ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

UN MIDDLEWARE DI COMUNICAZIONE E COORDINAZIONE PER SISTEMI MOBILE E WEARABLE BASATO SU TECNOLOGIA BLUETOOTH

Elaborato in
PROGRAMMAZIONE DI SISTEMI EMBEDDED

Relatore
Prof. ALESSANDRO RICCI

Co-relatore
Ing. ANGELO CROATTI

Presentata da
EDOARDO FRATI

Terza Sessione di Laurea
Anno Accademico 2016 – 2017

PAROLE CHIAVE

comunicazione

middleware

WBANs

bluetooth

android

a chi mi vuole bene

Indice

Introduzione

Sono trascorsi oramai più di settant'anni dall'invenzione dei primi calcolatori e il crescente sviluppo tecnologico ha portato a una larga diffusione di sistemi portabili facilmente accessibili da una vasta gamma di consumatori.

Nel decennio passato i computer sono stati i leader indiscussi del mercato, ma ad oggi qualcosa sta cambiando. Infatti se prima tutta l'attenzione era concentrata su computer desktop e portatili sempre più prestanti e meno ingombranti, ora ci si è focalizzati sui dispositivi mobile e sugli indossabili, ma più in generale sulla pervasività nell'uso quotidiano della tecnologia, dovuta principalmente alla miniaturizzazione dei componenti hardware e ai prezzi sempre più accessibili di questa.

In pratica il mobile e wearable computing si stanno intrecciando con la vita di tutti i giorni di tantissime persone, utilizzando lo smartphone come un centro di controllo attraverso il quale coordinare gli oggetti di uso comune trasformati in smart things. Per farsi un idea di quanto tutto questo sia un'argomento attuale è sufficiente pensare alle innovazioni nelle smart home, all'applicazione di queste nuove tecnologie in ambito sanitario o più in generale a tutti gli scenari produttivi, come industrie e grandi magazzini; analogamente un qualsiasi consumer oggi giorno sincronizza il telefono con un paio di cuffie senza fili e magari con uno smart watch.

Da questa rapida panoramica a tutto tondo è possibile constatare che l'utente si trova a convivere con un insieme numeroso di sistemi mobili, che da un lato ci permette di parlare di vere e proprie Wireless Body Area Networks (WBANs), ovvero reti di dispositivi di qualsiasi tipo che vengono utilizzati dalla stessa persone nel raggio di pochi metri.

Di conseguenza è necessario riconoscere che al cuore delle WBANs troviamo la questione dell'integrazione, nello specifico tutto quello che riguarda la comunicazione e la coordinazione dei dispositivi che le compongono, tanto che è possibile vedere queste reti di device come un vero e proprio sistema distribuito nel quale da un lato la computazione viene suddivisa sui vari dispositivi e dall'altro i diversi nodi hanno il crescente bisogno di scambiarsi messaggi, inviare comandi, ricevere notifiche o trasmettersi flussi d'informazioni.

Per questo motivo qualsiasi dispositivo ha integrato uno o più moduli per supportare le diverse tecnologie radio trasmissive, in particolare tra questi la più diffusa in ambito mobile e wearable è il Bluetooth.

Considerando quanto premesso finora e in seguito ad un'attenta ricognizione online tra le librerie open source disponibili è emersa una mancanza di un framework con questo scopo, per cui si è deciso di fissare come obbiettivo principale del lavoro di tesi quello di creare un middleware per dispositivi mobile e wearable al fine di agevolare la comunicazione e la coordinazione tra di questi, basato su tecnologia Bluetooth. Innanzitutto si è deciso di incapsulare il suo funzionamento sopra la logica radio trasmissiva, in modo da fornire un supporto di più alto livello utilizzabile dai vari applicativi. In particolare le fasi di analisi, progettazione e implementazione sono state finalizzate a rendere il layer di comunicazione il più generico possibile, in primis selezionando Android come sistema operativo di riferimento, coprendo così la maggior parte dei dispositivi delle categorie sopracitate. Inoltre si è optato per supportare tre diversi pattern di comunicazione di alto livello tra cui nello specifico quello a scambio di messaggi, la publish/subscribe e il modello di coordinamento basato su spazio delle tuple in modo da poter essere utilizzato nei più disparati scenari applicativi. Particolare attenzione si è dedicata nel garantire il funzionamento del middleware in modo robusto e resiliente davanti a malfunzionamenti e situazioni non previste, rendendolo così stabile e affidabile per i fruitori.

Infine si vuole arricchire l'intera comunità open source rendendo pubblico il codice sorgente del middleware attraverso il seguente repository GitHub: `https://github.com/edwfr/CommunicationMiddleware`.

Seguendo un percorso suddivisibile essenzialmente in due macro-fasi, la tesi si sviluppa attraverso sei capitoli. La prima fase, dedicata all'approfondimento del mobile computing, del wearable computing e dello standard Bluetooth sottolineano lo stato dell'arte di queste tecnologie. Nei successivi quattro capitoli, invece, viene descritta la fase relativa all'analisi, alla progettazione e allo sviluppo del middleware oggetto di tesi.

Nel primo capitolo vengono trattati gli argomenti del mobile e wearable computing, introducendo il concetto chiave delle WBANs e in conclusione definendo giù da subito l'obbiettivo del progetto.

Nel secondo capitolo viene presa in analisi lo standard Bluetooth a partire dalla diffusione a livello storico, passando per le caratteristiche tecniche ed infine il supporto che fornisce Android per questa tecnologia radio trasmissiva.

La seconda fase del percorso di tesi inizia con il terzo capitolo nel quale vengono analizzati i requisiti definiti per il middleware. Da questo emergono le caratteristiche funzionali e non che il sistema oggetto della progettazione dovrà implementare. Si conclude con i casi d'uso e la ricognizione in rete dei

framework open source che hanno simili funzionalità.

Nel quarto capitolo viene affrontata la progettazione del sistema precedentemente analizzato, definendo innanzitutto le API che questo dovrà mettere a disposizione e analizzando nel dettaglio le sue principali funzionalità.

Il quinto capitolo riporta la fase di implementazione del middleware, mettendo in luce le librerie utilizzate e i macro blocchi che compongono il progetto, mettendone in luce l'organizzazione e i principali dettagli di sviluppo.

Infine nel sesto ed ultimo capitolo vengono trattati i dati di validazione raccolti durante la fase di testing, sottolineando oltre al soddisfacimento di tutti i requisiti anche ottime prestazioni in termini di reattività e robustezza.

Capitolo 1

Mobile e Wearable Computing - Introduzione

The most profound technologies are those that disappear.
They weave themselves into the fabric of everyday life until they are indistinguishable from it. (Mark Weiser)[6]

In questo primo capitolo andremo ad analizzare una buona parte di tutte quelle tecnologie che, come afferma Weiser, si intrecciano nel tessuto della vita di tutti i giorni tanto che scompaiono. In particolare approfondiremo rispettivamente i temi del mobile computing e del wearable computing, sottolineando i principali eventi storici che hanno portato all'affermazione e lo sviluppo di queste tecnologie in larga scale nella nostra società fino ad introdurre il concetto di Wireless Body Area Networks e infine verrà fornita una prima descrizione del middleware oggetto di tesi.

1.1 Mobile computing

Il termine mobile computing definisce tutte quelle tecnologie che permettono lo scambio e l'elaborazione dei dati in ogni momento e da qualsiasi luogo con altri dispostivi anche attraverso l'utilizzo della connessione internet, in quanto alle volte in alternativa a questa vengono utilizzati altri tipi di reti per farli comunicare tra loro.

In particolare l'Information Systems Audit and Control Association's (ISACA) ha suddiviso i dispositivi mobile nelle seguenti sette tipologie:

- Computer portatile
- Tablet computer

- Smartphone
- Palmare - Portable Digital Assistants
- Dispositivi di archiviazione USB
- Radio Frequency IDentification (RFID)
- Dispositivi a infrarossi

In questa sezione andremo a ripercorrere la storia, la diffusione e le prospettive future di questo gruppo di dispositivi che sono diventati di uso comune durante tutte le nostre giornate.

1.1.1 Cenni storici

Il mobile computing nasce dall'intersezione della branca di sviluppo dei calcolatori con la branca di sviluppo delle radiocomunicazioni.
Per quanto riguarda l'evoluzione dei calcolatori, possiamo dividerla in sei differenti generazioni come emerge dall'analisi sostenuta nel Tanenbaum[7] ognuna delle quali caratterizzata da un periodo storico ed evidenti e distinte migliorie.

La prima generazione di computer meccanici (1642-1945) vede protagonisti Pascal (vissuto dal 1623 al 1662) e von Leibniz (vissuto dal 1646 al 1716) che inventarono delle macchine per effettuare somme e sottrazioni. A partire da questi progetti Babbage (vissuto dal 1792 al 1871) ideò prima la *difference engine* (macchina differenziale) in grado di effettuare le medesime operazione e poi l'*analytic engine*, macchina che non eseguiva più solamente uno stesso algoritmo ma bensì leggeva le istruzioni dalle schede perforate e le eseguiva.

La seconda generazione di calcolatori (1945-1955) venne alla luce nel periodo della Seconda Guerra Mondiale durante la quale questi appunto venivano utilizzati per lo scambio di messaggi cifrati tra le armate tedesche (ENIGMA e COLOSSUS). Questa fase sfocia nella costruzione da parte degli americani di ENIAC (Electronic Numerical Integrator And Computer) con lo specifico scopo di calcolare le tabelle per impostare la mira dell'artiglieria pesante, era costituita da 18.000 valvole termoioniche, 6000 interruttori e 1500 relè per un peso totale di 30 tonnellate. La realizzazione della macchina fu terminata solamente nel 1946, quando oramai non poteva più essere utile per lo scopo primario. Qualche anno dopo ci si accorse che la programmazione delle macchine attraverso un gran numero di interruttori era lenta e poco flessibile, motivo per cui nel 1951 Von Neumann insieme ad altri collaboratori progetto l'omonima macchina composta da memoria, unità di controllo e unità aritmetico logica, tale architettura viene tuttora utilizzata nei computer moderni.

La terza generazione di calcolatori (1955-1965) è caratterizzata dai Transistor, inventati nel 1948 nei laboratori Bell da cui uscì anche il TRADIC, primo calcolatore completamente a transistor (ben 684), a causa del quale i computer a valvole caddero in disuso. Nel 1958 nacquero i circuiti integrati, singoli chip in cui vengono integrati più componenti interconnessi tra di loro e l'anno successivo IBM mise in commercio una piccola macchina aziendale chiamata 1401 che ebbe grandissima diffusione e in seguito con il modello successivo (7094) l'azienda diventò leader del mercato.

La quarta generazione di calcolatori (1965-1980) è investita dalla novità dei circuiti integrati che avevano appena preso largo, facendo dimenticare rapidamente le vecchie macchine a interruttori. Di fatti nel 1971 Intel ha presentato la prima CPU basata su circuito integrato con conseguente lancio nel mercato del primo computer a 8 bit general purpose, così in quegli anni la Silicon Valley si popolò di migliaia di piccole start-up e delle più grandi aziende tecnologiche del mondo che contribuirono a dar forma ai computer che oggi giorno tutti utilizziamo, ad esempio tra le invenzioni più popolari di quegli anni possiamo citare i mouse e l'interfacce grafica per l'utente (GUI).

Dal 1980 fu possibile integrare gradualmente sempre più transistor in un'unica scheda, permettendo soprattutto di realizzare calcolatori più piccoli, più prestanti e più economici. I prezzi crollarono al punto che anche i privati potevano permettersi un computer, dando così alla luce la quinta generazione di calcolatori. Nello specifico IBM nel 1981 presentò il primo personal computer e successivamente Apple nel 1983 presentò Lisa il suo primo modello di PC dotato di interfaccia grafica, di mouse e lettore per floppy disk. Altre aziende come Commodore ed Atari provarono a conquistare il mercato di computer senza montare CPU Intel, ma la forza di mercato di quest'ultima era così grande che ne rimasero schiacciate. A fatica il Macintosh di Apple riuscì a mantenere la propria piazza di commercio nonostante gli elevati costi di produzione del primo modello, nonostante questo con i modelli successivi ebbe un enorme successo. A metà degli anni Ottanta iniziarono ad affermarsi nuovi tipi di progettazione CISC e RISC che prevedevano architetture più semplici e più veloci rispetto a quelle esistenti. Solamente qualche anno dopo, nel 1992 DEC presentò Alpha un calcolatore veramente innovativo con CPU a 64 bit di tipo RISC. Dopo di che quegli anni furono intensi di ottimizzazioni micro architetturali che aumentarono notevolmente la velocità di calcolo dei sistemi fino a che nei primi anni 2000 IBM lanciò il primo processore dual core con architettura parallela.

Infine la sesta generazione di calcolatori emerse negli ultimi anni del 1990 ed era caratterizzata da un'ulteriore riduzione delle dimensioni dei dispositivi, infatti i palmari Newton Message Pad di Apple del 1993 e Palm Pilot 1000 di US Robotics del 1996 dimostrarono che i computer non dovevano essere neces-

sariamente utilizzati sopra una scrivania o in una stanza apposita, ma potevano essere trasportati più o meno facilmente, potevano avere touch screen e integrare funzionalità più avanzate come il riconoscimento della scrittura manuale.

In secondo luogo analizzando il ramo dello sviluppo delle radiocomunicazione o comunicazioni wireless, ovvero senza fili, è necessario tornare al 1894 quando in Italia Guglielmo Marconi, a soli 20 anni era riuscito a costruire un apparecchio per inviare e ricevere segnali radio a lunga distanza. Pochi mesi dopo aveva brevettato a Londra la scoperta e successivamente fondato l'azienda *The Wireless Telegraph & Signal Company* attraverso la quale continuò gli studi di ricerca e iniziò la produzione di diversi tipi di dispositivi trasmissivi. Nel 1901 Marconi ricevette la prima comunicazione transoceanica, sostenendo che le onde radio potessero seguire la curvatura della terra e superare quindi l'oceano, di fatti riuscì a far comunicare Poldhu in Cornovaglia con St. John's di Terranova separati da più di 3000 chilometri. Tale comunicazione è in seguito stata rafforzata per garantire il primo servizio pubblico di radiotelegrafia attraverso l'Oceano Atlantico, fornendo ai naviganti la possibilità di lanciare messaggi di SOS senza fili.

Durante la Seconda Guerra Mondiale le radiocomunicazioni furono utilizzate per scambiarsi messaggi criptati a lunghe distanze, mentre per il corto raggio (1 miglio) l'esercito americano a partire dal 1942 utilizzò una sorta di walkie talkie attraverso il quale scambiarsi informazioni.

In seguito nel 1947 i ricercatori del AT&T Bell Labs iniziarono a progettare il primo modello di telefono, che però solamente nel 1973 verrà realizzato da Martin Cooper dalla Motorola. Parallelamente a questo sempre nei laboratori di AT&T Bell illustrarono un prototipo di rete cellulare pubblica, basata sull'utilizzo di piccole aree di servizio e celle radio che potevano riutilizzare le stesse frequenze. Solamente qualche anno dopo nel 1978 venne eseguita una prova pubblica della rete cellulare a Chicago con oltre 2000 utenze, ma si dovettero aspettare altri cinque anni prima che il servizio venisse messo in commercio nella città precursore (1G, prima generazione di trasmissione radio).

In Europa nello stesso periodo, più precisamente nel 1958 in Germania fu commissionata la prima rete wireless, che permetteva solamente chiamate in uscita (A-Netz). Dopo poco fu lasciata una seconda versione B-Netz che operava nelle stesse frequenze ma rendeva possibile la ricezione di chiamate da un'insieme determinato di telefoni, a condizione che la localizzazione della stazione mobile fosse nota. Queste reti erano wireless ma non erano reti cellulari, inoltre in questi sistemi non era possibile cambiare i ruoli delle stazioni nella comunicazione.

Nei primi anni del 1990, dopo le rilevanti invenzioni di Motorola e AT&T Bell Labs, si iniziò a lavorare allo standard IEEE802.11 per le Wireless Local

Area Network (WLAN) e al Hypertext Transfer Protocol (HTTP) dal quale prese vita il World Wide Web (WWW) con la pubblicazione del primo sito internet nel 6 Agosto 1991 da parte di Tim Berners-Lee del gruppo di ricercatori del CERN di Ginevra con il fine di condividere le informazioni tra le comunità dei fisici di tutto il mondo. Negli anni successivi fu sviluppato Masaic, il primo browser per la navigazione in rete, seguito subito da Netscape Navigator fino ad arrivare a tutti quelli conosciuti oggi tra cui Internet Explorer, Mozilla, Firefox e Safari.

Nel 2003 lo sviluppo dei calcolatori e quello delle radiocomunicazioni si incontro con la messa in commercio da parte dell'azienda Reaseach in Motion (oggi conosciuta come BlackBerry Limited) di BlackBerry Quark il primo modello di telefono che integrava a una tastiera qwerty funzionalità di chiamate, sms e servizi email.

Il 9 Gennaio 2007 Apple presentò il primo iPhone che in pratica è stato il primo smartphone commercializzato, padre dei dispostivi ampiamente diffusi al giorno d'oggi, che vedeva integrato un schermo multi-touch, tecnologia WI-Fi e rete dati 3G. Solamente tre anni dopo ci fu la messa in vendita del primo iPad sempre da parte di Apple, grazie al quale il colosso di Cupertino riscosse un grandissimo successo. Parallelamente nel 1998 Larry Page e Sergey Brin fondarono la oramai nota Google che rapidamente si fece riconoscere tra le tante start-up e dopo essersi consolidata tra le varie realtà della Silicon Valley nel 2005 decise di acquistare Android. Nel Novembre 2007 uscì la prima versione del sistema operativo che fu integrato per la prima volta nell'HTC Dream rilasciato l'anno successivo. Da lì in avanti Android prese rapidamente campo nel settore mobile rilasciando continuamente versioni aggiornate del sistema operativo e supportando sempre più dispositivi conquistando così una grande porzione di mercato (vedi grafico di figura 1.6).

1.1.2 Le caratteristiche

La maggior parte dei device elencati nell'introduzione alla sezione 1.1 condividono una serie di caratteristiche comuni tra cui le principali sono connettività, mobilità, usabilità e portabilità che analizzeremo brevemente di seguito.

Come riportano diversi documenti in letteratura, già dai primi anni '90 si intravedevano le grandi potenzialità della tecnologia wireless e nello specifico di connettere un dispositivo alla rete più vicina in base al luogo di utilizzo. Per confermare quanto queste intuizioni fossero corrette basta pensare che ora in qualsiasi smartphone sono salvate decine e decine di WLAN conosciute, una o due connessioni dati di terza o quarta generazione (3G - 4G) per rimanere collegati anche in assenza di Wi-Fi e un paio di auricolari più altri accessori accoppiati con il Bluetooth. I vantaggi delle reti wireless sono molteplici tra cui spiccano la flessibilità con la possibilità di connettere dispositivi non vicini tra di loro, la robustezza in quanto i collegamenti possono mantenersi attivi nei più disparati casi, la scalabilità andando a configurare la rete ad hoc per le rispettive esigenze e i costi ridotti dei device. D'altro canto abbiamo una qualità del servizio inferiore rispetto alle reti cablate in termini di larghezza di banda, velocità di trasferimento, interferenze, errori, perdita di dati, ritardi e soprattutto in sicurezza. È altresì vero che lo sviluppo delle diversi tecnologie wireless è in continuo miglioramento e si è raggiunto un livello più che soddisfacente per quanto riguarda la velocità di trasferimento dei dati e sofisticati standard di sicurezza, grazie ad algoritmi di crittografia sempre più complessi.

Oggi giorno basta prendere un treno anche per brevi spostamenti per notare quanto siano diffusi i dispostivi portatili e quanto siano fondamentali le dimensioni contenute e ottimizzate di questi, sia che si stia parlando di un computer, di un hard disk o di un tablet. Proprio per questo è necessario ottimizzare le dimensioni, il peso e la robustezza del device e di fatto questo mette di fronte al primo problema del mobile computing, ovvero le ridotte risorse in termini di memoria, batteria e potenza di calcolo. Se attraverso i Solid State Drive (SSD) si è risolta in parte il primo nodo del problema, essendo questi più leggeri e performanti degli hard disk meccanici, per il secondo non si sono ancora trovate soluzioni rivoluzionarie, infatti i dispostivi con più autonomia hanno batterie con capacità maggiore e quindi dimensione maggiore. Inoltre per migliorare ulteriormente la rendita delle batterie è necessario ricorrere all'implementazione di algoritmi di complessità inferiore al fine di far eseguire meno calcoli al processore e poter così ridurre la sua frequenza di clock. Quest'ultimo componente nei dispositivi mobile si tende ad averlo il più performante possibile anche se in questo preciso momento siamo arrivati ad un importante limite fisico, infatti se finora la prima legge di Moore che

affermava che "la complessità di un microcircuito, misurata tramite il numero di transistor per chip, raddoppia ogni 18 mesi" è stata rispettata ed ha permesso una crescente miniaturizzazione dei componenti elettronici ad oggi Intel, che vedeva Moore come uno dei storici co-fondatori della multinazionale statunitense, ha preso le distanze da questa osservazione empirica in quanto si è arrivati ad un limite sotto il quale è molto complesso scendere. Nello specifico i processori Intel Core di quinta, sesta e settima generazione commercializzati rispettivamente nel 2014, 2015 e 2016 sono composti da circuiti di dimensioni di circa 14 nanometri, è sufficiente pensare che un elettrone è una particella sub-atomica che occupa uno spazio di almeno 1-2 nanometri. Con queste dimensioni così ristrette diventa già arduo contenere la carica elettrica all'interno dei singoli elementi che compongono il transistor, senza che questa "salti fuori", motivo per il quale è difficile immaginare un ulteriore miniaturizzazione dei componenti[10].

Infine è necessario integrare nei singoli dispositivi un vasto set di sensori di input tra cui ad esempio puntatori, tastiera, touch pad, touch screen, accelerometro, giroscopio, GPS, fotocamera, microfono che permettono all'utente una vasta gamma di gesture per interfacciarsi velocemente con le funzionalità del sistema. Per non tralasciare poi l'user interface e il display, di alta risoluzione e alto refresh rate, attraverso i quali far vivere all'utente una straordinaria User Experience.

