Università degli Studi di Padova
Dipartimento di Ingegneria dell'Informazione

Corso di laurea magistrale in Ingegneria Informatica

Progettazione di un sistema di ausilio alla didattica frontale per la condivisione immediata di contenuti multimediali

---

Relatore: **Prof. Carlo Ferrari**

Studente: **Andrea Fregnan**
Matricola: **1058318**

Anno Accademico: 2013 - 2014

L'unione di cose semplici porta alla realizzazione
di qualcosa di immensamente grande.
Basta guardare la musica: sette note e l'infinito.

Dedicato alla mia famiglia e ai miei amici.

Indice

Elenco delle figure

Elenco delle tabelle

Abstract

Nel presente lavoro viene presentato un sistema di ausilio alla didattica frontale per la condivisione immediata di contenuti multimediali. In particolare, partendo dalla commissione del problema da parte di un ente esterno, vengono affrontate innanzitutto delle scelte decisionali che fanno parte della fase di analisi delle specifiche tecniche. Si studiano infatti in questa fase i vari protocolli di servizio, come lo standard Bluetooth, che serviranno nel livello successivo. Il progetto si evolve nella fase di progettazione logico-funzionale nella quale si sviluppa la parte di lavoro implementativo e di realizzazione dell'applicazione (versione desktop e mobile) per il *commitment* richiesto. Si affrontano quindi problemi di sicurezza, di affidabilità e di funzionamento del sistema, in quanto l'ambiente didattico in cui il sistema verrà utilizzato presenta delle casistiche particolari. Infine si arriva ad una fase di testing e release nella quale vengono discussi anche dei possibili miglioramenti, riportando per completezza delle sezioni teoriche che affrontano eventuali perfezionamenti.

In this paper is presented a system of aid to classroom teaching for instant sharing of multimedia content. In particular, starting from the Committee of the problem by an external entity, first decision-make are addressed, thay are part of the analysis phase of the specification, more in detail it comes to technical specifications or of the type of system to achieve. In fact, at this stage I study the various service protocols (such as the Bluetooth standard) which will be used in the next level. The project evolves in the design phase in which the logical-functional presents the work of an implementation and realization of the application for the commitment required. Then problems of safety and reliability of the system are faced, since the learning environment in which the system will be used with the cases presents details that are solved brilliantly. Finally, I arrive at a testing phase and release which are also discussed possible improvements, reporting for completeness the theoretical sections that address the improvment.

Capitolo 1

Introduzione

L'idea che vuole prendere forma nel presente lavoro di tesi è fondamentalmente quella di collegare gli studenti presenti in un'aula didattica in una rete ad hoc per permettere loro di interagire e comunicare tramite lo scambio di file multimediali come gli appunti della lezione in corso oppure tramite lo scambio di messaggi utilizzando i propri dispositivi informatici come laptop pc, tablet e smartphone. Oltretutto si intende anche specializzare il sistema per l'utilizzo di quest'ultimo da parte del docente che tiene la lezione. In particolare, per la figura del professore, verranno attivate delle funzionalità software aggiuntive come per esempio l'invio in broadcast degli appunti della lezione a tutti gli utenti presenti in aula. L'applicazione rientra nella categoria dei sistemi *ubiqui*, i quali verranno approfonditi nel corso della stesura e, proprio per la necessità della creazione di reti ad hoc per ogni aula didattica della struttura, verranno discusse delle possibili soluzioni di progettazione. L'utilizzo geograficamente localizzato del sistema porterà alla scelta di utilizzo dello standard bluetooth per il collegamento dei dispositivi presenti. Si approfondisce teoricamente il funzionamento di tale protocollo ed in particolare verranno affrontate le interfacce di applicazione per la programmazione in linguaggio Java. La scelta di utilizzo del servizio Bluetooth genera di conseguenza delle problematiche legate alla sicurezza e alla autenticazione dei device che sono affrontate a livello software tramite l'utilizzo di un server web remoto di autenticazione. Un'altra funzionalità che l'applicazione mette a disposizione è un'interfaccia con un servizio di repository cloud (nel caso in esame il servizio cloud è dropbox). In questo modo tutte le risorse prodotte nell'ambiente didattico possono essere preservate per utilizzi futuri. Viene sviluppata innanzitutto un'applicazione desktop e vengono poste le basi per l'applicazione mobile su piattaforme Android. Il linguaggio di programmazione utilizzato è Java in quanto specificatamente progettato per essere il più possibile indipendente dalla piattaforma di esecuzione. Infine vengono

discussi dei possibili miglioramenti sia a livello applicativo, ovvero di estensione delle funzionalità software, e sia a livello di struttura dei nodi della rete ad hoc.

1.1 Commitment

Realizzare una struttura software per i processi didattici condivisi che assolva i seguenti punti:

- Realizzazione di un network ad hoc che deve essere abilitato ad un utilizzo localizzato (aula scolastica) da parte degli utenti presenti (studenti e docenti)

- assolva di conseguenza i problemi di autenticazione e sicurezza generati dal punto 1

- l'infrastruttura non deve richiedere, per sua natura, un setup da parte del client ma, al contrario, si comporta come un sistema ubiquo.

- permetta l'interfacciamento con i repository per la fruizione di risorse didattiche in real time come paper, lucidi ecc. La pervasività deve permettere, eventualmente, il dialogo con il sistema di e-learning in modo da caricare il materiale prodotto in loco, ad esempio gli appunti dello studente, non appena la lezione è conclusa.

- Rendere la didattica bidirezionale:

  - quando viene utilizzato un articolo accademico o una risorsa didattica da parte del professore, il sistema ne deve permettere l'invio in *broadcast* agli studenti.
  - permetta di svolgere un learning by doing proponendo esercitazioni multimediali nel contesto didattico. Più precisamente permetta agli utenti l'interazione di gruppo, sempre mantenendo la supervisione del docente.

Nella presente tesi si affronta il problema commissionato da un punto di vista ingegneristico, cercando quindi di avere una visione ampia dello scenario di lavoro, utilizzando discipline dell'informatica potenzialmente utili alla progettazione del sistema quali la gestione di sistemi distribuiti, reti ad hoc, teledidattica e protocolli di comunicazione.

1.2 Didattica Condivisa

Nella didattica condivisa, un componente base è la piattaforma tecnologica (Learning Management System o LMS) che gestisce la distribuzione e la fruizione della formazione: si tratta infatti di un sistema gestionale che permette di tracciare la frequenza ai corsi e le attività formative dell'utente (accesso ai contenuti, tempo di fruizione, risultati dei momenti valutativi,...). Tutte le informazioni sui corsi e gli utenti restano indicizzate nel database della piattaforma: questa caratteristica permette all'utente di accedere alla propria offerta formativa effettivamente da qualsiasi computer collegato a Internet, generalmente senza la necessità di scaricare software ad hoc dal lato del client, e a volte perfino senza necessariamente consentire attraverso il proprio browser il deposito e la memorizzazione di cookies. L'utente è insomma in questo caso totalmente delocalizzato e in virtù di ciò più semplice risulta il suo accesso al proprio percorso formativo modellizzato sul server, anywhere/anytime, ovunque e in qualsiasi momento. Se la piattaforma risulta essere una componente fondamentale per la teledidattica, l'aula virtuale (o ambiente collaborativo) è la metodologia didattica che permette l'interazione (soprattutto in modalità sincrona) fra gli utenti: si tratta infatti di strumenti che favoriscono la comunicazione immediata tramite chat, lavagne condivise (interactive whiteboards), videoconferenza e così via. I software di ambiente collaborativo possono gestire anche l'apprendimento asincrono (che non necessita la presenza degli utenti nello stesso momento): forum di discussione, document repository, accesso ai materiali didattici o a materiali di supporto.

Figura 1.1: Rappresentazione grafica del concetto di didattica condivisa

In particolare, vengono elencate le strutture che intervengono in modo rilevante nello svolgimento del processo, suddividendole in:

- cliente: chi commissiona gli interventi formativi;
- learning company: l'azienda che risolve con diverse soluzioni i bisogni del cliente;
- fornitore di contenuti: la struttura che fornisce i contenuti oggetto del percorso di apprendimento;
- multimedia agency: l'azienda che collabora con la learning company alla progettazione del corso multimediale e ne realizza concretamente la grafica e il software;
- tester: le aziende che collaborano con le learning company alla verifica tecnica dei prodotti finiti.

In questo caso si può collassare l'entità di learning company, multimedia agency e tester in un'unica entità.

Capitolo 2

Concetti generali

2.1 Ubiquitous computing

Ubiquitous computing è un concetto di ingegneria del software e dell'informatica in cui l'idea di computazione è fatta per apparire ovunque e dovunque. A differenza della *desktop computing*, l'ubiquitous computing può verificarsi con qualsiasi dispositivo, in qualsiasi luogo e in qualsiasi formato. Un utente interagisce con il computer, che può esistere in molte forme diverse, tra cui computer portatili, tablet e terminali in oggetti di uso quotidiano, come un frigorifero o un paio di occhiali. Le tecnologie di base per supportare l'ubiquitous computing includono Internet, *middleware* avanzato, sistema operativo, mobile code, sensori, microprocessori, nuove interfacce utente di input/output, le reti, i protocolli di telefonia mobile, localizzazione e posizionamento.

Questo nuovo paradigma è anche descritto come pervasive computing, intelligenza ambientale, o "*everyware*". Ogni termine enfatizza leggermente diversi aspetti. Per quanto riguarda gli oggetti coinvolti, è anche conosciuto come physical computing, Internet of Things ed informatica tattile. Piuttosto che proporre una definizione unica per ubiquitous computing e per questi termini correlati, è stata proposta una tassonomia di proprietà per ubiquitous computing, da cui possono essere descritti diversi tipi di sistemi e applicazioni onnipresenti.

Ubiquitous computing tocca una vasta gamma di temi di ricerca, tra cui calcolo distribuito, mobile computing, reti mobili, context-aware computing, reti di sensori, interazione uomo-computer, e l'intelligenza artificiale.

2.2 Architettura multi-tier

Per la progettazione del sistema ci si orienta fin da subito ad una architettura **multi livello**. Nell'ingegneria del software, il termine architettura multi-tier (dall'inglese multi-tier architecture, un'architettura multi-strato, spesso definita come n-tier architecture) indica un'architettura software in cui le varie funzionalità del software sono logicamente separate ovvero suddivise su più strati o livelli software differenti in comunicazione tra loro (nel caso di applicazioni web questi strati sono la logica di presentazione, l'elaborazione dei processi e la gestione della persistenza dei dati).
Ciascuno strato è in comunicazione diretta con quelli adiacenti ovvero richiede ed offre servizi allo strato adiacente in maniera concettualmente simile a quanto accade con le architetture di rete a strati (in linguaggio strettamente informatico si dice che ciascuno strato è client-server per gli strati adiacenti, fatta eccezione per gli strati estremi che sono solo client o solo server). Ad esempio, un'applicazione che utilizza il **middleware** per gestire le richieste di dati tra un utente e un database, utilizza un'architettura multi-tier. In generale i vari strati possono risiedere sulla stessa macchina oppure su macchine elaboratrici differenti mappando così il relativo sistema informatico che ospita, a livello infrastrutturale, l'applicazione. Negli anni novanta si diffuse l'architettura client-server a due livelli, mentre negli anni 2000 l'impiego più diffuso di un'architettura multi-tier è l'architettura a tre livelli.
L'architettura delle applicazioni N-tier fornisce un modello per gli sviluppatori per creare vantaggiosamente un'applicazione flessibile e riutilizzabile ovvero *scalabile*. Con la separazione di un'applicazione in livelli, per modificare o aggiungere funzionalità, gli sviluppatori, possono infatti modificare solo uno specifico livello, piuttosto che dover riscrivere l'intera applicazione, garantendo dunque una maggiore semplicità di progettazione/implementazione secondo la filosofia del divide et impera ed una maggiore manutenibilità.
I concetti di strato e livello sono spesso usati come sinonimi. Tuttavia un punto di vista abbastanza comune è che uno strato è un meccanismo di strutturazione logica per gli elementi che compongono la soluzione software, mentre un livello è un meccanismo di strutturazione fisica per le infrastrutture di sistema.

Con il termine *middleware* menzionato precedentemente, si intende un insieme di programmi informatici che fungono da intermediari tra diverse applicazioni e componenti software. Sono spesso utilizzati come supporto per sistemi distribuiti complessi.

2.3 Architettura Software

L' architettura software è l'organizzazione fondamentale di un sistema, definita dai suoi componenti, dalle relazioni reciproche tra i componenti e con l'ambiente, e i principi che ne governano la progettazione e l'evoluzione. Questa definizione deriva dallo standard IEEE 1471-2000. Al pari delle architetture hardware, descrivere l'architettura software di una sistema significa elencarne le sottoparti costituenti ed illustrarne i rapporti interfunzionali. Più precisamente, l'architettura software include l'insieme delle decisioni significative sull'organizzazione di un sistema software.

Nella fase di progettazione del problema vengono proposte tre soluzioni praticabili. Da notare in particolare, che l'ambito di utilizzo di questo sistema è principalmente in ambito accademico universitario. In questo contesto vengono utilizzati pesantemente strumenti tecnologici come tablet, netbook e smartphone per la didattica. Questi ultimi si appoggiano solitamente ad un struttura di rete wireless già predisposta dall'ente didattico che prevede una procedura di autenticazione e successivamente di navigazione e utilizzo. L'obiettivo del presente lavoro è quello sfruttare tale sistema di rete già esistente per implementare un servizio pervasivo che permetta l'interazione degli utenti all'interno delle aule che mira ad aumentare la realtà didattica. Nelle sezioni successive verrà spiegata passo per passo la strategia modulare di soluzione del problema.

2.4 Programmazione ad eventi

La **programmazione a eventi** è un paradigma di programmazione dell'informatica. Mentre in un programma tradizionale l'esecuzione delle istruzioni segue percorsi fissi, che si ramificano soltanto in punti ben determinati predefiniti dal programmatore, nei programmi scritti utilizzando la tecnica a eventi il flusso del programma è largamente determinato dal verificarsi di eventi esterni. Gli eventi esterni a cui il programma deve reagire possono essere rilevati mediante *polling* (interrogazione) eseguito all'interno di un loop di programma, oppure in risposta ad un interrupt. In molte applicazioni si usa una combinazione di entrambe queste due tecniche. I programmi che utilizzano la programmazione a eventi (denominati spesso programmi "*event-driven*") sono composti tipicamente da diversi brevi sotto-programmi, chiamati gestori degli eventi (event handlers), che sono eseguiti in risposta agli eventi esterni, e da un *dispatcher*, che effettua materialmente la chiamata, spesso utilizzando una coda degli eventi che contiene l'elenco degli eventi già verificatisi, ma non ancora processati. In molti casi i gestori degli eventi

possono, al loro interno, innescare ("*trigger*") altri eventi, producendo una cascata di eventi. Un sistema, o programma, controllato da comandi può considerarsi un caso particolare di sistema o programma event-driven, in cui il sistema, normalmente inattivo, aspetta che si verifichi un evento molto particolare, cioè l'invio di un comando da parte dell'utente.

2.5 Ad Hoc Network

2.5.1 Introduzione alle Ad-Hoc Network

Il successo dell'architettura stratificata di Internet ha favorito l'adozione del modello a livelli anche per le reti wireless e mobile, così come per le ad-hoc network. Questo ha anche favorito lo scetticismo verso approcci alternativi. Tuttavia, un disegno *strick-layered* non è flessibile abbastanza per far fronte alle dinamiche introdotte dalle reti mobili e può prevenire molte classi di ottimizzazione delle prestazioni.
Internet connette in modo trasparente milioni di devices eterogenei, supportando una vasta varietà di comunicazioni. Da un punto di vista del networking, la sua popolarità è dovuta al suo core design che ne ha fatto il modello estensibile e robusto contro il continuo utilizzo così come contro le failure. Allo stato dell'arte, le comunicazioni wireless e i dispositivi mobili spingono la visione del networking ad essere senza una rete (ad-hoc networking). Questo porta nuovi problemi da affrontare ed un attento design dell'architettura per i protocolli di queste reti viene reso necessario per incorporare queste tecnologie emergenti. L'architettura dei protocolli divisi per layer di internet e le responsabilità della rete, dividono il sistema di networking in componenti modulari, e permettono il perfezionamento trasparente dei singoli moduli. In un sistema strict-layered, i protocolli sono indipendenti l'uno dall'altro e interagiscono attraverso interfacce ben definite: ogni implementazione di un layer dipende dall'interfaccia disponibile dal layer più basso e da quella esportata al livello più alto. Lo strict-layering fornisce flessibilità all'architettura del sistema: le estensioni apportate ad un singolo livello non vanno ad influenzare il resto del sistema. La separazione dei concetti porta l'aggiunta di benefici nella minimizzazione del costo di sviluppo tramite il riutilizzo del codice esistente. Questo approccio di design si affida alla comunicazione "orizzontale" tra i livelli di peer protocol sui device del ricevente e mittente. Diversi aspetti dell'architettura di internet hanno portato all'adozione dell'approccio strict-layer anche per le reti mobili e le reti ad-hoc. Alcuni di questi aspetti includono

- la visione "IP centric" delle ad hoc network;

- la flessibilità offerta dai livelli indipendenti, che permette il riutilizzo del software esistente.

La scelta dell'approccio a livelli è supportata dal fatto che le ad hoc networks sono considerate come estensione mobile di internet, e quindi lo *stack protocollare* deve essere adatto. Tuttavia, questo principio progettuale si scontra con i seguenti fatti:

- i problemi come la gestione energetica, la sicurezza, la cooperazione caratterizzano l'intero stack e non possono essere risolti all'interno di un singolo livello.
- le ad hoc network e internet hanno vincoli in conflitto; e mentre le prime sono dinamiche, quest'ultima è relativamente statica.

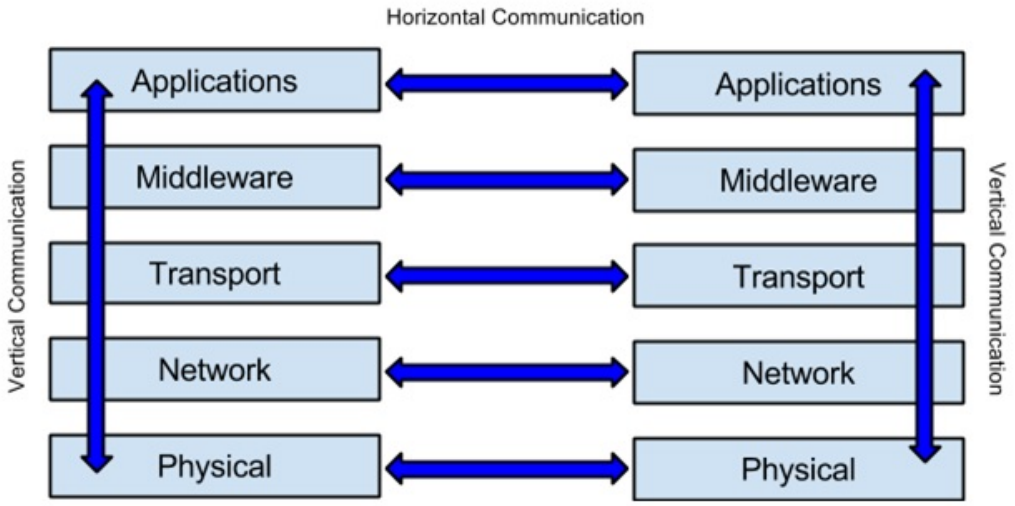

Figura 2.1: Comunicazione orizzontale e verticale tra i livelli

Alcune linee guida per approcciare questi problemi puntano a rafforzare le comunicazioni verticali nello stack dei protocolli, come un modo di ridurre le comunicazioni orizzontali tra i peer, e quindi conservare la banda. Le comunicazioni verticali, specialmente tra livelli non adiacenti, facilitano il reperimento di dati locali, altrimenti effettuato attraverso comunicazioni di rete. La pratica di accedere non solo al prossimo livello inferiore, ma anche agli altri livelli dello stack, portano al *cross-layering*. Il lato negativo dello strict-layeriing è che ostacola l'estensibilità: un nuovo componente a livello superiore può essere costruito solo su ciò che è fornito dal livello immediatamente inferiore. Quindi, se un livello necessita l'accesso ad una funzionalità

o informazione fornite da un livello non adiacente, deve essere progettata un'estensione intermedia . Il cross-layering permette ai livelli non adiacenti di interagire direttamente, rendendo possibile interamente l'ottimizzazione e permettendo l'estensione quando necessario.

