

ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Ingegneria e Architettura
Corso di Laurea in Ingegneria Elettronica, Informatica e
Telecomunicazioni

PROGETTAZIONE DI UN APPLICATIVO DI SUPPORTO AL MONITORAGGIO DI TEAM SITUATI

Elaborato in
FONDAMENTI DI INFORMATICA A

Tesi di Laurea di
LUCA MAESTRI

Relatore
Prof. MIRKO VIROLI

Correlatori
Ing. ANGELO CROATTI
Prof. ALESSANDRO RICCI

Anno Accademico 2013 – 2014
III^a Sessione di Laurea

PAROLE CHIAVE

Android

Web Service REST

Wearable Computing

Bluetooth

Activity Recognition

*A mio fratello, ai miei genitori, ai miei amici e a Martina
che mi hanno supportato durante questo lungo percorso.*

Indice

| | |
|--|-----------|
| Introduzione | ix |
| 1 Le tecnologie alla base | 1 |
| 1.1 Android | 1 |
| 1.1.1 Le Basi di Android | 2 |
| 1.1.2 Activity | 4 |
| 1.2 Web Service | 6 |
| 1.3 Bluetooth | 7 |
| 1.4 GPS | 8 |
| 2 Analisi | 11 |
| 2.1 Requisiti | 11 |
| 2.1.1 PARTE A - Sistema Wearable | 11 |
| 2.1.2 PARTE B - Sistema per la Centrale di Controllo | 12 |
| 2.1.3 Requisiti non funzionali | 12 |
| 2.2 Analisi dei Requisiti | 13 |
| 2.2.1 Casi d'uso | 14 |
| 2.2.2 Scenari | 17 |
| 2.3 Analisi del Problema | 19 |
| 2.3.1 Struttura | 19 |
| 2.3.2 Interazione | 21 |
| 2.3.3 Dispositivo WristOx2 | 21 |
| 3 Progetto | 27 |
| 3.1 Componente Android | 27 |
| 3.1.1 Logic Architecture | 27 |
| 3.1.2 Struttura | 27 |
| 3.1.3 Interazione | 32 |
| 3.2 Componente di Comunicazione | 33 |
| 3.2.1 Logic Architecture | 35 |
| 3.2.2 Struttura | 36 |
| 3.2.3 Interazione | 36 |

| | |
|----------------------------------|-----------|
| 4 Implementazione | 41 |
| 4.1 Componente Android | 41 |
| 4.2 Web Service REST | 48 |
| 4.3 Test | 53 |
| Conclusioni | 57 |
| Ringraziamenti | 59 |
| Bibliografia | 61 |

Introduzione

La sicurezza degli individui è da sempre una tematica d'interesse per l'uomo, attualmente questo problema viene trattato in diversi ambiti, dalla sicurezza sul posto di lavoro, alla sicurezza nelle scuole. La totale sicurezza si ha soltanto in situazioni prive di pericoli, condizione irrealizzabile in molti degli ambiti della vita quotidiana delle persone. Diventa quindi di grande interesse la tematica del pronto intervento degli organi di soccorso, in tutte le situazioni in cui le persone possono venirsi a trovare, anche durante lo svolgimento delle azioni che vengono compiute quotidianamente nella vita dell'uomo.

Con l'esplosione dell'innovazione tecnologica, costantemente accompagnata dalle discipline informatiche che assistono l'uomo in ormai tutti gli aspetti della propria vita, dal tempo libero all'ambito lavorativo, nasce l'idea di aiutare gli organi di soccorso nello svolgimento delle loro mansioni con il supporto delle tecnologie informatiche, con lo scopo di ridurre i tempi necessari al soccorso degli individui in situazioni di pericolo.

Il caso di studio reale, trattato in questo elaborato, nasce dall'osservazione che una sempre maggiore diffusione della pratica sportiva, come fenomeno di massa, genera una costante ricerca di incolumità personale e può trovare soluzioni dalla tecnologia, oramai largamente diffusa nella popolazione. L'idea, su cui si vuole focalizzare l'attenzione per lo sviluppo della tesi, è quella di realizzare un sistema informatico composto da più parti disgiunte che comunicano tra loro per rivelare, con estremo tempismo, situazioni di pericolo durante lo svolgimento dell'attività di allenamento di atleti, sia professionisti che dilettanti, supportati da tecnologie wearable per monitorarne lo stato fisico.

La tematica delle moderne tecnologie location-based, lo sviluppo di sistemi di comunicazione distinti, adatti sia a uno scambio di dati wireless tra dispositivi a stretto contatto, sia via rete, oltre alle conoscenze riguardanti l'ambiente mobile, saranno tutti ambiti che verranno affrontati nel corso di questo lavoro.

Lo scopo della tesi è quello di giungere alla realizzazione di un prototipo, che si basi su tecnologie specifiche quali GPS, Android e Bluetooth, per l'infrastruttura di un sistema che può essere visto come l'unione di tre macro

parti distinte, centrale di controllo, dispositivo mobile e pulsossimetro. Concentrandosi in particolare sugli ultimi due componenti citati e realizzando un sistema di comunicazione che si basa su un Web Service ispirato al modello REST, si giungerà, attraverso un attento processo logico dettato dai canoni dell'ingegneria del software, al prototipo finale. Il prototipo che sarà realizzato rappresenterà oltre che un primo sistema funzionante e operativo, un punto di partenza per estensioni future, con lo scopo di perfezionare le funzionalità già esistenti o di fornirne di aggiuntive. Infatti, un sistema di questo tipo può essere facilmente ampliato, vista la sua estendibilità e modularità, con il supporto di nuove tecnologie, come il sistema di connettività NFC. Ulteriormente, si può decidere di fornire un livello di assistenza maggiore andando a potenziare i meccanismi già presenti, come ad esempio migliorando l'algoritmo di rilevamento degli allarmi o agganciando un DBMS alla componente di comunicazione realizzata.

Panoramica dei contenuti

Il lavoro che verrà realizzato, riguardante il caso di studio preso in esame, sarà presentato attraverso quattro capitoli che andranno, in questo ordine, a trattare, da prima, le tecnologie alla base del sistema, successivamente i requisiti richiesti, accompagnati da una profonda analisi delle caratteristiche delle funzionalità necessarie, insieme ad un'analisi utile a individuare i problemi che nascono da queste esigenze, tramite lo studio delle tecnologie e dei dispositivi utilizzati. Questa analisi porterà successivamente allo sviluppo di un preciso progetto che verrà illustrato nel terzo capitolo, dove saranno riportati, in particolare, i passaggi fondamentali che porteranno alla realizzazione, nel quarto capitolo, di un applicativo di supporto al monitoraggio di team situati. In conclusione verranno presentati i punti di forza e le caratteristiche del sistema che verrà implementato, oltre ai possibili sviluppi futuri.

Capitolo 1

Le tecnologie alla base

1.1 Android

Android è un sistema operativo sviluppato da Google principalmente per device mobili con touchscreen, adotta una politica di licenza di tipo open source, escluse alcune versioni intermedie, ed è basato sul kernel Linux 2.6 e 3.x (da Android 4.0 in poi). Per via della sua natura open source molte società produttrici di smartphone e tablet hanno deciso di adottare Android per i loro dispositivi, portando il sistema operativo ad essere uno tra più diffusi a livello mondiale.

Il linguaggio di riferimento per programmare le applicazioni Android è essenzialmente Java, esteso con API specifiche, ad eseguire il codice non è però la normale JVM (Java Virtual Machine), ma una versione appositamente creata da Google, la Dalvik Virtual Machine. Quest'ultima utilizza un formato pensato per soddisfare le esigenze dei dispositivi mobili che, rispetto ai pc, soffrono di poca memoria e scarsa potenza di calcolo. Questa macchina virtuale è in grado di eseguire codice contenuto in file con estensione `.dex`. I file `.dex`, generati a partire dal bytecode Java, hanno un tipo di compressione che permette di dimezzare lo spazio utilizzato rispetto ai file `.jar` non compressi. Anche la Dalvik Virtual Machine, similmente per quanto accade per la JVM, implementa un Garbage Collector, liberando pertanto lo sviluppatore dall'onere della gestione della memoria.

Un'interessante particolarità riguarda la singolare nomenclatura utilizzata per le versioni di questo sistema operativo, il quale segue un ordine alfabetico e una precisa convenzione per i nomi, che appartengono tutti a nomi di dolci: partendo da Cupcake, seguita da Donut, proseguendo con Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream, Sandwich, Jelly Bean, Kit Kat, fino ad arrivare alla più recente, lanciata il 14 ottobre 2014, la versione 5.0, col nome di Lollipop.

Nei paragrafi successivi verrà illustrata in modo sintetico la struttura di Android, in particolare delle caratteristiche fondamentali dei principali meccanismi, per fornire una panoramica generale e comprenderne il funzionamento.

1.1.1 Le Basi di Android

I creatori di Android sviluppando questa piattaforma si sono ispirati a due principi fondamentali, questi sono alla base della maggior parte delle scelte progettuali operate e sono:

- **sicurezza**, ogni applicazione viene vista dal sistema come un utente e vive in un proprio processo dove viene allocata una nuova istanza della virtual machine, ciò per evitare che problemi a un'applicazione possano avere ripercussioni sulle altre, inoltre ogni applicazione ha il proprio spazio dedicato per custodire dati e lavorare in isolamento dal resto, ovviamente le applicazioni possono comunicare ed interagire, attraverso specifici meccanismi messi a disposizione.
- **tutela delle risorse**, benchè Android venga commercializzato su dispositivi come smartphone, dotati di hardware non di poco conto, esso è stato progettato per sistemi embedded, storicamente dotati di poche risorse di memoria a disposizione e ha avuto sin da subito uno spirito parsimonioso, infatti un'importante funzionalità offerta è quella di distruggere e ricreare parti di un'applicazione solo ed esclusivamente quando strettamente necessario, in maniera il più possibile impercettibile all'utente.

Per quanto riguarda l'aspetto strutturale delle applicazioni Android, esse sono basate su quattro elementi costitutivi fondamentali, i quali vengono elencati di seguito:

- **Activity**: un'applicazione Android presenta la propria specifica interfaccia grafica mediante le Activity. Ogni singola schermata di un'applicazione è rappresentata quindi grazie a un'Activity, siccome è la componente con cui l'utente ha il contatto più diretto questa verrà discussa più nel dettaglio nel prossimo paragrafo.

- **Service:** senza interazione diretta con l'utente, svolgono operazioni di lunga durata interamente in background, i service vengono creati da altri componenti delle applicazioni, ad esempio dalle activity, e continuano a svolgere il proprio compito anche quando l'utente passa a un'altra applicazione. Inoltre un componente può essere abbinato ad esso in modo da poterci interagire e con la possibilità inoltre di scambiare dati tra i diversi processi [4].
- **Content Provider:** incarnando il principio di sicurezza, permettono di attingere a dati presenti in processi differenti da quello attualmente in esecuzione, essi gestiscono quindi l'accesso ad un insieme strutturato di dati garantendone la sicurezza.
- **Broadcast Receiver:** sono componenti ideati per reagire a determinati eventi, vengono utilizzati per ricevere notifiche riguardanti l'avvenimento di determinate condizioni segnalate a livello di sistema, cioè in maniera *broadcast*.

Tutti le componenti di un'applicazione hanno un ciclo di vita che ha inizio nel momento in cui Android le istanzia, questo in seguito ad una richiesta asincrona chiamata intent, il ciclo di vita termina quando le loro istanze vengono distrutte. Più in generale gli **Intent** rappresentano un messaggio con cui si richiede, al sistema operativo, l'esecuzione di un'azione da parte di un'altra componente. Questi possono essere di due tipi differenti:

- **Intent Espliciti**, questi specificano un componente particolare in modo da indicare quale classe deve essere avviata, solitamente non specificano altre informazioni e vengono utilizzati essenzialmente per richiedere l'avvio di specifiche componenti per soddisfare le richieste dell'utente.
- **Intent Impliciti**, questi non specificano un componente preciso ma richiedono un'azione particolare specificando opportuni parametri, ma delegando al sistema la scelta di quale componente utilizzare per rispondere alla richiesta. Nel caso in cui il sistema individui più alternative in grado di soddisfare la richiesta ricevuta, la scelta finale verrà demandata all'utente.

Una componente fondamentale alla base delle applicazioni Android, di cui ancora non si è ancora discusso, è il file manifest. Ogni applicazione deve possedere un particolare file XML chiamato **AndroidManifest.xml** (esattamente con questo nome), nella propria root directory. In esso vengono riportate varie informazioni riguardanti l'applicazione, informazioni che il sistema deve possedere prima di poter eseguire qualsiasi parte di codice di essa. Questo file, tra le varie cose, descrive le componenti dell'applicazione, come le activity, di cui è composta, inoltre dichiara quali permessi deve possedere per il proprio funzionamento, oltre che per poter accedere ai dispositivi hardware che intende utilizzare.

1.1.2 Activity

Come accennato precedentemente, un'activity è un'entità alla quale è collegata una particolare interfaccia utente. Ogni applicazione mostra all'utente una serie di activity attraverso le quali è possibile interagire con il sistema. Di tutte le activity dichiarate all'interno del file manifest di un'applicazione, una deve essere marcata come iniziale, essa sarà il punto di partenza dell'app, e da questa, in seguito alle azioni dell'utente, saranno avviate le altre, attraverso il meccanismo degli intent descritto in precedenza, mano a mano che vengono aperte activity queste si dispongono a formare uno stack, in cui ogni istanza di activity è considerata autonomamente.

Comprendere il ciclo di vita di un'activity è fondamentale per poter programmare su Android, per fornire un livello di dettaglio maggiore, al fine di una migliore comprensione dell'argomento, esso viene riportato in figura 1.1.

I metodi che vengono forniti per interagire con il ciclo di vita di un'activity sono sette:

- **onCreate()**, questo metodo viene chiamato quando la activity viene creata per la prima volta ed è sempre seguito dal metodo **onStart()**.
- **onRestart()**, Chiamato dopo lo stop di un'applicazione, è sempre seguito dal metodo **onStart()**.
- **onStart()**, chiamato quando l'activity diventa visibile, può essere seguito da **onResume()** se l'activity va in primo piano e da **onStop()** se essa va in background.
- **onResume()**, chiamata quando l'activity è in cima allo stack, l'utente può quindi interagire con essa.

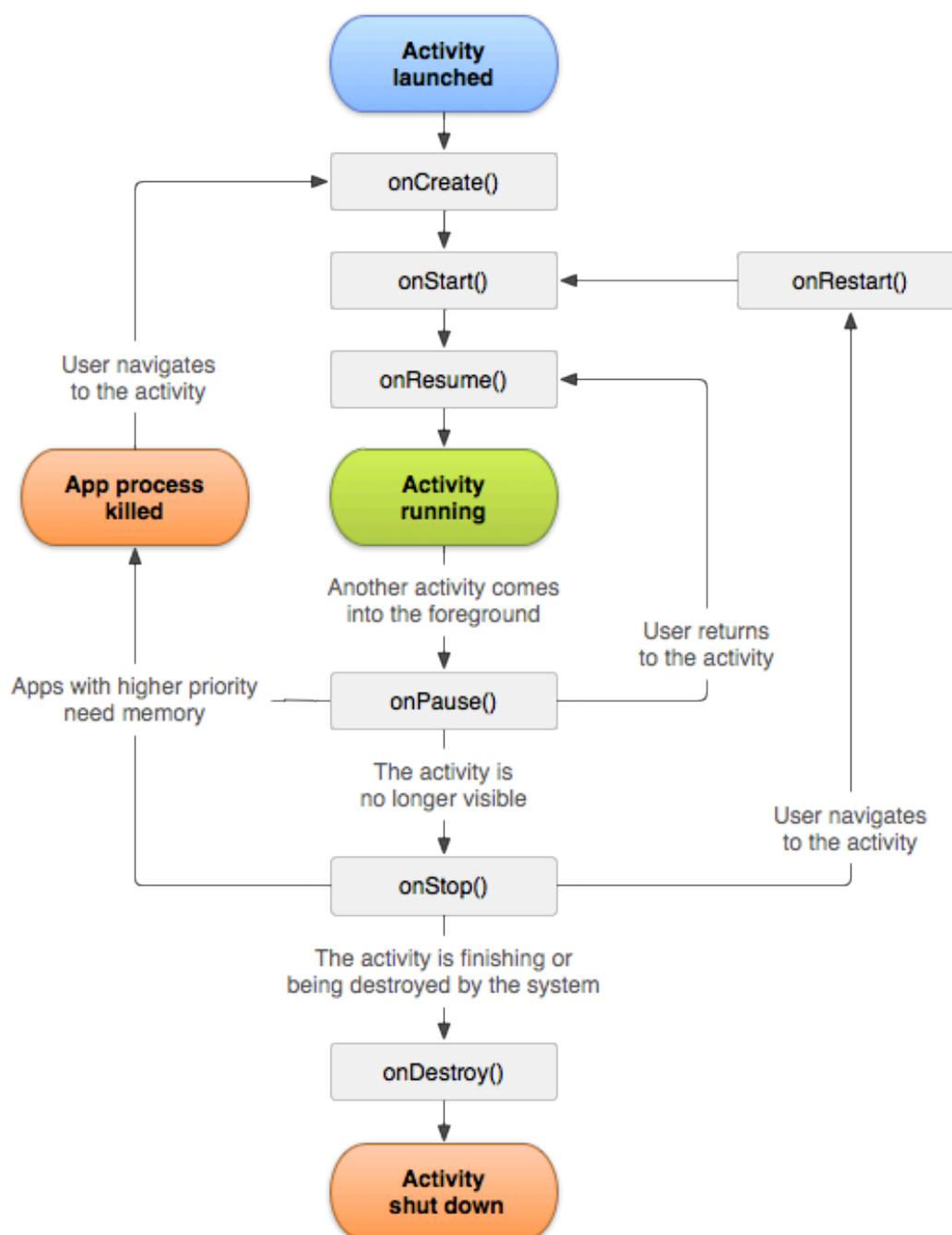


Figura 1.1: Ciclo di vita di un'activity in Android, fonte [7].

