UNIVERSITÀ DEGLI STUDI DI BOLOGNA

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

Reti di Calcolatori

INFRASTRUTTURA PER MONITORAGGIO E GESTIONE DI CANALI WIRELESS ETEROGENEI

Candidato:

Giuseppe Miniello

Relatore:

Chiar.mo Prof. Ing. Antonio Corradi

Correlatore:

Dott. Ing. Luca Foschini

Anno accademico 2004/2005

Parole chiave:

Wireless Internet
Context Awareness
Context Aware Middlewere
Vertical Handoff Management
WiFi e Bluetooth

Indice

Introduzione

In questi ultimi anni si è assistito ad una rapida crescita e diffusione delle tecnologie wireless che permettono di utilizzare normali strumenti come computer, telefonini, palmari per comunicare con altri dispositivi senza bisogno di nessun cavo e senza essere vincolati ad una posizione fissa.

In questo scenario, gli utenti desiderano accedere a tutti i servizi, acceduti abitualmente attraverso terminali e reti fisse, mediante terminali wireless e reti mobili. Ogni terminale mobile comunica con un Access Point (che indicheremo per brevità con la sigla AP), dispositivo che permette l'integrazione della rete wireless con una rete fissa, che è in grado di coprire un'area limitata e per garantire una copertura più vasta devono essere presenti più AP. In tale situazione avviene che mentre ci si allontana da un AP e ci si avvicina ad un altro si perde la comunicazione con il primo AP e si instaura una nuova comunicazione con il secondo. Questo fenomeno prende il nome di handoff. Se il passaggio avviene tra AP che utilizzano la stessa tecnologia l'handoff è detto orizzontale, se avviene tra AP con tecnologie differenti prende il nome di handoff verticale.

Con la possibilità di movimento sono però sorti nuovi problemi. In particolare, per alcune classi di applicazioni, come le applicazioni che forniscono un servizio multimediale, che richiede garanzie di continuità nella distribuzione del flusso multimediale, nasce l'esigenza di ottenere un servizio continuo, senza le interruzioni legate al cosiddetto fenomeno dell'handoff. L'handoff orizzontale è stato già gestito mediante un'opportuna infrastruttura ed ora si vuole realizzare un'infrastruttura per la gestione dell'handoff verticale.

Obiettivo di questa tesi è realizzare un'infrastruttura capace di recuperare e fornire in modo omogeneo informazioni sulle condizioni delle tecnologie wireless disponibili su un certo host, ad esempio WiFi e Bluetooth, in modo da semplificare lo sviluppo di servizi applicativi capaci di gestire in modo consapevole l'handoff verticale. In particolare l'infrastruttura attraverso l'introduzione di un'insieme di API (Application Programming Interface) dovrà semplificare l'accesso all'informazioni sullo stato delle connessioni disponibili e dovrà realizzare le funzionalità necessarie per la gestione diretta "da parte dei servizi applicativi" dell'handoff verticale.

Il primo capitolo fa una breve panoramica sul progetto introducendo lo scenario nel quale si andrà ad operare ed è trattato il fenomeno dell'handoff.
Il secondo capitolo introduce la struttura di una rete wireless, lo standard 802.11, il funzionamento del Bluetooth, e le problematiche legate alla gestione del vertical handoff; una panoramica sullo stato dell'arte della ricerca in quest'area conclude il capitolo.
Il terzo capitolo introduce l'architettura preesistente per quanto riguarda WiFi e BT ovvero le librerie per il WiFi realizzate dal DEIS e le librerie per il Bluetooth realizzate dal gruppo Mobilab di Napoli.
Il capitolo quattro ha lo scopo di evidenziare l'architettura adottata per realizzare l'infrastruttura e le relazioni fra le varie parti di essa.
Il capitolo quinto è invece dedicato all'implementazione dove sono evidenziate le varie classi introdotte e le relazioni tra queste.
Infine, il sesto capitolo descrive le modalità di sperimentazione e riporta i risultati.

Capitolo 1
Panoramica sul Progetto

1.1 Introduzione

Negli ultimi anni si è andato sempre più diffondendo l'uso di terminali mobili equipaggiati con interfacce multiple per la comunicazione wireless. Le tecnologie wireless più diffuse sono IEEE 802.11, nota anche come WiFi, Bluetooth e 3G; tali tecnologie hanno differenti caratteristiche in termini di larghezza della banda, copertura, costo di trasmissione ecc. Inoltre, a causa della natura volatile del mezzo trasmissivo (etere) delle tecnologie wireless, queste possono introdurre delle variazioni per quanto riguarda il ritardo, la larghezza della banda e la copertura del segnale. Per questo motivo i servizi devono conoscere le possibilità offerte dalle diverse tecnologie, la posizione e i movimenti degli utenti, in una sola parola *context-aware*. Tutto ciò richiede che le applicazioni siano in grado di controllare le condizioni della rete wireless, in modo da adattare il comportamento alla capacità ed al carico dell'infrastruttura wireless. In particolare, i movimenti degli utenti possono determinare degli handoff (situazione che si verifica nel momento in cui un utente passa da un punto di accesso verso un altro), i quali per essere gestiti hanno bisogno di un middleware in grado di recuperare e gestire tutte le informazioni sul contesto. Obiettivo di questa tesi sarà quello di realizzare un'infrastruttura che nasconderà gli aspetti di programmazione relativi alle diverse tecnologie e fornirà un insieme di API per accedere e controllare le informazioni di contesto e di basso livello non fornite da altre applicazioni.

1.2 Scenario Mobile

Un aspetto fondamentale da considerare nell'erogazione di servizi in reti wireless è rappresentato dalla mobilità dei terminali. Inizialmente, per lavorare o recuperare informazioni in rete, occorreva essere collegati alla rete fissa ed in questo modo non si aveva la possibilità di spostarsi da un luogo ad un altro. Con l'introduzione delle reti wireless si sono aperte nuove possibilità d'uso; gli utenti sono connessi alla rete fissa senza necessità di alcun collegamento fisico, come potevano essere i cavi. In questo modo è possibile utilizzare un dispositivo dotato di scheda wireless, per accedere alla rete o comunicare con altri dispositivi wireless, indipendentemente dalla loro posizione.

Per quanto riguarda le telecomunicazioni si hanno due tipi di mobilità: *del terminale*, ossia avere la possibilità di utilizzare la rete in modo continuativo spostandosi da un luogo ad un altro; *personale*, ossia avere la possibilità di fruire i propri servizi indipendentemente dal dispositivo di accesso.

Ci sono tre differenti tipologie di mobilità del terminale: *micro*, *macro* e *global.*

La micro è un tipo di mobilità supportata da una qualsiasi rete wireless, ed offre al terminale la possibilità di muoversi liberamente tra celle wireless appartenenti alla stessa sottorete.

Con la macro mobilità, un terminale ha la possibilità di muoversi tra diverse celle appartenenti al sottoreti diverse. Considerando la struttura dell'OSI, la micro mobilità è gestita a livello fisico e di data link mentre la macro mobilità è gestita a livello di rete. In questo modo, per la macro mobilità, si offre la possibilità di sviluppare soluzioni generali adattabili a qualunque rete mobile

La mobilità global, invece, offre al terminale la possibilità di muoversi da una cella ad un'altra, indipendentemente dal dominio, come mostrato nella seguente figura:

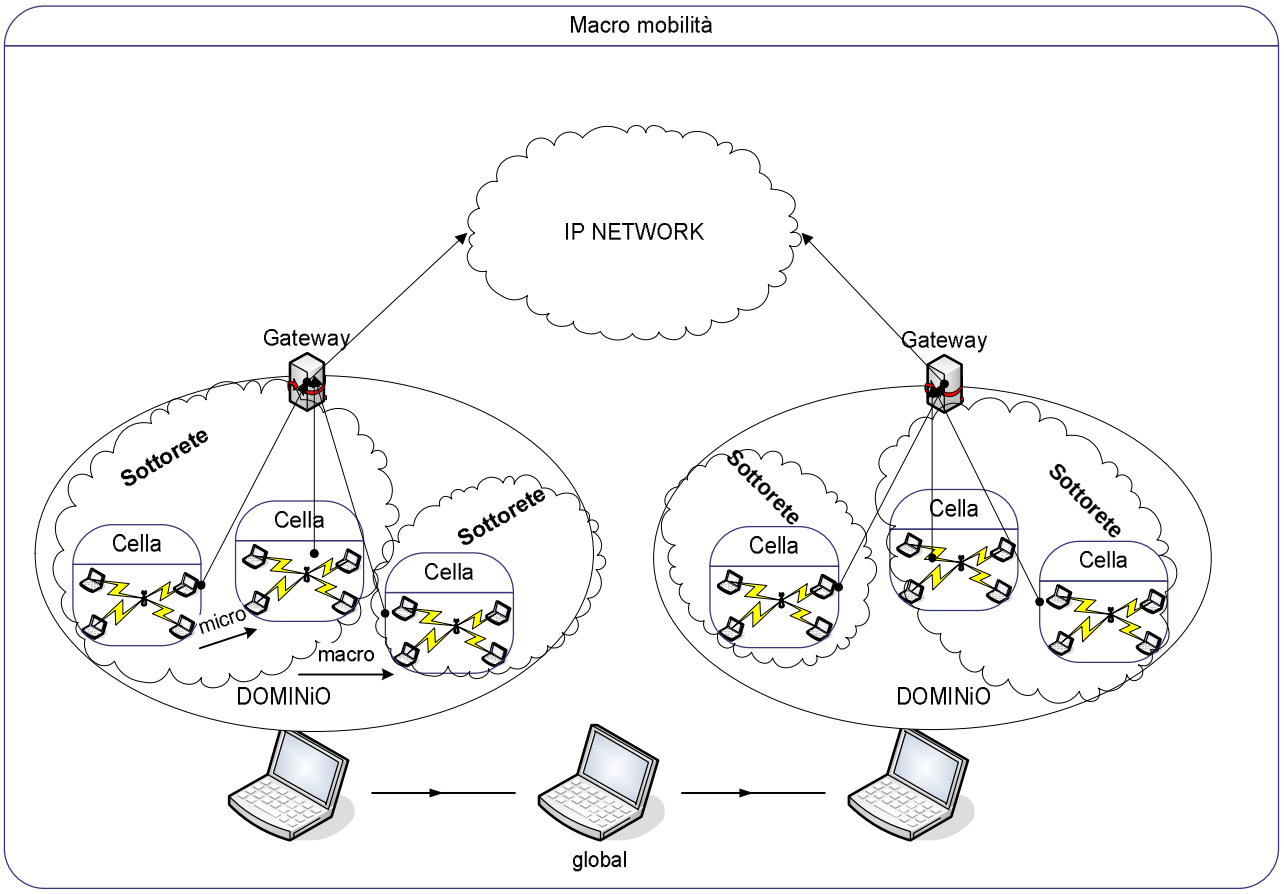


Figura 1.1: Esempio di global mobilità

1.3 Infrastruttura Wireless

Una WLAN (Wireless Area Network) è un sistema di comunicazione alternativo alla rete cablata o fissa detta anche Wired Lan. È possibile realizzare una rete wireless tra entità "pari" detta *AD-HOC*, in cui le varie entità possono comunicare direttamente tra loro in qualsiasi luogo e senza bisogno di nessuna infrastruttura a supporto. Questi tipi di collegamenti sono per lo più occasionali e di durata limitata nel tempo ed esulano dagli obbiettivi di questa tesi.

Al contrario, in questo lavoro di tesi consideriamo reti wireless infrastrutturate, come ad esempio la rete GSM, che estende la rete cablata con dei punti di accesso wireless, AP. Due entità che vogliono comunicare tra loro non possono farlo direttamente, ma utilizzano l'infrastruttura passando attraverso un AP, che metterà in comunicazione i due dispositivi. Le architetture wireless sono basate su due tipi di dispositivi, Access Point e Mobile Node (MN). Gli AP sono bridge che collegano i

dispositivi wireless con la rete fissa (solitamente ethernet), mentre i mobile node sono dispositivi che utilizzano i servizi di rete. Gli AP possono essere implementati in hardware o in software. I mobile node possono essere un qualsiasi tipo di dispositivo, che utilizzi lo standard IEEE 802.11, la tecnologia Bluetooth o 3G.
Nella seguente figura riportiamo un esempio WLAN infrastrutturato:

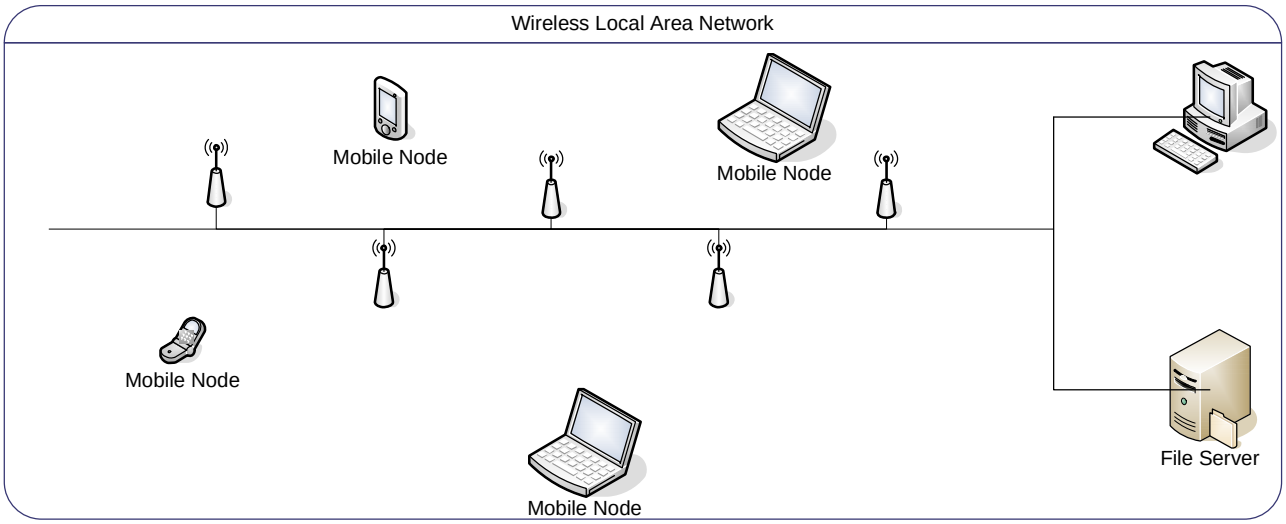


Figura 1.2: WLAN (Wireless Local Area Network)

1.4 Handoff

Ogni AP è in grado di coprire un'area limitata e per garantire una copertura più vasta devono essere presenti più AP. In tale situazione avviene che mentre ci si allontana da un AP e ci si avvicina ad un altro si perde la comunicazione con il primo AP e si instaura una nuova comunicazione con il secondo AP. Questo processo prende il nome di **handoff** e deve essere opportunamente gestito. In particolare l'handoff può verificarsi sia quando si cambia il punto di accesso alla rete fissa, condizione che si verifica quando un utente si sposta da una cella ad una adiacente, sia quando si cambia la tecnologia utilizzata, ad esempio quando si passa dalla tecnologia WiFi a Bluetooth. Il primo tipo di processo di handoff prende il nome di handoff *verticale* (si cambia tecnologia) mentre il secondo *orizzontale* (stessa tecnologia). In entrambi i casi si dovranno raccogliere tutte le informazioni necessarie per gestire nel modo opportuno questo processo. Le informazioni che interessano dipendono dalla tecnologia wireless utilizzata e dalle condizioni del canale di comunicazione, inoltre vi sono anche altre informazioni che variano dinamicamente quali la larghezza della banda, potenza del segnale ricevuto, AP corrente e stato della connessione (connesso, non connesso o handoff).

Per quanto riguarda l'handoff orizzontale, si possono individuare due politiche di gestione ossia:

- *Soft*: in base a questa politica il dispositivo, durante un handoff, può collegarsi contemporaneamente a più di un AP. In questo modo non saranno persi pacchetti durante l'handoff;
- *Hard*: con questa politica, durante l'handoff, il dispositivo può connettersi solamente ad un AP alla volta. In questo modo si avrà perdita di pacchetti durante l'handoff (questo tipo di politica è utilizzata in WiFi).

Per l'handoff verticale, che coinvolge differenti infrastrutture wireless, è possibile individuare due tipi di gestione, ossia:

- verso l'alto: quando l'handoff è verso una tecnologia wireless che utilizza celle di dimensione maggiori (es. da Bluetooth a WIFi);
- verso il basso: quando l'handoff è verso un'infrastruttura wireless costituita da celle di dimensione inferiore (es. da WiFi a Bluetooth).

Un'ulteriore suddivisione distingue gli handoff, sia orizzontale che verticale, in:

- *Reactive* (Reattivo): in questo caso si cerca di minimizzare il numero di handoff eseguendolo solamente se il segnale con l'AP è perso. In questo modo si avranno tempi di handoff più lunghi perché si dovrà prima cercare un nuovo AP e poi ci si dovrà connettere ad esso;
- *Proactive* (Proattivi): in questo caso si avvia la procedura di handoff prima di perdere il segnale con l'AP attuale, e cioè quando ci si accorge della presenza di un AP con un segnale migliore. In questo caso si avranno più handoff, ma con tempi minori.

Si noti che le strategie proattive determinano consumi di batteria maggiori, in quando c'è il monitoraggio continuo del segnale, mentre quelle reattive hanno dei costi minori, ma una durata di handoff maggiore.

1.5 Consapevolezza di contesto

Come accennato nei precedenti paragrafi, i dispositivi wireless hanno introdotto la possibilità di lavorare, utilizzare risorse di rete ed in generale interagire con altri componenti in movimento, svincolando l'utente dalla necessità di una locazione fissa dalla quale accedere a detti servizi. In questo ambiente totalmente dinamico nasce la necessità di ottenere continuamente informazioni sul contesto nel quale ci si trova per poter garantire la migliore qualità dei servizi. Si parla, in questo caso, di Context Awareness per indicare un'applicazione in grado di ottenere informazioni sul contesto operativo e di adattare il proprio comportamento in relazione ai cambiamenti dell'ambiente circostante. Il contesto di un'applicazione comprende oltre alle informazioni sulla locazione dell'utente anche altre informazioni, ad esempio, il livello di rumore, il costo della comunicazione, l'ampiezza di banda, ecc. La conoscenza di tali informazioni, permette di fornire servizi personalizzati per il particolare utente, luogo ed evento.

L'applicazione realizzata può così ottenere le informazioni sul contesto alle quali è interessata, quali lo stato della connessione in termini di disponibilità (connesso, non connesso ed handoff), di potenza del segnale (RSSI, *Received Signal Strength*

Indicator), di larghezza di banda e di ritardo, oppure lo stato della locazione in termini di AP correntemente utilizzato.

In base a tali informazioni, si possono conoscere le possibilità offerte da ogni tecnologia wireless, oltre alla posizione ed ai movimenti degli utenti: in questo modo sarà possibile prevedere e gestire nel modo opportuno le situazioni di handoff che si verranno a creare.

Obiettivo dell'infrastruttura che vogliamo realizzare è quello di fornire tutte le informazioni di interesse in modo omogeneo, indipendentemente dalla particolare tecnologia utilizzata. In questo modo le informazioni potranno essere utilizzate e trattate senza tenere in considerazione la tecnologia dalla quale queste informazioni sono state recuperate.

1.6 Introduzione all'architettura

Come già scritto, obbiettivo di questa tesi è quello di realizzare un'infrastruttura che fornirà un insieme di API per accedere e controllare le informazioni sul contesto e sull'handoff.

L'infrastruttura fornisce un insieme di servizi per recuperare le informazioni di interesse e per fare ciò utilizza le funzioni offerte dal Sistema Operativo sottostante come mostrato in figura:

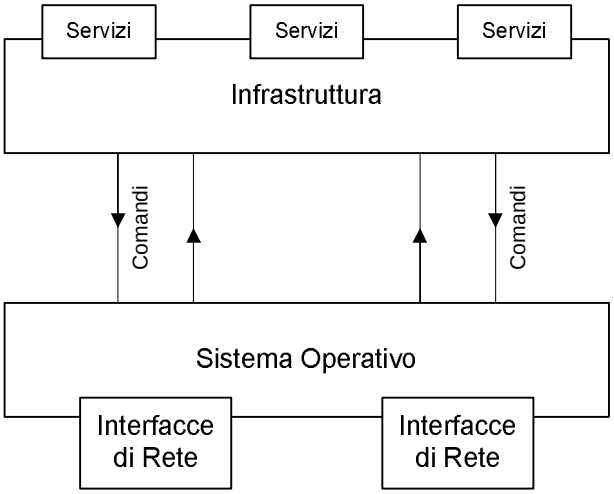


Figura 1.3 : Interazione con il Sistema Operativo

L'infrastruttura realizzata interagisce con il Sistema Operativo, recupera le informazioni di interesse relative alle diverse tecnologie utilizzate e le presenta in modo omogeneo. Queste informazioni possono essere o recuperate in conseguenza di una richiesta oppure fornite mediante eventi. In particolare l'infrastruttura può essere interrogata a polling, oppure può raccogliere le informazioni di interesse e comunicarle agli interessati mediante eventi.

