Support for Emulation of Services and Applications in Mobile Environments with Bluetooth

Abstract

La crescente diffusione di dispositivi portabili che possono fruire di connettività wireless e l'ampia disponibilità di soluzioni wireless, quali WiFi o Bluetooth, ed i recenti sviluppi delle tecnologie di Mobile Ad-hoc NETwork, aprono uno **scenario innovativo** in cui utenti mobili che condividono obiettivi, interessi o che sono impegnati nelle stesse attività, richiedono di collaborare insieme ovunque si trovino, in qualunque momento ed anche in assenza di una infrastruttura di comunicazione pianificata a priori.
Uno dei principali problemi per lo **sviluppo di applicazioni collaborative** nel nuovo scenario è però la mancanza di supporti in grado di facilitare lo sviluppo ed il testing delle applicazioni: allo stato attuale, i programmatori sono infatti costretti a testare i protocolli applicativi e ad effettuare le loro misurazioni utilizzando complessi strumenti di simulazione o effettuando esperimenti che richiedono la disponibilità di un numero considerevole di host mobili.
BlueSesame è un **supporto per l'emulazione di reti bluetooth** in scenari di mobile ad-hoc network pensato per emulare reti compatibili con lo standard IEEE 802.15.

Introduzione

Il testing di applicazioni in ambienti reali Bluetooth è reso difficile dal fatto che un device mobile è difficile da controllare e a volte interferenze possono perturbare i risultati sperimentali. Inoltre, le difficoltà aumentano se occorre tenere conto anche della mobilità dei nodi, in quanto bisognerebbe monitorare e dirigere movimenti di tutti i nodi della rete considerata.
Per lo studio delle reti bluetooth e per il testing delle applicazioni si fa quindi uso della simulazione.
Possiamo distinguere due tipi di **approcci alla simulazione**: l'emulazione di rete e la simulazione di rete.
I **simulatori** sono dei componenti software che, dato un modello della rete ne simulano il comportamento permettendo di ricavare dei risultati sperimentali.
Gli **emulatori**, invece, effettuano una simulazione usando le applicazioni simili a quelle che saranno usate dagli utenti finali. Le prove avvengono collegando fisicamente i nodi su cui gira l'applicazione ad un nodo centrale che si occupa di emulare la rete, permettendo così di ricavare dei dati sperimentali molto vicini agli scenari reali.
Gli strumenti esistenti, sia nella categoria dei simulatori che in quella degli emulatori, consentono di raccogliere dati sperimentali molto accurati, ma hanno il difetto di essere spesso difficili da configurare ed utilizzare, non fornendo quindi un supporto adeguato allo sviluppo ed alle prime fasi di testing di applicazioni distribuite.
BlueSesame si colloca nella categoria degli emulatori ed ha come obiettivi la semplicità d'uso, la portabilità e la facile integrabilità con le applicazioni da testare. L'emulatore è infatti scritto in Java ed è conforme alle specifiche **JSR-82** per reti Bluetooth, quindi è possibile utilizzare applicazioni reali, del tutto identiche a quelle che saranno usate dagli utenti finali, senza alcuna modifica al codice sorgente. Inoltre la creazione della topologia di rete è facilitata dall'interfaccia grafica attraverso cui è possibile spostare i nodi a piacere e visualizzare le proprietà associate ai nodi presenti.
Nel **primo capitolo** della relazione viene analizzato lo stato dell'arte. Verranno messi a confronto alcuni simulatori ed emulatori esistenti. Nel **secondo capitolo** verranno mostrate le linee guida del progetto, nel **terzo** e nel **quarto capitolo** saranno illustrati modello ed architettura del sistema, infine mostreremo le interazioni dei vari servizi per dimostrare la facilità di sviluppo di applicazioni per ambienti mobili grazie all'uso di BlueSesame. La relazione si conclude con la discussione dei test effettuati e dei possibili sviluppi futuri.

Stato dell'arte: emulatori, simulatori

I simulatori possono essere divisi in due grandi grandi categorie in base al metodo usato per gestire la simulazione: event-driven, time-driven.
Della categoria **event-driven** fanno parte tutti quei simulatori che sono sensibili allo scatenarsi di eventi sulla rete. In base al tipo di evento considerato possiamo suddividerli ulteriormente in:

1. **packet-driven**: in questo tipo di simulazione ogni pacchetto trasmesso sulla rete genera un evento.

2. **fluid**: la trasmissione dei dati è considerata come un flusso descritto dal loro bit rate, i soli eventi considerati sono le variazioni di bit rate di un dato flusso.

Nel caso di simulatori **time driven** la simulazione è tempo discreta e ciò ne facilita la diffusione nelle di architetture parallele: tutti i processori possono sincronizzarsi al termine di ogni time step.
Nel campo della simulazione di reti sono presenti vari tool di simulazione, i più utilizzati sono : OPNET, GloMoSim, QualNet, OMNeT++, NS-2 e WLAN Simulator.
OPNET (Optimized Network Engineering Tool) è un tool commerciale per la modellizzazione e simulazione di reti di comunicazioni, dispositivi e protocolli. E' possibile simulare tutti i tipi di reti cablate e diverse tipologie di reti wireless (implementato il livello MAC di 802.11).
OPNET è completo di una veste grafica e di animazioni, viene utilizzato per simulazioni da varie compagnie di telecomunicazioni ed il suo uso è gratuito solo per le università.
In **GloMoSim** [1] e **QualNet** [2] l'utilizzatore deve avere una buona familiarità con PARSEC per essere in grado di sviluppare nuovi protocolli o per modificare quelli esistenti. Inoltre GloMoSim non è freeware, ma per enti di ricerca, associazioni no-profit e per scopi educativi il suo uso è gratuito. Purtroppo la documentazione è molto ridotta, non esiste un manuale dell'utente ed è presente un esiguo numero di tool disponibili per la generazione di nuove topologie di rete, per il monitoraggio del comportamento del sistema e per l'analisi dei risultati di post-simulazione.
QualNet contiene invece molti modelli e protocolli sia per reti cablate che wireless, una completa documentazione ed il supporto tecnico ma per il suo utilizzo e necessario acquistare una licenza d'uso.
OMNeT++ [3] non è specificatamente creato per la simulazione di reti di telecomunicazioni e non dispone di una valida documentazione del codice che permetta di valutarne la corretta implementazione.
NS2 [4] si autodefinisce come “simulatore di eventi discreti orientato alla rete”, fornisce un supporto per la simulazione del TCP, del routing, e dei protocolli multicast su ambienti wireless. E' un simulatore di tipo “event driven” scritto in C++ e OTCL che implementa:

- Protocolli di “rete”: TCP, UDP, FTP .....
- Protocolli/Algoritmi di Routing
- Meccanismi di Gestione delle Code dei Router (DropTail, CBQ, RED, ...)
- Modelli per Sorgenti di Traffico (HTTP, FTP, ...)
- Meccanismi per Simulazione di “Eventi”.