- **onPause()**, chiamata quando il sistema è in procinto di riavviare un'activity precedente, viene utilizzata per fermare attività intensive per la CPU come le animazioni e per memorizzare dati non ancora salvati. Se l'applicazione torna in primo piano è seguito da **onResume()** altrimenti da **onStop()**.
- **onStop()**, chiamato quando l'activity non è più visibile all'utente, se questa activity proseguirà verso la terminazione sarà seguita dal metodo **onDestroy()** altrimenti sarà chiamato **onRestart()**.
- **onDestroy()**, questo metodo viene chiamato quando l'applicazione viene terminata, sia nel caso in cui sia l'utente a chiederlo, sia nel caso in cui il sistema decida di farlo per ragioni di spazio.

1.2 Web Service

I Web Service forniscono un mezzo standard di interoperabilità tra diverse applicazioni software, in esecuzione su una varietà di piattaforme e/o framework. Il consorzio OASIS (Organization for the Advancement of Structured Information Standards) ed il W3C sono i principali responsabili dell'architettura e della standardizzazione dei Web Service.

Secondo la definizione data dal World Wide Web Consortium (W3C) un Web Service è un sistema software progettato per supportare l'interazione tra diversi elaboratori su di una medesima rete, ovvero in un contesto distribuito. È dotato di un'interfaccia descritta in formato automaticamente elaborabile quale, ad esempio, il WSDL, *Web Services Description Language*, attraverso di essa altri sistemi possono interagire con il Web Service stesso, attivando le operazioni descritte nell'interfaccia tramite appositi messaggi, in genere veicolati tramite HTTP con una serializzazione XML in combinazione con altri standard Web-related[8].

Esistono due approcci architetturali differenti per la realizzazione di Web Service, Service Oriented Architecture (SOA) e Resource Oriented Architecture (ROA), queste due tipologie di architetture identificano due approcci distinti per la realizzazione di Web Service: SOAP (Simple Object Access Protocol) e REST (Representational StateTransfer). La prima evidente differenza tra i due tipi di Web Service è la visione del Web proposta come piattaforma di elaborazione. REST propone una visione del Web incentrata sul concetto di risorsa mentre i SOAP Web Service mettono in risalto il concetto di servizio. L'approccio dei SOAP Web Service si serve di protocolli di rete come trasporto sottostante e di SOAP come reale protocollo di trasporto, e utilizza WS-*, con un chiaro richiamo al prefisso utilizzato nei vari standard definiti dal W3C,

come insieme di specifiche per supportare funzionalità di alto livello. REST, invece, sfrutta appieno la semantica e la ricchezza dei comandi HTTP e le sue funzionalità. Perchè un Web Service sia conforme alle specifiche REST deve avere alcune determinate caratteristiche, in primo luogo deve essere basato su un'architettura client/server, inoltre la comunicazione deve essere di tipo *stateless*, cioè deve trattare ogni richiesta come un'operazione indipendente, senza quindi richiedere al server di memorizzare informazioni di sessione o di stato, su ogni partner di comunicazione per la durata di richieste multiple. In aggiunta ogni risorsa deve avere un indirizzo univoco Uniform Resource Identifier (URI), semplici stringhe di testo che si riferiscono alle risorse Internet (documenti, risorse, persone e indirettamente a qualunque altra cosa).

Il modello REST prevede poi, che l'interazione con le risorse avvenga attraverso un insieme di operazioni, le principali sono GET, POST, PUT, DELETE. Il formato usato per la rappresentazione di una risorsa è chiamato *media type* ed in genere vengono usati, a questo scopo, linguaggi come XML oppure JSON.

1.3 Bluetooth

Bluetooth[1] è una tecnologia wireless standard per il trasporto dati a corta distanza, che usa onde corte della banda 2.4 -2.485 Ghz, per dispositivi fissi e mobili. È uno degli standard maggiormente usati per la creazione di reti personali (PAN: Personal Area Network). Il nome è ispirato a Harald Blåtand, Harold Bluetooth in inglese, nome di un antico re vichingo, vissuto tra il 940 ed il 981 D.C. La particolarità di Harald Bluetooth fu l'unione di due paesi tanto diversi tra loro, come Danimarca e Norvegia, sotto un unico regno, con lui a capo, invece il celebre logo della tecnologia è composto da rune nordiche analoghe alle moderne H e B.

La specifica Bluetooth è stata sviluppata da Ericsson e in seguito formalizzata dalla Bluetooth Special Interest Group (SIG). La SIG, la cui costituzione è stata formalmente annunciata il 20 maggio 1999, è un'associazione formata da Sony Ericsson, IBM, Intel, Toshiba, Nokia e altre società che sisono aggiunte come associate o come membri aggiunti.

Questo standard è stato progettato con l'obiettivo primario di ottenere bassi consumi, un corto raggio d'azione (fino a 100 metri di copertura per un dispositivo di Classe 1 e fino ad un metro per dispositivi di Classe 3) e un basso costo di produzione per i dispositivi compatibili.

Lo standard doveva consentire il collegamento wireless tra periferiche come stampanti, tastiere, telefoni, microfoni, ecc. a computer o PDA o tra PDA



Figura 1.2: Logo Bluetooth.

e PDA. Attualmente più di un miliardo di dispositivi montano un'interfaccia Bluetooth.

Il protocollo Bluetooth lavora nelle frequenze libere di 2,45 GHz. Per ridurre le interferenze il protocollo divide la banda in 79 canali e provvede a commutare tra i vari canali 1.600 volte al secondo (frequency hopping).

La versione 1.1 e 1.2 del Bluetooth gestisce velocità di trasferimento fino a 723,1 kbit/s. La versione 2.0 gestisce una modalità ad alta velocità che consente fino a 3 Mbit/s. La versione 4.0 arriva ad una velocità di 24 mbps (3 MB al secondo). Questa modalità però aumenta la potenza assorbita. La nuova versione utilizza segnali più brevi, e quindi riesce a dimezzare la potenza richiesta rispetto al Bluetooth 1.2 (a parità di traffico inviato).

1.4 GPS

Il GPS[9], Global Positioning System, è un sistema di posizionamento e navigazione satellitare, è stato creato e realizzato dal Dipartimento della Difesa statunitense, originariamente disponeva di 24 satelliti, ed è diventato pienamente operativo circa a metà degli anni '90. Esso sfrutta una rete di satelliti artificiali in orbita intorno alla Terra per stimare la posizione geografica di un dispositivo in grado di ricevere un segnale radio da almeno quattro satelliti, il ricevitore basandosi sul metodo della trilaterazione, calcola la propria distanza dai satelliti deducendola dal ritardo di propagazione del segnale da essi inviato, in base a questo è in grado di stimare le coordinate terrestri in cui si trova.

Il GPS ha permesso lo sviluppo di software per dispositivi mobile inerente alla localizzazione, come mappe per la navigazione assistita, che hanno rivoluzionato il mercato dei servizi basati sul posizionamento. Questa tipologia di servizi è stata frenata, in un primo momento, dalla relativa lentezza con cui un dispositivo GPS la proprio posizione all'avvio. L'introduzione nei moderni smart-device di un sistema di posizionamento più sofisticato, chiamato Assisted Global Positioning System (A-GPS), ha però permesso di rimuovere gli ostacoli iniziali riscontrati dalla tecnologia. Per velocizzare il rilevamento della posizione si combina l'uso di reti telefoniche e Wi-Fi al GPS, il ricevitore

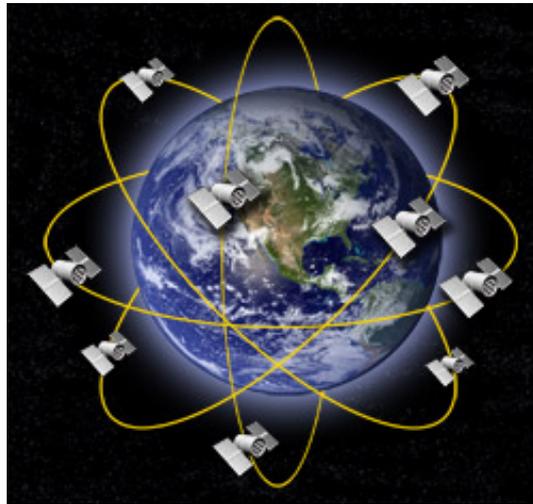


Figura 1.3: Rappresentazione del sistema di satelliti dedicati del GPS.

accedendo ai dati di un server ottiene la lista dei satelliti visibili avvalendosi del fatto che le celle di telefonia mobile hanno necessariamente una posizione fissa[10].

Il sistema GPS ha un errore di precisione che rientra nell'ordine dei 5-10 metri e risente dei fattori atmosferici che degradano il segnale prima che raggiunga i ricevitori. Inoltre perde notevolmente di precisione all'interno di edifici dove il segnale diventa più debole e incontra maggiori ostacoli.

Capitolo 2

Analisi

2.1 Requisiti

L'idea è quella di realizzare un sistema distribuito con lo scopo di supportare il monitoraggio dello stato e delle condizioni di salute di un gruppo di atleti in fase di allenamento con l'obiettivo di identificare e segnalare tempestivamente eventuali circostanze critiche e d'emergenza. Strutturalmente, il sistema è suddivisibile in due macroparti: quella relativa al sistema wearable per ciascun atleta e quella relativa al sistema della centrale di controllo.

2.1.1 PARTE A - Sistema Wearable

Si tratta di un sistema che si basa sull'utilizzo di una coppia di dispositivi - rilevatore di parametri biomedici (es. pulsossimetro WristOx2[2]) e smartphone - capaci di dialogare tra loro con lo scopo di monitorare costantemente lo stato dell'atleta e di inviare informazioni alla centrale di controllo. Nello specifico, al sistema è richiesto di:

- Determinare la posizione geografica dell'utente - all'interno di una certa area predefinita - e di inviarla costantemente alla centrale di controllo avvalendosi di una infrastruttura di comunicazione (es. basata su comunicazione WiFi)
- Acquisire dal pulsossimetro i parametri attuali relativi allo stato di salute dell'atleta che lo indossa e inviarli alla centrale di controllo qualora sussista (1) una condizione di emergenza - per parametri fuori soglia - oppure (2) una variazione sostanziale dello stato tale per cui debbano essere aggiornati i dati in centrale operativa senza necessariamente scatenare un allarme.

- Utilizzare i sensori di movimento di cui è provvisto lo smartphone (accelerometro, giroscopio, segnale GPS, ...) per determinare lo stato di attività dell'atleta (sta correndo, è fermo, è in piedi, è seduto/sdraiato) ed agire conseguentemente per inviare un eventuale allarme alla centrale operativa.
- Riceve dalla centrale operativa eventuali feedback sull'attività corrente o richieste di conferma sul proprio stato attualmente rilevato dal sistema (es. il sistema comunica alla centrale che l'atleta è fermo, la centrale può innescare un meccanismo che contatti l'atleta per capire se si tratta di un'anomalia oppure se è necessario inviare soccorsi).

2.1.2 PARTE B - Sistema per la Centrale di Controllo

Il sistema deve consentire il monitoraggio continuo dello stato degli atleti attivi attraverso le informazioni ricevute dal sistema wearable. Nello specifico, il sistema deve:

- Consentire la visualizzazione delle posizioni (real-time) degli atleti su una mappa (eventualmente precaricata, in relazione alla sessione di allenamento corrente) tenendo eventualmente traccia del percorso seguito.
- Associare a ciascun atleta un colore in funzione dello stato di salute in modo tale che osservando la mappa sia possibile identificare immediatamente lo stato generale di tutti gli atleti attivi.
- Permette di interrogare il sistema relativamente ad uno specifico atleta al fine di ottenere i valori attuali dei parametri vitali e, in generale, lo stato dell'atleta.
- Identificare autonomamente situazioni d'emergenza e conseguentemente scatenare allarmi determinati da specifici algoritmi (che considerano il valore dei parametri vitali combinati alla posizione assunta dall'utente).
- Consentire un accesso contemporaneo alla centrale di controllo da più postazioni distinte (es. postazione fissa in infermeria e postazione mobile).

2.1.3 Requisiti non funzionali

Entrambe le parti, devono essere progettate secondi i canoni dell'ingegneria del software al fine di garantire requisiti di robustezza ed estendibilità al prototipo, nonché di riusabilità dei singoli componenti interni.

Da un punto di vista tecnologico, per la PARTE A, il riferimento è Google Android (sfruttando la tecnologia Bluetooth per comunicare con il pulsossimetro). Per la PARTE B, l'idea è quella di realizzare un'applicazione Web (o comunque, riadattare un'applicazione pre-esistente alle specifiche esigenze) attraverso HTML5, Javascript e JQuery.

Estensioni possibili:

- PARTE A
 - consentire l'identificazione di ciascun atleta in modo rapido.
- PARTE B
 - memorizzare opportunamente i dati ricevuti per analisi a posteriori
 - invio messaggi agli atleti.

2.2 Analisi dei Requisiti

Da quanto emerge dai requisiti il sistema che viene chiesto di implementare è costituito da due macro-parti principali, la centrale di controllo e il sistema wearable. Quest'ultimo è composto da due dispositivi distinti, un rilevatore di parametri biomedici (es. pulsossimetro WristOx2) e uno smartphone. Queste tre entità principali, centrale di controllo, smartphone e pulsossimetro, costituiscono nel loro insieme il sistema finale, dove ognuna di esse fornisce particolari funzionalità aggiuntive utili al soddisfacimento dei requisiti proposti. Le funzionalità delle tre componenti principali vengono riassunte di seguito:

- *Centrale di controllo*, attraverso le informazioni ricevute dal dispositivo mobile la centrale di controllo deve consentire il monitoraggio degli atleti, al fine di identificare e segnalare tempestivamente eventuali circostanze d'emergenza. Le funzionalità richieste sono chiaramente espresse dai requisiti e riportate di seguito:
 - Visualizzazione delle posizioni (real-time) degli atleti su una mappa tenendo eventualmente traccia del percorso seguito.
 - Associazione di un indicatore di stato a ciascun atleta.
 - Visualizzazione per ciascun atleta dei valori attuali dei parametri vitali.
 - La possibilità di scatenare allarmi in situazioni di emergenza.
 - Consentire l'accesso ai dati degli atleti a eventuali postazioni mobili.

- *Smartphone*, ruolo chiave del dispositivo mobile è quello di acquisire i dati utili al monitoraggio degli atleti, attraverso i propri sensori e l'aiuto del pulsossimetro, per inviarli alla centrale di controllo. Inoltre deve ricevere dalla centrale operativa eventuali feedback sull'attività corrente o richieste di conferma sul proprio stato.
- *Pulsossimetro*, il compito del pulsossimetro è quello di ricavare i parametri dell'atleta che lo indossa ed inviarli allo smartphone con il quale deve comunicare direttamente.

Per maggiore chiarezza viene mostrato nei seguenti diagrammi di interazione come le tre entità interagiscono tra loro durante il ciclo di vita del sistema, prendendo in considerazione due scenari alternativi, a seconda di quale componente supporterà la funzionalità di analisi dei dati, al fine di individuare situazione di pericolo. Il primo diagramma riportato in figura 2.1 mostra il caso in cui sia esclusivamente la centrale operativa a implementare questa funzione.

In figura 2.2 viene invece riportato il caso in cui una fase preliminare di analisi dei dati e rilevamento di situazioni di pericolo venga implementata sul dispositivo Android.

I requisiti specificano anche i dati minimi ed essenziali che devono essere collezionati da smartphone e pulsossimetro, quindi inviati alla centrale di controllo. Le informazioni richieste sono le seguenti:

- La posizione GPS, per poter localizzare l'atleta.
- I parametri biomedici ricavati dal pulsossimetro, quali saturazione del sangue e frequenza cardiaca.
- I dati utili per poter svolgere l'attività di Activity Recognition (riconoscimento dello stato di attività dell'atleta (sta correndo, è fermo, è in piedi, è seduto/sdraiato)).

