UNIVERSITÀ DEGLI STUDI DI URBINO "CARLO BO"

Facoltà di Scienze e Tecnologie
Corso di Laurea in Informatica Applicata

Tesi di Laurea

ACCESSO ALLA RETE INTERNET ATTRAVERSO IL PROTOCOLLO BLUETOOTH

Relatore:
Chiar.mo Prof. Alessandro Bogliolo

Candidato:
Federico Torsello

Anno Accademico 2011-2012

Alla mia famiglia e in particolare a mio nonno Beniamino,
forse uno dei primi hacker del Salento

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Contesto

Il recente sviluppo della tecnologia mobile ha portato ad un'**espansione della connettività personale** senza precedenti, rendendo possibile l'accesso alla rete Internet e ai servizi Web (quasi) ovunque.

Qualche anno fa ciò era impensabile in quanto le infrastrutture telematiche, l'hardware e il software dei dispositivi mobili non erano in grado di garantire un *QoS* (qualità del servizio) all'altezza delle aspettative degli utenti.

Col passare del tempo la crescente richiesta di multimedialità e di interconnessione sociale ha spinto le aziende impegnate nel settore informatico a investire su prodotti sempre più complessi e portatili, fino ad arrivare alla concezione di una nuova generazione di telefoni cellulari, chiamati **smartphone**.

Uno ***smartphone*** è un dispositivo mobile che integra le funzionalità classiche di un telefono cellulare (effettuare/ricevere chiamate, SMS, ecc...) con altre funzionalità quali la gestione dei dati personali, la multimedialità e la connettività.

Per ampliare lo spettro delle funzionalità dei cellulari in questo senso si è resa necessaria una vera e propria rivisitazione del concetto di cellulare, portando a nuove soluzioni come lo sviluppo di **sistemi operativi dedicati e di *GUI*** (interfacce grafiche per l'utente) **molto più *user-friendly*** (con una grafica facile ed intuitiva).

Ciò ha reso gli smartphone dei **mobile devices** con caratteristiche tecniche superiori ai cellulari delle precedenti generazioni, aventi prestazioni comparabili a quelle dei PC fissi e possibilità di accesso ai servizi Web abituali per l'utente, **ottenendo così una nuova dimensione di networking detto *mobile networking***.

La semplicità d'accesso ai servizi Web con gli smartphone si è concretizzata nell'avvicinarsi di sempre più utenti al *cyberspazio*, diventando un'abitudine quotidiana di un numero crescente di persone. È ormai chiaro che l'uso di *chat* (messaggistica istantanea), videochiamate ed infine dei *social network*,

ha creato un nuovo modo di relazionarsi con gli altri, parallelo all'utilizzo di servizi noti (SMS, MMS, chiamate voce).

Questa evoluzione ha portato alla nascita di un nuovo *trend economico* dovuto alla scelta autonoma di molti utenti di **attivare un servizio a pagamento sulla propria *SIM card* (*Subscriber Identity Module*)** per poter godere dell'accesso ad Internet anche in mobilità.

Alla luce di quanto detto, non è difficile comprendere che **la connettività è una caratteristica fondamentale per questi apparecchi**. Dando una definizione esaustiva, per ***connettività di un dispositivo*** s'intende la capacità che ha quest'ultimo di connettersi con dei sistemi esterni al fine di trasferire dati. Per far questo sono necessari dei componenti hardware, controllati da software, per la messa in connessione del dispositivo.

Le connettività offerte da uno smartphone di grado medio-alto sono: dati a pacchetto (per esempio su reti **3G**), USB, **WLAN** (connessione **Wi-Fi**), **Bluetooth**.

È da sottolineare che tutti questi modi di trasferire dati sono disponibili solo per smartphone recenti. In particolare, **la possibilità di utilizzare una scheda wireless Wi-Fi integrata su dispositivi mobili è recentissima**.

Prima dell'introduzione della tecnologia Wi-Fi nella telefonia, l'accesso ad Internet con smartphone si basava esclusivamente su trasmissione su rete 3G, la quale originava un livello di traffico di dati mobile molto elevato. Naturalmente questo tendeva ad influenzare negativamente il sistema di connessione che spesso risultava congestionato o inefficiente.

La soluzione adottata per risolvere questo problema è stato **ricorrere al data offloading**.

Il ***data offloading* (scaricamento dati in modo alternativo)** permette di trasferire parte del carico dal traffico dati della rete cellulare ad altra rete (per esempio su WLAN), riducendo gli effetti negativi dell'accumulo di traffico dati a pacchetto, **limitando o eliminando l'effetto collo di bottiglia**.

Il modo più semplice per eseguire un data offload di uno smartphone su reti Wi-Fi è quello di utilizzare la scheda Wi-Fi integrata nel dispositivo, sfruttando così il collegamento wireless offerto da un router Wi-Fi al posto della classica connessione dati-cellulare.

In questo modo, la rete mobile e la rete wireless sono completamente separate e sarà un'applicazione installata sullo smartphone ad farsi carico dello *switching* da una rete all'altra.

Connessioni Wi-Fi di questo tipo possono essere offerte da *Hot Spot* (dispositivi connessi ad Internet che svolgono la funzione di *gateway*) che si differenziano in **pubblici (*Pubblic Hot Spot*)** o **privati (*Personal Hot Spot*)**.

Riassumendo quanto detto, **i vantaggi del data offloading riguardano sia gli operatori mobili** perché possono così decongestionare le reti cellulari, **sia gli utenti** i quali ottengono una maggiore larghezza di banda e l'accesso alla rete Internet in modo del tutto gratuito.

Per coloro che possiedono uno smartphone delle generazioni precedenti (che non integra connettività Wi-Fi, ma soltanto schede Bluetooth), non rimane altra soluzione che ricorrere a connessioni su rete 3G, quindi pagare un servizio di abbonamento per poter accedere ad Internet subendo delle limitazioni di tempo o di *bundle di download dati* (quantitativo massimo di dati scaricabili) di pochi gigabyte.

Nonostante siano apparentemente molto simili, lo **standard Bluetooth** e lo **standard Wi-Fi** presentano differenze molto marcate.

Il ***Wi-Fi*** è uno standard nato per fornire elevate velocità di trasmissione ad ampio raggio a costo di una maggiore potenza dissipata e di un hardware poco economico. Invece il ***Bluetooth* (*BT*)** è una tecnologia progettata per realizzare la comunicazione senza fili tra apparecchi di piccole dimensioni e non troppo distanti, in modo decisamente economico.

L'unica cosa che accomuna questi standard è che entrambi impiegano una trasmissione wireless per lo scambio di informazioni attraverso delle onde radio.

Dalla volontà di limitare questo *gap* tra vecchie e nuove generazioni di smartphone, nasce l'idea di **estendere la connettività di un cellulare smartphone che abbia almeno una connettività Bluetooth**, emulando, per quanto possibile, la connessione ad una WLAN.

In particolare, **il progetto sviluppato in questa tesi crea un accesso ad Internet attraverso *un canale alternativo non standard* e naturalmente *gratuito***, che impiega solo servizi di scambio dati con Bluetooth, prendendo spunto dalla tipica **connessione Wi-Fi tra smartphone e router**.

Una parte del sistema che si definirà in seguito si avvale di una caratteristica peculiare degli smartphone, cioè la possibilità di installare sul sistema operativo ulteriori applicazioni oltre a quelle nativamente inserite dal produttore. Questi programmi, dette *App*, aggiungono nuove funzionalità a quelle presenti di *default* sul dispositivo.

Il sistema descritto si compone di due software che lavorano in modo simbiotico: **MercuryBTClient** e **MercuryBTServer**.

- ***MercuryBTClient***: è l'App (*MIDlet Java*) installata sullo smartphone che svolge la funzione di client Bluetooth inviando le richieste delle pagine Web da visitare;

- ***MercuryBTServer***: è un'applet installato su un PC che fa da server Bluetooth. Il suo compito è riceve ed evade le richieste di indirizzamento Internet inviate dal client.

Il trasferimento dati tra questi programmi avviene attraverso una connessione Bluetooth *punto-punto* tra lo smartphone e il PC connesso ad Internet.

1.2 Organizzazione

Prima di analizzare nel dettaglio il funzionamento del progetto, nel Capitolo 2 si definiranno alcuni aspetti della tecnologia Wi-Fi richiamata più volte nell'introduzione.

Il Capitolo 3 tratterà le caratteristiche della tecnologia Bluetooth necessarie a comprendere lo sviluppo del progetto.

Il Capitolo 4 introdurrà il linguaggio di programmazione Java e più specificatamente del *J2ME*.

Il Capitolo 5 si descriverà il sistema client/server e le applicazioni Java MercuryBTClient e MercuryBTServer.

Il Capitolo 6 è il capitolo dedicato alla conclusioni dove verranno trattati i problemi aperti e sviluppi gli futuri del sistema illustrato nella tesi.

Capitolo 2

Connessioni wireless ad Internet

2.1 Il modello di riferimento ISO OSI

In telecomunicazioni e informatica **il modello *ISO OSI* è il modello di riferimento per i software di rete multilivello aperti** (disposti verso la comunicazione con altri sistemi). É uno **standard ISO (*International Standards Organization*)** concepito per *reti di telecomunicazioni a commutazione di pacchetto* anche molto diverse tra di loro.

Il modello ISO OSI è costituito da sette strati o livelli (*layer*) a cui è assegnata una determinata funzionalità. Bisogna però notare che ISO OSI non è un'architettura di rete, in quanto non specifica esattamente i servizi e i protocolli da usare in ogni strato, ma si limita a definire ciò che ogni strato deve compiere.

Stack ISO OSI:

7 Livello di applicazione

6 Livello di presentazione

5 Livello di sessione

4 Livello di trasporto

3 Livello di rete

2 Livello di data link o collegamento

- Sottolivello LLC
- Sottolivello MAC

1 Livello fisico

Il **livello fisico**, il **livello data link** e il **livello di rete** si occupano della gestione della sottorete di comunicazione e dipendono dal gestore della rete di comunicazione.

Il **livello di trasporto**, il **livello di sessione**, il **livello di presentazione** e il **livello di applicazione** riguardano l'elaborazione dell'informazione e la creazione di applicazioni indipendenti dalla rete.

2.2 Il modello di riferimento TCP/IP

Il ***TCP/IP* invece è un'architettura di rete** utilizzata per l'*internetworking*, cioè il collegamento di reti di natura diversa in modo affidabile.

Tabella 2.1: Stack TCP/IP

Numero di livello	Nome del livello
3	Livello di applicazione
2	Livello di trasporto
1	Livello rete o Internet
-	Livello host-to-network

Come si vede nella tabella Tab. 2.1, TCP/IP si compone di tre livelli che sono il **livello di rete o Internet**, il **livello di trasporto** e il **livello di applicazione**.

- Il **livello di rete** è lo strato dove le informazioni vengono spezzate in *pacchetti IP* (Internet Protocol). Ogni pacchetto IP può raggiungere la destinazione in modo indipendente, anche in ordine diverso da quello di invio.

- Il **livello di trasporto** offre due tipi di servizi, il protocollo **TCP (*Transmission Control Protocol*)** che è orientato alla connessione e il protocollo **UDP (*User Datagram Protocol*)** che è privo di connessione. L'applicazione, in base alle proprie esigenze, decide quale tra questi servizi usare.

- Il **livello di applicazione** contiene tutti i protocolli ad alto livello come *HTTP*, *SMTP*, *FTP*, *DNS*, ecc.