1.1.3 Diffusione

A partire dai dati del 2017 si nota che nonostante l'importante diffusione della tecnologia, le vendite di computer sono in continuo calo da diverso tempo, lo dimostrano i dati raccolti nell'ultimo trimestre del 2017 che segnano una riduzione di 67 milioni di unità (pari al 3,6%) rispetto allo stesso periodo dell'anno precedente. Solamente Apple, il colosso di Cupertino ha chiuso l'anno in positivo per quanto riguarda le vendite dei PC.

A dirla tutta il 22 Ottobre 2008 IBM, attraverso un sondaggio tra i propri clienti, aveva rivelato che più del 50% di questi avrebbe sostituito l'utilizzo del proprio personal computer con un dispositivo mobile. In particolare la maggior parte dei votanti dell'opzione *strong to full substitution* furono proprio gli utenti con età compresa dai 15 ai 30 anni, sostenitori in prima linea di tale rivoluzione.

A sostegno di questa tesi arrivano gli aggiornamenti da parte del Mobility Report del 2017 di Ericsson, illustrati graficamente dai seguenti snapshot.

Figura 1.1: Dati della diffusione di internet, social networks e telefoni cellulari nel mondo nel 2017[30]

Si può notare subito dalla figura 1.1 che attualmente nel mondo ci sono più SIM che persone, in quanto la popolazione mondiale è di 7,476 miliardi mentre le SIM attive superano gli 8 miliardi. In verità dato che attualmente un utente potrebbe aver sottoscritto più contratti si stimano circa 5,2 miliardi di persone abbonate a servizi di telefonia mobile. Altro dato importante è quello che vede circa il 50% della popolazione collegata ad internet, 2,8 miliardi dei quali sono anche utenti attivi sui social network.

Figura 1.2: Dati della diffusione di internet, social networks e telefoni cellulari in Italia nel 2017[30]

Per quanto riguarda il grado di diffusione in Italia possiamo tranquillamente dire di essere in proporzione sopra i livelli mondiali. Basta confrontare le percentuali contenute nei due precedenti snapshot (figura 1.1 e figura 1.2). In particolare su una popolazione di 59,80 milioni di persone, in Italia abbiamo circa 77 milioni di SIM attive. Per non parlare del rapporto degli italiani con il web, in particolare il 66% di questi è un utente di internet e più del 50% di tutta la popolazione è attivo sui social network.

Per concludere, se consideriamo che la stragrande maggioranza degli smartphone non è dual SIM, possiamo associarne uno ad ogni SIM al fine di sottolineare l'esponente diffusione dei dispositivi mobile, dovendo poi aggiungere a questa stima anche tutti quei smartphone e tablet che vengono utilizzati senza SIM e che contribuiscono ulteriormente ad aumentare il numero di device attivi in Italia e in generale nel mondo.

1.1.4 Scenari futuri

A questo punto dopo aver brevemente analizzato il percorso storico (sezione 1.1) che ha portato alla vasta diffusione del mobile computing e il relativo stato dell'arte attuale (sezione 1.1.3) non ci resta che interrogarci su come questo ramo della tecnologia potrà ulteriormente svilupparsi. Precedentemente nella sezione 1.2 sono già stati esposti i limiti della prima legge di Moore motivo per cui possiamo difficilmente prevedere grandi passi in avanti delle CPU dei dispostivi mobile, ma proprio per questo dovremo concentrare l'attenzione non più strettamente sui dispositivi, bensì sui servizi che ci circondano a partire da quelli cloud. Per introdurli è necessario riconoscere che il mobile computing si sta decentrando dai singoli device e questo aspetto viene nuovamente confermato dal Mobility Report di Ericsson il quale prevede un aumento esponenziale del traffico di dati entro il 2023 (da 2,9 GB/mese a 17 GB/mese per utente) sui dispositivi mobile caratterizzato soprattutto da un incremento di prestazioni e velocità di trasferimento di dati per via delle tecnologie e dello standard di quinta generazione (5G) come mostra il grafico della figura 1.3.

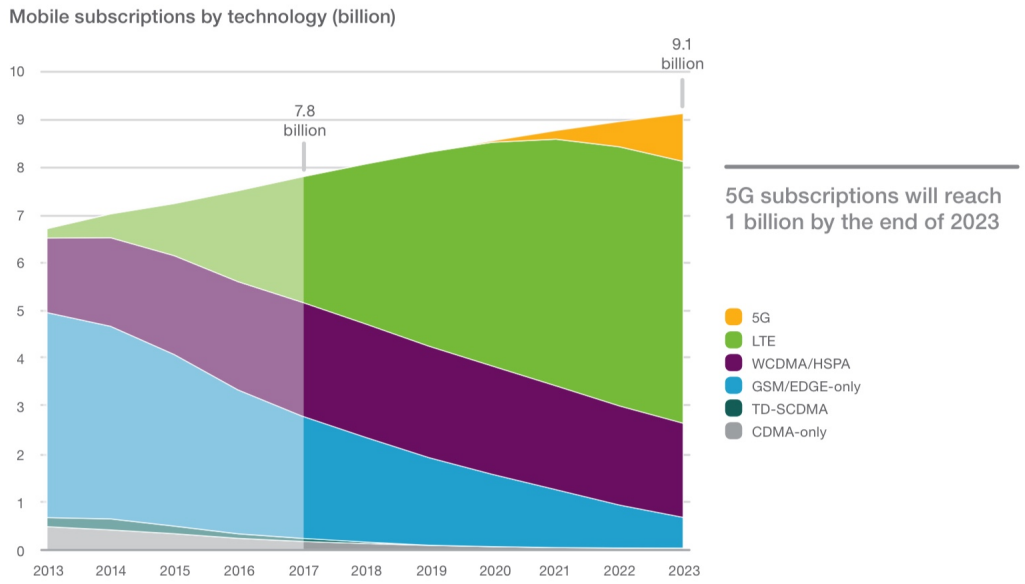


Figura 1.3: Grafico riguardante lo sviluppo e la diffusione delle tecnologie e dei protocolli di comunicazione nell'ambito della telefonia mobile entro il 2023, tratto dal Mobility Report di Ericsson.

Con il miglioramento delle tecnologie di connessione e le risoluzioni sempre più alte degli schermi dei nostri device cambierà anche il traffico di dati da dispositivi mobile, innanzitutto a quelli già noti si aggiungeranno anche dei computer e dei convertibili che hanno di default integrato nel sistema connettività LTE, come ad esempio i modelli presentati durante l'ultimo Consumer Electronics Show dell'8-11 Gennaio 2018 a Las Vegas (CES 2018). Come abbiano appena dettò il consumo di dati mensile aumenterà vertiginosamente nel giro di pochi anni e da quanto emerge dal grafico di fig.1.4 la maggior parte dei dati sarà per lo streaming video, ma allo stesso tempo guadagneranno spazio altri servizi che diventeranno sempre più fondamentali nelle nostre giornate, quelli strettamente collegati al cloud e in particolare nel *mobile edge computing (MEC)*. Quest'ultimo consiste in un architettura di rete basata sulla tecnologia 5G che consente funzionalità di cloud computing ai margini della rete cellulare. Alla base di tutto questo c'è la teoria che se le computazioni richieste vengono eseguite dal cloud center più vicino del client, il traffico e quindi la congestione della rete si riduce con conseguente miglioramento nell'esecuzione dell'applicazione. Inoltre il MEC è strettamente legato al *mobile cloud computing (MCC)* che garantisce la disponibilità di un insieme di risorse (ad esempio spazio di archiviazione, software, processori) attraverso la rete internet a tutti i dispositivi mobile, permettendone la rapida ed economica accessibilità. Tra i provider più diffusi di tali servizi ci sono *Amazon Elastic Compute Cloud* e *Microsoft Azure*,

per cui ad esempio attraverso un apposito servizio web è possibile eseguire da qualsiasi dispositivo applicazioni anche di grande complessità facendole girare sulle macchine virtuali, pagando i server per le ore di utilizzo.

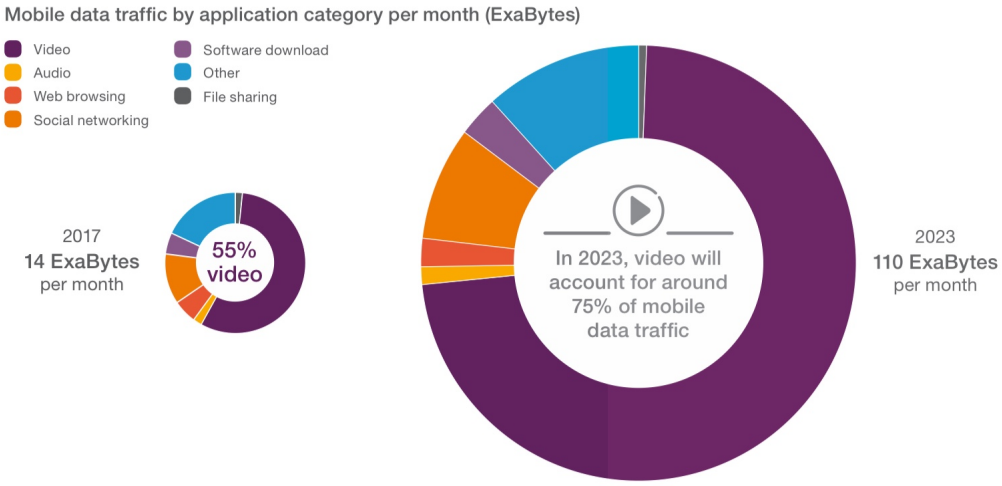


Figura 1.4: Grafico riguardante l'incremento del traffico dei dati da mobile entro il 2023 rispetto il 2017, tratto dal Mobility Report di Ericsson.

1.2 Wearable computing

In questa sezione andremo a trattare dei *dispositivi wearable*, ovvero piccoli computer, smartwatch, smartband, smartglass, visori per la realtà aumentate e in generale tutti quei device indossabili che in questi anni stanno facendo sempre più clamore nel mondo dell'informatica. Analogamente a quanto fatto nella sezione precedente andremo ad analizzare la storia, la diffusione e le prospettive future di queste tecnologie.

1.2.1 Introduzione

Prima di entrare nel dettaglio della storia e diffusione del wearable computing è necessario fornirne una definizione più dettagliata, per fare questo ne riporterò alcune tra quelle più diffuse in letteratura.

Wearable Computing must share the experiences of the user's life, drawing input from the user's environment to learn how the user reasons and communicates in relation to the world. As it learns, the computer can provide increasingly useful and graceful assistance. (Starner) [13]

Wearable computing facilitates a new form of human-computer interaction comprising a small body-worn computer system that is always on and always ready and accessible. In this regard, the new computational framework differs from that of hand held devices, laptop computers and personal digital assistants (PDAs). The "always ready" capability leads to a new form of synergy between human and computer, characterized by long-term adaptation through constancy of user-interface. (Mann) [14]

The fuzzy definition of a wearable computer is that it's a computer that is always with you, is comfortable and easy to keep and use, and is as unobtrusive as clothing. (Rhodes) [15]

Le tre citazioni sopra riportate vanno a definire sempre più nel dettaglio le caratteristiche dei wearable, infatti è possibile andare a delineare alcune proprietà che queste tecnologie hanno in comune.

- **funzionamento continuo**: questi dispostivi sono diversi dai computer perché possono essere utilizzati mentre si dorme, mentre si studia, mentre si fa sport, in qualsiasi momento insomma, senza la necessità di essere appositamente accesi e aperti come al contrario richiede un computer desktop. Di fatti sono progettati per essere sempre pronti all'interazione dell'utente e per non arrestare mai le loro funzioni. Inoltre devono essere comodi da portare in modo da consentire il loro prolungato impiego.

- **utilizzo hands-free**: in generale queste tecnologie non devono richiedere la costante e totale attenzione dell'utente, di fatti in pratica la maggior parte delle operazioni vengono eseguite mentre la persona sta facendo altro (ad esempio basta pensare ad uno smart watch che mentre mostra l'orario monitora i battiti cardiaci, i piani saliti e i spostamenti della giornata). Feature non secondaria è l'interazione vocale con questi dispositivi che hanno solitamente integrato il riconoscimento vocale dei comandi così da renderlo utilizzabile anche senza mani, *hands free*.

- **sensori**: oltre a quelli sopra citati per il monitoraggio della salute della persona il wearable solitamente ha anche altri sensori, tra cui i più diffusi sono accelerometro, GPS, microfono, altimetro e quello per il rilevamento della luce ambientale. A livello hardware sono solitamente dotati di connettività Bluetooth, Wi-Fi e alcuni hanno addirittura lo slot per la scheda SIM.

- **reattivo**: parallelamente ai dati che il device raccoglie, questo potrebbe avere la necessità di notificare qualche tipo di informazione all'utente per

cui deve essere capace di comunicare prontamente con l'utilizzatore. Ad esempio se durante un allenamento si raggiunge una soglia troppo elevata del battito cardiaco il dispositivo deve segnalarlo immediatamente così come deve mostrare una notifica per una chiamata persa o un email ricevuta.

1.2.2 Cenni storici

Tutti noi ci siamo resi conto della diffusione dei dispositivi wearable negli ultimi anni, dopo che le aziende come Apple e Samsung avevano presentato i loro smart watch e il programma di ricerca Google Glass aveva fatto passi da gigante. Nonostante questo i primi tentativi di progettazione di dispostivi indossabili risalgono al 1966 quando Ed Thorp e Claude Shannon rivelarono la loro invenzione: il primo computer tascabile, utilizzato per prevedere il risultato della roulette. L'anno successivo Hubert Upton inventò un dispositivo destinato alla lettura delle labbra attraverso l'applicazione di determinati filtri per individuare le parole pronunciate. Nel 1977 Hewlett-Packard rilasciò una sorta di calcolatrice da polso l'HP 01 con 28 tasti integrati e la possibilità di mostrare la data, l'ora e attivare e disattivare gli allarmi. La prima idea di ingannare al casinò per trarne profitto era sembrata così valida che nel 1978 Eudaemonic Enterprises inventò un piccolo calcolatore da posizionare sotto la suola della scarpa attraverso il quale predire il risultato della roulette, anche se con questa tecnologia gli inventori non hanno mai fatto il "big score" portando a casa piuttosto piccole somme e impreviste scosse elettriche . Qualche anno dopo allo stesso fine fu presentato un dispositivo in grado di contare le carte per i giochi d'azzardo. Nel 1981 Mann progettò una sorta di precursore degli attuali smar glass, infatti aveva integrato in un dispositivo multimediale un processore, un display laterale, visibile da un solo occhio e una fotocamera. Nel 1989 tale idea venne raffinata e fatta uscire dai laboratori della Reflection Tecnology, senza riscuotere grande successo per via della scarsa qualità del visore. Successivamente nel 1990 Gerald Chip Maguire e John Ioannidis presentarono un calcolatore per studenti composto dal sistema Private Eye e da un computer Toshiba senza disco rigido dotato di servizi TCP. Nel 1991 Doug Platt presenta il suo Hip-PC, un dispositivo con tecnologia 286, un floppy drive da 1,44 MB e una agenda palmtop usata come tastiera. In parallelo prese vita VuMan 1, dotato di CPU da 8 MHz e 0.5 MB di ROM, per la navigazione e visualizzazione di dati. Proprio in quegli anni Mark Weiser propose la sua idea di Ubiquitous Computing e Thad Starner cominciò la sperimentazione del suo sistema indossabile basato su Hip-PC. Finalmente nel 1993 BBN porta a termine il sistema Pathfinder, calclatore dotato di GPS e sistema di rilevazione delle onde radio. Nel frattempo alla Columbia University viene presentato il

primo progetto di realtà aumentata denominato KARMA (Knowledge-based Augmented Reality for Maintenance Assistance) che era composto da un Private Eye operativo su un occhio, attraverso il quale l'utente poteva avere un'esperienza di realtà aumentata in alcuni ambiti specifici del mondo reale. Ad esempio un tecnico chiamato a riparare una stampante poteva utilizzare il dispositivo per avere sottomano le istruzioni di manutenzione. Tutto questo era possibile grazie all'utilizzo di sensori appositi applicati agli oggetti del mondo fisico e dal collegamento ad un archivio dati disponibile attraverso un computer desktop. Sempre Mann nel 1994 inventò un videocamera indossabile attraverso la quale si potevano trasmettere le immagini ad una pagina web attraverso le frequenze radio e l'elaborazione di queste da dispostivi preposti. Da quegli anni in poi l'attenzione dei ricercatori si spostò prevalentemente sullo sviluppo di computer da polso, integrando prima tastiere qwerty, poi sempre più complesse interfacce grafiche per permettere una più agile interazione da parte dell'utente.

1.2.3 Stato dell'arte

Ad oggi, grazie alla sempre più vasta diffusione dello smartphone e alla miniaturizzazione dei componenti, il mercato si è arricchito di una quantità sproporzionata di wearable, facendo diventare il telefono il centro di controllo di tutti gli altri device. A partire dal settore sportivo che ha visto a tutti i livelli significative innovazioni, per esempio basta pensare che oramai qualsiasi amatore ha un fitness tracker o uno smartwatch per monitorare i parametri vitali sotto sforzo e a riposo, fornendo importanti statistiche sulla qualità del proprio allenamento. Ad esempio Fitbit è stata fondata nel 2007 e ha incentrato tutto il suo business attorno ai fitness tracker, diventando per tutti gli appassionati di sport un vero e proprio punto di riferimento. Allo stesso tempo i migliori club calcistici mondiali, tra cui Bayer Monaco, Real Madrid e Barcellona, hanno iniziato ad utilizzare la Audi Index Player, recente tecnologia per il monitoraggio degli allenamenti dei professionisti studiata dalla famosa casa automobilistica tedesca. In particolare facendo indossare ai calciatori delle speciali maglie con diversi microchip che producono milioni di bit, sono in grado di monitorare in tempo reale la loro forma fisica, velocità, agilità e forza, aiutando così lo staff a prendere le migliori decisioni in merito al allenamento per ogni singolo individuo.

Chiaramente i wearable sono largamente utilizzati anche in ambito sanitario, in particolare per abbassare le barriere che rinchiudono in se stessi i disabili. Ad esempio grazie a Talking-Hands, recente progetto della start-up Limix[1] dell'Università di Camerino, i segni prendono voce. Infatti mediante un

[1]Limix srl, `www.limix.it`

guanto riescono a registrare i movimenti delle mani durante l'utilizzo del linguaggio dei segni (LIS) e trasferendoli ad uno smartphone è possibile tradurli in parola, attraverso un sintetizzatore vocale che pronuncia la frase.

Un'altra applicazione molto interessante è quella di Tactile[2], progetto in via di perfezionamento, che permette la traduzione di un testo istantaneamente nel linguaggio braille per renderlo disponibile anche alle persone affette da cecità. Il dispositivo dispone di una fotocamera che cattura le parole e le traduce nel linguaggio braille. I puntini presenti sulla tastiera si attivano a seconda della parola e gli utenti riescono così a capire cosa c'è scritto.

Sicuramente i wearable mirano a rendere meno difficile la vita dei pazienti, senza però dimenticarsi di aiutare anche i medici nello svolgimento del loro lavoro. In questo ambito gli smart glass vengono utilizzati per visualizzare real time cartelle cliniche e dati riguardo al paziente oppure guidare il medico durante un operazione chirurgica mentre altri device potrebbero nello stesso tempo raccogliere informazioni direttamente riguardo la persona sotto i ferri.

Ovviamente anche tanti altri ambiti lavorativi sono stati influenzati dagli indossabili, ultimo esempio i grandi magazzini[3] che dotano alcuni dipendenti di smart glass attraverso i quali orientarsi e riconoscere agevolmente i prodotti direttamente dagli imballi camminando nei corridoi del magazzino. In particolare il 2017[4] ha visto una crescente diffusione di brand come Epson, Vuzix e Sony per quanto riguarda gli occhiali, ma soprattutto è stato considerato da molti come l'anno di svolta visori per la realtà aumentata, anche se modelli come Oculus Rift, HTC Vive e PlayStation VR, top di gamma del settore, hanno ancora molta strada da fare.

1.2.4 Scenari futuri

Per concludere il percorso iniziato nelle sezioni precedenti sulla progettazione di wearable sempre più portabili e di uso comune, sembra che in questo ambito la nuova frontiera sia quella degli smart clothes abbandonando così le band, gli occhiali e gli orologi intelligenti. Sono proprio le grandi multinazionali dell'abbigliamento sportivo insieme a quelle dell'informatica a collaborare per iniziare a ragionare su come rendere il vestiario combinato con lo smartphone il più smart possibile. Ad oggi nel mercato ci sono dei modelli di scarpe che hanno integrato il conta passi, calzini e suole che monitorano la pressione esercitata dal piede durante la corsa, collant per monitorare attività fisica di corridori e ciclisti. Durante CES 2018 la società giapponese Dai Nippon Prin-

[2]Il traduttore portatile testo-braille, Focus Tecnologia e Innovazione, 19 Maggio 2017

[3]Vision Picking in DHL, `https://youtu.be/I8vYrAUb0BQ`

[4]RV e AV, crescita a tre cifre nel 2017, `https://www.corrierecomunicazioni.it/digital-economy/realta-aumentata-e-virtuale-nel-2017-crescita-a-tre-cifre/`

ting Co ha presentato i display di carta elettronici. Questi sono stati utilizzati per realizzare un abito da donna, caratterizzato da effetti ottici e luminosi che non affaticano la vista grazie alla tecnologia E-ink, la stessa utilizzata per gli ebook reader. Tralasciando questa recente e alternativa nota colorata di applicazioni del wearable computing, la crescita e lo sviluppo di questo settore per i prossimi anni sarà focalizzato sull'aumento del monitoraggio della persona nella maniera meno intrusiva possibile così da facilitarne la diffusione.

1.3 WBANs

In quest'ultima sezione introdurremo le Wireless Body Area Networks (WBANs) e di conseguenza della necessità di integrazione tra i dispositivi che le compongono, argomenti fra di loro strettamente interconnessi.

Precedentemente abbiamo citato più volte i fattori che hanno portato allo sviluppo di tutte queste tecnologie, di fatti è proprio grazie alla grande diffusione delle reti wireless e alla miniaturizzazione dei componenti elettronici dei nostri dispostivi mobile che oggi stiamo parlando di Wireless Body Area Network. Con questo termine possiamo, in maniera sommaria, definire una rete di dispostivi mobile, wearable e sensori messi in collegamento tra di loro. Dato l'elevato numero di dispositivi mobile in circolazione al giorno d'oggi e vista l'esponente diffusione di dispositivi wearable, come abbiamo precedentemente affermato nella sezione 1.2.3, è doveroso introdurre le WBANs. Soprattutto perché in questo modo si è vista aumentare la mole di dati scambiati tra smartphone, indossabili e sensori, considerando lo sviluppo di questi ultimi che monitorano il nostro corpo con conseguente aumento dei dati prodotti sulla nostra salute, dai battiti cardiaci, ai passi giornalieri, alla temperatura corporea, alla postura mantenuta durante la giornata. Proprio per agevolare la comunicazione tra i vari device è necessario introdurre un architettura che ne ottimizzi il funzionamento, come vedremo di seguito.