2.2.1 Casi d'uso

I casi d'uso del sistema vengono riportati nella figura 2.3, come si può notare nell'immagine, il sistema, grazie al monitoraggio dei parametri vitali e degli altri dati riguardanti lo stato dell'atleta, permette di visualizzare queste informazioni all'operatore della centrale di controllo e a quelli di altre eventuali postazioni mobili. Inoltre fornisce la funzionalità di assistenza in casi di

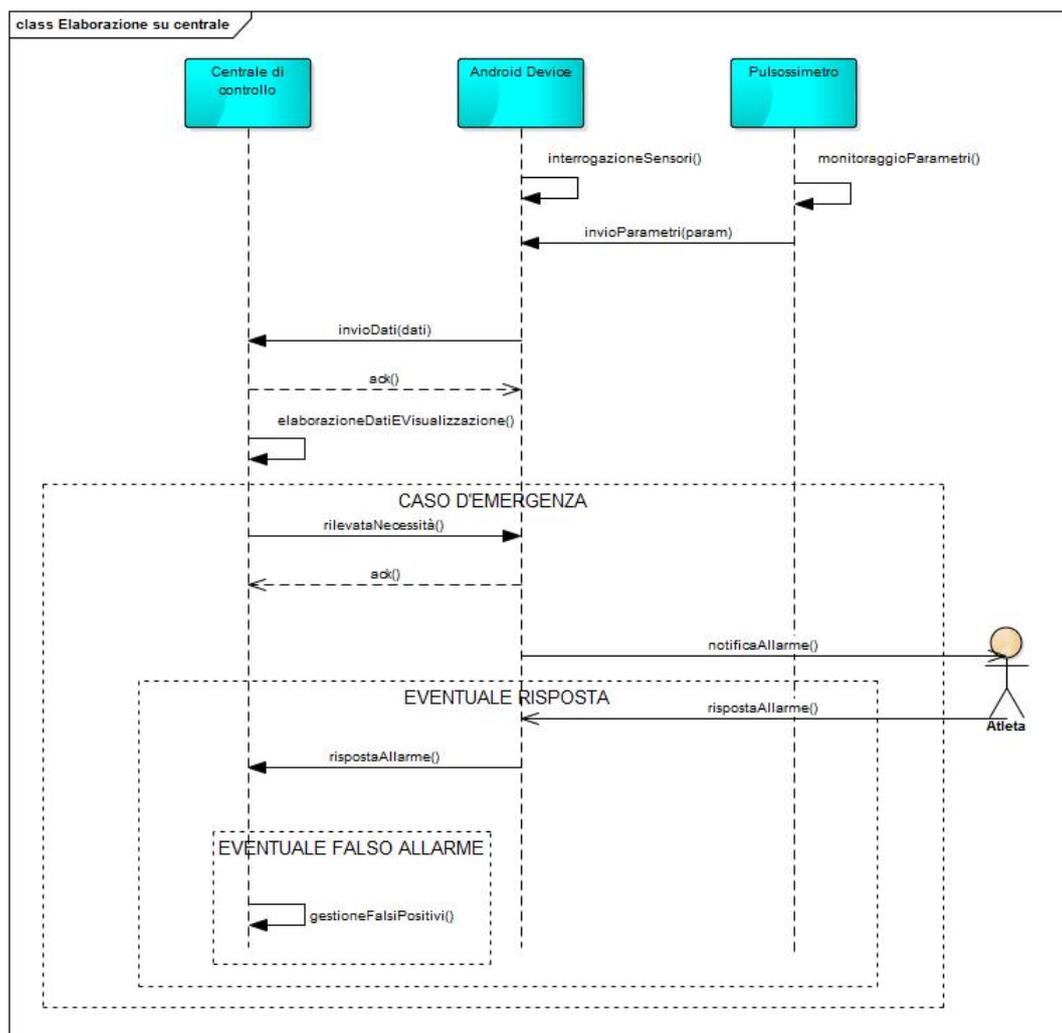


Figura 2.1: Rilevazione ad opera della centrale di controllo.

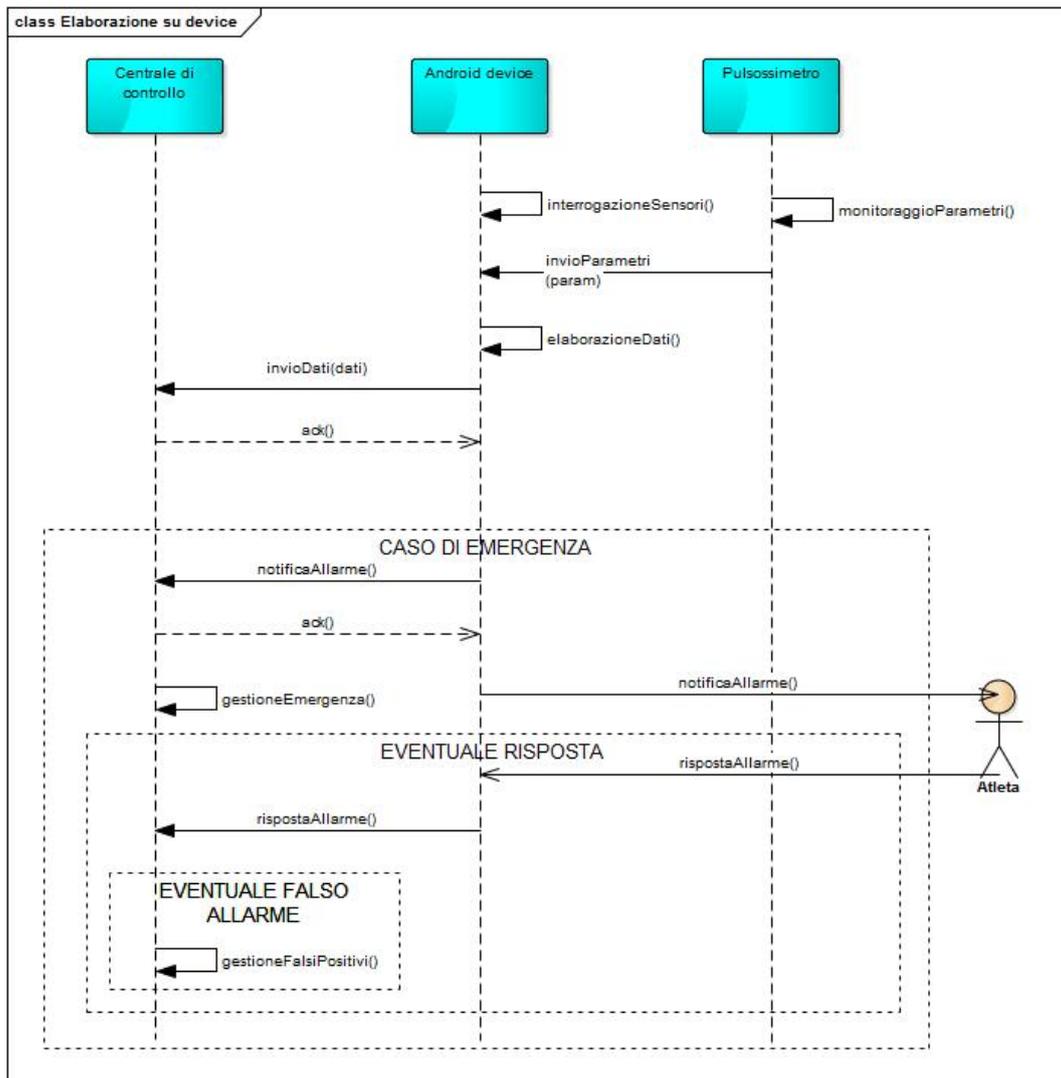


Figura 2.2: Rilevazione ad opera del dispositivo mobile.

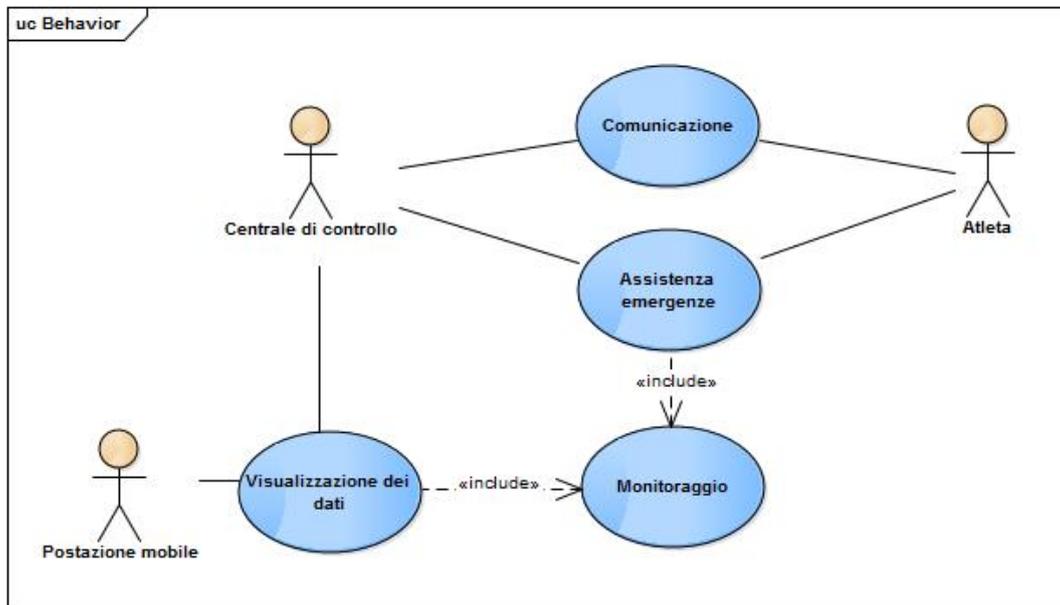


Figura 2.3: Diagramma dei casi d'uso.

emergenza, scatenando un allarme nel caso sia richiesto. Infine permette una comunicazione tra l'atleta e l'operatore della centrale di controllo.

2.2.2 Scenari

I seguenti scenari descrivono le funzionalità del sistema:

Scenario: Comunicazione

| | |
|--------------------------|---|
| ID (Nome) | Comunicazione. |
| Descrizione | Permette lo scambio di informazioni tra atleta e operatore della centrale di controllo. |
| Attori principali | Atleta, Operatore della centrale di controllo. |
| Condizioni Preesistenti | Non vi siano malfunzionamenti nel sistema. |
| Corso principale | I messaggi che atleta e centrale di controllo si vogliono scambiare devono essere recapitati al destinatario. |
| Condizioni di successo | I messaggi arrivano a destinazione. |
| Condizioni di fallimento | I messaggi non vengono recapitati. |

Scenario: Visualizzazione dei dati

| | |
|--------------------------|---|
| ID (Nome) | Visualizzazione dei dati. |
| Descrizione | Mostra i dati richiesti. |
| Attori principali | Operatore della centrale di controllo, Operatore della postazione mobile. |
| Condizioni Preesistenti | Non vi siano malfunzionamenti nel sistema. |
| Corso principale | Ogni volta che i dati in memoria del sistema riguardo gli atleti vengono modificati, questi devono essere visualizzati. |
| Condizioni di successo | I valori vengono mostrati correttamente. |
| Condizioni di fallimento | I valori non vengono visualizzati. |

Scenario: Monitoraggio

| | |
|--------------------------|--|
| ID (Nome) | Monitoraggio. |
| Descrizione | Attraverso i sensori e il pulsossimetro acquisiti i dati riguardanti gli atleti. |
| Condizioni Preesistenti | Non vi siano malfunzionamenti nel sistema. |
| Corso principale | Colleziona i dati ricevuti. |
| Condizioni di successo | I dati ottenuti vengono ricevuti correttamente dal destinatario. |
| Condizioni di fallimento | I dati non vengono ricevuti. |

Scenario: Assistenza emergenze

| | |
|--------------------------|--|
| ID (Nome) | Assistenza emergenze. |
| Descrizione | In casi di allarme la centrale di controllo deve avviare un protocollo di soccorso. |
| Attori principali | Operatore della centrale di controllo, Atleta. |
| Condizioni Preesistenti | Non vi siano malfunzionamenti nel sistema. |
| Corso principale | Quando i parametri vitali acquisiti dai sensori non soddisfano degli intervalli specificati viene generato un allarme. |
| Condizioni di successo | L'allarme viene generato nei casi richiesti e in maniera tempestiva. |
| Condizioni di fallimento | Non viene generato alcun allarme. |

2.3 Analisi del Problema

In accordo con quanto definito nell'analisi dei requisiti, si procede definendo le entità principali individuate, descrivendo attraverso i seguenti diagrammi, la loro struttura e il loro comportamento. Prima di procedere bisogna però decidere quale soluzione adottare tra le due alternative presentate in precedenza, vengono quindi elencate di seguito le principali differenze tra l'utilizzo dei due approcci:

- Rilevazione preliminare ad opera del dispositivo mobile:
 - minore frequenza di invio dei dati, con conseguente traffico di rete ridotto.
 - possibilità di rilevare situazioni critiche anche in assenza di rete.
 - maggior consumo di batteria.
- Rilevazione esclusiva ad opera della centrale di controllo:
 - maggior carico di lavoro.
 - potenza di calcolo maggiore.

Da quanto emerge da questa valutazione conviene implementare questa funzionalità di elaborazione dei dati dei sensori e del pulsossimetro sul dispositivo mobile, la centrale di controllo eseguirà comunque un'analisi collettiva dei dati ma ad entrambi verrà permesso di generare allarmi.

Un problema fondamentale, che si evidenzia dai requisiti, riguarda la necessità di fornire un servizio di comunicazione via rete, per permettere alle entità centrale operativa e dispositivo mobile di scambiarsi messaggi. In particolare, questo servizio dovrà fornire alla centrale operativa la possibilità di visualizzare i dati raccolti dallo smartphone, gli allarmi generati ed eventualmente le loro richieste di annullamento, oltre che dei messaggi testuali per permettere la comunicazione tra le parti. Inoltre il servizio dovrà essere il più possibile leggero, gestibile e scalabile, oltre al fatto che dovrà permettere l'interoperabilità tra diverse applicazioni software su diverse piattaforme hardware.

2.3.1 Struttura

Per quanto riguarda i diagrammi di struttura riportiamo nella figura 2.4 i dati scambiati tra la centrale operativa e lo smartphone, in base a quanto

In particolare l'interfaccia *IMessage* viene estesa dai quattro tipi di messaggi che vengono scambiati via rete tra le due entità:

- *DataMessage*, questo messaggio contiene i dati utili al monitoraggio dell'atleta.
- *AlarmMessage*, rappresenta una notifica di allarme e, oltre ai dati inviati regolarmente, fornisce anche alcune informazioni aggiuntive sull'evento che ha generato l'allarme.
- *DenyAlarm*, comunica l'annullamento della notifica di allarme.
- *TextMessage*, contiene un messaggio di testo, per la comunicazione tra utente della centrale di controllo e atleta.

2.3.2 Interazione

L'immagine seguente, figura 2.5 , mostra nello specifico l'interazione tra i componenti del dispositivo mobile durante il suo ciclo di vita.

Viene mostrata, nella figura 2.6 e nella figura 2.7, l'interazione tra i componenti della centrale di controllo mentre svolge le funzionalità di comunicazione e di monitoraggio:

2.3.3 Dispositivo WristOx2

Prima di proseguire con il Progetto occorre definire quale dispositivo utilizzare per raccogliere i parametri biomedicali dell'atleta. In seguito ad un'attenta analisi delle caratteristiche e delle peculiarità del dispositivo WristOx2, suggerito nei requisiti, si decide di proseguire utilizzando il suddetto pulsossimetro. Questo dispositivo comunica i parametri rilevati, avvalendosi del protocollo *Bluetooth*. “Bluetooth is a wireless technology standard for exchanging data over short distances (using short-wavelength UHF radio waves in the ISM band from 2.4 to 2.485 GHz) from fixed and mobile devices, and building personal area networks (PANs). Invented by telecom vendor Ericsson in 1994, it was originally conceived as a wireless alternative to RS-232 data cables. It can connect several devices, overcoming problems of synchronization” fonte [1].

La documentazione in dotazione con il dispositivo, reperibile a [2], descrive gli aspetti fondamentali del dispositivo WristOx2, in particolare, come si può osservare in figura 2.8, una volta che viene accoppiato con un altro dispositivo (nel caso specifico lo smartphone), esso rimane in attesa di una richiesta di connessione, una volta stabilita, procede inviando periodicamente i dati raccolti, fino a quando non verrà effettuata una richiesta di disconnessione.

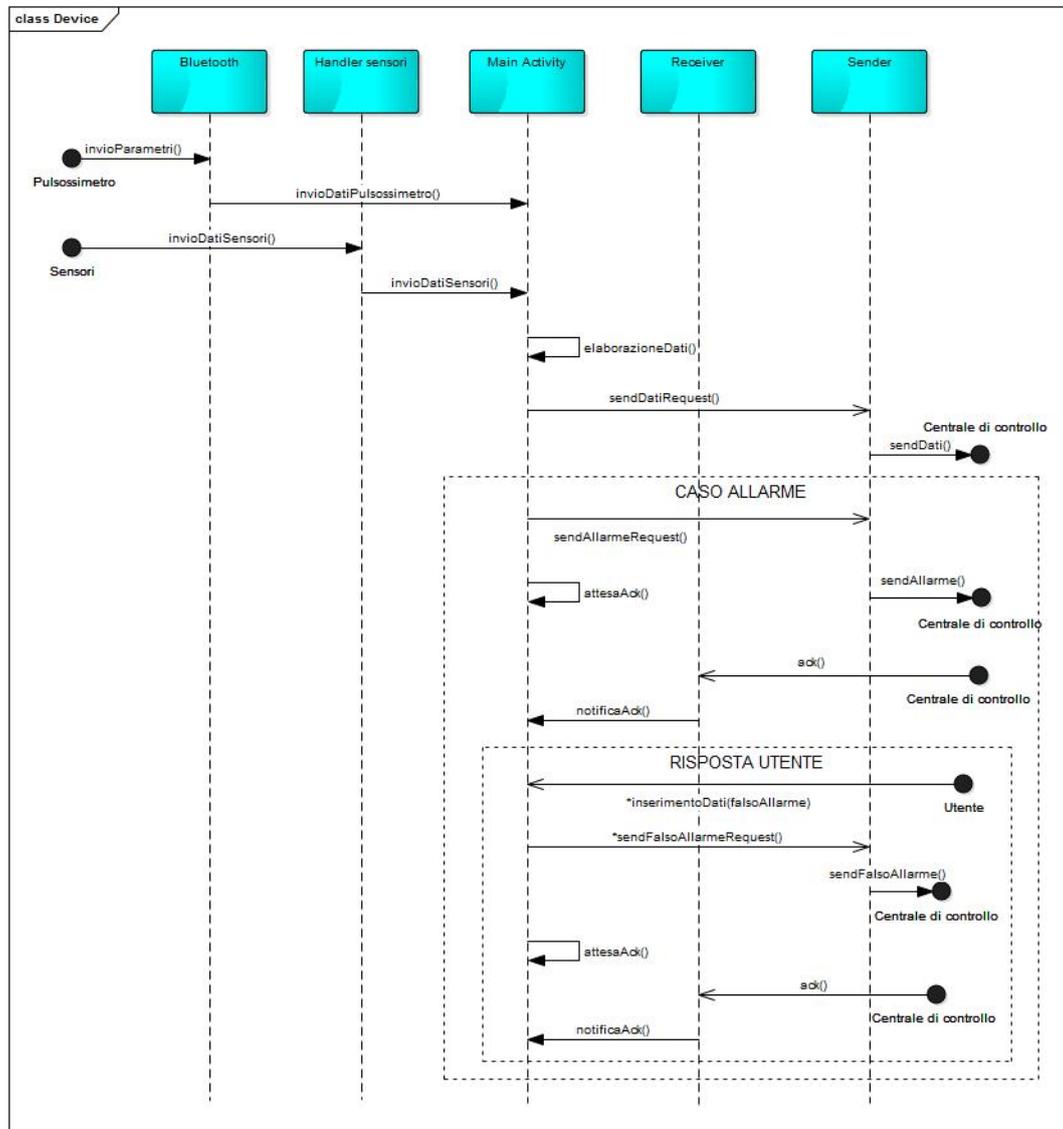


Figura 2.5: Interazione tra i componenti del dispositivo mobile.