Al di sotto del livello di rete non sono definiti altri livelli ma viene generalmente indicato un **livello host-to-network** che consente di utilizzare uno qualsiasi degli standard disponibili per il trasporto dei pacchetti IP; quindi qualsiasi tecnologia esistente come gli standard *LAN*, *MAN* e *WAN* può essere sfruttata da TCP/IP per la comunicazione dati.

2.3 Confronto tra ISO OSI e TCP/IP

I modelli ISO OSI e TCP/IP sono gli standard di riferimento più seguiti nel campo delle reti di calcolatori.

Il **modello ISO OSI è uno *standard de jure***, cioè standard formale e legale, adottato dalla ISO che costituisce un punto di riferimento per le architetture di rete a commutazione di pacchetto.

Il **modello TCP/IP è uno *standard de facto***, cioè diventato standard grazie al suo reale utilizzo e alla sua diffusione.

La differenza principale tra loro consiste nel fatto che il modello ISO OSI è definito come modello formale che non descrive i protocolli, mentre TCP/IP descrive protocolli già esistenti e realmente utilizzati.

2.4 Reti Wireless

Per *trasmissione wireless* si intende la trasmissione di dati (come ad esempio pacchetti IP) senza l'utilizzo di cavi. Il trasferimento di tali informazioni avviene sfruttando le onde elettromagnetiche inviate dall'antenna del trasmettitore e captate dal ricevitore il quale è posto ad una certa distanza dal trasmettitore.

Lo standard per le reti senza fili **WLAN (*Wireless LAN, IEEE 802.11[3]*)**, specifica diversi metodi di trasmissione dati e il metodo di accesso (livello fisico e MAC).

Durante la comunicazione wireless vi è la possibilità che vi siano interferenze causate dalle cosiddette **collisioni**, ovvero trasmissioni simultanee da parte di due o più terminali wireless nella stessa banda di frequenza.

Per evitare tali collisioni, la stazione che deve trasmettere un *frame* (piccolo pacchetto dati) verso una certa destinazione, fa in modo che la destinazione emetta un *frame di risposta* detto **ACK (*Acknowledge*)**, in modo che le stazioni vicine possano astenersi dal trasmettere.

Il metodo di accesso al canale di trasmissione che viene usato per evitare l'insorgere di collisioni si chiama **CSMA/CA(*Carrier Sense Multiple Access with Collision Avoidance* -** accesso multiplo a ricezione di portante con collisione evitate).

Secondo il CSMA/CA, quando una stazione deve trasmettere un frame verso una certa destinazione si seguono i seguenti passi:

1. la stazione mittente sollecita la stazione di destinazione inviandole un *frame di richiesta di trasmissione* che avverte dell'imminente spedizione di dati e della lunghezza dei frame. (Le stazioni vicine al mittente che ricevono il frame di richiesta di trasmissione, restano in attesa della conclusione della trasmissione);

2. la stazione di destinazione invia a sua volta un frame di risposta in cui si ricopia la lunghezza del frame atteso. (Anche le stazioni vicine al

destinatario ricevono il frame di risposta ed attendono la conclusione della trasmissione);

3. quando il frame di risposta torna al mittente ha inizio la trasmissione dei frame di dati.

2.5 Lo standard Wi-Fi

Nelle telecomunicazioni, per **Wi-Fi (*Wireless Fidelity*)** si intende la tecnologia che permette di collegare i dispositivi attraverso una WLAN, basandosi sulle specifiche dello standard IEEE 802.11[3]. La rete locale così ottenuta si può interconnettere alla rete Internet tramite un *router* ed usufruire di tutti i servizi di connettività offerti dall'**ISP (*Internet Service Provider*)**.

La connessione alla rete avviene tramite gli **AP (*Access Point*)**, dispositivi di ricetrasmissione radio che sono collegati tra loro tramite cablaggio in rete locale o in maniera wireless che danno vita ad un *sistema distribuito*.

La parte radio (o interfaccia radio AP-utente) costituisce la rete di accesso, mentre la LAN o WLAN che collega tutti gli APs rappresenta la rete di trasporto.

Qualunque dispositivo che integra le specifiche Wi-Fi può connettersi a reti di questo tipo, ottenendo sistemi di trasmissione dati molto flessibili ed estendibili.

Nel campo della telefonia, già da qualche anno, quasi tutte le case costruttrici di smarphone progettano e distribuiscono dispositivi mobili che integrano schede WI-Fi, rendendo possibile il collegamento smartphone-AP. In questo modo l'utente può sfruttare **reti WLAN spesso gratuite, accessibili attraverso Hot Spot pubblici** e navigare su Internet senza limitazioni di bundle di download.

Tutto questo ha reso il Wi-Fi uno standard direttamente concorrente all'accesso a Internet (a pagamento) attraverso reti GSM o UMTS.

2.6 Rete mobile GPRS

Il **GPRS** (***General Packet Radio Services***) è una tecnologia wireless per dispositivi mobili come smartphone e palmari che serve ad avviare la navigazione su rete Internet.

La velocità di trasmissione massima è 114 Kbps.

2.7 Rete mobile 3G

Col termine **3G** si indica la terza generazione di servizi telefonici senza fili che consentono il trasferimento di *dati voce* (per le telefonate digitali) e di *dati non-voce* (come download da Internet, *streaming video*, l'invio/ricezione di email, *instant messagging*, ecc).

La nascita di questa tecnologia è dovuta alla crescente domanda di servizi multimediali che richiedono caratteristiche tecniche superiori rispetto a quelle offerte da GPRS.

La velocità di trasmissione massima di download offerta da questa tecnologia è di 384 Kbps.

In confronto alla tecnologia GPRS, il 3G permette di avere una qualità dei servizi Web maggiore, un incremento della stabilità e dell'affidabilità del servizio, una velocità di trasmissione praticamente triplicata e una larghezza di banda trasmissiva aumentata.

Oltre che essere impiegata per il networking degli smartphone, recentemente questa tecnologia è stata estesa ad alcuni modelli di *tablet* e PC portatili.

Capitolo 3

Il protocollo Bluetooth

3.1 Standard Bluetooth (BT)

Nelle campo delle telecomunicazioni, il Bluetooth[1] è una specifica industriale per **WPAN (*Wireless Personal Area Network*)**, che fornisce un metodo standard, economico e sicuro per lo scambio dati tra dispositivi con caratteristiche diverse (spesso aventi piccole dimensioni) attraverso una frequenza radio a corto raggio.

Come accennato, lo scambio dati wireless (senza fili) è una comunicazione in cui i dati, sotto forma di segnali ad onde radio, viaggiano nell'aria eliminando qualsiasi tipo di connessione fisica tra i dispositivi. Il collegamento rimane comunque efficiente e affidabile anche in ambienti con forte presenza di interferenze e campi elettromagnetici (*elettrosmog*).

Affinché due dispositivi possano comunicare tra loro, entrambi i dispositivi devono contenere una **radio Bluetooth**. Questa radio ha un consumo energetico minimo e dimensioni ridotte per cui normalmente viene incorporata in un *chip* integrato.

Alcuni dispositivi di uso comune che integrano schede BT sono per esempio palmari, PC, *laptop*, stampanti BT, *console* per videogiochi e naturalmente gli smartphone.

3.2 Specifica Bluetooth

La specifica Bluetooth nasce nel 1998 ad opera della *Ericsson Mobile Communication*. Con l'instaurazione dalla **SIG [2] (*Bluetooth Special Interest Group*)**, associazione formata da grandi sviluppatori tecnologici come *Sony Ericsson*, *IBM*, *Intel*, *Toshiba*, *Nokia* e altre società associate, si è avuta la formalizzazione della specifica Bluetooth come lo standard internazionale **IEEE 802.15.1**[3].

Questo standard è stato progettato prefiggendosi vari obiettivi quali:

- ottenere bassi consumi elettrici,
- avere un'azione di copertura a corto raggio,
- avere un basso costo di produzione per i dispositivi compatibili,
- consentire il collegamento wireless tra dispositivi differenti.

3.3 Origini del nome e del logo

Il nome di questo standard è ispirato a *Harald Blåtand* (in inglese Harold Bluetooth), re Aroldo I di Danimarca, abile diplomatico che unì le popolazioni scandinave introducendo la religione cristiana; quindi un riferimento più che adatto per un protocollo capace di mettere in comunicazione dispositivi anche molto diversi tra loro.

Il logo del BT unisce due rune nordiche (raffigurazioni naturalistiche delle forze che regolano l'universo, *Hagall* e *Berkanan*).

Figura 3.1: Logo dello standard BT

3.4 Le caratteristiche delle comunicazioni BT

Le comunicazioni Bluetooth si servono di una **banda RF senza licenza**, nel campo di azione che si estende dai 2,4 ai 2,48 GHz. Questa banda di frequenza **ISM (*Industrial, Scientific and Medical*)** può essere utilizzata liberamente e gratuitamente da chiunque per più scopi.

L'aspetto positivo dell'impiego di ISM per le telecomunicazione è che ha delle frequenze utilizzabili e a costo zero ma che dispongono di una grandezza di banda limitata.

Nel caso peggiore, se più dispositivi devono sfruttare contemporaneamente questa banda, si potrebbero verificare degli errori o dei ritardi di trasmissione derivanti da interferenze reciproche.

Per risolvere questi problemi i dispositivi radio Bluetooth ricorrono ad una tecnica chiamata **hopping di frequenza a largo spettro**.

La **FHSS (*Frequency Hopping Spread Spectrum*)** è una tecnica che consiste nel trasmettere il segnale usando una sequenza pseudo casuale di frequenze, in cui un segnale passa (**salta**) rapidamente da una frequenza all'altra ad intervalli di tempo fissati (*hop rate - rapidità di salto*); solo i ricevitori che conoscono la *sequenza di hopping* del trasmettitore possono così ricevere correttamente l'informazione;

Il segnale radio passa tra 79 frequenze comprese tra 2,4 GHz e 2,48 GHz, compiendo 1600 hps (salti di frequenza al secondo), ad intervalli di 1 MHz.

La sequenza dei salti è determinata dal ***clock di sistema* (CLK)** che sincronizza i *timer* di tutte le unità connesse facendo in modo che le trasmissioni Bluetooth rimangono su una singola frequenza per un **tempo insufficiente** ad essere colpite dalle interferenze.

3.5 Topologie delle reti Bluetooth

Le specifiche Bluetooth prevedono tre tipi di topologie:

- **Punto-punto**: due dispositivi sono collegati direttamente (è la topologia usata in questa tesi);

Figura 3.2: Rete BT punto-punto

- **Punto-multipunto**: collegamento di un dispositivo con più dispositivi;
- **Scatternet**: collegamento di più reti di dispositivi (punto-punto o punto-multipunto).

3.5.1 Rete Piconet

Quando due dispositivi Bluetooth stabiliscono una connessione, creano una rete personale detta **Piconet**, che in base al numero di *Host Bluetooth* coinvolti può avere una topologia punto-punto o una punto-multipunto.

I dispositivi presenti in una Piconet possono essere di due tipi: **MASTER** o **SLAVE**.

