguente contenuto:

- **tagId**: number, seriale del tag; non realmente necessario ma per finalità di controllo e debug.
- **place**: string, nuova locazione (regione) in cui il tag è stato rilevato.

[1] Da intendersi come contatto verso l'esterno del servizio e non dell'applicativo "LocationService" in sé; difatti "LocationService" e annessa dashboard sono da intendersi come un unicum per una vista ad alto livello del sistema poiché realizzano il servizio di localizzazione nel suo complesso e, dunque, non si parla di vero e proprio contatto verso l'esterno nelle comunicazioni tra i due.

3.2 Valutazione delle prestazioni

Si consideri un sistema composto da 2 tag[2], 2 gateway, 2 regioni e un periodo di campionamento T = 5s (tempo tra l'inizio di due cicli successivi).
Si supponga di operare in un ambiente in cui la rete sottostante sia esclusivamente dedicata al "LocationService"; questa ipotesi costruisce comunque un modello semplificato del problema, andando ad eliminare eventuali problemi di congestione, VLAN ecc. Una valutazione delle prestazioni in un ambiente sottoposto a queste condizioni, anche se non riscontrabili in linea generale nella realtà, può dare un'idea a priori di come il sistema in sé possa comportarsi.

Caso base: sistema a regime senza errori o aggiornamenti dei dati

Il caso più semplice possibile in cui ogni gateway risponde correttamente e non ci sono casi di aggiornamento sulla batteria residua dei tag o sincronizzazione di nuovi dati con il DB. Si riportano nella tabella 3.1 alcuni dati raccolti che misurano il comportamento del sistema.

start time	end snd	snd time	end reply process	tot cycle time
46.022	46.024	0.002	46.060	0.038
51.023	51.024	0.001	51.054	0.032
56.023	56.024	0.001	56.059	0.036
01.022	01.024	0.002	01.057	0.035
06.022	06.023	0.001	06.054	0.032

Tabella 3.1: *Sono rappresentati i dati raccolti espressi in secondi e una loro prima analisi.*

Analizzando i dati, si può notare come la fase di invio delle richieste ai gateway sia trascurabile (∼0.001s) mentre la fase di sincronizzazione, per ipotesi, non richiede tempo (0.000s); si rileva dunque che un ciclo di richieste in media richieda alcuni centesimi di secondo (∼34-35 ms).

[2]Si intende un insieme n tag fisici, effettivamente presenti nel sistema; nel DB può essere presente un numero arbitrario di m tag t.c. n $\subseteq m$.

Caso base: sistema a regime con aggiornamento continuo valori batteria

Si rilasci il vincolo per il quale si ignora l'aggiornamento della batteria quando necessario e si consideri il caso peggiore in cui questo debba avvenire ogni volta.
Si può intuitivamente capire che ora la fase di elaborazione delle risposte possa richiedere un tempo maggiore poiché si avranno nuove informazioni da trattare; i nuovi dati rilevati sono riportati in tabella 3.2.

snd time	end reply process	time to process
39.190	39.229	0.039
44.190	44.228	0.038
49.191	49.224	0.034
54.190	54.226	0.036
59.191	59.225	0.035

Tabella 3.2: *Sono rappresentati i dati raccolti espressi in secondi rappresentanti il tempo speso per l'elaborazione delle risposte ricevute.*

Confrontando i tempi di esecuzione con i precedenti, si può notare come non ci sia una netta differenza se non un aumento medio di ∼2ms. Si può ora stabilire dunque un tempo più preciso per il completamento di ogni ciclo, ovvero ∼35 ± 1 ms.

Caso base: sistema a regime con sincronizzazione dati

Si rilasci ora l'ultimo vincolo sul trattamento dei dati, ovvero quello secondo cui i dati fossero statici e non dinamicamente aggiornati con quelli presenti su DB.
A questo punto si viene ad inserire una nuova fase da considerare nel valutare il tempo totale del ciclo. I dati relativi a questa fase sono riportati nella tabella 3.3.