In figura 2.9 viene mostrata la struttura dei pacchetti dati inviati dal WristOx2, ogni pacchetto è composto da 25 frame ognuno dei quali contiene 5 byte. Tre pacchetti, quindi 75 frame, vengono trasmessi ogni secondo.

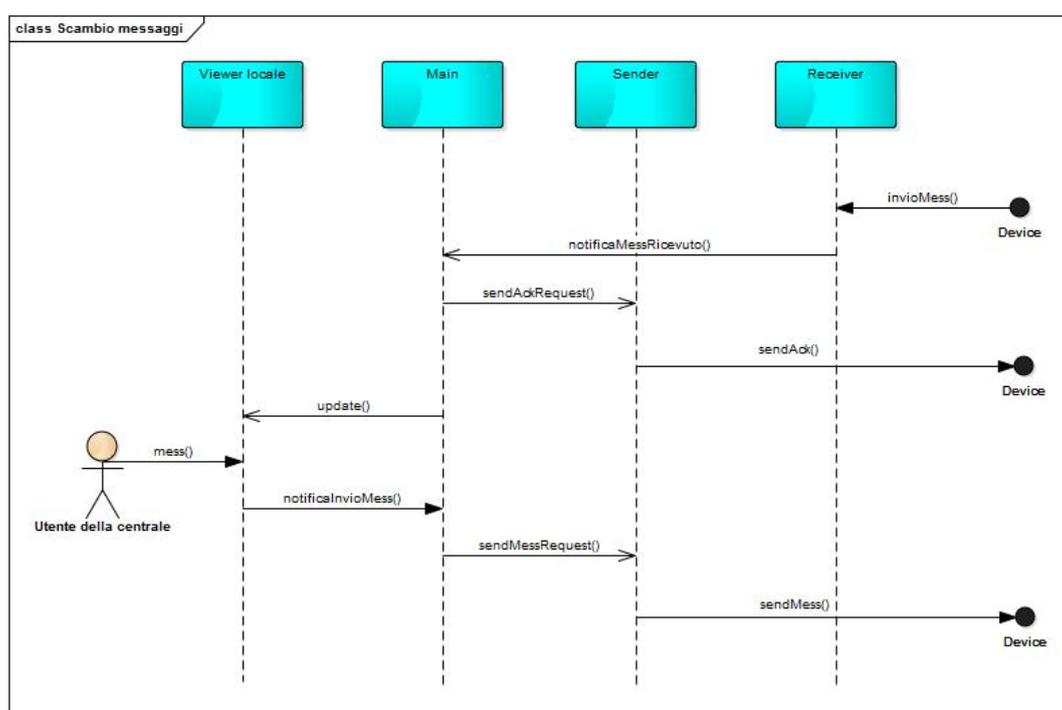


Figura 2.6: Interazione tra i componenti della centrale di controllo durante la comunicazione.

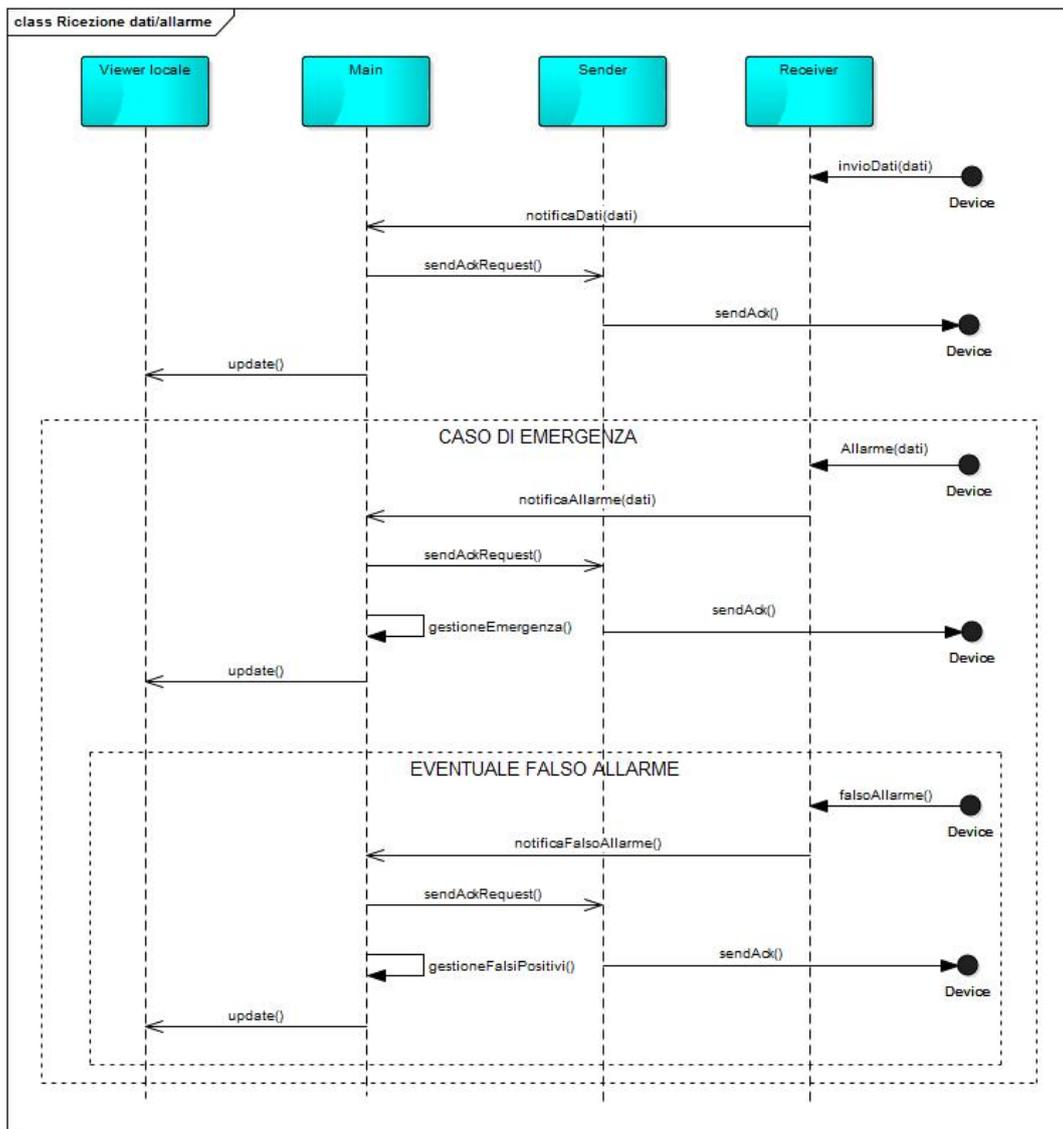


Figura 2.7: Interazione tra i componenti della centrale di controllo mentre svolge le sue funzionalità di monitoraggio.

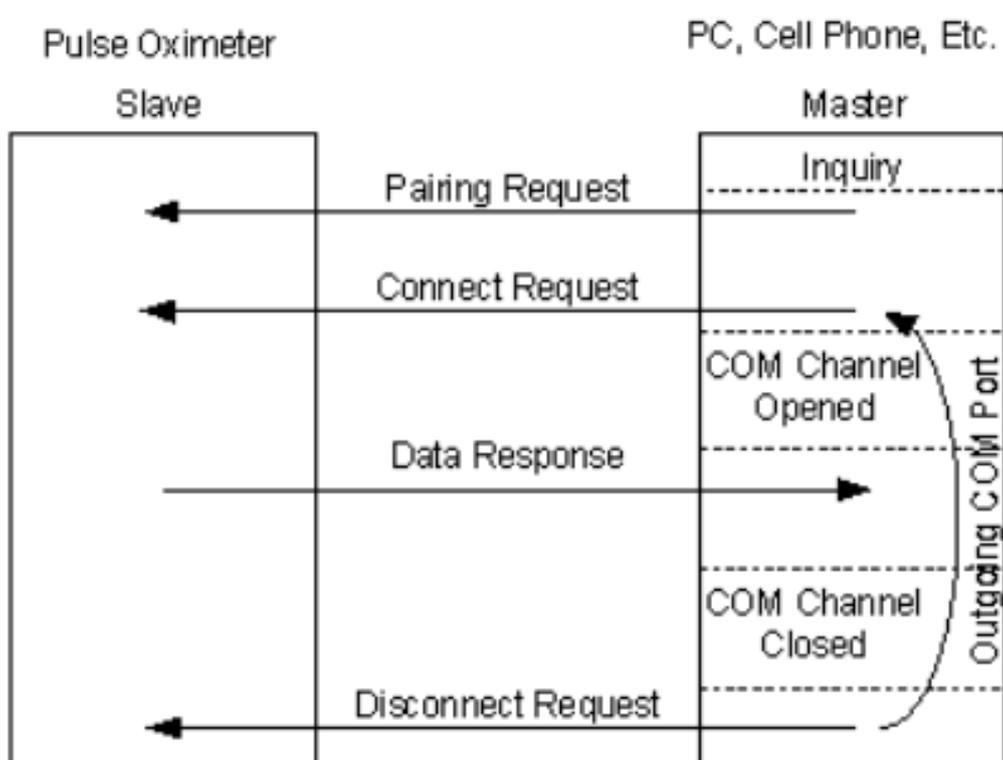


Figura 2.8: Comunicazione con il pulsossimetro WristOx2, fonte [3].

| Packet | Frame | | | | |
|--------|--------|--------|--------|-----------------------|--------|
| | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 |
| 1 | 01 | STATUS | PLETH | HR MSB | CHK |
| 2 | 01 | STATUS | PLETH | HR LSB | CHK |
| 3 | 01 | STATUS | PLETH | SpO ₂ | CHK |
| 4 | 01 | STATUS | PLETH | SREV | CHK |
| 5 | 01 | STATUS | PLETH | reserved | CHK |
| 6 | 01 | STATUS | PLETH | TMR MSB | CHK |
| 7 | 01 | STATUS | PLETH | TMR LSB | CHK |
| 8 | 01 | STATUS | PLETH | STAT2 | CHK |
| 9 | 01 | STATUS | PLETH | SpO ₂ -D | CHK |
| 10 | 01 | STATUS | PLETH | SpO ₂ Fast | CHK |
| 11 | 01 | STATUS | PLETH | SpO ₂ B-B | CHK |
| 12 | 01 | STATUS | PLETH | reserved | CHK |
| 13 | 01 | STATUS | PLETH | reserved | CHK |
| 14 | 01 | STATUS | PLETH | E-HR MSB | CHK |
| 15 | 01 | STATUS | PLETH | E-HR LSB | CHK |
| 16 | 01 | STATUS | PLETH | E-SpO ₂ | CHK |
| 17 | 01 | STATUS | PLETH | E-SpO ₂ -D | CHK |
| 18 | 01 | STATUS | PLETH | reserved | CHK |
| 19 | 01 | STATUS | PLETH | reserved | CHK |
| 20 | 01 | STATUS | PLETH | HR-D MSB | CHK |
| 21 | 01 | STATUS | PLETH | HR-D LSB | CHK |
| 22 | 01 | STATUS | PLETH | E-HR-D MSB | CHK |
| 23 | 01 | STATUS | PLETH | E-HR-D LSB | CHK |
| 24 | 01 | STATUS | PLETH | reserved | CHK |
| 25 | 01 | STATUS | PLETH | reserved | CHK |

Figura 2.9: Struttura dei pacchetti dati inviati dal WristOx2, fonte [3].

Capitolo 3

Progetto

3.1 Componente Android

Nelle seguente sezione viene descritto il sistema definitivo, sviluppato per la *PARTE A*, in base a quanto definito nelle fasi precedenti di analisi. Inoltre, nella figura 3.1, viene riassunto il comportamento dell'applicazione Android per fornire un'idea chiara di quanto verrà discusso nei seguenti paragrafi.

3.1.1 Logic Architecture

Nella figura 3.2 viene mostrato il modello del dominio del sistema finale, mostrando i vari componenti e la loro organizzazione.

3.1.2 Struttura

In questo paragrafo viene mostrata, nello specifico, la struttura finale delle componenti dell'applicazione che verrà realizzata. Trovandoci in ambiente *Android* dovremo inserire due *activity*, la prima, *LoginActivity*, sarà mostrata all'avvio dell'applicazione, dove l'utente potrà effettuare l'autenticazione delle proprie credenziali. La seconda sarà *MainActivity*, che rappresenta l'*activity* principale del sistema, essa è incaricata di tenere aggiornati i valori mostrati all'utente e gestire le varie componenti del sistema, utili alla funzione di monitoraggio, invio e ricezioni dei dati. La struttura del package *it.unibo.athletesupp.activity*, contenente le *activity*, viene mostrata nella figura 3.3.

Un'altra serie di componenti fondamentali dell'applicazione sono quelle che permettono la comunicazione con l'esterno, quindi via rete e via Bluetooth.

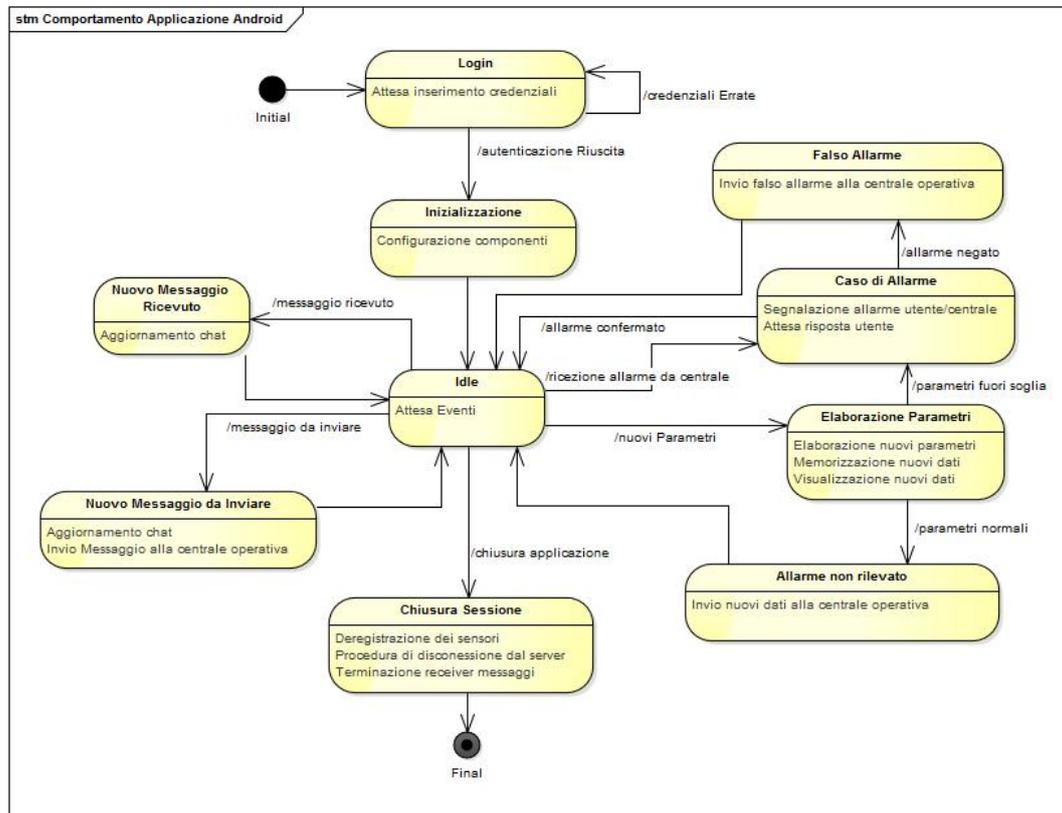


Figura 3.1: Comportamento del sistema Android.