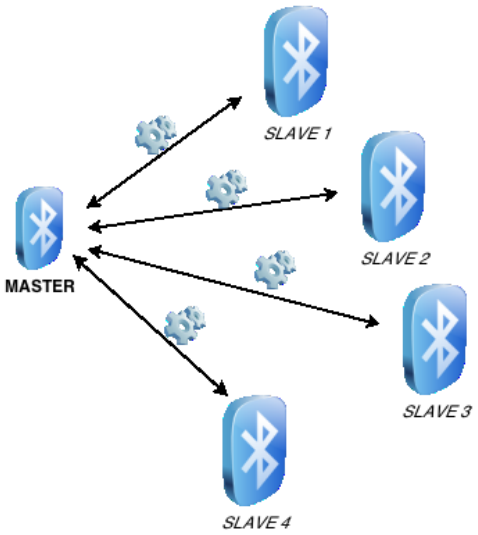

Figura 3.3: Rete Piconet

Ogni Piconet può contenere fino ad otto dispositivi Bluetooth dove uno di essi funge da MASTER, mentre gli altri sette fungono da SLAVE attivi (unità sincronizzate al MASTER).

- ***MASTER***: è il dispositivo che si occupa della sincronizzazione del clock dei dispositivi SLAVE e della sequenza di hopping di frequenza da eseguire;

- ***SLAVE***: sono Host BT che operano insieme, sincronizzati in base al clock e alla sequenza di hopping definiti dal MASTER.

Lo standard Bluetooth prevede alcune modalità di funzionamento che permettono di ampliare virtualmente la dimensione della Piconet, abilitando la dissociazione temporanea degli Host (riducendo il consumo delle batterie), che comunque restano informati periodicamente dal MASTER sulle attività della rete. Se ne hanno necessità, i dispositivi dissociati possono rientrare attivamente nella Piconet in qualsiasi momento, purché ci sia ancora posto.

Più Piconet possono condividere lo stesso spazio fisico senza interferire reciprocamente, in quanto usano diversi canali di frequenza.

3.5.2 Rete Scatternet

Una **Scatternet** è una rete formata da più Piconet che di fatto permette di espandere la rete BT oltrepassando il limite di degli otto elementi sulla singola Piconet.

In essa tutte le comunicazioni tra le diverse Piconet sono filtrate attraverso i singoli MASTER delle Piconet.

Ogni Scatternet ha la possibilità di includere fino a 10 Piconet, arrivando a gestire 80 dispositivi Bluetooth. Oltre questo numero la rete si satura, poiché lo standard Bluetooth si serve soltanto di 79 frequenze.

3.6 Temporizzazione e clock

La tecnologia Bluetooth sincronizza la maggior parte delle operazioni con un **clock in tempo reale**.

Il clock serve a sincronizzare gli scambi di dati tra i dispositivi, distinguendo tra pacchetti ritrasmessi o persi, generando una sequenza di segnali pseudo-casuale, predicibile e riproducibile.

Ogni dispositivo Bluetooth ha il suo **CLKN (*native clock*)** che ne controlla la temporizzazione rispetto alla rete.

Oltre a questo valore si definiscono altri due clock:

- **CLK (*clock della Piconet*)**: coincide con il CLKN dell'unità MASTER della Piconet. Gli SLAVE attivi devono sincronizzare il proprio CLKN con il CLK e per farlo aggiungono un *offset* al CLKN.

- **CLKE (*clock stimato*)**: serve al MASTER durante la creazione di una connessione verso uno SLAVE. La sua sincronizzazione deriva da un *offset* dal CLKN.

3.7 Modalità operative di un dispositivo BT

Un dispositivo Bluetooth si può trovare in due stati:

3.7.1 Standby

Questo è lo *stato di default* per un dispositivo Bluetooth e serve a far risparmiare batteria facendolo funzionare in *low-power*.

L'unità si trova automaticamente nello stato di *standby* se non è connesso o non è coinvolto dalle attività della Piconet. Durante questo stato l'Host continua ad ascoltare il canale per ricevere eventuali messaggi dal MASTER in modo da sapere se può/deve passare allo stato *connection*.

3.7.2 Connection

In questo stato la connessione tra MASTER e SLAVE è stata stabilita e lo scambio pacchetti è abilitato.

Lo stato *Connection* può essere terminato in qualsiasi momento da entrambe le parti.

Durante lo stato *Connection* un'unità Bluetooth può assumere diverse modalità operative:

- **Active mode**: l'unità partecipa attivamente alla Piconet;
- **Hold mode**: è una modalità a basso consumo di potenza in cui lo SLAVE non supporta temporaneamente l'invio e la ricezione di pacchetti;
- **Sniff mode**: è una modalità a basso consumo di potenza che riduce l'attività di ascolto degli SLAVE. In questa modalità lo SLAVE è comunque considerato un membro attivo della Piconet;
- **Park mode**: lo SLAVE non partecipa attivamente alla Piconet ma rimanere sincronizzato al canale riducendo il consumo di potenza.

Si aggiungono a questi stati dei **sottostati temporanei**, usati per costruire una nuova Piconet oppure per aggiungere nuovi SLAVE ad alla rete esistente:

- **Inquiry** e **Inquiry Scan**: ricerca ed interrogazione di nuovi dispositivi abilitati (in modalità *discoverable*) presenti nel *range* di copertura dell'antenna;
- **Inquiry Response**: sottostato in cui si acquisisce il numero d'identificazione e/o il nome del nuovo Host;
- **Page**, **Page Scan**, **MASTER Response**, **SLAVE Response**: servono a realizzare una nuova connessione tra due dispositivi BT.

3.8 Architettura dello standard Bluetooth

Similmente all'architettura ISO OSI, lo standard Bluetooth contiene una serie di specifiche che definiscono i vari livelli del protocollo stesso, anche se in questo caso alcune funzioni sono distribuite su più livelli; quindi differenti protocolli sono utilizzabili per differenti applicazioni.

Lo *stack* BT permette di eseguire vari attività come ricercare nuovi dispositivi, scoprire i servizi attivi su di essi e quindi usare tali servizi. Si distingue in una parte hardware ed una software.

Il sistema radio Bluetooth fa parte della parte hardware e consiste del **Livello Radio** e del **Livello Baseband**.

Il Livello Baseband a sua volta si suddivide in due sottolivelli: **Link Controller** (unità deputata al controllo del collegamento tra Host) e **Link Manager** (unità di supporto per il Link Controller).

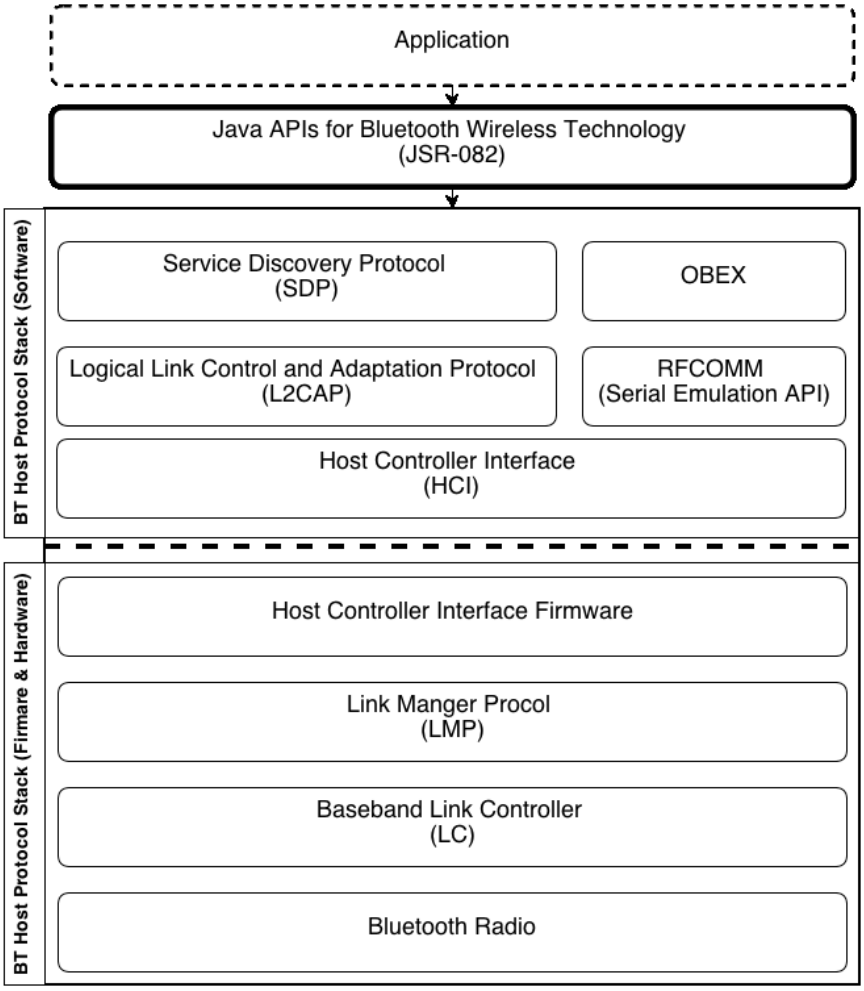

Figura 3.4: Stack del protocollo BT

3.8.1 Livello Radio

Il livello radio è il livello più basso dello standard Bluetooth. È una parte fondamentale di un dispositivo BT in cui sono definite tutte le specifiche del canale radio per la trasmissione fisica dei segnali radio.

Utilizza la banda ISM a 2,4 GHz, inviando i dati alla velocità di 1 Mbps, ricorrendo alla tecnica *FHSS*.

Tabella 3.1: Classi di potenza dei dispositivi Bluettoth

Classe	Potenza (mW)	Potenza (dBm)	Distanza (m)
Classe 1	100	20	100
Classe 2	2,5	4	10
Classe 3	1	0	1

3.8.2 Livello Baseband

Il **Baseband** è il livello in cui viene gestito il canale fisico dove si eseguono:

- la gestione della connessione;
- le operazioni di codifica/decodifica;
- il controllo della temporizzazione a basso livello;
- la correzione degli errori;
- l'applicazione dell'algoritmo della sequenza di *hopping* (salti) di frequenza;
- la sicurezza.

Il protocollo di questo livello è implementato dal ***Link Controller*** che gestisce i collegamenti sincroni e asincroni e dal ***Link Manager*** che invece realizza la procedure di *Page scan* e *Inquiry scan*.

Link Controller

Il **Link Controller** è il livello che si occupa di svolgere operazioni relative alla trasmissione di diversi pacchetti in risposta ai comandi del Link Manager, fornisce ai livelli superiori servizi quali le connessioni e il controllo della potenza trasmessa.

Durante la connessione i Link Controller dei dispositivi connessi gestiscono il processo di creazione della connessione stessa richiesta dal Link Manager, mantenendo nel tempo la connessione stabilita e scambiando informazioni col Baseband.

Link Manager

Si occupa della gestione delle WPAN, della configurazione del link e della sicurezza.

Per quanto riguarda la gestione delle WPAN tale livello richiede il nome dei dispositivi, sincronizza i clock ed infine controlla la potenza di trasmissione.

L'instaurazione e la supervisione del link viene fatta attivando le modalità di risparmio energetico e negoziando la dimensione dei pacchetti (rispettando la QoS).

La sicurezza dello standard Bluetooth si basa su tre servizi critici che sono l'**autenticazione**, l'**autorizzazione** e la **cifratura**.

1. **Autenticazione**: si utilizza una chiave di collegamento (*link key*) che può essere una *unit key* (la stessa chiave per tutte le connessioni) o una *combination key* (specifica chiave per una coppia di dispositivi). La link key può essere generata dinamicamente o durante il **processo di *pairing*** (accoppiamento di due dispositivi BT).

2. **Autorizzazione**: è il processo con cui un Host determina quali sono i permessi di accesso riguardanti un particolare servizio offerto da un altro dispositivo Bluetooth.