Capitolo 2
Tecnologie Wireless

Con il termine wireless si intende un qualsiasi tipo di dispositivo in grado di comunicare, mediante onde radio o raggi infrarossi, con altri dispositivi senza l'ausilio di fili. Nell'ambito delle tecnologie per la connessione senza fili rientrano le tecnologie per la telefonia cellulare, quelle per la copertura di reti mondiali (Wireless WAN o **wWAN**) e le tecnologie per la realizzazione di reti locali (**wLAN** e **wPAN**). Le prime si riferiscono a standard quali **GSM** (Global System for Mobile communications), **GPRS** (General Packet Radio Services), **HSCSD** (High Speed Circuit Switched Data) ed **UMTS** (Universal Mobile Telecommunications System). Le seconde si riferiscono a tecnologie quali **Bluetooth** per le reti wPAN (wireless Personal Area Network) e lo standard **IEEE 802.11** per le reti wLAN (wireless Local Area Network), in particolare lo standard certificato dall'IEEE come **802.11b** noto con il nome di **Wi-Fi** (Wireless Fidelty).

La comunicazione fra i vari dispositivi, in ambiente wireless, può avvenire grazie ad antenne che fungono sia da trasmittenti sia da riceventi. Il vantaggio è quello di non avere la necessità di una postazione fissa consentendo una certa mobilità all'utente che ha la possibilità di connettersi alla rete ovunque si trovi con l'unico vincolo di rimanere all'interno della zona coperta dalla rete Wireless.

Sia per la tecnologia Wi-Fi sia per quella Bluetooth una caratteristica molto interessante che sarà considerata spesso durante il lavoro è l'**RSSI** (Receive Signal Strength Indication) e cioè un valore rappresentante la potenza del segnale ricevuto sia dal terminale mobile sia dalla stazione base (Access Point). In Bluetooth il range in cui tale valore varia è [-25; 6] mentre in WiFi dipende dalla scheda utilizzata e per una scheda cisco varia circa tra [130; 10] dove un valore maggiore in realtà indica un segnale peggiore.

2.1 WiFi

2.1.1 Storia dell'IEEE 802.11

Lo standard IEEE 802.11 utilizza principalmente la radiofrequenza nella banda centrata attorno ai 2,45 GHz utilizzando tecniche per ottenere una maggiore robustezza nei confronti di segnali interferenti. Tale banda è identificata come **ISM** (Industrial Scientific and Medical purpose), e, tenendo conto delle leggi in vigore nella maggior parte degli stati per poter trasmettere nella ISM non è necessario ottenere particolari licenze o permessi.

Nel 1997 nasceva il primo standard di riferimento, noto con il nome di IEEE 802.11 (IEEE Institute of Elettrical and Electronics Engineers), che dettava le specifiche a livello fisico e di datalink per l'implementazione di una rete LAN Wireless. Questo standard però consentiva una trasmissione di 1 o 2 Mbps usando una tecnologia basata su onde radio nella banda 2,4 GHz e ciò ne determinò uno scarso utilizzo. L'evoluzione di questa tecnologia portò allo standard **802.11b** (detto anche WiFi o Wireless Fidelty) consentendo una trasmissione dai 5,5 agli 11 Mbps e mantenendo la compatibilità con lo standard precedente, utilizzando quindi sempre la stessa frequenza di 2,4 GHz. Contemporaneamente all'802.11b nasce anche **l'802.11a** che è uno standard ad alto costo, ma che offre migliori prestazioni rispetto al precedente. Questo standard opera su una frequenza di 5 GHz ed ha una velocità di 54 Mbps ed è incompatibile con lo standard 802.11. Ultimo standard sviluppato è l'**802.11g** che unisce le migliori caratteristiche degli standard precedenti, infatti, utilizza una frequenza di 2,4 GHz, avendo così una migliore tolleranza agli ostacoli, e supporta una velocità di 54 Mbps.

2.1.2 Struttura di una rete Wireless

L'IEEE 802.11 definisce un insieme di specifiche per il livello MAC (Medium Access Control) e per il livello fisico (Phisical Layer) per la realizzazione di una Wireless Lan (WLAN). Una Wireless Local Area Network o WLAN è un sistema di comunicazione che può essere pensato come alternativo alla rete fissa o Wired Lan. Una rete wireless può essere un'estensione di una normale rete cablata supportando, con un Access Point, la connessione a dispositivi mobili o fissi. Le architetture wireless sono basate su due tipi di dispositivi, Access Point e Wireless Terminal. Gli Access Point sono bridges che collegano i dispositivi wireless con la rete fissa, mentre i wireless terminal sono dispositivi che utilizzano i servizi di rete. Gli AP possono essere implementati in hardware o in software utilizzando ad esempio un pc che è dotato sia di un'interfaccia wireless sia di una ethernet. I wireless terminal possono essere un qualsiasi tipo di dispositivo che utilizzi lo standard IEEE 802.11 o tecnologia Bluetooth. Nelle reti wireless è comunemente utilizzato una Modalità infrastrutturata che permette a più dispositivi di rete di appoggiarsi ad un Access Point che funge da ponte con la rete ethernet. Anche all'interno di una rete wireless e possibile effettuare il roaming verso un altro AP.

Ogni rete wireless è costituita da due elementi quali la cella ed il dominio. La cella è costituita da un insieme di postazioni wireless che sono coordinate e controllate da una stazione radio base, che rende accessibili i vari servizi e possibile la comunicazione tra i vari dispositivi della rete. È possibile estendere la rete mediante il coordinamento di più celle ed in questo modo è anche possibile mettere in comunicazione dispositivi che si trovano in celle differenti. L'unione di più celle è detta dominio e, se la rete si basa su un protocollo IP, il dominio coincide con la sottorete. Considerando ciò si può affermare che tutte le postazioni di un dominio condividono parte dell'IP.

2.1.3 802.11: servizi logici

Lo standard 802.11 mette a disposizione due tipi di servizi:

- **Station Service – SS**: disponibili ad ogni terminale indipendentemente dalla struttura adottata per la rete;
- **Distribution System Service – DSS**: disponibili al terminale wireless solo se presente una Infrastructure Local Area Network.

I servizi di base sono indispensabili per rendere possibile lo scambio tra due postazioni wireless. Il servizio **MSDU** (MAC Service Data Unit) scambia l'unità dati tra due postazioni wireless. Questo servizio è sufficiente per lo scambio di informazioni in reti ad hoc, mentre in una Infrastructure LAN sono necessari anche altri meccanismi.

Il servizio **Distribution** si occupa del recapito dei messaggi considerando la presenza di più BSS (Basic Service Set).

Ci sono poi i servizi per la sicurezza che hanno come obbiettivo quello di rendere più sicura una rete wireless. Il servizio di Authentication fornisce, ad una postazione, la possibilità di essere autenticata. Il servizio di Deauthentication è richiesto per revocare l'autenticazione. Il servizio Privacy ha come obbiettivo quello di rendere sicuro il canale wireless mediante appropriati algoritmi.

Infine abbiamo i servizi per la gestione delle associazioni che permettono di recapitare correttamente i messaggi al giusto destinatario. Per fare ciò si utilizza il concetto di associazione (unione dei MAC address del terminale mobile e dell'AP cui è collegato). In particolare il servizio Association è invocato dopo l'autenticazione e permette di creare una nuova associazione. Il servizio Reassociation rende possibile il roaming all'interno di un'ESS tra BSS. Il servizio di Disassociation comporta l'eliminazione dell'associazione.

2.1.4 Reti BSS ed ESS

Una rete **BSS** (Basic Service Set) è una rete in cui tutti i dispositivi presenti invece di comunicare direttamente tra loro comunicano con un Access Point. Quindi un pacchetto sarà trasmesso all'Access Point il quale provvederà ad inviarlo al giusto destinatario. In questo modo si occuperà più banda, ma si avranno altri vantaggi quali:

- L'Access Point individua una BSS e per poter accedere al servizio l'unica condizione è quella di trovarsi all'interno di quest'area;
- L'Access Point è in grado di accertare, quando una stazione è in uno stato di basso consumo. In questa modalità l'AP memorizza tutti i pacchetti, che devono essere inviati al dispositivo, in un buffer. Il dispositivo, in modalità basso consumo, disattiva l'antenna e la riattiverà solo per richiedere l'invio di tutti i pacchetti memorizzati. In questo modo si avrà un minor consumo di batteria.

Le reti BSS possono coprire solo aree limitate e per superare questo limite è possibile connettere più BSS per formare una rete di dimensioni maggiori chiamata **ESS** (Extented Service Set). Per garantire in tutta la ESS il servizio si tende a sovrapporre le aree della BSS in modo tale da non avere mai delle zone "d'ombra".

2.2 Introduzione a Bluetooth

Nel 1994, Ericsson Mobile propone una iniziativa per eliminare la necessità di utilizzare i cavi per connettere tra loro periferiche portatili come i telefoni e i computer palmari. Per promuovere uno standard comune a livello mondiale Ericsson contatta altri costruttori di soluzioni portatili e insieme a Nokia, IBM, Toshiba e Intel danno vita al Bluetooth Special Interest Group (SIG). Bluetooth è uno standard per le reti senza fili, una tecnologia a corto raggio sviluppata in modo particolare per discorsi e comunicazione di dati locali, che permette quindi di trasferire voce e dati tra due periferiche. Lo standard Bluetooth rende possibile la creazione di reti di dispositivi che scambiano dati, offrendo anche la possibilità di accedere a diversi servizi nell'ambito di quella che è chiamata PAN (Personal Area Network) o meglio WPAN, che è costituita da reti circoscritte e da dispositivi di dimensioni ridotte (computer desktop e notebook, cellulari, palmari, stampanti…). Tutti questi dispositivi hanno in comune il basso consumo di energia in quanto utilizzano una potenza di trasmissione molto bassa, non esigono connessioni veloci e non necessitano di connessioni ad una rete locale.

2.2.1 Funzionamento Bluetooth

Una delle principali caratteristiche di Bluetooth è quella di dover funzionare in qualunque parte del mondo e per questo motivo la banda di frequenza su cui lavora è di 2,45 GHz. Questa banda di frequenza, come visto nel paragrafo 2.1.1, è utilizzata anche dallo standard IEEE 802.11 e questo può causare interferenze nella comunicazione Wi-Fi, ma in questa tesi non ci occuperemo di questo aspetto. Per la trasmissione viene adottata una tecnica chiamata Frequency Hopping Spread Spectrum (FHSS), che divide la banda in tanti canali di "salto". Durante una trasmissione, trasmittente e ricevente saltano in frequenza da un canale all'altro secondo una sequenza pseudocasuale, ciò assicura una bassa interferenza sulla comunicazione ed inoltre il protocollo prevede degli algoritmi per la correzione degli errori.

Due o più unità Bluetooth, che condividono lo stesso canale, formano un piconet che può essere costituito da un massimo di 8 dispositivi. La disposizione di queste unità prevede un'unica unità master, che detta la sequenza di salto, e tutte le altre unità sono slave. I dispositivi dotati di tecnologia Bluetooth comunicano tra loro creando e riconfigurando dinamicamente la rete. Ciò rende possibile, ad esempio, sincronizzare i dati di un Pc portatile e di un PDA semplicemente avvicinando i due dispositivi oppure passare in modo automatico al vivavoce in macchina quando si entra e si parla al cellulare. Tutto ciò è possibile grazie al Service Discovery Protocol (SDP) che permette ad un dispositivo Bluetooth di determinare i servizi che gli altri dispositivi Bluetooth mettono a disposizione. In questo modo ogni nodo può conoscere le funzioni ed i servizi di ogni altro nodo nella rete.

Gli elementi base di un sistema Bluetooth sono la **RU** (unità radio) e la **BU** (unità base); i dispositivi comunicano tra loro in modo dinamico e la piconet si configura in maniera automatica, quando si inserisce o si elimina un dispositivo. Inoltre più piconet possono connettersi tra loro aumentando così la possibilità di espansione. Questa modalità di interconnessione dinamica può essere utilizzata per sincronizzare due apparecchi Bluetooth automaticamente sfruttando l'SDP il cui raggio di copertura è di alcune decine di metri.

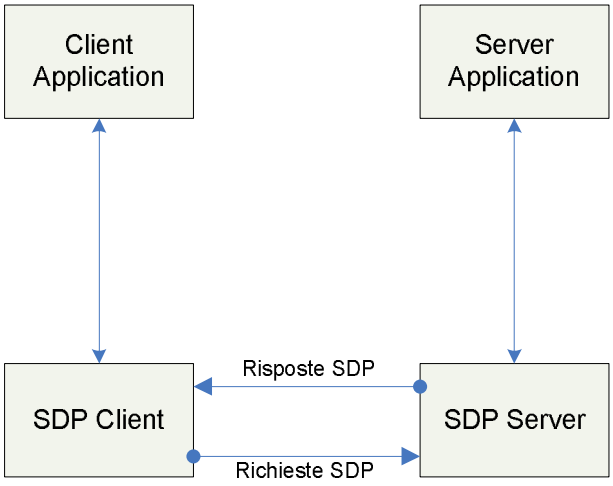


Figura 2.1: ***SDP Service Discovery Protocol***

Il protocollo SDP permette di determinare i servizi presenti e disponibili in una picorete sia in modalità client sia in modalità server. La modalità server consente le interrogazioni sui servizi e protocolli supportati e li renderà disponibili; la modalità

client invece consente l'interrogazione dei dispositivi connessi alla picorete per ottenere informazioni.

La tecnologia Bluetooth si offre di fornire servizi di rete ai livelli fisico e data-link del modello ISO/OSI di riferimento per le reti di calcolatori, e definisce le specifiche di uno stack di protocolli anche per i livelli più alti come riportato di seguito.

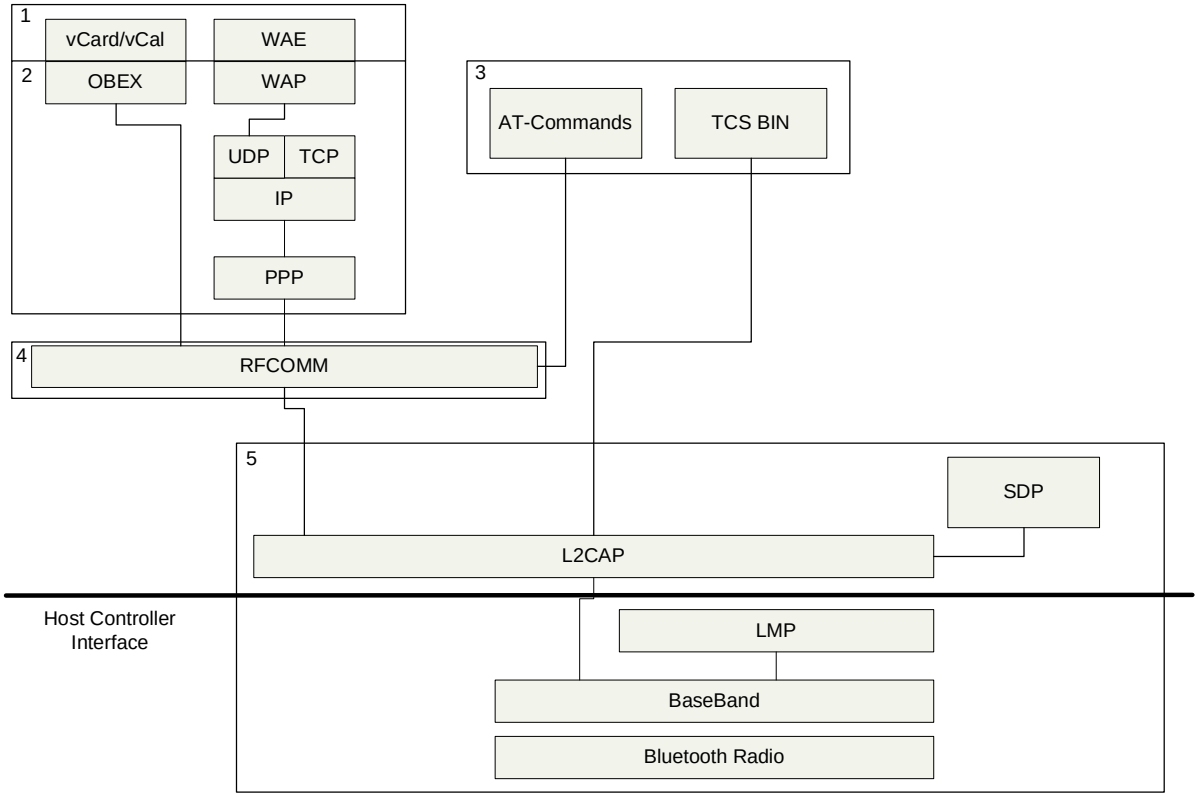


Figura 2.2: Stack di protocollo

I vari livelli dello stack sono implementati parzialmente in hardware, in modo particolare quelli vicino al mezzo fisico, e, parzialmente in software, in modo particolare quelli sopra dell'interfaccia host controller. Lo stack è stato concepito per consentire a vari dispositivi, costruiti da diversi produttori, di comunicare senza l'utilizzo di fili o cavi.

I protocolli di base formano uno stack a cinque livelli, come mostrato in figura 2.2, costituito dai seguenti elementi:

- **Radio Bluetooth**: specifica i dettagli di trasmissione delle onde radio nell'aria quali banda delle frequenze del segnale, uso di salti in frequenza, schema di modulazione e demodulazione etc…;

- **BaseBand**: relativo ad aspetti quali attivazione della connessione, indirizzamento, formato pacchetti, sincronizzazione e controllo potenza in un piconet;
- **Host Controller Interface**: gestisce la comunicazione tra un host e il dispositivo Bluetooth;
- **LMP – Link Manager Protocol**: attiva il collegamento fra dispositivi Bluetooth e gestisce il collegamento. Gestisce anche aspetti quali l'autenticazione, la crittografia, il controllo e la negoziazione delle dimensioni dei pacchetti in banda base;
- **L2CAP – Logical Link Control and Adaptation Layer**: adatta i protocolli superiori al livello in Banda Base. Fornisce servizi orientati alla connessione che lo rendono utilizzabile come livello di trasporto, allo stesso modo di come il TCP è utilizzato nell'architettura di internet;
- **SDP – Service Discovery Protocol**: fornisce informazioni sui dispositivi, sui servizi e sulle caratteristiche di essi che possono essere interrogati per consentire l'attivazione/disattivazione di una connessione fra più dispositivi Bluetooth;
- **RFCOMM – Radio Frequency COMMunication**: emula le caratteristiche della porta seriale RS-232, rende più trasparente la sostituzione delle tecnologie di trasmissione via cavo. Garantisce il trasporto dei dati binari ed emula i segnali di controllo RS-232 sul livello Bluetooth in Banda Base.

Nello stack sono anche presenti protocolli per la telefonia quali **TCS BIN** (Telephony Control Specification BINary), che definisce i segnali di controllo della chiamata per l'attivazione di collegamenti voce e dati fra dispositivi Bluetooth. Sono anche presenti protocolli che sono adottati da altre architetture come il **PPP (**Point to Point Protocol), i protocolli **TCP/UDP/IP** di Internet, **OBEX** per il trasferimento di file e i protocolli tipicamente wireless quali **WAE/WAP**.

2.2.2 Scansione della rete

Generalmente in un collegamento, tutti gli apparecchi Bluetooth sono in modalità standby, cioè attesa, ed effettuano una scansione ogni 1,28 secondi per verificare la presenza di eventuali altri dispositivi. In questa modalità i dispositivi Bluetooth sono a basso consumo energetico. I tipi di scansione che possono essere effettuati sono: **Page Scan** (**PE**) e **Inquiry Scan** (**IS**).

La scansione PE consente di ricercare un collegamento con un altro apparecchio Bluetooth e può essere in modalità *connectable mode* o *non-connectable mode.*

La scansione IS è simile alla PE e permette di identificare la tipologia di apparecchi presenti nella piconet e di approntare i necessari protocolli per il collegamento. Il comando inquiry è utilizzato quando l'indirizzo o il numero di identificazione di un dispositivo non è conosciuto e dopo aver effettuato il riconoscimento seguirà un comando di page che è utilizzato per risvegliare l'altro dispositivo ed instaurare una connessione.