2.5.2 Background sulle Mobile Ad Hoc Networks

Sin dal 1990 l'uso e la domanda per le reti wireless mobile ha continuato a crescere. Questo è largamente dovuto alla crescente popolarità dei telefoni cellulari. La crescita, in parallelo, della popolarità di Internet ha scatenato nuovi interessi nel fornire applicazioni Internet-type su reti wireless per mobile. Tradizionalmente, queste ultime possono essere classificate in due grandi categorie: con infrastruttura e senza infrastruttura. Le reti con infrastruttura sono coordinate da un controller centralizzato (anche chiamato stazione base o access point), che dirige il flusso del traffico da e per un nodo end-user. La rete senza infrastruttura (anche conosciuta come ad hoc) è fatta solamente di nodi end-users. Questo significa che tutti i nodi nella rete sono capaci di trasmettere, ricevere ed instradare i dati a differenti nodi nella loro rete senza utilizzare i servizi della stazione base (utilizzata per esempio dalle reti cellulari). Nelle ad hoc netowrks, i nodi possono essere statici o mobili, oppure un ibrido tra le due configurazioni. Le ad hoc network che possiedono nodi mobili sono spesso denotate come mobile ad hoc netorks (*MANETs*). Nelle MANETs, ogni nodo è caratterizzato dal suo range di trasmissione, che è limitato dalla potenza di trasmissione, attenuazione, interferenza, e dalla topologia del luogo. Comunicazioni dirette possono avvenire tra due nodi intermedi se esse sono vicini l'uno all'altro nel range di trasmissione. Comunicazioni indirette possono essere stabilite determinando un percorso (route) tra un numero di nodi intermedi tra la sorgente e la destinazione. Per esempio in figura 2.2, i nodi A e B possono comunicare direttamente in quanto sono nello stesso raggio di trasmissione, mentre il nodo C e il nodo A devono stabilire un route attraverso il nodo B per essere abilitati a comunicare.

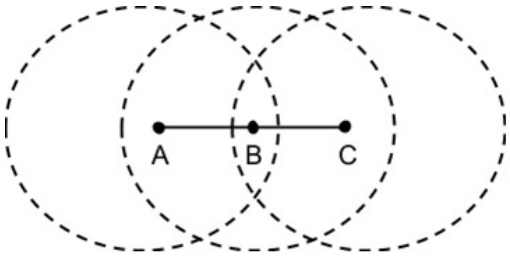

Figura 2.2: Visibilità tra i nodi nella comunicazione wireless

2.5.3 Applicazioni

Le MANET sono utili in ambiente di rete dinamici dove la topologia della rete cambia continuamente. Sono anche utili in aree dove l'infrastruttura di networking non può essere facilmente implementata. Alcune applicazioni tipiche di queste reti includono:

- coordinamento di operazioni militari
- operazioni di soccorso nei disastri
- conferenze
- networking di sensori
- networking di veicoli
- Personal Area Network (PAN)

In particolare l'ultimo punto vede nelle MANETs una applicazione per fornire comunicazione tra piccoli dispositivi in network geograficamente limitate con un ambiente dinamico di networking. Per esempio, un numero di persone in un centro commerciale utilizzano piccoli dispositivi come dei *personal digital assistants* (PDAs) che permettono loro di interagire utilizzando appunto una MANET. Quest'ultima può essere formata con i dispositivi PDA equipaggiati con tecnologia Bluetooth.

2.5.4 Sfide

L'applicabilità delle MANET ad una varietà di differenti applicazioni hanno attirato l'interesse di molte organizzazioni come grandi aziende, governi, ed università. Proprio questo interesse ha fatto sì che tale topic diventasse un'area di ricerca molto vasta negli ultimi anni. La ricerca in campo Mobile Ad Hoc Network spazia attraverso tutti i livelli del modello TCP/IP, dal momento in cui questa tipologia di rete richiede la riprogettazione di ogni livello per fornire efficienza nella comunicazione end-to-end. Inoltre, prima che le MANETs venissero utilizzate con successo negli scenari scritti precedentemente, deve essere affrontato un numero non indifferente di problemi critici:

- **banda**: la capacità delle reti wireless è significativamente inferiore della capacità delle reti cablate. Le operazioni di route discovery e update possono causare problemi significativi di banda in rapporto alla dimensione o densità della crescita del network.

- **end-to-end delay**: i pacchetti dati che viaggiano tra due nodi possono incontrare lunghi ritardi. Questo può essere dovuto al ricalcolo del percorso come risultato di collegamenti rotti, code per ottenere l'accesso al mezzo trasmissivo dovute alla contesa del canale, ritardi nel processing ad ogni nodo, e colli di bottiglia nel traffico di rete.

- **energia (power)**: ogni nodo mobile deve essere fornito di una fonte di energia mobile (come una batteria). Aggiornamenti periodici dei percorsi di route, beaconing, e la trasmissione di dati possono consumare quantità significative di energia, questo può richiedere ad ogni nodo di ricaricare frequentemente la batteria. Ciò significa che ogni nodo deve minimizzare la mole di processi per preservare la batteria, ovvero sono richiesti e/o desiderabili dei protocolli "leggeri".

- **Quality of Service (QoS)**: ogni nodo nella rete può trasmettere differenti tipi di informazioni con differenti livelli di importanza e priorità. Quindi, le risorse disponibili nella rete devono essere distribuite in modo tale che ogni utente ottenga differenti livelli di accesso in accordo al livello di servizio richiesto. La natura dinamica della MANETs, sulle quali le risorse limitate variano nel tempo, rendono la fornitura di QoS una sfida aperta.

- **Scalabilità**: dal momento in cui la densità della rete, in termini di numero di nodi e quantità di traffico, cresce, l'efficienza del routing e la trasmissione dei dati inizia a risentirne. Questo a causa della competitività per l'accesso alle risorse come la banda e il canale wireless. Tale fenomeno può causare problemi come perdita di pacchetti, lunghi ritardi e creazione di colli di bottiglia.

Capitolo 3

Il progetto

3.1 Soluzioni proposte in fase decisionale

3.1.1 Prima soluzione

L'architettura presentata in questa soluzione prevede l'installazione fisica di Server all'interno delle aule didattiche. È quindi necessario che si instauri una connessione wireless o cablata da parte degli utilizzatori del servizio del sistema didattico multimediale con il server di aula fisico. È poi quest'ultimo che avrà il compito di gestire le operazioni di interfacciamento con l'esterno (per esempio con i servizi di Cloud, eLearning, cartelle di repository) e tra gli utenti presenti . Tale soluzione, tuttavia, non è praticamente adottabile appunto per la necessità di installare postazioni in ogni aula didattica: questo sarebbe uno spreco significativo di risorse economiche e fisiche (dovute ad esempio al cablaggio dell'hardware infrastrutturale).

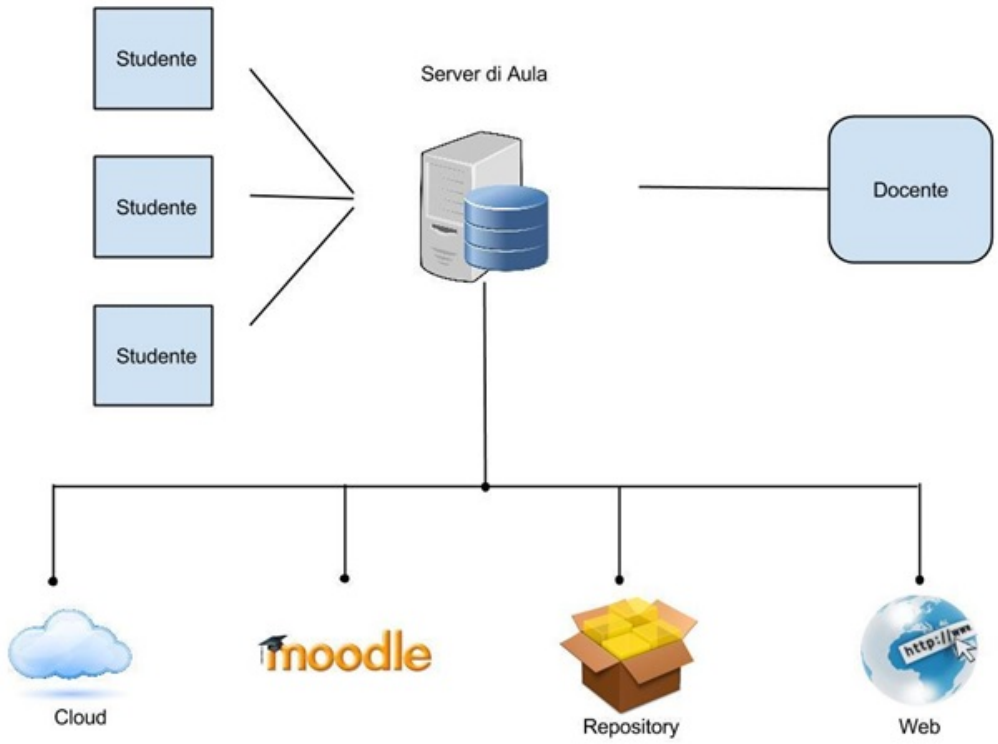

Figura 3.1: Prima soluzione proposta in fase decisionale

3.1.2 Seconda soluzione

L'evoluzione architetturale della prima soluzione proposta consiste nel virtualizzare il server di aula. In ambito informatico, con virtual private server (comunemente VPS) ci si riferisce ad una singola istanza di un sistema che viene eseguito in ambiente virtuale. Più VPS possono essere eseguiti contemporaneamente sullo stesso hardware (host computer) e, in base alle diverse implementazioni dell'*hypervisor*, possono essere migrati (in alcuni casi anche senza interruzione del servizio) su un differente host. Questa tecnica consente di ospitare più sistemi in esecuzione, anche con sistemi operativi differenti, sullo stesso hardware potendo risparmiare sui costi delle infrastrutture. Nelle principali implementazioni di virtualizzazione i VPS sono eseguiti in assoluta indipendenza: i processi di un contenitore non sono accessibili per gli altri, così pure per il file system. A seconda dell'implementazione dell'hypervisor e delle politiche impostate i singoli VPS possono disporre di quantità di risorse (CPU, memoria RAM, spazio sul disco, velocità di comunicazione su ethernet) fisse o variabili, o impiegare un sistema intermedio con quote di differenti livelli (periodo di basso utilizzo del sistema, periodo di sovraccarico).

Secondo la definizione, quindi, si tende a risparmiare in termini economici per quanto riguarda l'infrastruttura hardware, spostando la visione del problema ad un livello più alto, ovvero software. Il server virtualizzato ha il compito di gestire internamente le reti e le "*session*" che si creano all'interno delle aule didattiche. È chiaro comunque che la potenza di calcolo richiesta è mediamente alta. Guardando la soluzione proposta dal punto di vista degli utilizzatori del sistema, questi ultimi usufruiscono dell'infrastruttura di network messa a disposizione dall'università, quindi delegando l'aspetto di autenticazione dell'utente alla rete già esistente.

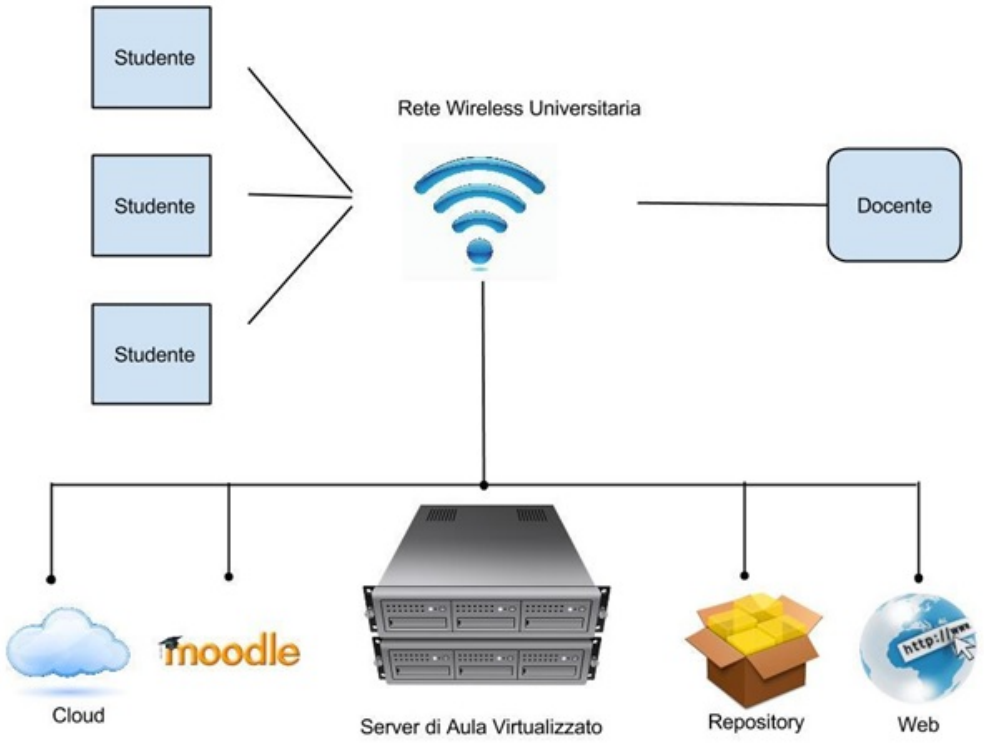

Figura 3.2: Seconda soluzione proposta in fase decisionale

3.1.3 Terza soluzione

La soluzione proposta precedentemente riuscirebbe a soddisfare il commitment richiesto in modo discretamente efficace. L'affinamento della soluzione due consiste nel raggruppare gli utenti di un'aula didattica in una rete ad-hoc creata dal sistema. Tale rete si collega al resto dell'architettura tramite la rete wireless messa a disposizione dell'università. Il modello da adottare potrebbe prevedere semplicemente la creazioni di network wireless caratterizzati

da diversi SSID [1] che referenziano le diverse aule e alle quali gli utilizzatori richiedono di collegarsi per partecipare alla session didattico-interattiva. Si può anche pensare di sfruttare un altro tipo di tecnologia per orientarsi maggiormente verso una visione localizzata e real time della didattica. A questo punto della progettazione ci si pone il problema di come costruire le reti ad-hoc all'interno delle aule. Ci sono varie soluzioni adottabili, tuttavia, se si pensa alla realtà geograficamente localizzata nella quale si sta operando (aula didattica), si intuisce che la tecnologia che più si presta allo scenario è la tecnologia Bluetooth. Chiaramente essa non risolve totalmente il problema in quanto bisognerà affrontare le complicazioni dovute all'interferenza e autenticazione di host. Verrà eventualmente discussa successivamente la topologia e tipologia di rete da adottare, per la costruzione delle ad-hoc network, finalizzata a rappresentare le relazioni di connettività, fisica o logica, tra gli elementi costituenti la rete stessa.

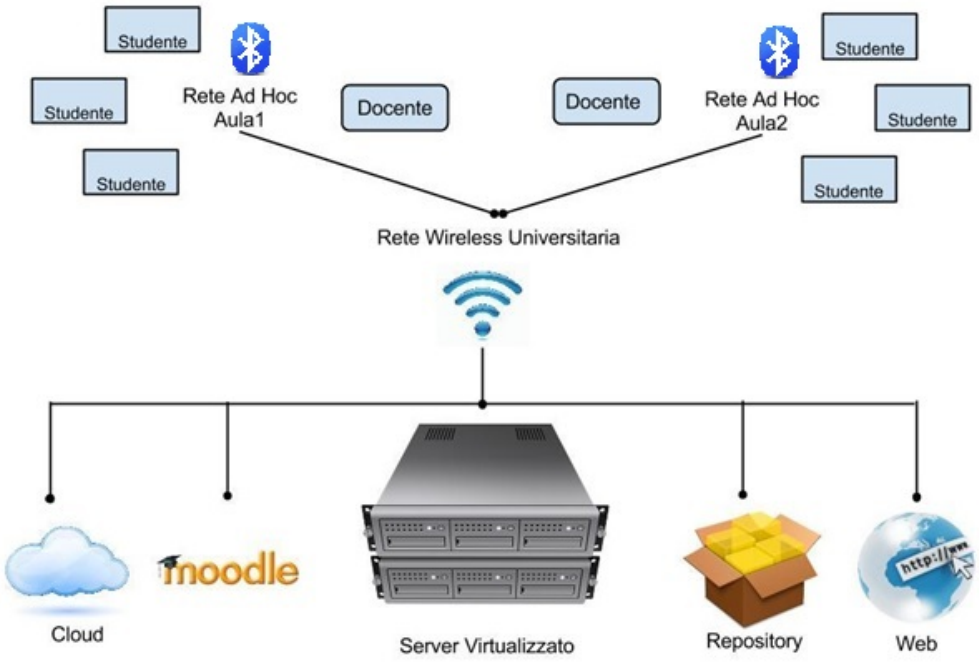

Figura 3.3: Terza soluzione proposta in fase decisionale

[1] Service Set IDentifier è il nome con cui una rete Wi-Fi o in generale una WLAN si identifica ai suoi utenti

3.2 Bluetooth

Nelle telecomunicazioni il **Bluetooth** [1] è uno standard tecnico-industriale di trasmissione dati per reti personali senza fili (WPAN: Wireless Personal Area Network). Fornisce un metodo standard, economico e sicuro per scambiare informazioni tra dispositivi diversi attraverso una frequenza radio sicura a corto raggio. Bluetooth (a volte abbreviato in BT) cerca i dispositivi coperti dal segnale radio entro un raggio di qualche decina di metri mettendoli in comunicazione tra loro. Questi dispositivi possono essere ad esempio palmari, telefoni cellulari, personal computer, portatili, stampanti, fotocamere digitali, console per videogiochi purché provvisti delle specifiche hardware e software richieste dallo standard stesso.

I collegamenti che possono essere stabiliti tra i diversi dispositivi sono di due tipi: orientati alla connessione e senza connessione. Un collegamento orientato alla connessione richiede di stabilire un collegamento tra i dispositivi prima di inviare i dati; mentre, un link senza connessione non richiede alcun collegamento prima di inviare i pacchetti. Il trasmettitore può in qualsiasi momento iniziare ad inviare i propri pacchetti purché conosca l'indirizzo del destinatario. La tecnologia Bluetooth definisce due tipi di collegamenti a supporto delle applicazioni voce e trasferimento dati: un servizio asincrono senza connessione (ACL, *Asynchronous ConnectionLess*) ed un servizio sincrono orientato alla connessione (SCO, *Synchronous Connection Oriented*).

3.3 Piconet

Due o più dispositivi collegati tra loro tramite standard Bluetooth formano una **piconet**. I dispositivi all'interno di una piconet possono essere di due tipi: master o slave. Il master è il dispositivo che all'interno di una piconet si occupa di tutto ciò che concerne la sincronizzazione del clock degli altri dispositivi (slave) e la sequenza dei salti di frequenza. Gli slave sono unità della piconet sincronizzate al clock del master e al canale di frequenza. Le specifiche Bluetooth prevedono 3 tipi di topologie: punto-punto, punto-multipunto e scatternet. Diverse piconet possono essere collegate tra loro in una topologia chiamata *scatternet*. Gli slave possono appartenere a più piconet contemporaneamente, mentre il master di una piconet può al massimo essere lo slave di un'altra. Il limite di tutto ciò sta nel fatto che all'aumentare del numero di piconet aumentano anche il numero di collisioni dei pacchetti e di conseguenza degradano le prestazioni del collegamento. Ogni piconet lavora indipendentemente dalle altre sia a livello di clock che a livello di salti di frequenza. Questo perché ogni piconet ha un proprio master. Un dispositivo

Bluetooth può partecipare sequenzialmente a diverse piconet come slave attraverso l'uso di tecniche TDM (*Time Division Multiplexing*), ma può essere master solo in una.

Capitolo 4

Lo standard Bluetooth in Java

Viene approfondito in questo capitolo lo standard Bluetooth, in particolare vengono presentate le caratteristiche tecniche e progettuali che verranno poi utilizzate nella fase di realizzazione del sistema.
Sviluppata da Bluetooth Interest Group, la tecnologia Bluetooth consiste di: tecnologia radio, uno stack protocollare, profili interoperabili. La tecnologia radio bluetooth è basata sulla banda di frequenze 2.45 Ghz Industriale, Scientifica, Medica, che non è coperta da licenza e disponibile globalmente. Quando i devices bluetooth si connettono tra di loro, formano una piconet, una rete dinamica creata che comprende il master device e fino a sette device slave. Lo standard bluetooth supporta connessione tra piconets: quando un master di una piconet diventa lo slave di un'altra, esso fornisce un bridge tra loro.

I profili di interoperabilità Bluetooth, descrivono l'interoperabilità cross-platform e i requisiti di coerenza. Essi includono il *Generic Access Profile* che definisce le funzionalità per il management dei device, il Service Discover Application Profile che definisce gli aspetti del servizio di scoperta, e il Serial Port Profile che stabilisce i requisiti di interoperabilità e le capacità per l'emulazione del cavo seriale.