Queste sono contenute all'interno del package *it.unibo.athletesupp.communication* mostrato in figura 3.4. Per quanto riguarda la comunicazione via rete abbiamo quattro oggetti:

- *AuthenticationTask*, *AsyncTask* incaricato di effettuare l'autenticazione delle credenziali inserite dall'utente, per permettere all'applicazione di accedere ai dati del server, in modo da poter ricevere e inviare messaggi.
- *LogoutTask*, *AsyncTask* pensato per disconnettere l'utente, informando così il server che l'applicazione cesserà di inviare e ricevere messaggi, con lo scopo di chiudere in modo completo e sicuro la sessione di lavoro.
- *Receiver*, è il *Thread* incaricato di ricevere i messaggi in arrivo dalla Centrale Operativa.
- *Sender*, *AsyncTask* che verrà avviato ogni qualvolta si presenterà la necessità di inviare un messaggio alla Centrale Operativa.

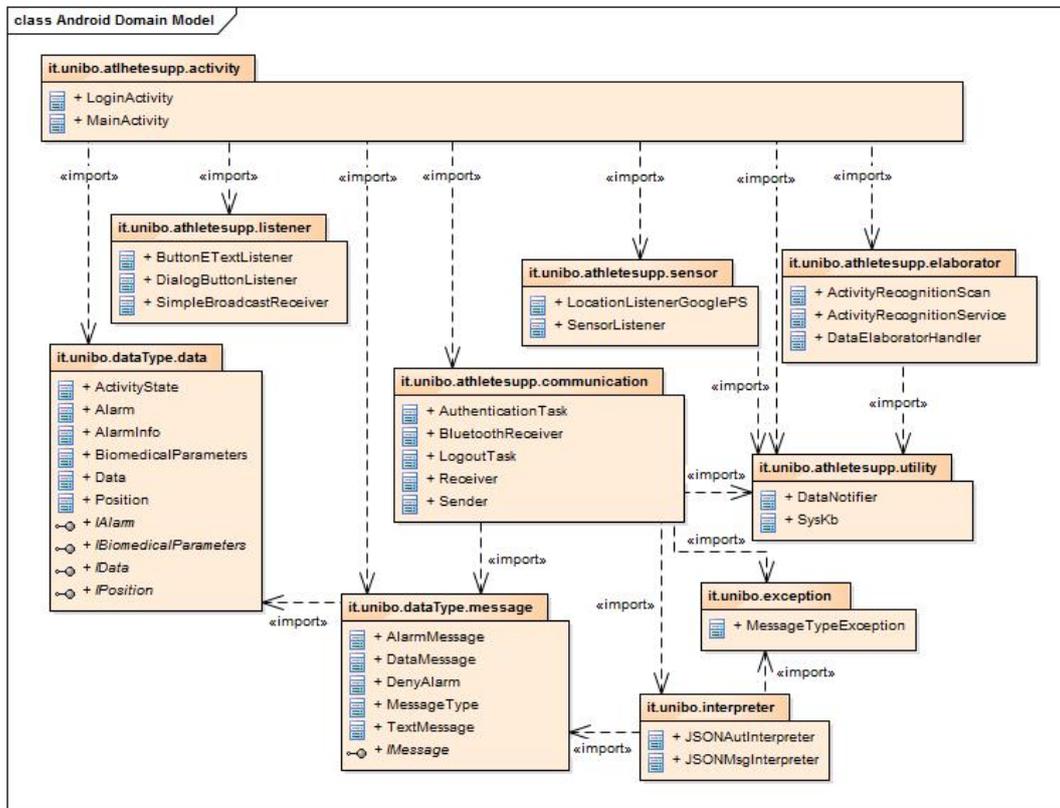


Figura 3.2: Struttura del modello del dominio del sistema finale Android.

BluetoothReceiver possiede invece il compito di ricevere, tramite Bluetooth, i parametri biomedici dal pulsossimetro e di comunicarli alla *MainActivity*.

La figura 3.5 mostra invece la struttura delle entità che fanno parte del package *it.unibo.athletesupp.elaborator*, esso contiene le componenti incaricate di svolgere l'elaborazione dei dati raccolti dall'applicazione mobile. In particolare, *ActivityRecognitionService* avrà il compito di analizzare i dati collezionati dai sensori dello smartphone, per fornire la funzionalità di rilevamento dell'attività in corso. Implementato come *Service*, esso sarà avviato dall'oggetto *ActivityRecognitionScan*, il quale avrà anche il compito di connettersi ai servizi di Google, siccome *ActivityRecognitionService* per svolgere il proprio compito dovrà utilizzare una innovativa libreria chiamata *Android Activity Recognition*. Le funzionalità di *Android Activity Recognition* forniscono ai dispositivi Android la capacità di rilevare una serie di attività fisiche compiute dall'utente, come camminare, andare in bicicletta, essere alla guida di un'auto o essere fermi. Quindi per implementare queste funzionalità si dovrà utilizzare queste specifiche API per accedere ai *Google Play Service*.

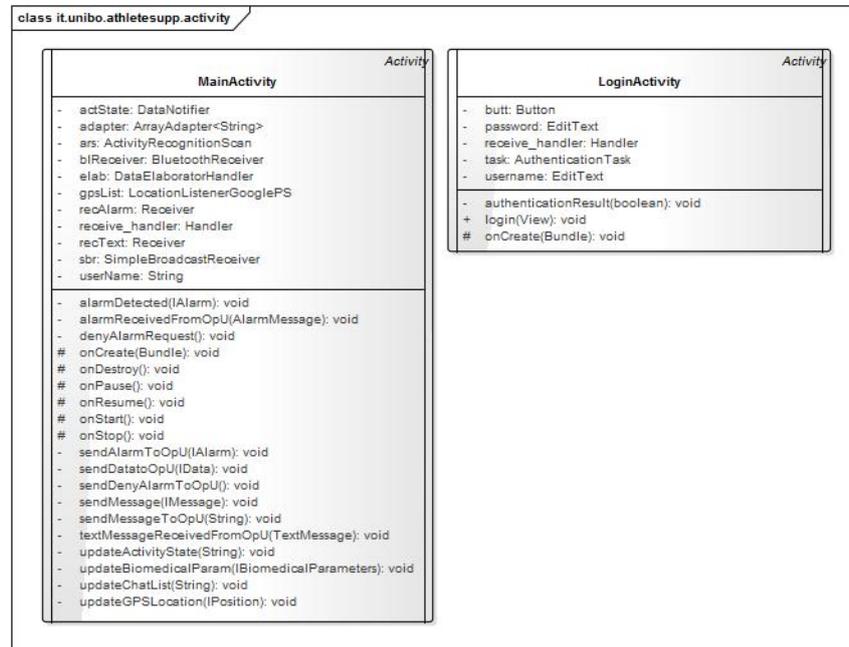


Figura 3.3: Struttura delle activity del sistema.

Un'altra importante entità è *DataElaboratorHandler*, la quale dovrà monitorare tutti i dati raccolti e analizzarli per individuare parametri fuori soglia oppure altri fattori anomali, inoltre dovrà notificare alla *MainActivity* eventuali irregolarità, generando un allarme nel caso in cui i dati raccolti evidenzino una situazione di pericolo per l'atleta.

Per permettere a *DataElaboratorHandler* di svolgere il suo compito di monitoraggio sull'oggetto *DataNotifier*, contenuto nel package *it.unibo.athletesupp.utility*, ci si è ispirati al design pattern *Observer*. I Design Pattern sono soluzioni tecniche a problemi comuni di progettazione del software, essi vengono classificati in tre diverse categorie in base al tipo di problema che cercano di risolvere, in particolare l'*Observer* pattern appartiene alla categoria dei pattern comportamentali, che forniscono soluzioni alle più comuni tipologie di interazione tra gli oggetti. Questo pattern si basa su uno o più oggetti, chiamati osservatori, che vengono registrati per ricevere notifiche generate dall'oggetto osservato quando cambia il proprio stato. Nel caso specifico l'entità *DataNotifier* dovrà tenere memorizzati gli ultimi dati ritornati dai sensori che dovranno essere analizzati e inviati alla centrale operativa, esso verrà aggiornato direttamente dalla *MainActivity* ogni volta che riceverà nuovi valori. Come descritto dal pattern *Observer* l'oggetto *DataElaboratorHandler* verrà notificato da *DataNotifier* ad ogni cambiamento del proprio stato, in questo modo *DataElaboratorHandler* potrà analizzare i nuovi valori, alla ricerca di parametri fuori soglia per comu-

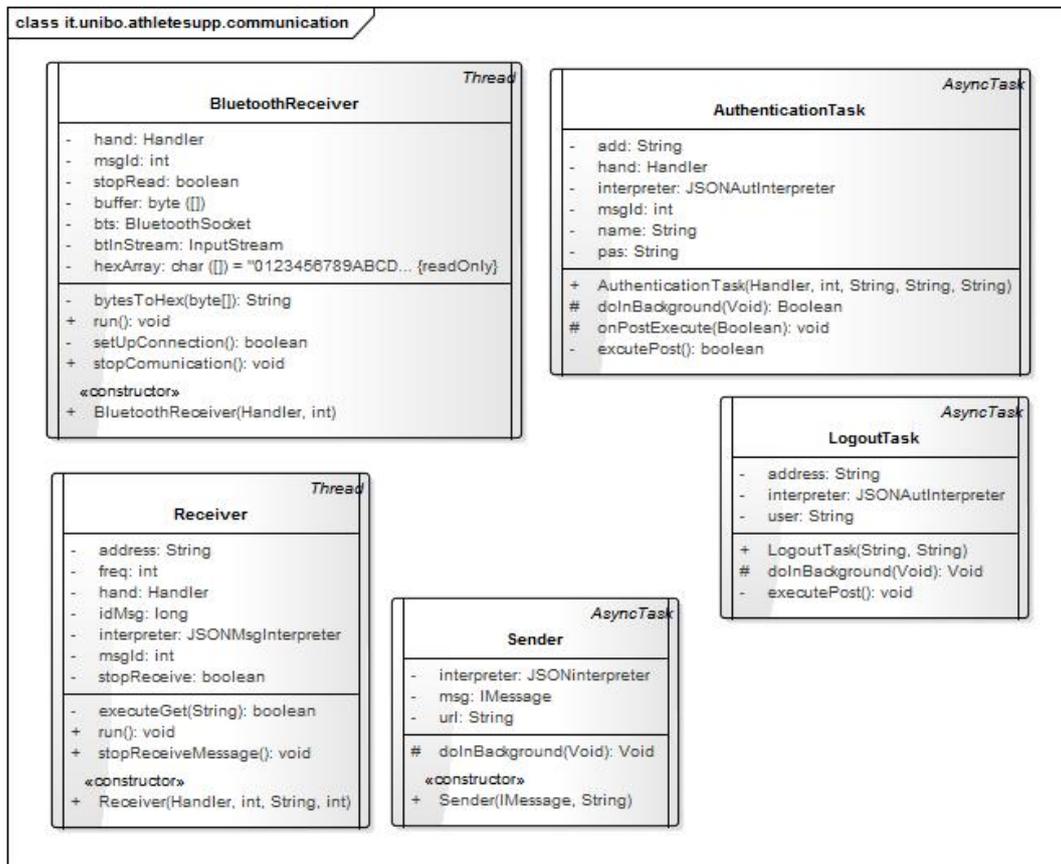


Figura 3.4: Struttura del package it.unibo.athletesupp.communication.

nicarli alla *MainActivity*.

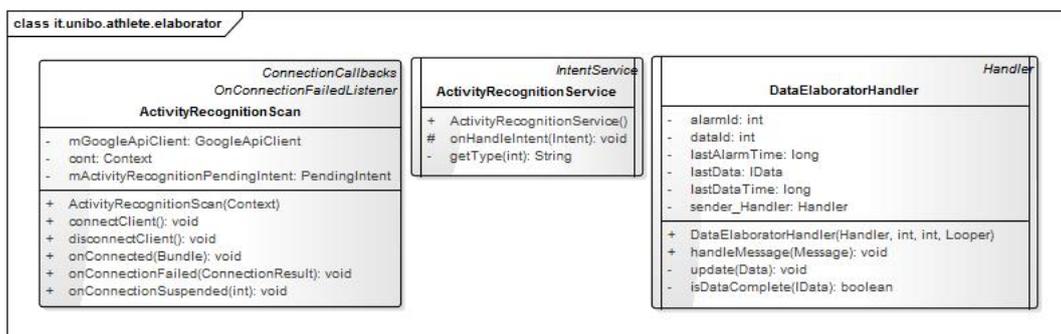


Figura 3.5: Struttura del package it.unibo.athletesupp.elaborator.

3.1.3 Interazione

L'interazione tra i componenti del sistema viene mostrata nella figura 3.6, utilizzando come caso specifico l'arrivo di nuovi dati dal pulsossimetro, come si può osservare, il nuovo messaggio viene notificato alla *MainActivity* inviando un messaggio all'oggetto *Handler*. Quest'ultimo ha il compito di ricevere messaggi da tutte le componenti che sono in attesa dei vari eventi al quale il nostro sistema è interessato, come nuovi messaggi da parte del pulsossimetro piuttosto che nuovi dati rilevati dai sensori oltre che ai messaggi ricevuti via rete e non solo, per poi notificarli alla *MainActivity*. Ogni volta che essa riceve una nuova notifica, viene aggiornata l'interfaccia grafica per mostrare i nuovi valori e aggiornato l'oggetto *DataNotifier*, dove viene mantenuto lo stato attuale dei dati ricevuti dai sensori.

Tutte le volte che lo stato interno dell'oggetto *DataNotifier* cambia, esso invia un messaggio all'oggetto *DataElaboratorHandler*, il quale elabora il nuovo dato ricevuto, considerando tutti i valori attuali del sistema, per poi comunicare il risultato alla *MainActivity*, sempre attraverso l'oggetto *Handler*. A questo punto possono verificarsi due situazioni distinte:

- *L'elaboratore rileva un caso di allarme*: in questo caso la *MainActivity* notifica all'utente la ricezione della situazione di pericolo e invia alla centrale operativa il messaggio di allarme, contenente anche i dati che l'hanno generato.
- *L'elaboratore non rileva alcuna anomalia*: in questo caso la *MainActivity* si limita a inviare i dati elaborati da *DataElaboratorHandler* alla centrale operativa.

Come spiegato in precedenza tutte le componenti del sistema che necessitano di interagire con la *MainActivity* lo fanno inviando messaggi all'oggetto *Handler*, il quale si occupa di notificare i messaggi ricevuti, andando a chiamare il metodo adatto, in base all'identificativo del messaggio. Nel caso in cui si tratti di messaggi che comunicano una variazione dei dati a cui il sistema è interessato, come descritto in precedenza, verrà aggiornato l'oggetto *DataNotifier*, il quale notificherà il cambiamento del proprio stato all'elaboratore di dati *DataElaboratorHandler*, chiamando ricorsivamente il metodo *callObserver*. In figura 3.7 viene riproposto un esempio di interazione tra i componenti, nel caso in cui venga notificata dal sistema un cambiamento di attività, senza però mostrare nuovamente il comportamento di *DataElaboratorHandler* per motivi di ridondanza (in quanto già mostrato nell'immagine 3.6 precedentemente discussa).

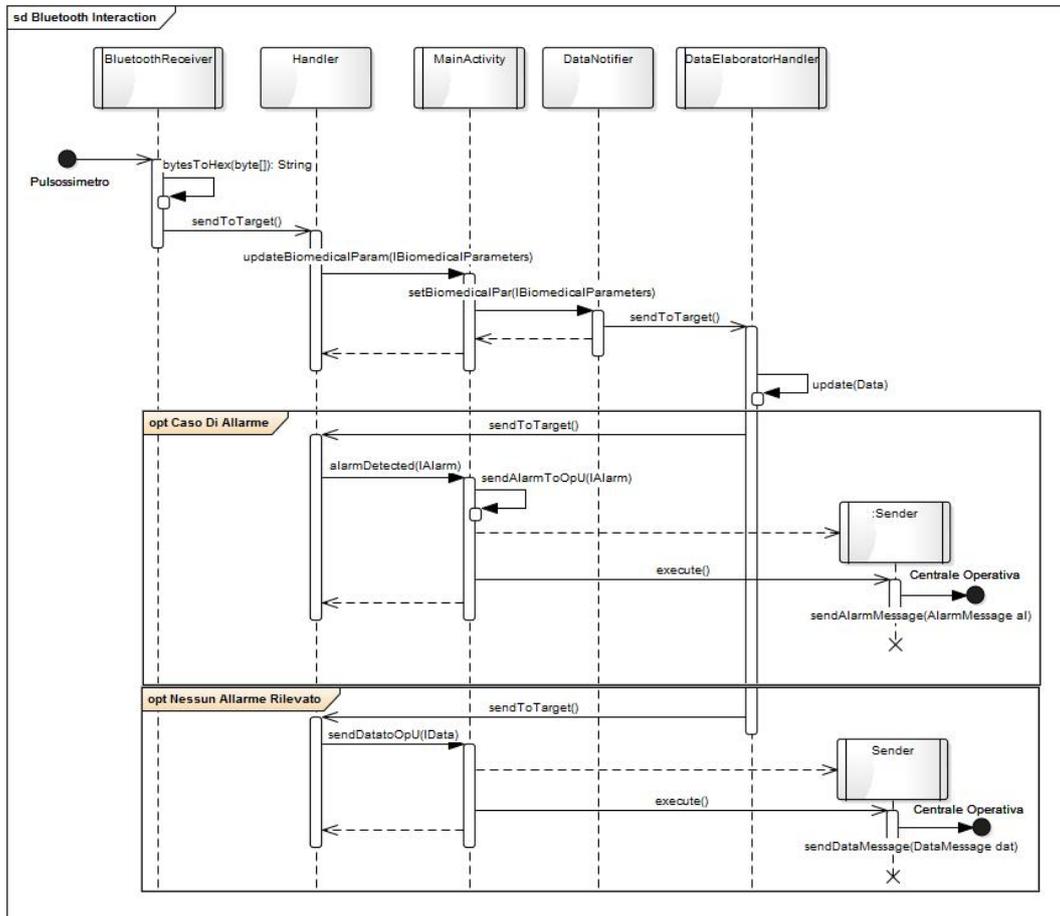


Figura 3.6: Interazione del sistema in seguito a un nuovo messaggio inviato dal Pulsossimetro.