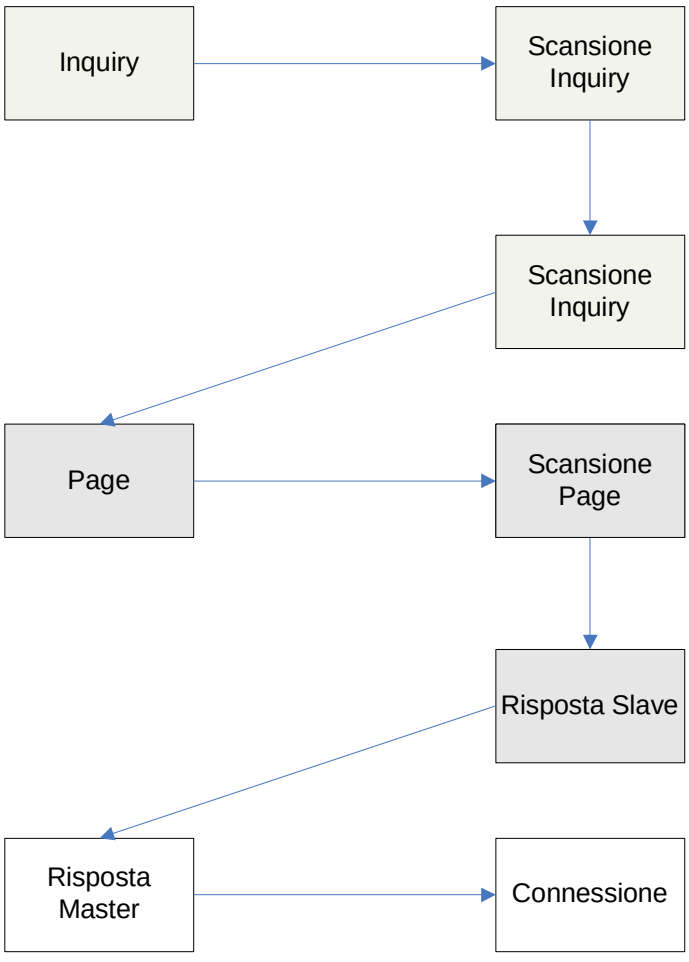


Figura 2.3: Fasi per la connessione

Una scansione può avere diversi risultati quali:

- Active, la connessione risulta essere attiva e può avvenire la trasmissione e la ricezione dei dati, gli slave sono sincronizzati con il master;
- Hold, può svolgere operazioni di IS e PS con basso consumo energetico
- Sniff, riduce il carico di lavoro in modalità di ascolto della piconet;
- Park è la modalità di attesa, rimanendo sincronizzato con la piconet.

2.2.3 Collegamenti Bluetooth

La tecnologia Bluetooth consente due particolari tipi di collegamento tra master e slave: **ACL** e **SCO**.

L'ACL (Asynchronous Connectionless) consente la trasmissione dei dati con una modalità asincrona. La velocità di trasmissione dei dati in modalità asimmetrica è di 723 Kbps e 57,6 Kbps nell'altra direzione mentre nella modalità simmetrica sarà pari a circa 434 Kbps.

Il collegamento SCO (Synchronous Connection Oriented) consente una trasmissione radio ed una voce. La velocità della trasmissione voce sincrona bidirezionale sfrutta una codifica voce *Continuous Variable Slope Delta Modulation* (***CVSD***) consentendo un bit rate di 64 Kbps.

Ogni master è in grado di gestire un totale di tre connessioni SCO simultanee verso slave, mentre gli ACL agiscono sui Time Slot liberi gestendo i dati generici.

I dati di una piconet sono trasmessi a pacchetti di 2745 bit e sono composti da un *Access Code* (AC), da un *Handler* (H) e da un *Payload* (P).

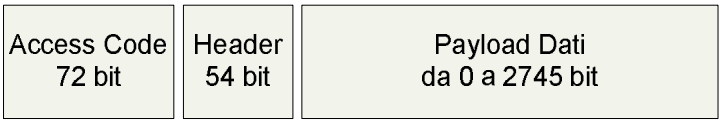


Figura 2.4: pacchetto bluetooth

Le modalità di trasmissione e ricezione dati possono variare in base alle esigenze di comunicazione delle unità Bluetooth, ad esempio passando da una sola comunicazione voce ad una dati. Una rete wireless composta da più dispositivi formerà una piconet e più piconet daranno vita ad un network wireless chiamato **Scatternet**. Due dispositivi vicini Bluetooth, che svolgono il ruolo di master, realizzano una scatternet su frequenze differenti ed ogni master gestisce gli slave della propria piconet.

2.3 Vertical Handoff

Le reti senza fili hanno introdotto la possibilità di utilizzare i servizi senza la necessità di una connessione fissa. Esistono diverse tecnologie wireless come, ad esempio, IEEE 802.11, Bluetooth, Hiperlan, GSM/GPRS e UMTS. Tutte queste tecnologie differiscono sotto vari aspetti ed in generale è possibile affermare che le reti senza fili con un'area di servizio piccola mettono a disposizione una larghezza di banda maggiore, mentre quelle con un'area di servizio maggiore forniscono una larghezza di banda minore.

Tradizionalmente il processo di handoff è effettuato fra due stazioni base con la stessa tecnologia. Questo tipo di handoff è definito handoff orizzontale.

A differenza dell'handoff orizzontale quello verticale è definito come l'handoff tra stazioni base con differenti tecnologie wireless. Il processo di handoff verticale si divide in tre fasi:

- nella prima, il terminale mobile deve conoscere quali sono le tecnologie wireless a disposizione;
- nella seconda, il terminale mobile esamina le tecnologie a disposizione per prendere una decisione sull'handoff;
- nella terza, il terminale mobile decide di effettuare l'handoff verticale ed esegue la procedura necessaria per associarsi ad una nuova rete senza fili.

Nella figura 2.5 è riportato lo schema dell'handoff verticale (rappresentato con una freccia continua), processo di cambiamento tra due differenti tecnologie, e dell'handoff orizzontale (rappresentato da una freccia discontinua), processo di cambiamento tra due celle.

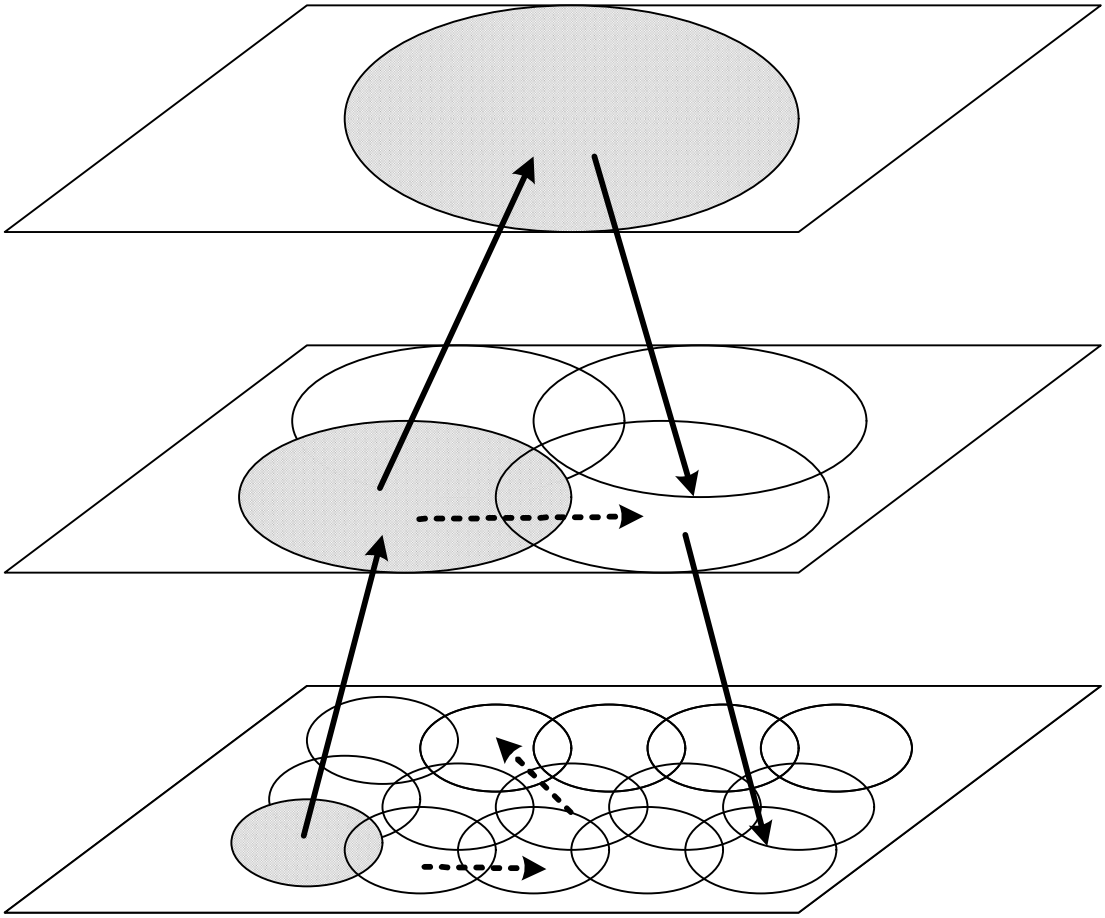


Figura 2.5: Handoff vertical ed orizzontale

Gli handoff verticali possono essere suddivisi in due categorie: verso il basso e verso l'alto. Negli handoff verso il basso il terminale mobile effettua il cambiamento da una tecnologia con celle di dimensione maggiori ad una con celle di dimensioni minori. Negli handoff verso l'alto il terminale mobile effettua il cambiamento da una tecnologia con celle di dimensioni minori ad una con celle di dimensioni maggiori. Il tutto è rappresentato nella figura 2.5. Generalmente l'handoff verticale verso il basso è meno critico rispetto a quello verso l'alto. Il terminale mobile ha necessità di effettuare l'handoff verso l'alto il più velocemente possibile se l'RSSI (potenza del segnale ricevuto) decresce velocemente. Il terminale mobile, invece, non effettua l'handoff verticale verso il basso immediatamente perché potrebbe prima spostarsi per breve tempo all'interno di una cella wireless e poi effettuare l'handoff verso il basso in modo da inserirsi in una cella con un segnale buono. Per questo motivo l'handoff verticale verso il basso è meno critico di quello verso l'alto.

Prendere decisioni relative all'handoff verticale è più complesso rispetto al prendere decisioni in merito all'handoff orizzontale. Un metodo di valutazione per l'handoff verticale richiede infatti la conoscenza di diversi parametri in modo da poter confrontare due reti senza fili, inoltre occorre evitare l'effetto ping-pong, fenomeno relativo al terminale mobile che effettua continuamente l'handoff tra due stazioni base. In ambienti omogenei, se la decisione di handoff è presa solo in base all'RSSI (potenza del segnale ricevuto), l'effetto ping-pong si può verificare sul bordo dell'area coperta da due stazioni base. Allo stesso modo l'effetto ping-pong in ambiente eterogeneo potrebbe verificarsi se il fattore di decisione cambiasse velocemente, il terminale mobile effettuerebbe l'handoff verticale immediatamente e, successivamente, sarebbe individuato un nuovo wireless network migliore del precedente.

2.4 Lavori di ricerca sul Vertical Handoff

Per risolvere i vari problemi legati all'handoff verticale sono state sviluppate diverse soluzioni derivanti da diversi lavori di ricerca; in questo paragrafo illustreremo brevemente tali soluzioni.

2.4.1 Active Application Oriented Vertical Handoff

Una prima soluzione realizzata da Wen-Tsuen Chen e Yen-Yuan Shu del dipartimento di scienza e computer dell'università National Tsing-Hua [AAO/05]. La soluzione si concentra sulla necessità, da parte del terminale mobile, di eseguire, nel giusto istante, l'handoff verso un network appropriato per ottenere la massima qualità di servizio (Quality of Service) e il miglior bilanciamento nell'utilizzo delle risorse. È quindi presentato uno schema che prende il nome di Active Application Oriented (AAO) il cui obbiettivo è quello di ottenere una gestione delle interfacce efficiente per offrire il miglior equilibro energetico ed eseguire l'handoff verso il network più adatto. Si fornirà al terminale mobile la possibilità di decidere attivamente quando eseguire l'handoff e a quale network collegarsi. Lo stile attivo permette un minor utilizzo di risorse e la decisione di handoff si basa

principalmente sull'applicazione in esecuzione in modo da avere un processo di handoff più accurato ed appropriato. È possibile eseguire l'handoff solo, quando il terminale mobile ne ha necessità, in questo modo lo spreco di risorse e batterie in seguito ad handoff non necessari sono ridotti al minimo.

2.4.2 Adaptive Schema

Un'altra soluzione consiste in uno schema adattivo che include i metodi per l'analisi della rete e per la decisione di handoff [AS/04]. Il metodo euristico proposto può equilibrare il tempo di ricerca nella rete e il consumo di energia. I metodi decisionali adattivi per la scelta dell'handoff possono essere sostituiti da metodi che migliorano le performances dello stesso:

- quando diminuiscono o l'indice di utilità o il periodo di stabilità che precede l'handoff;
- quando il precedente indice aumenta.

L'indice di utilità, calcolato da un'apposita funzione, si incrementa nel momento in cui si ha un effettivo incremento dell'ampiezza di banda della WLAN o la velocità di movimento del terminale mobile decresce, viceversa si decrementa.

Il periodo di stabilità indica il periodo di attesa del terminale mobile prima di effettuare l'handoff verticale.

In particolar modo, il secondo metodo per l'handoff adattivo comporta significativi miglioramenti della sensitività rispetto ai cambiamenti dell'indice di utilità ossia, si evitano handoff non necessari, quando tale indice è in rapida diminuzione.

2.4.3 Optimization Scheme

Un altro metodo si basa sul *pattern recognintion* e, grazie alle sue specifiche, questo algoritmo può essere un buon candidato per le reti wireless di prossima generazione [OS/05]. Tale metodo è usato per classificare una rete neurale probabilistica ed i risultati indicano che offre migliori performance rispetto ad un tradizionale algoritmo di classificazione. Nella simulazione effettuata è stato considerato un percorso rettilineo che attraversa varie celle con tecnologia WLAN e UMTS. Per massimizzare la larghezza della banda ricevuta dall'utente, la migliore strategia che può essere definita è massimizzare l'uso della WLAN. L'algoritmo proposto minimizza l'effetto ping-pong, incrementando leggermente il fattore d'uso della WLAN.

2.4.4 Optimization for VH Decision Algorithms

Un'ulteriore soluzione propone un'ottimizzazione per l'algoritmo decisionale per l'handoff verticale con l'obbiettivo di massimizzare i benefici dell'handoff sia per gli utenti sia per la rete [OVHDA/04]. Le ottimizzazioni comprendono caratteristiche che comportano una riduzione del ritardo e dell'elaborazione delle richieste nella stima della funzione costo e, per i multi-network, il miglioramento del rendimento di terminali mobili con più sessioni attive. Un'analisi delle prestazioni evidenzia miglioramenti nella qualità del servizio ed un miglior utilizzo delle risorse.

2.4.5 End-to-End Approach

Altra soluzione presenta un insieme di aggiunte e modifiche allo stack TCP/IP per migliorarne le performance nelle reti con collegamenti wireless e terminali mobili [EtE/04]. Questo protocollo lavora modificando il software del livello di rete alla stazione base ed al terminale mobile e non include ulteriori modifiche agli hosts fissi della rete. Le due idee alla base del nuovo protocollo hanno l'obbiettivo di gestire i problemi legati all'alto livello di errori nel trasferimento dei pacchetti nei collegamenti wireless ed ai dati persi causati dagli handoff. La soluzione ideata per

il primo problema è quella di memorizzare i pacchetti destinati al terminale mobile nella stazione base ed effettuare delle ritrasmissioni locali attraverso il collegamento wireless. È eliminata la perdita di pacchetti causata dalla mobilità utilizzando un algoritmo di handoff low-latency, basato sul multicast. Le sperimentazioni mostrano che questo stack di protocollo è significativamente più robusto del TCP normale in presenza di collegamenti non affidabili oppure errori multipli sulla finestra. Sono stati misurati miglioramenti nelle prestazioni per il trasferimento dei dati rispetto al protocollo TCP/IP dalla rete fissa al terminale mobile, ed è stata eliminata la perdita di pacchetti durante l'handoff.

Capitolo 3
Architettura preesistente

In questo capitolo sarà illustrata la struttura dell'architettura preesistente che si basa principalmente su tre entità: client, proxy e server. Il compito del proxy è quello di recuperare i dati dal server ed inviarli al client e questo può, a sua volta, inviare delle richieste al proxy. Per rendere meglio l'idea è riportata, in Figura 3.1, una rappresentazione grafica e seguirà una breve spiegazione delle funzioni di ogni parte.

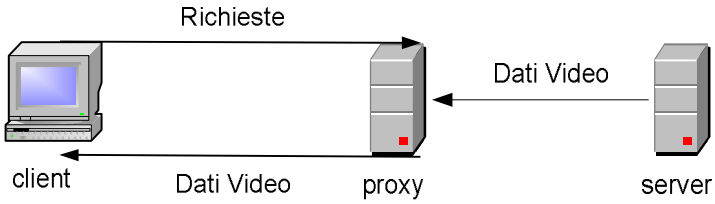


Figura 3.1: Struttura Proxy-based

3.1 Server

Il server riceve le richieste provenienti da diversi client e, durante la fase iniziale, sono scambiate alcune informazioni come l'identificatore del client, il nome del file di cui si dovranno inviare i dati per la riproduzione sul client e la velocità con cui sono trasmessi i dati.

3.2 Proxy UDP

Il proxy UDP, o semplicemente proxy, è il modulo che si occupa della gestione del file richiesto dal client. Ad ogni client che richiede un servizio è assegnato un identificatore, in questo modo può essere creata un'entità che si occupa di servire quello specifico client per tutta la durata del servizio. Il proxy può essere suddiviso in due moduli, uno che si occupa del riconoscimento dei client - chiamato *Proxy Manager* - e l'altro chiamato *Proxy* che si occupa di servire il client. Il proxy manager, quando un client si identifica e richiede un servizio, controlla se per quel client è stato già creato un proxy per la gestione del servizio, ed in caso contrario crea un proxy dedicato. È possibile rappresentare il tutto nel seguente modo:

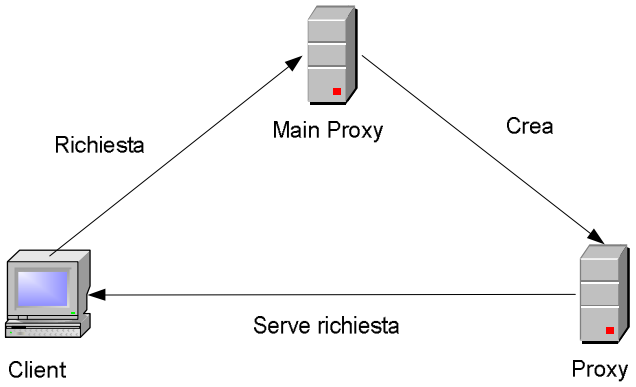


Figura 3.2: Struttura del Proxy

Una volta che è stato assegnato un proxy al client, è instaurato tra le due entità un canale di comunicazione privato, sul quale il client invia le richieste. Queste, una volta intercettate dal proxy, sono redirette verso il server che le elabora e ne restituisce al proxy i risultati. È previsto anche un meccanismo per la ritrasmissione delle informazioni al client da parte del proxy, il quale durante questa operazione deve anche memorizzare le informazioni che nel frattempo il server continua ad inviargli. Il proxy è un'entità dinamica in quanto deve adattare il proprio comportamento alle esigenze del client.