Lo stack bluetooth comprende uno stack software [2] che si interfaccia alle Application Program Interface (API) prima del livello applicazione:

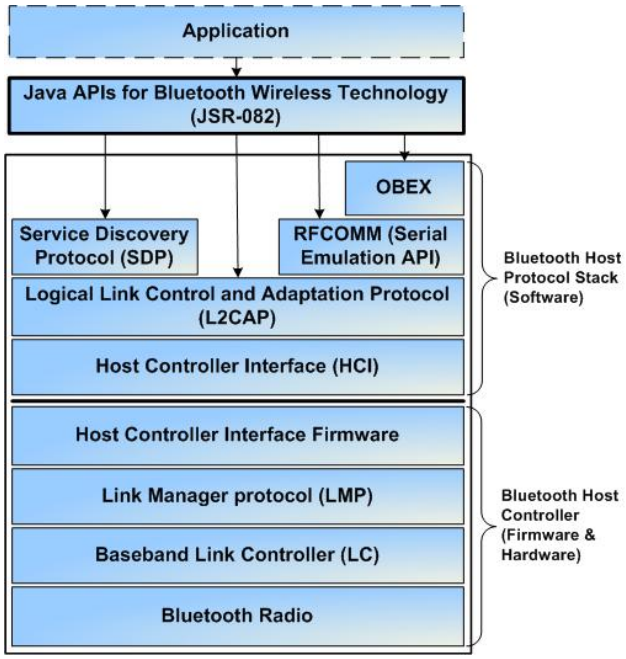

Figura 4.1: Stack Bluetooth

JSR 82 (nome del pacchetto di API per lo standard Blueooth) espone agli sviluppatori che operano con la piattaforma Java lo stack software Bluetooth. Di interesse speciale sono il Service Discovery Protocol (SDP), il Serial Port Profile RFCOMM [1] per l'emulazione seriale e il Logical Link Control and Adaptation Profile (L2CAP), che fornisce dei servizi dati orientati alla connessione ai livelli superiori nel protocollo.

4.1 Caso di utilizzo tipico di un'applicazione Bluetooth-Enabled

Un'applicazione Bluetooth-enabled può essere un server o un client - un produttore di servizi o un consumatore - oppure può comportarsi come un vero e proprio endpoint peer-to-peer esponendo sia servizi client sia servizi server. La figura 4.2 illustra un caso di utilizzo delle specifiche Bluetooth in un'applicazione:

[1] Radio frequency communication, fornisce un data stream semplice all'utente ed è usato da molti dispostivi mobili come livello di trasporto per il servizio OBEX su Bluetooth.

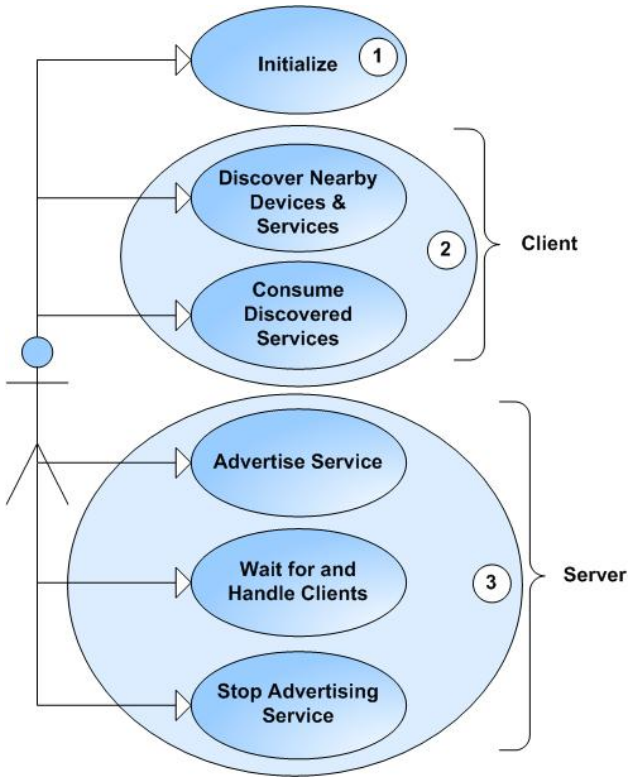

Figura 4.2: Caso di utilizzo delle specifiche Bluetooth

I casi di utilizzo possono essere riassunti nei seguenti:

- Inizializzazione: tutte le applicazioni bluetooth-enabled devono per prima cosa inizializzare lo stack Bluetooth.

- Client: un client "consuma" servizi remoti. Esso scopre innanzitutto i device vicini, e successivamente per ogni device scoperto ricerca i servizi di interesse.

- Server: un server rende i servizi disponibili ai client. Tali servizi vengono registrati nel Service Discovery Database (SDDB). Successivamente attende per le connessioni in entrata, le accetta e serve i client. Infine, quando il servizio non è più richiesto l'applicazione lo rimuove dal SDDB.

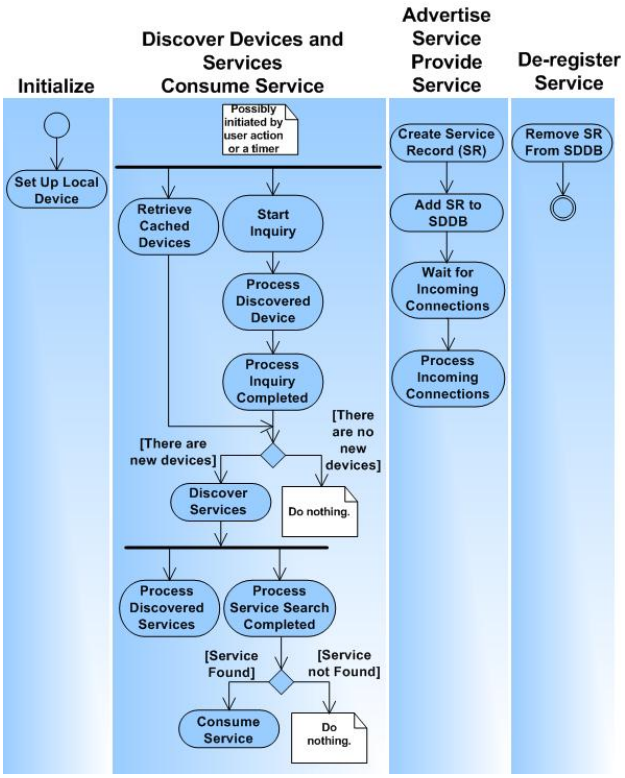

Figura 4.3: Controllo di flusso nelle fasi di applicazione.

La Figura 4.4 illustra gli elementi di una tipica applicazione Bluetooth-enabled, nel nostro caso una MIDlet [2]

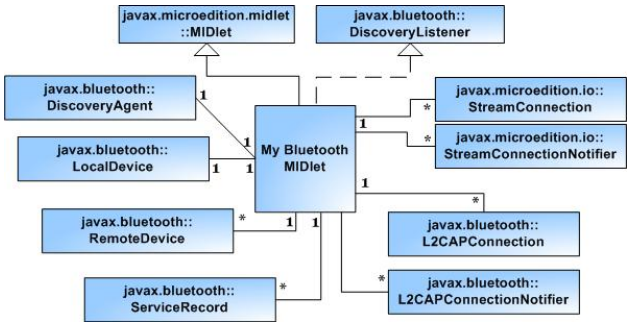

Figura 4.4: Elementi di una MIDlet.

Al centro è collocata la My Bluetooth MIDlet, l'applicazione core che estende javax.microedition.midlet.MIDlet. Il resto delle classi o interfacce

[2]Una MIDlet JAVA è un'applicazione creata per sistemi embedded, specificamente per i sistemi con installata una J2ME Virtual Machine.

che l'applicazione usa sono incluse per gli utilizzi Bluetooth-specific, come il discovering di devices e servizi, la connessione e il consumo di servizi e per pubblicizzare e fornire servizi. É una buona pratica l'utilizzo del design pattern Model-View-Controller, che scompone l'applicazione in interfaccia utente (views), comportamento (controller), e dati (modello), e in più nel nostro caso sostiene classi e interfacce come le API Bluetooth. É anche importante in fase di design separare il client e il server in classi differenti in modo da poter riutilizzare i componenti in momenti successivi.

4.2 Overview sul Core Java API per lo standard Bluetooth

Si può suddividere il core delle API Java per il Bluetooth, racchiuso nel namespace javax.bluetooth, in tre categorie che analizzeremo in seguito: discovery, device management e communication.

4.2.1 Bluetooth Discovery APIs

Le applicazioni client utilizzano le APIs di discovery JABWT [3] per ricercare devices e servizi vicini. La classe DiscoveryAgent supporta il discovery di entrambi device e servizi. Le applicazioni client che vogliono essere avvisate quando vengono scoperti i dispositivi e i servizi devono implementare e registrare l'interfaccia DiscoveryListener che definisce le callbacks per le notifiche di device e service discovery. La relazione tra un'applicazione client bluetooth e il DiscoveryAgent è tipicamente uno-a-uno.

4.2.2 APIs per il Device Discovery

Si può utilizzare il metodo DiscoveryAgent per il device discovery al fine di inizializzare e cancellare un'operazione di scoperta:

- retrieveDevices(): reperisce devices già scoperti o conosciuti che sono vicini.
- startInquiry(): inizia l'operazione di discovery di devices vicini, chiamata anche Inquiry.

[3]Java APIs for Bluetooth Wireless Technology, è una specifica J2ME per API che permettono l'esecuzione di applicazioni java MIDlets su device come mobile phones per l'utilizzo del bluetooth.

- cancelInquiry(): cancella ogni operazione di inquiry attualmente in progresso.

Il Bluetooth Discovery Agent invoca il DiscoveryListener del device e metodi di callback [4] discovery in differenti punti della fase di Inquiry:

- deviceDiscovered() indica se un device è stato scoperto.
- inquiryCompleted() indica se l'inquiry ha successo, ha innescato un errore o è stata cancellata.

Il diagramma di stato in Figura 4.5 illustra gli stati di device-discovery raggiunti come risultato delle callbacks DiscoveryListener.

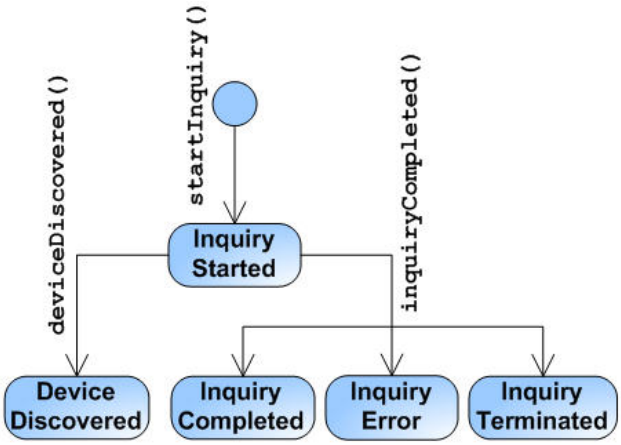

Figura 4.5: Stati di device discovery raggiunti con le callbacks.

4.2.3 API per Service Discovery

Si possono utilizzare i servizi della classe DiscoveryAgent che servono per iniziare o interrompere un'operazione di discovery:

- selectService(): selezione un servizio.
- searchServices(): inizia l'operazione di discovery dei servizi esposti dai device.
- cancelServiceSearch(): cancella ogni operazione di service discovery in corso.

[4] In programmazione, un callback è, in genere, una funzione, o un blocco di codice che viene passata come parametro ad un'altra funzione.

Il Bluetooth Discovery Agent invoca dei metodi di callback per la scoperta di servizi in punti differenti nella fase di service-discovery:

- servicesDiscovered() indica se sono stati scoperti dei servizi.

- serviceSearchCompleted() indica che l'operazione di service discovery è stata completata.

La figura 4.6 illustra gli stati di service discovery raggiunti come risultato delle callback DiscoveryListener:

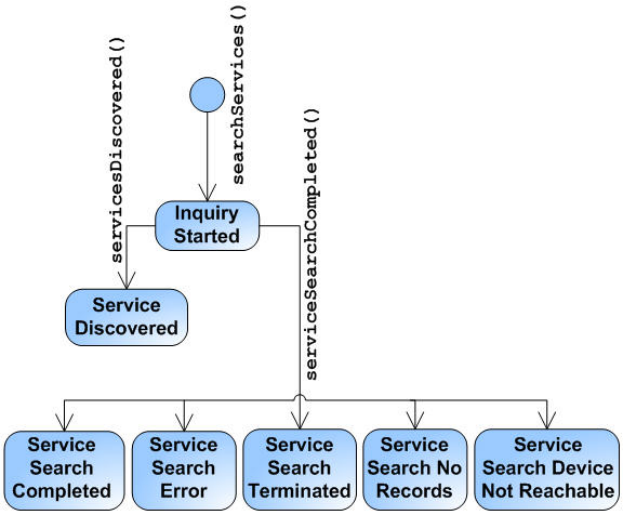

Figura 4.6: Stati di service discovery raggiunti con le callbacks.

Il servizio di discovery inizia con una chiamata al metodo searchServices(). Mentre il servizio di ricerca continua la sua esecuzione, il Bluetooth Discovery Agent chiama i metodi servicesDiscovered() e serviceSearchCompleted(). In aggiunta al DiscoveryAgent e al DiscoveryListener, si possono utilizzare le classi UUID, ServiceRecord e DataElement.

4.2.4 La classe UUID

Nel Bluetooth, ogni servizio ed attributo è identificato da un identificatore unico universale (UUID). Come suggerisce il nome, per ognuno di questi identificatori è garantita l'unicità nel tempo e nello spazio. La classe UUID può rappresentare UUIDs corti (16 o 32 bit) e lunghi (128 bit). Essa è provvista di un costruttore per creare un UUID da una Stringa o da un valore a 16-32 bit, un metodo per confrontare due UUID e un metodo per

convertire un UUID in una stringa. Le istanze UUID sono immutabili e solo i servizi identificati da uno UUID possono essere scoperti.

Si può generare un valore UUID con il comando: *uuidgen -t* sotto Linux e *uuidgen* sotto Windows. Uno UUID apparirà nel seguente modo: 2d266186-01fb-47c2-8d9f-10b8ec891363.

4.2.5 L'interfaccia ServiceRecord e SDDB

Al centro del servizio di discovery ci sono il Service Discovery Database (SDDB) e il Service Discovery Protocol (SDP). Il SDDB è un database mantenuto dall'implementazione Bluetooth che contiene i service records, i quali rappresentano i servizi disponibili ai client. L'SDP è utilizzato per il service discovery. Per ottenere un service record, un SDP client sul device locale fa una richiesta ad un SDP server su un device remoto.

Ogni Service record è rappresentato da un istanza della classe ServiceRecord. Questo record contiene gli attributi che descrivono nel dettaglio il servizio. La classe fornisce diversi metodi utili:

- getAttributeIDs() e getAttributeValue() recupera gli attributi dei service record.
- getConnectionURL() ottiene l'URL connection per il server.
- getHostDevice() ottiene il RemoteDevice che ospita il servizio.
- populateRecord() e setAttributeValue() setta gli attributi del service record.
- setDeviceServiceClasses() setta le classi del servizio.

La figura 4.7 mostra le relazioni tra i device bluetooth locali e remoti, il SDDB e i service records.

Per rendere un servizio disponibile ai client, un'applicazione server crea un service record tramite l'operazione di istanza di un connection notifier, successivamente inserisce il record nel SDDB invocando il metodo acceptAndWait() del connection notifier. Le applicazioni server possono reperire il service record e aggiornarlo nel modo più appropriato, e le applicazioni client richiedono al SDDB remoto la disponibilità dei servizi.

4.2.6 La Classe DataElement

Un servizio può avere numerosi attributi, alcuni obbligatori, altri opzionali. Un attributo è rappresentato da un oggetto DataElement, che fornisce i

metodi per settare e ottenere il valore degli attributi. Gli attributi obbligatori sono settati automaticamente quando un servizio è registrato. Questi includono: ServiceRecordHandle, ServiceClassIDList, ServiceRecordState, ServiceID, and ProtocolDescriptorList.

Ci sono molti attributi opzionali, ma tre di questi sono di speciale interesse: ServiceName, ServiceDescription, e ProviderName.

4.2.7 API per Device Management

Ci sono tre classi importanti per la gestione dei Device:

- LocalDevice
- RemoteDevice
- DeviceClass

La classe Local Device rappresenta il dispositivo bluetooth locale. La relazione tra le applicazioni Bluetooth e il LocalDevice è tipicamente uno a uno:

Figura 4.7: Relazione tra le applicazioni Bluetooth e il LocalDevice

Un'istanza della classe RemoteDevice rappresenta un device bluetooth remoto. Prima che l'applicazione client Bluetooth possa consumare dei servizi, deve scoprire remotamente tali servizi tramite l'inizializzazione di un device inquiry. Tipicamente, la relazione tra l'applicazione Bluetooth e il RemoteDevice è uno-a-molti.

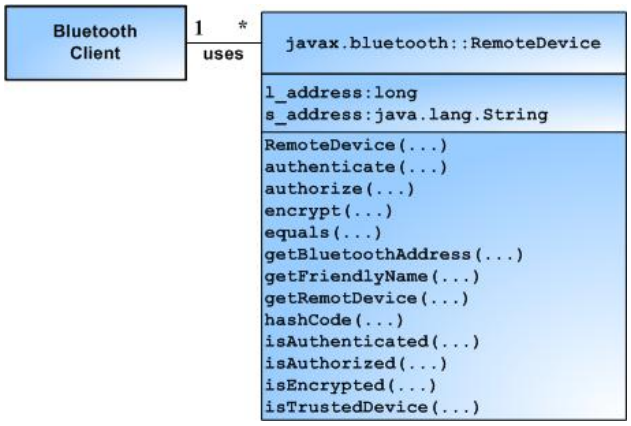

Figura 4.8: Relazione tra applicazione Bluetooth e il RemoteDevice.

La classe DeviceClass rappresenta il tipo di oggetto con il quale si ha a che fare e fornisce i seguenti metodi:

- getMajorDeviceClass() reperisce la classe maggiore del dispositivo.
- getMinorDeviceClass() ritorna la classe minore del dispositivo.
- getServiceClasses() ritorna la classe di servizio del device.

Quando viene scoperto un device, viene anche scoperta la sua classe/tipologia; quando il Discovery Agent invoca il metodo deviceDiscovered(), uno degli argomenti è proprio DeviceClass.

4.3 Comunicazione Bluetooth

Le connessioni JABWT sono basate sul Logical Link Control e Adaptation Layer Protocl. L2CAP è un protocollo a basso livello per la gestione dei pacchetti dati lunghi fino a 64 kilobytes. Esso gestisce i dettagli come la segmentazione dei messaggi e il riassemblaggio e la connessione multiplexing. In aggiunta, il Serial Port Profile (SPP) fornisce un protocollo di emulazione seriale che va sotto il nome di RFCOMM. Le connessioni L2CAP e RFCOMM sono basate su un framework Generic Connection, una semplice gerarchia di interfacce e classi per creare connessioni ed attuare operazioni di I/O.

Il formato dell'URL per una connessione L2CAP (L2CAPConnection) è il seguente:

btl2cap:

- hostname:[
- PSM —
- UUID];
- parameters

mentre per una connessione RFCOMM (StreamConnection)

btspp:

- hostname:[
- CN —
- UUID];
- parameters

- btl2cap è lo schema URL per una connessione L2CAP.
- btspp è lo schema URL per una connessione RFCOMM.
- hostname è il localhost per imbastire la server connection oppure l'indirizzo bluetooth per creare la connessione client.
- PSM è il Protocol/Service Multiplexer value, usato dal client nella connessione con il server. É un concetto simile alla porta TCP/IP.
- CN è il valore del numero del canale usato dal cliente per connettersi al server.
- UUID è il valore identificativo unico quando si inizializza un servizio sul server.
- parameters includono il nome e parametri di sicurezza (authenticate, authorize, e encrypt).

4.3.1 Eccezioni

Le API javax.bluetooth definiscono tre classi di eccezioni:

- BluetoothConnectionException è lanciata quando una connessione Bluetooth L2CAP, RFCOMM, or OBEX-over-RFCOMM non può essere stabilita con successo.
- BluetoothStateException è lanciata quando si è nello stato sbagliato di un'operazione.
- ServiceRegistrationException è lanciata quando c'è un failure nell'aggiunta o cambiamento di un service record nel local Service Discovery Database.

4.3.2 Sicurezza Bluetooth

Una connessione bluetooth sicura utilizza l'autenticazione, l'autorizzazione e la cifratura e può essere resa tale quando è stabilita o successivamente. Da notare che non tutte le implementazioni bluetooth sono sicure. Dal punto di vista implementativo Java mette a disposizione javax.microedition.io.Connector connection URL con i parametri appropriati.

btspp:

- hostname:[
- CN —
- UUID];
- authenticate=true;
- authorize=true;
- encrypt=true

Dove:

- authenticate verifica l'identità del device in connessione.
- authorize verifica se l'accesso è permesso ad un device.

- encrypt specifica se la connessione è criptata.

Si è già visto che un cliente che vuole connettersi ad un servizio può reperire le stringa di connessione URL del servizio stesso tramite la chiamata al metodo ServiceRecord.getConnectionURL(). Uno dei dei parametri di questo metodo, requiredSecurity, specifica se l'URL ritornato debba includere i parametri opzionale di cifratura e autenticazione. I valori validi per requiredSecurity sono:

- ServiceRecord.NOAUTHENTICATE_NOENCRYPT indica authenticate=false; encrypt=false.
- ServiceRecord.AUTHENTICATE_NOENCRYPT indica authenticate=true; encrypt=false.
- ServiceRecord.AUTHENTICATE_ENCRYPT indica authenticate=true; encrypt=true;

4.4 Gestione dei dispositivi

LocalDevice e RemoteDevice sono le due classi principali nelle specifiche Java per il Bluetooth che permettono di attuare la gestione dei device. Queste classi forniscono l'abilità di chiedere informazioni sul proprio device Bluetooth (LocalDevice) e informazioni su device nell'area (RemoteDevice). Il metodo statico LocalDevice.getLocalDevice() ritorna un istanza dell'oggetto LocalDevice per l'utilizzo. Per ottenere l'indirizzo unico della radio Bluetooth locale, basta chiamare il metodo getBluetoothAddress() sull'oggetto del device locale. L'indirizzo Bluetooth svolge lo stesso ruolo del MAC address; infatti ogni device Bluetooth possiede un indirizzo univoco. Se si vogliono raggiungere altri device Bluetooth vicini nell'area, si attua una chiamata al metodo setDiscoverable() nell'oggetto LocalDevice.