1.3.1 Wireless Body Area Networks

Poche righe sopra abbiamo delineato una definizione approssimativa, ora cerchiamo di raffinarla ulteriormente specificando che una Wireless Body Area Network è un'architettura di rete che permette l'interconnessione tra più dispositivi utilizzati dalla stessa persona. In generale i device che vengono collegati alle WBANs sono a basso consumo energetico e di dimensioni ridotte, ma questo non vieta che siano attuatori o sensori, invasivi o non invasivi.

Nella pratica queste reti possono avere molteplici scenari di utilizzo, tra cui intrattenimento, addestramento militare, gaming, monitoraggio dell'ambiente,

allenamenti sportivi, ma l'ambito di maggior interesse e più sviluppato è quello dell'healthcare dato il significativo aumento di sensori per il monitoraggio real-time dei parametri vitali.

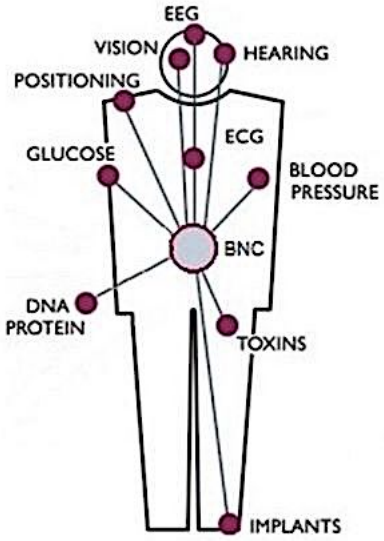


Figura 1.5: Esempio di Wireless Body Area Network in ambito healthcare

Per scendere ulteriormente nei dettagli si deve anche riconoscere che il concetto di Wireless Body Area Networks è piuttosto recente per cui le caratteristiche più rilevanti, che elencheremo brevemente di seguito, sono anche quelle che maggiormente le distinguono, rappresentando allo stesso tempo anche le sfide più interessanti per il futuro.

- **architettura**: in una WBAN possiamo avere solamente due tipi di nodi i sensori/attuatori e i router, i primi che come abbiamo detto più volte raccolgono dati e comunicato con l'utente e i secondi che fungono da infrastruttura per trasmettere i dati.

- **densità**: il numero di sensori e attuatori impiegati dipende dal singolo caso d'uso, alle volte potrebbe essere prevista ridondanza di sensori per prevedere malfunzionamenti e avere maggior accuratezza dei dati. In particolare nel campo healthcare è necessario prevedere un accurato sistema di validazione delle misure al fine di ridurre la generazione di falsi allarmi.

- **durata della batteria**: dato che si sta parlando di una rete di dispositivi è necessario tenere in considerazione la ridotta capacità delle singole batterie al fine di ottimizzare il software progettato ad hoc.

- **mobilità**: è necessario avere l'accortezza di non eccedere con la ridondanza di dispostivi che misurano i stessi parametri in quanto si deve comunque garantire una confortevole mobilità e portabilità del sistema.

- **data rate**: dato che solitamente questa architettura viene utilizzata per monitorare lo stato di salute di una persona, gli eventi che si vanno a studiare hanno frequenza costante e non variano sostanzialmente nel tempo, motivo per cui il flusso di dati dell'applicazione rimane costante durante l'utilizzo.

- **sicurezza**: la sicurezza è un punto fondamentale per tutte le reti, soprattutto nel momento in cui andiamo a monitorare i parametri dell'utente trattando così dei dati sensibili. Nello standard IEEE 802.15.6, specifico per Wireless Body Area Networks, vengono definite le regole per reti a basso consumo e a corto raggio; inoltre tutti i nodi devono garantire tre livelli di sicurezza: livello 0 - insicuro, livello 1 - con autenticazione ma nessuna crittografia e livello 2 - con autenticazione e crittografia.

- **mole di dati**: considerando i due punti appena citati, una WBAN genera una grande mole di informazioni che necessità delle giuste tecnologie in primis per essere trasmessa, ma anche per mantenerla e gestirla al meglio nei dispositivi preposti.

- **interoperabilità**: le Wireless Body Area Network devono garantire un trasferimento dati senza interferenze e con il minor numero di interruzioni possibile, attraverso i diversi standard di comunicazione per agevolare lo scambio di informazioni e il rapido collegamento tra i diversi device.

1.3.2 Necessità di integrazione tra dispositivi

Dopo aver introdotto brevemente le Wireless Body Area Network e le rispettive caratteristiche nella sezione precedente, emerge chiaramente che al cuore di questa architettura troviamo la necessità di integrazione tra i vari dispositivi, in modo da rendere un set di sensori distinti in una reta interconnessa di device che comunicano tra di loro, proprio come anticipava l'ultimo punto della sezione 1.3.1. Anche se alle volte le WBANs vengono semplicemente chiamate con il termine Body Area Networks (BANs), è necessario sottolineare l'importanza della parola Wireless, poiché non a caso alla base di questa architettura di rete c'è la comunicazione senza fili a corto raggio (solitamente entro 1 o 2 metri), al fine di restringerne l'utilizzo da parte di una stessa persona. Infatti da più definizioni che si trovano nella letteratura viene posta l'attenzione non tanto sui dispositivi che vengono interconnessi quanto sulla

prossimità tra di questi durante l'utilizzo. Durante gli anni passati ci furono dei progetti che prevedevano il collegamento cablato tra i vari dispositivi, ma tale idea ricevette poco successo data la crescente affermazioni delle tecnologie senza fili che aumentavano soprattutto la mobilità e la portabilità del sistema, aprendo allo stesso tempo interessanti sfide per l'ottimizzazione dei sensori e delle tecniche di elaborazione dei dati, in modo da garantire in primo luogo una resa soddisfacente delle batterie dei device e un miglior funzionamento dell'applicativo. Per cui è altrettanto interessante sottolineare che anche in reti più generiche di dispositivi [29] si può parlare di Wireless Body Area Networks e ancor più che il collegamento e l'interazione tra di essi sono funzionalità sempre più necessarie nei complessi sistemi di oggi.

Attualmente le tecnologie radio trasmissive maggiormente utilizzate nelle Wireless Body Area Networks sono quelle general purpose tra cui Bluetooth Classic, Bluetooth Low Energy e ZigBee[5], dato il grande supporto che hanno alle loro spalle. Anche se il dirompente ingresso della tecnologia nei diversi settori produttivi, ha contribuito alla nascita e diffusione di framework special purpose come ANT[6], RuBee[7], Radio Frequency Identification RFID, Sensium[8], Insteon[9] e Z-Wave[10], solitamente basati sulle tecnologie senza fili sopra citate ma con l'integrazione di protocolli e standard proprietari ottimizzati per lo specifico ambito applicativo.

[5] ZigBee, `www.zigbee.org`
[6] ANT, `www.thisisant.com`
[7] RuBee, `www.rubee.com`
[8] Sensium, `www.sensium.co.uk/sensiumprod/`
[9] Insteon, `www.insteon.net`
[10] Z-Wave, `www.z-wave.com`

1.4 Obbiettivo della Tesi

Al termine di questo capitolo, dopo aver analizzato lo stato dell'arte in ambito mobile e wearable computing e dopo aver riconosciuto la necessità di integrazione tra dispositivi di diverso tipo, dai tablet ai sensori con la conseguente diffusione delle Wireless Body Area Networks possiamo delineare l'obiettivo principale del lavoro di tesi, ovvero quello di creare un middleware per lo scambio di messaggi secondo pattern di alto livello tra dispositivi mobile, embedded e wearable al fine di astrarre la tecnologia trasmissiva su cui questo si basa e i suoi rispettivi modelli di comunicazione. Proprio per questo si parla di middleware e non di framework o libreria, perché l'intento è quello di realizzare a livello concettuale un layer che incapsuli tutta la logica della comunicazione, senza richiedere all'utilizzatore di scendere sotto il livello della tecnologia trasmissiva. Inoltre con tale termine si identifica un livello software tra il sistema operativo e le applicazioni in grado di fattorizzare servizi e funzionalità di alto livello e renderle disponibile ad ogni applicazione che ne ha bisogno. In particolare oltre a fornire funzionalità reattive per instaurare e mantenere attiva la connessione tra due o più device, queste deve inoltre prevedere tre differenti tipi di comunicazione, quello a scambio di messaggi, il pattern publish/subscribe e il modello di coordinazione basato sullo spazio delle tuple proprio per favorire la sua diffusione in larga scala a prescindere dall'ambito applicativo o dai dispositivi che ne fanno utilizzo. Tutti gli altri requisiti verranno poi definiti dettagliatamente nei capitoli successivi.

1.4.1 Android e Bluetooth

Nello specifico si è scelto Android come sistema operativo di riferimento, questa scelta non è casuale ma bensì è dettata da una crescente diffusione del sistema operativo sviluppato da Google. In primis Android è leader indiscusso nel settore mobile con più dell'85% del mercato, come riportano i dati raccolti da IDC nel grafico di figura 1.6 aggiornato a Maggio 2017. Inoltre viene sempre più utilizzato anche nei sistemi embedded e nel settore di Internet of Things (IoT) al di fuori degli indossabili e in generale delle piattaforme mobile. Tipicamente il suo impiego viene esteso a tutti quei sistemi che richiedono un interfaccia utente e servizi multimediali. Questo perché a differenze delle soluzioni proprietarie mette a disposizione una piattaforma corredata da una puntuale documentazione, un kernel stabile e soprattutto un'ampia API open source senza la richiesta di licenza da pagare o restrizioni, non è comunque esente da criticità in quanto globalmente richiede più memoria e più potenza di calcolo rispetto ad un sistema proprietario ottimizzato ad hoc per lo specifico scenario applicativo.

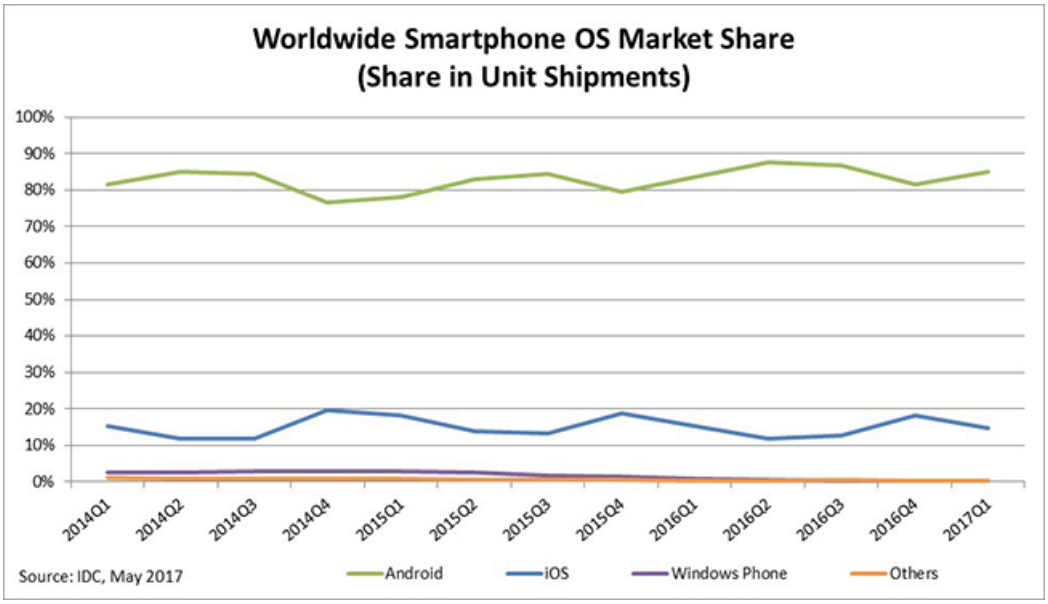


Figura 1.6: Diffusione dei sistemi operativi nel mondo,
tratto da: `www.idc.com/promo/smartphone-market-share/os`

"Bluetooth has been one of the key technologies underpinning the consumer wireless revolution." (Stuart Carlaw, ABI Research)

Infine tra le tecnologie trasmissive citate nella sezione 1.3.2 si è deciso di appoggiare il middleware al Bluetooth Classic che come aveva intuito Carlaw avrebbe rivoluzionato il mercato wireless. Rispetto agli standard più nuovi come Bluetooth Low Energy e Bluetooth 5 si è preferito supportare lo standard Classic perché grazie a questo avremo coperto la maggior parte dei dispositivi mobile, wearable e embedded, a partire dai più datati fino ai più moderni, dal modello più recenti di smart glass al sensore meno diffuso. Basti pensare che nel 2007 nel mondo si contavano un miliardo [20] di dispositivi che integravano il Bluetooth Classic e solamente quattro anni dopo, nel 2011, agli albori del Bluetooth Low Energy, si erano raggiunti i 4 miliardi di dispositivi attivi[21]. Inoltre per gli anni a venire si prevede una sempre maggiore diffusione dello standard Bluetooth dato che oramai viene integrato su dispositivi mobile di qualsiasi tipo, a partire dagli smart watch, alle smart tv, fino agli oggetti intelligenti dell'IoT che popolano le nostre smart home e compongono le future smart city[22], motivo per cui tali stime tendono a crescere esponenzialmente di anno in anno prevedendo ora un aumento di altri 4 miliardi di dispositivi entro la fine del 2018.

Capitolo 2

La tecnologia Bluetooth

In precedenza è stata svelata la tecnologia trasmissiva su cui basare il middleware oggetto di tesi, per cui in questo secondo capitolo andremo a contestualizzare lo standard Bluetooth, a partire dalle sue origini passando per le diverse versioni rilasciate, analizzando i suoi punti di forza e le sue criticità, successivamente prenderemo in considerazione il suo funzionamento e la sua struttura protocollare. Infine scenderemo nel dettaglio dell'integrazione del Bluetooth con il sistema operativo Android, scorrendo brevemente le funzionalità che ci mette a disposizione l'API di Google. Così con le conoscenze premesse in questi primi due capitoli saremo in grado di addentrarci nella parte progettuale del lavoro di tesi che occuperà le successive sezioni.

2.1 Standard Bluetooth

Il Bluetooth è uno standard per la trasmissione dei dati per Wireless Personal Area Network (WPAN), ovvero per reti personali senza fili, sostituendo quest'ultimi con collegamenti radio a corto raggio. Come abbiamo precedentemente affermato oramai tantissimi dispositivi integrano lo stardard Bluetooth, tra cui auricolari, smartphone e computer, in quanto fornisce funzionalità economiche e sicure per scambiare informazioni tra device e a differenza di altri standard, come ad esempio il Wi-Fi è di minor impatto energetico e minor costo di produzione a discapito di un inferiore raggio di copertura e velocità di trasmissione. Come è stato già accennato nella sezione 1.4.1 e come approfondiremo nella 2.1.1, inizialmente il Bluetooth era stato ideato per connettere esclusivamente periferiche come mouse, tastiera, stampante e microfono al computer, ma con il passare degli anni ha ricevuto sempre più successo tanto da portare ad una continua evoluzione di questo standard, rilasciando nell'arco di venti anni circa una decina di versioni con continue migliorie e diffondendosi gradualmente in larga scala su miliardi di dispositivi.

Figura 2.1: Logo del protocollo Bluetooth

Da diversi anni con la crescente integrazione di questa tecnologia nei device di uso comune, il logo del Bluetooth, raffigurato in figura 2.1, è oramai conosciuto da tutti. Questo infatti unisce le lettere H e B dell'antico alfabeto utilizzato dalle popolazioni germaniche. Le due lettere stavano per Harald Blåtand (pronunciato Bluetooth in lingua inglese), abile sovrano diplomatico che fece riunire le varie popolazioni scandinave, al quale probabilmente gli inventori si ispirarono ritenendo che fosse un nome adatto per uno standard in grado di mettere in comunicazione dispositivi così diversi.

2.1.1 Storia e diffusione

Nel 1994 l'azienda svedese Ericsson avviò un programma di ricerca al fine di ideare un nuovo protocollo di comunicazione radio senza fili a basso consumo e con basso costo di produzione. L'idea di partenza era quella di integrare in ogni dispositivo un piccolo ricevitore radio per farli comunicare senza collegamenti fisici, ma ben presto fu surclassata dal progetto di creare una nuova tecnologia in toto. Così Ericsson capì subito la necessità di allargare il gruppo di sviluppo ad altre aziende, formando così nel Febbraio 1998 il Bluetooth Special Interest Group (SIG) insieme a Nokia, Intel, IBM e Toshiba. L'obbiettivo primario del gruppo d'innovazione era quello di sviluppare un nuovo standard wireless a corto raggio per far comunicare tra loro dispostivi di aziende diverse. Nel giro di pochi mesi iniziarono a prendere parte al SIG centinaia e centinaia di piccole e grandi realtà, tra cui 3COM, Microsoft e Motorola. Ad oggi, dopo circa 20 anni, come sostenuto durane una relazione interna al CES 2018 di Las Vegas, il Bluetooth Special Interest Group è composto da 33000 compagnie con il chiaro intento di perfezionare e aumentare la stabilità e flessibilità del protocollo. In particolare alla base del successo di questo standard di comunicazione c'è l'elevata interoperabilità tra i dispositivi e l'integrazione dei Profili, ovvero una specificazione sulla produzione dei device Bluetooth per interagire tra di loro, per contribuire a semplificarne lo sviluppo. Già dal 2000 furono messi in commercio prodotti che integravano funzionalità Bluetooth, da quel momento in poi la diffusione dello standard fu in continua crescita. Se infatti lo scenario applicativo primario era quello di collegare il telefono con gli auricolari o il computer con le periferiche esterne, al giorno d'oggi nella quasi totalità dei dispositivi mobili, wearable ed embedded troviamo integrato

questa tecnologia di comunicazione e la loro esponente diffusione porta ad una stima elevatissima di device Bluetooth, come riportano i dati di ABI Research, precedentemente riportati alla fine della sezione 1.4.1.

2.1.2 Caratteristiche tecniche

In venti anni di utilizzo il Bluetooth ha ricevuto continui aggiornamenti e migliorie, in termini di velocità di trasferimento, consumo di batteria e raggio di funzionamento, ma nonostante questo si basa sulle stesse linee guida della prima versione. In particolare i device utilizzano le frequenze libere ISM (Industrial, Scientific and Medical) intorno a 2,4 GHz (per la precisione da 2.402 GHz a 2.480 GHz)[1] suddividendo la banda in 79 canali a 1 MHz per le prime versione oppure in 40 canali con larghezza di banda a 2 MHz per ridurre le interferenze, utilizzando la Frequency Hopping Spread Spectrum (FHSS), tecnologia che permette di scambiare informazioni ad alta velocità variando la frequenza di trasmissione a intervalli regolari in maniera pseudo casuale accordata tra mittente e ricevitore, con una frequenza fino a 1600 volte al secondo.

Mentre per quanto riguarda la potenza massima di trasmissione e il raggio di funzionamento, questo dipende dalla classe del dispositivo, come mostra la tabella sottostante. Inoltre data la bassa potenza del segnale, ogni ostacolo fisico che blocca e/o riflette le onde radio porta ad una drastica diminuzione del raggio di copertura del segnale. Attualmente la maggior parte dei dispositivi in commercio sono di Classe 2, mentre nelle applicazioni del Bluetooth in ambito industriale sono altrettanto diffusi anche quelli di Classe 1.

Classe	**Potenza massima**	**Potenza massima**	**Portata**
Classe 1	100 mW	20 dBm	$\sim 100m$
Classe 2	2,5 mW	4 dBm	$\sim 10m$
Classe 3	1 mW	0 dBm	$\sim 1m$

Come già accennato, con il susseguirsi delle versioni il protocollo Bluetooth ha visto incrementare le velocità teorica di trasmissione dai dati come riporta la tabella 2.1.

[1] Radio Versions, `www.bluetooth.com/bluetooth-technology/radio-versions`

Versione	Transfer rate
1.0 - 1.0B - 1.1 - 1.2	1 Mbps
2.0 + EDR	3 Mbps
3.0 + HS	24 Mbps
4.0 + LE	1 Mbps
5.0	2 Mbps

Tabella 2.1: Versioni Bluetooth

Le migliorie apportate al Bluetooth non riguardano solamente il transfer rate ma anche l'aggiunta di nuove funzionalità per correggere gradualmente le problematiche emerse. In particolare nella versione 1.1 dello standard vengono sistemati i problemi riguardanti la sicurezza e l'interoperabilità tra dispositivi di diversi costruttori caratterizzanti le due precedenti release. Nella 1.2 viene aggiunta la Adaptive Frequency Hopping (AFH), tecnica che fornisce maggior resistenza alle interferenze elettromagnetiche, evitando l'utilizzo dei canali soggetti a forti interferenze, la extended Synchronous Connections (eSCO) che garantisce una modalità di trasmissione audio di alta qualità, ritrasmettendo i dati qualora questi vengano perduti e la funzionalità di rilevatore della qualità del segnale. La versione 2.0 è caratterizzata dall'intruduzione dell'Enhanced Data Rate (EDR) che incrementa la velocità di trasmissione fino a 3 Mbps, riducendo i tempi di risposta e dimezzando la potenza utilizzata grazie all'impiego di segnali radio di minore potenza. Lo standard 3.0 + HS è caratterizzato appunto dall'High Speed ottenuta grazie al decentramento del Bluetooth, trasferendo grande moli di dati attraverso l'eccezionale apertura e chiusura di connessioni Wi-Fi. Viene inoltre migliorata la sicurezza del trasferimento dei dati attraverso un periodico cambio di password per collegamenti criptati e grazie al miglioramento nel controllo dei bit di parità. Inoltre viene introdotto l'Extended Inquiry Response che prevede lo scambio di più informazioni durante la procedura di richiesta di collegamento in modo da permettere un miglior filtraggio dei dispositivi prima di effettuare la connessione, tra queste informazioni ci sono il nome del dispositivo e una lista di servizi. Nel 2010 viene presentato il tanto atteso Bluetooth Low Energy (BLE) che rispetto alle versioni precedenti, ha come obbiettivo principale la riduzione dei consumi energetici e l'aggregazione dei dati provenienti da diversi sensori, tramite un'ottimizzazione della struttura comunicativa e l'impiego di dispositivi più efficienti, ma a discapito della velocità, che in questa modalità si riduce a 1 Mbps. Infine lo standard più recente, presentato nel Giugno 2016, è il 5.0 e incrementa di quattro volte l'area di trasmissione, mantenendo lo stesso livello di consumo per connettere spazi di grandi dimensioni, raddoppia la velocità

fino a 2 Mbps in modalità a basso consumo e aumenta otto volte tanto la capacità di trasmissione, inoltre gestisce in maniera più efficientemente la banda 2,4 GHz, andando ad individuare ed eliminare le interferenze.