3. **Cifratura dei dati trasmessi tramite chiave a 128 bit**: serve a mantenerne la riservatezza dei dati trasferiti, garantendo che solo il destinatario che ha la chiave di cifratura potrà decifrare e vedere tali dati, impedendo attacchi di *sniffing*.

3.9 Protocolli di comunicazione Bluetooth

I protocolli di comunicazione BT sono la parte software dell'architettura dello stack Bluettoth.

3.9.1 SDP (Service Discovery Protocol)

SDP si occupa di scoprire quali servizi sono supportati da un dispositivo. Il meccanismo di scoperta si basa sulla presenza di due attori: un server e un client.

Il client interroga il server per scoprire quali servizi e quali caratteristiche specifiche (secondo un numero detto *UUID*) sono attivi.

È da notare che **il protocollo SDP fornisce dei mezzi per scoprire i servizi ma non per accedervi**.

3.9.2 RFCOMM (Radio Frequency Communication Protocol)

RFCOMM* è un protocollo *stream-oriented (che richiede di stabilire una connessione tra i dispositivi prima di inviare i dati), che fornisce l'emulazione di una linea seriale di tipo **RS-232** tra due dispositivi Bluetooth.

Questo protocollo è in grado di eseguire una correzione degli errori basato sul *controllo del flusso* e di emulare un *null-modem*.

Il protocollo RFCOMM è un protocollo molto utile in quanto permette di continuare ad usare applicazioni già esistenti che utilizzavano un meccanismo di comunicazione seriale.

Le API Java **JSR-82** non definiscono nessuna nuova classe o interfaccia per utilizzare RFCOMM ma si limitano a riutilizzare delle classi e delle interfacce già esistenti che fanno parte del **GCF (*Generic Connection Framework*)**.

Questa scelta permette di avere flessibilità nell'utilizzo dei protocolli di comunicazione permettendo di utilizzarne indifferentemente diversi senza modificare il codice, agevolandone il *porting* su piattaforme diverse.

Le JSR-82 saranno trattate più diffusamente nel prossimo capitolo.

Nel progetto trattato RFCOMM è stato utilizzato per il trasferimento di dati seriali tra MercuryBTClient e MercuryBTServer.

3.10 Profili Bluetooth

Un profilo Bluetooth definisce le procedure, le specifiche (i messaggi) ed i protocolli necessari all'implementazione di ogni applicazione o modello di utilizzo del dispositivo.

I dispositivi (spesso mobili) che implementano la tecnologia Bluetooth devono essere necessariamente conformi ad una serie di profili che garantiscano l'operatività a contatto con dispositivi diversi o applicazioni scritte in linguaggi differenti.

3.10.1 SPP (Serial Port Profile)

È il profilo che riguarda la comunicazione seriale. Definisce i requisiti di interoperabilità fra i livelli più bassi e i ruoli di parità tra i dispositivi.

In questo tipo di comunicazione la relazione MASTER/SLAVE non esiste, cioè durante le comunicazioni seriali tutti i dispositivi sono uguali.

SPP serve ad emulare la porta seriale RS-232, sfruttando RFCOMM.

Nel corso del progetto si vedrà come SPP permetta la comunicazione tra MercuryBTClient e MercuryBTServer.

Capitolo 4

J2ME - Java 2 Micro Edition

4.1 Programmazione in Java

Java è un linguaggio di programmazione **OOP (*Object Oriented Programming*)** nato nel 1991 dal lavoro del gruppo di ricerca e sviluppo dell'azienda californiana *Sun Microsystems*.

Dal 27 gennaio 2010 Java è un marchio registrato di proprietà della *Oracle Corporation* dopo che la Sun Microsystem è stata acquistata per 7,4 miliardi di dollari.

Lo scopo iniziale di questo linguaggio era creare una tecnologia affidabile per l'industria elettronica, basata su un **software multipiattaforma di facile utilizzo**.

Nel corso degli anni il linguaggio Java ha modificato i suoi obiettivi diffondendosi prima con lo sviluppo di **applet lato client** e poi con **tecnologie *servlet*** (applet che operano all'interno di server) come a esempio Tomcat.

Figura 4.1: Logo dell'OOP Java

4.2 Le distribuzioni di Java

Con il rilascio di Java 2 si è deciso di raggruppare piattaforme, **API (*Application Programming Interface* - librerie per Java)** e strumenti di sviluppo di ogni specifico settore di mercato, in *platform-edition*.

Ogni **platform-edition** è basata su una **JVM (*Java Virtual Machine*)** che si appoggia sull'hardware e sul sistema operativo sottostante, conferendo a Java un alto grado di portabilità e di astrazione.

Altre funzionalità principali di Java sono la creazione di un oggetto e le operazioni di *reset* e pulizia della memoria (svolte dalla *routine* di **garbage collection**).

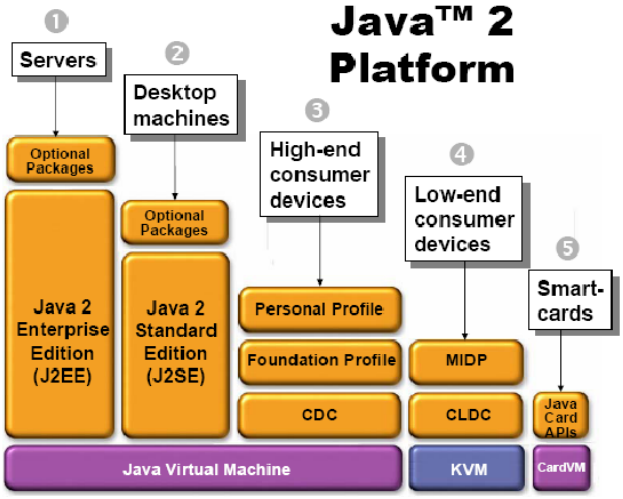

Figura 4.2: Piattaforme Java a confronto

Le piattaforme di Java 2 (Fig. 4.2)sono:

- **J2SE (*Java 2 Platform Standard Edition*)**: fornisce un ambiente per lo sviluppo di applet per *workstation* e PC. Qualsiasi applicazione compilata in *bytecode* per J2SE necessita di una **JRE (*Java Runtime Environment*)** per l'esecuzione. Il Java 2 SE Runtime Environment contiene la JVM, le API standard Java e un *launcher* per le applicazioni Java.

- **J2EE (*Java 2 Platform Enterprise Edition*)**: fornisce un ambiente basato su J2SE in cui sono state aggiunte API per la realizzazione di architetture solide, complete e scalabili (che possono espandersi), rivolte prevalentemente ad applicazioni lato server.

- **J2ME (*Java 2 Platform Micro Edition*)**: è un insieme di tecnologie e specifiche software per lo sviluppo di applet lato client in **dispositivi embedded** (sistemi elettronici progettati per determinate applicazioni)

e per dispositivi di elettronica di consumo che hanno risorse limitate, come i telefoni cellulari, i PDA, le stampanti, gli smartphone e simili.

4.3 La piattaforma J2ME

J2ME[4] (*Java 2 Micro Edition*) è la piattaforma Java per dispositivi con scarse risorse e caratteristiche disomogenee.

J2ME può essere considerata come una collezione di specifiche, ciascuna delle quali è adatta ad un sottoinsieme di dispositivi che hanno le medesime caratteristiche.

Questa piattaforma è stata concepita per ottimizzare l'utilizzo della memoria, del microprocessore e delle ridotte capacità di I/O dei dispositivi che ne fanno uso.

L'utilizzo di *tool* dedicati come *Wireless Toolkit* semplifica l'attività di sviluppo e di *testing* di applet scritte in J2ME su PC in quanto ne permettono l'emulazione senza necessità di continue installazioni fisiche.

È da sottolineare il fatto che **lo sviluppo di un'applicazione su J2ME è più complesso rispetto ad quella di una normale applicazione scritta in J2SE** in quanto bisogna considerare nuovi concetti riguardanti la **Configurazione** e il **Profilo** del dispositivo utilizzato.

4.4 Architettura della piattaforma J2ME

La piattaforma J2ME ha un'architettura modulare che consiste di tre componenti che sono la **Virtual Machine**, la **Configurazione** e il **Profilo**.

4.4.1 Virtual Machine

La **VM (*Virtual Machine*)** è il componente software che esegue le applicazioni J2ME traducendo uno speciale gruppo di istruzioni detto *bytecode* nelle corrispondenti istruzioni interpretabili dal microprocessore del dispositivo.

Il **bytecode** è un linguaggio intermedio tra il linguaggio macchina e il Java, usato per descrivere le operazioni che costituiscono un programma come se questo fosse un flusso formattato di *byte*.

La VM è inoltre responsabile:

- del caricamento del codice in memoria;
- dell'isolamento del *runtime* Java dal resto del sistema operativo impiegato;
- della gestione della memoria RAM;
- della gestione dei *thread*;
- delle risorse fondamentali per l'esecuzione delle applicazioni.

4.4.2 Configurazione

Una **configurazione** è quella parte dell'architettura di J2ME che fornisce le funzionalità di base per la VM, oltre a stabilire il linguaggio supportato dall'ambiente d'esecuzione.

Nello specifico una configurazione costituisce l'astrazione delle funzionalità di base offerte dal microprocessore e dai sistemi d'interfacciamento verso quelle che sono le periferiche del dispositivo stesso.

Una configurazione però non costituisce un ambiente di esecuzione completo in quanto non descrive ne il modello applicativo (gestione del ciclo di vita delle applicazioni) ne l'interfaccia utente dell'App.

Le caratteristiche prese in considerazione in una configurazione sono:

- il tipo e la potenza di calcolo del microprocessore;
- la quantità di *memoria RAM* e di massa disponibile;
- le connettività di rete;
- la presenza o meno di un display e le sue caratteristiche tecniche.

Attualmente sono disponibili due configurazioni, dette **CLDC** e **CDC**.

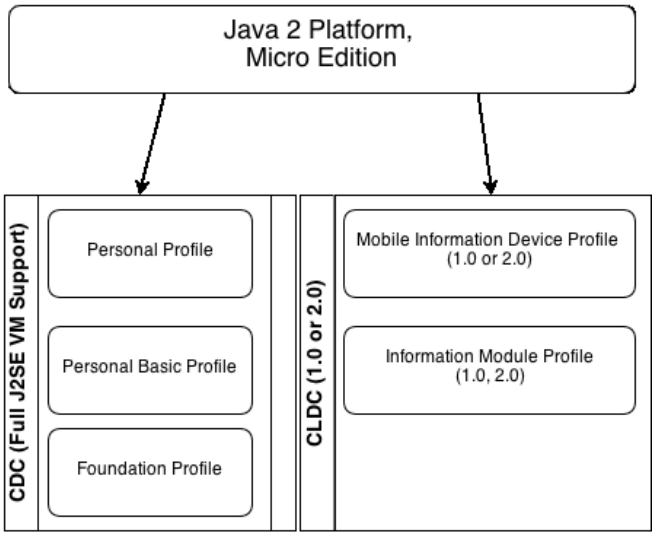

Figura 4.3: Architettura CLDC/CDC

CLDC (Connected Limited Device Configuration)

La **CLDC[5]** è una configurazione rivolta a dispositivi come telefoni cellulare consumer e smartphone delle vecchie generazioni.

La memoria RAM minima richiesta per il suo utilizzo è di 128 KByte. I microprocessori sono tipo *RISC* (*Reduced Instruction Set Computer*) oppure *CISC* (*Complex Instruction Set Computer*) a 16-32 bit.