Come già accennato questo modulo è diviso in due classi principali: il *ProxyManger* ed il *Proxy*. Il primo raccoglie le richieste dei client gestendo i Proxy in modo opportuno. Se ad un client non è stato assegnato già un Proxy allora il ProxyManger ne crea uno e glielo assegna, altrimenti se risulta già essere

associato (controllando l'identificatore del client in una tabella), il Proxy è riassegnato al client.
Si può schematizzare il tutto nel modo seguente:

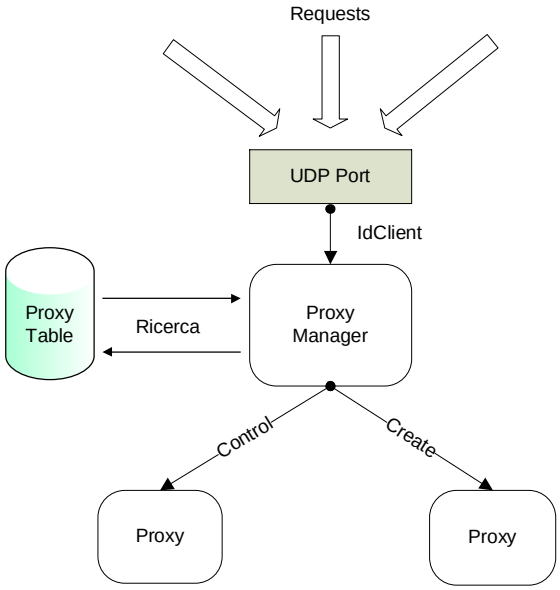


Figura 3.3: Struttura del Proxy

La ProxyTable è implementata dalla classe MemoryBroker, che ha il compito di tenere traccia di tutti i client che hanno richiesto i servizi e dei Proxy associati. Questa classe contiene inoltre un insieme di metodi utili per manipolare la tabella.

3.2.1 Proxy

È un componente attivo che è creato dal MainProxy e assume lo stesso identificatore del client che serve ed accetta solo pacchetti provenienti da esso. Questo componente crea un'istanza della classe AckController che ha come scopo quello di controllare l'arrivo di acknowledgement da parte del client. Se non arriva nessun acknowledgement entro 2,5 volte l'intervallo di tempo considerato, è terminato il proxy. Il Proxy contiene un insieme di metodi utilizzati per recuperare delle informazioni dal client e per effettuare operazioni come il rinvio di pacchetti, adattamento della dimensione del buffer, variazione della velocità di invio dei frames ed altre operazioni utili.

3.3 Client

Quando il client è avviato, deve fornire al proxy tutte le informazioni necessarie per servirlo nel modo opportuno. Per poter fornire informazioni utili, quali la marca della scheda wireless, informazioni sullo stato dell'utente, una parte del client deve interfacciarsi con la scheda wirless ed inviare tutte le informazioni al proxy. Questa entità si occupa anche della visualizzazione dei filmati che risiedono sul server e che sono trasmessi mediante un flusso continuo di dati. Provvede alla visualizzazione del filmato durante il trasferimento dei dati ed elimina quella parte di filmato già visualizzata in modo da avere spazio per i dati in ingresso. È inoltre previsto un modulo per la ricezione dei dati via wireless ed un altro per reperire i dati ricevuti e provvedere alla loro visualizzazione. Per poter visualizzare i dati deve essere riconosciuto il tipo di codifica giusta. Essendo in un contesto mobile possono verificarsi degli handoff, il client si accorgerà di questa situazione in seguito alla perdita dei dati e per risolvere questo problema è previsto un ulteriore modulo che chiede la ritrasmissione dei dati al proxy. Durante questo periodo di tempo il client continua la visualizzazione dei dati senza che l'utente si accorga di nulla. Un ultimo modulo è utilizzato per effettuare previsioni sugli spostamenti dell'utente e per questo sono recuperate tutte le informazioni necessarie dalla scheda, elaborate ed inviate al proxy che si adatta alle esigenze del client.
Il client invia al server la richiesta per il filmato, predispone le porte per la ricezione dei pacchetti, effettua la decodifica di questi, li visualizza ed infine interroga la scheda wireless per inviare al proxy la probabilità di handoff. Per svolgere queste operazioni il client è diviso in tre parti:

1. Player Video, si occupa della ricezione dello stream video, della sua riproduzione e della comunicazione con il proxy;
2. Client Stub, contiene un insieme di comandi di basso livello utilizzati per interagire con la scheda wireless e recuperare informazioni da essa. Per ottenere le informazioni di interesse è utilizzata l'entità WirelessCardController. Sono previsti due stati del client relativi alla probabilità di handoff: LOW_PROB e HIGH_PROB. In seguito ad un

cambiamento di stato il ClientStub risveglia il Controller che provvede ad inviare le informazioni al proxy.

3. Controller, ha il compito di avviare il Player e il ClientStub, stabilisce una connessione con il proxy e si pone in attesa di una risposta da parte di questo.

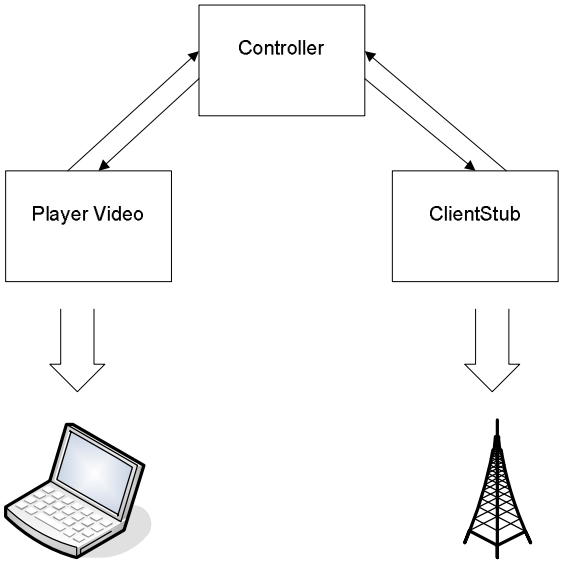


Figura 3.4: Struttura del Client

3.4 Simulatore

Per poter effettuare tutti gli esperimenti è stato introdotto un simulatore, che può essere diviso in due parti: la prima si occupa della simulazione dei segnali che sarebbero captati, nel mondo reale, dalla scheda wireless, e che proverrebbero dagli AP, la seconda invece recupera tutti i dati offerti dalla prima parte ed emula il comportamento di una scheda wireless. Lo schema che si ottiene è il seguente:

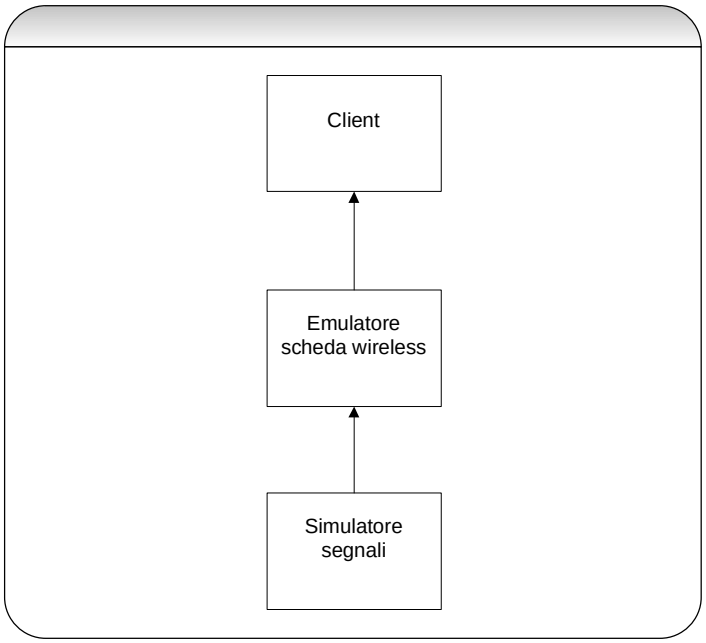


Figura 3.5: Emulatore schede wireless

Il simulatore inizialmente recupera tutti i dati relativi agli AP presenti e successivamente effettua un ciclo in cui calcola la potenza di ciascun segnale. I dati relativi ad ogni AP sono incapsulati in una struttura chiamata LocationOss che mantiene le informazioni quali il MAC e la potenza del segnale di un AP. Il Simulatore utilizza una socket TCP per trasmettere i dati; questa scelta è stata fatta in modo tale da poter utilizzare macchine diverse per eseguire i diversi processi. Sono poi valutate le potenze dei segnali ricevuti da ogni client e la politica della sua scheda (Hard Proactive o Soft Proactive). Il Simulatore indica a quale AP è logicamente collegato e poi, il programma che raccoglie i dati dal Simulatore, valuterà in maniera indipendente l'handoff.

3.4.1 Emulatore schede wireless

Questo componente ha il compito di fornire informazioni al client e simulerà il comportamento di una scheda wireless. In particolare, l'emulatore simula l'handoff e quindi la perdita di connessione che ne deriva. La decisione di effettuare l'handoff è presa in base ai dati forniti dal modulo sottostante ossia dal simulatore dei segnali. Per offrire i servizi principali delle schede wireless, questo modulo avrà dei metodi che permettono di ottenere informazioni quali gli AP visibili, le potenze di questi e il MAC dell'AP attuale.

3.4.2 Simulatore segnali

Questo modulo prevede un insieme di meccanismi che simulano la visione del mondo da parte della scheda wireless e fornisce informazioni che dipendono dalla posizione della scheda all'interno del contesto. Il contesto non deve essere fisso ma variare per poter rappresentare un insieme di situazioni e sperimentare al meglio la validità delle soluzioni adottate.

3.4.3 SimulatorStub

È il modulo intermedio che è a conoscenza delle caratteristiche del simulatore e delle richieste del ClientStub, come mostrato in figura:

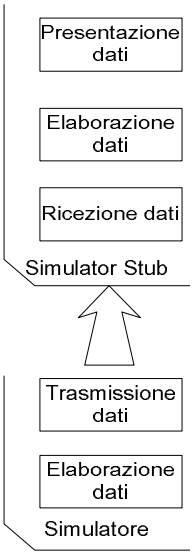


Figura 3.6: Comunicazione Simulatore-SimulatorrStub

La simulazione procede poi nel seguente modo:

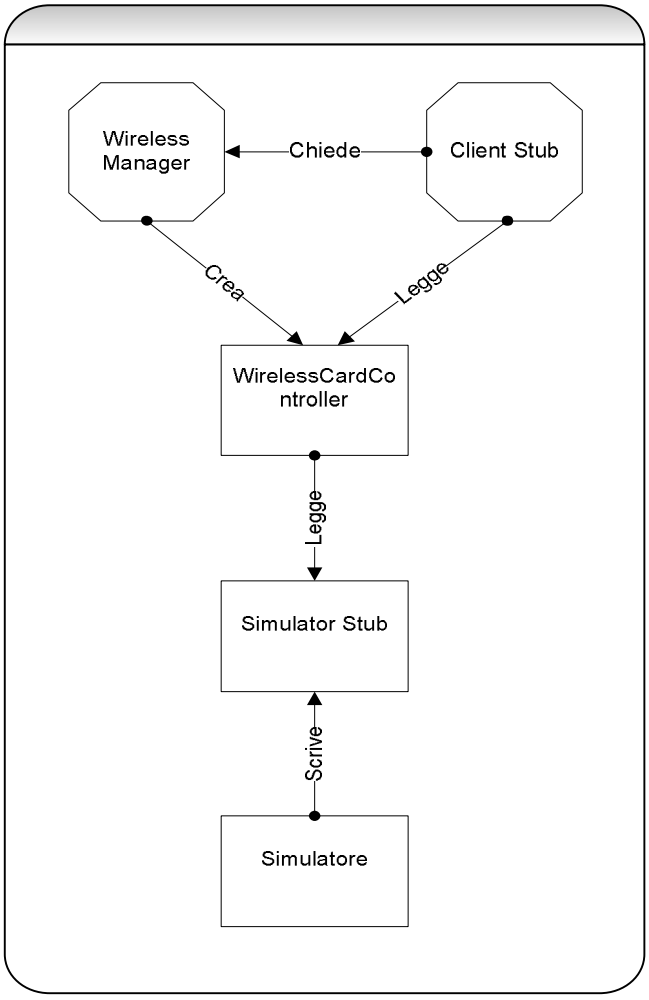


Figura 3.7:Creazione WirelessCardController

Il SimulatorStub riceve informazioni dal Simulatore ed interrogazioni da parte di altre entità. Questa classe è costituita da due server, il primo si pone in ascolto e riceve i dati dal Simulatore, il secondo accetta richieste da parte di un particolare tipo di WirelessCardController.

Il WirelessCardController è introdotta per fornire tutte le informazioni standard sulla scheda al ClientStub e svincolarlo così dalla conoscenza del tipo di scheda. Fornisce informazioni sulla presenza o meno di una connessione ad internet, sul tipo di scheda, restituisce l'AP a cui si è connessi o la lista degli AP visibili con i relativi RSSI.

In realtà al ClientStub verrà fornito un WirelessCardManger mediante il WirelessManager che è a conoscenza della reale situazione del sistema (se è in

atto una simulazione o è una vera e propria connessione). Restituisce o una istanza di LogRunner o di SimWirelessController.

Particolarmente importante è il SimWirelessController, modulo che implementa la WirelessCardController. Il WirelessManager alla richiesta del ClientStub restituisce una SimWirelessController come se si trattasse di un WirelessCardController. Inizialmente il controller comunica con il SimulatorStub e recupera alcune informazioni quali il tipo di Card e l'AP cui è connesso per la simulazione. Durante la simulazione chiede al SimulatorStub le informazioni sugli AP visibili e sull'AP cui è connesso, queste informazioni sono memorizzate per poter essere lette dal ClientStub. In seguito al cambiamento di AP è avviata una nuova entità che ha il compito di interrompere la comunicazione tra client e proxy.

Per l'invio di pacchetti è utilizzato l'HackDatagramSocketImpl che estende DatagramSocketImpl, provvede a creare una nuova istanza di PlainDatagramSocketImpl, che è la classe Java che estende ed implementa la classe astratta DatagramSocketImpl. In essa è presente una funzione utilizzata per settare un parametro interno che governa il comportamento di invio e ricezione di pacchetti.

Infine oltre al SimWirelessController è presente anche la classe LogRunner che è restituita al ClientStub dal WirelessManager sostituendo il SimWirelessController. Questa classe rappresenta un Thread che è lanciato e successivamente chiede i diritti di lettura sul file di log. È anche creato un nuovo Thread chiamato Wall che blocca le socket e le risveglia simulando, in pratica, l'handoff. Il LogRunner effettua il parsing delle righe dei files di log cambiando i valori degli AP attuali e di quelli visibili. Ad ogni ciclo è comunicata la lettura della riga successiva per poter calcolare il tempo di inattività prima di poter operare il successivo cambiamento. Quando è incontrata una riga che segnala l'handoff, il LogRunner configura il Wall per la durata prevista ed infine lo risveglia.

3.5 Classi del gruppo Mobilab

In questo lavoro sono state utilizzate anche due classi sviluppate dal gruppo Mobilab di Napoli il cui compito è quello di recuperare e fornire alcune informazioni sul contesto.

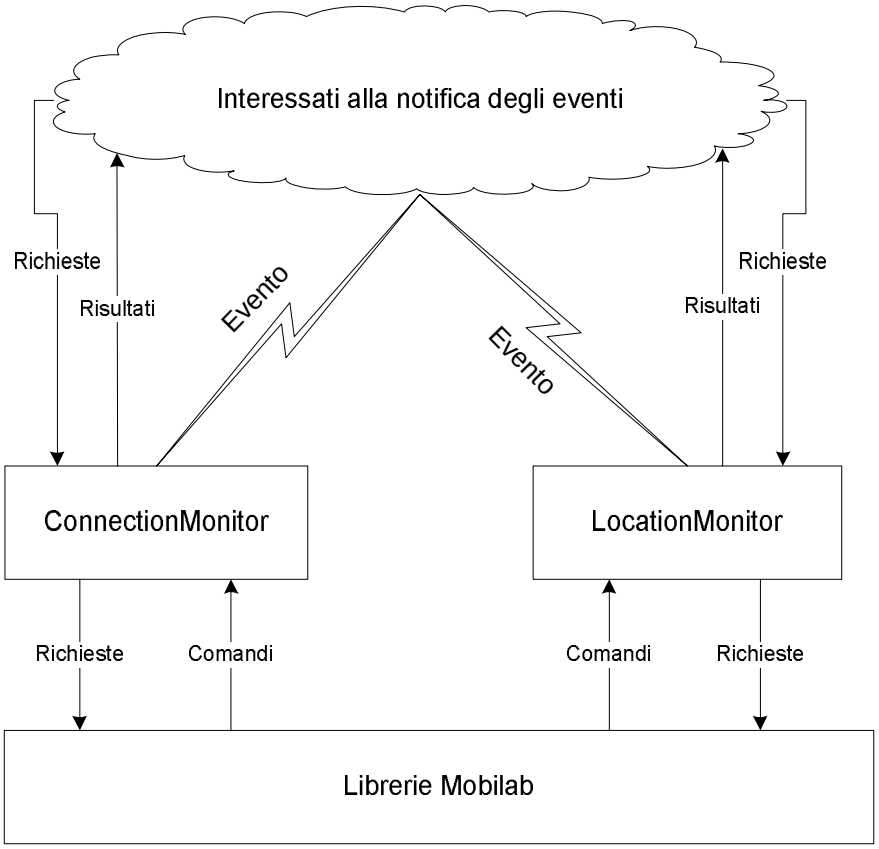


Figura 3.8: Classi del gruppo Mobilab utilizzate

Le entità ConnectionMonitor e LocationMonitor possono essere sia interrogate a polling sia fornire informazioni mediante eventi che saranno intercettati e gestiti in modo opportuno. In particolare queste due entità forniscono informazioni relativamente alla connessione (RSSI corrente, numero di connessioni create fino a questo momento, disponibilità canale, probabilità di handoff) ed alla locazione (nome e grado della locazione).

È stato anche utilizzato il demone “conmand”, entità sempre attiva sul lato client che, una volta attivata (mediante l’utilizzo dell’entità CLM), effettua la scansione utilizzando una tecnologia Bluetooth. Questo demone si occupa del rilevamento del demone Napd sul lato server, ed in caso questo esista instaura una comunicazione con quel server.

Il demone Napd utilizza Bluetooth Network Encapsulation Protocol (bnep), all'atto della creazione di una connessione di tipo Personal Area Network (PAN) il demone (distribuito con lo stack BlueZ, stack ufficiale bluetooth per Linux) crea un'interfaccia ad hoc che prende il nome di bnepx (dove x indica la x-esima connessione). Per consentire la comunicazione mediante una sola interfaccia virtuale il demone Napd fonde l'interfaccia virtuale bnepx e pan0, in modo che i pacchetti possono raggiungere la rete fissa attraverso l'interfaccia eth0.

Capitolo 4
Analisi del Progetto

In questo capitolo illustrerò in breve la struttura del progetto senza entrare nel dettaglio dell'implementazione dando un'idea dell'organizzazione dello stesso e delle funzionalità dalle varie entità.

4.1 Architettura

L'obiettivo da raggiungere è quello di fornire informazioni in modo omogeneo indipendentemente dalla particolare tecnologia utilizzata. È stata pensata un'architettura in cui si ha un'entità, che è chiamata Manager, il cui compito è quello di recuperare queste informazioni e fornirle agli interessati. Questa entità può essere sia attiva e sia passiva a seconda delle esigenze e delle risorse a disposizione. L'effettivo recupero delle informazioni di interesse è effettuato da altre due entità distinte, che sono chiamate BluetoothExecutor e WiFiExecutor entrambe entità passive, il cui compito è quello di recuperare alcune informazioni per il Bluetooth e per il WiFi e di fornirle al Manager.