#items	start time	after sync	time spent
3	18.931	18.932	0.001
10	06.810	06.810	0.000
15	51.685	51.685	0.000

Tabella 3.3: *Vengono mostrati i tempi necessari per la sincronizzazione della vista dei dati, al variare del numero di elementi salvati su DB.*

Si può notare come la sincronizzazione non richieda tempi particolari e il rilascio del vincolo non abbia portato effetti significativi; il variare del numero di elementi da aggiornare non intacca il tempo impiegato[3].

Caso di errore: sistema a regime con risposte non ricevute

Si consideri ora il caso in cui ci sia un gateway da cui, per qual si voglia motivazione, non si riesca ad avere alcuna risposta. Per ogni richiesta, dopo un tempo TO = 20s dall'invio, l'applicazione crea una sua fittizia risposta in modo tale da informare l'applicazione sullo stato del gateway. Questo fa intendere che ci si accorga di una problematica sul gateway dopo un tempo minimo di 20s e che anche il relativo aggiornamento dell'applicazione adotta queste tempistiche. A questo punto però, visto l'accavallamento delle richieste in sospeso, l'aggiornamento torna al periodo prefissato T; l'immagine 3.1 esplica questo concetto in modo grafico.

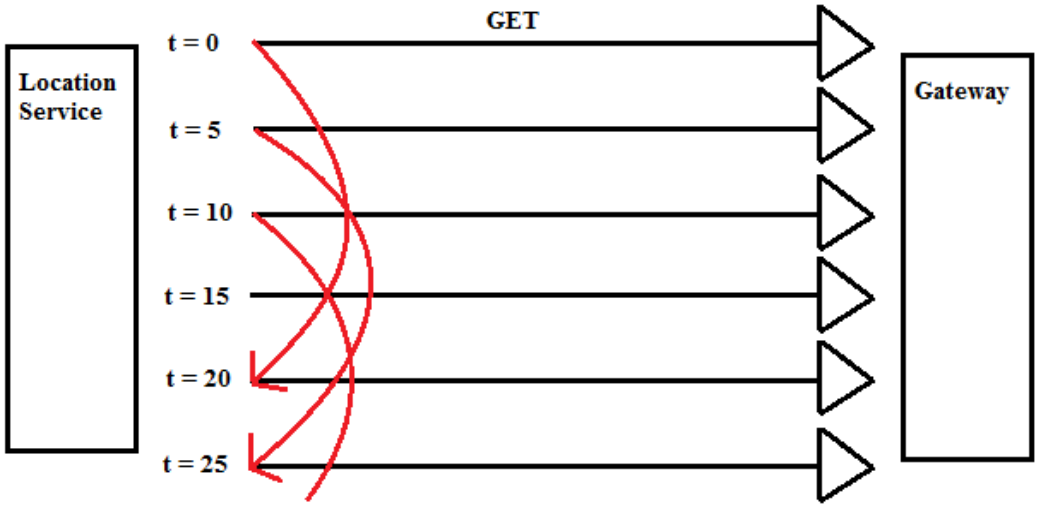

Figura 3.1: *Si rappresenta l'interazione periodica tra il "LocationService" e il gateway; il gateway non risponde alle richieste "GET", l'applicazione scatena un timeout dopo un un tempo fissato di TO = 20s e crea una risposta fittizia (in rosso). Poiché l'invio periodico di richieste non cessa, dopo un tempo pari a TO, l'aggiornamento torna ad essere con un periodo T.*

[3]I dati riportati sono rapportati al dominio applicativo del problema affrontato; ovviamente sono necessarie misure su scala opportuna nel caso in cui si cali il sistema in un differente dominio in cui la cardinalità dei tag può assumere un diverso ordine di grandezza.