I dispositivi che ne fanno parte sono mobili (senza fili) ed hanno un'alimentazione a batteria con un consumo energetico relativamente basso.

Le connessioni sono non continue, rivolte a reti wireless eterogenee aventi una larghezza di banda minima di 9600 bps.

La JVM di questa configurazione si chiama **KVM (*Kilobyte Virtual Machine*)** il cui *core* occupa circa 32-80 KByte.

Date le sue dimensioni molto ridotte KVM non supporta **JNI (*Java Native Interface*)** e il calcolo in virgola mobile rendendo inutilizzabili le variabili di tipo *float*, *double* o ad *array* che ricorrono a questi tipi di dato.

CDC (Connected Device Configuration)

La CDC[6] è una configurazione per dispositivi con prestazioni a metà strada tra CLDC e i *sistemi desktop*.

Questi dispositivi dispongono di almeno 2 MB totali di memoria (*RAM* e *ROM*) e processori a 32 bit più veloci che supportano un ambiente graficamente più complesso. CDC si può applicare a PDA evoluti, smartphones evoluti, *gateways*, ecc.

La JVM utilizzata per questa configurazione è la **CVM (*C Virtual Machine*)**.

4.4.3 Profilo

Il ***profilo*** completa la configurazione del dispositivo, definendo:

- il modello applicativo;
- le classi per l'interfacciamento verso l'utente;
- le funzionalità per l'esecuzione di un'applicazione.

Il profilo del dispositivo aggiunge nuove API permettendo al programmatore di scrivere applicazioni per specifici settori di mercato.

Una configurazione di J2ME può avere più profili, a loro volta estendibili.

Per ciascuna configurazione sono disponibili più profili applicativi:

CLDC:

- **MIDP[7] (*Mobile Information Device Profile*)**: è il più conosciuto tra i profili perché è la base per la trasmissione wireless con Java su **protocollo HTTP 1.1**. Ha una GUI abbastanza semplice e una memorizzazione locale dei dati temporanea o persistente. La persistenza dei dati dipende dal **RMS (*Record Management System*)**.

  Le applet scritte per MIDP sono chiamate MIDlets le quali possono far ricorso alle API di MIDP e di CLDC.

- **IMP(*Information Module Profile*)**: è un profilo per sistemi che non necessitano di interfaccia utente, come per esempio nei sistemi di telecontrollo domotico.

CDC:

- **FP(*Foundation Profile*)**: profilo include la maggior parte delle API di J2SE versione 1.3. È richiesto per il funzionamento di dispositivi che necessitano di una completa implementazione della VM.

- **PBP(*Personal Basis Profile*)**: aggiunge le funzionalità di base per l'interfaccia utente al FP. È rivolto a dispositivi con display semplici, non permettendo l'attivazione di più finestre nello stesso momento.

4.5 Requisiti hardware di MIDP

I requisiti minimi di MIDP sono:

- **Memoria**: 128 KB di RAM e 32 KB per l'*heap* Java. Data la ridotta dimensione dell'*heap*, il *garbage collector* libera dalla memoria gli oggetti deallocandoli appena non sono più necessari. Per la memorizzazione permanente dei dati sono richiesti altri 8 KB di memoria non volatile.

- **Display**: si possono utilizzare display con dimensioni minime di 96x54 pixel con almeno due colori.

- **Dispositivi di input**: tastierino con numeri da 0 a 9, frecce per la navigazione e un bottone di selezione.

- **Connettività**: è richiesto il supporto per HTTP 1.1 direttamente sullo stack dei protocolli Internet o utilizzando una connessione wireless mediante un *gateway*.

4.6 MIDlet

Una MIDlet è un'applicazione Java installabile su un dispositivo in cui è predente MIDP. È composta da almeno una classe Java che deriva dalla classe astratta javax.microedition.midlet.MIDlet ed eventualmente da altre classi e file richiesti nel progetto.

Il codice compilato di tale applicazione mobile si identifica come un file con estensione **JAR** ***(Java ARchive)*** che può contenere più di una MIDlet al suo interno, diventando così una *MIDlet suite*. Le MIDlet di una suite vengono installate/disinstallate tutte insieme, come fossero una singola entità.

All'interno del JAR è presente il **Manifest**, un file di testo (certificato) che garantisce la provenienza e l'attendibilità del codice, descrivendo il contenuto dell'archivio ed altre informazioni (una serie di attributi e valori) necessarie all'**AMS (*Application Manager Software*)** per verificare se l'applicazione è in grado di funzionare sul dispositivo. Informazioni simili sono riportate dal file **JAD (*Java Application Descriptor*)**.

Le MIDlet operano in un ambiente ristretto detto **sandbox** che ne limita l'utilizzo assicurando che l'applicazione non possa danneggiare il dispositivo su cui viene eseguita.

Quando vengono eseguite all'interno della VM seguono un **ciclo di vita** controllato attraverso dei metodi standard. Una MIDlet si può trovare in uno dei tre stati: *Paused*, *Active*, *Destroyed*.

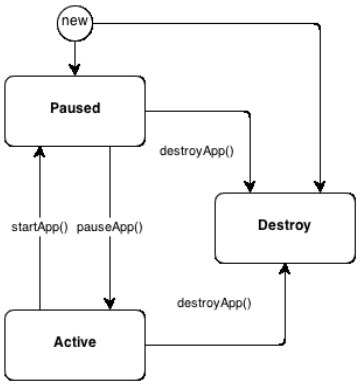

Figura 4.4: Ciclo di vita di una MIDlet

All'avvio una MIDlet viene caricata in memoria ed inizialmente si trova nello stato *Paused*. Se durante l'esecuzione si verificano dei problemi, viene sollevata un'eccezione e si raggiunge lo stato Destroyed; in caso contrario viene chiamato il metodo startApp() e lo stato passa da *Paused* ad *Active*.

Se non si riscontrano problemi, l'applicazione può continuare la propria esecuzione e terminare senza sollevare eccezioni.

4.7 La specifica JSR-82

La specifica **JSR-82[8]** definisce lo standard Java per l'accesso alla tecnologia Bluetooth definendo soltanto le interfacce alle quali un *vendor* deve attenersi per realizzare uno *stack* proprietario conforme allo standard BT.

JSR-82 è un'estensione opzionale della piattaforma J2ME, progettata per esser supportata da calcolatori dotati di risorse limitate come gli smartphone. Essendo compatibile con la piattaforma J2SE, permette di far comunicare applet sviluppate per J2SE con delle MIDlet tramite il protocollo Bluetooth.

La JSR-82[9] consiste nei seguenti *package* indipendenti:

- **javax.microedition.io**: gestore delle connessioni Bluetooth (*L2CAP* e *RFCOMM*). (Fig. 4.5);
- **javax.obex**: implementazione delle funzioni OBEX;
- **javax.bluetooth**: è il *core* delle API Java Bluetooth che mette a disposizione delle MIDlet le funzionalità dei profili Bluetooth Generic Access Profile e Service Discovery. (Fig. 4.6).

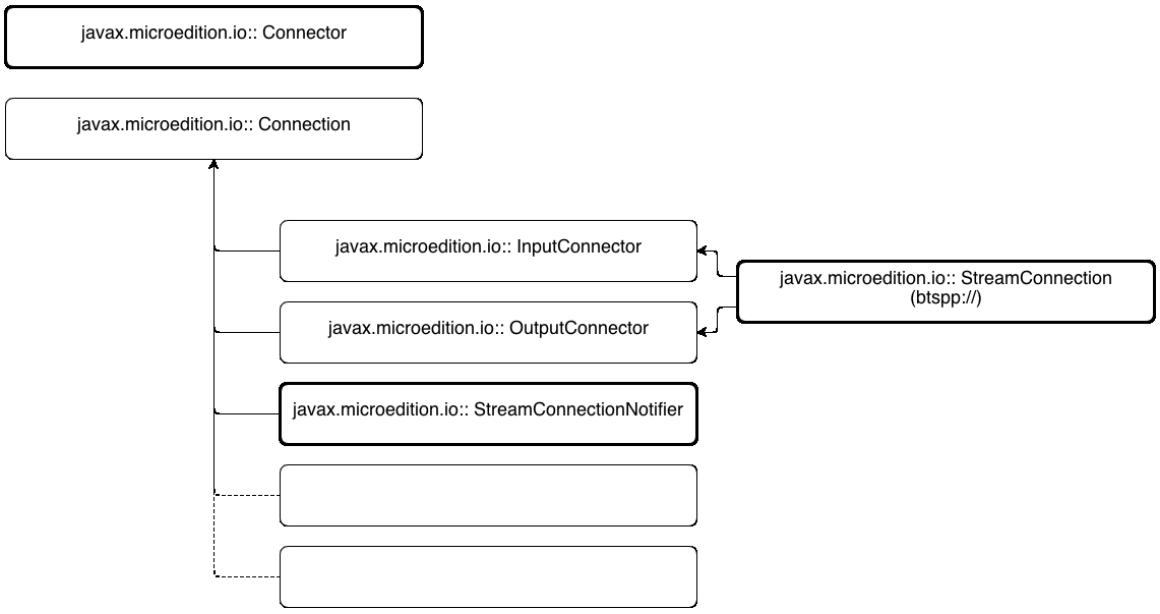

Figura 4.5: Package javax.microedition.io

Le classi presenti in JSR-82 sono:

- la classe **javax.bluetooth.LocalDevice** rappresenta il dispositivo Bluetooth locale.
  Contiene metodi per accedere alle proprietà generali del dispositivo come nome ed indirizzo Bluetooth e il metodo setDiscoverable() per rendere individuabile il dispositivo durante l'*Inquiry scanning*;

- la classe **javax.bluetooth.RemoteDevice** rappresenta le informazioni raccolte su di un dispositivo Bluetooth remoto.
  Contiene metodi per accedere alle proprietà generali del dispositivo come nome ed indirizzo Bluetooth e altri metodi che servono per l'autenticazione e la cifratura delle connessioni;

- la classe **javax.bluetooth.DiscoveryAgent** fornisce metodi per il *Device discovery* e per il *Service discovery*.
  Attraverso il metodo startInquiry(), permette di avviare la sessione di *inquiry* per l'individuazione dei dispositivi visibili;

- l'interfaccia **javax.bluetooth.DiscoveryListener** consente alla MIDlet di ricevere gli eventi legati alle richieste di *Device discovery* e *Service discovery*.