In generale i simulatori non possono essere usati con protocolli di rete e applicazioni reali ma modificando l’applicazione per l’uso nel simulatore in quanto basano la simulazione su un modello astratto della rete.
Durante lo sviluppo e il testing di applicazioni si producono molte versioni del software che devono essere tutte trasportate dall’ambiente reale a quello simulato. Oltre alle difficoltà che potrebbero sorgere nel passare dall’ambiente reale a quello simulato, la maggior parte dei simulatori non forniscono un punto di vista real time della simulazione. Per ovviare a questi problemi occorre utilizzare dei simulatori di rete real time o emulatori di rete più vicini alla realtà e con un ambiente più controllato, che consentano agli sviluppatori di creare delle topologie di rete che sarebbero difficili da ottenere con dei test nel mondo reale, o che permettano di realizzare dei test real time attraverso vari protocolli.
La maggior parte degli emulatori di rete forniscono la possibilità di fornire ritardi differenti, di simulare la perdita di pacchetti e funzionalità di accodamento per simulare diversi canali di comunicazione e diversi tipi di rete in modo da poter raccogliere dati in diverse condizioni di utilizzo della rete.
Tra gli emulatori esistenti si distinguono NetEmulator ed Itheon NetEmulator ed il recente NIST Net.
NIST Net [5] permette ad un singolo PC equipaggiato con Linux di emulare una grande varietà di reti in diverse condizioni. NIST Net è uno strumento general-purpose per emulare le prestazioni dinamiche in reti IP. E' progettato per consentire esperimenti riproducibili e controllati accompagnati da report con le prestazioni della rete. Operando a livello IP, NIST Net è in grado di emulare le caratteristiche prestazionali imposte da vari scenari di Wide Area Network (ad esempio, la larghezza di banda asimmetrica). NIST Net, tuttavia, non prevede la virtualizzazione del protocollo Bluetooth.
Ci sono quindi tanti simulatori ed emulatori di reti ad hoc, ma non troviamo nessuno tra questi che abbia tutti i nostri requisiti:

- Real-time;
- Interfaccia Wireless/Bluetooth Standard;
- Links unidirezionali;
- Virtualizzazione dei ranges di trasmissione;
- Emulazioni con nodi mobili;
- Logging centralizzato della comunicazione tra i vari nodi.

Gli emulatori esistenti forniscono delle API non standard per la comunicazione: l’applicazione dovrà

essere modificata ad-hoc per l'emulatore portando un allungamento dei tempi di sviluppo e ad una successiva traduzione per poter essere eseguita in ambiente reale.
BlueSesame, a differenza degli emulatori citati, è totalmente compliant sia alle architetture (grazie al linguaggio JAVA) che alle tipologie di rete (WAN o PAN) ed ai protocolli di comunicazione tra i nodi (Wireless o Bluetooth). BlueSesame è uno strumento facile da configurare e che rende immediato lo switch dall'ambiente simulato a quello reale.

Linee guida

Le caratteristiche del nostro sistema dipendono quindi dalle fasi di sviluppo di un applicazione Bluetooth e l'obiettivo finale è fornire un valido supporto per debugging, testing e deployment.
L'utilizzo di un emulatore come BlueSesame entra in gioco nell'implementazione di un sistema che utilizza il protocollo di rete definito. Deve quindi supportare lo sviluppo di applicazioni basate sui protocolli Bluetooth, consentendo:

- il deployment su diversi dispositivi;
- il design facile di protocolli;
- la centralizzazione delle comunicazioni su un unico componente che consenta quindi un facile debugging di ciò che poi accadrà nell'ambiente distribuito;
- l'utilizzo di API che consentano il passaggio immediato all'ambiente reale;
- una facile e familiare coordinazione e protocollo nello sviluppo delle applicazioni.

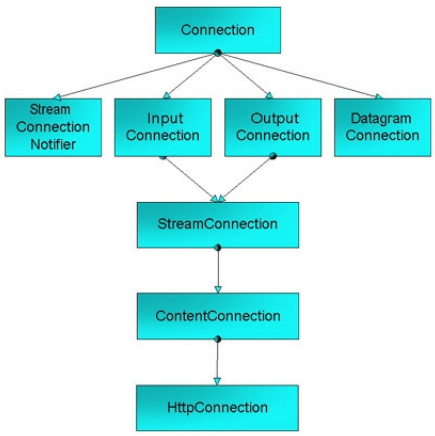
L'obiettivo perseguito è quello di realizzare un sistema di immediata comprensione e facile utilizzo ma che, allo stesso tempo, offra tutte le opzioni necessarie per il testing e debugging, nonché la possibilità di realizzare facilmente demo delle applicazioni eseguite su reti mobili ad hoc emulate.
Le caratteristiche chiave del sistema sono quindi riassunte come segue:

- Facilità d'uso;
- Rappresentazione grafica della rete;
- Simulazione realtime della rete;
- Il codice sviluppato sull'emulatore deve funzionare allo stesso modo su hardware reale senza alcuna modifica;
- Ogni nodo deve avere un range di trasmissione dei messaggi;
- Implementazione della connessione tra due o più dispositivi mobili;
- Supporto alla programmazione JAVA del protocollo bluetooth e delle librerie JSR-82.