La rete Bluetooth base viene chiamata piconet, basata su architettura master-slave, all'interno della quale un dispositivo alla volta può comunicare con il master che si occupa della sincronizzazione della comunicazione grazie al suo clock. Inoltre il master è in grado di gestire simultaneamente la comunicazione con altri 7 dispositivi slave. È possibile connettere tra loro più piconet per formare una scatternet, in particolare gli slave possono appartenere a piu piconet contemporaneamente mentre il master di una piconet puo al massimo essere lo slave di un'altra.

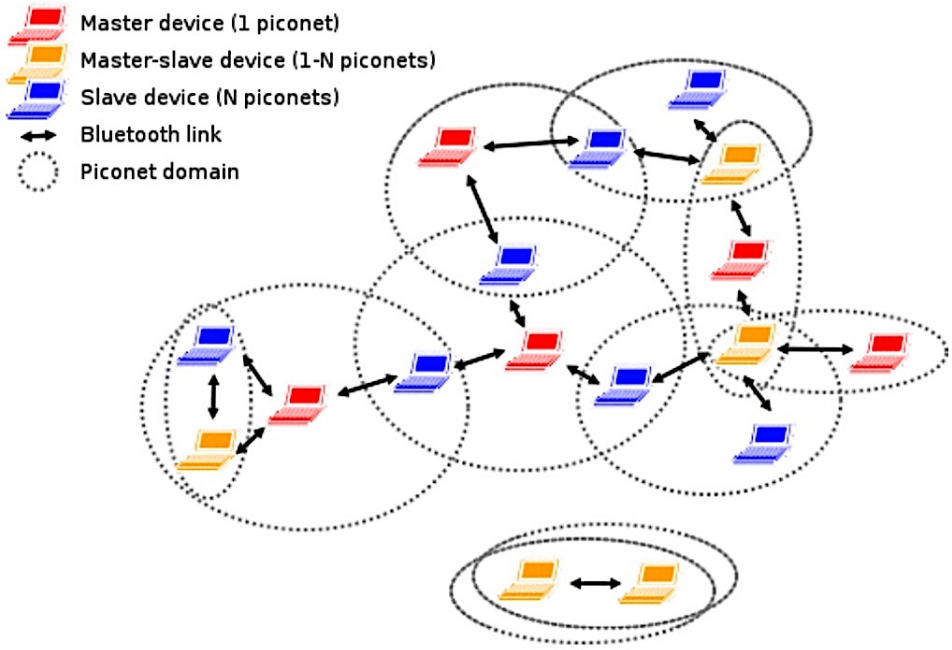


Figura 2.2: Tipologie di reti Bluetooth

In generale ci sono due tipi di collegamenti quelli orientati alla connessione e quelli senza connessione. Il primo richiede di stabilire una connessione tra i dispositivi prima di inviare i dati, il secondo non richiede alcuna connessione prima di inviare i pacchetti, ma il trasmettitore in qualsiasi momento può inviare i propri pacchetti purche conosca l'indirizzo del destinatario. In conclusione la tecnologia Bluetooth definisce due tipi di collegamenti, un servizio asincrono senza connessione (ACL, Asynchronous ConnectionLess) per traffico di tipo dati, servizio di tipo best-effort e un servizio sincrono orientato alla

connessione (SCO, Synchronous Connection Oriented) per il traffico di tipo real-time e multimediale.

2.1.3 Bluetooth Protocol Stack

Così come avviene per l'architettura ISO/OSI anche lo standard Bluetooth è organizzato a livelli. Indipendentemente del tipo di applicazione, lo stack protocollare include sempre i livelli data-link e fisico. Analogamente non tutte le applicazioni utilizzano tutti i livelli dello Bluetooth Protocol Stack, in quanto esso viene rappresentato su più livelli verticali (vedi figura 2.3), al di sopra dei quali c'è la specifica applicazione.

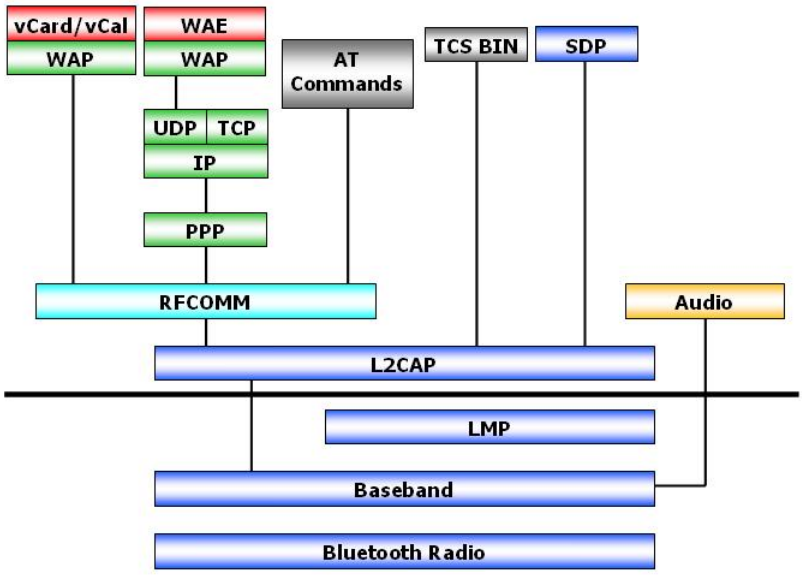


Figura 2.3: Schema del Bluetooth Protocol Stack

È inoltre possibile ricondurre a ciascun livello le sue specifiche funzionalità:

- **Bluetooth Radio**: qui è dove vengono definiti i requisiti radio e processati i segnali
- **Baseband**: abilita il collegamento fisico tra dispositivi all'interno di una piconet. Permette inoltre di stabilire i due diversi tipi di connessione (ACL e SCO).
- **LMP**: *Link Management Protocol* è responsabile dell'organizzazione del collegamento e della negoziazione della dimensione dei pacchetti scambiati. Si occupa inoltre di autenticazione e crittografia, in particolare della generazione, scambio e controllo chiavi.

- **L2CAP**: *Logical Link Control and Adaptation Protocol* utilizzato per fare il multiplexing dei protocolli di livello superiore, la segmentazione e il riassemblaggio dei pacchetti e il trasporto di informazione relativa alla QoS (Quality of Service). L2CAP permette ai protocolli dei livelli superiori ed alle applicazioni di trasmettere e ricevere pacchetti di dati di dimensione superiore a 64 KB.

- **RFCOMM**: emula una tradizionale porta seriale RS-232 sul protocollo L2CAP.

- **SDP**: *Service Discovery Protocol* permette alle applicazioni di avere informazioni sui dispositivi, sui servizi offerti e sulle caratteristiche dei servizi disponibili.

- **AUDIO**: strato responsabile della codifica audio

  Sopra i layer appena elencati ci sono gli *Adopted Protocol*, ovvero protocolli definiti da altre organizzazioni di standardizzazione e incorporati nell'architettura Bluetooth tra cui PPP, lo standard Internet per trasportare pacchetti IP su una connessione punto a punto, TCP/UDP-IP (le fondamenta della suite TCP/IP), OBEX (object exchange) un protocollo a livello sessione sviluppato dalla Infrared Data Association per lo scambio di oggetti, simile all'HTTP ma piu semplice e WAE/WAP, Wireless Application Environment e Wireless Application Protocol.

  In conclusione per facilitare l'utilizzo dei dispositivi Bluetooth la Special Interest Group ha introdotto una serie di profili che identificano una vasta gamma di possibili applicazioni tra cui: GenericAccessProfile (GAP), CordlessTelephonyProfile (CTP), SerialPortProfile (SPP), HeadsetProfile (HSP), Dial-up Networking Profile (DUNP), FaxProfile, ObjectPush-Profile (OPP), FileTransferProfile (FTP) e SynchronisationProfile(SP).

2.2 Bluetooth e Android

Nell'ottobre 2003 Andy Rubin, Rich Miner, Nick Sears e Chris White fondarono la Android Inc con l'obbiettivo di sviluppare software per dispositivi mobile, pochi mesi dopo nel 2005 la società fu acquistata da Google. La prima versione del sistema operativo di Palo Alto fu rilasciata il 23 Settembre 2008 e fu integrato sull'HTC Dream. Il sistema anche tutt'ora viene continuamente aggiornato al fine di aumentarne le funzionalità supportate e accrescere la sua diffusione. In particolare nel 2009 con la release 2.0 **Eclair** (SDK API level 5) venne introdotto il supporto allo standard Bluetooth. In Android 4.3 **Jelly**

Bean (SDK API level 18) nel 2013 vennero integrate anche le librerie per il Bluetooth Low Energy, protocollo presentato qualche anno prima dalla SIG, principalmente allo scopo di trasferire piccoli pacchetti di dati tra dispositivi vicini e per l'interazione con i Beacon. Infine con l'introduzione nell'estate del 2016 del nuovo Bluetooth 5.0, Android ha dovuto prontamente inserire nella versione 8.0 **Oreo** di fine 2017 (SDK API level 26) il supporto a questa nuova tecnologia.

2.2.1 API Android

Dato che tra i requisiti nella sezione 1.4.1 è stato definito l'utilizzo del Bluetooth Classic, di seguito andremo ad analizzare le caratteristiche dell'API che Android mette a disposizione per tale tecnologia, grazie alle quali è possibile effettuare le seguenti operazioni:

- scansione per la ricerca di altri dispositivi
- richiedere la lista di dispositivi accoppiati
- stabilire canali RFCOMM
- connettersi ad altri dispositivi
- scambiare dati da e verso altri device
- gestire connessioni multiple

In primis per utilizzare le funzionalità Bluetooth è necessario andare ad inserire nel file manifest del progetto due permessi come indicato di seguito. Il primo serve per completare qualsiasi operazione Bluetooth, a partire dalla richiesta di connessione fino alla comunicazione vera e propria. Il secondo è necessario per rendere il dispositivo rilevabile, in quanto questa è un'operazione ad alto impatto energetico. Mentre nei dispositivi che hanno versione 6.0 (SDK API level 23) o superiore del sistema operativo è necessario inserire un ulteriore permesso, in particolare se ne deve aggiungere uno, tra il terzo e il quarto tra quelli riportati nel listato 2.1, che abilita la localizzazione del dispositivo in quanto potrebbe essere utilizzato in scenari applicativi in cui le scansioni Bluetooth vengono utilizzate per la raccolta di informazioni sulla posizione dell'utente, come avviene ad esempio con l'utilizzo dei Beacon disposti nei negozi o per le strade.

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

Listato 2.1: Permessi AndroidManifest.xml

In particolare l'API per gestire la tecnologia Bluetooth mette a disposizione le seguenti classi, disponibili nel package `android.bluetooth`.

- `BluetoothAdapter`: se ne ottiene un istanza con l'invocazione il metodo `BluetoothManager.getDefaultAdapter()`, punto di partenza fondamentale per tantissime operazioni in quanto rappresenta il dispositivo Bluetooth locale sul quale l'applicazione è in esecuzione.

- `BluetoothDevice`: invocando `BluetoothAdapter.getRemoteDevice(MAC)` si ha una rappresentazione di un dispositivo Bluetooth remoto con un determinato indirizzo MAC, con il quale è possibile instaurare una connessione o richiedere informazioni su di esso, ad esempio nome, indirizzo, classe e stato del collegamento.

- `BluetoothSocket`: viene utilizzata nei client prima per effettuare una richiesta di connessione al server e poi per scambiare informazioni con quest'ultimo. Dall'istanza di questa classe con i metodi `getOutputStream()` e `getInputStream()` è possibile ottenere l'OutputStream e l'InputStream relativi alla socket.

- `BluetoothServerSocket`: viene utilizzata nel server per accettare richieste di connessione in entrata grazie al metodo `accept()` che a sua volta ritorna delle socket per la gestione del collegamento.

Prima di iniziare il vero e proprio scambio di messaggi, i due dispositivi devono instaurare un canale di comunicazione, mediante un processo di *pairing*. Un device rilevabile si rende disponibile, con il metodo `startDiscovery()`, per accettare connessioni in entrata. Nel momento in cui questo viene trovato da un secondo dispositivo in modalità *discovery*, viene avviata una richiesta di accoppiamento e una volta accettata da entrambi, i due dispositivi possono finalmente scambiarsi informazioni.

Per collegare due device è necessario implementare entrambi i meccanismi lato client e lato server, il primo deve avviare la connessione utilizzando l'Universally Unify Identiier (UUID) concordato con il server, passandolo al metodo `listenUsingRfcommWithServiceRecord(UUID)`, mentre quest'ultimo deve aprire una ServerSocket con lo stesso UUID e attendere le richieste in ingresso `listenUsingRfcommWithServiceRecord(String, UUID)` . Tutt'e due, in maniera differenti, ottengono un'istanza della BluetoothSocket attraverso la quale possono dare inizio al flusso di input e di output avviando così il trasferimento dei dati. Al server viene ritornata la socket nel momento in cui viene accettata la connessione in entrata, mentre al client viene restituita la socket nel momento in cui apre un canale RFCOMM sul server. Proprio per questo sono considerati collegati l'uno all'altro quando hanno una BluetoothSocket connessa sullo stesso canale RFCOMM. Questi rimangono collegati in modo da potersi riconnettere automaticamente purché restino nel raggio di funzionamento oppure finché uno dei due non rimuove l'accoppiamento.

Capitolo 3

Un middleware di comunicazione e coordinazione su Bluetooth

Nel terzo capitolo andremo a specificare nel dettaglio l'obbiettivo e i requisiti che il middleware dovrà soddisfare, chiarendo volta per volta gli scenari esemplificativi e le rispettive motivazione che hanno portato a tali richieste. Per quanto riguarda il sistema operativo e la tecnologia trasmissiva su cui l'API si dovrà basare è stato precedentemente trattato nell'ultima sezione del primo capitolo, durante la presentazione del layer di comunicazione.

3.1 Analisi dei requisiti

Emerge chiaramente da tutto quello che è stato premesso e approfondito nelle pagine precedenti, la necessità di un middleware per la comunicazione e il coordinamento di alto livello. Come è stato già detto il suo obiettivo primario consiste nel rendere agevole l'interconnessione e lo scambio di messaggi tra più dispostivi che si trovano in una ristretta area, andando così ad abbracciare il concetto più esteso delle Wireless Body Area Network, precedentemente definito, in quanto vorremmo collegare qualsiasi tipo di dispositivo mobile, embedded e wearable, quindi non solamente sensori e attuatori, ma in generale tablet, sensori, smart glass, smartphone, ecc. Tutto questo proprio perché con il procedere degli anni ci siamo ritrovati a vivere con un numero sempre più grande di device che potrebbero avere la necessità di interagire, di scambiarsi informazione oppure di richiedere funzionalità attraverso dei specifici comandi. Questo avviene da un lato in scenari professionali, dall'altro quotidianamente a tutti noi, l'esempio che rimbalza subito agli occhi anche se è stato già citato è il diffuso monitoraggio dell'attività fisica durante la quale, dopo aver avviato l'allenamento da smartphone, attraverso il fitness tracker si monitorano i passi,

le calorie bruciate, i chilometri percorsi e tanti altri dati, mentre con le cuffie senza fili si ascolta la musica e si effettuano chiamate.

Data la crescente diffusione di dispositivi intelligenti questo era solamente uno degli scenari possibili dove si ha la necessità di integrare più device tra di loro, ma per andare al centro della questione la particolarità sarà quella di realizzare un API di alto livello per la comunicazione tra dispositivi wearable e mobile in modo da essere impiegata su più device possibile, motivo per cui si è scelto Android come sistema operativo di riferimento, essendo quest'ultimo il più diffuso nei due settori appena indicati (vedi grafico di figura 1.6). Inoltre tra le tecnologie di comunicazione wireless, dopo un attenta analisi, ci si è appoggiati sul Bluetooth Classic in quanto rappresenta il miglior compromesso tra stabilità, efficienza, consumo di energia e soprattutto diffusione in ambito Mobile, Wearable ed Embedded. Altra peculiarità del layer di comunicazione è la scelta di supportare tre diversi modelli per lo scambio di messaggi, in particolare la send/receive, il publish/subscribe e lo spazio delle tuple al fine di rendere l'interazione agile e flessibile. Tutto quanto è stato introdotto nelle righe precedenti verrà analizzato dettagliatamente nelle sezioni successive, inoltre nella sezione 3.1.4 verranno introdotti altri requisiti non funzionali, che dovranno essere tenuti in considerazione durante la progettazione e la realizzazione del middleware.

In conclusione è necessario sottolineare che avendo come obbiettivo del middleware la sua diffusione su più dispositivi e quindi lo scambio di messaggi tra di questi, si deve puntualizzare che si sta prendendo in considerazione il problema della progettazione e dello sviluppo su Wireless Body Area Networks e quindi su sistemi distribuiti ovvero device distinti. Per questo motivo principalmente si deve fare attenzione alle seguenti criticità: concorrenza, malfunzionamenti, funzionamenti diversi in dispositivi diversi, mancanza di uno stato globale che indichi lo stato del sistema.

3.1.1 Modello a scambio di messaggi

Secondo questo pattern la comunicazione tra dispositivi deve avvenire tramite invio e ricezione di messaggi, deve inoltre essere asincrona e indiretta. Le primitive di questo modello sono:

- `send(destId, Msg)`
- `receive(Callback):Msg`

La comunicazione, essendo tra dispositivi distinti, deve essere indiretta in quanto il flusso di messaggi deve passare attraverso il canale trasmissivo che funge anche da coordinatore della rete, mentre con asincrona si vuole specificare che la `send` ha successo ed è completata quando il messaggio è stato

inviato, ma non necessariamente ricevuto dal destinatario. Inoltre la `receive` deve essere bloccante, definendo una callback da attivare non appena arriva un messaggio da processare nella mailbox. Chiaramente l'invio di messaggi per uno specifico destinatario deve avvenire solamente nel caso in cui questo sia collegato alla rete, altrimenti il messaggio deve essere bufferizzato in attesa di una successiva ri-connessione del dispositivo. Invece nel momento in cui il messaggio viene ricevuto dal device, questo deve essere reso persistente finché non viene consumato. Uno scenario reale in cui potrebbe esser applicato questo modello di comunicazione è sicuramente la sincronizzazione tra un fitness tracker e uno smartphone, quest'ultimo attende attraverso la `receive` di ricevere le informazioni non appena l'altro dispositivo si connette e lo aggiorna con una `send` contenente gli ultimi dati rilevati.

3.1.2 Modello publish/subscribe

Questo modello a scambio di messaggia viene tipicamente utilizzato per strutturare applicativi ad eventi ad alto livello, infatti è molto diffuso in ambito Internet of Things. Il meta modello si compone delle seguenti entità:

- **channel**: canale su cui è possibile inviare messaggi e per il quale è possibile riceve notifiche mediante sottoscrizione
- **publisher**: pubblica messaggi sui canali
- **subscriber**: si registra su uno o più canali al fine di ricevere tutti i messaggi che vengono pubblicati su di questi

Le primitive di questo modello di comunicazione sono:

- `publish(Msg, Channel)`
- `subscribe(Channel)`

Inoltre devono essere rese persistenti le sottoscrizioni dei dispositivi, così da ottimizzare il funzionamento del middleware in caso di malfunzionamenti e disconnessioni improvvise. Il pattern publish/subscribe in applicazioni reali potrebbe essere necessario ad esempio in ambito sanitario per il monitoraggio del battito cardiaco di un paziente attraverso specifici dispositivi, i quali rilevano e inviano i dati in tempo reale ad un altro device che attraverso una GUI li mostra al personale medico. Tutto questo avviene mediante la `subscribe` del dispositivo di interfaccia al canale dei battiti cardiaci sul quale parallelamente il rilevatore effettua la `publish` dei parametri real-time.

3.1.3 Modello di coordinazione: spazio delle tuple

Prima di tutto è necessario introdurre Linda, che è un linguaggio di coordinamento che estende i linguaggi tradizionali e che è indipendente dall'architettura sottostante, permettendone l'utilizzo in applicazioni in ambienti distribuiti. Il concetto alla base di Linda è l'esistenza di uno spazio delle tuple, una sorta di memoria globale condivisa in cui è possibile inserire, leggere e rimuove una tupla o in generale un messaggio. Per chiarire ulteriormente potremo paragonare lo spazio delle tuple ad una lavagna e le singole tuple a post-it che vengono attaccati, letti e rimossi da questa. Nello specifico le primitive di questo modello di comunicazione sono:

- `out(Tuple)`
- `in(Template):Tuple`
- `in(Template, Callback):void`
- `rd(Template):Tuple`
- `rd(Template, Callback):void`

Innanzitutto è necessario introdurre il funzionamento generale delle principali operazioni di questo pattern di comunicazione, con la `out` viene inserita una nuova tupla nello spazio delle tuple, con `in` viene restituita e rimossa una tupla che soddisfa il template passato per parametro, mentre `rd` ha lo stesso comportamento di `in` con la differenza che la tupla selezionata non viene rimossa dalla memoria. Si deve specificare che ci sono due versioni delle operazioni di in e di read, quelle bloccanti o sincrone `in` e `rd` e quelle non bloccanti o asincrone `inp` e `rdp`. Le prime sospendono il processo fino alla ricezione della tupla richiesta, mentre le seconde definiscono una callback che prende in input una tupla e la consumano restituendo void.

Se al momento dell'invocazione di `in` o `rd` nello spazio delle tuple non c'è nessuna tupla che soddisfa il template passato come parametro, la richiesta deve essere mantenuta in memoria per essere soddisfatta il prima possibile e poiché queste chiamate sono asincrone richiedono la definizione di una callback che verrà poi invocate non appena si riceve una tupla. Inoltre nel caso in cui ci fossero richieste pendenti per un determinato template, in seguito all'inserimento di una nuova tupla che lo soddisfa, prima vengono accontentate tutte le `rd` e poi nel caso fosse presente una `in`. Infine lo spazio delle tuple deve essere persistente al fine non perdere messaggi in caso di disconnessioni improvvise.