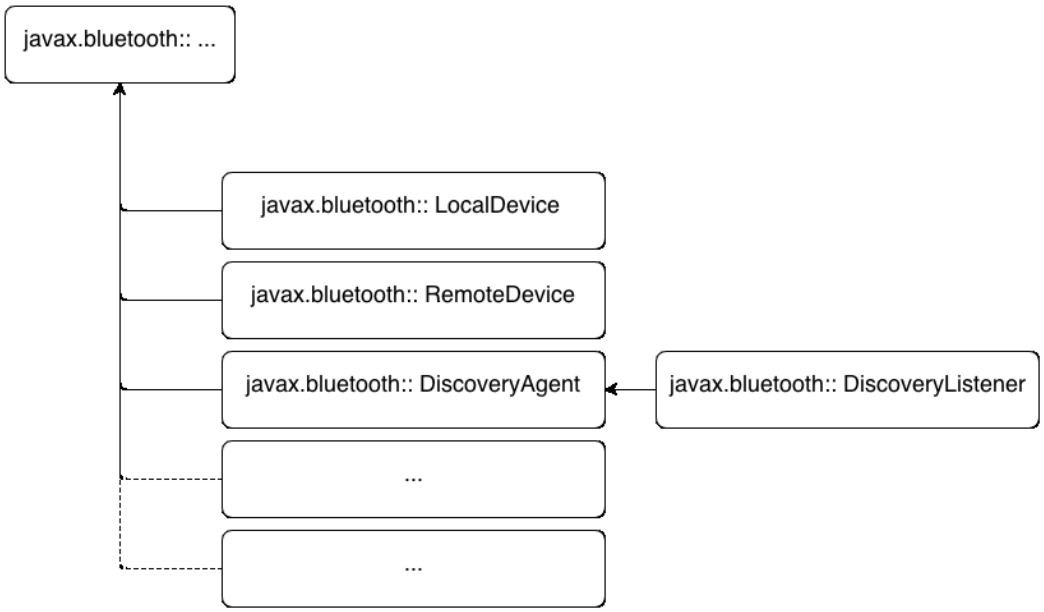

Figura 4.6: Package BT per J2ME

Capitolo 5

Sistema

5.1 Descrizione del sistema

Parte del codice di questa tesi si rifà al progetto open source **Hiisi Suite 1.6.3**, rilasciato sotto licenza **GPL**[13] (***GNU General Public License***) dal programmatore *Hiisi*.

Questa licenza consente agli utenti finali di copiare, ridistribuire e aggiungere modifiche al software, con l'unico obbligo di redistribuire il codice secondo il vincolo del ***copyleft***, e quindi continuando a mantenere libero il codice rimanendo sotto licenza GPL.

Come già anticipato nell'introduzione, il sistema descritto si compone di due programmi:

- **MercuryBTClient**;
- **MercuryBTServer**;

Questi due programmi fanno ricorso al **paradigma di comunicazione client/server**, che è caratterizzato dal **server** (servitore) che attende una richiesta e dal **client** (cliente) che invia le richieste al server tramite il protocollo di comunicazione dello stack Bluetooth.

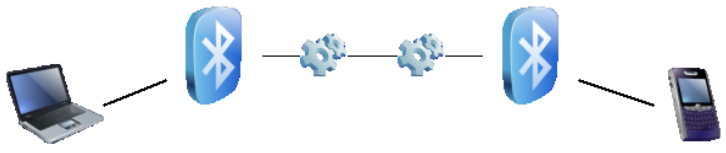

Figura 5.1: Paradigma client/server

Nello specifico l'applicazione server resta in attesa di una richiesta da parte del client che una volta ricevuta viene computata e rispedita al mittente.

Tale risposta fa da input al client che con i dati ricevuti può continuare la sua elaborazione che nel sistema descritto abilita una navigazione Internet.

5.1.1 MercuryBTServer

È un'applet J2SE che per essere eseguita richiede che sul sistema operativo su cui è eseguito sia installata una JRE. Per i test di utilizzo è stata utilizzata la JRE 7 e i sistemi operativi MS Windows 7 e GNU/Linux Ubuntu 12.10.

MercuryBTServer è un'applet tipo server che per interfacciarsi con lo stack BT del sistema ospite richiede l'utilizzo della libreria Java **BlueCove** (distribuita sotto licenza *Apache Software*[14] versione 2.0).

BlueCove[10] è una libreria Java che implementa la specifica **JSR-82**, *API Java for Bluetooth in J2SE*. È in grado di funzionare egregiamente sia su sistemi operativi proprietari *Microsoft* che su sistemi operativi liberi **GNU/Linux** a 32 o 64 bit (i386 o x86-64).

Su GNU/Linux è però necessario aggiungere il modulo **BlueCove-gpl** (licenziato sotto GPL) oltre a quello standard. Il modulo secondario è distribuito in modo separato a causa delle differenze di licenza tra la licenza Apache Software versione 2.0 e la GPL.

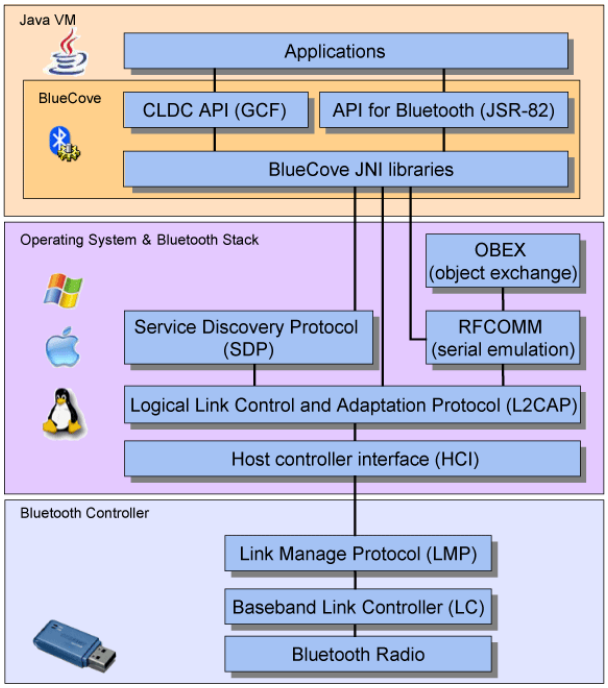

Figura 5.2: Architettura della BlueCove library

MercuryBTServer svolge due funzionalità, quella di *server Bluetooth* e quella di *gateway di rete*.

Un **gateway di rete** è un dispositivo hardware che permette la cooperazione di due o più reti differenti. In questo progetto MercuryBTServer (emulando un *hardware dedicato*) svolge questa funzionalità in quanto, partendo da una WPAN, permette di accedere alla rete Internet.

Figura 5.3: Schema di funzionamento del server (gateway)

MercuryBTServer inoltre è un **server Bluetooth** che grazie a BlueCove crea e gestisce un servizio seriale per volta. Tale servizio è registrato nel *service records* dove viene descritto da una serie di attributi che specificano il nome del servizio, il tipo di connessione ed altre informazioni utili al suo utilizzo.

La *service registration* è la procedura con la quale il server rende visibili il proprio *service record* ai potenziali client. Tale procedura inizia con la chiamata a Connector.open() che restituisce uno StreamConnectionNotifier; cioè si crea un nuovo service record per il servizio avente una serie di attributi di default che sono settati in base alla stringa passata come argomento al metodo Connetor.open().

La stringa ricevuta come parametro da Connector.open() identifica il protocollo da utilizzare e il dispositivo al quale connettersi. Tale stringa ha una struttura simile: **schema://target;parametri opzionali**, dove:

1. **schema**: rappresenta il protocollo di comunicazione da utilizzare. Nel caso della comunicazione seriale RFCOMM si utilizza **btspp** (**bt** = Bluetooth, **spp** = Serial Port Profile);

2. **target**: è l'indirizzo Bluetooth e il numero del canale utilizzato. Nella stringa di connessione lato server il target è la parola chiave **localhost** separata dall'**UUID** (***Universally Unique IDentifier***) del servizio;

3. **parametri opzionali**: è la parte conclusiva della stringa che raccoglie informazioni relative alle impostazioni di sicurezza.

A questo punto il servizio non è ancora visibile agli altri dispositivi. Per renderlo visibile occorre usare il metodo bloccante acceptAndOpen() sull'istanza della classe StreamConnectionNotifier di ritorno da Connector.open(). L'utilizzo di questo metodo pone l'applicazione server in attesa della connessione da parte del client.

Solo quando il client si connette, il metodo riattiva il *thread* restituendo un oggetto StreamConnection. Con questo oggetto l'applet crea gli *stream* di

input/output richiamando i metodi openDataInputStream() ed openDataOutputStream().

Il metodo openDataInputStream() riceve uno *stream* di caratteri in un *ciclo while* nel quale si esegue una serializzazione che serve a popolare le variabili necessarie alla navigazione dal browser dello smartphone.

Per questa serializzazione si utilizza il metodo read() che dallo *stream* prende in input un carattere alla volta fino a che non vi sono più caratteri da trattare.

A questo punto si utilizza uno StringBuffer (classe le cui istanze sono stringhe modificabili nel contenuto e nella lunghezza) in una serie di *if-else* nidificati dove si suddividono le stringhe bufferizzate in sottostringhe. Queste sottostringhe modificano il valore inizialmente vuoto delle variabili.

Le variabili tipo stringa di cui si è parlato finora sono:

- **requestMethod**: rappresenta i metodi del protocollo **HTTP** che specificano l'azione che il client ha richiesto al server Web. Il server esegue tale azione sulla risorsa identificata dall'**URI (*Uniform Resource Identifier* - sistema per indicare in modo univoco una risorsa)**.

  I metodi gestiti in questo progetto sono *GET* e *POST*.

  - **GET**: metodo che richiede informazioni al server, inviandogli pochi parametri tramite l'**URL** (***Uniform Resource Locator***, metodo standard per comunicare con un server in rete), attraverso una *query string* (parte di un URL che contiene i dati da passare in input al programma);
  - **POST**: metodo che invia informazioni al server, senza limite di tipo o di quantità dati;

- **requestURI**: è la variabile che indica l'URL che si vuole raggiungere durante la navigazione Web;

- **requestProtocol**: variabile che definisce il protocollo di trasmissione delle informazioni utilizzato come per esempio l'**HTTP/1.1 (*HyperText Transfer Protocol* - protocollo di trasferimento di ipertesti)**.

A questo punto vi è un if che verifica che requestProtocol abbia un valore stringa uguale alla stringa "HTTP/1.1". Se questo è vero, la connessione HTTP all'URL indicato ha inizio.

Una connessione HTTP in Java coinvolge gli oggetti **java.net.URL** e **java.net.HttpUrlConnection**. L'oggetto URL raccoglie tutte le caratteristiche della locazione della risorsa e apre la connessione. In questo caso queste informazioni dipendono dalla variabile requestURI eseguendo le istruzioni:

```
URL url = new URL(requestURI);
```

L'oggetto HttpUrlConnection invece si occupa della gestione degli *stream* coinvolti nella connessione e della costruzione dei parametri della richiesta.

Per fa questo si esegue:

```
HttpURLConnection hc =
   (HttpURLConnection) url.openConnection()
```

con il quale si apre una connessione tramite il metodo openConnection() che ritorna un riferimento alla connessione appena creata cioè ritorna un oggetto di tipo URLConnection che implementa i metodi necessari per interagire col server.

Una volta che la pagina Web è stata scaricata, viene letto il DataInputStream dalla variabile reader e vengono settate altre variabili per prepararle a quella che sarà la risposta da inviare al client.

I dati raccolti fin'ora dal server quindi devono essere resi fruibili al client, cioè trattati in modo tale da fargli credere di essere realmente connesso ad una rete Internet.

Per far questo il server invia una serie di dati serializzati e ben formattati che saranno elaborati e convertiti nel client.

Come ultimo passo i dati convertiti saranno interpretati dal browser permettendo così la navigazione Web all'utente.

5.1.2 MercuryBTClient

È una MIDlet J2ME tipo client che si interfaccia con lo stack Bluetooth del sistema operativo Symbian[11][12]. Il *testing virtuale* di questa MIDlet è stato eseguito sul ***Sun Java Wireless Toolkit for CLDC*** lanciato da ***NetBeans***.

In questo progetto il client BT è una MIDlet che si collega al servizio seriale creato dal server.