Ora si vedrà in dettaglio il concetto che permette la scoperta dei devices bluetooth: il processo di device discovery.

4.4.1 Device Discovery

Il device bluetooth non ha idea di quali altri dispositivi ci siano nei paraggi. Possono essere laptops, desktop, stampanti, così come telefoni cellulari, PDA. Le possibilità sono molteplici. Per il processo di scoperta si utilizzano classi di Device Discovery che sono fornite nelle API Bluetooth di Java. Le due principali sono DiscoveryAgent e DiscoveryListener. Dopo aver ottenuto un

oggetto LocalDevice, è sufficiente creare un'istanza dell'oggetto Discovery Agent con la chiamata al metodo LocalDevice.getDiscoveryAgent().

LocalDevice localdevice = LocalDevice.getLocalDevice(); DiscoveryAgent discoveryagent = localdevice.getDiscoveryAgent();

Ci sono diversi modi per scoprire device remoti di tipo Bluetooth, tuttavia nel seguito verrà approfondito uno in particolare. Innanzitutto l'oggetto deve implementare l'interfaccia DiscoveryListener. Questa interfaccia lavora come ogni *listener*, quindi manderà una notifica ogni qualvolta accade un evento. In questo caso, il device locale verrà avvisato che altri device bluetooth sono nell'area. Per iniziare il processo di discovery, è necessario chiamare il metodo startInquiry() sull'oggetto DiscoveryAgent. Questo è un metodo non bloccante, quindi il dispositivo è libero di eseguire altre operazioni mentre è in attesa della scoperta di altri device.

Quando viene scoperto un nuovo device, la Java Virtual Machine (JVM) chiamerà il metodo deviceDiscovered() della classe che implementa l'interfaccia DiscoveryListener. Questo metodo passerà al local agent un oggetto chiamato RemoteDevice che rappresenta il device scoperto dal metodo inquiry.

4.4.2 Service Discovery

Ora che il processo di scoperta è terminato, ci si chiede quali servizi offrono i device nell'area coperta dal dispositivo. Chiaramente se il device remoto è una stampante, il servizio offerto sarà un printing service. Ma se il dispositivo remoto è un computer? A questo punto entra in gioco il Service Discovery. Esso funziona come il Device Discovery nel senso che viene utilizzato il DiscoveryAgent per attuare l'operazione di "Discovery". Il metodo searchServices() della classe DiscoveryAgent permette di cercare i servizi offerti dall'oggetto RemoteDevice che gira sul dispositivo remoto. Quando i servizi sono trovati, viene chiamato il metodo servicesDiscovered() dalla JVM se l'oggetto locale implementa l'interfaccia DiscoveryListener. Questo metodo callback attraversa l'oggetto ServiceRecord che riguarda il servizio per il quale è cercato. Con un ServiceRecord in mano, è possibile fare molteplici operazioni, ma la cosa più importante è connettere il RemoteDevice da cui ha origine il ServiceRecord:

String connectionURL = servRecord[i].getConnectionURL(0, false);

4.4.3 Service Registration

Prima che un device Bluetooth di tipo client possa usare il Service Discovery su un device server, quest'ultimo deve registrare i suoi servizi all'interno del

Service Discovery Database (SDDB). Questo processo è chiamato Service Registration per un device Bluetooth.

Da notare che in un'applicazione peer to peer come trasferimento di file o chat, è importante ricordare che ogni device può agire sia da client che da server, quindi è necessario incorporare quelle funzionalità nel codice per gestire entrambi gli scenari di Service Discovery e Service Registration. Viene presentato uno scenario di ciò che richiede il processo di registrazione del servizio e memorizzazione nel SDDB:

1. chiamata al metodo Connector.open() e cast del risultato Connection all'oggetto StreamConnectionNotifier. Connector.open() crea un nuovo ServiceRecord e setta alcuni attributi.

2. Usa il LocalDevice object e l'oggetto StreamConnectionNotifier per ottenere il ServiceRecord che è stato creato dal sistema.

3. Aggiungere o modificare gli attributi nel ServiceRecord (opzionale).

4. Usare l'oggetto StreamConnectionNotifier e chiamare il metodo acceptAndOpen() ed attendere che il client Bluetooth scopra il servizio e si connetta. Il sistema crea una record service nel SDDB.

5. Attendere finché un cliente si connette.

6. Quando il server è pronto ad uscire, si chiama la primitiva close() sull'oggetto StreamConnectionNotifier. Il sistema rimuove il service record dal SDDB.

4.5 Servizio OBEX

OBEX (abbreviazione di OBject EXchange, a volte chiamato IrOBEX) è un protocollo di comunicazione che facilita lo scambio di oggetti binari tra dispositivi. Esso è mantenuto da Infrared Data Association ma è anche stato adottato da Bluetooth Special Interest Group e da Open Mobile Alliance. Una delle prime e più popolari applicazioni di Obex fu il Palm III personal digital assistant. Questo PDA e i suoi molteplici successori utilizzarono OBEX per lo scambio di business card, dati, e addirittura applicazioni. Sebbene OBEX fu inizialmente progettato per l'infrarosso, è stato adottato dal protocollo Bluetooth ed è anche utilizzato su RS-232, USB, WAP, e su device come Livescribe smartpens. OBEX funziona tramite lo scambio di oggetti, che sono utilizzati per una discreta varietà di scopi: stabilire i parametri di una connessione, spedire e richiedere dati, cambiare il path corrente o gli

attributi di un file.
Gli oggetti sono composti rispettivamente da fields (campi) e headers (intestazioni). Come esempio, la seguente tabella può essere l'oggetto utilizzato per la richiesta di un contatto della rubrica telefonica di un mobile phone:

Object	Fields	Command	GET, Final	0x83
		Length	total length of object	0x00 0x29
	Headers	Connection ID	1	0xCB 0x00 0x00 0x00 0x01
		Name	telecom/pb.vcf	0x01 0x00 0x1e 0x00 0x74 0x00 0x65 0x00 0x6c 0x00 0x65 0x00 0x63 0x00 0x6f 0x00 0x6d 0x00 0x2f 0x00 0x70 0x00 0x62 0x00 0x2e 0x00 0x76 0x00 0x63 0x00 0x66 0x00 0x00

Tabella 4.1: Parametri di richiesta per il servizio OBEX

Questo oggetto contiene due campi (command and length) e due headers. Il primo campo (command) specifica che è una richiesta per dati (GET). Il secondo campo è la dimensione totale dell'oggetto, inclusi i due campi appena descritti.

Questo oggetto contiene due headers, nello specifico un "Connection ID" e un "Name". Il primo byte di ogni header è il nome dell'header e il tipo contenuto. In questo caso:

- 0xCB significa che questo header è un "Connection ID", un numero ottenuto precedentemente; i due bit più significativi di OxCB sono 11, e questa coppia specifica una quantità di 4 byte;
- il primo byte del secondo header è 0x01; questo byte identifica questo header come "Name"; i primi due bit di 0x01 sono 00, ciò significa che il contenuto di questo header è una stringa di terminazione in formato unicode (in UCS-2), preceduta dal numero di byte con cui essa è formata (0x00 e 0x1e).

Una possibile risposta contenente i dati richiesti può essere quella mostrata in tabella 4.2

Object	Fields	GET, Final	0x83
		total length of object	0x00 0x29
	Headers	1	0xCB 0x00 0x00 0x00 0x01
		telecom/pb.vcf	0x01 0x00 0x1e 0x00 0x74 0x00 0x65 0x00 0x6c 0x00 0x65 0x00 0x63 0x00 0x6f 0x00 0x6d 0x00 0x2f 0x00 0x70 0x00 0x62 0x00 0x2e 0x00 0x76 0x00 0x63 0x00 0x66 0x00 0x00

Tabella 4.2: Parametri di risposta per il servizio OBEX

In questo esempio, il contatto di rubrica può essere contenuto in un singolo oggetto di risposta.. L'unico header ha 0x49 come identificatore il quale indica "End of Body", l'ultimo *chunk* di informazione. I primi due bit di 0x49 sono 01, e ciò significa che il contenuto di questo header sono dati length-prefixed: i successivi byte 0x00 e 0x2F indicano la lunghezza di questi dati (in decimale, 47) e i successivi byte sono i dati veri e propri.

Capitolo 5

Il sistema

Il programma software per lo sviluppo del codice utilizzato per la realizzazione del sistema è *Eclipse* [5]. Eclipse è un ambiente di sviluppo integrato multi-linguaggio e multipiattaforma che può essere utilizzato per la produzione di software di vario genere, si passa infatti da un completo IDE [1] per il linguaggio Java (JDT, "Java Development Tools") a un ambiente di sviluppo per il linguaggio C++ (CDT, "C/C++ Development Tool") e a plug-in che permettono di gestire XML, Javascript, PHP e persino di progettare graficamente una GUI per un'applicazione JAVA (Eclipse VE, "Visual Editor").
Grazie al Visual Editor è stata creata in modo intuitivo l'interfaccia grafica del progetto "trascinando" i componenti elementari come bottoni, campi di testo, etichette, all'interno del *form* visuale. Tramite una apposita sezione è possibile rinominare le variabili che referenziano gli oggetti grafici, modificare le proprietà e generare i metodi di gestione degli eventi collegati ad essi (vedi figura 5.1).

[1] Integrated Development Environment

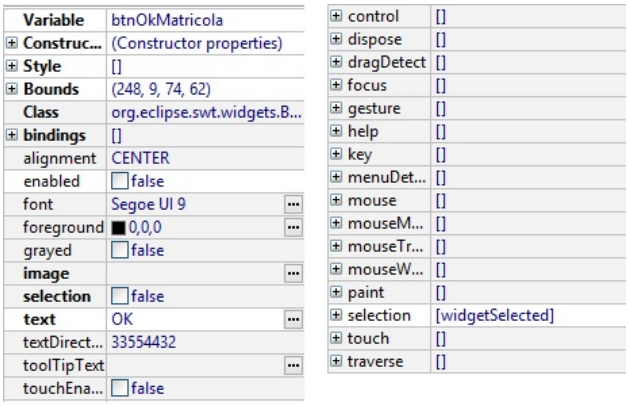

Figura 5.1: Pannelli di Eclipse per la gestione di proprietà ed eventi dell'oggetto grafico

Per una completa visione del problema viene esposto, in figura 5.2, uno schema modulare del progetto. Quest'ultimo serve come riferimento cronologico per le fasi di implementazione, con una chiave di lettura dal sinistra verso destra.

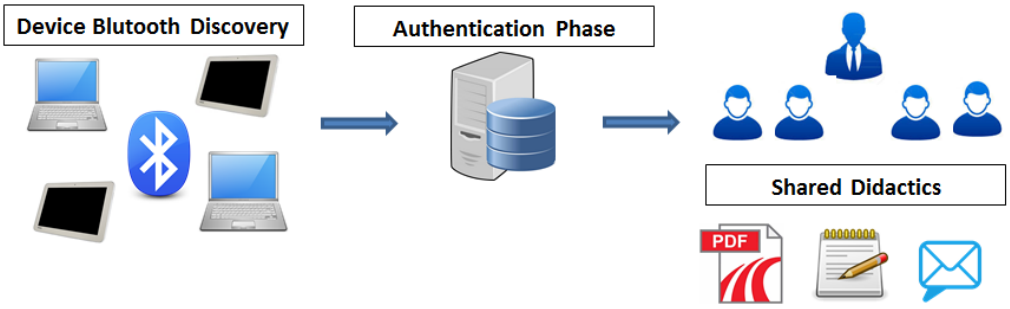

Figura 5.2: Moduli del sistema progettato.

In figura 5.3 è possibile visualizzare lo schema progettuale all'interno dell'ambiente di programmazione. Nella sezione 5.2 vengono spiegate in maniera esaustiva le funzionalità delle varie classi presenti nel progetto.

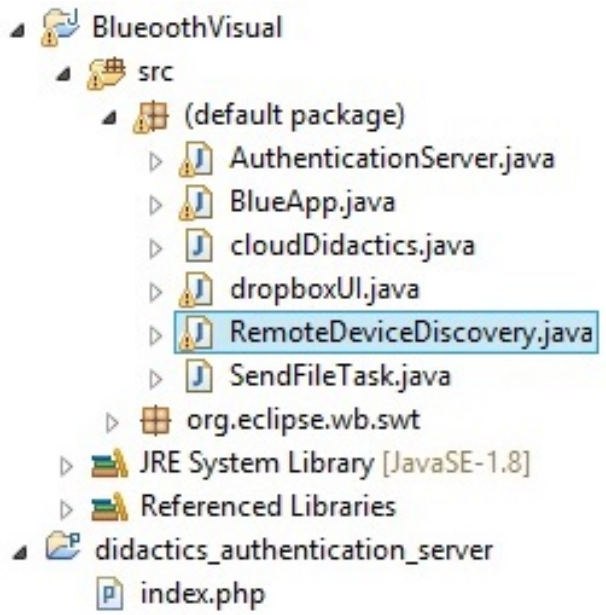

Figura 5.3: Schema del progetto all'interno dell'IDE di programmazione

5.1 Le classi del progetto

5.1.1 BlueApp

BlueApp è la classe core dell'applicazione e in essa viene costruito il form di interfaccia grafica per l'utente finale. I metodi contenuti all'interno sono propriamente finalizzati alla gestione degli eventi generati dai componenti grafici, come la pressione di un *bottone* o la modifica di un campo di testo. Grazie a questi è possibile gestire, oltretutto, l'integrità e la correttezza dell'input dell'utente: ad esempio, viene attivato il bottone per la ricerca dei dispositivi Bluetooth solamente quando vengono compilati entrambi i campi di testo con le informazioni necessarie per l'elaborazione.
All'interno di questa classe principale vengono costruiti gli oggetti delle altre classi ausiliarie. In figura 5.4 viene riportato lo screenshot dell'interfaccia grafica dell'applicazione per la fase di utilizzo da parte dell'utente finale.

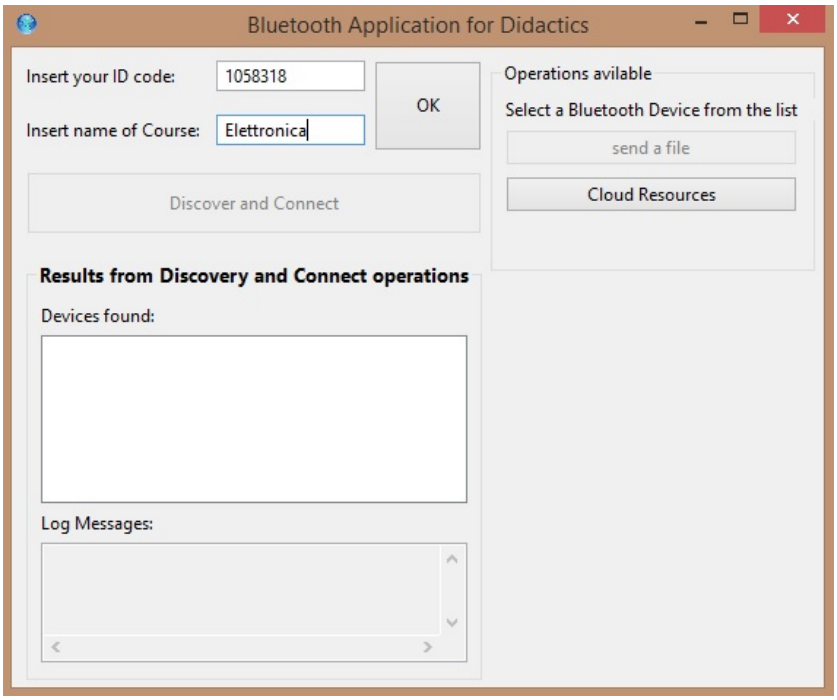

Figura 5.4: Interfaccia grafica lato utente dell'applicazione per la didattica condivisa

5.1.2 RemoteDeviceDiscovery

RemoteDeviceDiscovery è la classe utilizzata per la ricerca di device bluetooth remoti e contiene al suo interno i metodi per la gestione della suddetta operazione. Secondo lo schema 5.2, la prima fase consiste proprio nella ricerca dei dispositivi presenti nello spazio didattico.
Per l'utilizzo delle classi che permettono la gestione del protocollo Bluetooth viene utilizzata una libreria esterna chiamata *BLUECOVE* [3][4] che si interfaccia con Mac OS X, WIDCOMM, BlueSoleil e Microsoft Bluetooth Stack in Windows XP, Vista, 7 e Mobile. La versione di bluecove utilizzata nel progetto è la 2.1.1 ed è importata nell'ambiente di programmazione tramite l'apposita funzionalità *Add External JAR* nelle proprietà del progetto.

Viene riportata la procedura per l'attività di scoperta dei device bluetooth geograficamente vicini al dispositivo che avvia tale metodo. La riga di codice più importante all'interno del codice mostrato di seguito è la seguente:

LocalDevice.getLocalDevice().getDiscoveryAgent().startInquiry(DiscoveryAgent.GIAC, listener);

```
import java.io.IOException;
import java.util.Vector;
import javax.bluetooth.*;

/**
 * Minimal Device Discovery example.
 */
public class RemoteDeviceDiscovery {

    public static final Vector/*<RemoteDevice>*/ devicesDiscovered
        = new Vector();

    public static void main(String[] args) throws IOException,
        InterruptedException {

        final Object inquiryCompletedEvent = new Object();

        devicesDiscovered.clear();

        DiscoveryListener listener = new DiscoveryListener() {

            public void deviceDiscovered(RemoteDevice btDevice,
                DeviceClass cod) {
                System.out.println("Device " +
                    btDevice.getBluetoothAddress() + " found");
                devicesDiscovered.addElement(btDevice);
                try {
                    System.out.println("   name " +
                        btDevice.getFriendlyName(false));
                } catch (IOException cantGetDeviceName) {
                }
            }

            public void inquiryCompleted(int discType) {
                System.out.println("Device Inquiry completed!");
                synchronized(inquiryCompletedEvent){
                    inquiryCompletedEvent.notifyAll();
                }
            }

            public void serviceSearchCompleted(int transID, int
                respCode) {
```

```
            }

            public void servicesDiscovered(int transID,
                ServiceRecord[] servRecord) {
            }
        };

        synchronized(inquiryCompletedEvent) {
            boolean started =
                LocalDevice.getLocalDevice().getDiscoveryAgent().
            startInquiry(DiscoveryAgent.GIAC, listener);
            if (started) {
                System.out.println("wait for device inquiry to
                    complete...");
                inquiryCompletedEvent.wait();
                System.out.println(devicesDiscovered.size() + "
                    device(s) found");
            }
        }
    }
}
```

5.1.3 AuthenticationServer

AuthenticationServer: questa classe si occupa della gestione degli aspetti di autenticazione per gli utenti presenti. In particolare, per ogni device trovato dal processo di discovery, viene generata una richiesta http ad un server di autenticazione remoto. All'interno di quest'ultimo viene mantenuto un database, che può essere pensato come la simulazione della piattaforma Moodle, [2] e sono quindi memorizzate le tabelle relative agli utenti, ai corsi attivi all'interno della struttura didattica, ed infine una tabella di associazione tra queste due (per maggiore chiarezza si faccia riferimento alla figura 5.5).

[2]Modular Object-Oriented Dynamic Learning Environment: è un ambiente informatico per l'apprendimento modulare, dinamico, orientato ad oggetti per la gestione di corsi didattici

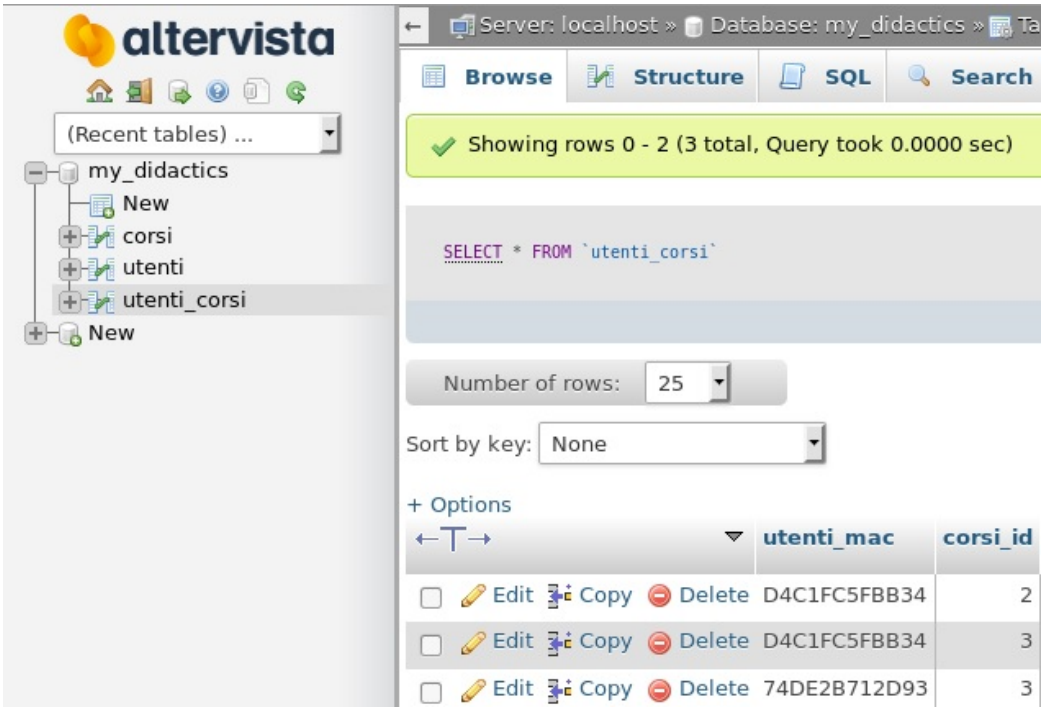

Figura 5.5: Database interno al server di autenticazione.