Dopo un rapido sviluppo di un primo prototipo semplice simulato, BlueSesame ha quindi l'obiettivo di consentire di produrre nuovi prototipi di complessità sempre crescente tramite emulazione, testare il tutto sui dispositivi reali e passare alla fase di release.

BlueSesame

BlueSesame fa parte della categoria degli emulatori di rete. Si occupa di simulare una data topologia di rete e la mobilità dei nodi, cercando di fornire un supporto computazionalmente leggero e flessibile che consenta di testare rapidamente applicazioni e servizi distribuiti che sfruttano le reti Bluetooth ad-hoc.
BlueSesame, inoltre, si pone come obiettivo la semplicità d'uso relativa sia all'interfaccia utente sia all'integrazione con l'applicazione da testare: è dotato, infatti, di una GUI intuitiva, che consente di creare topologie di rete a piacere e di eseguire dei test anche tenendo conto della mobilità dei nodi, ed è fully compliant con le specifiche Bluetooth JSR-82 permettendo di testare applicazioni reali e di effettuarne direttamente il deploy su dispositivi reali.
BlueSesame integra al suo interno log4j al fine di generare dei file di log con report dettagliati relativi al traffico di rete e agli eventi principali che avvengono durante la simulazione.
BlueSesame deve emulare soprattutto dispositivi come cellulari o PDA che, oltre ad avere risorse limitate, utilizzzano una runtime ed una collezione di API dedicata: la Java Micro Edition. Nella J2ME, diverse tipologie di apparati sono identificate da diversi profili a loro volta riferiti a diverse configurazioni. La configurazione Connected Limited Device Configuration (CLDC), per esempio, include un sottoinsieme minimo di classi Java, ed è utilizzata su dispositivi con scarsissime capacità di calcolo. Fra i profili che operano in configurazione CLDC compare il Mobile Information Device Profile (MIDP), pensato per i cellulari con un sistema di GUI orientato a display a cristalli liquidi e una API di base per giochi in 2D. Molti cellulari moderni vengono forniti con un'implementazione residente dell'MIDP. Il MIDP viene combinato con CLDC, le applicazioni risultanti si chiamano **MIDlet**
Nella configurazione CLDC, data la varietà di dispositivi che possono supportare J2ME e le differenti tipologie di connessione che questi possono avere (porta seriale, USB, rete telefonica cellulare a commutazione di pacchetto o a commutazione di circuito, porta infrarossi, bluetooth, ecc.), l'implementazione dei "meccanismi" di comunicazione non viene effettuata a livello di configurazione



ma viene lasciata ai profili proprio perché più "legati" alle piattaforme host. Il MIDP si occupa quindi dell'implementazione del cosiddetto Generic Connection Framework.
Si tratta di un insieme di interfacce (le stesse previste dalla configurazione CLDC), che rappresentano diversi gradi di astrazione dal particolare tipo di connessione (o di protocollo).
La radice della gerarchia, Connection, costituisce la rappresentazione "più astratta" di connessione stabilendo semplicemente che qualunque tipo di connessione può essere aperta (`open(..)`) oppure chiusa (`close()`).
Per rispondere ai requisiti delle JSR-82 e realizzare un sistema JSR-82 compliant abbiamo incorporato all'interno di BlueSesame **Microemulator** [6], una runtime che permette di avere le funzionalità di J2ME in un sistema J2SE. Grazie a Microemulator, BlueSesaeme è in grado di emulare qualsiasi tipo di dispositivo Bluetooth, dai grandi calcolatori (configurazione CDC) ai piccoli dispositivi con capacità limitate (CLDC).

Requisiti per sistemi Bluetooth

Prima di procedere ad illustrare l'architettura e i componenti che sono stati realizzati è opportuno chiarire quali sono i requisiti richiesti da un infrastruttura di rete di tipo Bluetooth e quali sono le caratteristiche delle applicazioni e dei dispositivi sui quali queste vengono eseguite.

Device Requirements

Lo standard Bluetooth e le specifice della Jsr-82 richiedono che i dispositivi abbiano i seguenti requisiti minimi:
• 512K minimo di memoria totale disponibile per Java 2 Platform
• Hardware di comunicazione Bluetooth con Bluetooth stack. (vedi prossimi paragrafi)
• Implementazione della configurazione **J2ME** Connected Limited Device Configuration (CLDC) o superiore. (vedi prossimi paragrafi)

JSR-82 Requirements

SUN ha rilasciato le specifiche JSR-82 per J2ME. Tali specifiche descrivono le API di comunicazione per lo stack protocollare Bluetooth.
Le specifiche ufficiali della JSR-82 indicano inoltre quali devono essere le caratteristiche di un implementazione delle API:

1. Dipendenza esclusivamente dalle librerie CLDC.
2. Scalabilità – Deve essere runnable su qualsiasi piattaforma Java 2 che fornisce il Generic Connection Framework (GCF), inclusi i profili J2ME.
3. Le API devono permettere l'esecuzione sia di applicazioni server che client.

Bluetooth System Requirements

Un sistema Bluetooth deve avere uno stack protocollare conforme alle specifiche e deve quindi supportare almeno i seguenti layer, su cui sono costruite le API

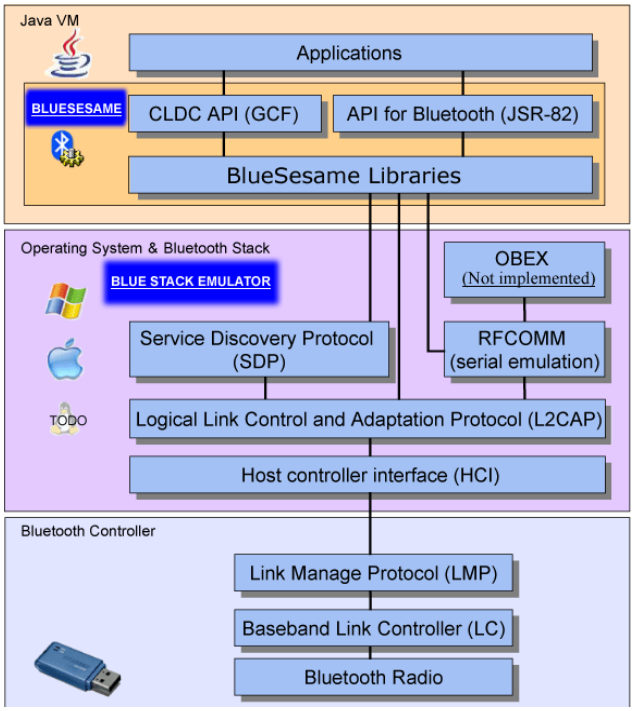
- Service Discovery Protocol (SDP)
- RFCOMM (type 1 device support)
- Logical Link Control and Adaptation Protocol (L2CAP)