In figura 3.8 viene presentata l'interazione tra le componenti del sistema quando viene ricevuto un nuovo messaggio di testo dalla Centrale Operativa. Come si può notare l'oggetto *Receiver* riceve i messaggi dalla rete e, attraverso l'oggetto *Handler*, notifica alla *MainActivity* l'arrivo del nuovo messaggio, che mostrerà poi all'utente.

3.2 Componente di Comunicazione

In fase di analisi del problema è emersa la necessità di fornire al sistema complessivo un servizio di comunicazione via rete, che permetta lo scambio di messaggi tra più dispositivi Android e che supporti inoltre l'eventuale pre-

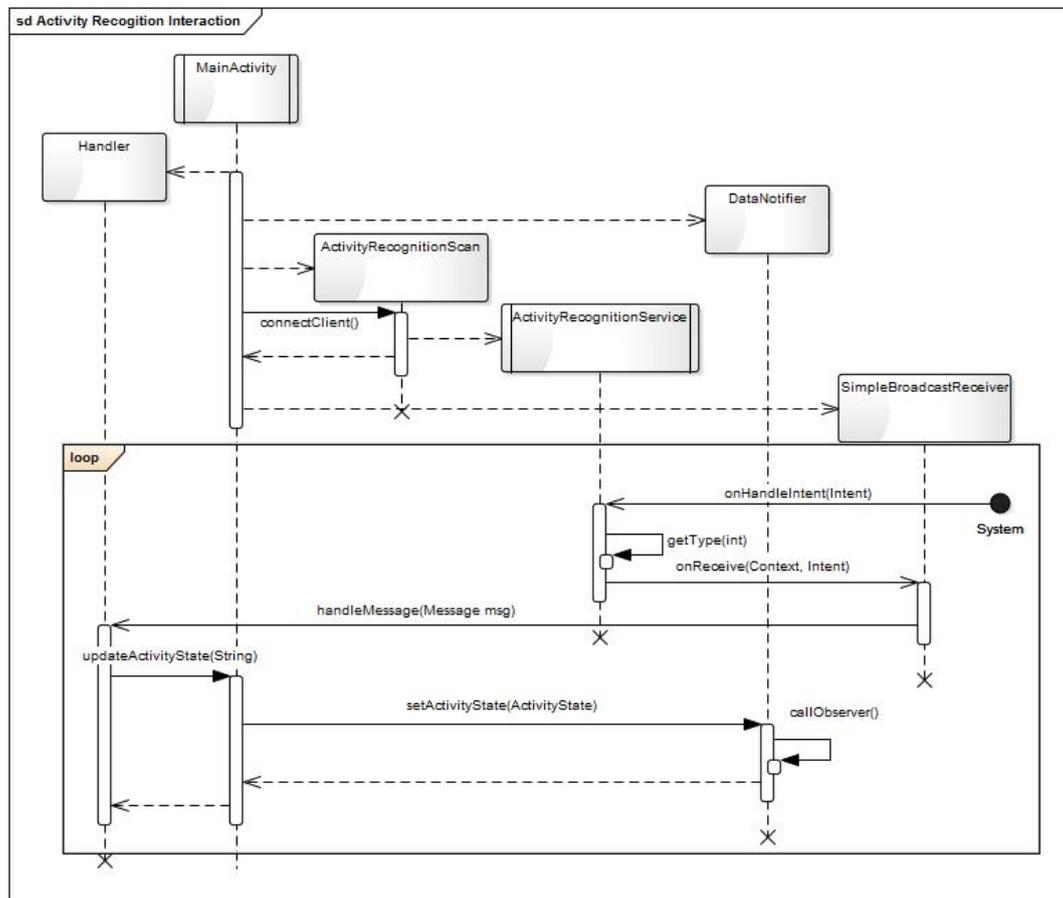


Figura 3.7: Interazione del sistema durante il lavoro di Activity Recognition.

senza di più centrali di controllo. Per fornire, viste le specifiche del caso, un ulteriore livello di astrazione tra le componenti centrale operativa e dispositivo mobile, si decide di sviluppare questo servizio di comunicazione progettando un *Web Service*, in quanto, oltre ad altri notevoli vantaggi, permette, fintanto che l'interfaccia rimane costante, di effettuare modifiche ai servizi rimanendo trasparenti agli utenti.

L'introduzione di un *Web Service*, basato su modello *REST*, consente inoltre, attraverso l'utilizzo del protocollo di comunicazione HTTP (HyperText Transfer Protocol), di utilizzare TCP (Transmission Control Protocol) come protocollo di rete, per rendere affidabile la comunicazione dati tra mittente e destinatario. Lo scambio di messaggi verrà quindi reso disponibile alle entità fondamentali del sistema attraverso l'uso di chiamate *GET* e *POST*, dopo aver opportunamente definito degli specifici Uniform Resource Identifier (URI) per ognuna delle risorse alla quale vogliamo rendere disponibile l'accesso.

Per quanto riguarda il formato dei dati scambiati si decide di adottare JSON

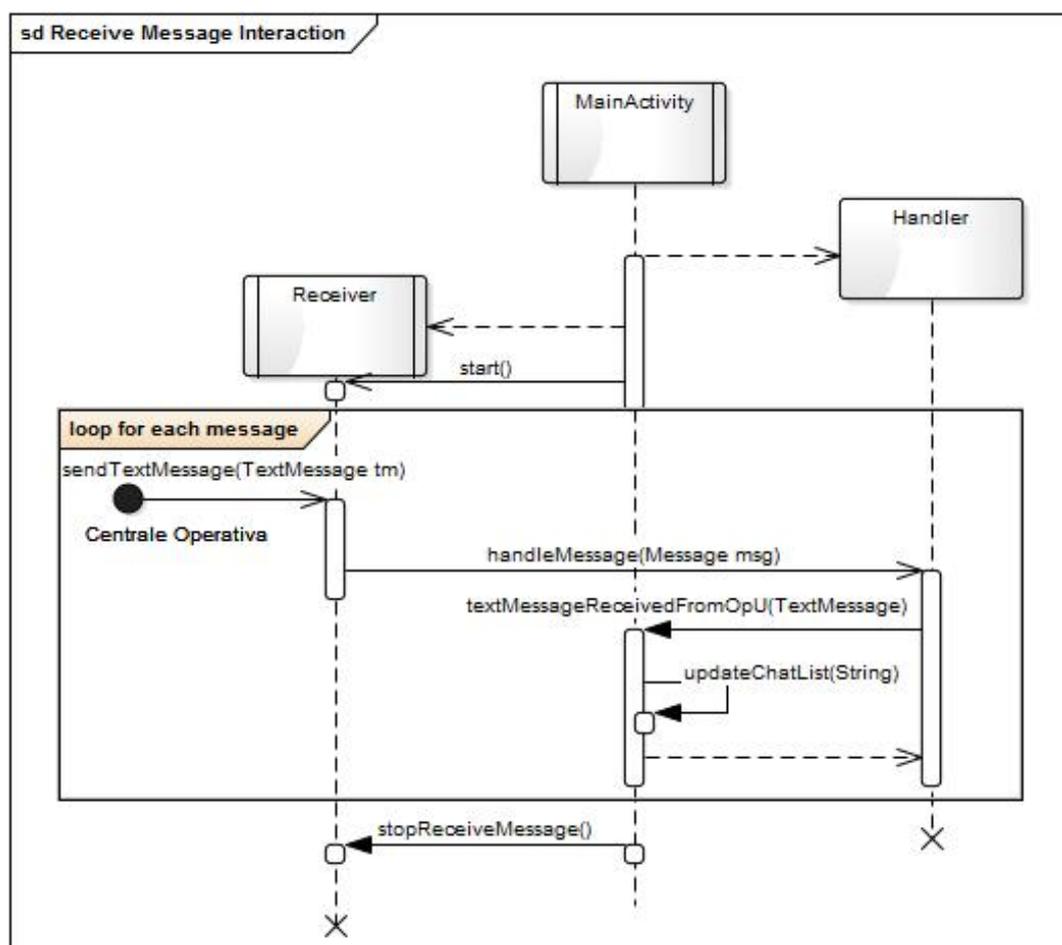


Figura 3.8: Interazione tra le componenti del sistema a fronte dell'arrivo di un nuovo messaggio dalla Centrale Operativa.

(JavaScript Object Notation), in quanto è un formato di testo completamente indipendente dal linguaggio di programmazione, ma che utilizza strutture di dati universali che lo rendono particolarmente adatto per lo scambio dei dati, in applicazioni client-server.

3.2.1 Logic Architecture

Nella figura 3.9 viene mostrato il modello del dominio del *Web Service*, mostrando i vari componenti e la loro organizzazione.

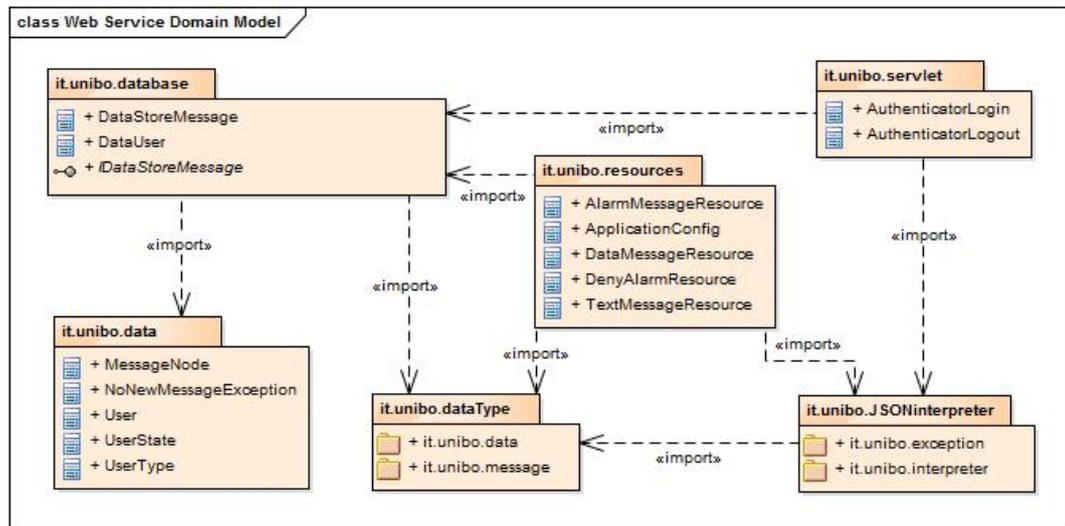


Figura 3.9: Struttura del modello del dominio del sistema finale riguardante la parte di comunicazione.

3.2.2 Struttura

Nella struttura del dominio, figura 3.9, già citata precedentemente, oltre ai dati contenuti nel package *it.unibo.data*, propri del sistema di comunicazione, e ai dati comuni al sistema complessivo, contenuti nel package *it.unibo.dataType*, vengono mostrati gli elementi contenuti nel package *it.unibo.database*, che comprende gli oggetti utili alla gestione dei dati del sistema e alla loro memorizzazione. Infatti l'oggetto *DataStoreMessage*, oltre a permettere un accesso comodo e veloce ai dati inseriti, è anche incaricato di provvedere a un scrittura su file dei suddetti, una volta che sono state compiute il numero adeguato di letture da chi di dovere.

La struttura delle risorse del *Web Service*, basato su modello *REST*, viene presentata nella figura 3.10, dove vengono creati quattro oggetti, uno per ogni tipo di messaggio definito, in modo che ognuno di essi gestisca l'accesso alla corrispettiva tipologia di risorsa che identifica.

Inoltre in figura 3.11, vengono illustrate le due entità *AuthenticatorLogin* e *AuthenticatorLogout*, entrambe *servlet* incaricate di gestire rispettivamente l'operazione di autenticazione e disconnessione degli utenti al sistema.

3.2.3 Interazione

L'interazione tra le componenti del sistema, nel caso di una richiesta di lettura di un eventuale nuovo messaggio, viene mostrata in figura 3.12, nella

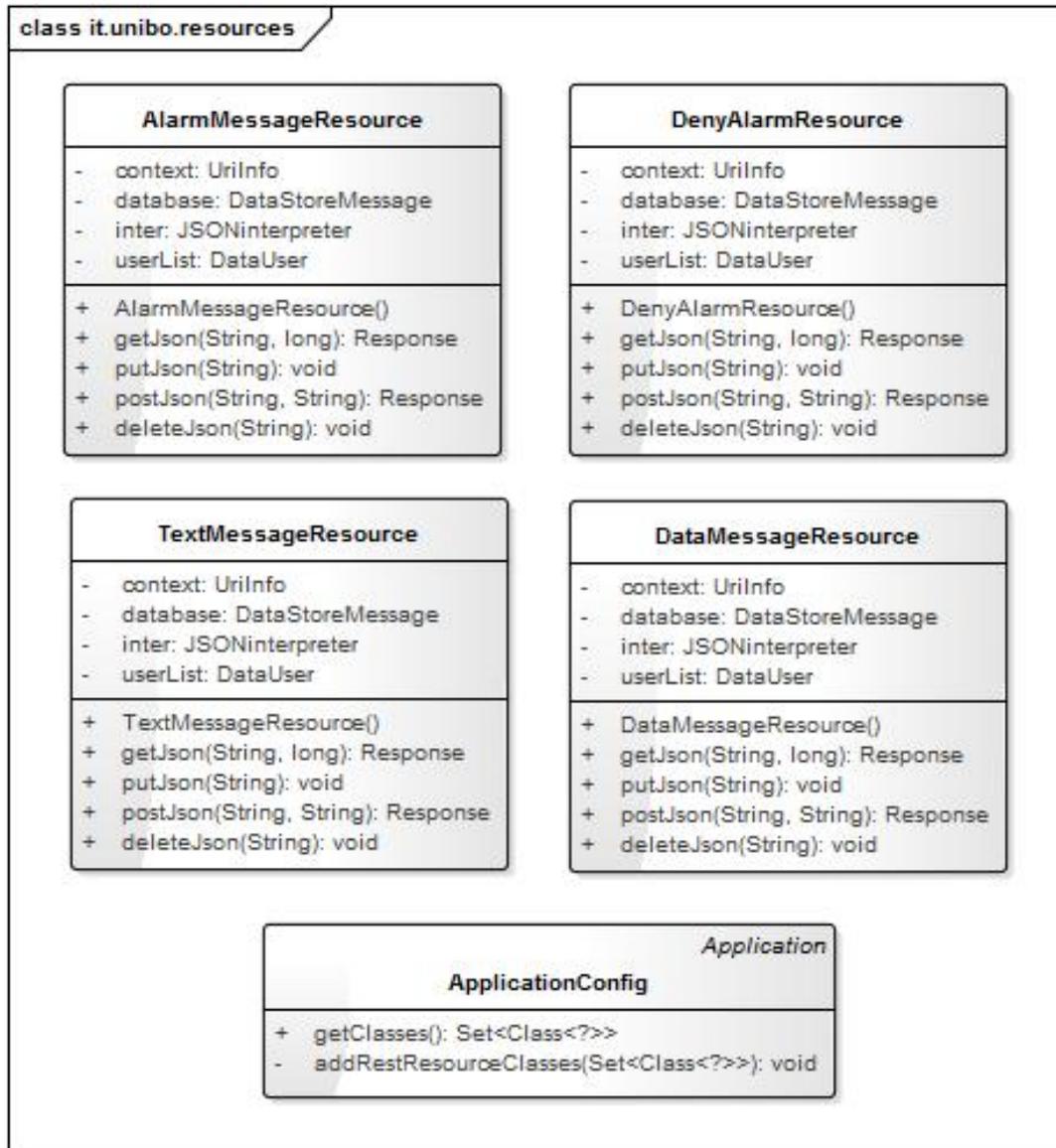


Figura 3.10: Struttura delle risorse del sistema.

circostanza di una richiesta di messaggi contenenti nuovi dati. Il comportamento del sistema rimane il medesimo anche nel caso di richiesta di altri tipi di messaggi. Nello specifico, in seguito a una richiesta di tipo *GET* inviata da un utente, viene prima eseguito un controllo per verificare che l'utente sia registrato e connesso, nel qual caso si procede interrogando l'oggetto *DataStoreMessage* per recuperare l'identificativo del prossimo messaggio che deve essere recapitato all'utente, quindi il messaggio, se presente, deve essere reperito, codificato in una Stringa JSON ed infine ritornato al richiedente.

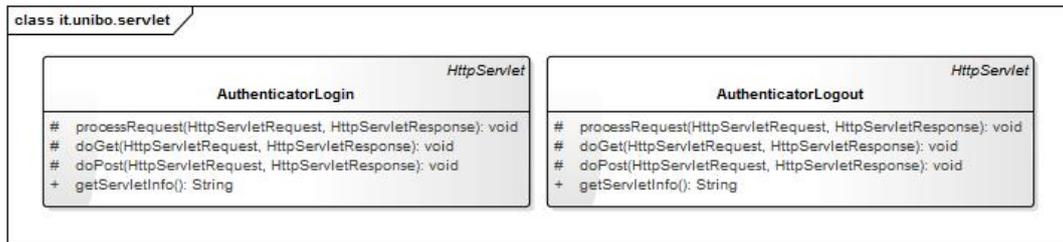


Figura 3.11: Struttura delle componenti *servlet* del sistema.