Con l'ausilio del database appena descritto è facile confrontare le informazioni fornite dall'applicazione (Mac address del dispositivo e corso che si intende seguire) con le informazioni contenute nella base di dati remota. Infatti, con questo confronto è possibile ammettere uno studente abilitato alla sessione di didattica condivisa e negare l'accesso ad utenti non autorizzati. Questa metodologia gestisce anche il seguente scenario: uno studente che si trova in un'aula vicina può essere trovato dal processo di scoperta dei dispositivi bluetooth. Automaticamente questo utente viene scartato dall'applicazione grazie all'informazione sul corso attivato evitando anche l'invasività che può essere generata dalle connessioni entranti. Viene mostrato il codice principale che gestisce la richiesta di autenticazione al server.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.ProtocolException;
import java.net.URL;

public class AuthenticationServer {
        private static final String USER_AGENT = "Mozilla/5.0";
```

```
private String mac;
private String corso;

public AuthenticationServer(String mac, String corso) {
	this.mac = mac;
	this.corso = corso;
}

public String SendConnectionRequest() throws IOException {
	String url = "http://didactics.altervista.org/?mac="
	    + mac + "&corso="+ corso;

	URL obj = new URL(url);
	HttpURLConnection con = (HttpURLConnection)
	    obj.openConnection();

	// optional default is GET
	try {
		con.setRequestMethod("GET");
	} catch (ProtocolException e) {
		e.printStackTrace();
	}

	// add request header
	con.setRequestProperty("User-Agent", USER_AGENT);

	int responseCode;
	responseCode = con.getResponseCode();

	System.out.println("\nSending 'GET' request to URL :
	    " + url);
	System.out.println("Response Code : " +
	    responseCode);

	BufferedReader in = new BufferedReader(new
	    InputStreamReader(con.getInputStream()));
	String inputLine;
	StringBuffer response = new StringBuffer();

	while ((inputLine = in.readLine()) != null) {
		response.append(inputLine);
	}
	in.close();
```

```
            return response.toString();
        }
}
```

5.1.4 DropboxUI

DropboxUI: questa è la classe che realizza l'interfaccia utente grafica per l'interazione con le risorse Cloud. Nello specifico viene costruito un *Windows Form* tramite il quale è possibile collegarsi, dopo una procedura di autenticazione, allo spazio virtuale che ospita le risorse didattiche gestito dall'ente didattico. É possibile selezionare, con il settaggio di due particolari "*radiobutton*" l'operazione di interesse. Sono state utilizzate le API Java messe a disposizione da Dropbox [3] [6] per svolgere le operazioni suddette. Le operazioni disponibili sono rispettivamente:

- Upload di un file: nel contesto il file conterrà gli appunti della lezione di uno studente oppure più generalmente una risorsa multimediale.
- Download di un file: documenti caricati dal docente nella cartella, lucidi della lezioni e, in particolare, appunti di altri studenti.

Di seguito è possibile visualizzare il codice completo per la gestione di queste operazioni. Da notare che nessun client può cancellare o modificare i file presenti nel servizio di Cloud, in quanto l'interfaccia non permette le operazioni menzionate.

```
// Include the Dropbox SDK.
import com.dropbox.core.*;
import java.io.*;
import java.util.Locale;

public class Main {
    public static void main(String[] args) throws IOException,
        DbxException {
        // Get your app key and secret from the Dropbox developers
            website.
        final String APP_KEY = "INSERT_APP_KEY";
        final String APP_SECRET = "INSERT_APP_SECRET";

        DbxAppInfo appInfo = new DbxAppInfo(APP_KEY, APP_SECRET);
```

[3] Un software di cloud storage multi piattaforma, che offre un servizio di file hosting e sincronizzazione automatica di file tramite web.

```
DbxRequestConfig config = new
    DbxRequestConfig("JavaTutorial/1.0",
    Locale.getDefault().toString());
DbxWebAuthNoRedirect webAuth = new
    DbxWebAuthNoRedirect(config, appInfo);

// Have the user sign in and authorize your app.
String authorizeUrl = webAuth.start();
System.out.println("1. Go to: " + authorizeUrl);
System.out.println("2. Click \"Allow\" (you might have to
    log in first)");
System.out.println("3. Copy the authorization code.");
String code = new BufferedReader(new
    InputStreamReader(System.in)).readLine().trim();

// This will fail if the user enters an invalid
    authorization code.
DbxAuthFinish authFinish = webAuth.finish(code);
String accessToken = authFinish.accessToken;

DbxClient client = new DbxClient(config, accessToken);

System.out.println("Linked account: " +
    client.getAccountInfo().displayName);

File inputFile = new File("working-draft.txt");
FileInputStream inputStream = new
    FileInputStream(inputFile);
try {
    DbxEntry.File uploadedFile =
        client.uploadFile("/magnum-opus.txt",
        DbxWriteMode.add(), inputFile.length(), inputStream);
    System.out.println("Uploaded: " +
        uploadedFile.toString());
} finally {
    inputStream.close();
}

DbxEntry.WithChildren listing =
    client.getMetadataWithChildren("/");
System.out.println("Files in the root path:");
for (DbxEntry child : listing.children) {
```

```
            System.out.println("      " + child.name + ": " +
                child.toString());
        }

        FileOutputStream outputStream = new
            FileOutputStream("magnum-opus.txt");
        try {
            DbxEntry.File downloadedFile =
                client.getFile("/magnum-opus.txt", null,
                outputStream);
            System.out.println("Metadata: " +
                downloadedFile.toString());
        } finally {
            outputStream.close();
        }
    }
}
```

L'interfaccia grafica è attivabile tramite il pulsante "Cloud Resource" del form principale ed è mostrata in figura 5.6

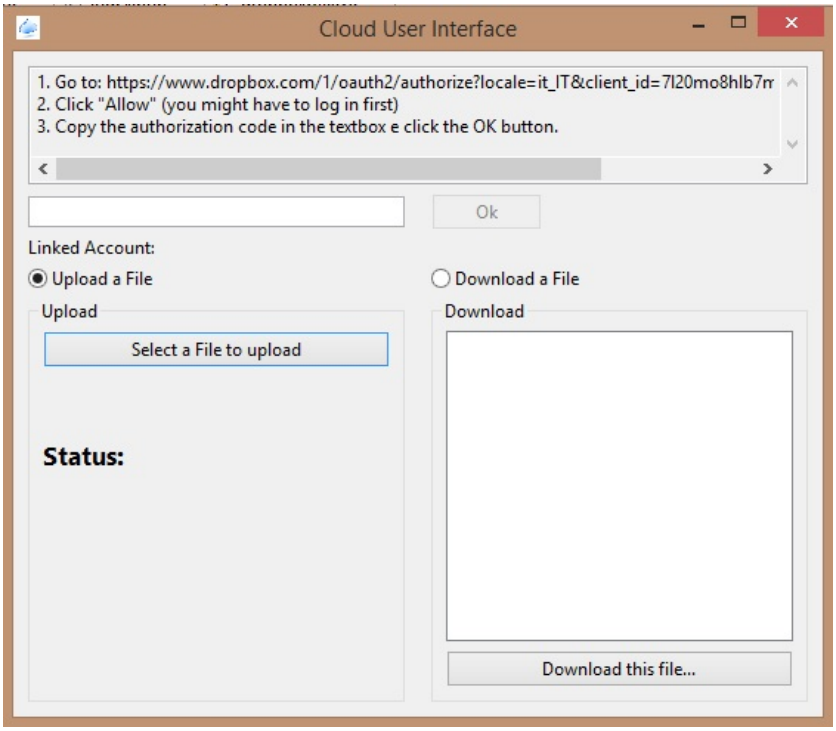

Figura 5.6: Interfaccia grafica lato utente per l'utilizzo delle risorse Cloud

5.1.5 SendFileTask

SendFileTask: questa classe è utilizzata dal form principale dell'applicazione per inviare un file qualsiasi scelto dall'utente ad un dispositivo bluetooth selezionato tramite la lista dei device disponibili. É eventualmente necessaria un'operazione di *Pairing* (accoppiamento) dei dispositivi comunicanti tramite lo scambio di una chiave di sicurezza. Tale operazione viene eseguita una sola volta. Per lo scambio dei file viene utilizzato il servizio OBEX [7] spiegato dettagliatamente nella sezione 4.5. Di seguito viene mostrato il codice sorgente che implementa tale funzionalità:

```
import java.io.OutputStream;
import javax.microedition.io.Connection;
import javax.microedition.io.Connector;
import javax.obex.ClientSession;
import javax.obex.HeaderSet;
import javax.obex.Operation;

public class SendFileTask implements Runnable {

    private String btConnectionURL;
    private byte[] file;
    private String filename;

    public SendFileTask(String url, byte[] file, String filename) {
        this.btConnectionURL = url;
        this.file = file;
        this.filename = filename;
    }
        public void run() {
               try {
                       Connection connection =
                           Connector.open(btConnectionURL);
                       // connection obtained
                       // now, let's create a session and a
                           headerset objects
                       ClientSession cs = (ClientSession) connection;
                       HeaderSet hs = cs.createHeaderSet();
                       // now let's send the connect header
                       cs.connect(hs);
                       hs.setHeader(HeaderSet.NAME, filename);
                       hs.setHeader(HeaderSet.TYPE, "image/jpeg");
                       hs.setHeader(HeaderSet.LENGTH, new
```

```
                        Long(file.length));
                    Operation putOperation = cs.put(hs);
                    OutputStream outputStream =
                        putOperation.openOutputStream();
                    outputStream.write(file);
                    // file push complete
                    outputStream.close();
                    putOperation.close();
                    cs.disconnect(null);
                    connection.close();
                } catch (Exception e) {
                    System.err.println("Exception: " +
                        e.getMessage());
                    e.printStackTrace();
                }
        }
}
```

5.1.6 DidacticsAuthenticationServer

index.php in **DidacticsAuthenticationServer**: l'interfacciamento con il database viene gestito tramite un file scritto in linguaggio *php* all'interno del quale sono generate le *query sql* atte alla raccolta delle informazioni sugli studenti abilitati alla session didattica. Oltretutto viene ottenuto, sempre tramite una query sql, il professore del corso. Questa operazione viene fatta agevolmente con l'utilizzo di un flag all'interno del database che viene settato a 0 se l'utente in questione è un professore, e viene settato a 1 se invece è uno studente. Il codice completo in php della risorsa di autenticazione remota viene mostrata di seguito:

```
<?php
// we connect to example.com and port 3307
$link = mysql_connect('didactics.altervista.org', 'didactics',
    'dursurafbi21');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}

mysql_select_db('my_didactics', $link);
$mac = $_GET['mac'];
$corso = $_GET['corso'];
$matricola = $_GET['matricola'];
```

```
$risultato = mysql_query("SELECT COUNT(*) as allowed FROM
    utenti_corsi WHERE utenti_mac='$mac' AND corsi_id IN
                (SELECT id FROM corsi WHERE nome='$corso')")
        or die("Query non valida: " . mysql_error());
$prof = mysql_query("SELECT ruolo FROM utenti_corsi WHERE
    utenti_mac='$mac' AND corsi_id IN
                (SELECT id FROM corsi WHERE nome='$corso')") or
                    die("Query non valida: " . mysql_error());
$isprof = mysql_fetch_row($prof);
$row = mysql_fetch_row($risultato);
if($row[0]>0 and $isprof[0] == 0)
        echo "0";
else if ($row[0]>0 and $isprof != 0)
        echo "1";
else echo "-1";
mysql_close($link);
```

Il file prodotto viene caricato sul server tramite l'utilizzo di *FileZilla Client*, un software libero multi-piattaforma che permette il trasferimento di file in Rete attraverso il protocollo FTP [4].

5.2 Funzionamento dell'applicazione

All'utente viene inizialmente chiesto di inserire il proprio identificativo (matricola) e il corso che intende seguire nei componenti grafici (Textbox) adibiti alla raccolta delle suddette informazioni. Tramite il tasto "Discover and Connect" l'applicativo avvia la procedura di scoperta dei Device Bluetooth che si trovano nel raggio di copertura del dispositivo. A questo punto viene effettuata l'operazione di autenticazione descritta precedentemente tramite il server remoto dei dispositivi scoperti. Il server controlla nel database interno quale MAC Address, e quindi quale studente, è abilitato a seguire il corso. In definitiva, esso fornisce di ritorno una lista dei dispositivi che parteciperanno alla session di didattica condivisa. Tramite il componente visuale *lista* è possibile interagire con i dispositivi singolarmente (scambiare messaggi e risorse didattiche come appunti e slide). Se l'utente che utilizza l'applicazione è il professore del corso, il sistema è progettato per la rilevazione automatica di tale casistica grazie alla fase di autenticazione. In questo scenario vengono abilitate delle funzioni speciali per tale utente (professore) come l'invio in *broadcast* di un file come ad esempio le slide della lezione giornaliera oppu-

[4] File Transfer Protocol (protocollo di trasferimento file), in informatica e nelle telecomunicazioni, è un protocollo per la trasmissione di dati tra host basato su TCP.

re la funzionalità di registro elettronico attivabile tramite la pressione del pulsante "Electronic Register" che compare all'interno dell'area *Operation available*. Quest'ultimo servizio scrive in un file di testo la data corrente e l'elenco dei dispositivi bluetooth (corredati di nome e mac address) presenti nella lista di device disponibili. Lo *screenshot* di figura 5.7 mostra lo scenario appena esplicato.

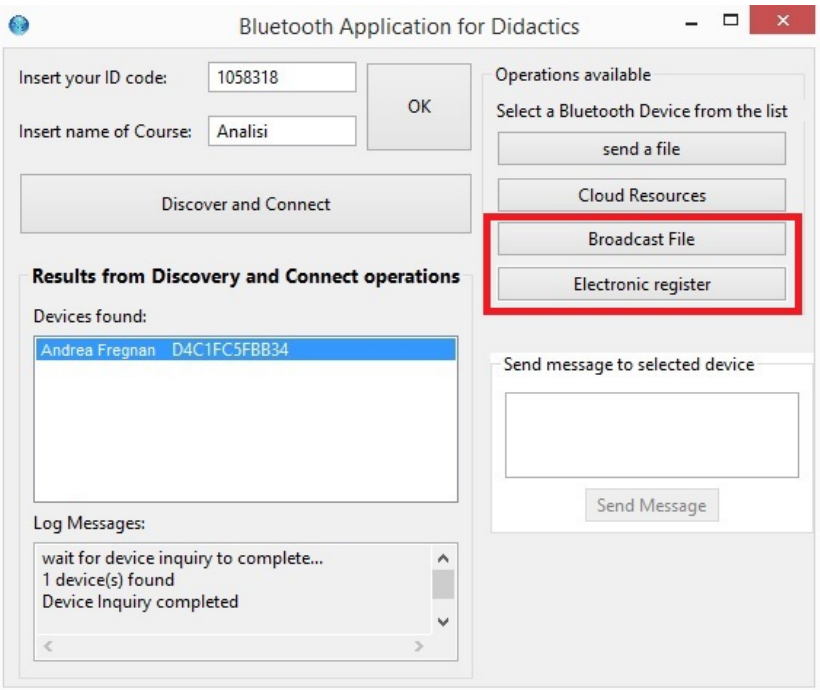

Figura 5.7: Comportamento dell'applicazione quando il LocalDevice corrisponde al Mac Address del professore.

5.3 Android Platform

5.3.1 Cos'è Android

Android è un sistema operativo per dispositivi mobili come smartphone e Tablet, sviluppato da Google Inc. sulla base del kernel Linux. Android adotta una politica di licenza di tipo open source (escluse alcune versioni intermedie). La licenza (Licenza Apache) sotto cui è rilasciato consente di modificare e distribuire liberamente il codice sorgente. Inoltre, Android dispone di una vasta comunità di sviluppatori che realizzano applicazioni con l'obiettivo di aumentare le funzionalità dei dispositivi. Queste applicazioni sono scritte soprattutto in linguaggio di programmazione Java. Per questi

motivi è presente in rete una grande quantità di materiale di supporto agli sviluppatori come forum, codici sorgenti, problematiche sollevate da utenti che utilizzano il sistema operativo mobile e le relative soluzioni.

5.3.2 App per la didattica condivisa

Eclipse permette di sviluppare applicazioni per Android all'interno dell'ambiente di programmazione integrato tramite l'installazione di particolari *plugin* che vanno sotto il nome di Android Development Tool (ADT). L'operazione di estensione avviene sotto la supervisione del cosiddetto Android SDK (Software development kit) mostrato in figura 5.8.

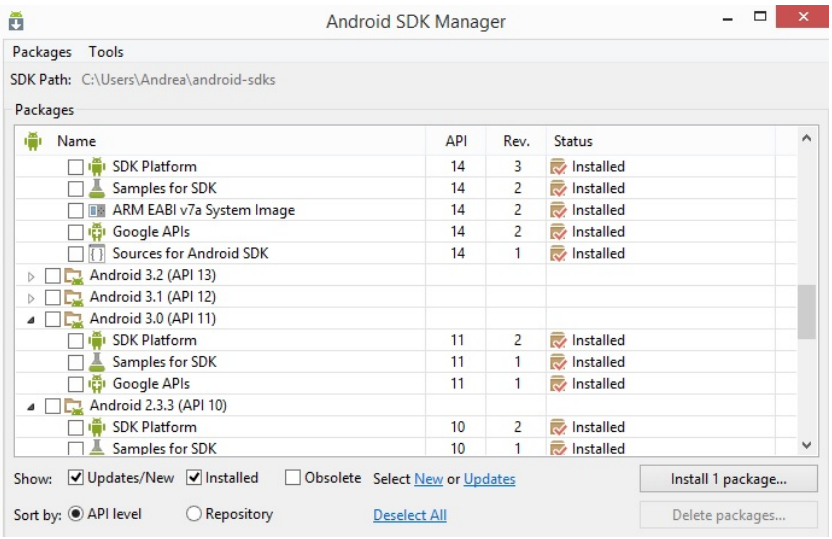

Figura 5.8: SDK di Android in Eclipse

Ora Eclipse ospita le nuove funzionalità di sviluppo per Android. É quindi possibile creare un nuovo progetto e, tramite un visual editor, creare l'interfaccia grafica adattata per i dispositivi mobili. Cambia leggermente la modalità di assegnazione delle proprietà e degli eventi agli oggetti grafici in quanto viene effettuata tramite righe di codice in linguaggio *xml* [5]. Ad esempio per i *widget* campo di testo e *button* vengono prodotte le seguenti righe di codice:

[5] eXtensible Markup Language: è un linguaggio di markup, ovvero un linguaggio marcatore basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo.