Da questa breve analisi si evince che per realizzare un emulatore dalle caratteristiche descritte si ha la necessità di implementare la **JSR-82** in modo che sia compliant con qualsiasi versione della piattaforma **JAVA 2**, inclusa **J2ME**. E' necessario emulare dispositivi CLDC compliant (con i servizi minimi offerti dalla J2ME) e realizzare uno **stack Bluetooth** che agisca, non su hardware reale, bensì sui dispositivi emulati e sulla topologia di rete emulata da BlueSesame.

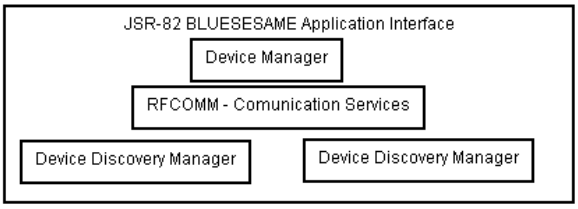
Architettura

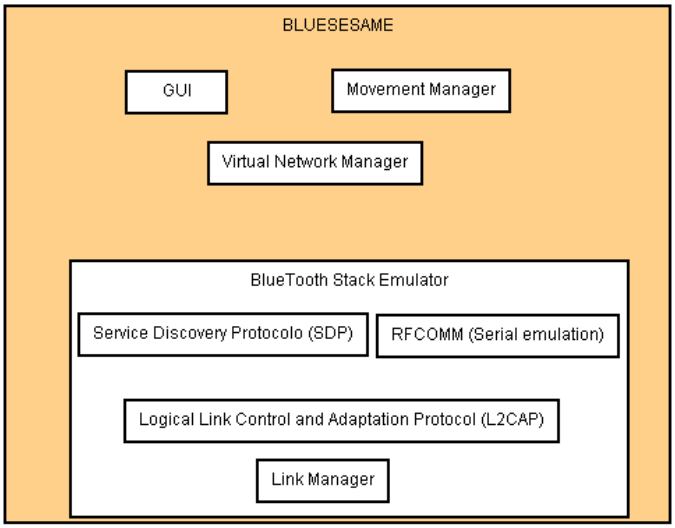
L'architettura di BlueSesame è un estensione di un supporto per reti mobili ad hoc con tecnologia Wireless (IEEE 802.11) chiamato Sesame.
Sesame, realizzato da M. Baldassarro e G. Cocci Grifoni, è un emulatore per le WLAN (Wireless Local Area Network), pensato le fasi di sviluppo di applicazioni e servizi distribuiti che sfruttano le reti WLAN per la comunicazione.
In particolare, Sesame realizza un supporto per il livello applicativo che consente di creare una

MANET, disponendo i nodi a piacere dell'utente, permettendo quindi di testare un'applicazione distribuita in modo semplice e veloce avendo a disposizione sulla stessa macchina più istanze dell'applicazione su JVM distinte.
Estendere Sesame al protocollo Bluetooth significa dunque estendere le caratteristiche del suo core a comunicazioni Bluetooth-compliant, fornendo quindi un middleware che realizzi le librerie della jsr82 e le peculiarità della j2me, secondo l'architettura mostrata in figura.



L'architettura di Sesame è basata sul **modello client/server** in cui i nodi simulati rappresentano i client mentre il server è il nucleo di Sesame dove viene eseguita l'emulazione effettiva della rete.
Tutti i client ed il server girano su JVM distinte e possono essere in esecuzione sulla stessa macchina o su macchine diverse.
La **jsr-82 application interface** ha il compito di rendere possibile l'uso del simulatore alle applicazioni, fornendo un'interfaccia compliant con le specifiche JSR-82 per le reti Bluetooth in modo da facilitare il passaggio della applicazione dall'ambiente reale all'ambiente emulato e viceversa e deve, quindi, essere implementata da tutti i client della rete.
Il server invece si occupa dell'emulazione di rete vera e propria offrendo una GUI intuitiva e una rapida configurazione dell'ambiente di lavoro.
Il server esegue l'emulazione utilizzando i seguenti servizi:
• **virtual network manager**: gestisce la rete virtuale creata dall'utente su cui viene eseguita l'emulazione. Questo servizio si occupa della gestione della topologia della rete e degli indirizzi Bluetooth virtuali: ad ogni nodo su cui gira l'applicazione da testare viene assegnato indirizzo Bluetooth fittizio che verrà usato dai nodi della rete per comunicare. Tale virtualizzazione è necessaria in quanto si vuole rendere possibile l'emulazione di una rete avendo a disposizione più nodi su una stessa macchina e ogni nodo deve essere caratterizzato da un indirizzo diverso.





• **movement manager**: questo servizio si occupa della simulazione del movimento dei nodi utilizzando due modelli di mobilità: random walk model e boundless simulation area mobility model. Durante il movimento dei nodi mantiene coerenti le strutture dei servizi sottostanti;
• **GUI**: è l'interfaccia utente attraverso cui è possibile accedere a tutte le funzionalità del simulatore: è possibile creare la topologia desiderata spostando con il mouse i nodi, è possibile ottenere informazioni relative ad un nodo facendo doppio click sul nodo desiderato o è possibile eseguire gli algoritmi di mobilità implementati.