Nel caso in cui venga effettuata una richiesta di tipo *POST*, l'utente deve specificare il destinatario del messaggio, quindi l'esistenza dello specifico utente deve essere verificata e solo nel caso in cui esso sia effettivamente registrato al sistema, il messaggio deve essere decodificato attraverso l'oggetto *JSONInterpreter* e successivamente aggiunto al database, in modo che possa essere letto dal destinatario.

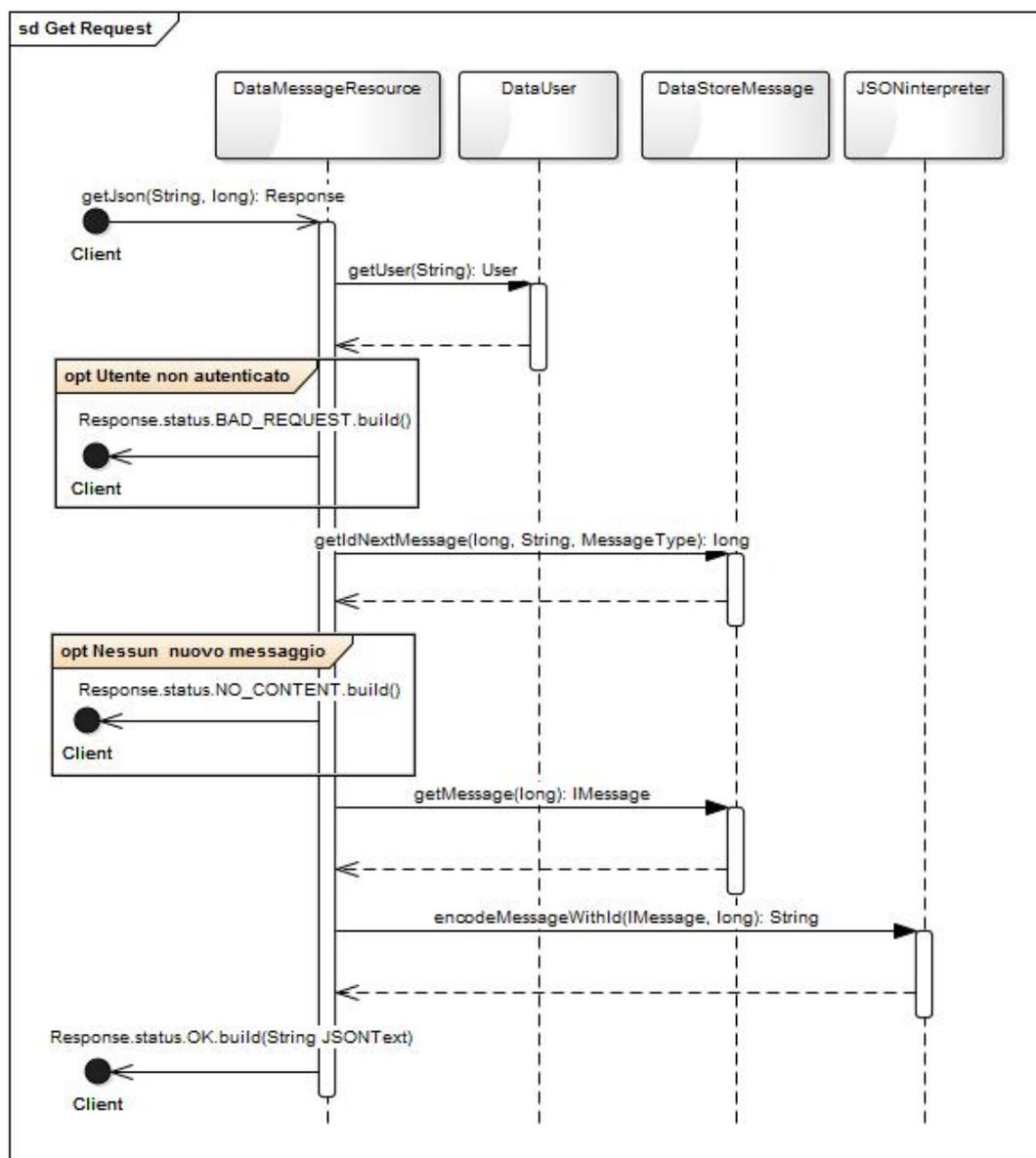


Figura 3.12: Interazione tra le componenti del Web Service REST a fronte di una richiesta di Post.

Capitolo 4

Implementazione

4.1 Componente Android

Per l'implementazione del prototipo dell'applicazione Android sono state quindi sviluppate due activity, *LoginActivity*, per permettere all'utente di effettuare l'autenticazione delle proprie credenziali, e *MainActivity*, che rappresenta l'activity principale del sistema. Inoltre è stato sviluppato un Service, *ActivityRecognitionService*, che utilizza la libreria chiamata *Android Activity Recognition* dopo che viene effettuato l'accesso ai *Google Play Service*. Inoltre è stato implementato un *BroadcastReceiver*, chiamato *SimpleBroadcastReceiver*, che riceve gli intent generati da *ActivityRecognitionService* e li comunica alla *MainActivity* sfruttando l'apposito oggetto *Handler*.

L'accesso ai *Google Play Service* sarà effettuato, oltre che dal service, anche dall'oggetto *LocationListenerGooglePS* per ricevere e tenere aggiornata la posizione GPS del dispositivo. Questa scelta implementativa viene adottata dal fatto che: "The Google Play services location APIs are preferred over the Android framework location APIs (android.location) as a way of adding location awareness to your app" [5] come riportato dal sito ufficiale della comunità Android.

Ai fini di una più facile comprensione del lavoro svolto, viene mostrata in questo paragrafo l'implementazione di alcuni dei componenti fondamentali dell'applicazione Android.

ActivityRecognitionService:

```
public class ActivityRecognitionService extends IntentService{  
  
    public ActivityRecognitionService() {
```

```

    super("ActivityRecognitionService");
}

@Override
protected void onHandleIntent(Intent intent) {
    if( ActivityRecognitionResult.hasResult(intent)){

        ActivityRecognitionResult res =
            ActivityRecognitionResult.extractResult(intent);
        DetectedActivity highPAct = res.getMostProbableActivity();
        int conf = highPAct.getConfidence();
        if( highPAct.getType() == DetectedActivity.ON_FOOT ){
            highPAct = walkingOrRunning( res.getProbableActivities());
        }
        Intent i = new Intent(SysKb.ACTIVITY_RECOGNITION_MESSAGE);
        i.putExtra("ActivityStateType", highPAct.getType() );
        i.putExtra("Accuracy", conf );
        sendBroadcast(i);
    }
}

private DetectedActivity walkingOrRunning(List<DetectedActivity>
probableActivities) {
    DetectedActivity myActivity = null;
    int confidence = 0;
    for (DetectedActivity activity : probableActivities) {
        if (activity.getType() == DetectedActivity.RUNNING ||
            activity.getType() == DetectedActivity.WALKING){
            if (activity.getConfidence() > confidence){
                myActivity = activity;
                confidence = activity.getConfidence();
            }
        }
    }
    return myActivity;
}
}

```

Receiver:

```

public class Receiver extends Thread {

```

```
private Handler hand;
private int msgId;
private boolean stopReceive;
private JSONInterpreter interpreter;
private String address;
private long idMsg;
private int freq;
private HttpURLConnection connection;
private StringWriter writer;

public Receiver(Handler handle, int mId, String address, int
frequenza ) {
    this.hand = handle;
    this.msgId = mId;
    this.address = address;
    this.stopReceive = false;
    this.interpreter = new JSONInterpreter();
    this.idMsg = 0;
    this.freq = frequenza;
    this.writer = new StringWriter();
}

public void run() {

    while (!this.stopReceive) {
        try {
            IMessage responseMsg = executeGet( this.address );
            if( responseMsg != null ){
                this.hand.obtainMessage(this.msgId,
                    responseMsg).sendToTarget();
            }
        } catch (IOException e1) {
            Log.d("receiver", "errore esecuzione get: errore di IO");
            e1.printStackTrace();
        } catch (MessageTypeException e) {
            Log.d("receiver", "errore esecuzione get: tipo di
                messaggio errato");
            e.printStackTrace();
        }
        try {
            sleep( this.freq );
        } catch (InterruptedException e){}
    }
}
```

```
public void stopReceiveMessage() {
    this.stopReceive = true;
}

private IMessage executeGet(String address) throws IOException,
    MessageTypeException {

    responseMsg = null;

    connection = (URLConnection) new
        URL(address+"?id="+this.idMsg).openConnection();
    connection.setRequestMethod("GET");
    connection.setDoOutput(false);
    connection.setRequestProperty("Accept", "application/json");
    connection.setRequestProperty("Content-Type", "application/json;
        charset=utf-8");

    int code = connection.getResponseCode();
    if (Response.Status.OK.getStatusCode() == code) {
        IOUtils.copy(connection.getInputStream(), this.writer,
            "UTF-8");
        String jsonText = writer.toString();
        this.idMsg = this.interpreter.getIdFromMessage(jsonText);
        return this.interpreter.decodeMessage(
            this.interpreter.removeExtra(jsonText) );
    }
    connection.disconnect();
    return null;
}
}
```

Sender:

```
public class Sender extends AsyncTask<Void, Void, Void> {

    private IMessage msg;
    private String url;
    private JSONinterpreter interpreter;

    public Sender(IMessage msg, String u){
        this.url = u;
        this.msg = msg;
    }
}
```

```
        this.interpreter = new JSONinterpreter();
    }

    @Override
    protected Void doInBackground(Void... params) {
        try{
            URL url = new URL( this.url );
            HttpURLConnection connection = (HttpURLConnection)
                url.openConnection();

            connection.setRequestMethod("POST");
            connection.setDoOutput(true);
            connection.setRequestProperty("Accept", "application/json");
            connection.setRequestProperty("Content-Type",
                "application/json; charset=utf-8");

            OutputStream out = connection.getOutputStream();
            this.interpreter.encodeMessage( msg, out );
            out.close();
            int code = connection.getResponseCode();

            if( code == Response.Status.CREATED.getStatusCode()){
                Log.d( "Sender", "post request effettuata
                    correttamente.");
            }else{
                Log.d( "Sender", "invio messaggio fallito, resp code:
                    " + code);
            }
            connection.disconnect();
        }catch( Exception e ){
            Log.d( "Sender", "invio messaggio fallito.");
            e.printStackTrace();
        }
        return null;
    }
}
```

BluetoothReceiver:

```
public class BluetoothReceiver extends Thread{

    private Handler hand;
    private int msgId;
```

```
private boolean stopRead;
private byte[] buffer;
private BluetoothSocket bts;
private InputStream btInStream;
private final char[] hexArray = "0123456789ABCDEF".toCharArray();

public BluetoothReceiver( Handler handle, int mId ){

    this.hand = handle;
    this.msgId = mId;
    this.buffer = new byte[SysKb.BL_BUFFER_DIMENSION];
    this.stopRead = false;
    this.btInStream = null;
    this.bts = null;
}

public void stopComunication(){
    this.stopRead = true;
}

private boolean setUpConnection(){

    Set<BluetoothDevice> pairedDevices =
        BluetoothAdapter.getDefaultAdapter().getBondedDevices();
    if (pairedDevices.size() > 0) {
        for (BluetoothDevice device : pairedDevices) {
            if( SysKb.BL_DEVICE_ADDRESS.equals( device.getAddress()
            ) ){
                try{
                    this.bts = device.createRfcommSocketToServiceRecord(
                        device.getUuids()[0].getUuid() );
                } catch( IOException e ){
                    Log.d( "blListener", "impossibile stabilire una
                        connessione");
                    return false;
                }
                try{
                    this.bts.connect();
                    this.btInStream = this.bts.getInputStream();
                } catch( IOException e1 ){
                    try {
                        this.bts.close();
                    } catch (IOException e2) {}
                    return false;
                }
            }
        }
    }
}
```

```
        }
        return true;
    }
}
}else{
    Log.d( "blListener", "dispositivo non trovato");
    return false;
}
return false;
}

public void run(){

    if ( !setUpConnection() ){
        stopCommunication();
        Log.d( "blListener", "connessione fallita");
    }

    while( !this.stopRead ){
        try {
            this.btInStream.read(buffer,0,SysKb.BL_BUFFER_DIMENSION);
            String data = bytesToHex(buffer);
            int heartRate,oxygenSat;
            int index = data.indexOf(SysKb.BL_FIRST_FRAME);
            if((index+27)< data.length() &&
                data.substring(index+10,index+14).equals(
                    SysKb.BL_NOT_FIRST_FRAME ) &&
                data.substring(index+20,index+24).equals(SysKb.BL_NOT_FIRST_FRAME)){
                heartRate=buffer[SysKb.BL_HRM_OFFSET+(index/2)] & 0x03;
                heartRate<<=7;
                heartRate = heartRate |
                    buffer[SysKb.BL_HRL_OFFSET+(index/2)];
                oxygenSat = Integer.parseInt("" +
                    data.charAt(SysKb.BL_OX_OFFSET*2+index) +
                    data.charAt(SysKb.BL_OX_OFFSET*2+1+index), 16);

                //invio parametri a handler
                par = new BiomedicalParameters( heartRate, oxygenSat);
                this.hand.obtainMessage( this.msgId , par
                    ).sendToTarget();
            }
        } catch (IOException e1) {
            e1.printStackTrace();
            stopCommunication();
        }
    }
}
```

```
    }

    try{
        this.bts.close();
    } catch(Exception e){}
}

private String bytesToHex(byte[] bytes) {

    char[] hexChars = new char[bytes.length * 2];
    for (int j = 0; j < bytes.length; j++) {
        int v = bytes[j] & 0xFF;
        hexChars[j * 2] = hexArray[v >>> 4];
        hexChars[j * 2 + 1] = hexArray[v & 0x0F];
    }
    return new String(hexChars);
}
}
```

4.2 Web Service REST

Per implementare il Web Service si fa ampio uso delle API JAX-RS. Esse sono delle API Java progettate per rendere più facile lo sviluppo di applicazioni che utilizzano l'architettura REST, attraverso l'utilizzo di particolari *Java programming language annotations*, pensate per semplificare lo sviluppo di servizi web RESTful [11].

Vengo elencate di seguito le particolari annotazioni fornite, utili per mappare una classe che contiene delle risorse accessibili via web:

- *@Path*, Il valore di questa annotazione indica il percorso relativo URI che riporta dove sarà ospitata la determinata classe Java che rappresenta una risorsa.
- *@GET*, *@PUT*, *@POST*, *@DELETE*, specificano le tipologie di richieste HTTP che possono essere chiamate su una risorsa.
- *@Produces*, questa annotazione specifica i MIME, media type, che la risorsa può produrre e che possono essere quindi ritornati al client.
- *@Consumes*, questa annotazione specifica i MIME, media type, che la risorsa può consumare e che possono essere quindi inviati dal client.

- *@PathParam*, è un tipo di parametro che può essere estratto all'interno della classe risorsa, i parametri di path vengono specificati nell'URI dal mittente e provengono quindi dall'URL.
- *@QueryParam*, è un tipo di parametro che può essere estratto all'interno della classe risorsa, i parametri di query provengono dai parametri di tipo query dell'URI.
- *@Provider*, questa annotazione viene utilizzata per tutto ciò che è di interesse per il JAX-RS runtime, come *MessageBodyReader* e *MessageBodyWriter*.

Ai fini di una più facile comprensione del lavoro svolto, viene mostrata in questa sezione l'implementazione di alcuni dei componenti fondamentali del Web Service, basato su modello REST, implementato.