I dati empirici rilevati a supporto nella tabella 3.4 supportano queste asserzioni.

start cycle	end cycle	tot time
56.069	77.175	21.106
61.070	82.074	21.004
66.070	87.075	21.005
71.070	91.071	20.001
76.071	97.078	21.007

Tabella 3.4: *Tabella che illustra tempistiche di richieste e risposte dopo il timeout, espresse in secondi.*

Qualora il gateway problematico dovesse tornare ad essere operativo, l'applicazione torna ad integrare le informazioni ricevute da tale gateway con una tempistica dipendente da alcuni parametri, precisamente:

$t_r = t_c + t_o + t_w$

dove:

- t_r: tempo di riattivazione, necessario per far considerare nuovamente al sistema le informazioni provenienti dal gateway.
- t_c: tempo fisso dovuto alle richieste fittizie che l'applicazione ha creato e che devono essere finite di processare; $t_k = TO \sim 20s$.
- t_o: indica il tempo che il gateway, facendo il boot del sistema, impiega a tornare online e capace di inviare risposte a seguito di richieste ricevute; questo fattore può essere presente o meno a seconda se le mancate risposte siano state dovute ad un mancato funzionamento del gateway o della rete; dati empirici rilevano che un boot del sistema, fino al raggiungimento della totale operatività del gateway, richiede ca. 15-20 s.
- t_w: tempo compreso tra 0 e T s; indica il tempo che il programma deve aspettare prima di iniziare un nuovo ciclo di campionamento e dunque poter interrogare il gateway nuovamente disponibile.

Si riportano i dati rilevati nella tabella 3.5.

gw online again	full system	tot time
30.000	71.464	41.464
30.000	69.321	39.321
30.000	70.089	40.089

Tabella 3.5: *I dati riportati in tabella sono stati rilevati a fronte di alcuni tentativi, ciascuno dei quali, per comodità pratica, mandando il gateway offline e dunque comprendendo il fattore* t_k.

3.3 Misurazioni sui valori di potenza

3.3.1 Misurazioni in ambiente isolato

Si definisca RSSI[4] come una rappresentazione numerica della potenza di segnale in comunicazioni wireless, attuando una proporzionalità diretta tra potenza di segnale e valore RSSI. Il massimo valore rilevabile è di 0 mentre un minimo in sé non esiste (dipende dal costruttore del dispositivo) ma, tuttavia, sono possibili diverse indicazioni su cui formulare le opportune considerazioni. Difatti, in linea generale, un valore RSSI fino a -65/70 dBm ca. è indice di buona connessione mentre un valore minore segnala una connessione debole, che potrebbe portare a perdita di dati o interferenze nei segnali analogici dovute al rumore di fondo; un valore sopra i -30 dBm resta invece molto difficile da ottenere se non a distanze veramente esigue.

Basandosi sul valore RSSI, sono state necessarie diverse misurazioni riguardanti la potenza del segnale proveniente dai tag e ricevuto dai gateway per diversi fini:

- Verificare a priori la presenza o meno di possibili problemi di ricezione del segnale.
- Stabilire in modo ottimo le soglie delle regioni[5].
- Evitare problemi di "overlapping"("sovrapposizione", stessa zona fisica coperta da più gateway).

Un esempio dei dati rilevati nelle misurazioni empiriche su varie fasce di distanza è riportato nella tabella 3.6, mentre il grafico in figura 3.2 mostra l'andamento dal valore RSSI al crescere della distanza.

[4] "Received Signal Strength Indication"

[5] Si ricordi, come già detto in precedenza nella sezione 2.1, la proporzionalità inversa tra potenza di segnale ricevuta e distanza tag-gateway; questo fa si che l'area fisica in cui il gateway dovrà operare sia definita in base ai valori di potenza ricevuti dai tag.