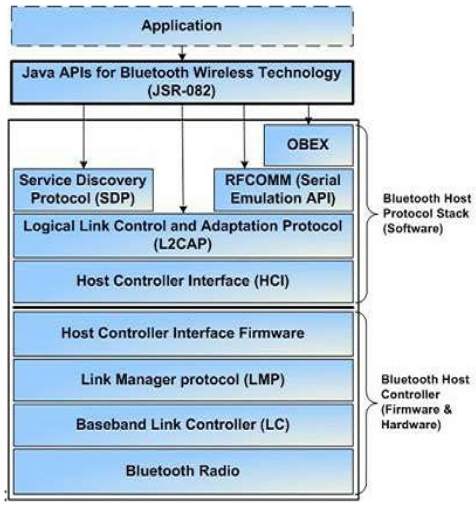
Stack Emulator

• Il *service discovery protocol* (**SDP**) che permette ad un dispositivo Bluetooth di determinare quali sono i servizi che gli altri apparecchi presenti nella picorete mettono a disposizione. Tale protocollo può fungere sia da server (ossia può essere interrogato da un altro dispositivo e rispondere con i propri servizi) sia da client (interrogando gli altri dispositivi) e ogni dispositivo dispone delle informazioni relative ai servizi di cui è capace e dei protocolli supportati: altri apparati potranno fare uso di queste

informazioni per determinare le possibilità di interazione con i nodi della picorete.
• *RFCOMM* è un semplice protocollo di trasport, che fornisce un'emulazione delle porte seriali RS232 sopra il protocollo L2CAP. Il protocollo RFCOMM supporta fino a 60 connessioni simultanee tra due dispositivi Bluetooth, anche se questo numero è specifico all'implementazione di un particolare dispositivo Bluetooth.
Il protocollo L2CAP è posizionato, nello stack dei protocolli Bluetooth, sopra il protocollo Baseband e risiede nel livello data link. L2CAP permette, ai protocolli dei livelli più alti e alle applicazioni, di trasmettere e ricevere pacchetti dati lunghi fino a 64 KB.
• Link Manager: controlla le connessioni tra dispositivi bluetooth attraverso RMI e si occupa dello scambio di ruolo (master-slave) nella comunicazione.
• Virtual network manager: il suo ruolo fondamentale è quello della creazione e gestione delle socket virtuali: l'applicazione si interfaccia con esso per creare ed usare le socket virtuali. Si occupa in fase iniziale di negoziare con il server l'indirizzo virtuale e, durante la vita dell'applicazione, di tenere aggiornato il server sui cambiamenti del nodo.

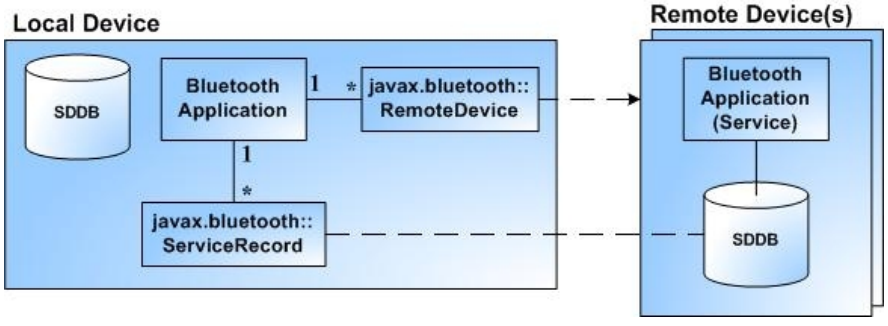
Descrizione componenti

Lo **stack protocollare Bluetooth** è il software che ha accesso diretto ai dispositivi Bluetooth e consiste di più livelli:

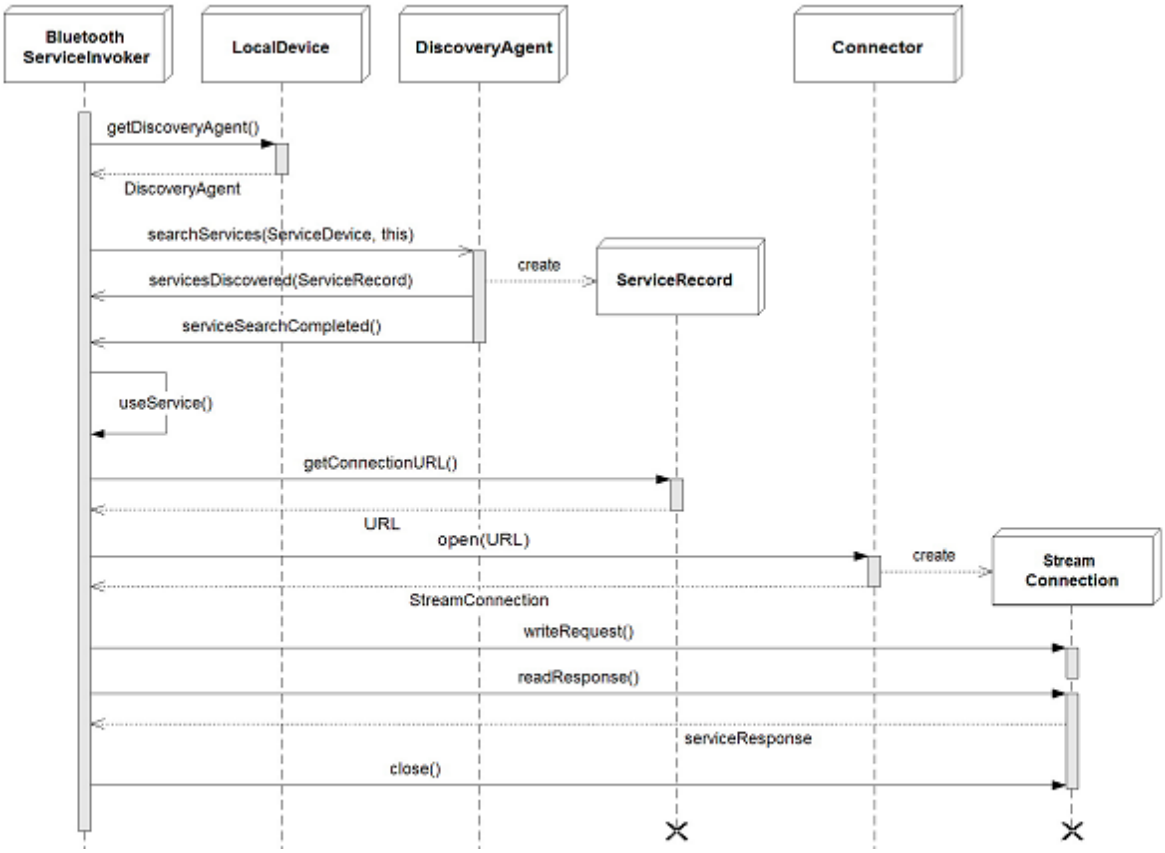


- Link Manager Protocol: raccoglie gli strati più vicini all'hardware e si occupa della creazione, configurazione e del management delle comunicazioni oltre che dell'autenticazione. In pratica associa ad ogni deviceDescriptor un nodo, e per ogni nodo calcola una lista dei nodi raggiungibili a seconda del range;
- RFCOMM: è un semplice protocollo di trasporto basato sullo standard ETSI TS 07.101. Esso definisce dei metodi per emulare una connessione RS-232.
- OBEX: specifica un protocollo che permette lo scambio di oggetti e strutture dati tra dispositivi remoti e non è oggetto del nostro lavoro.
- SDP: è un protocollo per il discovery dei dispositivi al fine di scoprire i servizi disponibili su una connessione Bluetooth e definirne le caratteristiche.

La nostra implementazione rende possibile eseguire tali servizi nella topologia di rete messa a disposizione da BlueSesame. A tal scopo, ad ogni dispositivo è associato un **`DeviceDescriptor`** che contiene tutte le informazioni del dispostivo nella rete, come il Bluetooth address e la potenza del segnale Bluetooth. L'**indirizzo Bluetooth** è naturalmente virtuale e viene creato dallo Stack in modo automatico e la potenza del segnale viene emulata tramite il parametro **range** che rappresenta il raggio di ricezione. Questi parametri sono personalizzabili dall'utente e la loro configurazione permette l'emulazione di dispositivi con caratteristiche differenti.
BlueSesame associa ad ogni `DeviceDescriptor`, e quindi a ogni dispositivo virtuale, un nodo nella topologia di rete.



Tutte le operazioni di **`DeviceDiscovery`** avvengono dunque all'interno della rete di BlueSesame. Una volta ottenuto la lista dei dispositivi raggiungibili è possibile eseguire servizi di service discovery direttamente sui dispositivi emulati che sono nel range di comunicazione.
Ogni dispositivo ha associato un SDDB e fornisce le funzionalità del protocollo SDP.
Una caratteristica essenziale delle tecnologia Bluetooth risiede nella capacità di instaurare una connessione in maniera flessibile e dinamica tra dispositivi che non sono stati precedentemente configurati, quindi qualora un dispositivo client voglia usufruire di un servizio non sa preventivamente se esiste nei paraggi un dispositivo server che possa fornire il servizio richiesto, pertanto, per poter iniziare la connessione, deve necessariamente verificare che vi siano altri dispositivi interlocutori nel proprio raggio d'azione e solo successivamente andrà accertarsi che tra i dispositivi ricercati ve ne sia qualcuno che possa fornire il servizio richiesto.



I passi fondamentali di una **connessione lato client** saranno:

- Inizializzazione del dispositivo Bluetooth locale;
- Ricerca dei dispositivi Bluetooth presenti nei paraggi;
- Ricerca dei servizi presso ciascun dispositivo Bluetooth d'interesse tra quelli scoperti;
- Connessione ad un servizio presso un determinato dispositivo remoto;
- Utilizzo del servizio;
- Disconnessione dal servizio remoto;

Nella prima fase della connessione Bluetooth si procede con l'inizializzazione del dispositivo Bluetooth locale, che consiste nell'impostazione del nome simbolico (friendly name) del dispositivo e della sua visibilità (GIAC o LIAC),

Il secondo passo che viene eseguito è la ricerca dei dispositivi remoti che sono presenti nei paraggi. Tale operazione risulta necessaria ogni volta che un client vuole accedere ad un servizio proprio perché come visto in precedenza esso non conosce quali siano i dispositivi che forniscono il servizio di interesse. Per ogni dispositivo ottenuto come risultato della ricerca, vengono forniti al client una serie di parametri che permettono di identificarlo univocamente. Tali parametri sono:

- Il Bluetooth Address che identifica univocamente il dispositivo;
- l'indirizzo di rete Bluetooth;
- Il friendly name con cui è stato registrato precedentemente il dispositivo nel BCC;

A questo punto, in base ai risultati della ricerca, il client procede con la selezione del dispositivo ed effettua la ricerca dei servizi che il dispositivo scelto offre. La ricerca dei servizi risulta fondamentale perché potrebbe anche accadere che un dispositivo non abbia al momento disponibile il servizio richiesto o che non fornisca un unico servizio.

Ad ogni servizio viene associato un identificativo univoco chiamato **UUID** [7] la cui lunghezza varia tra 16 e 128 bit. Come suggerisce il nome, ogni identificativo garantisce l'univocità del servizio nel tempo e nello spazio. La classe UUID mette a disposizione più costruttori per costruire l'UUID a partire da una stringa o da valori a 16 o 32 bit, un metodo per il confronto tra due UUID e un metodo per convertire l'UUID in stringa. L'UUID è immutabile ed un servizio può essere scoperto solo se identificato da un UUID. Una volta ottenuta la lista dei servizi, il client può selezionare da tale lista il servizio d'interesse e solo a questo punto si procede con instaurazione della connessione tra client e server.

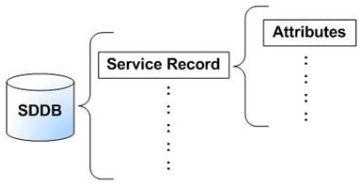
La **connessione lato server** sostanzialmente è simile a quelle lato client se non per la necessità di pubblicare i servizi piuttosto che ricercarli.