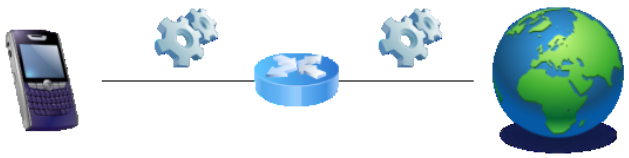

Figura 5.4: Schema di funzionamento del client (proxy)

Funzionalità di MercuryBTClient:

0. Accensione della radio BT e avvio in standby mode;
1. Ricerca del server BT;
2. Raccolta del nome e dell'indirizzo del server;
3. Connessione al server (rete Piconet punto-punto);

4. Ricerca servizi seriali attivi;

5. Connessione ad un servizio seriale RFCOMM (passaggio alla connection mode);

6. Trasferimento dati per la navigazione Web.

Prima che la navigazione Web sia possibile bisogna che vi sia una connessione tra il client-server che permetta il trasferimento dei dati.

La prima azione che MercuryBTClient esegue è la ricerca dei dispositivi utilizzando il metodo startInquiry() oppure il metodo retrieveDevices(), entrambi della classe DiscoveryAgent.

1. **startInquiry()**: è una chiamata non bloccante (cioè ritorna prima del termine della procedura) che da inizio alla ricerca prendendo come argomento due parametri che rappresentano il tipo di *inquiry* da eseguire (*general* o *limited*) ed l'interfaccia DiscoveryListener;

2. **retrieveDevices()**: non esegue una nuova *inquiry* ma si limita a riportare i dati trovati di quei dispositivi già scoperti da *inquiry* precedenti, non fornendo nessuna garanzia sulla loro reale presenza.

Per l'*Inquiry scan* l'interfaccia DiscoveryListener implementa i metodi deviceDiscovered() e inquiryCompleted():

- **deviceDiscovered()**: viene richiamato automaticamente ogni volta che viene scoperto un nuovo dispositivo. A questo metodo viene passato come argomento un'istanza della classe RemoteDevice che rappresenta il dispositivo remoto appena trovato. La classe RemoteDevice permette di raccogliere alcune informazioni molto importanti sul dispositivo remoto come il suo indirizzo Bluetooth o il nome alfanumerico (*user-friendly name*) che lo identifica;

- **inquiryCompleted()**: serve a far capire alla MIDlet che la ricerca di nuovi dispositivi è conclusa.

Una volta che si sono raccolte le informazioni del server, si determinano i servizi che mette a disposizione utilizzando il Service Discovery Protocol definito dalle specifiche Bluetooth.

Per fare questo si richiama il metodo searchServices() che esegue una chiamata non bloccante della classe DiscoveryAgent che prende come argomento:

1. la lista degli UUID che si stanno cercando;

2. la lista dei *service attributes* che devono essere ritornati per ogni *service record* trovato e identificato con uno degli UUID ricercati.

Anche in questo caso occorre implementare l'interfaccia DiscoveryListener (passando il Listener alla funzione searchServices()) che definisce i metodi servicesDiscovered() e serviceSearchCompleted().

L'implementazione delle JSR-82 si preoccupa di:

- richiamare automaticamente il metodo servicesDiscovered() ogni volta che si ricevono dei service records di risposta.
- richiamare automaticamente il metodo serviceSearchCompleted() al termine della ricerca. Tale metodo riporta il modo in cui l'operazione è terminata (ricerca completata correttamente, ricerca terminata dall'utente, ricerca terminata a causa di errori, ecc).

Una volta che la ricerca dei servizi è completata l'utente seleziona da un elenco il servizio seriale offerto dal server e MercuryBTClient e MercuryBTServer possono comunicare attraverso il protocollo Bluetooth.

Per l'invio delle richieste al server, sul client si avvia un nuovo *thread* che apre una *server socket* in *localhost* (all'indirizzo **http://127.0.0.1**) sul dispositivo eseguendo l'istruzione:

```
ServerSocketConnection ssc =
  (ServerSocketConnection) Connector.open("socket://:1234");
```

Attraverso la *socket* si ricevono le richieste del client, che poi verranno analizzate ed eseguite dal server. Tali richieste derivano dalla navigazione Web che l'utente ha sul browser dello smartphone.

Il browser viene aperto in modo automatico dall'istruzione:

```
platformRequest("http://" + stringUrl)
```

dove stringUrl è una variabile tipo stringa che dipende dal settaggio che si ha nel menù ***Impostazioni***.

A questo punto si ha un ciclo while(true), che come in server MercuryBTServer gestisce gli *stream* di input/output. Come nel server, la lettura serializzata dei caratteri andrà a popolare alcune variabili:

- **requestMethod**: metodi GET o POST del protocollo;
- **requestURI**: è la variabile che indica l'url che si vuole raggiungere durante la navigazione Web;
- **requestProtocol**: variabile che definisce il protocollo di trasmissione delle informazioni utilizzato;
- **requestHost**: è l'indirizzo del server Web a cui ci si è connessi.

Dopo che la lettura seriale della pagina Web termina, si chiude lo *stream* di input della *socket* e si inviano al server le informazioni della pagina richiesta.

Le variabili utilizzate in questa fase sono:

- **responseProtocol**: in base al tipo di navigazione che l'utente sta eseguendo, sarà elaborata una GET o una POST;
- **responseCode**: questa variabile indica con delle *linee di status* la risposta del server HTTP da in base all'esito della richiesta fatta dal client. Indica solo il codice numerico.

  Gli status più comuni sono:

  - 200 - richiesta riuscita;
  - 404 - risorsa non trovata.

- **responseMessage**: indica il messaggio delle linee di status (per esempio OK o Page Not Found).

Infine, una volta che tutti i dati vengono spediti al browser, si chiude la connessione alla *socket* e il ciclo può ripetersi.

Capitolo 6

Conclusioni

L'esigenza di una connessione costante ad Internet per un numero sempre crescite di utenti è ormai un dato di fatto. Gli utenti ricorrono a dispositivi mobili come gli smartphone, grazie ai quali navigano quotidianamente nella **rete delle reti** utilizzando connessioni su rete 3G o WiFi.

Nei capitoli di questa tesi si è descritto un sistema software in grado di collegare ad una rete Internet uno smartphone che integra una connettività Bluetooth, realizzando così un *data offloading* del tutto gratuito e funzionante entro certi limiti.

Essendo la connessione ad Internet via Bluetooth una modalità di connessione non standard, presenta alcune problematiche ancora irrisolte.

6.1 Problemi aperti e sviluppi futuri

Il sistema proposto non mira ad essere un'alternativa ad un collegamento Internet standard come quelli illustrati in precedenza, ma piuttosto si pone l'obiettivo di sottolineare il fatto che la tecnologia Bluetooth può servire ad eseguire compiti che esulano dalle normali mansioni, realizzando un vero e proprio *hacking*[16].

> *"La parola **hacking** deriva dal verbo inglese to hack, che significa intaccare. In ambito strettamente informatico, **si può definire l'hacking come l'insieme dei metodi, delle tecniche e delle operazioni volte a conoscere, accedere e modificare un sistema hardware o software**".*

Le limitazioni di questo sistema riguardano:

- le prestazioni riguardanti la velocità di caricamento delle pagine Web;
- la bassa affidabilità del servizio;
- l'impossibilità di caricare pagine superiori a 3 megabyte;

- l'impossibilità di caricare pagine con protocollo https o con sessioni private;
- l'impossibilità di caricare video (il caricamento dei video viene eseguito da *RealPlayer*, un programma esterno che anche se settato correttamente non funziona).

6.2 Testing

Per riuscire a comprendere quelle che sono le limitazioni prestazionali del sistema proposto in questa tesi si è confrontata la velocità di caricamento di alcune pagine Web da parte del browser presente sullo smartphone che implementa MercuryBTClient rispetto alla velocità di caricamento delle stesse pagine da parte di un browser (*Mozilla Firefox*) installato su un PC che impiega un classico collegamento ad Internet.

Il *timer* è stato implementato con una funzione JavaScript la quale permette di stampare a schermo l'intervallo di tempo trascorso dall'inizio del caricamento della singola pagina fino alla sua terminazione.

```
<script language="JavaScript" type="text/javascript">
 <!--
    var then,now=new Date();

    function stopclok(){
      then=new Date();
      alert('Questa pagina ha impiegato '
        +((then-now)/1000)
        +' secondi per essere caricata.');
    }

    window.onload=function(){
      stopclok();
    }
 // End -->
</script>
```

Le pagine utilizzate per il test si sono distinte in pagine aventi solo testo e pagine con testo ed immagini.

Nella Tab. 6.1 si può notare come i tempi di caricamento delle pagine Web sul browser dello smartphone sono più elevati rispetto a quelle su PC in quanto il sistema proposto soffre di alcuni ritardi dovuti alla codifica/decodifica delle informazioni da visualizzare a schermo.

Tabella 6.1: Testing di caricamento di pagine Web eterogenee

Pag.	Contenuto	KB	Loading PC (s)	Loading smartphone (s)
1	solo testo	1	0.026	0.163
2	solo testo	5	0.029	0.276
3	solo testo	10	0.032	0.424
4	solo testo	20	0.035	0.721
5	solo testo	60	0.053	1.863
6	testo + img PNG	5	0.257	1.101
7	testo + img JPEG	5	0.270	3.325
8	testo + img JPEG + img PNG	100	0.879	4.101

Dalla tabella si evince che a parità di peso delle pagine, nel sistema proposto si avvertono dei forti ritardi di caricamento. Tali ritardi si amplificano ulteriormente se nella pagina sono presenti anche delle immagini.

Questi ritardi nel caricamento non sono eliminabili in quanto la codifica e la decodifica delle informazioni (che rappresentano il collo di bottiglia) sono processi intrinsechi al sistema che ne permettono il corretto funzionamento.

Appendice A

Setting di sistema

In questa appendice verranno descritti tutti i passi necessari per configurare correttamente Symbian OS[11] presente sullo smartphone prima dell'installazione o dell'avvio di MercuryBTClient.

A.1 Punto di accesso

Per far accedere MercuryBTClient alle informazioni ricevute/inviate dal browser durante la navigazione Web, si rende necessaria la creazione e la configurazione di un nuovo *punto di accesso*.

Un **punto di accesso** nella sua accezione classica serve ad accedere a reti a pacchetto dei vari **ISP mobili**.

In questo progetto il punto di accesso (rinominato Bt) permette di lavorare sullo smartphone in modalità *localhost*.

Per far questo l'utente deve settare alcuni parametri come l'*indirizzo proxy*, emulando così il funzionamento di un *gateway*.

A.2 Creazione di un nuovo punto di accesso

A.2.1 Setting generale

Per creare un nuovo punto di accesso l'utente deve portarsi sul menù ***Impostazioni***, selezionare ***Connessione*** ed in fine selezionare ***Punti di accesso***.

Figura A.1: Setting delle impostazioni di sistema

A questo punto deve accedere al menù ***Opzioni*** per poter creare e rinominare il nuovo punto di accesso.

La rinomina non è obbligatoria, ma è consigliata in quanto potrebbero essere presenti più punti di accesso che rischiano di far confondere l'utente durante l'utilizzo del sistema.

Figura A.2: Creazione e gestione del punto di accesso 'Bt'

Per quanto riguarda il settaggio generale, non rimane altro che selezionare ***Normale*** alla riga ***Autorizzazione***.

A.2.2 Setting avanzato

Per accedere alla ***Impostazioni Avanzate***, l'utente deve selezionare il menù ***Opzioni*** nella scheda del punto di accesso Bt.

In questa sezione si setteranno i parametri che permetteranno a MercuryBTClient di interfacciarsi con il browser dello smartphone, facendo lavorare questo dispositivo come fosse un *gateway* di rete.