3.1.4 Requisiti non funzionali

In questa sezione andremo a definire ulteriori specifiche che il sistema dovrà soddisfare. In particolare verranno definiti altri requisiti non funzionali, cioè proprietà del sistema che dovranno essere rispettate e tenute in considerazione durante le successive fasi di progettazione e implementazione.

- **sessione di connessione**: si vuole estendere lo scambio di dati e il collegamento dei dispositivi in sessioni di connessione, caratterizzate da tre fasi: connessione, lavoro in sessione e disconnessione. La prima prevede una join da parte del dispositivo slave al master, così da potersi unire alla rete. La seconda consiste nel dialogo vero e proprio tra dispositivi, secondo i pattern introdotti nelle sezioni 3.1.1, 3.1.2 e 3.1.3. Infine la terza concerne la chiusura della connessione da parte di uno dei lati della connessione.

- **configurabilità del sistema**: l'API deve permettere agli utilizzatori di determinare il numero di dispositivi della rete, la durata del processo di discovery[1] e il tempo massimo di visibilità[2] dei device, al fine di rilasciare prima le risorse così da ottimizzarne il funzionamento.

- **agilità nella connessione**: nel momento in cui si perde accidentalmente la connessione o se ne vuole aprire una nuova, il middleware deve rendere veloci tali operazioni per non rallentare il flusso della comunicazione.

- **reattività**: il middleware deve garantire un alta capacità di reagire ai cambiamenti, agli eventi esterni e agli input. Viene richiesta particolare attenzione nel caso di perdita della connessione e conseguente ri-connessione.

- **robustezza**: al presentarsi di situazioni impreviste, come ad esempio errori, interferenze, dati di input scorretti, il sistema deve comportarsi in maniera ragionevole garantendo sempre la connessione.

- **resilienza**: l'API deve essere capace di adattarsi alle condizioni d'uso e di resistere all'usura in modo da garantire la disponibilità dei servizi erogati.

[1] processo ad alto impatto energetico attraverso il quale un dispositivo avvia la ricerca di altri segnali radio Bluetooth nelle vicinanze, vedi sezione 2.2.1

[2] processo ad alto impatto energetico attraverso il quale si permette ad altri dispositivi di trovare il proprio segnale Bluetooth, vedi sezione 2.2.1

3.1.5 Casi d'uso

Nel diagramma di figura 3.1 vengono riportati i principali casi d'uso emersi dell'analisi dei requisiti definiti nelle sezioni precedenti, dal quale emerge che l'attore principale del sistema è il *dispositivo*. Questi daranno l'idea dello scenario in cui verrà applicato il middleware e guideranno l'intero progetto di sviluppo, costituendo il punto di partenza per la successiva fase di progettazione e implementazione.

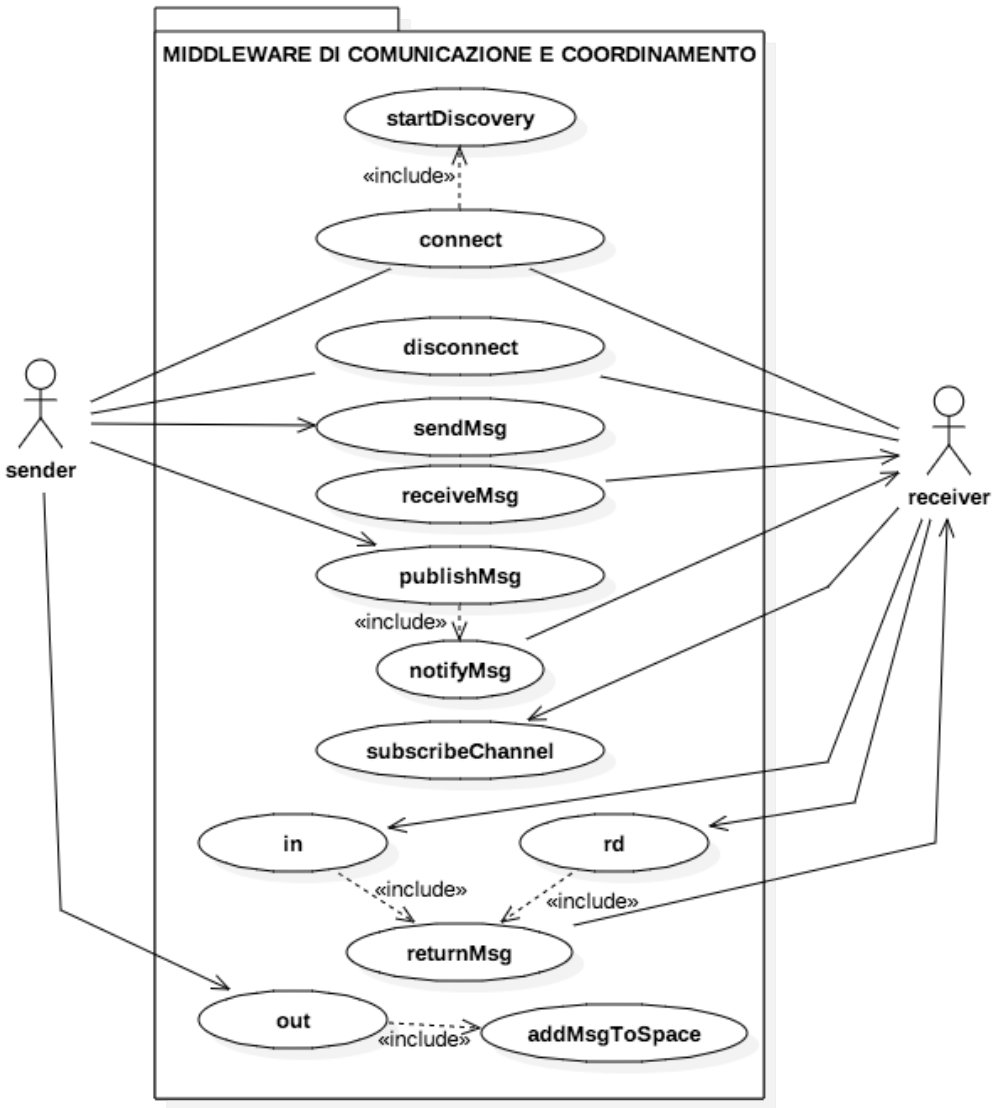


Figura 3.1: Diagramma dei casi d'uso

3.2 Ricognizione framework esistenti

Dopo aver riconosciuto la necessità di integrazione tra più dispositivi che compongono le WBAN e aver delineato le caratteristiche che deve avere il layer di comunicazione è stato necessario, prima di intraprendere la fase di progettazione e implementazione dell'API, effettuare una ricognizione delle librerie open source attualmente disponibili online. In rete l'argomento dell'integrazione e della comunicazione tra più dispositivi è molto discusso, tanto che sono presenti svariate librerie che si propongono di semplificare tali operazioni. Le più interessanti che sfruttano lo standard Bluetooth nelle versioni Classic e Low Energy sono le seguenti:

- **RxBluetooth**[3]: racchiude le funzionalità messe a disposizione dal Bluetooth Adapter con l'integrazione della libreria RxJava
- **Blueteeth**[4]: prende ispirazione dalla facilità di utilizzo di LGBluetooth su iOS e si pone l'obiettivo di realizzare una libreria semplice e leggere finalizzata ad eliminare le criticità nell'utilizzo dell'API BLE di Android
- **RxBluetoothAuto**[5]: combina l'API Android con la libreria RxJava per fornire funzionalità di connessione e auto connessione nel caso di perdita della connessione
- **NeatLE**[6]: gestisce e monitora le connessione Bluetooth Low Energy.
- **EasyBluetooth**[7]: permette la facile creazione di un canale trasmissivo bluetooth tra due dispositivi
- **Android Multi Bluetooth Library**[8]: consente di collegare al master fino a sette dispositivi e di fargli scambiare messaggi attraverso delle socket

Dopo aver attentamente studiato la documentazione e il codice sorgente delle diverse librerie è possibile affermare che queste permettono il parziale funzionamento delle classiche operazioni dello standard Bluetooth più o meno agilmente (discovery, connessione, scambio di messaggi, disconnessione), inoltre allo stesso tempo emerge chiaramente che nessuna di queste si è posta l'obbiettivo di realizzare un middleware, o una framework, di alto livello

[3]RxBluetooth, `https://github.com/IvBaranov/RxBluetooth`
[4]Blueteeh, `https://github.com/RobotPajamas/Blueteeth`
[5]RxBluetoothAuto, `https://github.com/tawaasalage/RxBluetoothAuto`
[6]NeatLE, `https://github.com/inovait/neatle`
[7]EasyBluetooth, `https://github.com/NewtronLabs/EasyBluetooth`
[8]AMBL, `https://arissa34.github.io/Android-Multi-Bluetooth-Library`

completo per la comunicazione e il collegamento di più dispositivi, che sarebbe utile per qualsiasi progetto a prescindere dal contesto, che si stia parlando di Android o di Wireless Body Area Networks. Proprio per questo motivo si è voluto procedere in tale direzione per lo sviluppo del lavoro di tesi e ancor più per rendere il framework disponibile alla comunità dell'open source.

Capitolo 4

Progettazione architetturale

A partire dai casi d'uso del paragrafo 3.1.5 e dopo aver esaminato attentamente le librerie open source citate nella sezione 3.2, emerge subito la necessità di utilizzare per il middleware la topologia di rete a stella. In particolare al centro di questa avremo un dispositivo server (chiamato anche master o hub) collegato a tanti client (slave o host), così ogni device potrà comunicare con tutti gli altri grazie al funzionamento da coordinatore della rete del nodo centrale, attraverso il quale vengono veicolati tutti i messaggi ai rispettivi destinatari, riducendo in questo modo l'impatto di un guasto sulla connessione, essendo collegato in modo indipendente ciascun client al master. In questo modo il malfunzionamento di un collegamento tra un host e l'hub determinerà, l'isolamento di tale host da tutti gli altri, ma il resto della rete continuerà a funzionare senza interruzioni.

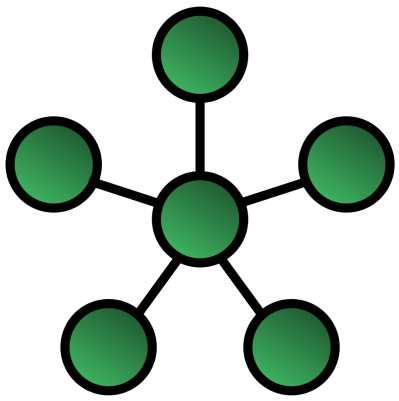


Figura 4.1: Topologia di rete a stella

Il *server*, come è stato già detto, è l'entità che funziona da coordinatore all'interno della rete, prima di tutto si prende cura di definire il massimo numero di device slave supportati, dopo di che si mette in attesa di ricevere connessioni in entrate da parte dei client. Una volta che viene stabilita la connessione con almeno un dispositivo, questo può iniziare a ricevere, inviare e indirizzare i messaggi ricevuti. Infatti potremo dire che il master estende il device slave, in quanto entrambi hanno le medesime funzionalità per quanto riguarda la parte di scambio di comunicazioni.

Mentre il *client* è l'entità dalla quale parte la richiesta di connessione verso il master e una volta che questa viene accettata, si stabilisce un canale diretto di comunicazione tra i due dispositivi sul quale inviare i dati.

Inoltre dalla precedente fase di analisi è stato possibile suddividere il funzionamento del middleware in tre macro funzioni: l'instaurazione della connessione, lo scambio di messaggi e la disconnessione, che approfondiremo nelle sezioni successive.

La *Android Multi Bluetooth Library*[1], analizzata durante la ricognizione dei framework open source esistenti, gestisce le connessioni tra più dispositivi sulla falsa riga di quanto premesso finora. Motivo per il quale si è deciso di prenderla come spunto di partenza per le successive fasi di progettazione e implementazione. Questa infatti è strutturata su un'architettura di tipo client server e permette lo scambio di array di byte, stringhe e oggetti, dallo slave al master e broadcast dal master verso tutti gli slave. Nonostante questo necessita l'integrazione di nuove funzionalità, tra cui la connessione automatica e il supporto ai tre modelli di comunicazione e la risoluzione di diverse criticità, in primis la gestione delle disconnessioni.

In conclusione nelle sezioni successive analizzeremo le singole funzionalità, aiutati dai diagrammi di sequenza che specificheranno il comportamento delle singole entità all'interno del middleware.

4.1 API

Prima di entrare nel dettaglio della fase di progettazione e di implementazione è necessario definire le API messe a disposizione dal middleware per l'utilizzatore. Obbiettivo di questa fase di progettazione sarà quello di nascondere il più possibile le questioni di basso livello e del layer trasmissivo al programmatore che ne fa utilizzo.

Nello specifico verranno fornite le seguenti funzionalità:

[1] https://arissa34.github.io/Android-Multi-Bluetooth-Library

procedura	**descrizione**
createClient(serverId)	metodo bloccante che restituisce il controllo una volta che è stata instaurata la connessione con il server
createServer(clientId)	metodo bloccante che restituisce il controllo una volta che è stata accettata la richiesta di connessione del client
scanAllBluetoothDevice()	metodo che restituisce una lista con gli identificativi dei dispositivi rilevati nelle vicinanze
startDiscovery()	metodo per avviare l'operazione di discovery dei dispositivi nelle vicinanze
startDiscoverability()	metodo per rendere il dispositivo rilevabile dagli altri
disconnect()	metodo per dissociarsi dalla rete
sendMsg(target, msg)	metodo per inviare al dispositivo target un messaggio con il contenuto passato come secondo parametro
receiveMsg(callback)	metodo bloccante che attende la disponibilità di un messaggio da consumare nella mailbox
publish(msg, eventType)	metodo per generare un evento con il contenuto passato come primo parametro del tipo passato come secondo parametro
subscribe(eventType, sub)	metodo per sottoscriversi o annullare la sottoscrizione (in base al secondo parametro sub) al tipo di evento passato come primo parametro
in(template, callback)	metodo non bloccante per effettuare una richiesta di tipo *in* allo spazio delle tuple attraverso il template pssato come primo parametro, definendo una callback da eseguire alla ricezione della risposta
rd(template, callback)	metodo non bloccante per effettuare una richiesta di tipo *read* allo spazio delle tuple attraverso il template passato come primo parametro, definendo una callback da eseguire alla ricezione della risposta
out(msg, template)	metodo per effettuare la pubblicazione di una tupla contenente il messaggio passato come primo parametro e che soddisfa il template passato come secondo parametro

Tabella 4.1: API messe a disposizione dal middleware

4.2 Connessione e disconnessione

Il diagramma di sequenza di figura 4.2 verrà utilizzato per spiegare la fase di progettazione dell'instaurazione della connessione, il modello di comunicazione a scambio di messaggi (trattato nella sezione 4.3.1) e le disconnessioni. Per cui in questa sezione prenderemo in considerazione la prima e l'ultima parte di diagramma in quanto la connessione e la disconnessione tra dispositivi client e server sono le medesime a prescindere dal pattern utilizzato.

Per prima cosa analizzando la fase di connessione (indicata nel diagramma dalla nota 'connection phase') è possibile sottolineare che da un lato il client deve aver avviato l'attività di discovery mentre dall'altro il server deve essersi reso individuabile, così che il primo può inviare una richiesta di connessione a quest'ultimo, il quale non appena la riceve apre una sessione mediante la quale i dispositivi potranno scambiarsi le informazioni. Attraverso questo schema viene delineato il comportamento del sistema in tutti i possibili casi di connessione, infatti il client prima di effettuare la richiesta deve selezionare il server, questa operazione può avvenire in maniera diretta andando a selezionare di volta in volta il server a cui collegarsi, oppure renderla automatica andando a leggere e scrivere un determinato parametro.

In conclusione da quanto è rappresentato nel diagramma e da quello che è stato appena detto è evidente che la richiesta di connessione è unilaterale, ovvero parte solo ed esclusivamente dal client, in nessun modo questa può scaturire dal server che in questa fase è solamente passivo, aspettando le connessioni in ingresso.

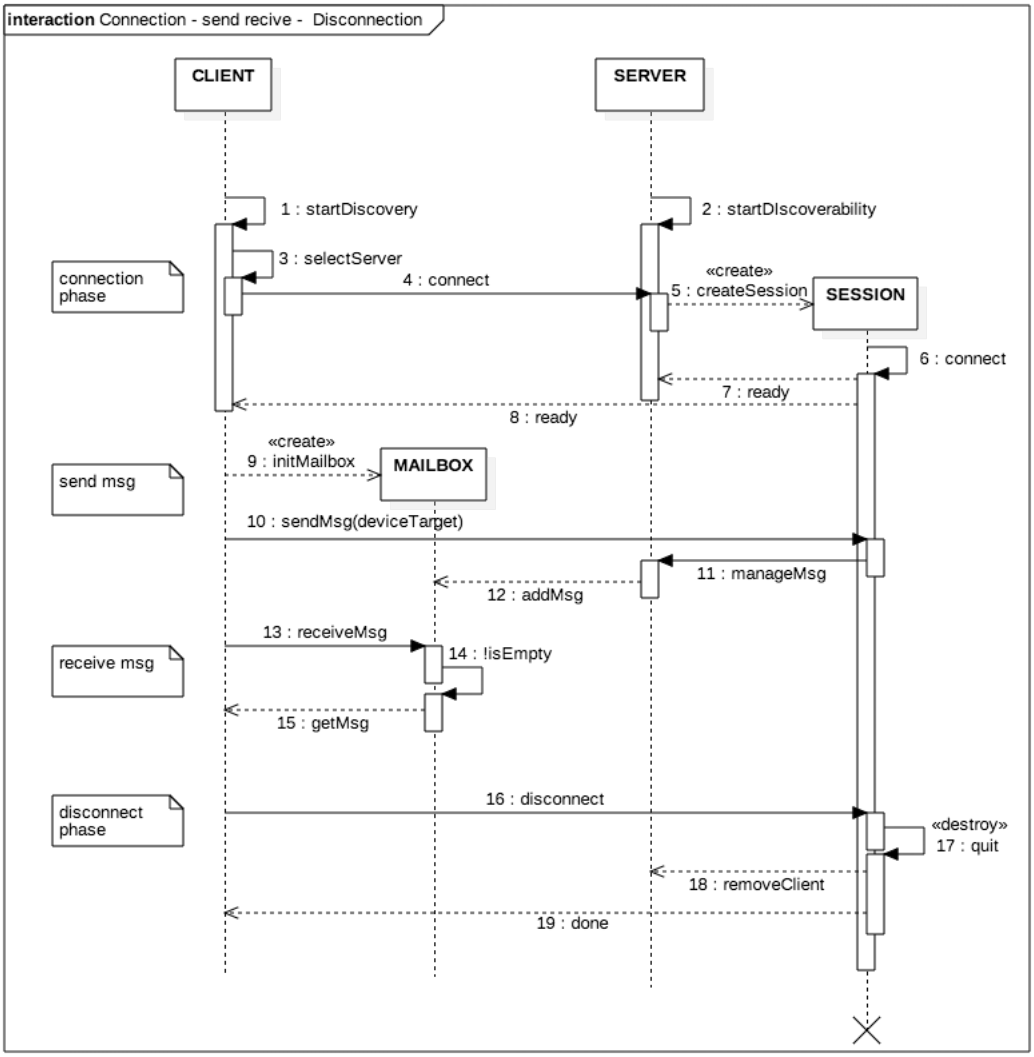


Figura 4.2: Diagramma di sequenza rappresentante le operazioni di connessione, disconnessione e modello di comunicazione a scambio di messaggi

Mentre per quanto riguarda la fase di disconnessione (indicata nel diagramma dalla nota 'disconnection phase') questa può essere avviata da entrambi i lati della connessione, per non ripetere le stesse interazioni due volte nel diagramma viene rappresentata solamente la disconnessione da parte del dispositivo slave, ma nel caso in cui sia il master a interrompere la connessione il comportamento è analogo; ovvero il dispositivo segnala la volontà di disconnettersi, non appena la sessione riceve tale messaggio lo comunica al server e lo conferma al client (viceversa nel caso in cui sia il server a interrompere la

connessione), che rilasciano rispettivamente le risorse impiegate.

4.3 Modelli di comunicazione

Per analizzare al meglio i diversi modelli di comunicazione nelle sezioni successive verranno riportati distinti diagrammi di sequenza, eccetto per il pattern a scambio di messaggi che era incluso nel grafico di figura 4.2. Inoltre non verranno riportate tutte le volte le fase di connessione e disconnessione, ma verranno raffigurate solamente le interazioni proprie dei singoli modelli, chiarendo che tutte queste potevano essere raffigurate in un unico schema in quanto la progettazione del middleware prevede che il funzionamento dei tre modelli sia indipendente l'uno dall'altro ed è inoltre possibile l'utilizzo simultaneo di questi.

4.3.1 Pattern a scambio di messaggi

Come per la fase di disconnessione, il funzionamento del pattern a scambio di messaggi è riportato solo parzialmente nella figura 4.2 in quanto è analogo lato server e lato client.

In particolare è possibile suddividere questo modello di comunicazione in due fasi, quella di invio del messaggio e quella di ricezione del messaggio. Nella prima (indicata nel diagramma dalla nota 'send msg') il client invia il messaggio al server tramite la sessione, dopo di che questo viene reindirizzato alla mailbox (casella postale) del destinatario dal coordinatore della rete, portando così a termine l'operazione di invio.

Mentre per la seconda (indicata nel diagramma dalla nota 'receive msg') il dispositivo che effettua la richiesta di un messaggio, non fa altro che domandarne uno alla sua mailbox, la quale a meno che non sia vuota restituisce un messaggio.

Come è stato precedentemente affermato tale funzionamento è il medesimo per qualsiasi dispositivo sia che questo sia il destinatario sia che questo sia il mittente, sia per i client sia per il master. Analogamente infatti se il destinatario fosse il server, il messaggio verrebbe inserito all'interno della sua casella di posta, allo stesso modo se questo fosse il mittente processerebbe direttamente il messaggio inviandolo ai device target.