```
<EditText
        android:id="@+id/tbCourse"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/tbId"
        android:layout_below="@+id/tbId"
        android:layout_marginTop="18dp"
        android:ems="10"
        android:text="Insert Course Name"
        android:inputType="text"/>

    <Button
        android:id="@+id/btnSearch"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/tbCourse"
        android:layout_alignParentRight="true"
        android:text="Search.."
        android:onClick="discoveryDevice" />
```

É necessario inoltre inserire le seguenti righe di codice nel file "*AndroidManifest.xml*" per abilitare i permessi di utilizzo del bluetooth, del network e della scheda di memoria sul device Android.

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"
    />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission
    android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission
    android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission
    android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

A questo punto della progettazione si tratta di svolgere un riadattamento del codice sorgente già prodotto per l'applicazione desktop in quanto i servizi offerti dall'applicazione mobile sono molto simili. Come esplicato nella sezione 5.3, esistono in rete delle risorse per la programmazione Android molto utili agli sviluppatori in particolare librerie e Application Program Interface (API). Per l'applicazione di didattica condivisa viene richiesto l'utilizzo del protocollo bluetooth su dispositivi mobili.

La piattaforma Android include il supporto per lo stack di rete Bluetooth [8], che consente a un dispositivo di scambiare dati in modalità wireless con altri dispositivi Bluetooth. Il framework consente di accedere alle funzionalità Bluetooth attraverso il Bluetooth API di Android. Queste API consentono alle applicazioni che si connettono in modalità wireless ad altri dispositivi Bluetooth, l'abilitazione di funzioni point-to-point, multipoint e funzioni wireless.

Utilizzando le API Bluetooth, un'applicazione Android in grado di eseguire le seguenti operazioni:

- Cercare altri dispositivi Bluetooth
- Interrogare l'adattatore Bluetooth locale per i dispositivi Bluetooth accoppiati
- Stabilire canali RFCOMM
- Collegamento ad altri dispositivi attraverso il rilevamento dei servizi
- Trasferire dati da e verso altri dispositivi
- Gestire connessioni multiple

Tutte le API Bluetooth sono disponibili nel pacchetto *android.bluetooth*. All'interno di questo pacchetto sono contenute diverse classi. Di seguito ne vengono elencate le principali:

- **BluetoothAdapter**: rappresenta l'adattatore Bluetooth locale (radio Bluetooth). Il BluetoothAdapter è il punto di ingresso per tutte le interazioni Bluetooth. Con questo, è possibile scoprire altri dispositivi Bluetooth, interrogare un elenco di dispositivi accoppiati, un'istanza di un BluetoothDevice utilizza un indirizzo MAC noto, e crea un BluetoothServerSocket per ascoltare le comunicazioni da altri dispositivi.
- **BluetoothSocket**: rappresenta l'interfaccia per un socket Bluetooth (simile alla classe TCP Socket). Questo è il punto di connessione che consente a un'applicazione di scambiare dati con un altro dispositivo Bluetooth tramite InputStream e OutputStream.
- **BluetoothServerSocket**: rappresenta un server socket aperto che ascolta le richieste in ingresso (simile alla classe TCP ServerSocket). Per collegare due dispositivi Android, un dispositivo deve aprire un socket server con questa classe. Quando un dispositivo Bluetooth remoto

effettua una richiesta di connessione al dispositivo, il BluetoothServerSocket restituirà una BluetoothSocket collegato quando la connessione viene accettata.

- **BluetoothClass**: descrive le caratteristiche generali e le capacità di un dispositivo Bluetooth. Questa è un set di proprietà in sola lettura insieme che definiscono le classi di unità maggiore e minore del dispositivo e dei suoi servizi. Tuttavia, questo non descrive in modo affidabile tutti i profili e i servizi Bluetooth supportati dal dispositivo, ma è utile come un suggerimento per il tipo di dispositivo.

- **BluetoothProfile**: un'interfaccia che rappresenta un profilo Bluetooth. Un profilo Bluetooth è una specifica interfaccia wireless per la comunicazione basata su Bluetooth tra dispositivi. Un esempio è il profilo Hands-Free.

Prima di proseguire, viene fatto cenno alla modalità di esecuzione dell'operazione di *debug*. Nello specifico per il running, il testing, e il debugging dell'applicazione Android in via di sviluppo vi sono due modalità selezionabili nell'ambiente Eclipse:

- tramite Android Virtual Device (AVD): questa modalità permette di eseguire l'applicazione in un ambiente Android simulato e virtuale, creato appositamente via software nell'ambiente di sviluppo e dimensionato tramite il settaggio di alcuni parametri come il nome del device, il tipo, il livello delle Application Program Interface del sistema operativo (Target), il tipo di CPU e lo skin grafico. In questa modalità virtuale tuttavia non è abilitato il protocollo bluetooth.

- tramite il collegamento di un device Android fisico via USB. É necessario settare su quest'ultimo l'impostazione di "sviluppatore" accedendo al menù impostazioni - opzioni sviluppatore. L'applicazione verrà quindi avviata direttamente sul dispositivo.

É possibile a questo punto del progetto scrivere il codice per la realizzazione dell'applicazione. Il punto di partenza è sempre la fase di *discovery* come illustrato nello schema di figura 5.2. Nel codice sottostante viene illustrato il funzionamento delle librerie bluetooth di Android per la ricerca di dispositivi nelle vicinanze:

```
import java.util.ArrayList;
import android.support.v7.app.ActionBarActivity;
import android.media.audiofx.Visualizer;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.ListAdapter;
import android.widget.ListView;
import android.widget.Toast;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
public void discoveryDevice(View view) {
                BluetoothAdapter mBluetoothAdapter = BluetoothAdapter
                                .getDefaultAdapter();
                if (mBluetoothAdapter == null) {
                        Toast.makeText(getBaseContext(),
                                        "Device does not support
                                            Bluetooth",
                                            Toast.LENGTH_LONG).show();
                }
                if (!mBluetoothAdapter.isEnabled()) {
                        Intent enableBtIntent = new Intent(
                                        BluetoothAdapter.ACTION_REQUEST_ENABLE);
                        startActivityForResult(enableBtIntent,
                            REQUEST_ENABLE_BT);
                }
                // enable discoverability
                Intent discoverableIntent = new Intent(
                                BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
                discoverableIntent.putExtra(
                                BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION,
                                    300);
                startActivity(discoverableIntent);
                // device discovery process
                final ListView list = (ListView)
                    findViewById(R.id.listDevice);
```

```
                final BroadcastReceiver mReceiver = new
                    BroadcastReceiver() {
                    public void onReceive(Context context, Intent
                        intent) {
                        String action = intent.getAction();
                        // When discovery finds a device
                        if
                            (BluetoothDevice.ACTION_FOUND.equals(action))
                            {
                            // Get the BluetoothDevice object from
                                the Intent
                            BluetoothDevice device =
                                intent.getParcelableExtra
                                (BluetoothDevice.EXTRA_DEVICE);
                            // Add the name and address to an array
                                adapter to show in a ListView
                            mArrayAdapter.add(device.getName() + "\n"
                                + device.getAddress());
                            list.setAdapter((ListAdapter)
                                mArrayAdapter);
                        }
                    }
                };
                // Register the BroadcastReceiver
                IntentFilter filter = new
                    IntentFilter(BluetoothDevice.ACTION_FOUND);
                registerReceiver(mReceiver, filter);
        }
}
```

In figura 5.9 viene mostrato, tramite l'utilizzo di un diagramma di flusso, il funzionamento del codice appena riportato.

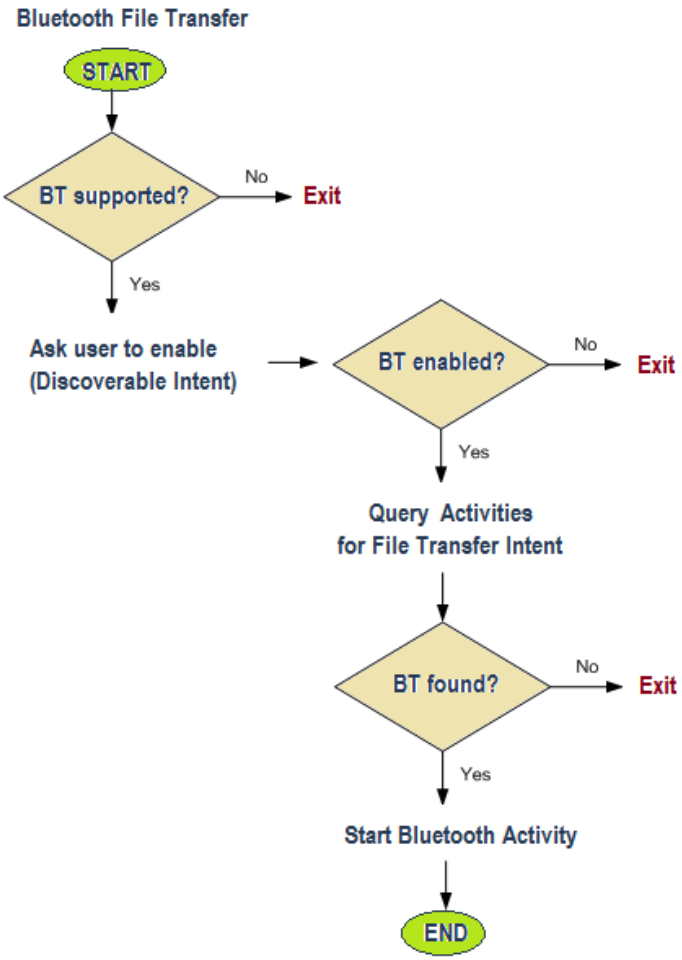

Figura 5.9: Diagramma di flusso per il codice Android.

Lo schema progettuale all'interno dell'ambiente di programmazione viene mostrato in figura 5.10

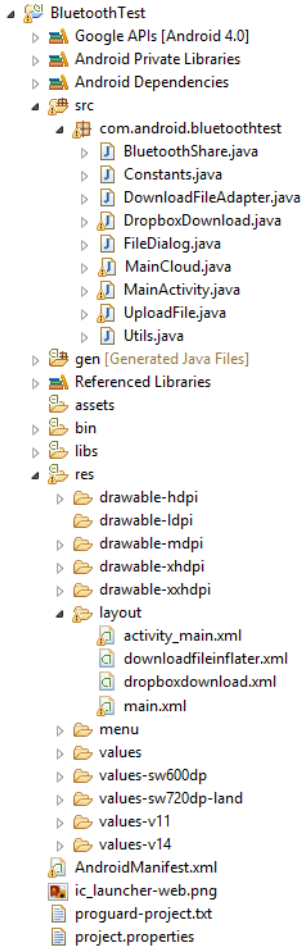

Figura 5.10: Package Explorer per l'applicazione Android

La classe **MainActivity.java** contiene i metodi per la gestione degli eventi generati dall'interfaccia grafica descritta e progettata nel file *activitymain.xml*
La classe **MainCloud.java** implementa invece i metodi per la gestione degli eventi generati dall'interfaccia grafica per il dialogo virtuale con il servizio che ospita la risorse cloud (Dropbox), esattamente come avveniva per l'applicazione desktop. Sono necessarie delle API specifiche per utilizzare Dropbox su sistemi operativi Android. É possibile infatti scaricare tali librerie ed importarle nel progetto all'interno della cartella *Android Private Libraries*.
Le classi appena esposte sono le classi principali per i processi fondamentali

dell'applicazione. Le altre sono classi ausiliarie utilizzate dalle *main classes* per le varie operazioni eseguite dal sistema.

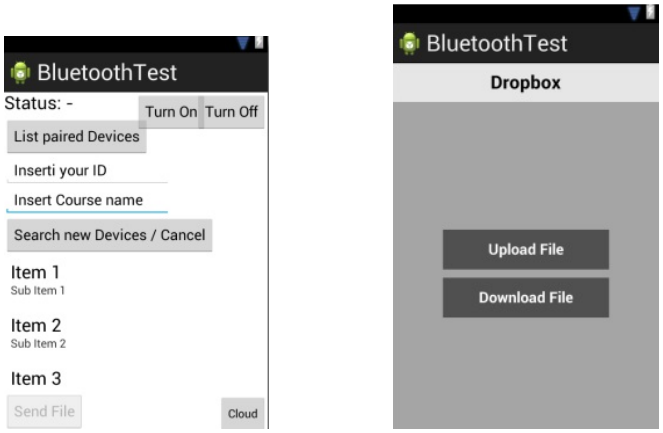

Figura 5.11: Interfaccia grafica per l'applicazione di didattica condivisa in Android

Viene sottolineato un aspetto riguardante la sicurezza nello scambio dei file tramite piattaforma Android. Sono disponibili infatti due metodi per creare una connessione di tipo *Rfcomm*:

- createRfcommSocketToServiceRecord(uuid): funziona e connette il dispositivo ad altri device in modo sicuro sulle versione Android 2.1 - 3.x, ma non funziona tuttavia per le versioni 4.x,
- createInsecureRfcommSocketToServiceRecord(uuid): funziona bene in tutte le versioni di Android, eccetto per la 2.1 in quanto le connessioni insicure sono state introdotte a partire dal livello 10 delle API della piattaforma.

Da notare in particolare il leggero cambiamento di classi e metodi per la gestione delle entità. In particolare per la parte di autenticazione dei dispositivi viene effettuato un processo asincrono in *background* che svolge le richieste *http* al database del server che simula l'ambiente Moodle. In particolare viene riportato il codice sorgente che implementa tale funzione:

```
private class HttpAsyncTask extends AsyncTask<String, Void,
    String> {
                @Override
                protected String doInBackground(String... urls) {
                        return GET(urls[0]);
                }
```

```
            // onPostExecute displays the results of the
                AsyncTask.
            @Override
            protected void onPostExecute(String result) {
                    String[] tokens = result.split("\\|");
                    if("0".equals(tokens[1])){
                            BluetoothDevice d =
                                devices.get(tokens[0]);
                            BTArrayAdapter.add(d.getName() + "|" +
                                d.getAddress());
                            macprof = tokens[0];
                            BTArrayAdapter.notifyDataSetChanged();
                    }
                    else if("1".equals(tokens[1]))
                    {
                            BluetoothDevice d =
                                devices.get(tokens[0]);
                            BTArrayAdapter.add(d.getName() + "|" +
                                d.getAddress());
                            BTArrayAdapter.notifyDataSetChanged();
                    }
                    else{
                            Log.e("wrong","Error"+result);
                    }
            }
    }
```

Data la presenza di processi asincroni all'interno dell'applicazione viene costruita una struttura dati che si utilizza per la fase di autenticazione. Tale struttura dati è una *HashMap*, la quale viene popolata dopo il processo di discovery ed è dimensionata in modo tale da mettere in corrispondenza una data chiave con un dato valore (nel caso in questione la stringa mac address del device blutooth che opera come chiave e l'entità BluetoothDevice che funge da valore indicizzato dalla chiave). In genere la tabella di Hash viene usata per l'implementazione di strutture dati astratte associative come Map o Set ed è molto utilizzata nei metodi di ricerca nominati *Hashing* in quanto resi più veloci da queste strutture.

5.4 Javadoc

L'applicazione sviluppata per la didattica condivisa può essere naturalmente estesa tramite l'aggiunta di altre *feature* da sviluppatori esterni. Per questo motivo vengono messi a disposizione i **Javadoc** sia per la versione desktop e sia per la versione mobile. Javadoc è un applicativo incluso nel Java Development Kit della Sun Microsystems, utilizzato per la generazione automatica della documentazione del codice sorgente scritto in linguaggio Java. Con questa procedura è possibile conoscere la funzione di ogni singolo metodo senza dover entrare nel dettaglio del codice sorgente del metodo stesso. I Javadoc vengono inseriti all'interno del codice tramite una apposita sintassi. Nel codice riportato di seguito sono evidenziati in verde dall'ambiente di programmazione. Eclipse mette a disposizione una procedura automatizzata per la creazione dei Javadoc. In particolare è possibile utilizzare un *Javadoc doclet* per customizzare l'output e il formato della documentazione stessa (nel presente lavoro è stato utilizzato il formato di output HTML).

```
/**
* Event Handler for text modification of textBox ID. When it is
* pressed, botton Discover and Connect is set on true.
*/
btnOkMatricola.addSelectionListener(new SelectionAdapter() {
	@Override
	public void widgetSelected(SelectionEvent e) {
		btnDiscConn.setEnabled(true);
		groupBox.setEnabled(true);
	}
});