Figura A.3: Menù delle impostazioni avanzate

Alla riga ***Indirizzo servizio proxy*** si indica l'indirizzo *localhost* **127.0.0.1** che permetterà alla MIDlet di accedere alle risorse di rete in locale.

Come ultimo *setting*, l'utente dovrà impostare la porta di accesso della *socket* alla riga ***Numero porta proxy*** inserendo la combinazione di numeri **1234**.

Figura A.4: Setting dell'indirizzo di localhost e del numero di porta proxy

Una volta che tutti i punti descritti in questa appendice sono stati seguiti, la configurazione di Symbian OS è terminata e si può avviare MercuryBTClient.

Appendice B

Installazione e utilizzo del sistema software

Lo scopo di questa appendice è descrivere i passi necessari ad installare e settare correttamente MercuryBTServer e MercuryBTClient.

B.1 Installazione di MercuryBTServer

MercuryBTServer è un'applet *stand-alone*, quindi è un programma eseguibile in modo diretto senza che sia necessaria un'installazione da parte di un programma *installer*.

Per il suo corretto funzionamento necessita di una JRE installata sul PC in uso e di una radio Bluetooth posta in **modalità visibile**.

Il suo avvio si ha utilizzando una *shell di sistema* come ***Terminale*** o simili per GNU/Linux oppure ***COMMAND.exe*** o ***Pront dei comandi*** per MS Windows.

Per l'esecuzione dell'applet l'utente deve portandosi alla cartella dove è presente il file **MercuryBTServer.jar** ed inserire il comando:

```
java -jar MercuryBTServer.jar
```

Ora non rimane altro che avviare il servizio seriale RFCOMM premendo sul bottone ***Start*** che si trova sul *frame* della schermata iniziale.

B.2 Installazione di MercuryBTClient

Per installare MercuryBTClient sullo smartphone impiegato per la navigazione Web è necessario trasferire il file **MercuryBTClient.jar** sulla sua memoria non volatile (memoria interna del dispositivo o sulla *memory stick*) via Bluetooth (in modalità **OBEX**) o via USB.

Se per il trasferimento di questo file si utilizzano sistemi Microsoft, è consigliabile utilizzare il *tool* di gestione dato dal fornitore dello smartphone come ad esempio **Nokia PC Suite** o **Nokia Ovi Suite**. Tali *tools* permettono di navigare tra le cartelle del *file system* dello smartphone ed eseguire operazioni di copia-incolla nella cartella di destinazione desiderata oppure hanno la capacità di avviare l'installazione della MIDlet da remoto.

Se MercuryBTClient.jar è trasferito via BT, viene gestito dal cellulare come fosse stato inviato attraverso un SMS(vedere Fig. B.1).

Figura B.1: MercuryBTClient.jar trasferito via BT

Una volta che il file con estensione *.jar* viene ricevuto, lo si deve selezionare per far partire l'installazione vera e propria.

A questo punto il sistema operativo **Symbian**[11] pone all'utente delle richieste di conferma sulla volontà di proseguire con l'istallazione e sulla sicurezza del software in questione.

Per portare a termine la fase di installazione l'utente deve rispondere affermativamente a tutte le richieste che appaiono sullo schermo (vedere Fig. B.2).

Figura B.2: Installazione di MercuryBTClient

Una volta che MercuryBTClient è installata l'utente deve abilitare e porre in modalità *visibile a tutti* la connettività Bluetooth presente nello smartphone.

Avviando MercuryBTClient appare la schermata iniziale che visualizza la ***Home*** dal quale l'utente può interagire col programma.

Le operazioni selezionabili dalla ***Home*** sono:

1. Ricerca dispositivi;
2. Dispositivi trovati;
3. Impostazioni;

Figura B.3: Menù Home

B.3 Ricerca di nuovi dispositivi BT

Selezionando ***Ricerca dispositivi*** dal menù ***Home***, si richiede l'avvio della funzione di ricerca di nuovi dispositivi Bluetooth che risultano attivi e visibili nel raggio di azione dello smartphone.

Figura B.4: Alert - Messaggio di conferma per la ricerca di nuovi dispositivi BT

Se viene selezionata la ricerca di nuovi dispositivi BT, si visualizzerà a schermo un ***Alert*** tipo ***Confirmation*** (messaggio di conferma) che richiede all'utente se è veramente intenzionato a continuare la procedura di ricerca o desidera tornare al menù ***Home***.

Se l'utente risponde in modo affermativo, la ricerca ha effettivamente inizio richiamando la funzione `startInquiry()`. La procedura di ricerca termina automaticamente dopo circa 30 sec.

Non essendo strettamente necessario attendere la conclusione di questo periodo, in quanto dopo pochi secondi la registrazione dei nuovi dispositivi avviene correttamente, si può decidere di interrompere la ricerca, terminandola in modo manuale selezionando il bottone ***Stop*** (richiamando la funzione `stopInquiry()`.

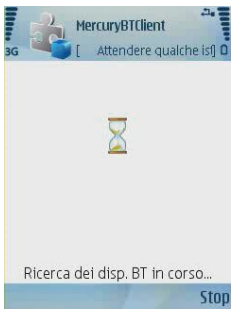

Figura B.5: Ricerca di nuovi disp. BT in corso

Se nel campo d'azione dello smarphone vi sono uno o più dispositivi Bluetooth attivi e visibili, vengono registrati e momentaneamente salvati nella lista ***Dispositivi BT***.

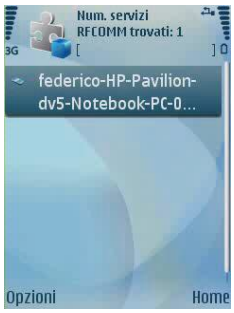

Figura B.6: Lista dei dispositivi Bluetooth trovati

Nel caso in cui non vi siano dispositivi attivi e visibili nell'area di copertura, viene visualizzato a schermo un ***Alert*** tipo ***Alarm*** che richiede all'utente se desidera ripetere la ricerca o si vuole ritornare alla ***Home***.

Figura B.7: Alert - Nessun dispositivo BT trovato nell'area di copertura

B.4 Ricerca dei servizi RFCOMM

Selezionando l'indirizzo Bluettoth del PC su cui è avviato MercuryBTServer, si avvia la ricerca dei servizi RFCOMM attivi sul server.

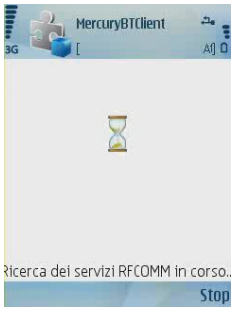

Figura B.8: Schermata di ricerca dei servizi RFCOMM

Se il server è effettivamente attivo viene visualizzato a schermo l'indirizzo del servizio RFCOMM ad esso assegnato come si può vedere nella Fig. B.9.

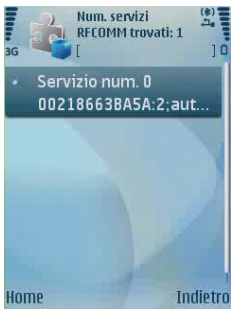

Figura B.9: Lista servizi RFCOMM avviati sul server

Nel caso in cui il server non fosse stato avviato o fosse stato chiuso, terminata la ricerca dei servizi seriali si visualizza un ***Alert*** che permette all'utente di ritentare la ricerca o tornare indietro.

Figura B.10: Alert - Nessun servizio RFCOMM attivo trovato sul server

B.5 Avvio della navigazione Web

Per avviare la navigazione Web l'utente deve selezionare il servizio seriale avviato sul server (riconoscibile da un indirizzo univoco), consentendo cosi a MercuryBTClient e a MercuryBTServer di connettersi, creando un canale per il successivo trasferimento di dati bidirezionale.

Figura B.11: Creazione del canale BT client/server

A questo punto l'App apre la **socket** per mandare e ricevere informazioni dal browser.

La prima volta che questa procedura viene avviata, la MIDlet richiama in automatico il browser preinstallato sullo smartphone, passandogli come parametro l'indirizzo salvato come default nel menù ***Impostazioni*** della MIDlet stessa.

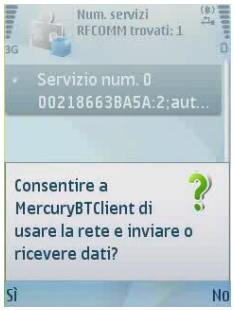

Figura B.12: Richiesta di connessine al servizio RFCOMM selezionato

Giunti alla schermata del browser non rimane altro che acconsentire all'ultima richiesta di sicurezza, permettendo così a MercuryBTClient di aprire uno **stream di dati bidirezionale client/server**, abilitando il trasferimento dei dati che verranno elaborati nel sistema la navigazione Web dallo smartphone.

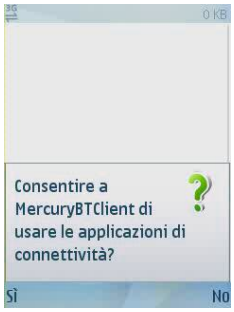

Figura B.13: Richiesta di accesso alla rete da parte del browser

Bibliografia

[1] Standard Bluetooth,
http://www.bluetooth.org

[2] Buetooth SIG (Bluetooth Special Interest Group),
v1.1 2001 edition.

[3] International Standard ISO/IEC 8802-111999(E)
ANSI/IEEE Std 802.11, 1999 edition.

[4] J2ME Documentation,
http://java.sun.com/j2me/docs/index.html

[5] CLDC 1.0 (JSR 30),
http://docs.oracle.com/javame/config/cldc/ref-impl/cldc1.0/jsr030/index.html

CLDC 1.1 (JSR 139),
http://docs.oracle.com/javame/config/cldc/ref-impl/cldc1.1/jsr139/index.html

http://www.oracle.com/technetwork/java/cldc-141990.html

[6] CDC 1.1.2 (JSR 218),
http://docs.oracle.com/javame/config/cdc/ref-impl/cdc1.1.2/jsr218/index.html

http://www.oracle.com/technetwork/java/javame/tech/index-jsp-139293.html

[7] MIDP 1.0 (JSR 37),
http://docs.oracle.com/javame/config/cldc/ref-impl/midp1.0/jsr037/index.html

MIDP 2.0 (JSR 118),
http://docs.oracle.com/javame/config/cldc/ref-impl/midp2.0/jsr118/index.html

[8] Specifica JSR-82,
http://jcp.org/aboutJava/communityprocess/final/jsr082

[9] JSR-82 Bluetooth API and OBEX API,
http://docs.oracle.com/javame/config/cldc/opt-pkgs/api/bluetooth/jsr082/index.html

[10] BlueCove library,
http://sourceforge.net/projects/bluecove

[11] Symbian OS,
http://www.symbian.com

[12] J2ME on Symbian OS,
http://www.developer.nokia.com/Resources/Library/Java/#!index.html

[13] GNU General Public License,
http://www.gnu.org/licenses/old-licenses/gpl-2.0.html

[14] Apache License, Version 2.0,
http://www.apache.org/licenses/LICENSE-2.0

[15] JCP (Java Community Process),
http://jcp.org

[16] Wikipedia Italia,
http://it.wikipedia.org

Wikipedia International,
http://en.wikipedia.org

Ringraziamenti

Vorrei ringraziare tutti coloro che mi hanno aiutato durante la stesura di questa tesi, in particolare la mia famiglia tutta e il mio relatore, il Prof. Alessandro Bogliolo.