4.3.2 Pattern Publish/Subscribe

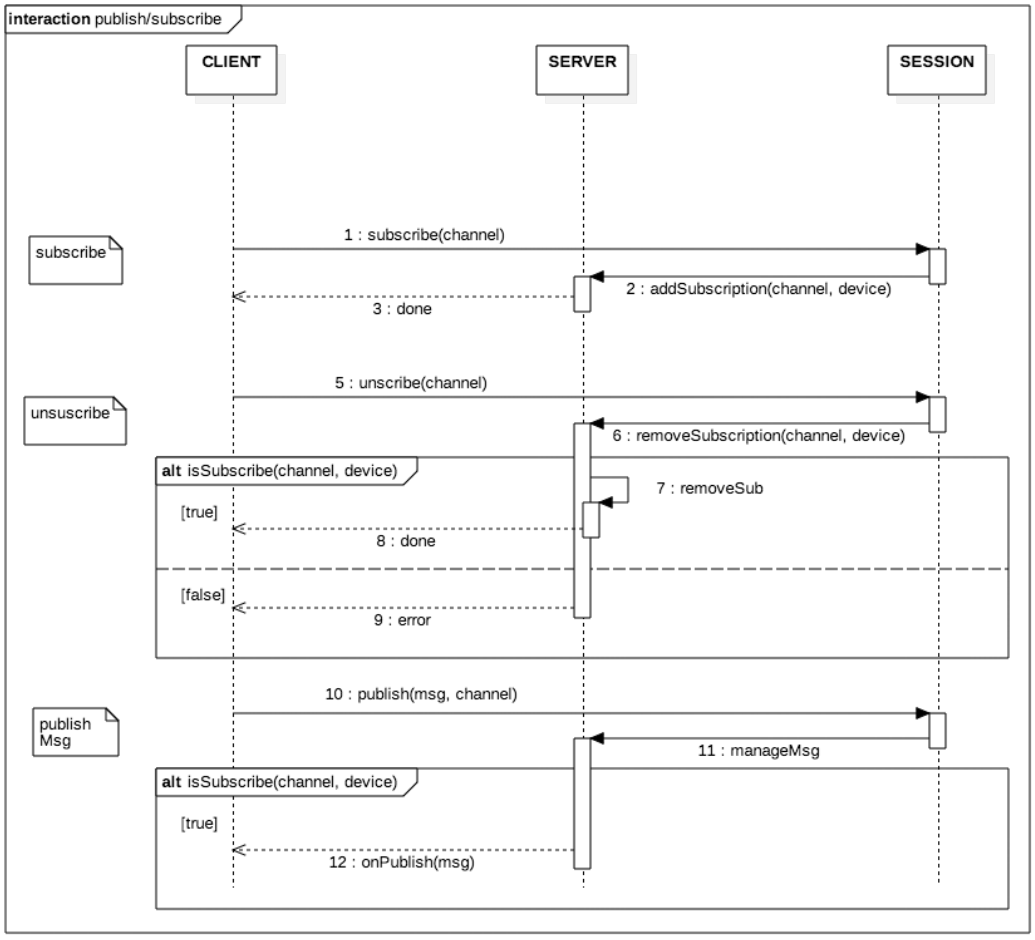


Figura 4.3: Diagramma di sequenza rappresentante il funzionamento del modello di comunicazione publish/subscribe

Dal diagramma sopra riportato, possiamo trovare le tre operazioni caratteristiche del modello publish/subscribe applicate al client. Analogamente a quanto visto nelle sezioni precedenti e ricordando che in senso lato il server estende le funzionalità di scambio di messaggi del client, tutto quello che viene rappresentato per quest'ultimo può essere applicato anche per il nodo centrale della rete, nel grafico non viene riportato per evitare di appesantirlo duplicando tutte le interazioni.

In particolare un dispositivo per sottoscriversi ad un canale (indicato nel diagramma dalla nota 'subscribe') deve effettuare la richiesta tramite la ses-

sione al server, quest'ultimo registra la sottoscrizione e conferma il termine dell'operazione.

Chiaramente è possibile rimuovere quanto appena fatto (indicato nel diagramma dalla nota 'unsuscribe') inoltrando sempre attraverso la sessione una diversa richiesta al server, il quale controlla di aver precedentemente accoppiato il dispositivo e il canale dopo di che si appresta a rimuove tale coppia, altrimenti lo segnala con un errore al device che aveva effettuato la richiesta di annullamento dell'iscrizione.

Infine la pubblicazione di un messaggio (indicata nel diagramma dalla nota 'publishMsg') avviene inviando sulla sessione la coppia messaggio canale, questa viene gestita dal master che per ogni dispositivo registrato su quel determinato canale invia una copia del messaggio, ovviamente se nessun dispositivo è sottoscritto a quel canale il messaggio non viene recapitato a nessun destinatario, viene quindi perduto.

4.3.3 Pattern Spazio delle Tuple

In quest'ultimo diagramma (figura 4.4) è riportato il funzionamento del modello di comunicazione che si appoggia allo spazio delle tuple. Come possiamo vedere ci sono tre operazioni principali out, in e rd.

La prima, indicata dalla nota 'out', invia la tupla al server tramite la sessione, questo controlla se ci sono richieste pendenti, in caso affermativo soddisfa tali richieste secondo le regole del pattern, altrimenti aggiunge il messaggio allo spazio delle tuple.

Per la seconda operazione, indicata dalla nota 'in', il device effettua una richiesta tramite la solita sessione al server, il quale verifica la presenza di una tupla che soddisfa il template passato come parametro ed eventualmente la rimuove dalla memoria condivisa e la ritorna al dispositivo richiedente, mentre in caso negativo mantiene internamente la richiesta finché non può essere evasa.

La read, indicata dalla nota 'rd', funziona allo stesso modo della in spiegata sopra a differenza che l'istanza che soddisfa il template non viene rimossa dallo spazio delle tuple, ma ne viene restituita solamente una copia.

Così facendo si rendono persistenti i messaggi attraverso lo spazio delle tuple e le richieste pendenti all'interno del master.

È nuovamente necessario sottolineare che nel diagramma il comportamento e le interazioni del client possono essere applicate anche al server, godendo questi delle stesse funzionalità nello scambio dei messaggi, ma che per evitare interazioni ridondanti non sono state incluse nel diagramma.

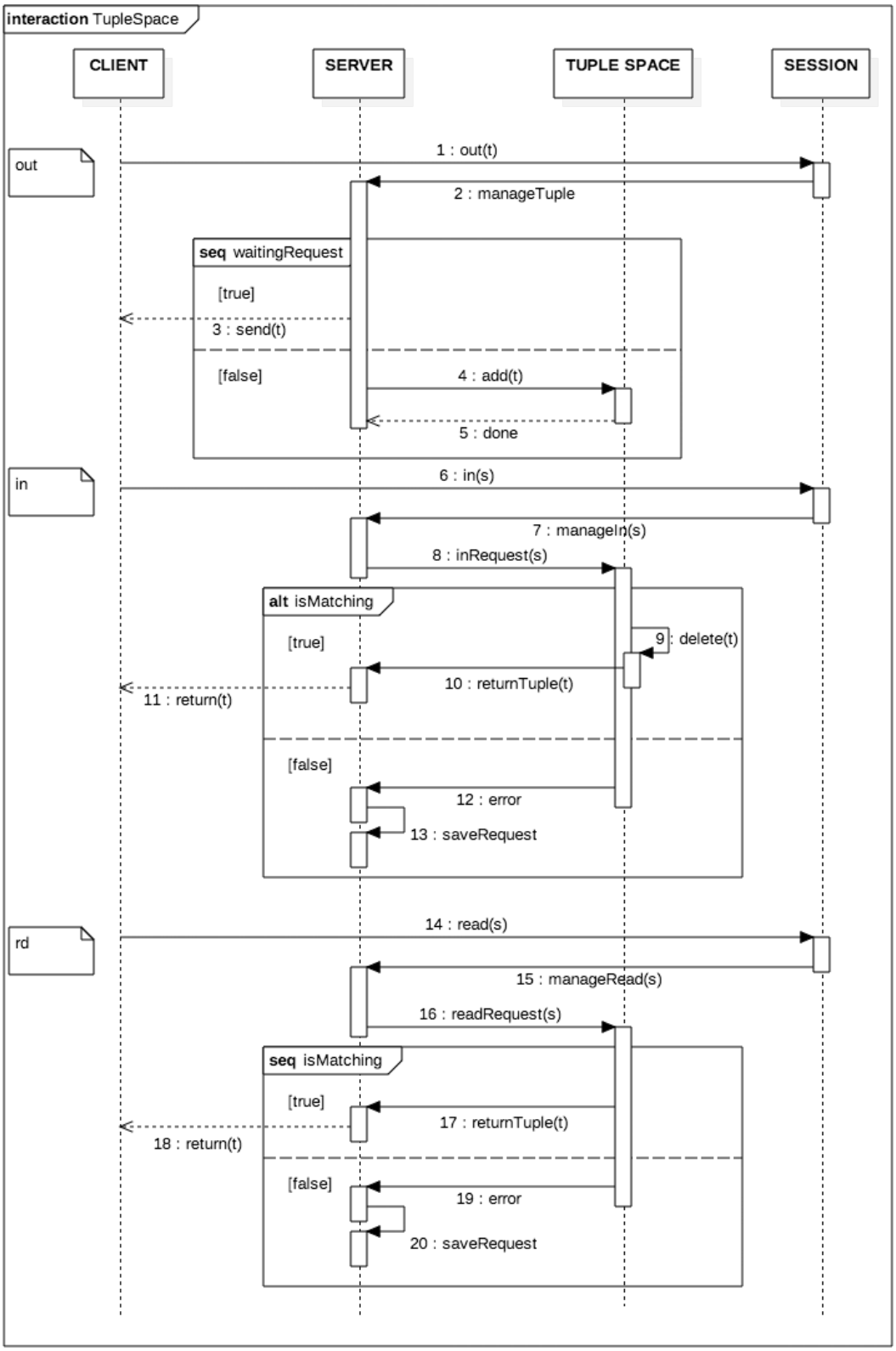


Figura 4.4: Diagramma di sequenza rappresentante il funzionamento del modello di comunicazione basato sullo spazio delle tuple

Capitolo 5

Implementazione prototipale

In questo capitolo verranno analizzati i componenti fondamentali della fase d'implementazione che costituiscono il middleware. In particolare dopo aver introdotto le tecnologie utilizzate, si suddividerà il sistema in tre parti, quella inerente al core della connessione all'interno del sistema, quella in cui vengono considerati i diversi modelli di comunicazione e infine quella riguardo la persistenza dei dati.

Primi di entrare in merito alle questioni sopra citate è necessario sottolineare nuovamente che il middleware e parte della sua struttura, utilizza la libreria open source Android Multi Bluetooth Library, che fornisce funzionalità specifiche per la connessione tramite socket bluetooth tra più dispositivi e lo scambio di messaggi fra di questi. Al contrario non gestisce minimamente problematiche relative a disconnessioni improvvise, collegamenti successivi e sequenze di connessione alla rete dei dispositivi differenti da quella canonica (ad esempio connessione anticipata del client rispetto al server). Nonostante questo è sembrata subito un interessante punto di partenza sul quale ragionare per mettere in piedi un solido middleware di alto livello per la comunicazione tra device Android. Infine è necessario specificare che le API definite nella sezione 4.1 sono state progettate e implementate per funzionare correttamente su sistemi Android, per cui il loro funzionamento è pensato ad hoc per questo tipo di dispositivi, ragionando anche con le logiche interne del sistema di Google. Nello specifico le procedure indicate come bloccanti lo sono a basso livello, ovvero a quello di thread definendo delle callback da implementare per definirne il comportamento specifico, ma chiaramente non ad alto livello cioè activity e main thread in quanto è espressamente vietato da Android.

Per concludere si vuole ulteriormente ribadire che tutto quanto è frutto di questo lavoro di tesi è reso pubblico alla comunità open source attraverso il repository GitHub: `https://github.com/edwfr/CommunicationMiddleware`.

5.1 Tecnologie

In questa sezione verranno riportate le tecnologie che sono state impiegate per l'implementazione del middleware e che contribuiscono al suo corretto funzionamento e al soddisfacimento dei requisiti.

5.1.1 EventBus

Eventbus[1] è una libreria open-source per Android e Java che permette la comunicazione tra i componenti attraverso il pattern publish/subscribe. Il suo funzionamento, riportato in figura 5.1, consente la comunicazione centralizzata tra classi diverse, disaccoppiando i mittenti e i ricevitori.

Funziona molto bene con i componenti principali della programmazione Android come Activity, Fragment e Thread, semplificando notevolmente il codice, rimuovendo le dipendenze e accelerando lo sviluppo delle app, tanto che si stimano circa 100.000.000 applicazioni installate che ne fanno utilizzo. Ancor più viste le sue ridotte dimensioni (50kB) e la sua elevata efficienza, dato che evita le dipendenze complesse e soggette a errori e riduce i problemi del ciclo di vita. Per il suo corretto funzionamento è necessario registrare il componente Subscriber e implementare in questo le callback `onEvent()` per ricevere notifica degli eventi di interesse, che vengono lanciati dai componenti Publishers con il comando `post()`.

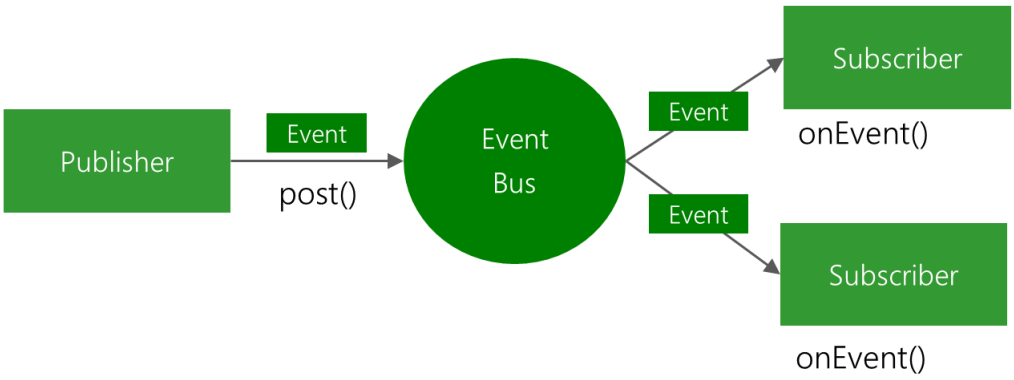


Figura 5.1: Schema di funzionamento publish/subscribe di EventBus

[1]EventBus, `http://greenrobot.org/eventbus/`

5.1.2 JSON

JSON[2], JavaScript Object Notation, è un formato standard per lo scambio di informazioni, essendo facile da leggere e scrivere per le persone e agile da generare e analizzarne la sintassi dalle macchine.

Inoltre è completamente indipendente dal linguaggio di programmazione, proprio per questo viene ampiamente utilizzato per lo scambio di dati in applicazioni di qualsiasi natura. JSON è basato su oggetti e array:

```
{
    "msgType" : 1,
    "msgContent" : "Hello"
    "isToDelate" : false
}
```

Listato 5.1: Object, insieme di coppie chiave-valore

```
[
    "Ford",
    "BMW",
    "Fiat"
]
```

Listato 5.2: Array, raccolta ordinata di valori

Come è possibile notare dai precedenti listati, la sua struttura è fondata su chiavi e valori. Le prime devono essere di tipo stringa, mentre per i secondi sono ammessi number, object, array, boolean, null e string. Dato che la sintassi di dichiarazione delle strutture è molto semplice il parsing JSON risulta essere molto veloce.

Nello specifico per la realizzazione di questo progetto è stata utilizzata la libreria Java Gson[3] sviluppata da Google, grazie alla quale è possibile convertire rapidamente oggetti Java in JSON e viceversa. Inoltre tra le varie librerie di parsing questa emerge come la più prestante per la conversione di JSON sotto i 100kb[4], per cui considerando i principali scenari applicativi dell'API risulta come la miglior scelta per il middleware oggetto di tesi.

[2]JSON, `https://www.json.org/json-it.html`

[3]Gson Library, `https://github.com/google/gson/blob/master/UserGuide.md`

[4]The ultimate JSON Library: JSON.simple vs GSON vs Jackson vs JSONP, `https://blog.takipi.com/the-ultimate-json-library-json-simple-vs-gson-vs-jackson-vs-json/`

5.2 Organizzazione

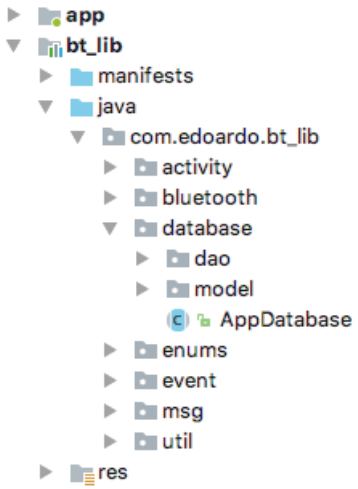


Figura 5.2: Organizzazione dei package del middleware

Al fine di rendere il middleware portabile in diverse applicazioni e in più dispositivi, questo è stato implementato in modo completamente indipendente attraverso un nuovo modulo esterno di tipo Android Library. Come riporta la figura 5.2, questo è strutturato in sette package che di seguito analizzeremo brevemente:

- **activity**: al suo interno troviamo la classe `CommunicationMiddleware`, attraverso la quale vengono fonite tutte le API definite nella sezione 4.1.
- **bluetooth**: questo package potrebbe essere definito il core del progetto in quanto contiene le classi principali per la gestione della connessione bluetooth e per lo scambio di messaggi.
- **database**: è suddiviso in due ulteriori package, nel `dao` ci sono le interfacce per interagire con le tabelle in cui vengono mantenute le informazioni, a loro volta rappresentati nel `model`. Le funzionalità di persistenza dei dati vengono fornite attraverso la classe `AppDatabase` che raggruppa i singoli Data Access Object.
- **enums**: contiene alcune enumeration Java necessarie per il corretto funzionamento della libreria.
- **event**: in questo package sono presenti delle classi utilizzate per coordinare lo scambio di messaggi asincroni tra componenti all'interno del

middleware, corrispondono a parte degli eventi generati e gestiti da EventBus.

- **msg**: le classi che lo compongono sono impiegate per lo scambio di messaggi tra i diversi dispositivi. Ognuna di queste, come vedremo in seguito, porta un insieme di informazioni specifico per il scambio di messaggi durante il normale funzionamento del sistema.
- **util**: questo package contiene delle utility impiegate nella libreria, tra cui un helper per la gestione dei JSON e un'interfaccia per la gestione delle SharedPreferences.

5.3 Bluetooth Core

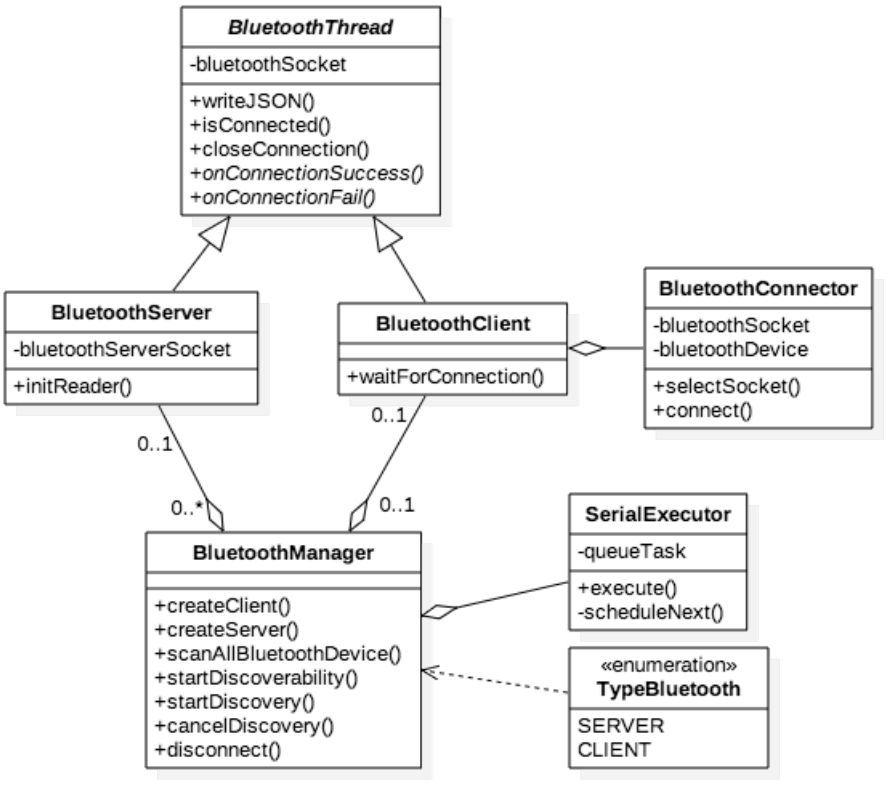


Figura 5.3: Diagramma delle classi inerente alla gestione delle connessioni

Nel diagramma 5.3 è riportata la struttura core del middleware, in particolare alla base di tutto si trova la classe astratta `BluetoothThread` che

implementa l'interfaccia Runnable e nel metodo *run* generalizza il comportamento dei dispositivi. Inoltre sempre in questa stessa classe è incapsulata la logica di basso livello per l'invio e la ricezione dei messaggi, sfruttata poi dai metodi di più alto livello come vedremo nella sezione successiva.

`BluetoothServer` e `BluetoothClient` estendono la classe sopra citata specificando il comportamento di questa nell'inizializzazione della connessione e nelle callback in seguito ad avvenuta connessione o disconnessione.

Infine il `BluetoothManager`, che vedremo più volte durante il corso di questo capitolo, si occupa di gestire in toto la connessione. In particolare questo al suo interno mantiene il riferimento al ruolo che il dispositivo ricopre nella connessione, se è quindi client o server. Nel primo caso conterrà al suo interno un'unica istanza della classe BluetoothClient tramite la quale potrà comunicare con il server. Nel secondo caso sarà necessario mantenere più istanze di BluetoothServer, una per ogni dispositivo collegato, per mantenere le connessioni indipendenti e quindi più robuste. Una prima configurazione di sistema viene resa persistente all'interno del middleware tramite l'utilizzo delle SharedPreferences, andando a salvare il TypeBluetooth del dispositivo e nel caso questo fosse di tipo client, il riferimento al server al fine di velocizzare l'operazione di ripristino della connessione in caso di successivi collegamenti o improvvise disconnessioni. Inoltre il manager grazie alla classe `SerialExecutor` coordina l'esecuzione dei thread (BluetoothServer o BluetoothClient a seconda del ruolo del device nel collegamento). Nello specifico proprio all'interno di quest'ultima classe è possibile andare a modificare un parametro per aumentare o diminuire la reattività con cui i thread vengono eseguiti in base alle particolari esigenze dello scenario applicativo introducendo così nel sistema una sorta di latenza attraverso la quale è possibile ridurre le risorse impiegate.