/**
* Start the inquiry process with a RemoteDeviceDiscovery call.
* Check every MAC address with the AuthenticationServer and
* show the enable devices on the list component
*/
btnDiscConn.addSelectionListener(new SelectionAdapter() {
	@Override
	public void widgetSelected(SelectionEvent e) {
```

5.5 Release e deployment

L'applicazione stabile per dispositivi desktop viene rilasciata alla versione 1.0 per l'utilizzo in ambito didattico. Il deployment viene effettuato utilizzando una apposita funzione dell'ambiente di programmazione Eclipse che permette di esportare il progetto in un file di estensione *jar*. É possibile convertire successivamente questo file in un eseguibile (estensione .exe) utilizzando il software gratuito *Launch4j* disponibile in rete. Ottenuto questo file eseguibile si può quindi iniziare ad utilizzare l'applicazione sviluppata. Il processo di release e deployment è riassunto in figura 5.12.

Figura 5.12: Processo per la conversione in file eseguibile

É possibile inoltre distribuire l'applicativo sotto forma di pacchetto di installazione. Quest'ultimo viene creato tramite l'utilizzo di un programma gratuito disponibile in rete chiamato "CyberInstaller" che permette, per l'appunto, di creare un pacchetto di setup guidato per l'installazione su dispositivi Windows oriented direttamente dal codice sorgente del progetto sviluppato.

Per l'operazione di release dell'applicazione per dispositivi mobile il processo è più complesso: prima di effettuare i test definitivi sulla app si deve preparare la release candidate build. Per fare questo è necessario aver implementato e testato tutte le funzionalità utili al corretto funzionamento dell'applicazione, sistemato tutti i bug e rimosso tutto il codice utilizzato per la diagnostica per evitare che incida nelle performance della app. A questo punto si può passare alla fase di testing e, superata questa fase, si ottiene la *release canditate build*. Successivamente si genera il package Android, nella pratica un file con estensione .apk, e si svolge l'operazione di ufficializzazione tramite la firma digitale. Per completare facilmente questi passaggi il plugin di Android per Eclipse mette a disposizione un wizard che semplifica notevolmente il lavoro.

Capitolo 6

Conclusioni

6.1 Possibili miglioramenti

In questa sezione vengono presentati i possibili *improvment* che possono essere adottati e implementati per rendere l'applicazione più robusta e completa.

1. Estensione dell'applicazione a dispositivi mobili con sistema operativo iOs

2. Realizzazione di un servizio di messaggistica *istantanea* all'interno dell'aula per gli utenti connessi alla rete ad-hoc

3. Progettazione e sviluppo di un'estensione applicativa per lo svolgimento di esercitazioni multimediali sui dispositivi

4. Implementazione di un protocollo di gestione delle piconet ed eventuali protocolli di routing per l'instradamento di pacchetti tra i vari nodi della rete.

Per il punto 3, si può pensare di utilizzare un servizio multimediale online attivato esclusivamente per gli studenti collegati alla rete ad-hoc all'interno dell'aula didattica. In particolare, il professore del corso (per il quale sono attivate funzionalità speciali) è abilitato alla condivisione di un link in broadcast agli utenti presenti. Il link alla risorsa online può essere criptato.

Il punto 4 richiede particolare attenzione: nello specifico è possibile che si crei, all'interno della rete, la situazione illustrata in figura 2.2 nella quale il nodo B funge da intermediario e unico punto di comunicazione tra i nodi A e C. Cosa succede se tale nodo non fosse più disponibile all'interno dell'intera rete? Il nodo A non potrebbe più comunicare con il nodo C, in quanto le due

sotto-reti non avrebbero più un nodo in comune. Si rende quindi necessario lo sviluppo di una metodologia che affronti lo scenario presentato precedentemente. La soluzione viene individuata nell'utilizzo delle *politiche di routing*: periodicamente i nodi della rete generano dei segnali che informano i vicini dello stato in cui si trova un nodo. In questo modo vengono aggiornate le tabelle dei vicini o tabelle di routing all'interno delle quali sono memorizzate informazioni relative, appunto, ai nodi raggiungibili nella sotto-rete ed eventualmente al percorso più breve per raggiungere i nodi in sottoreti diverse. Sono presenti in letteratura diversi protocolli di routing, per precisione si veda l'appendice B. In fase di progettazione è comunque necessario tenere conto dell'overhead generato all'interno della rete a causa dell'introduzione di politiche di routing. La migliore soluzione individuata in fase decisionale prevede l'utilizzo di protocolli di routing proattivo con update delle tabelle di routing in modalità condizionale o event-driven B.1.5.

La presente tesi risolve in modo distinto il problema commissionato nella sezione 1.1, unendo elementi semplici già esistenti come la tecnologia bluetooth, il linguaggio di programmazione Java e l'ambiente di sviluppo ad una discreta creatività personale.

Appendices

Appendice A

Beaconing

A.1 Overview

La mobilità è una proprietà essenziale di una vasta gamma di reti wireless che pone diversi problemi nello spazio di progettazione di protocolli. Le reti wireless di cellulari e dispositivi mobili sono ambienti dinamici in cui i nodi adiacenti possono essere scoperti o persi in qualsiasi momento. In un ambiente mobile con rapide modifiche alla topologia, un problema comune a diversi protocolli è quello di mantenere le informazioni sullo stato dei collegamenti vicini circa un *hop* (diretti).

Viene inserito questo capitolo di approfondimento della presente tesi in quanto il processo di scoperta dei nodi all'interno della rete ad hoc che viene creata nel contesto di lavoro è avviato dai primi due nodi fisicamente presenti e ai quali i successivi si connettono. Inoltre è necessario tenere traccia dello stato interno di ogni nodo della rete, in particolare se sono ancora attivi oppure se non fanno più parte dell'interno sistema e quindi le problematiche derivanti da questo scenario (aggiornamento dei collegamenti ed eventuali tabelle di stato). Viene quindi studiato dal punto di vista ingegneristico questo processo.

Un approccio ampiamente utilizzato per raccogliere informazioni sullo stato dei collegamenti è emettere pacchetti *beacon* o segnali beacon [9]. I *beacons* possono essere classificati come espliciti o impliciti. Un beacon esplicito (per esempio: il messaggio ciao) è un segnale o pacchetto inviato da un nodo al fine di notificare nodi vicini sulla sua presenza. Il beacon implicito è il risultato dell'ascolto di pacchetti di dati provenienti da un nodo adiacente. Questo approccio è utilizzato, ad esempio nel controllo della topologia *MobileGrid*. Qui, i beacons sono modellati come un processo general point, che

riguarda i casi sia impliciti ed espliciti o addirittura uno schema ibrido (come un processo aggregato). I beacons potrebbero essere classificati come sincroni o asincroni. I beacon sincroni sono inviati da tutti i nodi (o un gruppo di nodi) in particolari istanze temporali. Nello schema di beacon sincroni, i nodi che sono interessati ai beacons potrebbero svegliarsi solo per un breve periodo, ascoltare per eventuali segnali, e andare in stato di sleep quando non sono rilevati beacon da altri nodi. Inoltre i beacons possono essere periodici o aperiodici.

Il tempo di inter-beacon è costante per un processo periodico di beaconing e segue una distribuzione casuale per un processo di beaconing aperiodico. I beacons potrebbero essere unidirezionali oppure bidirezionali. Nelle segnalazioni a senso unico solo un end point invia beacon e l'altra estremità riceve , ad esempio nelle reti cellulari (o basati su cluster) in cui alcune stazioni base sono responsabili per trasmettere beacon e fare collegamenti efficaci. IEEE 802.11 in modalità infrastruttura e reti Bluetooth tradizionali (in cui un nodo può inviare o ricevere beacon in un momento particolare) sono altri esempi di segnalazioni ad una via. In altri casi, un canale efficace bidirezionale è stabilito tra due nodi e subito dopo uno di essi invia un beacon all'altro. Questo tipo di segnalazioni si riferisce al beaconing bidirezionale. Numerosi protocolli che utilizzano i beacon per la scoperta di vicini sono riportati in letteratura come DSDV, AMRoute, On -Demand Multicast Routing Protocol (ODMRP), AODV, Safari , e Internet MANET Encapsulation Protocol (IME). Vale la pena notare che i beacon hanno più applicazioni rispetto allo stato di monitoraggio del collegamento, che non sono qui affrontate. L' analisi dell'efficienza e dell'overhead del procedimento di trasmissione di beacon in presenza di mobilità è la principale preoccupazione. Il modello si basa esclusivamente sulle informazioni statistiche delle durate dei collegamenti fisici , che è oggetto di studi approfonditi in letteratura sia analiticamente sia sperimentalmente. Basandosi sulle statistiche della durata del collegamento fisico si possono nascondere i dettagli della mobilità del modello. Pertanto, l'approccio è abbastanza generale per coprire diversi modelli di mobilità. Inoltre, nasconde tutti gli effetti che contribuiscono alla stabilità del collegamento, come non perfettamente omnidirezionale, oppure la presenza di antenne e ostacoli nel percorso di trasmissione.

A.2 Definizioni

La durata del collegamento fisico (PLD) è definita come una variabile casuale che indica la durata di un collegamento da un nodo ad un altro nodo fisico dal momento in cui un ricevitore entra nella regione di comunica-

zione del trasmettitore al momento in cui il ricevente esce dalla regione di comunicazione.

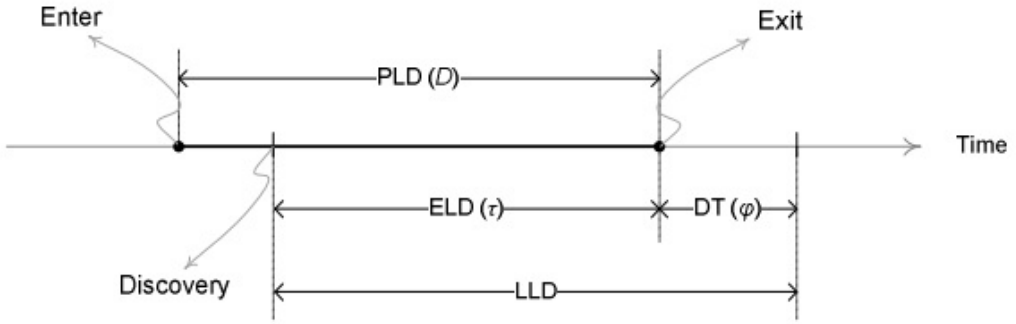

Figura A.1: Illustrazione di PLD, LLD, ELD, e dead time (DT) quando i nodi vicini si muovono nell'area di trasmissione e al di fuori.

La durata del Logical Link (LLD) è il lasso di tempo di esistenza di un legame logico, a partire dal momento in cui un nodo inserisce un nodo adiacente nel suo elenco di vicini fino a rimuovere il nodo dalla lista . Vale la pena notare che un link logico può presentarsi anche quando il nodo adiacente non è più nel raggio di trasmissione (vedere figura A.1). La durata effettiva del Link (ELD) è definita come una variabile casuale che indica l'intervallo di tempo di un collegamento efficace, a partire dal momento in cui viene rilevato il nodo adiacente e presente nella lista dei vicini fino al momento in cui il nodo adiacente si sposta fuori dalla zona di trasmissione . Il punto di tempo in cui il nodo u rileva il nodo z viene considerata un punto casuale nel tempo e indicato come tempo di rilevamento, come mostrato in figura A.1. ELD è successivamente indicato con τ . Il metodo qui proposto non dipende da nessuna particolare impostazione di rete.

Scenario di esempio: l'area di rete è un campo bidimensionale. La propagazione radio è il modello del disco in cui due nodi possono comunicare se e solo se rientrano distanza R l'una dall'altra. Il modello di mobilità è la velocità costante (CV) del modello in cui ogni nodo sceglie una direzione casuale in [0,2 π) e si muove con una velocità costante predefinita (v_c). È proposta un'espressione in forma chiusa per la p.d.f. (Probability Density Function - funzione di densità di probabilità) di ELD con il modello di CV è ottenuta come:

$$f_\tau(\tau) = (\pi^2 R^2 \tau)^{-1}[4\tau v_c R + 2(R - \tau v_c)(R + \tau v_c) ln(\frac{1}{|R - \tau v_c|}(R + \tau v_c))]$$

dove R è il raggio di trasmissione fisso per tutti i nodi. Nel resto della carta, assumiamo R = 16.221m e v_c = 1 m / s, che fornisce PLD medio di

20 secondi, se non diversamente indicato. La probability density function di PLD può essere ottenuto dalla p.d.f. di ELD:

$$f_D(\tau) = -\frac{df_\tau(\tau)}{d\tau} E[D]$$

che fornisce

$$E[D] = \pi^2 R(8v_c)^{-1}$$

A.3 Valutazioni

Un nodo può inviare o ricevere una sequenza di beacon, che è modellata come un processo stocastico. Il tempo tra due beacon consecutivi viene indicato come tempo di inter-beacon (IB). Si presume che il processo di beaconing sia un processo stazionario. Tuttavia, i tempi di inter-beacon non sono necessariamente variabili casuali iid. Per studiare l'affidabilità dei collegamenti stabiliti, la Efficace Link Duration con le procedure Beacon (ELDB o τ_B) è definita come una variabile casuale che indica la durata della collegamento efficace, che viene rilevato dal processo di beacon. La probabilità che un nodo in arrivo ad un'area di trasmissione di un nodo viene rilevata dal processo beacon prima di lasciare la zona è chiamato probabilità di successo (p_{BH}) e questa viene utilizzata come misura di efficienza di colpire un beacon. L'elemento probabilità di τ_B è dato da:

$$f_{\tau_B}(\tau)d\tau = Pr(\tau_B \in [\tau, \tau + d\tau] | BH) = p_{BH}^{-1} Pr(\tau_B \in [\tau, \tau + d\tau] \& BH)$$

dove BH sta per evento beacon hit. Così,

$$f_{\tau_B}(\tau) = \frac{1}{p_{BH}} \int_0^\infty p_{BH}(D) f_{\tau_B|D}(\tau|D) f_D(D) dD$$

dove $p_{BH}(x)$ è la probabilità condizionata di beacon hit quando D=x. Dato $p_{BH}(x)$ è ottenuta come

$$p_{BH} = \int_0^\infty p_{BH}(D) f_D(D) dD$$

Al fine di agevolare la valutazione dei $p_{BH}(x)$, una variabile casuale S è introdotta e indica l'intervallo di tempo tra la ingresso di un nodo entrante nella zona di trasmissione di un nodo e il beacon successivo, indipendentemente dal fatto che il prossimo beacon colpisca o meno. Utilizzando la definizione di S, abbiamo $p_{BH}(x) = F_s(x)$, dove F_s è la c.d.f. (Cumulative Distribution Function - funzione di distribuzione cumulativa) di S. La probabilità di un nodo che viene rilevato dal processo di beacon prima di lasciare la zona è chiamata beacon hit probability p_{BH}. Viene utilizzato come misura di efficienza di beacon colpito. L'elemento probabilità di τ_B è data da:

$$f_{\tau_B|D}(x|D) = \begin{cases} F_s(D)^{-1} f_s(D-x) & 0 \le x \le D \\ 0 & otherwise \end{cases}$$

Così si ottiene:

$$f_{\tau_B}(\tau) = E[F_s(D)]^{-1} \int_\tau^\infty f_s(D-\tau) f_D(D) dD$$

La funzione di densità di probabilità di S si ottiene dallo schema di beaconing. Se il processo di beacon è considerato come un processo punto stazionario con $\frac{1}{\tau_B}$ intensità, la p.d.f. di S è data dal calcolo

$$f_S(s) = T_b^{-1}(1 - F_{I_B}(s))$$

dove $F_{I_B}(.)$ è la distribuzione dei tempi di inter-beacon. Due casi primitivi ben noti sono il beaconing periodico ed esponenziale. Per segnalazioni periodiche, S ha una distribuzione uniforme e in caso di segnalazioni esponenziale, S ha una distribuzione esponenziale con la stessa media, come i tempi di inter-beacon. La p.d.f. di D è derivato dalle statistiche PLD, che poteva essere ottenuto analiticamente, da misure, o mediante studi di simulazione. Vale la pena notare che il PLD è più facile da misurare in scenari realistici e in esperimenti sul campo di modelli di mobilità. Ad esempio, la misurazione di PLD non necessita di informazioni sulla posizione, che è difficile da ottenere in ambienti coperti a causa dell'impossibilità di utilizzare i dispositivi GPS in tali luoghi. Segnalazione periodica: personalizzazione del modello di segnalazioni periodiche con intervallo T_B offre:

$$f_{T_B}(\tau) = (T_B p_{BH})^{-1}(F_D(\tau + T_B) - F_D(\tau))$$
$$\text{dove } p_B H = 1 - T_B^{-1} \int_0^{\tau_B} F_D(D) dD$$

I risultati per lo scenario di esempio descritto assumendo un processo periodico di segnalazioni con intervallo di beacon T_B sono illustrati nella Figura 2. Come illustrato nella figura, diminuendo T_B si riduce notevolmente la probabilità di rottura del legame precoce. D'altra parte, l'aumento del valore di TB là E[D] = 20s ha effetto trascurabile sul c.d.f. di ELDB. Sorprendentemente, dato che T_B è aumentato, E $[\tau_B]$ diminuisce e poi aumenta lentamente. Ci sono due fattori che influenzano E $[\tau_B]$ in direzione inversa. In primo luogo, con i valori più piccoli di T_B, ogni collegamento fisico è rilevato in precedenza. Quindi, il collegamento effettivo avrebbe vita più lunga. In secondo luogo, con un piccolo valore di T_B, è più probabile individuare tutti i nodi in arrivo con un PLD basso o alto, ma siccome T_B è aumentato, il tasso di nodi in arrivo mancanti con bassi PLD è aumentato anche. Più intuitivamente, quando T_B è molto piccola sia i collegamenti corti e lunghi vengono rilevati,

mentre con una grande T_B tutte i lunghi collegamenti vengono rilevati e solo una parte di brevi collegamenti. Così, la media tende verso quelli lunghi. Beaconing esponenziale: Supponiamo che il processo di beacon è un processo di Poisson, ovvero, gli intervalli di beacon sono variabili casuali esponenziali i.i.d. con media T_B. Una variabile casuale S ha un distribuzione esponenziale con la T_B media:

$$F_s(s) = 1 - \frac{1}{e^{ST_B}}$$

Beaconing bidirezionale: Al fine di studiare l'effetto di segnalazioni two way, S viene ridefinito come l'intervallo di tempo dal punto in cui due nodi possono comunicare fisicamente fino alla beacon successivo inviato da uno di loro. Così,

$$S_{twoway} = Min(S^1_{one-way}, S^2_{one-way})$$

dove $S^1_{one-way}$ e $S^2_{one-way}$ sono due variabili aleatorie iid disegnate dalla distribuzione di S in caso di beaconing one-way. La figura **??** mostra i risultati per il caso di beaconing periodico bidirezionale. I risultati mostrano che l'affidabilità degli effective links non è influenzata notevolmente utilizzando un sistema a due vie invece di uno schema unidirezionale . Inoltre, beaconing ad una via è vicino a quello bidirezionale in termini di beacon hit efficiency per piccoli valori di intervalli di beacon e si avvicina a beaconing one way con la doppia velocità di trasmissioni beacon come intervalli di beacon sono aumentati. Tuttavia, non è più efficiente di un regime unidirezionale di beaconing con il rate doppio. Questo risultato conferma che un clusterbased o una rete cellulare che rende solo la testa cluster o stazione base di inviare beacon è molto più efficiente di un caso che tutti i membri del cluster possono anche inviare beacon (considerando che in quest'ultimo caso il sovraccarico di energia totale di beacon trasmissioni viene moltiplicato per il numero medio di cluster di membri). In altre parole , al fine di aumentare la probabilità di beacon hit è più efficiente raddoppiare il beacon rate della stazione base invece di fare si che i membri del cluster inviino beacon.

Tempo di attivazione del collegamento: in alcune applicazioni, ad esempio in Networking Bluetooth, c'è bisogno di tempo per impostare le connessioni, che potrebbe essere indagato aggiungendo il tempo per la configurazione della connessione (TS) alla variabile casuale S. La Figura 5 mostra l'impatto del tempo di setup sulla probabilità di fare una connessione con un prossimo arrivo. Come mostrano i risultati, aumentando il tempo di configurazione si riduce notevolmente la beacon hit efficiency. Tuttavia, la durata efficace del collegamento non viene ridotta notevolmente ed è anche migliorata in alcuni scenari . questo è a causa del fatto che un valore elevato di tempo di setup

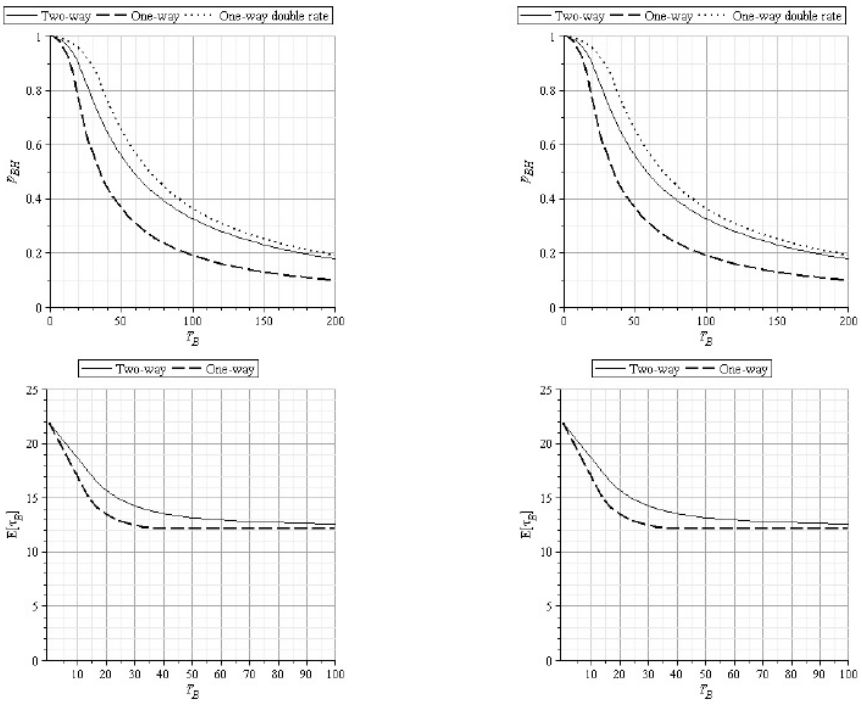

Figura A.2: Probabilità di beacon hit (top) e media ELDB (bottom) per beaconing periodico a una e due vie.

filtrerà tutti i collegamenti brevi . Nei casi in cui l'affidabilità dei link ha una priorità più alta della beacon hit , ad esempio in alcune reti dense , si può usare il tempo di setup virtuale per ottenere collegamenti efficaci più affidabili . In questo caso , un collegamento viene prima rilevato al tempo t0 e quindi viene convalidato a $t_0 + T_S$. Se è ancora vivo , allora è considerato come un efficace collegamento (supponendo che sullo stato dei collegamenti non ha le notevoli fluttuazioni durante questo periodo) .

A.3.1 Beaconing ottimale

Beaconing ottimale è qui definito come uno che dà la più alta beacon hit efficency per un dato tasso di segnalazioni. Primo, il discovery delay ($\ni$) è definito come l'intervallo dal punto di tempo in cui un ricevitore entra nella regione di comunicazione di un trasmettitore al momento in cui il ricevitore è scoperto dal trasmettitore . La funzione di distribuzione cumulativa condizionale del ritardo scoperta è ottenuta come:

$$F_{\eta|D}(\eta|D) = Pr(S \leq \eta|S \leq D) = \frac{Pr(S \leq \eta and S \leq D}{Pr(S \leq D}$$

quindi

$$f_{\eta|D} = \begin{cases} p_{BH}(D)^{-1} f_s(\eta) & \eta \leq D \\ 0 & otherwise \end{cases}$$

Otteniamo

$$f_\eta(\eta) = \frac{1}{p_{BH}} \int_0^\infty p_{BH}(D) f_{\eta|D}(\eta|D) f_D(D) dD = p_{BH}^{-1} \int_\eta^\infty f_s(\eta) f_D(D) dD =$$
$$p_{BH}^{-1} f_s(\eta)(1 - F_D(\eta)) = (p_{BH} T_B)^{-1}(1 - F_{I_B}(\eta))(1 - F_D(\eta))$$

La probabilità di scoprire un nodo vicino nel tempo η_0 dopo essere entrato nell'area di trasmissione è data da:

$$Pr(\eta < \eta_0 \text{and} BH) = p_{BH} x F_\eta(x)$$

In particolare, quando η_0 tende all'infinito è uguale a p_{BH}. Quindi,

$$p_{BH} = T_B^{-1} \int_0 \infty (1 - F_{I_B}(\eta))(1 - F_D(\eta)) d\eta$$

Nella equazione precedente, $1 - F_D(\eta)$ e $1 - F_{I_B}(\eta)$ sono due funzioni cdf complementari, che valgono 1 quando n=0 e sono decrescenti. $1 - F_{I_B}(\eta))$ non è influenzata dallo schema di beaconing. D'altra parte, si mantiene fissa il tempo medio di inter-beacon. Quindi -formula è fisso. In altre parole, il moltiplicando T_B^{-1} e l'area sottesa da $1 - F_{I_B}(\eta)$ sono fissati. Così, dato che $1 - F_D(\eta)$ è una funzione decrescente, p_{BH} è massima quando l'area sottesa dalla funzione $1 - F_{I_B}(\eta)$ è condensato in aree con un n inferiore. Quindi lo schema di beacon ottimo è il beaconing mono-direzionale periodico.

A.4 Conclusioni su beaconing

Nel lavoro presentato , il beaconing è studiato poiché è uno degli strumenti importanti ampiamente utilizzati in reti radiomobili per scoprire nodi vicini. Il materiale presentato non è basato su una particolare modello di mobilità. Invece, si basa sulle p.d.f. della durata fisica del collegamento, che è stato ampiamente studiato. Le p.d.f. della durata del collegamento fisico astrae sia la mobilità e tutti gli effetti che contribuiscono alla stabilità del collegamento. Il regime beacon di beacon periodico viene confrontato con il caso esponenziale di intervalli di beacon e si è dimostrato essere più efficiente nella scoperta di nodi adiacenti e meno ridondante rispetto all'istituzione di collegamento efficaci. L'effetto di impostazione della connessione temporale che è un parametro inevitabile in tutti i link - su Radio la disponibilità dei collegamenti, è studiata in un approccio sistematico e generale. É anche dimostrato che le segnalazioni periodiche è il sistema di segnalazioni ottimale in termini di efficienza.

Appendice B

Routing nelle reti Ad Hoc e reti Mobile Ad hoc

Le risorse limitate delle MANETs hanno reso necessario il design di *strategie di routing* [10] efficienti ed affidabili. Sono richieste strategie di routing intelligenti per utilizzare efficientemente le risorse limitate mentre, allo stesso tempo, esse devono essere in grado di adattarsi ai cambiamenti delle condizioni della rete come la dimensione, la densità di traffico e il partizionamento della rete. In parallelo a questo, il protocollo di routing deve fornire differenti livelli di QoS per differenti tipi di applicazioni e utenti. Prima del crescente interesse nelle reti wireless, nelle reti cablate venivano utilizzati principalmente due algoritmi. Questi algoritmi sono comunemente riferiti come **link-state** e **distance-vector**. Nel primo tipo di routing, ogni nodo mantiene una vista aggiornata della rete facendo un broadcast periodico di informazioni sullo stato dei collegamenti tramite un'operazione di flooding a tutti gli altri nodi. Quando ogni nodo riceve un pacchetto aggiornato esso esegue un'operazione di update alla sua view della rete e alle sue informazioni link-state applicando un algoritmo di *shortest path* per decidere il nodo next-hop per ogni destinazione. Nel secondo tipo di routing, per ogni destinazione x, ogni nodo i mantiene un set di distanze $D_{ij}(x)$, dove il range di j spazia sui vicini del nodo i. Il nodo i seleziona un vicino k per essere il next-hop per x se $D_{ij}(x) = min_j(D_{ij}(x))$. Questo permette ad ogni nodo di selezionare il percorso più breve per ogni destinazione. Le informazioni nel vettore di distanze sono aggiornate ad ogni nodo da una disseminazione periodica della corrente distanza più breve stimata per ogni nodo. Gli algoritmi tradizionali di link-state e distance-vector non sono molto scalabili nelle MANETs di larghe dimensioni. Questo a causa del consumo significativo della banda disponibile e crescente contesa del canale provocato dagli aggiornamenti periodici o frequenti dei percorsi di route. Inoltre questo fenomeno può ri-

chiedere ad ogni nodo una frequente ricarica di energia. Per superare questi problemi associati agli algoritmi di link-state e distance-vector, un numero di protocolli di routing sono stati proposti per le MANETs. Questi protocolli possono essere classificati in tre differenti gruppi illustrati in figura B.1

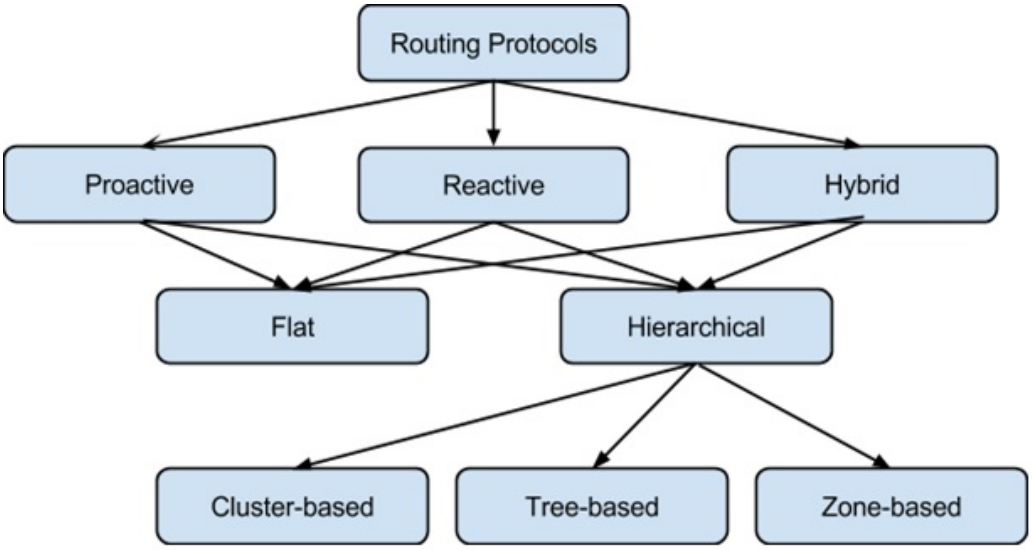

Figura B.1: Schema riassuntivo dei protocolli di routing.

Nei protocolli di routing *proattivi*, i percorsi verso tutte le destinazioni (o parti della rete) sono determinati dalla fase di start up e mantenuti utilizzando un processo di aggiornamento di route periodico. Nel protocolli *reattivi*, i percorsi sono determinati quando sono richiesti dalla source utilizzando un processo di route discovery. I protocolli di routing *ibridi* combinano le proprietà base delle prime due classi algoritmiche in una. Successivamente vengono descritte in dettaglio le strutture menzionate.

B.1 Protocolli di Routing Proattivo

I protocolli di routing proattivo mantengono informazioni globali o parziali di routing. Le informazioni di routing sono di solito mantenute in un numero differente di tabelle. Queste tabelle sono aggiornate periodicamente oppure quando la topologia della rete cambia. Il vantaggio del processo di discovery tramite il proactive route è che il ritardo end-to-end è ridotto durante la trasmissione dei dati, quando comparata alla determinazione dei percorsi reattivi. Le simulazioni hanno evidenziato alti livelli di *throughput* di dati per differenti protocolli proattivi e significative diminuzioni di ritardi rispetto ai

protocolli on-demand (come DSR) per le reti composte da più di 50 nodi con alti livelli di traffico. D'altra parte, in reti piccole nelle quali vengono utilizzate applicazioni in real-time (come ad esempio le videoconferenze), dove il ritardo end-to-end deve essere minimizzato, i protocolli di routing proattivo possono essere benefici. Le prossime sezioni descrivono un numero di differenti strategie di route update proposti in letteratura per implementare il routing proattivo.

B.1.1 Update Globali

I protocolli di routing proattivo che usano gli aggiornamenti globali di route sono basati sugli algoritmi di link-state e distance-vector, i quali sono stati originariamente progettati per le reti cablate. In questi protocolli, ogni nodo scambia periodicamente le sue tabelle di route con ogni altro nodo della rete. Per fare questo, ogni nodo trasmette un messaggio di update ogni T secondi. Usando questi messaggi di update, ogni nodo mantiene le proprie tabelle di routing, le quali memorizzano il più recente o il migliore percorso ad ogni destinazione conosciuta. Lo svantaggio degli update globali è che utilizzano una quantità significativa di banda in quanto non prendono nessuna precauzione per ridurre il controllo di overhead. Come risultato di ciò, il throughput dei dati può soffrire in modo significativo, specialmente quando il numero di nodi nella rete cresce. Due di questi protocolli sono DSDV [1] e WRP [2].

B.1.2 Update Localizzati

Per ridurre l'overhead negli update globali, un numero di strategie per gli aggiornamenti localizzati sono introdotti nei protocolli come GSR [3] e FSR [4]. In queste strategie, la propagazione del route update è limitata ad una regione localizzata. Per esempio, nel protocollo GSR, ogni nodo scambia informazioni di routing con i suoi vicini solamente, e con ciò si elimina il flooding di pacchetti utilizzando nella metodologia dell'aggiornamento globale. FSR è un diretto discendente di GSR. Questo protocollo mira ad incrementare la scalabilità di GSR aggiornando i nodi vicini ad una frequenza maggiore rispetto ai nodi che sono allocati più lontani. Per definire il concetto di "regione vicina", FSR introduce il "fisheye scope". Esso copre un set di nodi che possono essere raggiunti entro un certo numero di hopes da un nodo centrale. I messaggi di update con contatore di hop maggiore sono spediti

[1] Destination-Sequenced Distance Vector routing
[2] Wireless Routing Protocol
[3] Global State Routing
[4] Fisheye State Routing

con minor frequenza. Questo riduce l'accuratezza del routing nelle locazioni più remote; comunque, esso riduce significativamente la quantità di routing overhead disseminato nella rete. L'idea dietro questo protocollo è che i pacchetti di dati più vicini alla destinazione hanno una maggiore accuratezza nel routing. Quindi, se i pacchetti conoscessero approssimativamente in quale direzione viaggiare, nell'avvicinarsi alla destinazione, loro viaggerebbero su un route più accurato e avrebbero probabilità più alte di raggiungere questa destinazione.

B.1.3 Updates Mobility Based

Un'altra strategia che può essere usata per ridurre il numero di update packet è chiamata DREAM. Gli autori di tale metodo affermano che l'overhead di routing può essere ridotto rendendo il rate al quale i route updates sono spediti proporzionale alla velocità con la quale ogni nodo viaggia. Quindi, i nodi che viaggiano ad una velocità maggiore producono più update packets rispetto a quelli che possiedono velocità più ridotte. Il vantaggio di questa strategia è che nelle reti con bassa mobilità, questo approccio di aggiornamento può produrre un numero inferiore di update packet rispetto all'utilizzo di approcci di update statici come DSDV. In modo simile a FSR, il protocollo DREAM spedisce gli aggiornamenti con maggior frequenza ai nodi più vicini rispetto a quello dislocati a distanze maggiori.

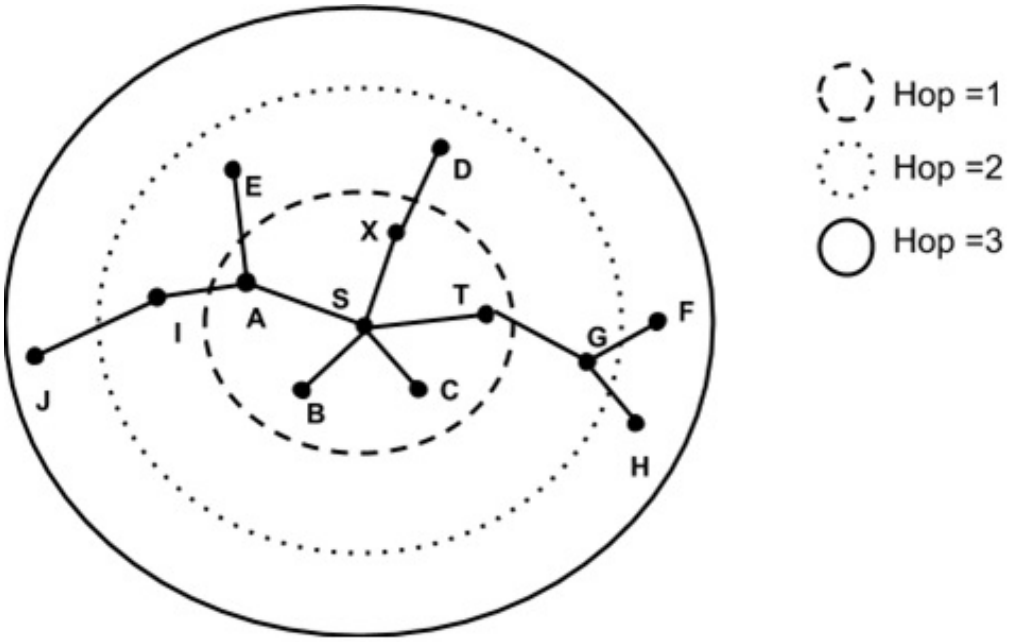

Figura B.2: Schema per il funzionamento del caso update mobility based in protcolli di routing proattivo.

B.1.4 Updates Displacement Based

Gli update *displacement-based* sono una strategia di route update che mira a disseminare i pacchetti di aggiornamento dei percorsi nella rete solo quando questi sono richiesti, al posto di utilizzare aggiornamenti periodici puri. Questo è raggiunto rendendo il rate con il quale gli aggiornamenti sono spediti proporzionale al rate con il quale un nodo migra dalla sua locazione ad una nuova locazione. Così, quando un nodo cambia locazione di una distanza di soglia, un route update è trasmesso all'interno della rete. Il posizionamento richiesto può essere misurato utilizzando il Global System Position (GPS). Il vantaggio di questa strategia su aggiornamenti mobility-based è che gli update sono spediti solo quando c'è un cambiamento topologico, che può alterare la connettività della rete.

B.1.5 Updates condizionali o Event Driven

Il numero di update ridondanti può essere ridotto utilizzando una strategia di aggiornamento condizionale o event-drive. In questa strategia, un nodo spedisce un update se certi eventi si verificano in un dato momento. Alcuni eventi che possono generare un update sono quando un link diventa invalido

o quando un nuovo nodo si unisce alla rete (oppure quando un nuovo vicino è scoperto). Il vantaggio di questa strategia è che se la topologia della rete o le condizioni non cambiano, allora non viene spedito nessun pacchetto di update. Quindi, la disseminazione di update periodici periodici nella rete viene eliminata.

B.2 Protocolli di Routing Reattivo

I protocolli di routing reattivo determinano e mantengono le informazioni di routing per i nodi attivi che richiedono l'invio di dati ad una particolare destinazione. L'operazione di route discovery di solito avviene attraverso il flooding di pacchetti RREQ (Route Request) attraverso la rete. Quando viene raggiunto un nodo con un route alla destinazione, una RREP (Route Reply) è spedita indietro alla sorgente il link reversal se la richiesta ha viaggiato attraverso link bidirezionali o tramite il piggy-backing del percorso in un pacchetto di risposta via flooding. I protocolli reattivi possono essere classificati in due categorie: source routing e hop-by-hop routing.

B.2.1 Protocolli di Source Routing

In questa tipologia di routing (DSR e SSA), ogni pacchetto di dati trasporta gli indirizzi completi di sorgente e destinazione. Quindi, i nodi intermediari attuano una operazione di forward a questi pacchetti in accordo alle informazioni mantenute nell'header di ogni pacchetto. Ciò significa che i nodi intermediari non hanno bisogno di mantenere aggiornate le informazioni di routing per ogni nodo attivo. Inoltre, i nodi non hanno bisogno di mantenere la connettività con i vicini attraverso dei messaggi di beaconing periodici. Il maggior svantaggio di questo tipo di protocolli è che in reti di grandi dimensioni le performance non sono buone. Questo è dovuto a due ragioni principali:

1. al crescere del numero di nodi nella rete, cresce anche la probabilità di route failure;

2. al crescere del numero di nodi intermedi in ogni percorso, cresce anche la quantità di overhead trasportata in ogni pacchetto.

Quindi, nei network di grandi dimensioni con un livello significativo di multihopping e di mobilità, questo protocollo potrebbe non scalare adeguatamente.

B.2.2 Protocolli di Routing Hop by Hop

Nel routing hop-by-hop (anche conosciuto come routing point-to-point) AODV [5], ogni pacchetto di dati trasporta solamente l'indirizzo di destinazione e l'indirizzo dell'hop successivo. Quindi, ogni nodo intermedio nel percorso verso la destinazione utilizza la sua tabella di routing per compiere il forward di ogni pacchetto di dati verso la destinazione. Il vantaggio di questa strategia è che i routes sono adattabili al cambiamento dinamico dell'ambiente delle MANET perchè ogni nodo può aggiornare la sua tabella di routing quando riceve informazioni nuove sulla topologia e quindi può trasmettere i pacchetti su percorsi migliori. Utilizzando percorsi sempre aggiornati significa anche che sono richiesti minori operazioni di ricalcolamento dei percorsi durante la trasmissione. Lo svantaggio di questa strategia è che ogni nodo intermedio deve mantenere le informazioni di routing per ogni nodo attivo, ed ogni nodo può richiedere di essere a conoscenza dei suo vicini attraverso l'uso di messaggi di beaconing.

B.3 Protocolli di Routing Ibrido

I protoccoli di routing ibridi sono una nuova generazione di protocolli che sono sia proattivi sia reattivi. Essi sono progettati per aumentare la scalabilità permettendo ai nodi che si trovano in prossimità tra di loro di cooperare al fine di formare una sorta di backbone e di ridurre l'overhead dovuto al route discovery. Questo è inizialmente ottenuto mantenendo proattivamente percorsi di route ai nodi vicini e determinando i percorsi dei nodi più lontani utilizzando una strategia di route discovery. Molti protocolli ibridi proposti fino ad oggi sono zone-based, ovvero la rete è partizionate o vista come un numero di zone da ogni nodo. Altri protocolli raggruppano i nodi in alberi o clusters. Questa sezione descrive differenti protocolli di routing ibridi proposti per le MANETs.

B.3.1 Protocolli di Zone Routing (ZRP)

In ZRP, i nodi hanno una zona di routing che definisce un range (in hops) che ogni nodo usa per mantenere la connettività proattiva della rete. Quindi, per i nodi entro la routing zone, i percorsi sono disponibili immediatamente. Per i nodi che si trovano al di fuori della routing zone, i percorsi sono determinati on-demand (per esempio in modo reattivo). Il vantaggio di questo protocolli è che riduce significativamente l'overhead di comunicazione se comparato con

[5] Ad-hoc On-demand Distance Vector

i protocolli proattivi puri. Esso riduce anche il ritardo rispetto ai protocolli reattivi puri, come DSR, permettendo la scoperta più veloce dei percorsi. Questo perchè, per determinare un percorso verso un nodo oltre la zona di routing, il routing deve viaggiare verso un nodo che si trova sui confini (edge della routing zone) della destinazione richiesta, perchè i nodi di confine mantengono proattivamente i percorsi verso la destinazione. Lo svantaggio di ZRP è che se il raggio di ogni routing zone è troppo largo, il protocollo può attuare un comportamento simile a un protocollo proattivo puro; mentre se il valore del raggio è settato ad un valore troppo basso, allora ZRP si comporta come un protocollo reattivo. Può essere importante ottimizzare il valore del raggio in ZRP in ogni rete specifica per soddisfare al meglio le sue caratteristiche (come la densità dei nodi).

B.3.2 Zone-Based Hierachical Link State (ZHLS)

ZHLS, contrariamente a ZRP, utilizza un struttura gerarchica. In particolare, la rete è divisa in zone non sovrapposte tra di loro, e ogni nodo è caratterizzato da un ID e da un ID di zona, che è calcolato tramite un GPS. La topologia gerarchica è costruita su due livelli:

1. livello di topologia del nodo,

2. livello di topologia della zona.

In ZHLS, la gestione delle locazioni è stata semplificata. Questo perchè nessun cluster head o location manager è utilizzato per coordinare la trasmissione dei dati. Ciò significa che non c'è nessun overhead di processing associato alla selezione di cluster head o location manager quando comparato ai protocolli HSR [6], MMWN [7], CGSR [8]. Questo significa anche che possono essere evitati singoli punti di rottura e colli di bottiglia. Un altro vantaggio di di ZHLS è che esso riduce l'overhead di comunicazione se comparato ai protocolli come DSR e AODV. In questo protocollo, quando è richiesto un percorso ad una destinazione remota (per esempio una destinazione che si trova in un'altra zona), il nodo sorgente fa un broadcast di una richiesta zone-level location a tutte le altre zone. In questo modo, esso genera un overhead significativamente inferiore se comparato con l'approccio flooding nei protocolli reattivi. In più, in ZHLS, il path di routing è adattabile ai cambiamenti della topologia in quanto solamente l'ID di nodo e l'ID di zona

[6] Hierarchical state routing
[7] Multimedia support in mobile wireless networks
[8] Clusterhead Gateway Switch Routing protocol

della destinazione sono richiesti per il routing. Questo significa che non sono richiesti ulteriori location search mentre la destinazione non migra verso un'altra zona. Tuttavia, nei protocolli reattivi, nessuna rottura di collegamento intermedia può invalidare il route e può iniziare un'altra procedura di route discovery. Lo svantaggio di ZHLS è che tutti i nodi devono avere una mappa di zona statica preprogrammata per funzionare. Questo può non essere possibile in applicazione dove i confini geografici della rete sono dinamici. Tuttavia, è altamente adattabile a topologie dinamiche e genera un overhead inferiore rispetto ai protocolli reattivi puri, ed è quindi scalabile bene a reti di grandi dimensioni.

B.3.3 Protocollo Distributed Spanning Trees Based Routing (DST)

In DST, i nodi della rete sono raggruppati in un numero di alberi. Ogni albero ha due tipi di nodi: (1) nodo di route e (2) nodo interno. La radice controlla la struttura dell'albero e se l'albero può essere unito con un altro albero, e il resto dei nodi all'interno dell'albero sono nodi regolari. Ogni nodo può essere in uno di questi tre stati differenti:

1. Router
2. Merge
3. Configure

dipendentemente dal tipo di task che deve svolgere. Per determinare un route, DST propone due differenti strategie di routing:

- flooding-tree ibrido (HTF),
- spanning tree distribuito (DST).

In HTF, il controllo dei pacchetti è spedito a tutti i vicini e ai bridge adiacenti (un bridge è formato quando due nodi di spanning tree differenti sono nel radio range) nello spanning tree, dove ogni pacchetto è mantenuto per un periodo di tempo chiamato holding time. L'idea dietro l'holding time è che all'aumentare della connettività e stabilità della rete, può essere utile per bufferizzare e spedire pacchetti quando la connettività della rete cresce nel tempo. In DST, il controllo dei pacchetti è disseminato dalla sorgente e rispedito in broadcast lungo i lati dell'albero. Quando un controllo scende fino a un nodo foglia, il pacchetto di controllo può essere spedito verso il basso dell'albero o ai bridge adiacenti. Il principale svantaggio degli algoritmi di

DST è che si basa si un nodo radice per configurare l'albero, e questo crea un singolo punto di rottura. Inoltre, l'holding time utilizzato per bufferizzare i pacchetti può introdurre ritardi aggiuntivi nella rete.

Bibliografia

[1] http://www.bluetooth.com/Pages/Bluetooth-Home.aspx

[2] http://www.oracle.com/technetwork/systems/bluetooth2-156149.html

[3] https://code.google.com/p/bluecove/

[4] http://bluecove.org/bluecove/apidocs/overview-summary.html

[5] https://www.eclipse.org/

[6] https://www.dropbox.com/developers/core/start/java

[7] https://developer.bluetooth.org/TechnologyOverview/Pages/OBEX.aspx

[8] http://developer.android.com/guide/topics/connectivity/bluetooth.html

[9] Abbas Nayebi, Gunnar Karlsson Royal Institute of Technology (KTH) Stockholm, Sweden. *Beaconing in Wireless Mobile Networks.*

[10] Jie Wu. *Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks.*