0.5m	1m	2m	4m	5m
-51	-52	-52	-72	-66
-47	-57	-59	-64	-59
-47	-56	-52	-73	-58
-49	-57	-51	-65	-69
-45	-59	-61	-67	-89
-48	-65	-52	-72	-58
-46	-59	-58	-65	-59
-45	-56	-50	-66	-59
-44	-60	-52	-74	-58
-48	-56	-50	-66	-66
-54	-63	-59	-74	-58

Tabella 3.6: *Tabella raffigurante un andamento del segnale nel tempo secondo diverse fasce di distanza tra tag e gateway; il periodo di campionamento usato è di 2s.*

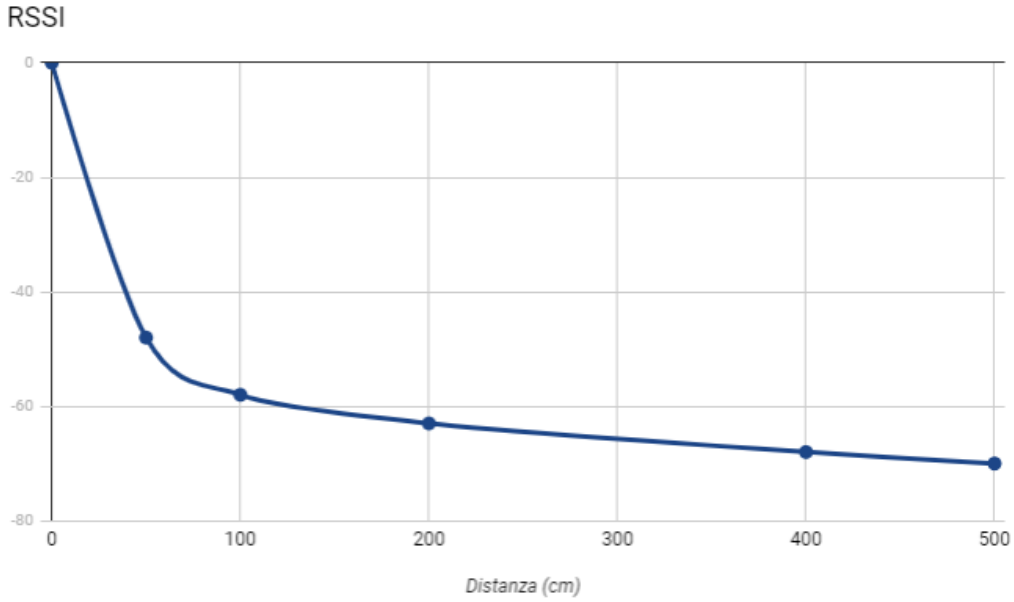

Figura 3.2: *Grafico raffigurante il valore RSSI come funzione della distanza.*

Come si può facilmente notare, l'andamento non è lineare ma bensì logaritmico (questo intrinsecamente dovuto al modo di calcolare l'RSSI).

Considerando il nostro dominio, il più delle locazioni in cui sono posizionati i gateway rientrano entro i 5m di raggio per la rilevazione e dunque, almeno

in linea teorica, non si avranno problemi nella corretta ricezione di segnali; inoltre va considerato attentamente che, anche se si dovesse intervenire in uno spazio di dimensione maggiore, vista la decrescenza logaritmica del grafico 3.2, si avranno solo esigue diminuzioni nel valore RSSI.

3.3.2 Misurazioni sul campo

Una serie di misurazioni sul campo espone l'effettivo comportamento del sistema a fronte di un insieme di caratteristiche dell'ambiente fisico non riscontrabili a priori in un'analisi teorica. Questo tipo di misurazioni sono necessarie per avere corretto "deploy" in cui le soglie delle regioni sono impostate in modo ottimale.
Le soglie sono dunque state decise solamente dopo alcune prove manuali, usando un approccio incrementale in modo tale da restringere la regione fino dove necessario. Per prima cosa si sono definite le aree fisiche da dover coprire con ciascun gateway e successivamente si sono misurate le potenze di volta in volta rilevate; inoltre si è fatto in modo anche da evitare problemi di "overlapping" tra regioni potenzialmente vicine, come mostrato in figura 3.3.