In questo primo diagramma non sono incluse tutte le funzionalità che sono messe a disposizione ma solo in parte quelle relative alla fase di instaurazione della connessione e disconnessione. Nello specifico si può notare che sono presenti i metodi per connettere il dispositivo come client o per inizializzarlo come server, quelli per avviare e fermare la scoperta di device bluetooth nelle vicinanze e per rendersi visibile agli altri.

Per analizzare correttamente le funzionalità dello scambio di messaggi è necessario dividere queste operazioni in più livelli e il diagramma 5.4 specifica alcuni dettagli del funzionamento dei thread, ma soprattutto evidenzia il nucleo dello scambio dei messaggi. Infatti tutto questo è impacchettato in tre diversi layer, come era già stato accennato e come gradualmente scopriremo, e quello mostrato dal diagramma rappresenta quello di più basso livello.

5.4 Scambio di messaggi

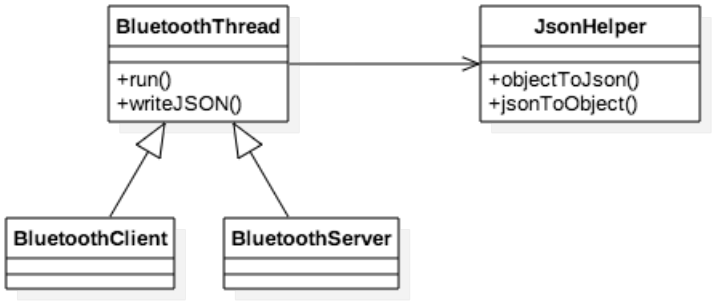


Figura 5.4: Diagramma delle classi inerente ai Thread Server e Client

Nel metodo *run* del BluetoothThread è implementata la logica di ricezione del messaggi, che non appena vengono ricevuti sull'input stream vengono trasformati da Json a oggetti (grazie all'utility JsonHelper) e inoltrati al sistema come eventi tramite EventBus, a prescindere che il dispositivo sia client o server. In questo modo, purché il sistema risulti connesso, lo si rende reattivo agli input grazie alla chiamata a funzione `inputStream.read()` che è bloccante ed attende la ricezione di un messaggio sullo stream di input della socket. Mentre con *writeJSON()* è possibile effettuare il parsing di un oggetto in un Json, sempre attraverso la classe helper, per poi essere inviato tramite l'output stream della socket aperta tra i due dispositivi.

Considerando che uno degli obiettivi principali del middleware è di supportare modelli di comunicazione di alto livello, è necessario incapsulare il funzionamento appena analizzato nel Bluetooth Manager, all'interno del quale vengono implementate la maggior parte delle operazioni e dei controlli al fine di far funzionare correttamente il sistema e di rendere persistenti le diverse informazioni che vengono gestite, come vedremo nella sezione successiva. Dopo di che tutte le funzionalità di questa classe sono messe a disposizione del CommunicationMiddleware, che contribuisce ulteriormente a nascondere i dettagli implementativi di basso livello all'utilizzatore, semplificandogli notevolmente la fruizione dei servizi dell'API.

Tutto questo emerge nel diagramma di figura 5.5, dal quale si nota l'ulteriore incapsulamento dell'API. Infatti i parametri in ingresso dei metodi della CommunicationMiddleware sono decisamente di alto livello e quindi più intuitivi per il programmatore, mentre quelli richiesti dal BluetoothManager sono specifici e creati ad hoc per l'impiego all'interno di questo middleware.

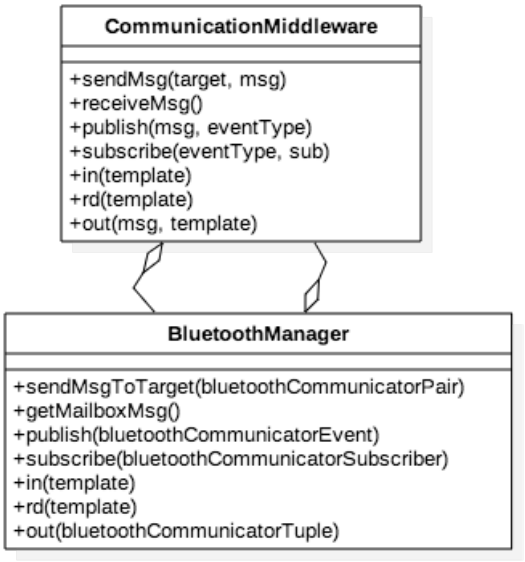


Figura 5.5: Diagramma delle classi inerente ai modelli di comunicazione

Come era stato già affermato, all'interno del packege `msg` sono contenuti le classi per gestire lo scambio di informazioni tra dispositivi, ma non sono solamente quelle che si possono vedere nel diagramma 5.5, in quanto il manager ne utilizza internamente anche altre.

- **BluetoothCommunicatorEvent**: è utilizzato per lanciare nuovi eventi durante l'utilizzo del pattern publish/subscribe, contiene il tipo di evento (inteso come canale), l'identificatore del mittente e il messaggio.

- **BluetoothCommunicatorPair**: viene impiegato per l'invio di singoli messaggi, racchiudendo l'identificativo del mittente e il messaggio.

- **BluetoothCommunicatorSubscriber**: viene utilizzato per effettuare o rimuovere sottoscrizioni ad un determinato tipo di evento nel modello di comunicazione publish/subscribe, contiene infatti il tipo di evento, il riferimento al dispositivo e un booleano per la sottoscrizione o la rimozione.

- **BluetoothCommunicatorTemplate**: è formato da un booleano che indica se la richiesta è read o di in, dal riferimento al mittente e dal template richiesto. Ovviamente serve per effettuare una richiesta allo spazio delle tuple.

- **BluetoothCommunicatorTuple**: a fronte di richieste che possono essere soddisfatte il server risponde con questi oggetti che hanno al loro interno il messaggio, il suo template e l'identificativo del mittente. All'interno del middleware rappresenta la tupla.

Dopo aver scorso tutti i diversi tipi di messaggi, potrebbe saltare all'occhio che questi presentino delle informazioni superflue o che alle volte siano ridondanti. Infatti ad una prima analisi è così, ma è necessario sottolineare che per il pattern publish/subscribe e lo spazio delle tuple sono state implementate due versioni di tutti i metodi, una che corrisponde a quella mostrata sopra nel diagramma e l'altra *safe*, leggermente raffinata in termini di sicurezza, con delle piccole accortezze per variare ragionevolmente il funzionamento del middleware in specifici scenari. Ad esempio nel caso in cui un dispositivo sia registrato ad un tipo di evento e proprio questo ne manda di quel tipo, tale evento non gli viene notificato.

All'inizio del capitolo abbiamo introdotto la libreria EventBus grazie alla quale è possibile comunicare tra i diversi componenti dell'applicazione tramite degli eventi, in particolare questi sono composti dalle classi appena elencate `BluetoothCommunicator*` e da quelle contenute nel package `event`. Per il corretto funzionamento della libreria è necessario registrare l'activity, in questo caso la classe CommunicationMiddleware, durante la `onStart()`, prevedere le callback alla ricezione degli eventi e infine rimuoverla dai listener nella `onDestroy()`.

5.5 Persistenza

In questa sezione vedremo come il sistema, nello specifico il BluetoothManager riesce a rendere persistenti le informazioni necessarie. In generale vengono utilizzate due diverse tecnologie, le SharedPreferences e le Room Persistence Library. Con le prime è possibile salvare delle coppie chiave valore nel dispositivo mentre attraverso le seconde si consente un accesso rapido e fluente al database senza scendere a basso livello con SQLite.

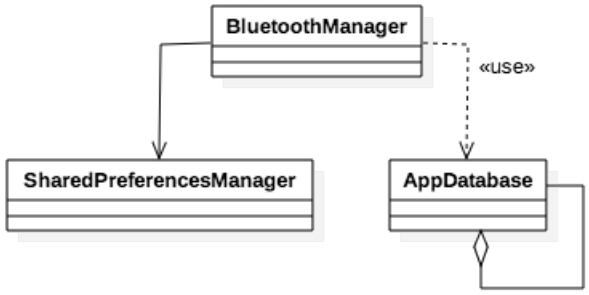


Figura 5.6: Diagramma delle classi inerente alla persistenza dei dati

In particolare attraverso la classe SharedPreferencesManager vengono salvati i dati di configurazione del middleware, ovvero il ruolo che il dispositivo ha nella rete e nel caso in cui questo sia un client si mantiene in memoria anche il riferimento al server. In questo modo è possibile rimediare rapidamente a perdite di connessione e allo stesso tempo trovare pre-configurato il sistema dopo ogni suo avvio. Si è però deciso di non utilizzare solamente questa tecnologia, in quanto permette di mantenere come valori esclusivamente stringhe, che seppur di dimensioni grandi richiedono tempo di esecuzione per il parsing da oggetti a json e viceversa.

Per cui per quanto riguarda le informazioni più complesse che transitano all'interno della rete di dispositivi, queste vengono rese persistenti attraverso la classe AppDatabase, ma in generale grazie alle Room Persistence Library, la cui architettura è possibile vederla nella figura 5.7. Nello specifico questa nuova funzionalità dell'API Android, è stata introdotta durante l'evento Google I/O del 2017 ed è strutturata su tre livelli, le Room Database, i Data Access Object e le Entities. In pratica attraverso quest'ultime vengono definite le caratteristiche delle tabelle (colonne, tipi di dati, chiavi primarie, ecc...) contenute nelle Room e gestite tramite le operazioni implementate nel DAO.

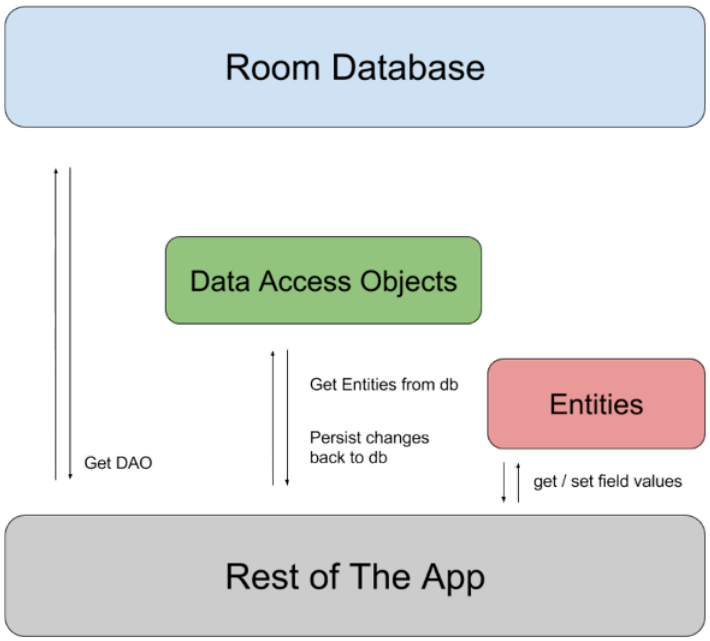


Figura 5.7: Architettura delle Room Persistence Library

Grazie all'utilizzo di questa nuova tecnologia si rendono separati gli aspetti di interazione con il database e il resto dell'applicazione, inoltre grazie ai vari tag e la logica che ne sta alla base è decisamente semplifica la scrittura di query. Come è stato già detto nella pratica per ogni entità che si desidera rendere persistente è necessario creare una classe di *model* e un *DAO*, per cui nel middleware sono presenti le seguenti tabelle:

- **Msg**: questa tabella è presente in tutti i dispositivi e serve a mantenere in memoria i messaggi ricevuti.
- **MsgDispatcher**: nel caso in cui un dispositivo invia un messaggio ad un destinatario che al momento non è connesso, il server utilizza questa tabella per salvare il messaggio pendente e non appena il destinatario si connette nuovamente glieli recapita tutti in modo da non perdere nessuna comunicazione.
- **RoutingTable**: il server dovendo fare il coordinatore della rete ha bisogno di tenersi in memoria gli accoppiamenti tra dispositivi e canali sottoscritti, questa è il compito della tabella di routing.
- **Subscription**: viene utilizzata per mantenere in ogni device le sottoscrizioni effettuate ai tipi di eventi, per rendere più agevoli alcune operazioni interne.

- **TupleSpace**: questa è il vero e proprio spazio delle tuple che viene mantenuto dal server, in attesa di richieste o di nuovi messaggi da inserire.
- **Request**: nel momento in cui il server non può rispondere ad una richiesta di un client, questa non viene persa ma bensì viene salvata in questa tabella, che poi sarà consultata alla ricezione di un messaggio per verificare la presenza di richieste pendenti da soddisfare.

```
1  @Dao
2  public interface DaoMsgDispatcher {
3
4      @Insert(onConflict = REPLACE)
5      void insertMsg(MsgDispatcher msg);
6
7      @Query("SELECT * FROM MsgDispatcher WHERE mac = :mac LIMIT 1")
8      MsgDispatcher getMsgForClient(final String mac);
9
10     @Query("DELETE FROM MsgDispatcher WHERE uid= :id")
11     void deleteMsg(final int id);
12
13 }
```

Listato 5.3: Esempio di implementazione di un DAO

Attraverso la combinazione di queste due tecnologie i dati all'interno del middleware vengono mantenuti in memoria, rendendolo stabile e robusto di fronte a malfunzionamenti o comportamenti inaspettati, in quanto tutto i messaggi che vengono scambiati, le sottoscrizioni e le richieste effettuate vengono mantenute in tabelle predefinite e non in strutture dati temporanee.

Capitolo 6

Validazione

In questo capitolo verrà presa in considerazione la fase di validazione del middleware di comunicazione e coordinazione, durante la quale è stato possibile sottolineare il pieno soddisfacimento dei requisiti definiti nelle varie sezioni del capitolo 3. Dato che uno degli obiettivi principali era quello di creare una libreria generica e utilizzabile in diversi scenari applicativi, la fase di testing del middleware è stata effettuata tramite una semplice applicazione attraverso la quale è stato possibile provare tutte le caratteristiche dell'API così da comprenderne a pieno le sue potenzialità.

In particolare dai dati che sono contenuti nelle seguenti sezioni emerge chiaramente il corretto funzionamento del middleware, soprattutto in termini di agilità e reattività a partire dall'instaurazione di una connessione, al ripristino di un collegamento caduto, allo scambio di messaggi tra dispositivi che avviene senza perdere informazioni e in tempi decisamente ridotti a prescindere dal modello di comunicazione utilizzato. Allo stesso modo il sistema mostra grande sicurezza, robustezza e resilienza comportandosi ottimamente davanti ai malfunzionamenti che causano la perdita di connessione, ristabilendo il collegamento tra i due dispositivi in pochi secondi. Tutto questo è facilitato dalla configurabilità del sistema (impostazione del numero di client e del tempo di discovery, scelta del dispositivo server) per rendere fluida la fase di connessione senza sprecare risorse.

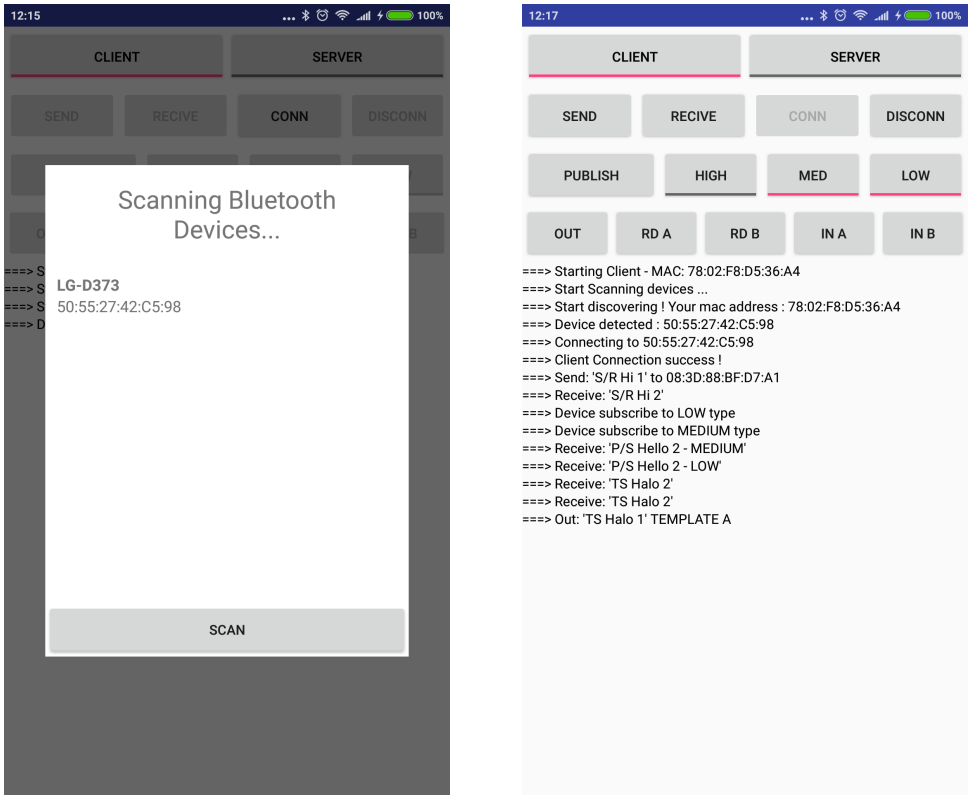
Infine tutto questo è possibile grazie all'impiego delle Room Persistance Library e alla Shared Preferences che permettono di rendere persistenti i dati e contribuiscono alla robustezza del middleware nei diversi scenari applicativi e di fronte a imprevedibili malfunzionamenti, in combinazione con le tecnologie applicate come EventBus e Gson.

6.1 Introduzione

Nelle due schermate di figura 6.1 sono riportate le parti essenziali dell'applicazione di test sviluppata per provare le funzionalità messe a disposizione dell'API. In particolare dalla schermata 6.1a si può vedere come aprire una nuova connessione da parte di un client; semplicemente selezionando il server dalla lista di dispositivi trovati nelle vicinanze viene avviato il processo di instaurazione della connessione, sarà necessario eseguire tale procedura solamente in fase di configurazione della rete poiché tutte le volte successive questa operazione sarà automatica, connettendosi all'ultimo server con cui si era collegato.

Mentre dallo screenshot 6.1b si ha una panoramica di tutte le funzionalità utilizzabili tramite un vasto set di button. Nello specifico *CLIENT* e *SERVER* servono a definire il ruolo del device all'interno della rete; *SEND* e *RECEIVE* per inviare e ricevere messaggi secondo il pattern a scambio di messaggi. *CONN* per aprire la dialog sopra citata e avviare la richiesta di un nuovo collegamento, mentre *DISCONN* per dissociarsi dalla rete. La terza fascia di bottoni è composta da *PUBLISH*, *HIGH*, *MED* e *LOW*, il primo per pubblicare un nuovo evento nella rete, mentre gli altri tre sono degli switch attraverso i quali il dispositivo si può sottoscrivere o meno ai diversi tipi di eventi. Inoltre con *OUT*, *RD A*, *RD B*, *IN A* e *IN B* è possibile testare le funzionalità del modello basato sullo spazio delle tuple, specificate chiaramente dalle label di questi ultimi. Infine sotto al set dei bottoni una text view sulla quale effettuare delle stampe, a mo' di console di log. È evidente che l'applicazione è tanto semplice quanto funzionale ad evidenziare le caratteristiche del middleware, inoltre i tipi di evento e i template dello spazio delle tuple sono di numero ridotto per non rendere caotica l'interfaccia di testing, chiaramente queste possono essere ridotte e aumentate in base alle esigenze dello scenario applicativo.

Al termine dello sviluppo del middleware è stato possibile analizzarne il comportamento attraverso l'esecuzione dell'applicazione di testing collegando tre dispostivi Android (LG D373, Samsung S3 Neo e Xiaomi Mi5S Plus) secondo diverse configurazione di rete senza notare però variazioni significative dei tempi impiegati per ultimare le operazioni. Per cui si è presa come riferimento la rete così organizzate: Xiaomi Mi5S Plus - server, LG D373 - client e Samsung S3 Neo - client.



(a) Dialog per l'instaurazione di una nuova connessione

(b) Schermata principale per il test del middleware

Figura 6.1: Schermate dell'applicazione di testing

Nei paragrafi successivi verificheremo il comportamento dell'API in ogni sua fase di funzionamento. Una volta per tutte si specifica che, anche se in maniere differenti che verranno poi specificate di volta in volta, i tempi delle successive tabelle sono stati ricavati da mirati log di sistema che riportavano i `currentMillis()` delle fasi delle varie operazioni del middleware dello stesso dispositivo, ovviando così a problemi di sincronizzazione degli orologi e dei millisecondi dei diversi dispositivi di testing.

In particolare per ogni operazione verrà riportato il valore medio, la mediana, la moda, la deviazione standard (rms), il valore minimo e il valore massimo, che sono stati calcolati su un campione di 100 simulazioni per operazione. Infine per testare le funzionalità dei modelli di comunicazione verranno eseguiti dei test con messaggi di un singolo byte e di 100 byte, questo per provare il funzionamento del middleware e per osservare i livelli di prestazione in caso di messaggi di diversa lunghezza. Si è scelto questo range di validazione in quanto l'applicazione dell'API potrebbe avvenire in contesti in cui è sufficien-

te inviare piccole quantità di dati, anche singoli caratteri o valori, per ogni comunicazione.

Per chiarire ulteriormente, prima di entrare nel dettaglio della fase di testing, con il successivo schema si vuole spiegare come sono state effettuate le rilevazioni per quanto riguarda la parte dello scambio dei messaggi. Nel dispositivo client veniva preso il tempo T0 di avvio della procedura e il tempo T3 di ricezione dell'ack o dello stesso pacchetto inviato. Mentre nel server veniva isolato il periodo di elaborazione sottraendo al tempo T2 di invio della risposta, il tempo T1 ovvero quello di ricezione della richiesta. Per cui eseguendo la differenza tra T3 e T0 si otteneva il tempo totale dell'operazione, dall'invio alla ricezione della risposta ed è stato calcolato sempre sullo stesso dispositivo. Per specificare il tempo di trasferimento da client a server e viceversa, si è calcolato in quest'ultimo T2-T1, ovvero il tempo di elaborazione della richiesta, enfatizzato nello schema dalla freccia verde. Per cui se dalla successive tabelle si vuole calcolare il tempo di trasferimento delle informazioni sarà sufficiente, in linea di massima eseguire l'operazione (T3-T0)-(T2-T1), se si vuole raffinare la stima per un unico trasferimento si può dividere tale valore per 2 nel caso di invio e ricezione dello stesso messaggio altrimenti nel caso di utilizzo di ack va pesata in base ai byte trasmessi.