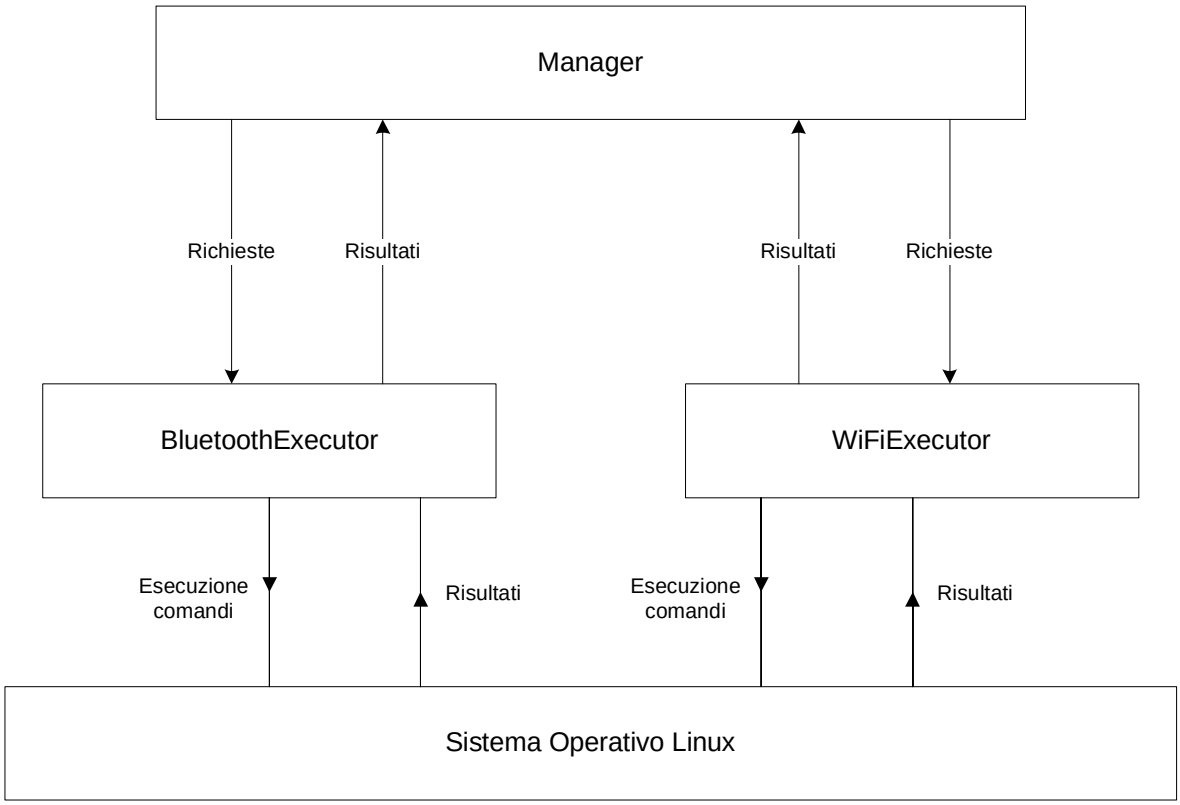


Figura 4.1: Architettura 1°Passo

Inoltre queste due entità si occuperanno anche dell'attivazione e disattivazione del dispositivo Bluetooth e WiFi e restituiranno al Manager due oggetti, rispettivamente BtInfo per il Bluetooth e WifiInfo per il WiFi, che incapsulano le informazioni di interesse. È stata prevista la possibilità, da parte del BluetoothExecutor, di fornire altre informazioni mediante un ulteriore oggetto chiamato BtDevice.

Il Manager consente inoltre di gestire l'accensione e lo spegnimento dei dispositivi di rete in base a politiche standard. A tal fine è stata introdotta un'ulteriore entità, chiamata PolicyType, contenente i diversi tipi di politica in base ai quali sono selezionati i dispositivi da attivare o disattivare.

La comunicazione tra Manager e lo strato superiore avviene mediante la notifica di eventi asincroni. Sono stati previsti alcuni tipi di eventi con lo scopo di fornire informazioni addizionali sul cambiamento dell'indirizzo IP del Client, della probabilità di handoff in Bluetooth e WiFi e per il cambiamento della locazione in Bluetooth. I parametri operativi degli eventi citati sono incapsulati all'interno di oggetti di tipo ClientAddressEvent, ConnectionEvent, HandoffProbBTEvent, HandoffProbWiFiEvent e LocationEvent. Le entità interessate alla notifica devono

implementare i relativi listeners e registrarsi presso il Manager. Le entità OnAddressEvent, OnHandoffBTEvent, OnHandoffWiFiEvent e OnLocationBTEvent, sono tutte entità attive e sono state introdotte per gestire il meccanismo di notifica degli eventi, in modo da evitare blocchi del Manager.

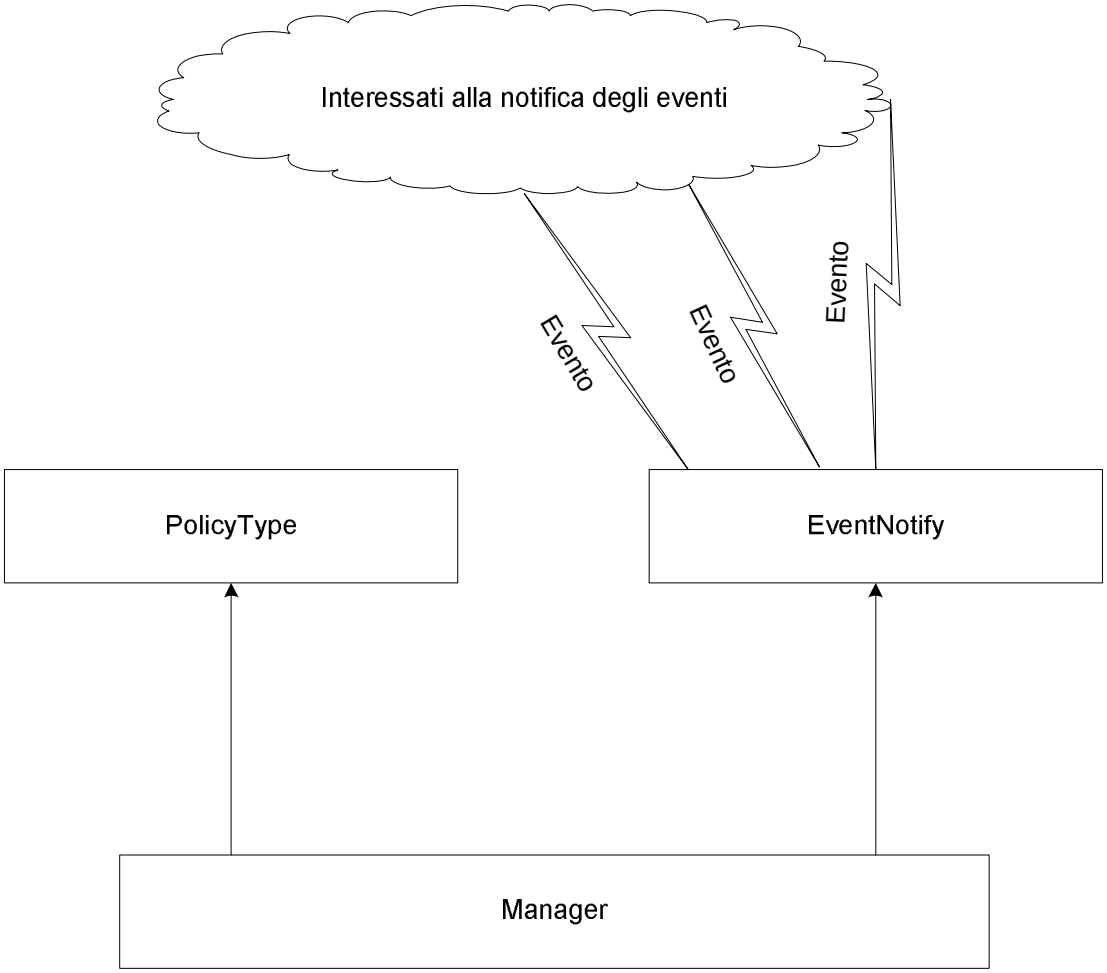


Figura 4.2: Architettura 2°Passo

Infine è stata prevista un'ulteriore entità che prende il nome di CLM utilizzata dal Manager per attivare o disattivare il demone Bluetooth denominato "conmand" e sono state utilizzate due interfacce, ConnectionMonitor e LocationMonitor, per interagire con la libreria legacy del gruppo Mobilab di Napoli.

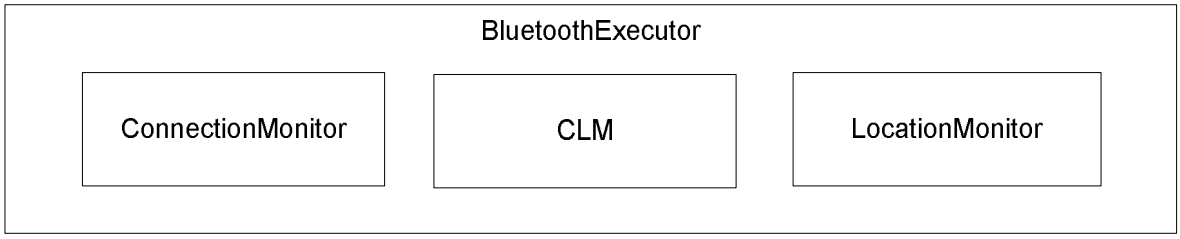


Figura 4.3: Architettura 3°Passo

Inoltre anche per quanto riguarda il WiFiExecutor sono state utilizzate entità preesistenti come il ClientStub.

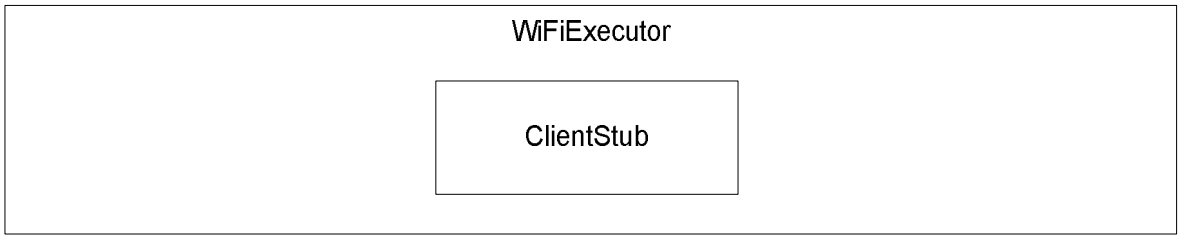


Figura 4.4: Architettura 3°Passo

4.2 Interfacce

4.2.1 IExecutor

Questa entità rappresenta le operazioni che gli Executor devono realizzare ed è implementata dal WiFiExecutor e dal BluetoothExecutor come mostrato da diagramma UML:

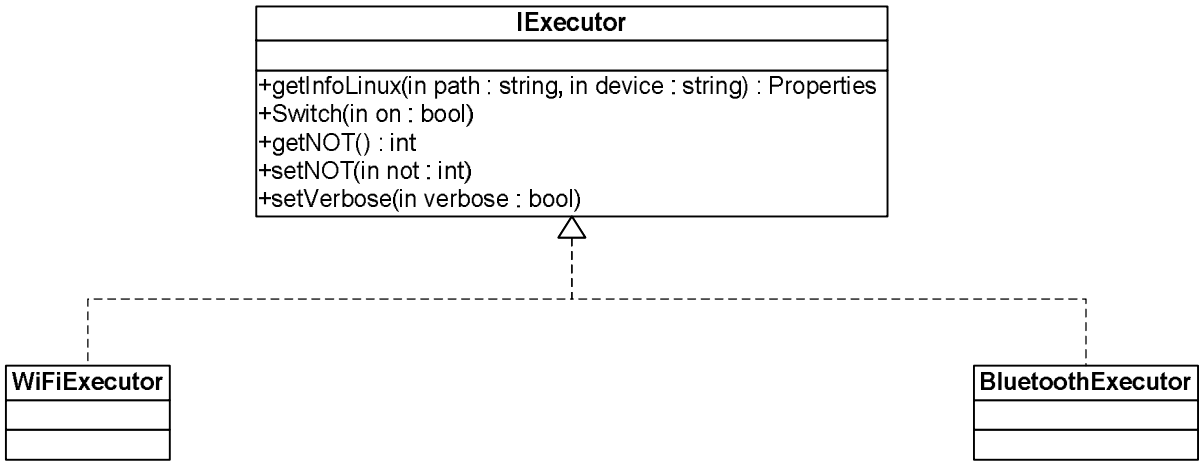


Figura 5.1: Interfaccia IExecutor

Contiene un insieme di servizi comuni alle due entità quali:

- `getInfoLinux(String path, String device)`: metodo utilizzato per recuperare le informazioni desiderate per i due dispositivi. Accetta come parametro due stringhe che rappresentano rispettivamente il percorso dal quale eseguire il comando, se necessario, e il nome logico del dispositivo da interrogare. Restituisce un oggetto di tipo Properties contenente coppie costituite dal nome dell'informazione e dal rispettivo valore;
- `Switch(boolean on)`: questo metodo serve ad attivare o disattivare il dispositivo. Accetta come parametro un booleano che se vero accende il dispositivo altrimenti lo spegne;
- `getNOT()`: utilizzato per recuperare il numero di tentativi da effettuare per accendere o spegnere il dispositivo;
- `setNOT(int not)`: è utilizzato per settare il numero di tentativi da effettuare per accendere o spegnere il dispositivo. Accetta come parametro un intero rappresentante il numero di tentativi;
- `setVerbose(boolean verbose)`: utilizzato per impostare la quantità di informazioni da visualizzare durante l'esecuzione dei metodi. Accetta come parametro un booleano che indica se attivare o no la modalità verbosa.

4.2.2 ClientAddressListener e HandoffProbListener

Queste due interfacce rappresentano il contratto che deve essere realizzato dalle entità le quali vogliono gestire l'evento per il cambiamento dell'IP del Client e per il cambiamento della probabilità che si verifichi un handoff in Wi-Fi o in Bluetooth.

4.2.2.1 ConnectionListener e LocationListener

Queste due interfacce rappresentano il contratto che deve essere realizzato dalle entità che intendono gestire l'evento per il cambiamento della probabilità di handoff e per il cambiamento di locazione in Bluetooth. Queste sono utilizzate dal Manager per poter essere notificato in conseguenza della notifica dei due eventi. Quando questi si verificano il Manager non fa altro che notificarli a tutte le entità che si sono registrate presso di lui.

Capitolo 5
Implementazione del Middleware

In questo capitolo entreremo nel dettaglio dell'implementazione mettendo in evidenza i dettagli e le scelte che sono stati effettuati. Nella prima parte del capitolo sono introdotti gli strumenti utilizzati per eseguire i comandi mentre nella seconda parte sono introdotte e descritte le entità progettate.
Come introdotto brevemente nel capitolo uno, l'obiettivo dell'infrastruttura è quello di recuperare un insieme di informazioni per i dispositivi wireless e fornire servizi per gestire nel modo opportuno l'handoff verticale.
Nel capitolo saranno descritte in dettaglio le operazioni principali delle varie classi, come queste interagiscono e riporteremo anche del codice relativo a soluzioni particolari che sono state adottate. Per leggibilità ed interesse non saranno riportati tutti i metodi con le relative funzionalità, ma saranno riportati solo quelli di maggiore interesse.

5.1 Strumenti

Per poter recuperare le informazioni di interesse sono stati utilizzati alcuni comandi del Sistema Operativo Linux che fanno parte di due librerie per la gestione di interfacce di rete wireless: Wireless Tools e BlueZ.
Le estensioni wireless per Linux e il Wireless Tools (WT) sono progetti Open Source. Le Wireless Extension (WE) sono un insieme di generiche API che permettono ad un dispositivo di fornire le informazioni relative alla rete. Il WT, invece, è un insieme di strumenti che permettono di manipolare le WE ed hanno un'interfaccia testuale. Questi strumenti sono stati pensati per essere il più possibile simili ai comandi utilizzati per una normale interfaccia di tipo la ethernet. In particolare, sono stati utilizzati i comandi *ifconfig* e *iwconfig*; il primo è utilizzato per accendere o spegnere il dispositivo WiFi mediante la sintassi "ifconfig

nomeDevice up/down” mentre il secondo è utilizzato per recuperare le informazioni di interesse mediante la sintassi “iwconfig nomeDevice”.
Lo stack BlueZ è utilizzato, nel Sistema Operativo Linux, per recuperare le informazioni di interesse per il Bluetooth. Lo stack è stato creato da Qualcom e rilasciato con licenza Open Source, BlueZ è lo stack Bluetooth maggiormente più diffuso. BlueZ è composto da alcune librerie che servono a gestire i dispositivi Bluetooth ed in particolare sono stati utilizzati alcuni comandi quali *hciconfig* ed *hcitool*. Il primo è utilizzato per accendere o spegnere il dispositivo mediante la sintassi “hciconfig nomeDevice up/down”, il secondo, invece, è usato per recuperare informazioni sul’RSSI, per connettere un dispositivo ad un altro e per disconnetterlo.

5.1.1 Esecuzione comandi shell da Java

Utilizzando Java per sviluppare l’infrastruttura è necessario riuscire ad eseguire i comandi che sono stati introdotti nel paragrafo precedente; ciò è possibile mediante l’utilizzo della classe Runtime. Un’istanza di questa classe offre all’applicazione la possibilità di interfacciarsi con l’ambiente nel quale l’applicazione è in esecuzione. L’esecuzione del comando mediante la classe Runtime determina la creazione di un nuovo processo e da questo è possibile recuperare le informazioni di interesse.

5.2 OnHandoffEvent, OnHandoffBTEvent, OnHandoffWiFiEvent, OnAddressEvent e OnLocationBTEvent

OnHandoffEvent rappresenta la classe radice per la gerarchia rappresentata in figura 5.2, è una classe astratta che estende Thread ed è per questo utilizzata come un’entità attiva per effettuare la notifica dell’evento relativo al cambiamento della probabilità di handoff a tutte le entità interessate. È stata effettuata la scelta di utilizzare un’entità attiva per la notifica degli eventi per evitare un blocco sincrono del Manager durante la notifica degli stessi cosa che accadrebbe utilizzando questa entità come una passiva.

Si riporta di seguito il diagramma UML associato:

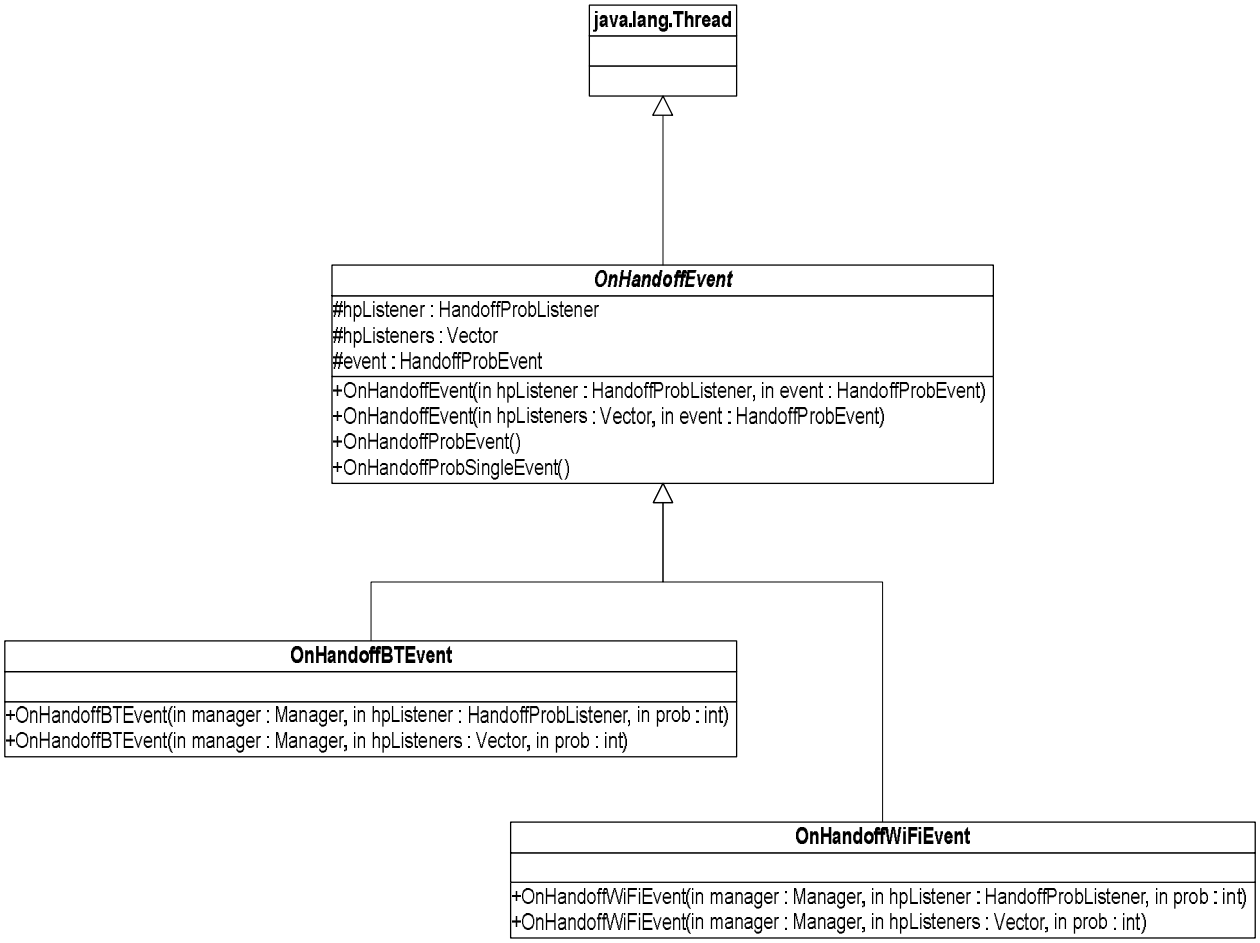


Figura 5.2:Gerarchia OnHandoffEvent

La classe OnHandoffEvent è utilizzata per effettuare la notifica per l'evento relativo al cambiamento della probabilità di handoff. Sono presenti due costruttori dei quali il primo accetta come argomento un listener per l'evento relativo alla probabilità di handoff ed un generico evento del tipo HandoffProbEvent. Il secondo costruttore accetta come argomenti un vettore di listener ed un generico evento del tipo HandoffProbEvent. Questa classe è utilizzata per effettuare la notifica, per l'evento relativo al cambiamento della probabilità di handoff, sia ad un singolo listener sia ad un gruppo di listeners.