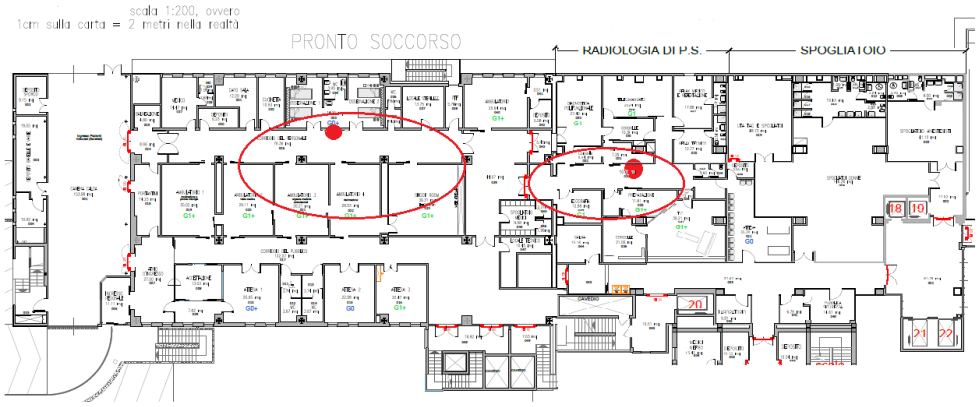

Figura 3.3: *Nella figura è riportata la piantina comprendente i settori di "Shock-Room" e "Tac-PS"; si può notare come ci siano due gateway (punti in rosso) che si trovano fisicamente vicini e le loro regioni possono teoricamente interferire se non si configurassero opportunamente le rispettive soglie.*

Una sostanziale differenza con quanto riportato nelle misurazioni teoriche a priori risulta essere una diminuzione delle potenze ricevute da gateway a parità di distanza dei tag. Questo è dovuto principalmente alle caratteristiche ambientali.
Considerando che il collocamento dei gateway è stato dettato da esigenze pratiche (prese a muro in postazioni fisse) e dunque non ottimizzato per ricevere

il massimo del segnale, ci si è ritrovati a gestire una serie di circostanze in cui ci fossero fisicamente frapposti oggetti altamente schermanti tra tag e gateway, muri in cemento armato e porte a piombo su tutti. Infine, in ambienti particolari, come le sale angiografiche ad esempio, si ha anche la presenza di macchinari provvisti di particolari magneti che interferiscono con i segnali dei tag riducendone il segnale. Operando in queste circostanze, le soglie per l'identificazione delle regioni sono risultate essere più basse di quanto ci si potesse aspettare ma si è riusciti tuttavia a creare un ambiente adeguato per l'esecuzione del sistema rispettando le specifiche provviste. Si riportano ora alcuni esempi di gateway e relative regioni nelle figure 3.4, 3.5 e 3.6.