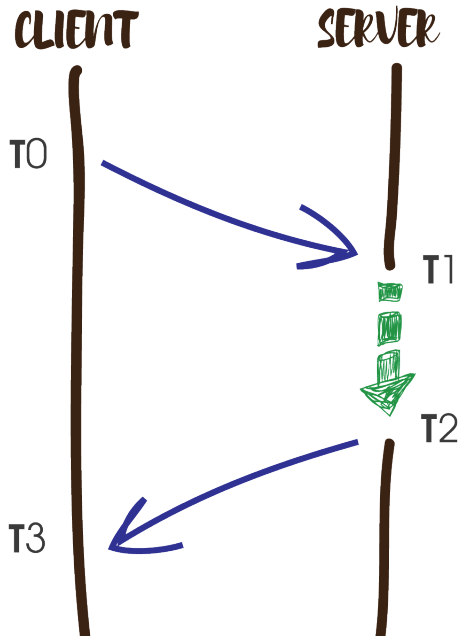


Figura 6.2: Schema della comunicazione nel middleware

6.2 Instaurazione della connessione

	scenario 0	**scenario 1**	**scenario 2**
media	3170,92	2325,47	1145,25
mediana	3022	1949	1035
moda	2128	1848	842
rms	1285,63	1020,89	483,08
min	1298	1046	442
max	6744	4449	3744

Tabella 6.1: Tempi in millisecondi per instaurare una connessione

I dati della tabella 6.1 mostrano i tempi necessari per instaurare una connessione tra server e client in tre diversi scenari, nuova connessione tra client e server (scenario 0), riconnessione dei dispositivi in seguito alla disconnessione del client (scenario 1) e riconessione dei dispositivi dopo la disconnessione del server (scenario 2).

Si deve tenere conto che i tempi sono stati calcolati tra dispositivi precedentemente accoppiati, altrimenti tale stima sarebbe di gran lunga superiore e poco significativa in quanto è necessaria l'interazione dell'utente per confermare da entrambe le parti l'operazione di *pairing*. È inoltre necessario specificare che per tutti e tre i casi entrambi i dispositivi devono essere visibili e il client deve avviare la *discovery* dalla quale trovare il server e avviare la connessione.

I tempi riportati in tabella sono calcolati dal momento in cui il client trova il server al momento in cui viene aperta la socket e quindi abilitata la comunicazione, in questo modo viene isolata la fase di instaurazione della connessione, tralasciando i tempi di scandaglio delle frequenze e ricerca dei device nelle vicinanze. Nel caso di una connessione nuova il client e il server impiegano 3170,92±1285,63 ms per collegarsi, mentre nel caso di perdita della connessione da parte del client i due lati impiegano 2325,47±1020,89 ms per ristabilirla, infine nel caso in cui sia il server a disconnettersi la connessione viene ripristinata in 1145,25±483,08 ms simultaneamente con più client.

Chiaramente durante il normale utilizzo dell'applicazione in cui durante l'esecuzione vengono richiesti i permessi di rendere visibile il dispositivo e quindi è necessaria l'interazione dell'utente, globalmente i tempi di connessione aumentano in quanto dipende dalla reattività dell'utente e alla gestione di Android dei componenti grafici.

6.3 Modello a scambio di messaggi

Per quando riguarda il modello a scambio di messaggi faremo riferimento alla tabella 6.2 per l'operazione di invio e alla tabella 6.3 per la ricezione di un messaggio. Per testare queste funzionalità sono stati utilizzati due messaggi di diversi dimensioni, il primo composto da un 1 byte e il secondo da 100 byte, in pratica questi sono incapsulati negli oggetti precedentemente citati nella sezione 5.4 che includono altre informazione per cui le loro dimensioni effettive sono maggiori, per la precisione 110 byte nel primo caso, 209 byte nel secondo.

In merito alla send i tempi sono stati presi dal momento in cui il dispositivo invia il messaggio al momento in cui lo riceve indietro dal server, per queste operazioni in media ci impiega 53,78±11,11 ms (18.31 dei quali di elaborazione da parte del server e circa 13 del client). Così indicativamente l'intervallo di trasmissione tra client e server e viceversa è attorno agli 11 ms. Tale stima varia di poco nel caso in cui le dimensioni del messaggio siano di 100 bytes, infatti come emerge sempre dalla tabella 6.2 il server impiega circa 1 ms in più per l'elaborazione del messaggio e questo si rispecchia sulla durata complessiva dell'invio del messaggio.

Mentre l'operazione di ricezione di un messaggio, considerando il caso in cui il messaggio sia disponibile escludendo così le chiamate bloccanti, è decisamente rapida in quanto consiste nella sola esecuzione di una query al Room Database nel dispositivo, infatti questa avviene in 12,71±4,76 ms nel caso di messaggi da 1 byte, mentre impiega 13,97±5,14 ms nel caso di messaggi di 100 byte.

	1 byte		**100 byte**	
	client	server	client	server
media	53,78	18,31	55,7	19,16
mediana	52	20	54	18
moda	52	20	54	18
rms	11,11	4,30	11,19	2,77
min	29	4	38	19
max	84	23	91	29

Tabella 6.2: Tempi in millisecondi per effettuare una `send` di un messaggio, le colonne client indicano il tempo totale dell'operazione, mentre quelle server indicano il tempo necessario di elaborazione

	1 byte	**100 byte**
media	12,71	13,98
mediana	11	13
moda	8	10
rms	4,76	5,14
min	7	7
max	25	26

Tabella 6.3: Tempi in millisecondi per effettuare una `receive` di un messaggio

6.4 Modello publish/subscribe

	client	**server**
media	50,98	16,15
mediana	51	17
moda	43	19
rms	11,11	6,10
min	25	5
max	90	30

Tabella 6.4: Tempi in millisecondi per effettuare una `subscribe` ad un canale

La tabella 6.4 riporta i dati riguardo alla sottoscrizione di un dispositivo ad un canale o ad un tipo di evento, l'intervallo di tempo per l'operazione, dalla richiesta del client alla conferma del server è di 50,98±11,11 ms, di cui 16,15±6,10 ms utilizzati per l'elaborazione dal server in particolare per aggiornare il database e altri 13 per il parsing nel client. In totale vengono trasmessi due tipi di messaggi, uno ad hoc per la sottoscrizione al canale di 131 bytes e uno di ack di 31 bytes, quindi 162 bytes in circa 23 ms. Nel caso in cui si voglia annullare l'iscrizione ad un tipo di evento avviene la stessa operazione con una diverso parametro nel messaggio, che però non altera né il numero di byte trasmessi né il tempo impiegato per tale operazione.

Mentre per quanto riguarda la publish, l'intervallo di tempo rappresentato nella tabella 6.5 è quello che va dalla invio del messaggio dal client al server fino alla ricezione da parte del client dello stesso messaggio. Anche in questo caso sono stati effettuati test con messaggi da 1 byte e da 100 bytes. Nel primo caso vengono impiegato 56,24±11,11 ms per pubblicare un messaggio e riceverne la

notifica, di cui il server ne utilizza 18,69±7,46 ms per reindirizzare il messaggio ai client registrati su quel tipo di evento. Considerando che il messaggio da 1 byte viene incapsulato in un oggetto che ne pesa 119 bytes e calcolando indicativamente sempre 13 ms di elaborazione nel client, il trasferimento di 238 bytes avviene in 37,55 ms. Nel caso del messaggio di 100 bytes la differenza tra i tempi medi rispecchia la differenza tra i tempi di elaborazione nel server che aumenta di circa 8 ms. È necessario precisare che alla ricezione di un messaggio il server è costretto a controllare ogni volta nel database quali dispositivi sono registrati su quel canale e di conseguenza reindirizzargli il messaggio.

	1 byte		**100 byte**	
	client	server	client	server
media	56,24	18,69	62,45	27,32
mediana	55	20	60	26
moda	61	24	60	25
rms	11,11	7,46	11,19	5,41
min	40	4	43	18
max	86	32	98	52

Tabella 6.5: Tempi in millisecondi tra una `publish` di un messaggio e la ricezione di tale messaggio, le colonne client indicano il tempo totale dell'operazione, mentre quelle server indicano il tempo impiegato per elaborare le informazioni

6.5 Modello di comunicazione basato su spazio delle tuple

Nella tabella 6.6 sono rappresentati gli intervalli di tempo che intercorrono tra l'invio di un messaggio dal client al server (`out`) e la ricezione di avvenuta aggiunta di questo allo spazio delle tuple con un ack. Nel caso di un messaggio di 1byte viene spedito un pacchetto di 132 bytes e un ack di 31 bytes, in totale vengono impiegati mediamente 62,47±11 ms per il compimento di tale operazione. Come mostra la colonna del server gran parte di questo tempo è impiegato per l'elaborazione del messaggio nel dispositivo coordinatore della rete in cui viene mantenuto lo spazio delle tuple e i soliti 13 ms utilizzati dal mittente per portare a termine le varie operazioni. Infatti sempre dalla tabella sopracitata si può notare che anche se si aumentano le dimensione del messaggio a 100 bytes e quindi del pacchetto trasmesso, il tempo totale tra l'invio e la ricezione dell'ack varia di pochi ms, infatti nel caso di messaggi di

100bytes vengono impiegati 66,48±14,29 ms, 28,04±6,56 ms dei quali utilizzati dal server per rendere persistente il messaggio.

	1 byte		**100 byte**	
	client	server	client	server
media	62,47	26,5	66,48	28,04
mediana	63	29	66	29,50
moda	57	32	58	30
rms	11	7,59	14,29	6,56
min	28	10	38	10
max	87	38	130	48

Tabella 6.6: Tempi in millisecondi per effettuare una `out` di un messaggio, le colonne client indicano il tempo totale dell'operazione, mentre quelle server indicano il tempo necessario di elaborazione

	1 byte		**100 byte**	
	client	server	client	server
media	40,35	7,55	66,63	23,53
mediana	52	8	69	26
moda	34	7	73	27
rms	11,40	0,58	11,50	6,07
min	24	7	45	6
max	76	9	92	30

Tabella 6.7: Tempi in millisecondi per effettuare una `rd` di un messaggio dallo spazio delle tuple, le colonne client indicano il tempo totale dell'operazione, mentre quelle server indicano il tempo necessario di elaborazione

Per quanto riguarda l'operazione di `rd` i dati sono riportati nella tabella 6.7 e sono state prese in considerazione le richieste che potevano essere soddisfatte, escludendo così ritardi dovuti a operazioni di richiesta pendenti e quindi bloccanti. Sono necessari 40,35±11,40 ms per ricevere la tupla richiesta al server, quest'ultimo elabora rapidamente la richiesta in quanto deve semplicemente richiedere la tupla al database e ritornarla al client. Nel caso di messaggi di 1 solo byte questi vengono impacchettati in 121 bytes totali e complessivamente impiegano circa 33 ms per trasferire 242 bytes, mentre nel caso di informazioni

di 100 bytes a loro volta inclusi in messaggi di 231 bytes il middleware impiega 66,63±11,50 ms per soddisfare la richiesta, rispetto al caso precedente tale aumento di tempo è dovuto ad una maggiore elaborazione da parte del server che impiega in media 23,53 ms per rispondere alla read, mentre resta invariato il tempo di trasmissione.

	1 byte		**100 byte**	
	client	server	client	server
media	36,38	7,16	68,19	22,80
mediana	35	7	67	24
moda	31	7	58	24
rms	8,79	0,83	13,98	4,34
min	10	6	32	13
max	60	11	125	31

Tabella 6.8: Tempi in millisecondi per effettuare una `in` di un messaggio dallo spazio delle tuple, le colonne client indicano il tempo totale dell'operazione, mentre quelle server indicano il tempo necessario di elaborazione

Infine per quanto riguarda l'operazione di `in` è necessario fare riferimento alla tabella 6.8. Essendo logicamente simile all'operazione precedente i tempi sono molto affini. Analogamente vengono presi in considerazione solamente i casi in cui è possibile soddisfare le richieste per escludere tempi di attesa. Infatti per effettuare tale richiesta il client invia un pacchetto di 120 bytes e nel caso di risposta con un messaggio contenente 1 byte il server gli risponde con uno da 120 bytes. Per cui per trasferire completamente 240 bytes vengono impiegati 36,38±8,79 ms, 7,16 ms dei quali utilizzati dal server per controllare la soddisfacibilità della richiesta e per rimuove la tupla dalla memoria. Di nuovo come prima, all'incremento delle dimensioni del messaggio da ritornare corrisponde l'aumento del tempo impiegato dal server per elaborare la richiesta, questo è dovuto principalmente all'esecuzione della query e all'operazione di parsing del messaggio, di conseguenza per rispondere con messaggi di 100 bytes, incapsulati in pacchetti da 219 bytes, vengono impiegati complessivamente 68,19±13,98 ms per trasferire i 120 bytes della richiesta e i 219 bytes della risposta.

Conclusioni

La crescente diffusione delle tecnologie mobile e wearable ci pone davanti a innumerevoli nuovi scenari applicativi, inoltre l'utilizzo simultaneo di più dispositivi da parte dello stesso utente ci permette di identificare tale insieme di sistemi mobili come una Wireless Body Area Network all'interno della quale la questione principale è l'integrazione. In particolare la comunicazione e il coordinamento tra i diversi device che compongono la rete, risultano essere da un lato le sfide più interessanti dall'altro funzionalità primarie e sempre più necessarie.

Proprio per questo motivo con il lavoro di tesi si è voluto progettare e sviluppare un apposito middleware che ha come obbiettivo proprio quello di isolare in un layer, sopra quello radio trasmissivo Bluetooth, che supporti modelli di comunicazione e coordinamento di alto livello. Adottando Android come sistema operativo di riferimento è stato possibile renderlo universale in ambito mobile, wearable ed embedded, in quanto la maggior parte dei dispositivi di queste categorie hanno integrato il sistema sviluppato da Google. Supportando inoltre diversi pattern di scambio di messaggi e di coordinazione il suo impiego può avvenire a tutti tondo in una moltitudine di scenari applicativi, eventualmente andando anche ad ottimizzare le dimensioni dei pacchetti dati trasferiti, riducendo così i tempi di trasmissione. Nello specifico il middleware e l'idea di fondo del lavoro di tesi è scaturito dalla necessità di integrazione all'interno dei progetti portati avanti dal gruppo di ricerca, in particolare per *Trauma Tracker* all'interno del quale verrà sicuramente impiegato per favorire la comunicazione tra gli smart glass e il tablet. Infine al termine del progetto di tesi per esprimere la soddisfazione del lavoro compiuto si vuole concludere citando sommariamente i risultati della fase di validazione del middleware: 3000 ms per effettuare una nuova connessione, in media 1500 ms per ripristinarne una caduta in precedenza e infine attorno ai 50 ms per effettuare l'invio di un messaggio a prescindere dal modello di comunicazione.

6.6 Sviluppi futuri

In quest'ultima sezione verranno elencate alcuni possibili miglioramenti da applicare al middleware, che sono emersi in parte durante la fase di progettazione e in parte in quella di testing e validazione.

- *Bluetooth Low Energy*: come accennato precedentemente nella sezione riguardante la tecnologia bluetooth, nei prossimi anni potremo riconoscere una grande diffusione di dispositivi che integrano il BLE, motivo per cui potrebbe essere necessario rivedere la parte di basso livello del middleware adattandola al più recente standard di comunicazione.

  Ancora meglio potrebbe essere estendere il funzionamento dell'API su altre tecnologie radio trasmissive in modo da poter essere applicato trasversalmente nel settore mobile, embededd e wearable, a prescindere che il dispositivo abbia Bluetooth Classic, Low Energy, Wi-Fi, ecc.

- *specializzazione dei messaggi*: dato che il middleware è volutamente generico e vengono utilizzati dei JSON per la comunicazione, potrebbe essere necessario specializzare i messaggi, andando a incapsulare in questi oggetti specifici con caratteristiche e proprietà proprie delle informazioni da trasferire tra i dispositivi nel particolare settore di utilizzo, velocizzando così la fase di parsing da stringa a oggetto e viceversa e quindi del funzionamento di tutto il sistema.

Ringraziamenti

"Sono le scelte che facciamo, Harry,
che dimostrano quel che siamo veramente,
molto più delle nostre capacità"
Albus Silente, Harry Potter e la Camera dei segreti

Al termine di questo lavoro desidero ringraziare di cuore tutte le persone che mi sono state vicine negli ultimi mesi e nei tre anni di università.

Innanzitutto voglio dedicare questo risultato alla mia famiglia, in particolare ai miei genitori che mi hanno incoraggiato a intraprendere questo percorso formativo e hanno da sempre creduto in me, più di quanto non ne facessi io. A mio fratello e mia sorella che hanno reso più divertenti le giornate di studio.

Un ringraziamento particolare al Prof. Alessandro Ricci e all'Ing. Angelo Croatti che con grande passione e infinita disponibilità mi hanno accompagnato instancabilmente per tutto lo sviluppo del progetto, spronandomi a dare sempre il meglio di me.

A questo punto non posso certo dimenticare gli amici, quelli vecchi e quelli nuovi. Grazie quindi ai compagni d'università, conosciuti un po' per caso ma sempre più azzeccati per camminare insieme. Grazie dunque agli amici della diocesi e in particolare all'equipe giovani, che mi ha accolto proprio prima di intraprendere il percorso universitario e che in questi anni si è rivelata una preziosa palestra di vita. Infine grazie a tutti gli amici del borgo che ogni volta ricordandomi che sarei finito disoccupato mi facevano sentire il loro affetto e la loro vicinanza.

Condivido con tutti voi l'immensa gioia di questo traguardo che, vi assicuro, in parte è anche vostro.

Bibliografia

[1] G. H. Forman, J. Zahorjan, *The challenges of mobile computing*, Computer - vol. 27 - n.4 , 1994.

[2] S. Johnson, Nick Twilley, T. Zhang, Z. Zhouu, S. Wul, *Mobile computing*, Kluwer academic publishers Dordrecht, 1996.

[3] International Business Machines Corporation, *Ibm study finds consumers prefer a mobile device over the pc.*, `http://www-03.ibm.com/press/us/en/pressrelease/25737.wss`, 2008.

[4] B. Burd, Barros, J. P. Johnson, C. Johnson, S. Kurkovsky, A. Rosenbloom, N. Tillman, *Proceedings of the final reports on Innovation and technology in computer science education 2012 working groups*, ACM, 2012.

[5] I. Chlamtac, J. Redi, *Mobile computing: Challenges and potential*, Encyclopedia of Computer Science vol.4, 1998.

[6] M. Weiser, *The computer for the 21st century*, Scientific american - vol. 265 - n. 3, 1991

[7] A. S. Tanenbaum, T. Austin, *Architettura dei calcolatori. Un approccio strutturale*, Person, 2006

[8] D.Livingston, *Introduction & History of Mobile Computing*, Slideshare, Linkedin Corporation, 5 Dec. 2013

[9] G. Rusconi, *Semiconduttori in forte crescita, pc in discesa. Le memorie spingono le vendite oltre i computer*, Il Sole 24 Ore, 12 Ottobre 2017

[10] A. Dini, *Addio legge di Moore: i chip dei computer non corrono più come una volta*, La Stampa, 13 Febbraio 2017

[11] Ericsson, *Ericsson Mobility Report*, Stoccolma, Novembre 2017

[12] B. Liang, *Mobile edge computing*, Cambridge University Press, 2017

[13] T. Starner, *The challenges of wearable computing: Part 1*, IEEE Communications Magazine, 2001

[14] S. Mann, *Wearable computing as means for personal empowerment*, Proc. 3rd Int. Conf. on Wearable Computing (ICWC), 1998

[15] B. J. Rhodes, *The wearable remembrance agent: A system for augmented memory*, IEEE Communications Magazine, 1997

[16] B. Rhodes, *A brief history of wearable computing*, `https://www.media.mit.edu/wearables/lizzy/timeline.html`

[17] R. Cavallari, F. Martelli, R. Rosini, C. Buratti, R. Verdone, *A Survey on Wireless Body Area Networks: Technologies and Design Challenges*, IEEE Communications Magazine, 2014

[18] H. Cao, V. Leung, C. Chow, H. Chan, *Enabling technologies for wireless body area networks: A survey and outlook*, IEEE Communications Magazine, 2009

[19] M. Chan, S. H. Gonzalez, A. Valilakos, H. Chao, V. Leung, *Body Area Networks: A Survey*, Mobile Networks and Applications (MONET), Springer Netherlands, 2010

[20] Bluetooth Special Interest Group, *Bluetooth wireless technology surpasses one billion devices*, `www.bluetooth.com/news/pressreleases/2006/11/13/bluetoothwireless-technology-surpasses-one-billion-devices`, 13 Novembre 2006

[21] M. Foley, *Installed Base Of 4 Billion Products With Bluetooth Technology*, `https://blog.bluetooth.com/installed-base-of-4-billion-products-with-bluetooth-technology`, 22 Giugno 2011

[22] Bluetooth Special Interest Group, *From Prototype to Global Standard—Celebrating 20 Years of Market Creation*, `https://blog.bluetooth.com/celebrating-20-years-of-market-creation`, 26 Gennaio 2018

[23] K. V. S. S. S. S. Sairam, N. Gunasekaran, S. R. Reddy, *Bluetooth in wireless communication*, IEEE Communications Magazine, 2002

[24] R. Meier, *Professional Android 2 Application Development*, Wrox Pr Inc, 1 edizione, 2010

[25] Android Developers, *Bluetooth*, `https://developer.android.com/guide/topics/connectivity/bluetooth.html`

[26] Android Developers, *Room Persistance Library*, `https://developer.android.com/reference/android/content/SharedPreferences.html`

[27] Android Developers, *SharedPreferences*, `https://developer.android.com/topic/libraries/architecture/room.html`

[28] *Introduzione a JSON*, `https://www.json.org`

[29] L. Huan-Bang, Y. Kamya Yekeh, *Wireless body area network*, River publishers, 2010

[30] Slideshare, *2017 Digital Yearbook*, `https://www.slideshare.net/wearesocialsg/2017-digital-yearbook/`