Sono inoltre presenti due metodi quali:

- OnHandoffProbEvent(): utilizzato per la notifica dell'evento a tutti i listeners contenuti nel vettore;
- OnHandoffProbSingleEvent(): utilizzato per la notifica ad un singolo listener.

Le classi OnHandoffBTEvent e OnHandoffWiFiEvent estendono entrambe OnHandoffEvent e sono utilizzate, rispettivamente, per la notifica dell'evento di cambiamento della probabilità di handoff in Bluetooth ed in WiFi. Entrambe sono dotate di due costruttori dei quali il primo accetta come parametri un riferimento al Manager, un listener da notificare e un intero che rappresenta la probabilità di handoff. Il secondo invece, accetta come parametri un riferimento al Manger, un gruppo di listeners che devono essere notificati ed un intero che rappresenta la probabilità di handoff.

La classe OnAddressEvent è utilizzata dal Manager per la notifica dell'evento relativo al cambiamento dell'indirizzo IP a tutte le entità interessate, mentre la classe OnLocationBTEvent è utilizzata per la notifica dell'evento relativo al cambiamento della locazione in Bluetooth, a tutte le entità interessate. Anche queste ultime classi sono dotate di due costruttori, simili a quelli descritti in precedenza, che accettano un listener e le informazioni relative agli eventi da notificare. Sono anche dotati di due metodi per la notifica sia ad un singolo listener sia ad un gruppo di listeners.

5.3 Implementazione del Manager

Come accennato nel capitolo relativo all'architettura, per raggiungere l'obbiettivo preposto, si è introdotto un'entità principale chiamata Manager. Questa classe raccoglie le informazioni di interesse utilizzando altre due entità importanti quali il BluetoothExecutor ed il WiFiExecutor. È stata pensata per poter essere utilizzata in due modalità:

1. modalità attiva: grazie alla struttura ereditata dalla classe Thread, è possibile lanciare il Manager come un processo separato, invocandone il metodo `start()`. Durante l'esecuzione, esso raccoglie le informazioni di interesse e le comunica alle entità interessate attraverso degli eventi.
2. modalità passiva: è possibile utilizzare la classe Manager come una normale classe java; in questo caso, per recuperare le informazioni di interesse, la si dovrà interrogare a polling, invocando quindi direttamente i metodi specifici.

Con la prima modalità si ha il vantaggio di un monitoraggio continuo delle informazioni: in questo modo saranno continuamente aggiornate, al costo però di un processo in più in esecuzione sulla macchina.
Con la seconda modalità, le informazioni non saranno aggiornate in tempo reale ma solo su richiesta: si ha però il vantaggio di non avere un ulteriore processo in esecuzione, alleggerendo il carico di lavoro della macchina. Ciò può essere di estrema importanza nel caso in cui l'elaborazione avviene su sistemi con risorse limitate.
Per limitare il carico computazionale della classe, si è pensato di introdurre un insieme di parametri per settare delle soglie in base alle quali sarà deciso se eseguire un comando di sistema oppure restituire il risultato precedente dello stesso. Per realizzare questo meccanismo sono previste due Hashtable, nella prima (metodTime) è memorizzato l'istante dell'ultima esecuzione del comando utilizzando come chiave il nome dello stesso, mentre nella seconda (opTimesBT) è memorizzato il tempo impiegato per l'esecuzione utilizzando come chiave il nome del comando. Quando è eseguito un comando, si controlla inizialmente se è stato

già eseguito verificando se nella prima Hashtable è presente un valore riferito al metodo stesso, se presente, si determina il tempo passato dall'ultima esecuzione, calcolandone il rapporto con la durata media di ogni operazione. Se il rapporto supera una certa soglia (DataRefreshRate) è eseguito di nuovo il comando, altrimenti è restituito il precedente risultato. Inoltre, sempre per alleggerire il carico computazionale del Manager, è adottato il meccanismo di delega in base al quale i vari metodi non sono eseguiti direttamente dal Manager, ma sono delegati ad altre entità. In tutti i metodi, che utilizzano funzioni per recuperare informazioni relative ai dispositivi WiFi e Bluetooth, si procede inizialmente al controllo del tipo di sistema operativo e, successivamente, in caso di sistema Linux, si procede con l'esecuzione dei metodi.

Il metodo riportato di seguito evidenzia la gestione descritta in precedenza:

```
public BtDevice getBtDevice() throws InterruptedException
  {
    //si recupera l'istante corrente
    long now = System.currentTimeMillis();
    double ris = 0;
    long preOp = 0, postOp = 0, opTime = 0;

    //è verificato se il commando è stato già eseguito
    if(metodTime.containsKey("getBtInfo"))
    {
      //se già eseguito si calcola il tempo passato dall'ultima esecuzione
      millis = now - ((Long) this.metodTime.get("getBtInfo")).longValue();
      // si recupera il tempo che si impiega ad eseguire il comando
      double optime = this.getOpTimeBt("getBtInfo") + 1
      ris = millis / optime;
    }
    if((ris > this.DataRefreshRate) || !(metodTime.containsKey("getBtInfo")))
    {
      //si memorizza l'istante di ultima invocazione del comando
      metodTime.put("getBtDevice", new Long(now));
      try
      {
        if (!osName.startsWith("Windows"))
        {
          //si esegue il commando e si calcola il tempo impiegato ad eseguirlo
          preOp = System.currentTimeMillis();
          this.btStart(true);
          this.btDevice = bluetoothExecutor.getBtDevice();
```

```
          postOp = System.currentTimeMillis();
          opTime = postOp - preOp;
          //se è la prima volta che si esegue il comando
          if(!opTimesBT.containsKey("getBtDevice"))
          {
            //si utilizza un vettore per memorizzare le ultime durate in modo da poterne
            //calcolare la media
            Vector app = new Vector();
            //si aggiunge al vettore la durata del comando
            app.add(new Double(opTime));
            //si memorizza il vettore
            opTimesBT.put("getBtDevice", app);
          }
          else
          {
            //si recupera il vettore contenente le ultime durate
            Vector v = (Vector)opTimesBT.get("getBtDevice");
            //se il vettore ha tanti elementi quanti sono quelli utilizzati per
            //per calcolare la media
            if(v.size() == this.elMedia)
              //si rimuove l'elemento più vecchio
              v.removeElementAt(0);
            //si memorizza il nuovo elemento
            v.add(new Double(opTime));
          }
        }
        else
          System.out.println("Il sistema utilizzato non è Unix. Impossibile eseguire i
comandi.");
      }catch(IOException e)
      {
        return null;
      }
    }
    return this.btDevice;
  }
```

Figura 5.3: esempio di riesecuzione

Per quanto riguarda gli eventi si è pensato di fornire la possibilità di due modalità di notifica:

1. *singola notifica*: in questo modo è creato un nuovo processo per ogni entità che si è registrata per ricevere la notifica (primo ciclo for);

2. *gruppi di notifica*: in questa modalità tutte le entità che si sono registrate per la notifica dell'evento sono suddivise in gruppi e per ogni gruppo è creato un nuovo processo (secondo ciclo for).

Sono state realizzate queste modalità per evitare che un'entità interessata all'evento potesse determinare una sospensione del Manager. In questo modo, anche se l'entità si blocca durante la gestione dell'evento, si sospenderà solamente il processo che si occupa della notifica e non anche il Manager.

Di seguito riportiamo un esempio esplicativo per questa gestione:

```
public void OnClientAddressEvent(String address)
{
  OnAddressEvent ae;
  if(caListeners.size() <= this.threadsLimit)
  {
    //Creo un nuovo processo per la notifica per ogni listener
    for(int i = 0; i < caListeners.size(); i++)
    {
      ae = new OnAddressEvent(this, (ClientAddressListener)caListeners.elementAt(i),
address);
    }
  }
  else
  {
    //Suddivido i listener in gruppi e per ogni gruppo effettuo la notifica con un nuovo
    //processo
    Vector tempListeners = new Vector();
    for(int i = 0; i < this.caListeners.size(); i++)
    {
      tempListeners.add(caListeners.elementAt(i));
      if((((i+1) % this.bufferSize) == 0) || (i == (caListeners.size() - 1)))
      {
        ae = new OnAddressEvent(this, tempListeners, address);
        tempListeners.clear();
      }
    }
  }
}
```

Figura 5.4: esempio di singola notifica o di gruppi di notifica

Una volta istanziato il Manager, passandogli il nome logico del dispositivo WiFi e Bluetooth (es. eth1 e hci0), è possibile recuperare tutte le informazioni di interesse

utilizzando nel modo opportuno i metodi messi a disposizione. Come evidenziato nell'architettura, l'effettivo recupero delle informazioni è effettuato da due entità distinte chiamate BluetoothExecutor e WiFiExecutor delle quali parleremo in seguito. Di seguito sono illustrate le interazioni del Manager con le altre classi.

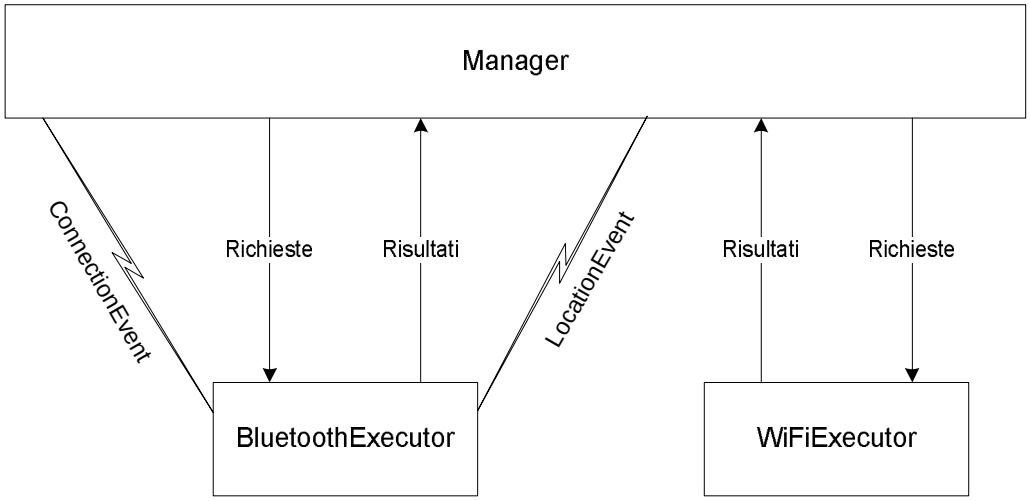


Figura 5.5: Interazione del Manager

Entrando più nel dettaglio, il Manager recupera informazioni relative al Bluetooth interrogando un istanza del BluetoothExecutor che restituisce l'RSSI simbolico o reale del dispositivo oppure la lista dei dispositivi connessi attraverso un istanza della classe BtInfo, oppure il mac e l'RSSI del dispositivo con il miglior segnale utilizzando oggetti di tipo BtDevice. In aggiunta, il BluetoothExecutor, offre la possibilità di recuperare ulteriori informazioni sulla locazione del dispositivo (grado e nome) e sulle connessioni effettuate dal dispositivo (numero connessioni, RSSI, probabilità di handoff…) interrogando le entità attive LocationManager e ConnectionManager. Il BluetoothExecutor recupera queste informazioni o interrogando queste classi a polling oppure registrando il Manager presso queste classi in moda tale da notificare il Manager quando si hanno cambiamenti sulla locazione o sulle connessioni fatte dal dispositivo.

Allo stesso modo è possibile recuperare informazioni sul WiFi interrogando un istanza della classe WiFiExecutor che fornirà informazioni quali l'RSSI, la qualità del collegamento ed il livello di rumore attraverso un istanza della classe WifiInfo.

Le entità WiFiExecutor e BluetoothExecuto sono entità passive che sono interrogate dal Manager.

5.4 BluetoothExecutor

L'obbiettivo di questa entità passiva è quello di fornire alcune informazioni al Manager quali: l'RSSI in forma normale, RSSI in forma simbolica (cioè segnale alto, basso o medio) e l'RSSI più alto tra i dispositivi connessi. Fornisce, inoltre, delle funzionalità come l'accensione e spegnimento del dispositivo, la connessione e disconnessione con un particolare dispositivo e la lista dei dispositivi ai quali si è connessi. Nel momento in cui è effettuata la richiesta del massimo segnale, è restituito un oggetto (BtDevice) rappresentante il dispositivo Bluetooth con il segnale migliore. Anche per quanto riguarda la richiesta di informazioni per il Bluetooth (BtInfo), è restituito un oggetto (BtInfo) che contiene tutti i dispositivi presenti con i relativi segnali. Di seguito è riportato in particolare l'interazione tra il Manager ed il BluetoothExecutor e tra questo ed il Sistema operativo Linux.

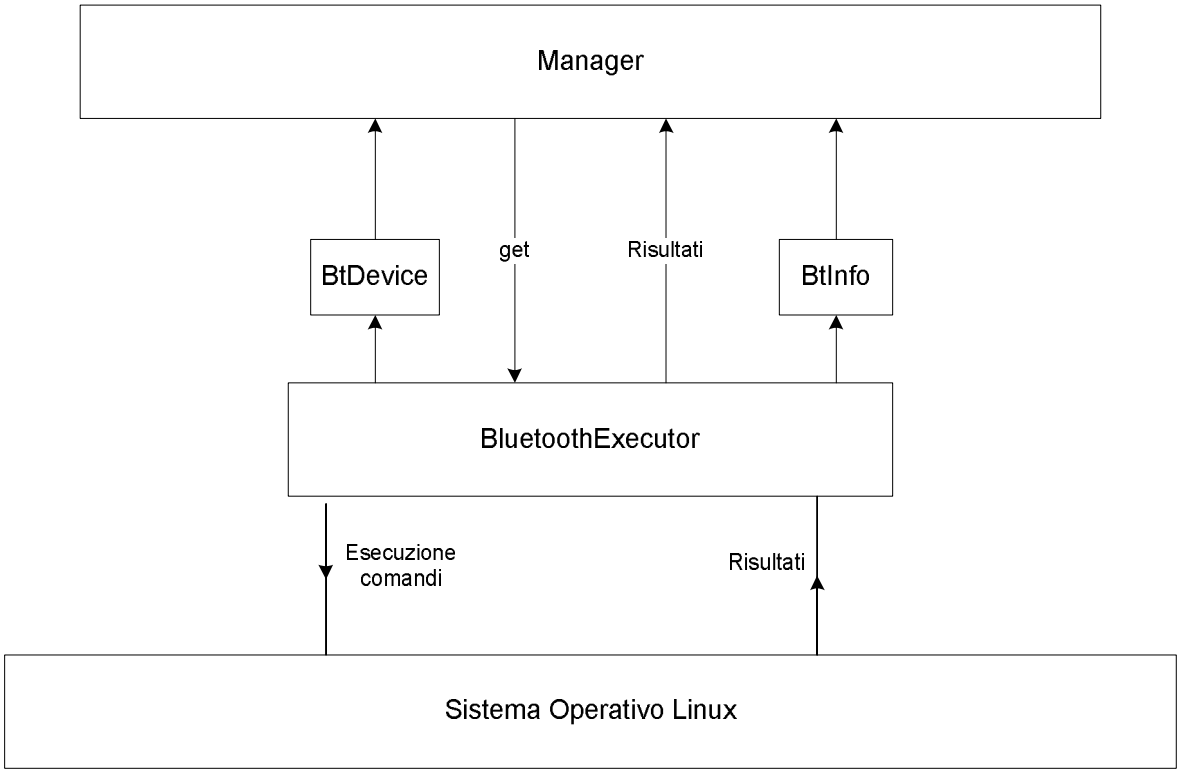


Figura 5.6: Interazioni del BluetoothExecutor

In particolare, per quanto riguarda l'accensione e spegnimento del dispositivo Bluetooth, se il comando non è eseguito, si effettuano un numero impostabile di tentativi per l'accensione/spegnimento del dispositivo. Se una volta eseguiti tutti i tentativi, il comando non è ancora stato acceso è restituito un messaggio di errore.

È previsto anche un metodo che, fornendo in ingresso l'RSSI, restituisce una rappresentazione simbolica del segnale (es. alto, medio o basso).
Per quanto riguarda l'esecuzione dei comandi, come descritto nel paragrafo 5.1.1, è utilizzata la classe Runtime per interfacciarsi con l'ambiente di seguito è riportato un esempio di esecuzione del comando per attivare o disattivare il dispositivo Bluetooth:

```
public void Switch(boolean on) throws IOException, InterruptedException
{
  String command, status;
  Process process;
  if(on)
    status = "up";
  else
    status = "down";
  //Attivazione o disattivazione del dispositivo bluetooth
  command = new String("hciconfig");
  //ciclo in cui si tenta di attivare/disattivare il dispositivo per un certo numero di
  //tentativi
  for(int i = 0; i < this.getNOT(); i++)
  {
    try
    {
      //esecuzione del comando
      if(path != null)
        process = Runtime.getRuntime().exec(path+"/"+command+" "+device+" "+status);
      else
        process = Runtime.getRuntime().exec(command+" "+device+" "+status);
      process.waitFor();
      Thread.sleep(20);

      //verifico se il comando è stato eseguito correttamente ed esco dal ciclo
      if(process.exitValue() >= 0)
      {
        System.out.println("Dispositivo BT "+status);
        i = this.getNOT();
      }
      else if(verbose)
      {
        System.out.println("Tentativo BT "+(i+1)+" su "+this.getNOT()+" fallito!
Ritento...");
      }
```

```
      //se ho tentato già per il numero di volte prefissato avviso
      if((i+1) == this.getNOT())
        System.out.println("Impossibile attivare o disattivare il dispositivo BT.
Controllare che sia presente.");
    }
    catch (InterruptedException e)
    {
      if(i == (this.getNOT() - 1))
      {throw e;}
      else
      {System.out.println("Tentativo BT "+(i+1)+" su "+this.getNOT()+" fallito!
Ritento...");}
    }
 }
}
```

Figura 5.7: esecuzione comando per attivazione/disattivazione dispositivo Bluetooth

Nel blocco di codice riportato in figura 5.7 è evidenziato il metodo per l'attivazione o disattivazione del dispositivo Bluetooth mediante l'esecuzione del comando "hciconfig" attraverso la classe Runtime. Questa restituisce il processo che ha eseguito il comando ed attraverso questo è possibile verificare se il comando è stato eseguito correttamente o meno. Con un approccio di questo tipo avremo una maggiore modularità in quando basta inserire un modulo simile per ogni sistema operativo con l'esecuzione del comando opportuno.

5.5 WiFiExecutor

Questa classe è un'entità passiva utilizzata dal Manager per attivare o meno il dispositivo e per recuperare le informazioni desiderate relative al WiFi come l'RSSI, la qualità del collegamento e il livello di rumore. Una volta recuperate le informazioni e prima di restituire il risultato al Manager, è effettuata la conversione dei risultati acquisiti dalla forma testuale a quella numerica. Una volta effettuata la conversione è restituito al Manager un oggetto di tipo WifiInfo contenente informazioni sull'RSSI, la qualità del collegamento e il livello del rumore. È previsto anche un metodo che, fornendo in ingresso l'RSSI, restituisce in uscita una rappresentazione simbolica del segnale (es. alto, medio o basso). Di seguito è riportato in particolare l'interazione tra il Manager ed il WiFiExecutor e tra questo ed il Sistema operativo Linux.