I passi in cui viene suddivisa tale connessione sono:

- Inizializzazione del dispositivo Bluetooth locale;

- Creazione del servizio remoto e impostazione dei relativi parametri;
- Aggiunta del servizio Bluetooth creato al database dei servizi Bluetooth pubblicati dal device locale (SDDB);
- Attesa di una connessione entrante da parte di un client;
- Scambio di dati con un client che si è connesso;
- Fine connessione quando il client ha finito di usufruire delle prestazioni del servizio;

La fase di inizializzazione del dispositivo è simile a quella che viene eseguita dal client con l'unica differenza che il dispositivo debba essere visibile al fine di garantirne la visualizzazione da parte dei client. Inizializzato il dispositivo locale bisogna creare il servizio da pubblicare per poi aggiungerlo al Service Discovery Database (SDDB), ovvero il database dei servizi Bluetooth presenti sul dispositivo locale. All'interno del database, per ogni servizio per il quale viene definito un UUID, viene inserito un record e per ogni record vengono definiti una serie di attributi che poi saranno necessari al client per effettuare la scelta.
Dopo aver pubblicato il servizio, il server rimane in attesa di una connessione da parte di un client.
Al momento della connessione inizia lo scambio dei dati. Quando il client si disconnette il server rimane in attesa di un'ulteriore connessione.



Il nucleo della scoperta dei dispositivi è rappresentato dal **Service Discovery Database** (SDDB) e dal Service Discovery Protocol (SDP). L'SDDB è il database che contiene i service record, che rappresentano tutti quei servizi che sono disponibili per i client. Per richiedere i service records, un SDP client su un dispositivo locale fa richiesta ad un SDP server su un altro dispositivo.
Ogni service record è un'istanza della classe `ServiceRecord`. Questo record contiene una serie di attributi che descrivono il servizio più in dettaglio. Per rendere disponibile un servizio ai clienti, un'applicazione server crea un service record instaurando una connessione del tipo `ConnectionNotifier`, successivamente inserisce il service record nell'SDDB invocando il metodo `acceptAndWait` della classe `ConnectionNotifier`.
L'applicazione server può recuperare il service record e aggiornarlo in maniera appropriata, mentre l'applicazione client può interrogare l'SDDB per ottenere i servizi disponibili.
Siccome i dispositivi wireless sono mobili, hanno bisogno di un meccanismo che permetta di trovarne altri e accedere ai servizi da essi offerti: la **discovery dei dispositivi**. Un dispositivo Bluetooth utilizza un oggetto `DiscoveryAgent` per ottenere una lista di dispositivi accessibili invocando metodi che offrono modalità diverse di ricerca. Il metodo `DiscoveryAgent.startInquiry` recupera i dispositivi con un inquiry mode. Per poter utilizzare questa modalità, l'applicazione deve specificare un event listener per immagazzinare i dispositivi trovati in una lista.
Il metodo `DiscoveryListener.deviceDiscovery` viene chiamato ogni volta che un'inquiry trova un dispositivo. Quando l'inquiry è completata o sospesa viene invocato il metodo `DiscoveryListener.inquiryCompleted`. Se il client non vuole attendere che i dispositivi vengano scoperti può invocare il metodo `DiscoveryAgent.retrieveDevices` per ottenere una lista di dispositivi già esistente. In dipendenza dei parametri che vengono passati a questo metodo esso ritorna una lista dei dispositivi ottenuti in una precedente inquiry, o una lista di dispositivi già noti (preknown devices) che il dispositivo locale ottiene contattando il BCC.
Quando un dispositivo locale ha trovato un dispositivo remoto esso può iniziare la **ricerca dei servizi** Bluetooth disponibili utilizzabili dalle applicazioni Bluetooth. Siccome il discovery dei servizi è molto simile a quello dei dispositivi, vengono utilizzati sempre i metodi della classe DiscoveryAgent sul dispositivo server Bluetooth. Prima che un servizio possa essere scoperto è necessario registrarlo nel SDDB. Il server si occupa della creazione del service record che descrive il servizio offerto per poi aggiungerlo all'SDDB, così che possa essere visibile agli altri dispositivi. Per ogni servizio pubblicato vengono definite delle misure di sicurezza. Pubblicato il servizio, il server si pone in attesa di una connessione e aggiorna o elimina gli attributi del servizio qualora fosse necessario.
La scoperta dei servizi inizia con una chiamata al metodo `searchServices`. Continuando con la ricerca dei servizi, il Bluetooth DiscoveryAgent invoca in maniera appropriata i metodi `servicesDiscovered` e `servicesSearchCompleted`.
Oltre al `DiscoveryAgent` e al `DiscoveryListener`, si utilizzano le classi UUID, `ServiceRecord` e `DataElement` durante la scoperta dei servizi.
Una applicazione client Bluetooth deve poter essere in grado di scoprire altri dispositivi che siano nelle sue vicinanze. Questo processo viene definito **device discovery**. L'applicazione client potrà poi interrogare ciascun device per verificare quali sono i servizi offerti. Le classi necessarie per poter effettuare il discovery si trovano nel package `javax.bluetooth`.