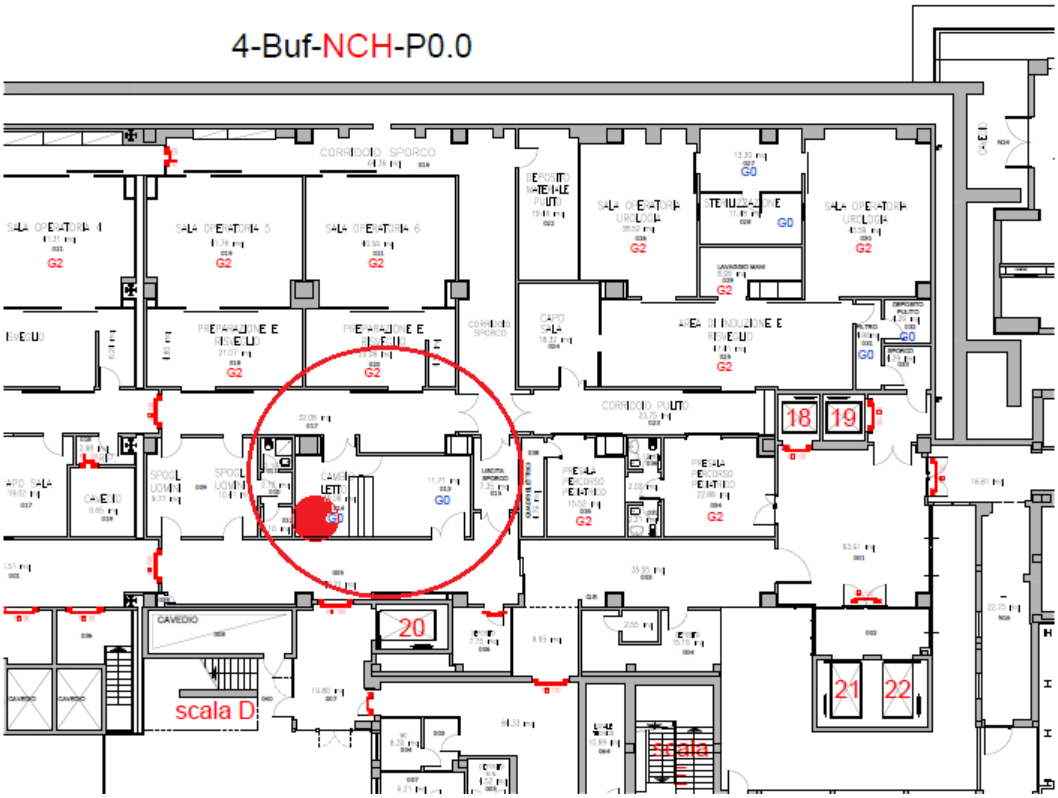

Figura 3.4: *Piantina del settore "blocco operatorio"; in rosso il gateway e relativa regione.*