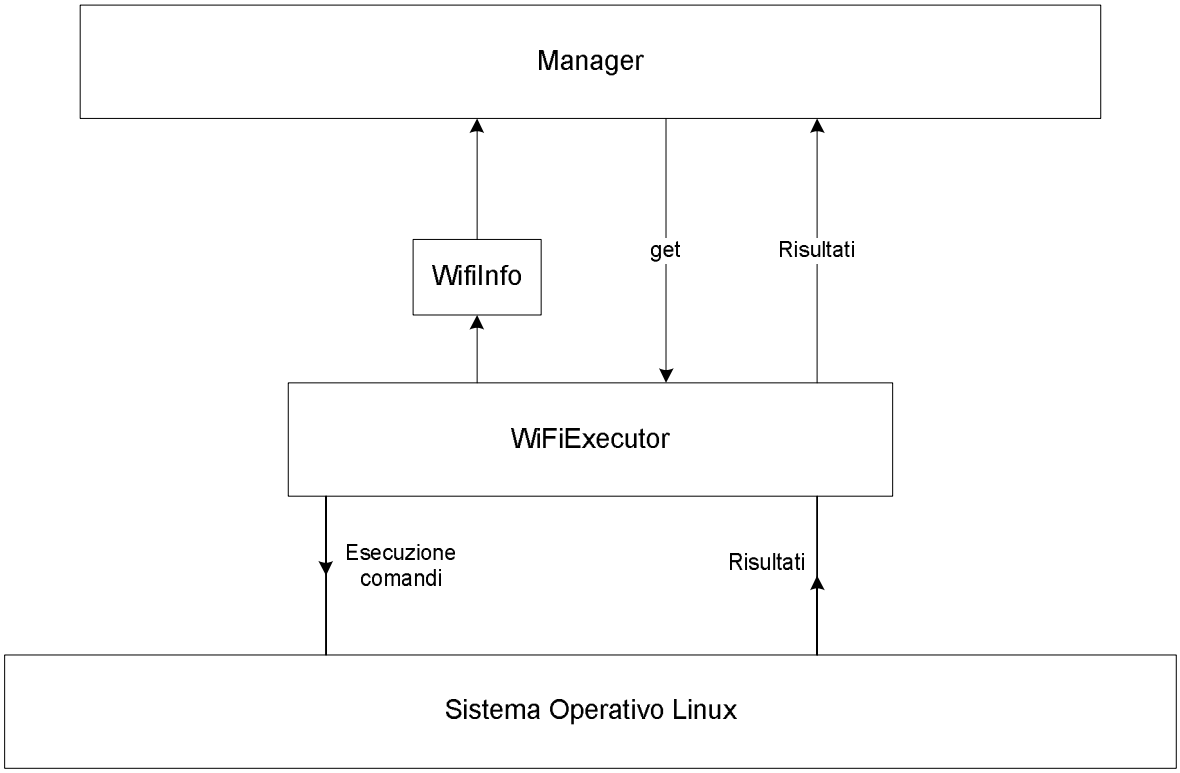


Figura 5.8: Interazione del WiFiExecutor

Come visto anche per il BluetoothExecutor, anche in questo caso è riportato un esempio di codice relativo all'esecuzione del comando per l'attivazione o disattivazione del dispositivo WiFi attraverso la classe Runtime:

```
public void Switch(boolean on) throws IOException, InterruptedException
{
  String command, status;
  Process process;

  if(on)
    status = "up";
  else
    status = "down";
  //Attivazione o disattivazione della scheda wi-fi
  command=new String("ifconfig");

  //ciclo in cui si tenta di attivare/disattivare il dispositivo per un certo numero di
  //tentativi
  for(int i = 0; i < this.getNOT(); i++)
  {
    try
    {
      //esecuzione del comando
      if(path != null)
        process = Runtime.getRuntime().exec(path+"/"+command+" "+device+" "+status);
      else
        process = Runtime.getRuntime().exec(command+" "+device+" "+status);
      process.waitFor();

      //verifico se il comando è stato eseguito correttamente ed esco dal ciclo
      if(process.exitValue() >= 0)
      {
        System.out.println("Dispositivo WiFi "+status);
        i = this.getNOT();
      }
      else if(verbose)
      {
          System.out.println("Tentativo  WiFi  "+(i+1)+"  su  "+this.getNOT()+"  fallito!
Ritento...");
      }

      //se ho tentato già per il numero di volte prefissato avviso
      if((i+1) == this.getNOT())
        System.out.println("Impossibile  attivare  o  disattivare  il  dispositivo  WiFi.
Controllare che sia presente.");
    }
    catch(InterruptedException e)
    {
      if(i == (this.getNOT() - 1))
```

```
      {throw new InterruptedException();}
      else
      {
        System.out.println("Tentativo  WiFi  "+(i+1)+"  su  "+this.getNOT()+"  fallito!
Ritento...");
      }
    }
  }
}
```

Figura 5.9: esecuzione comando per attivazione/disattivazione dispositivo WiFi

Nel blocco di codice riportato in figura 5.9 è evidenziato il metodo per l'attivazione o disattivazione del dispositivo WiFi mediante l'esecuzione del comando "ifconfig" attraverso la classe Runtime. Questa restituisce il processo che ha eseguito il comando ed attraverso questo è possibile verificare se il comando è stato eseguito correttamente o meno.

Come visto anche in precedenza un approccio di questo tipo porta ad una maggiore modularità in quando basta inserire un modulo simile per ogni sistema operativo con l'esecuzione del comando opportuno. In questo modo l'applicazione può essere estesa facilmente.

5.6 WifiInfo

La classe WifiInfo contiene tutte le informazioni principali per la scheda wireless. Si riporta di seguito il diagramma UML relativo:

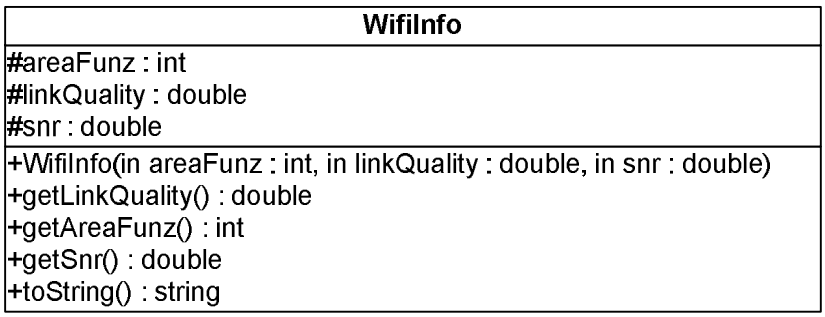


Figura 5.10: Classe WifiInfo

La classe è dotata di un costruttore che accetta come parametri tre valori rappresentanti l'RSSI, la qualità del collegamento e il livello di rumore e provvede a memorizzarli.

È presente un metodo quale `getLinkQuality()`utilizzato per recuperare il valore relativo alla qualità del collegamento.

`GetAreaFunz()`è un metodo utilizzato per recuperare il valore relativo all'RSSI simbolico mentre `getSnr()` è utilizzato per recuperare il valore relativo al livello di rumore.

Ultimo metodo è `toString()` utilizzato per visualizzare le informazioni memorizzate nell'oggetto (vale a dire l'RSSI simbolico, il livello di qualità e di rumore) nel modo desiderato.

Questa classe è utilizzata da altre entità quali il WiFiExecutor ed il Manager per memorizzare le informazioni raccolte per il WiFi.

5.7 BtInfo

La classe BtInfo è utilizzata per rappresentare un dispositivo Bluetooth. In particolare, in essa sono memorizzati il mac e l'RSSI simbolico (areaFunz) associati al dispositivo Bluetooth dell'AP.

Si riporta il diagramma UML relativo alla classe:

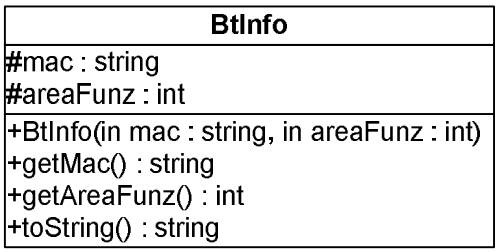


Figura 5.11 Classe B tInfo

È dotata di un unico costruttore che accetta come argomenti una stringa che rappresenta il mac ed un intero rappresentante l'RSSI simbolico (potenza del segnale diviso per aree di funzionamento) associati al dispositivo Bluetooth dell'AP.

Sono presenti vari metodi, quali:

- `getMac()`: utilizzato per recuperare il mac dell'AP;
- `getAreaFunz()`: utilizzato per recuperare la potenza del segnale associato al dispositivo in formato simbolico;
- `toString()`: restituisce una stringa contenente le informazioni memorizzate per il dispositivo ossia il mac e l'RSSI simbolico.

5.8 BtDevice

Questa classe contiene le informazioni relative ai mac dei dispositivi Bluetooth presenti. Queste informazioni possono essere sfruttate per verificare se si è connessi ad un dispositivo (verificando se è presente l'RSSI passando il mac) ed eventualmente per forzare la connessione con un particolare dispositivo. Questa entità non è stata introdotta anche per il WiFi in quanto questo può connettersi solamente ad un dispositivo contemporaneamente, cosa che invece non è vero per il Bluetooth. Si riporta di seguito il diagramma UML relativo alla classe:

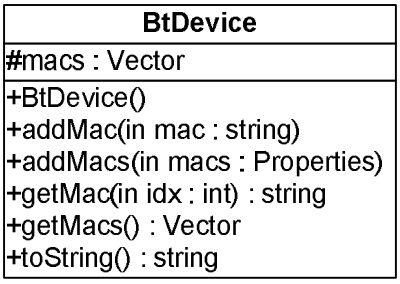


Figura 5.12: Classe BtDevice

La classe è dotata di un unico costruttore vuoto e questo perché è prima istanziata e poi successivamente riempita.

Sono presenti un insieme di metodi che offrono vari servizi.

C'è il metodo `addMac(String mac)` che è utilizzato per aggiungere il mac del dispositivo Bluetooth, passato da argomento, alla lista memorizzata.

Il metodo `addMacs(Properties prop)` è, invece, utilizzato per memorizzare la lista dei mac dei dispositivi presenti. Accetta come parametro un oggetto di tipo Properties, che contiene le coppie costituite da un progressivo e dal mac del dispositivo.

`GetMac(int idx)` è un metodo utilizzato per recuperare il mac, in formato stringa, del dispositivo dalla lista nel caso in cui si è a conoscenza della sua posizione nella lista. Accetta come parametro un intero che rappresenta la posizione del mac nella lista.

Il metodo `getMacs()` è utilizzato per recuperare l'intero vettore contenente i mac dei dispositivi Bluetooth.

Ultimo metodo presente è `toString()` utilizzato per ottenere una stringa rappresentante tutti i mac contenuti nella lista.

5.9 PolicyType

Questa classe è utilizzata per rappresentare le diverse politiche associate all'accensione e spegnimento di un dispositivo. Di seguito si riporta il diagramma UML associato:

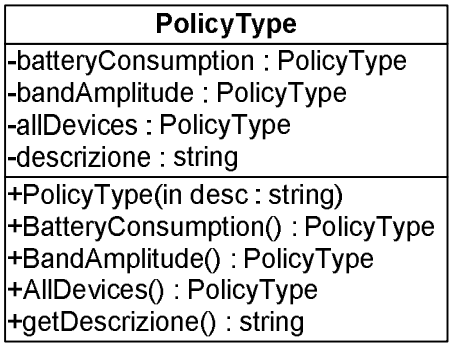


Figura 5.13: Classe PolicyType

È presente un unico costruttore che accetta come parametro una stringa rappresentante la descrizione per la particolare politica.

Il metodo `BatteryConsumption()` è utilizzato per creare una nuova istanza di PolicyType, che individua una politica associata all'accensione del dispositivo Bluetooth.

`BandAmplitude()` è il metodo utilizzato per creare una nuova istanza di PolicyType che individua una politica associata all'accensione del dispositivo WiFi, mentre il metodo `AllDevices()` è utilizzato per creare una nuova istanza di PolicyType che individua una politica associata all'accensione di entrambi i dispositivi.

Ultimo metodo presente è `getDescrizione()` che restituisce una stringa contenente la descrizione associata alla particolare politica.

5.10 CLM

Questa classe è utilizzata per attivare o disattivare il demone conmand sul client. Implementa l'interfaccia CLMManager e di seguito ne riportiamo il diagramma UML associato:

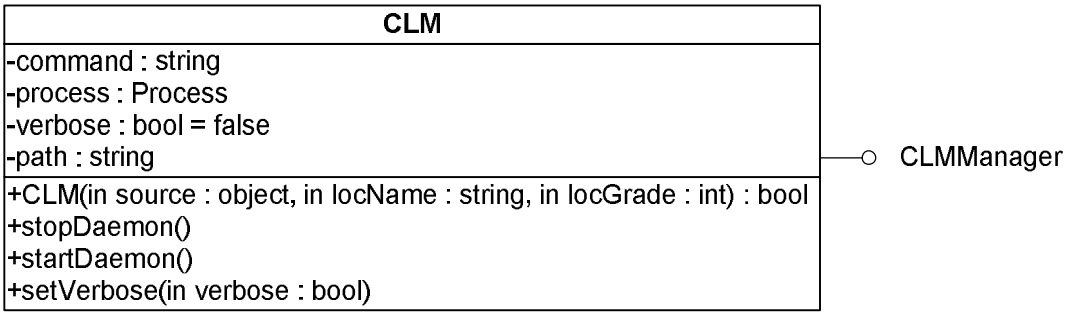


Figura 5.14: Classe CLM

La classe è dotata di un costruttore che accetta come argomenti un booleano rappresentante l'attivazione o meno della modalità verbosa. Sono presenti due metodi:

- `stopDaemon()`: questo metodo è utilizzato per disattivare il demone "conmand" e per liberare la memoria condivisa;
- `startDaemon()`: metodo utilizzato per attivare il demone "conmand";
- `setVerbose(boolean verbose)`: questo metodo è utilizzato per attivare o meno la modalità verbosa del demone "conmand".

Capitolo 6

Testing

Come ultimo passo del lavoro sono stati compiuti dei test per valutare l'handoff da una tecnologia WiFi ad Bluetooth e viceversa in modo tale da ottenere delle informazioni relative ai tempi che si impiegano ad effettuare queste operazioni. Gli strumenti utilizzati per lo scopo sono stati:

- sul Client: Dell laptop Centrino, 1.5GHz, 512MB RAM, Linux Fedora Core 2, connesso mediante una scheda IEEE 802.11b Cisco e doungle Mopogo BT, classe 1 versione 1.1;
- sul Server: Pc Dell, 3GHz, 512MB RAM, Linux Gentoo dotato di un AP Wi-Fi Cisco Aironet 1100 e di un doungle Mopogo BT classe 1 versione 1.1.

Per raggiungere gli obiettivi preposti sono state realizzate quattro classi tra cui il TestLogger che permette di stampare i risultati su file ed il Tester che realizza l'handoff verticale. Di è riportato la struttura per il test.

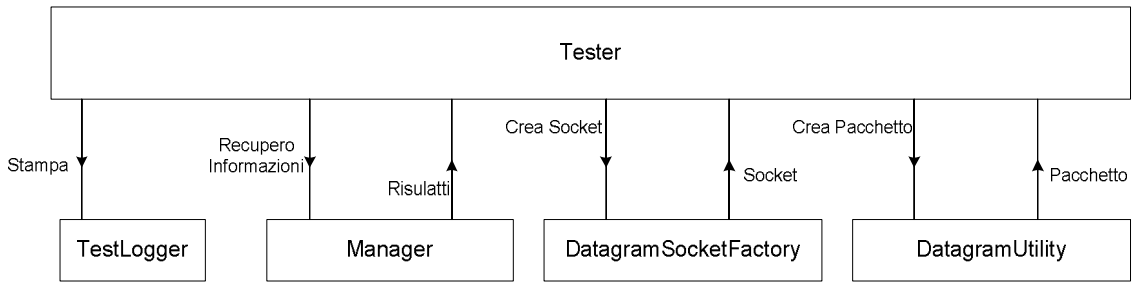


Figura 6.1: Struttura del tester

6.1 Tester

Come accennato questa entità è stata creata per valutare i tempi impiegati ad effettuare l'handoff verticale da una tecnologia WiFi ad una Bluetooth e viceversa. Inizialmente è stata creata una nuova istanza del Manager cui si passa il nome logico del dispositivo WiFi e Bluetooth. Il Tester una volta creato si registra presso il

Manager per ricevere notifiche sugli eventi; inizialmente si accende la tecnologia WiFi, è poi recuperato l'indirizzo associato alla scheda WiFi e sulla base di questo è creata una socket. Infine si procede con l'invio di pacchetti finché non è ricevuto l'evento relativo al cambiamento di IP. Quando si ricevono notifiche sull'evento relativo al cambiamento di indirizzo IP è verificata quale tecnologia è attiva. Se sono attive entrambe le tecnologie, si recuperano informazioni relative alle aree di funzionamento di entrambi i dispositivi e confrontandoli si verifica verso quale tecnologia conviene effettuare l'handoff. Se invece è attiva solo una delle due tecnologie allora si provvede ad attivare anche l'altra tecnologia, si recuperano informazioni relative ad entrambi i dispositivi per confrontarne le aree di funzionamento e decidere se è necessario effettuare l'handoff e, una volta confrontati, si decide se effettuare l'handoff verso l'altra tecnologia o meno.

In entrambi i casi, se necessario effettuare l'handoff, si spegne il primo dispositivo e si chiude la socket inizialmente utilizzata dopodichè si crea un'altra socket utilizzando il nuovo indirizzo IP associato al dispositivo verso il quale si è effettuato l'handoff e si invia un pacchetto al Server attendendo una sua risposta.

Alla fine è effettuato il calcolo del tempo che si impiega ad effettuare queste operazioni per avere una stima del tempo medio impiegato ad effettuare l'handoff da una tecnologia verso un'altra.

È stato osservato che se in precedenza sono state già compiute operazioni con il Bluetooth, il tempo per passare ad una tecnologia Bluetooth decresce sensibilmente in quanto i risultati dell'operazione di "Inquiry" sono memorizzati e restituiti dal Sistema Operativo.