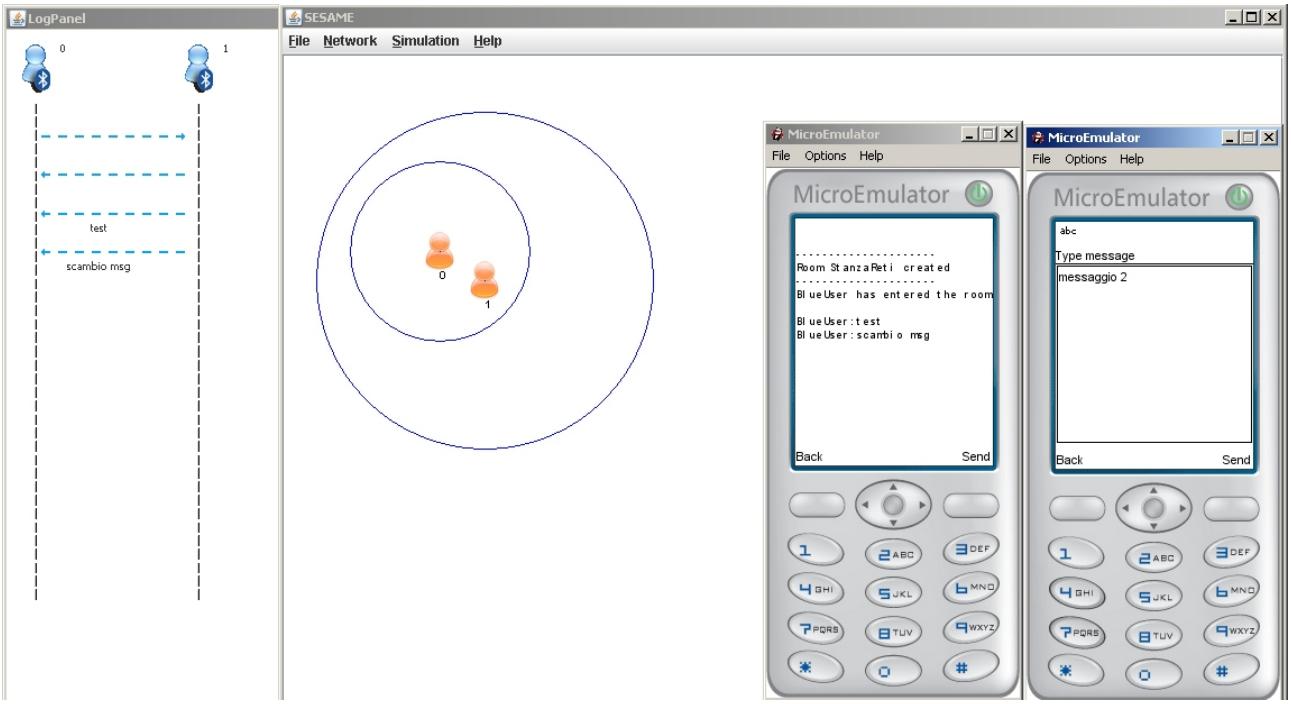
Per eseguire un device inquiry bisogna eseguire i seguenti passi:
• ottenere un DiscoveryAgent;
• implementare il DiscoveryListener interface;
• utilizzare il DiscoveryAgent per iniziare la ricerca, fornendo il listener che riceverà la notifica dei dispositivi scoperti.
Un server Bluetooth deve pubblicare i dettagli dei **servizi offerti** al fine di permettere ai potenziali client di trovarli ed utilizzarli. Per fare ciò, un server deve creare un service record per il servizio fornito. Il service record è memorizzato nel Service Discovery Database (SDDB) del server. Quest'ultimo deve anche poter aggiornare il service record se il servizio disponibile cambia, e rimuoverlo dal SDDB quando il servizio non è più disponibile.
Un service record descrive le caratteristiche del servizio e fornisce le informazioni necessarie per accedervi. E' strutturato da un insieme di attributi. Ciascun attributo ha:

- ID , è un 16 bit unsigned integer;
- valore di tipo `javax.bluetooth.DataElement`.

Per la creazione di un nuovo service record, il server invoca il metodo `javax.microedition.io.Connector.open`. In questo modo viene creato un notifier che il client potrà utilizzare per connettersi al server. A tale metodo viene passato una String `URL` che include tutte le informazioni necessarie alla creazione del servizio.
Il service record è pubblicato quando il server indica che è pronto ad accettare connessioni al servizio. Ciò viene fatto invocando sul notifier il metodo `acceptAndOpen()`. Il service record rimarrà memorizzato nel SDDB fino a quando il server invocherà il metodo `close()`.

Screenshots

Abbiamo sviluppato una piccola demo che implementa una semplice chat tra dispositivi mobili, in cui un telefono cellulare crea una stanza a cui si possono collegare tutti i telefoni cellulari nel range per poter scambiare messaggi.
Ecco qui lo screenshot che mostra, da un lato il server BlueSesame con il suo pannello di log centralizzato della comunicazione, dall'altro il dispositivo mobile bluetooth con lo scambio di messaggi avviato.



Conclusioni

BlueSesame modella le reti mobili ad hoc con protocollo bluetooth, trascurando tutta una serie di peculiarità che suggeriamo come ulteriori sviluppi.
Tra le innovazioni apportabili all'emulatore, il **consumo di energia** rappresenta un fattore importante su cui diverse aziende stanno investendo in maniera consistente. In un emulatore come BlueSesame, tale

caratteristica sarebbe tanto utile quanto facile da implementare.
E' altresì interessante concentrare l'attenzione sulla **modellazione del range di discovery dei dispositivi** che, allo stato attuale, è rappresentato da un cerchio perfetto. Potrebbe essere sviluppato un sistema che aggiunga l'influenza di perturbazioni della banda fornendo, in tal modo, una rappresentazione più realistica dello spazio.
La tecnologia Bluetooth costituisce parte fondamentale nello sviluppo del VoIP. Ad oggi viene già impiegata nei microfoni usati come estensioni wireless dei sistemi audio dei cellulari e dei PC. In questo senso sarebbe utile inserire la **trasmissione di flussi multimediali** [8].
In un'ottica riguardante l'emulazione di reti bluetooth in scenari di mobile ad-hoc network, ogni innovazione che avvicini BlueSesame ad una modellazione più efficace della realtà concorre a migliorarne la fruibilità rendendo il nostro emulatore uno strumento sempre più indispensabile per lo sviluppo di applicazioni su reti compatibili con lo standard IEEE 802.15.

Bibliografia

[1] http://pcl.cs.ucla.edu/projects/glomosim/
[2] http://www.soe.ucsc.edu/~michael/manual/introduction.html
[3] http://www.omnetpp.org/
[4] http://www.isi.edu/nsnam/ns/
[5] http://snad.ncsl.nist.gov/nistnet/
[6] http://www.microemu.org/
[7] http://avetana-gmbh.de/avetana-gmbh/produkte/doc/javax/bluetooth/UUID.html
[8] http://medien.informatik.uni-ulm.de/~frank/research/hicss36.pdf