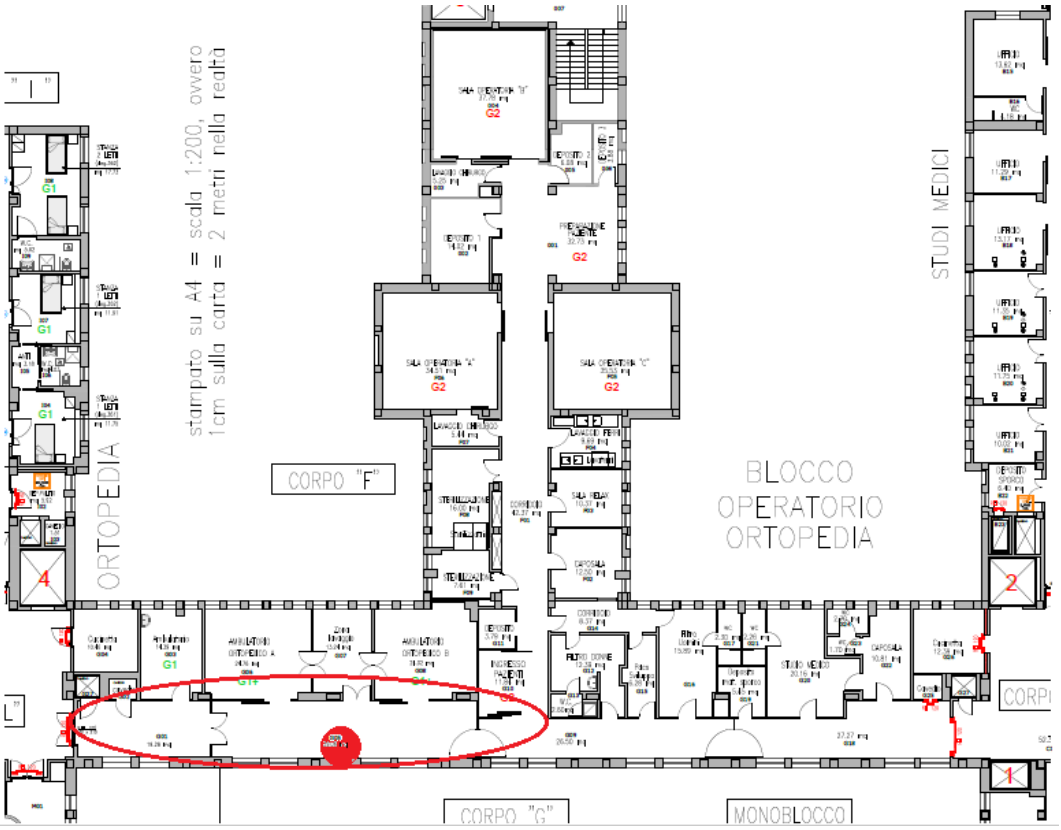

Figura 3.5: *Piantina del settore "ortopedia"; in rosso il gateway e relativa regione.*

Figura 3.6: *Piantina del settore "sala angiografica"; in rosso il gateway e relativa regione.*

Conclusioni

Concluso il percorso di sviluppo e test del servizio, posso ritenere che lo sviluppo di un sistema di tale complessità mi sia stato decisamente utile sotto diversi punti di vista.
In primo luogo ho avuto modo di apprendere numerose nuove competenze nel settore informatico e non solo, oltre ad approfondire diversi argomenti di cui avevo una conoscenza solamente superficiale. Inoltre ho avuto modo di confrontarmi con una realtà effettiva e tutte le sue problematiche che all'interno dell'ambiente universitario risultano spesso trasparenti ad uno studente.
In vista di futuri sviluppi, il progetto può fungere da base per la creazione di nuovi sistemi di tracciamento o l'estensione di questo già presente, scenario tra l'altro già tenuto in conto durante le fasi di sviluppo. Difatti è già possibile un'estensione dei settori ospedalieri coperti dal servizio ma non è escluso inoltre un'ampliamento del dominio del problema, passando dal solo tracciamento dei traumi all'inclusione anche della localizzazione di macchinari, strumenti e simili. In questo secondo caso, con l'inserimento di opportune modifiche, si può riutilizzare il "LocationService" poiché dotato di un sufficiente livello di astrazione e di una validità generale, non legata strettamente al dominio specifico del problema affrontato.
Il sistema non soffre di particolari problematiche ma è opportuno ricordare la discrepanza tra valori attesi e valori effettivi sul campo nelle potenze di segnale rilevate dai gateway .

Bibliografia

[1] http://www.multimac.it/soluzioni_scheda_ita.php/
nomeProdotto=Localizzazione_in_Tempo_Reale_nella_Sanit\%C3\
%A0/idcat=3/idsottocat=80/idprodotto=1910

[2] http://strategieaziendaliealtro.blogspot.com/2012/01/
cagr-compound-annual-growth-rate.html

[3] https://www.infotn.it/Comunicazione/
Newsletter/Link-n.-54-gennaio-2015/\\
Progetto-I-Locate-sperimentazioni-localizzazione-indoor-all-Ospedale-S.
-Maria-di-Rovereto

[4] https://www.tapmylife.com/

[5] http://www.eximia.it/soluzioni/localizzazione-e-sicurezza/

[6] http://www.id-est.it/impianti-rfid/97-localizzazione-macchine\
\-apparecchiature-medicali.html

[7] https://kontakt.io/

[8] http://www.multimac.it/soluzioni_scheda_ita.php/
nomeProdotto=Il_BLE_nel_controllo_degli_accessi/idcat=1/
idsottocat=178/idprodotto=1684

[9] http://elettronica-plus.it/bluetooth-4-2-lo-standard-ideale-per-applicazioni
90456/

[10] http://www.jimprice.com/prosound/db.htm

[11] http://www.sengpielaudio.com/calculator-volt.htm

[12] https://www.metageek.com/training/resources/
understanding-rssi.html