Riportiamo di seguito alcuni grafici rappresentanti i risultati ottenuti durante i test:

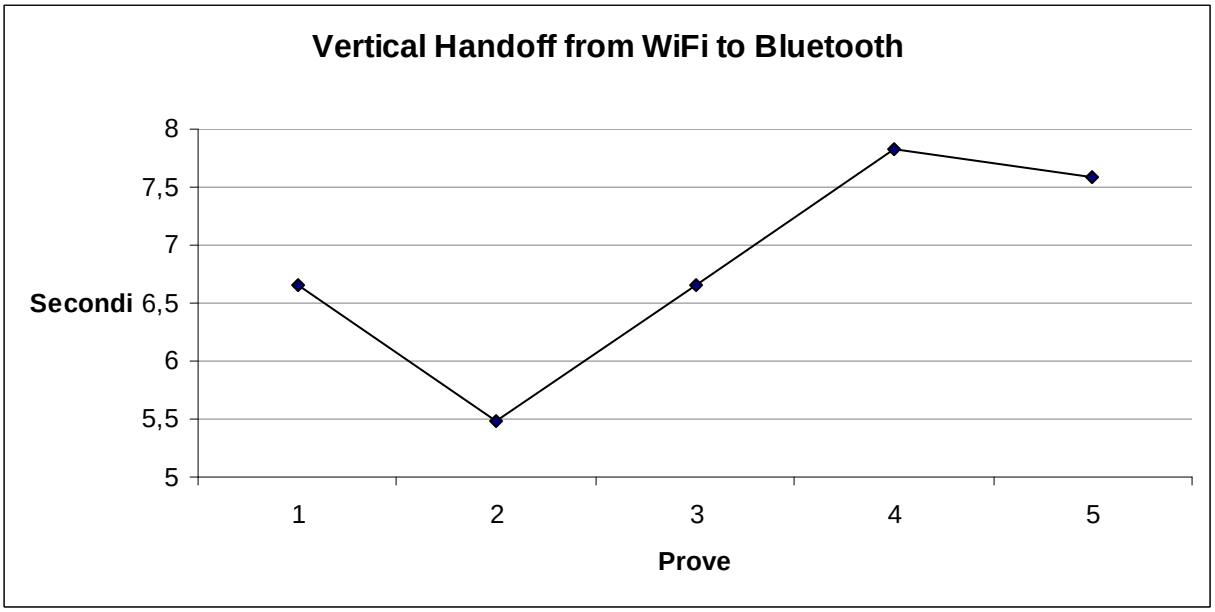


Figura 6.2: Tempi da handoff dal WiFi al Bluetooth

Com'è possibile vedere anche dal grafico, nel caso di handoff da una tecnologia WiFi ad una Bluetooth, il tempo impiegato in media è circa di 6,700 secondi. Di seguito è riportato più in dettaglio le medie per ciascun esperimento:

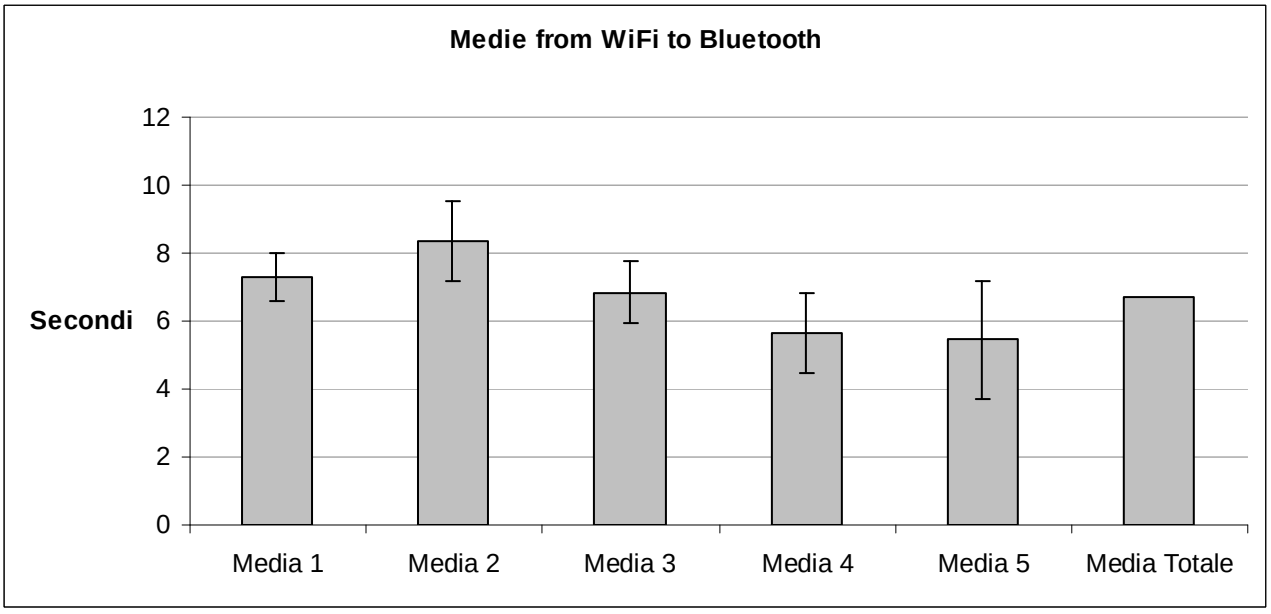
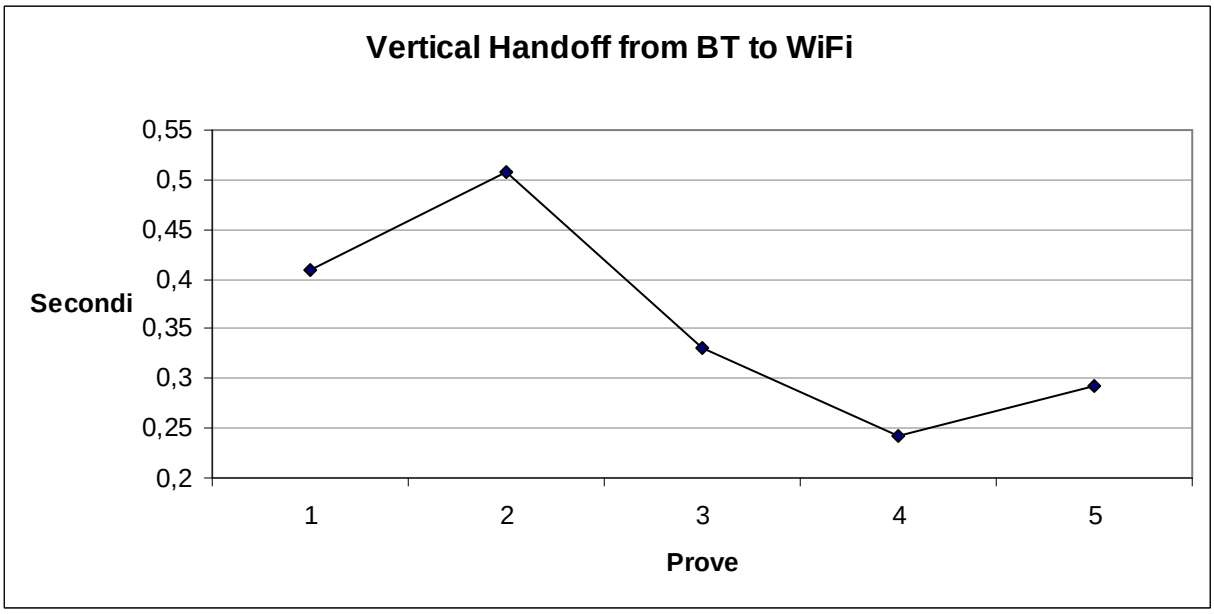


Figura 6.3: Media dei tempi di handoff da una tecnologia WiFi ad una Bluetooth

In questo grafico sono riportate le medie delle varie prove, la deviazione standard ed infine è riportata la media totale.

È stato anche osservato che se non è la prima volta che si utilizza il Bluetooth, e quindi i risultati dell'operazione di Inquiry sono stati memorizzati dal Sistema Operativo, la media dei tempi di handoff si assestano nell'intorno di 1,3 secondi. Questo però non è molto indicativo ed è stato riportato solo come osservazione in quanto i risultati delle operazioni con il Bluetooth sono memorizzati dal Sistema Operativo solo per qualche decina di secondi.
Vediamo ora il grafico che riporta i tempi riscontrati per effettuare l'handoff da una tecnologia Bluetooth ad una WiFi.



Grafo 6.4: Tempi di handoff dal Bluetooth al WiFi

Com'è possibile vedere anche dai risultati riportati nel grafico, il passaggio da una tecnologia Bluetooth ad una WiFi risulta essere molto più veloce rispetto al caso di passaggio da una tecnologia WiFi ad una Bluetooth. La media del tempo impiegato ad eseguire l'handoff si assesta intorno a 0,3620 secondi.
Di seguito è riportato il grafico che rappresenta le medie delle varie prove con la deviazione standard riportando anche la media totale:

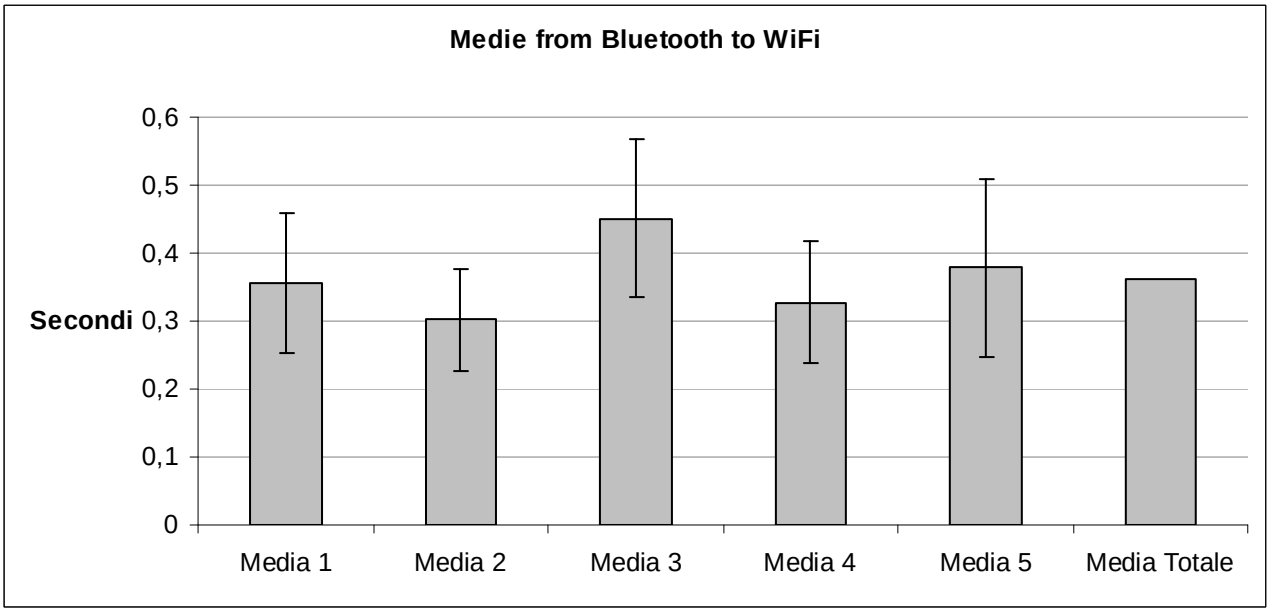


Figura 6.5: Media delle prove effettuate per handoff da una tecnologia Bluetooth ad una WiFi

Questa differenza nei tempi di handoff è imputabile al fatto che quando si passa dal WiFi al Bluetooth occorre recuperare le informazioni sul dispositivo Bluetooth (dispositivi visibili, RSSI…) e ciò richiede l'esecuzione di comandi che necessitano di un certo tempo per fornire dei risultati. Al contrario, il passaggio dal Bluetooth al WiFi è molto più veloce in quanto il recupero delle informazioni dal WiFi richiede dei comandi che impiegano meno tempo a fornire un risultato.

Infine, come ultima prova, è stato preso in considerazione l'occupazione percentuale della CPU. Inizialmente è stata considerata l'occupazione della CPU in condizioni di lavoro normali (CPU scarica), poi si è avviato solamente il Manager (Manager) ed infine è stata avviata l'applicazione per recuperare i tempi di handoff (Applicazione).

Queste valutazioni sono state fatte sia considerando il Manager come un'entità attiva e quindi un processo sempre in esecuzione, e sia considerando il Manager come un'entità passiva interrogata a polling.

Di seguito è riportato un grafico con i risultati riscontrati per il Manager in modalità passiva:

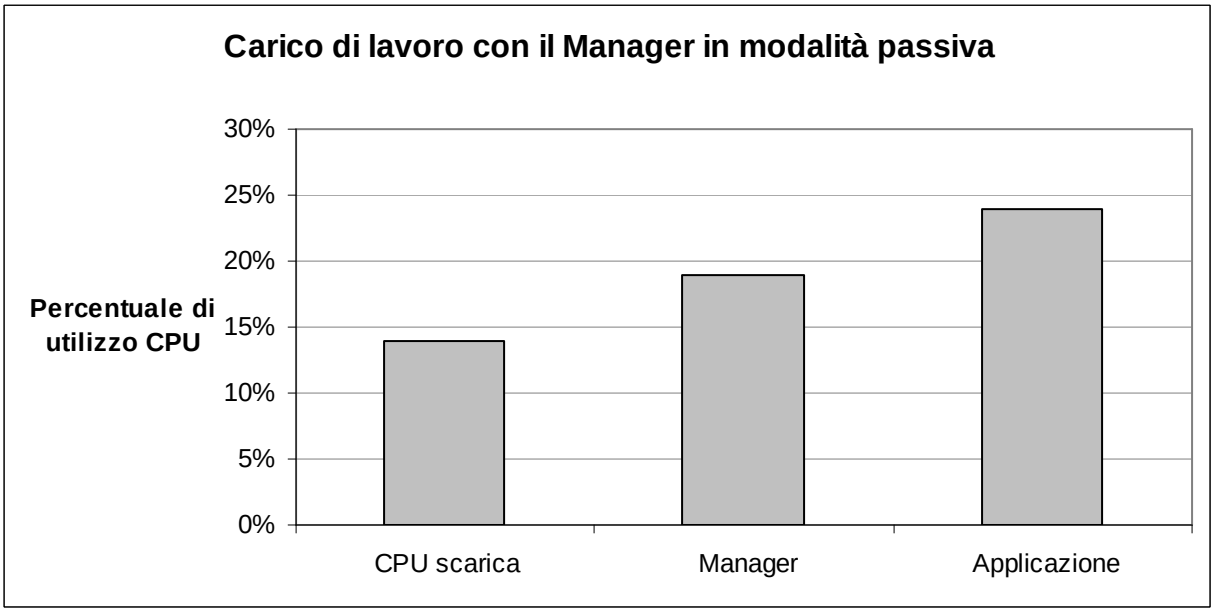


Figura 6.6: Utilizzo percentuale della CPU con Manager in modalità passiva

Inizialmente l'occupazione della CPU è del 14%, avviando il Manager in modalità passiva l'utilizzo della CPU sale all'19% ed infine è stata avviata l'intera applicazione e si è verificato un incremento dell'utilizzo della CPU di circa il 5%.
Riportiamo ora il grafico dell'occupazione di CPU nel caso di avvio del Manager in modalità attiva:

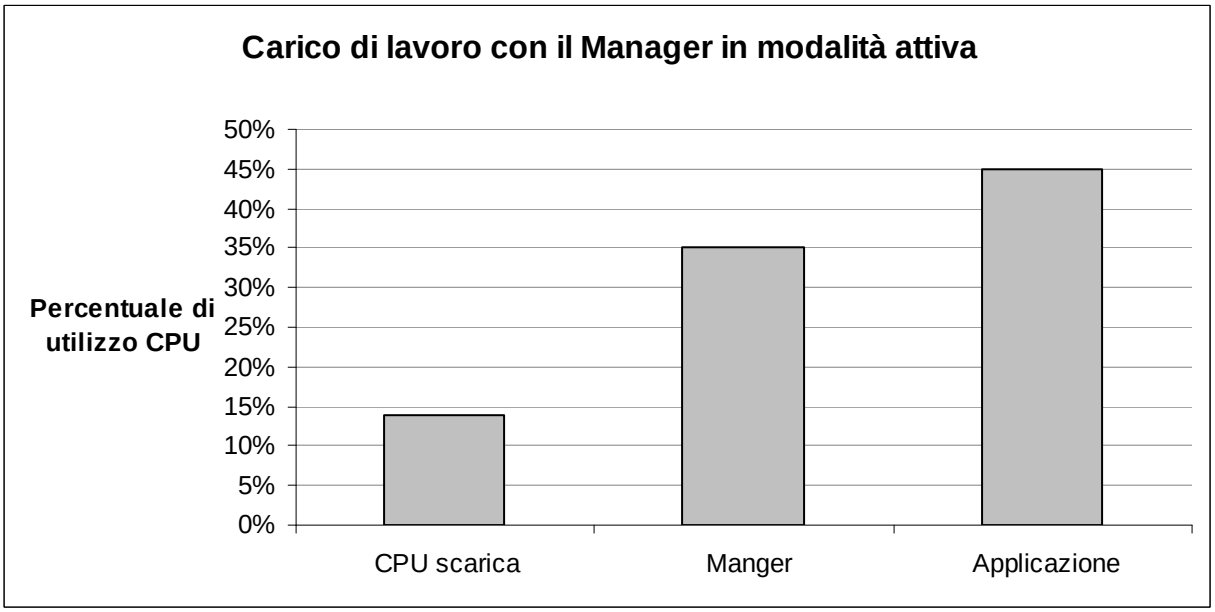


Figura 6.7: Utilizzo percentuale della CPU con Manager in modalità attiva

Come è possibile vedere, utilizzando il Manager in modalità attiva e quindi come un processo separato si avrà un incremento dell'utilizzo di CPU rispetto al caso precedente. Inizialmente l'occupazione di CPU è sempre pari al 14% poi, in seguito all'avvio del Manager in modalità attiva, si ha un'occupazione di CPU del 35% ed avviando l'intera applicazione l'occupazione sale al 45%.

Conclusioni

Nel corso di questa tesi è stata di sviluppata un'infrastruttura in grado di recuperare e fornire in modo omogeneo informazioni sulle condizioni delle tecnologie wireless disponibili su un certo host, ad esempio WiFi e Bluetooth, in modo da semplificare lo sviluppo di servizi applicativi capaci di gestire in modo consapevole l'handoff verticale.

La tesi ha portato allo sviluppo di un Manager che, facendo uso di un sistema per il recupero delle informazioni, offre la possibilità di notifica e fornisce informazioni agli interessati, in modo tale da poter decidere se effettuare l'handoff verticale e verso quale tecnologia effettuarlo.

Durante questo lavoro sono stati incontrati e risolti diversi problemi. Innanzi tutto sono state previste due modalità di funzionamento per il Manager e cioè una modalità "*attiva*" e una "*passiva*". La prima prevede l'attivazione del Manager come un processo separato che raccoglie le informazioni di interesse e le comunica alle entità interessate, la seconda, invece, prevede l'utilizzo del Manager come una normale classe java che dovrà essere interrogata a polling per ottenere informazioni. Queste due modalità sono state previste per non appesantire il carico di lavoro della macchina sulla quale si andrà ad operare, se questa ha risorse limitate.

Altro problema che si è cercato di risolvere riguarda la notifica degli eventi. Per evitare un blocco sincrono del Manager durante la notifica degli eventi è stato implementato un meccanismo in base al quale le entità da notificare sono divise in gruppi e per ogni gruppo è creato un processo che si occupa della notifica. In questo modo se si verifica un blocco questo riguarda solamente il gruppo associato al processo per la notifica e non anche il Manager.

Infine per cercare di limitare il carico computazionale dovuto al monitoraggio delle interfacce di rete, si è pensato di introdurre un insieme di parametri per impostare delle soglie in base alle quali sarà deciso se eseguire un particolare controllo, oppure restituire il risultato precedente dello stesso. È stato quindi realizzato un meccanismo mediante il quale è memorizzato il nome, l'istante di ultima esecuzione e il tempo impiegato per l'esecuzione del metodo. Quando un metodo è

eseguito, si controlla se è stato già eseguito, ed in caso affermativo, si determina il tempo passato dall'ultima esecuzione, calcolandone il rapporto con la durata media di ogni operazione. Se il rapporto supera una certa soglia, è eseguito di nuovo il metodo, altrimenti è restituito il precedente risultato.
Infine per completare il lavoro sono stati eseguiti alcuni esperimenti sul tempo che si impiega per passare da una tecnologia WiFi ad una Bluetooth e viceversa.

Ci sono diverse linee possibili di sviluppi futuri; al momento tutto il lavoro svolto si concentra sul recupero delle informazioni utilizzando comandi tipici del Sistema Operativo Linux: questo in futuro potrà essere esteso per utilizzarlo anche sotto altri Sistemi Operativi come ad esempio Windows. Inoltre, la parte concernente la divisione del segnale ricevuto dal WiFi (RSSI) in tre zone logiche, attualmente è stato cablato per l'utilizzo di una scheda wireless "cisco", ma in futuro potrebbe essere esteso al caso di una qualunque scheda wireless.

Bibliografia

[AAO/05] Wen-Tsuen Chen e Yen-Yuan Shu, “Active Application Oriented Vertical Handoff in Next-Generation Wireless Networks”, IEEE 0-7803-8966-2, 2005

[AS/04] Wen-Tsuen Chen, Jen-Chu Liu e Hsieh-Kuan Huang, “An Adaptive Scheme for Vertical Handoff in Wireless Overlay Networks“, IEEE 1521-9097, 2004

[OS/05] Abolfazl Mehbodniya e Jalil Chitizadeh, “An Intelligent Vertical Handoff Algorithm for Next Generation Wireless Networks”, IEEE 0-7803-9019-9, 2005

[OVHDA/04] Fang Zhu e Janise McNair, “Optimizations for Vertical Handoff Decision Algorithms“, IEEE 0-7803-8344-3, 2004

[EtE/04] Hung-Yun Hsieh, Kyu-Han Kim e Raghupathy Sivakumar, “An End-to-End Approach for Transparent Mobility across Heterogeneous Wireless Networks”, Kluwer Academic Publishers 363–378, 2004

[BTE] Bluetooth evoluzione:
http://users.unimi.it/metis/tesidiego/tesionline/bluetooth.htm

[BTF] Bluetooth funzionamento: http://bluetooth.interfree.it/bluetooth1.htm

[J2se] Java Documentation ondine:
http://java.sun.com/j2se/1.5.0/docs/api/index.html

[CSAW] Carlo Giannelli, “CONTINUITÀ DI SERVIZIO IN AMBIENTE WIRELESS”, 2003

[PB] Pierangeli Diego, “POLITICHE DI BUFFERIZZAZIONE IN INFRASTRUTTURE PER L’EROGAZIONE DI SERVIZI MULTIMEDIALI A DISPOSITIVI WIRELESS”, 2004

[AV] Paolo Angioletti, “Analisi e valutazione di un middleware per dispositivi mobili”, 2003

Vorrei ringraziare tutti coloro che mi hanno aiutato durante questo periodo ed in particolar modo i miei compagni di casa che hanno dimostrato grande pazienza nel sopportarmi.

Dedico questo lavoro di tesi alla mia famiglia che mi è stata sempre vicino e mi ha sostenuto ed incoraggiato nei momenti difficili.

Con Amore,

Pino.