DataMessageResource:

```
@Path("{destinatario}/datamessage/next")
public class DataMessageResource {

    @Context
    private UriInfo context;
    private DataStoreMessage database;
    private JSONInterpreter inter;
    private DataUser userList;
    /**
     * Creates a new instance of DataMessage
     */
    public DataMessageResource() {
        this.database = DataStoreMessage.getDBRef();
        this.inter = new JSONInterpreter();
        this.userList = DataUser.getDBRef();
    }

    /**
     * Retrieves representation of an instance of
     * resources.DataMessage
     * @return an instance of java.lang.String
     */
    @GET
    @Produces("application/json")
```

```

public Response getJson( @PathParam("destinatario") String dest,
    @QueryParam("id") long previousId ) {

    Logger.getLogger(DataMessageResource.class.getName()).log(Level.SEVERE,
        "nuova get data richiesta."+previousId );
    if( this.userList.getUser( dest ) == null ){
        return
            Response.status(Response.Status.BAD_REQUEST).build();
    }
    try {
        long id = this.database.getIdNextMessage(previousId, dest,
            MessageType.DATA);
        IMessage msg = this.database.getMessage(id);
        if( msg == null ){
            return
                Response.status(Response.Status.NO_CONTENT).build();
        }
        String res = this.inter.encodeMessageWithId(msg, id);
        return Response.status(Response.Status.OK).entity( res
            ).build();
    } catch (NoNewMessageException ex) {
        return Response.status(Response.Status.NOT_FOUND).build();
    } catch (JsonProcessingException ex) {
        Logger.getLogger(DataMessageResource.class.getName()).log(Level.SEVERE,
            null, ex);
        return
            Response.status(Response.Status.BAD_REQUEST).build();
    } catch (IOException ex) {
        Logger.getLogger(DataMessageResource.class.getName()).log(Level.SEVERE,
            null, ex);
        return
            Response.status(Response.Status.BAD_REQUEST).build();
    }
}

/**
 * PUT method for updating or creating an instance of DataMessage
 * @param content representation for the resource
 * @return an HTTP response with content of the updated or
 *         created resource.
 */
@PUT
@Consumes("application/json")
public void putJson(String content) {

```

```

    }

    @POST
    @Consumes("application/json")
    public Response postJson(String
        content, @PathParam("destinatario") String dest) {

        try {
            DataMessage msg = this.inter.decodeDataMessage(content);
            if( this.userList.getUser( dest ) == null ){
                return
                    Response.status(Response.Status.BAD_REQUEST).build();
            }
            this.database.addMessage(msg, dest);
            return Response.status(Response.Status.CREATED).build();
        } catch (IOException ex) {
            Logger.getLogger(DataMessageResource.class.getName()).log(Level.SEVERE,
                null, ex);
            return
                Response.status(Response.Status.NOT_ACCEPTABLE).build();
        } catch (MessageTypeException ex) {
            Logger.getLogger(DataMessageResource.class.getName()).log(Level.SEVERE,
                null, ex);
            return
                Response.status(Response.Status.NOT_ACCEPTABLE).build();
        }
    }
}

@DELETE
@Consumes("application/json")
public void deleteJson(String content) {
}
}

```

AuthenticatorLogin:

```

public class AuthenticatorLogin extends HttpServlet {

    /**
     * Processes requests for both HTTP GET and
     * POST
     * methods.
     *

```

```

    * @param request servlet request
    * @param response servlet response
    * @throws ServletException if a servlet-specific error occurs
    * @throws IOException if an I/O error occurs
    */
protected void processRequest(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
        /* TODO output your page here. You may use following
           sample code. */
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet Authenticator</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet Authenticator at " +
            request.getContextPath() + "</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}

// <editor-fold defaultstate="collapsed" desc="HttpServletRequest
// methods. Click on the + sign on the left to edit the code.">
/**
 * Handles the HTTP <code>GET</code> method.
 *
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException if an I/O error occurs
 */
@Override
protected void doGet(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    response.getWriter().print("<html lang=\"en\"><body><h1>Not
        implemented</body></h1></html>");
}

```

```
/**
 * Handles the HTTP <code>POST</code> method.
 *
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException if an I/O error occurs
 */
@Override
protected void doPost(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    //response.setContentType("application/json");
    JSONAutInterpreter in = new JSONAutInterpreter();
    String[] res = in.decodeUserWithPass(
        request.getInputStream() );
    DataUser userList = DataUser.getDBRef();
    if( userList.getUser(res[0])!= null &&
        userList.getUser(res[0]).verifyPassword(res[1]) ){
        userList.getUser(res[0]).setUserState(UserState.ONLINE);
        response.setStatus(SC_OK);
    }else{
        response.setStatus(SC_BAD_REQUEST);
    }
}

/**
 * Returns a short description of the servlet.
 *
 * @return a String containing servlet description
 */
@Override
public String getServletInfo() {
    return "";
} // </editor-fold>
}
```

4.3 Test

I test svolti hanno avuto come obiettivo quello di verificare il corretto svolgimento delle funzionalità richieste dal sistema in diverse condizioni. Per svolgere questi test è stato necessario implementare un'applicazione desktop sviluppata

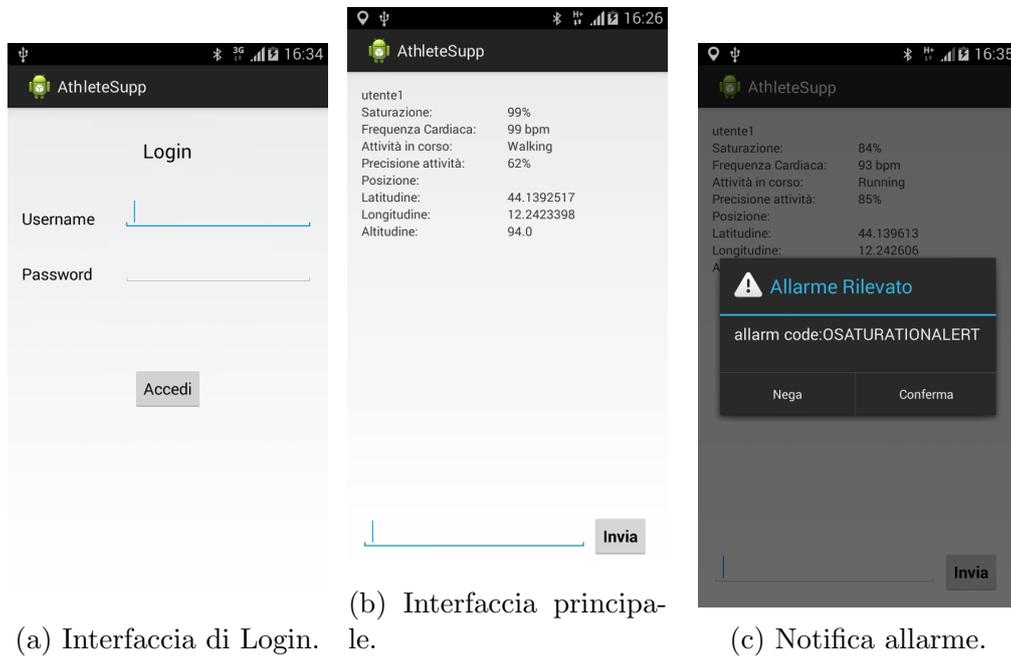


Figura 4.1: Screenshot applicazione Android in funzione.

in Java, sfruttando delle specifiche Java Applet chiamate PApplet, per verificare il corretto funzionamento del sistema complessivo, senza però soffermarsi a curare nel dettaglio quest'ultima parte implementata, riguardante la centrale operativa.

Come richiesto il sistema supporta agevolmente più centrali operative, scegliendo il giusto compromesso di aggiornamento dei dati, in modo tale da visualizzare sempre uno stato aggiornato degli utenti, ma senza sovraccaricare il sistema di comunicazione con l'invio di dati ridondanti o dallo scarso contenuto informativo.

In figura 4.1 sono riportati alcuni screenshot relativi all'applicazione sviluppata, in particolare viene visualizzato in figura 4.1a l'aspetto del sistema all'avvio dell'applicazione durante la fase di autenticazione, in figura 4.1b viene invece mostrata l'interfaccia, nella quale è possibile leggere i dati dell'utente e i messaggi ricevuti, inoltre è anche possibile inviare messaggi alle centrali operative attive. Infine in figura 4.1c viene riportata la notifica di allarme ricevuta dall'utente.

Per quanto riguarda la centrale operativa, viene mostrato in figura 4.2 l'interfaccia grafica proposta nel prototipo sviluppato per eseguire i test, come si può notare dall'immagine viene riportata all'utente una mappa del territorio, ottenuta utilizzando i dati forniti da *OpenStreetMap*, un progetto collaborativo

nato per rendere disponibile a qualsiasi persona dati cartografici gratuiti, senza quindi restrizioni di tipo tecnico o legali, per maggiori informazioni si rimanda alla pagina principale del progetto [6]. Sulla mappa esposta vengono quindi visualizzati degli indicatori, che mostrano la posizione attuale degli utenti Android connessi al sistema. Selezionando gli utenti è possibile visualizzare i dati attuali ricevuti, in modo da poter monitorare ogni singolo utente, inoltre si ha la possibilità di inviare loro messaggi, mentre i messaggi ricevuti vengono inseriti nella apposita chat sul lato destro dello schermo.

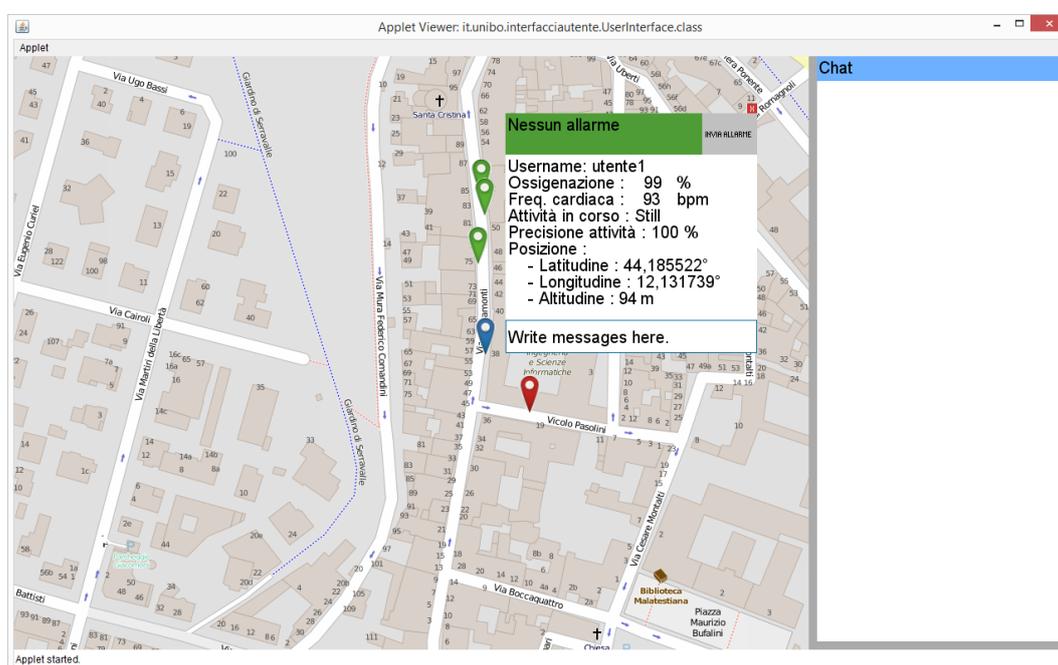


Figura 4.2: Interfaccia del prototipo di centrale operativa.

Conclusioni

Partendo dai requisiti proposti, il caso di studio sviluppato ha attraversato varie fasi, seguendo con rigore i canoni dettati dall'ingegneria del software, da prima con un'attenta analisi dei requisiti e del problema individuato, soffermandosi sulle precise richieste ricevute e ciò che ne consegue. Passando da un'attenta fase di progetto si è giunti infine all'implementazione di un sistema stabile, con varie caratteristiche. Prima tra tutte una forte estendibilità e notevole modularità, che renderanno il lavoro di perfezionamento, e magari anche di ampliamento, più facile e veloce.

Inoltre, il sistema implementato gode di un ottimo compromesso tra reattività e numero di utenti simultaneamente connessi supportati.

Questo sistema costituisce un buon punto di partenza anche per applicazioni più complesse, in quanto il numero di funzionalità può aumentare, andando ad estendere il progetto originale. Come sviluppi futuri si potrebbe, per esempio, affinare l'algoritmo di rilevamento allarmi potenziando quello già presente, semplicemente modificando l'elaboratore implementato. Oppure è possibile estendere il sistema aggiungendo un DBMS, utilizzando ad esempio a SQL (Structured Query Language), esso è un linguaggio standardizzato per database basati sul modello relazionale (RDBMS), il quale permetterebbe di creare e gestire strumenti di controllo ed accesso ai dati in maniere ottimale e performante, sempre mantenendo la consistenza dei dati. L'introduzione di questo sistema permetterebbe di estendere le funzionalità fornite dal prototipo, ad esempio, si potrebbero creare profili personali degli utenti che raccolgono informazioni riguardanti lo stato clinico del soggetto, in modo tale da fornire dati aggiuntivi agli operatori di soccorso in casi di allarme. Inoltre, si potrebbero sfruttare questi dati per affinare l'algoritmo di rilevamento allarmi, facendo in modo che patologie o disturbi dell'atleta, i quali possono minacciare lo stato di salute dell'utente durante la sessione di allenamento, vengano tenuti sotto maggiore controllo, ed inneschino il protocollo di emergenza in seguito alla rilevazione di parametri solitamente non allarmanti per soggetti in perfetto stato di forma.

Rimane inoltre da sviluppare un'adeguata centrale operativa, che segua quanto definito nel corso dell'analisi affrontata e che possa prendere spunto dal

prototipo sviluppato per la fase di test. La stessa potrebbe implementare un sistema che rilevi allarmi, non solo inerenti a un soggetto, ma anche a tutto il team, in presenza di situazioni di pericolo per il gruppo, o che permetta ad altri utenti di fornire assistenza di primo soccorso a soggetti in difficoltà nelle vicinanze, in attesa degli operatori addetti.

Un ulteriore scenario interessante potrebbe essere quello di consentire l'identificazione di ciascun atleta in modo più rapido, come specificato nei requisiti non funzionali, sfruttando il supporto della tecnologia NFC (Near Field Communication), che può essere tradotto letteralmente in italiano come comunicazione in prossimità. Si tratta di una tecnologia che fornisce connettività wireless a corto raggio, fino ad un massimo di circa dieci centimetri, permettendo una comunicazione bidirezionale utile allo scambio di un quantitativo limitato di dati. Si potrebbe immaginare, grazie a questa tecnologia, di sviluppare un meccanismo di rapido accesso al sistema, il quale, presupponendo di dotare ciascun atleta di una personale smartcard, effettui l'autenticazione delle credenziali dell'utente, semplicemente avvicinando la smartcard al dispositivo mobile, velocizzando l'accesso al sistema e ovviando al problema di memorizzazione delle proprie credenziali da parte dell'utente.

Oltre ai possibili sviluppi già elencati si potrebbe, senza alcuno sforzo, aggiungere, a quanto già sviluppato nel prototipo del dispositivo mobile, una funzionalità che permetta, analizzando i dati ricevuti dai sensori di cui è provvisto lo smartphone (giroscopio, accelerometro), di rilevare eventuali cadute dell'utente, in modo da inviare soccorsi se necessario.

È stato quindi sviluppato un sistema solido e completo, che lascia aperti interessanti sviluppi futuri e permette di essere esteso con numerose e stimolanti funzionalità.

Ringraziamenti

Arrivare in fondo a questo percorso è per me motivo di grande gioia e soddisfazione, la strada che mi ha portato fino a questo punto è stata lunga e tortuosa, ma mi ha regalato bellissimi momenti e tantissimi splendidi ricordi. Le conoscenze che ho acquisito in questi anni sono numerose, ma non sono cresciuto solo sotto questo punto di vista, sono migliorato sotto molti altri aspetti, dalla sicurezza in me stesso alla consapevolezza delle mie capacità, doti molto utili che mi aiuteranno nelle future sfide che mi verranno poste. Arrivato a questo punto è doveroso per me ringraziare numerose persone senza le quali non sarei riuscito a portare a termine questa impresa.

Primi tra tutti i miei genitori e mio fratello, che mi hanno sempre sostenuto e appoggiato nelle mie scelte, credendo in me e nelle mie capacità, sperando di essere, per loro, sempre fonte di orgoglio.

Un ringraziamento speciale a Riccardo, mio compagno e amico fidato, presente sin dall'inizio, con il quale ho trascorso parecchie ore di studio e condiviso soddisfazioni.

I miei amici, compagni di università e non, con i quali ho passato splendidi momenti e che mi hanno aiutato a svagarmi dopo giorni di studi.

Ai professori, che hanno contribuito ad ampliare il bagaglio delle mie conoscenze e a farmi maturare come persona, in particolare al Professor Mirko Viroli, che più di tutti mi ha seguito ed è per me grande fonte di ispirazione.

Un ringraziamento ad Angelo e Pietro, che mi hanno gentilmente assistito nell'ultimo periodo, fornendomi preziosi e utili consigli.

Infine un ringraziamento va a Martina, che mi ha sempre supportato e spronato a dare il massimo di me stesso.

Bibliografia

- [1] Wikipedia, Bluetooth <http://en.wikipedia.org/wiki/Bluetooth>, [Online; in data 6-marzo-2015].
- [2] Pagina ufficiale del dispositivo WristOx2, <http://www.nonin.com/OEMsolutions/WristOx23150-OEM>, [Online; in data 23-febbraio-2015].
- [3] Manuale ufficiale del dispositivo WristOx2, http://www.nonin.com/documents/IFUManuals/3150TechSpec_ENG.pdf, [Online; in data 23-febbraio-2015].
- [4] Google Inc *Service*, <http://developer.android.com/guide/components/services.html>, [Online; in data 9-marzo-2015].
- [5] Making Your App Location-Aware (Android Developer Website), <https://developer.android.com/training/location/index.html>, [Online; in data 4-marzo-2015].
- [6] OpenStreetMap project, <https://www.openstreetmap.org/about>, [Online; in data 11-marzo-2015].
- [7] Google Inc *Activity*, <http://developer.android.com/reference/android/app/Activity.html>, [Online; in data 9-marzo-2015].
- [8] Web Services Architecture, W3C Working Group Note, <http://www.w3.org/TR/ws-arch/>, [Online; in data 10-marzo-2015].
- [9] Wikipedia, Global Positioning System, http://en.wikipedia.org/wiki/Global_Positioning_System#Applications, [Online; in data 10-marzo-2015].
- [10] Andrea Fortibuoni, Sviluppo di una infrastruttura location-based per l'auto-organizzazione di smart-devices, 2013/2014.
- [11] Oracle, The Java EE 6 Tutorial, <http://docs.oracle.com/javaee/6/tutorial/doc/gilik.html#gilk>, [Online; in data 25-febbraio-2015].