Alma Mater Studiorum · Università di Bologna

facoltà di scienze matematiche, fisiche e naturali

Corso di Laurea in Scienze di Internet

ELABORAZIONE CONDIVISA DI TESTI CON ANDROID

Tesi di Laurea in Programmazione di Internet

Relatore:
Chiar.mo Prof.
Antonio Messina

Presentata da:
Francesco Bevilacqua

I Sessione
Anno Accademico 2009-2010

A Livia

Che mi è sempre stata vicina

Alla mia famiglia

Che mi accompagna in ogni momento della mia vita

E ai miei amici...

Indice

Introduzione

In questi ultimi 10 anni Internet ha rivoluzionato le nostre vite. Ad oggi si contano 1.5 miliardi di utenti, che in ogni momento della giornata accedono a questa enorme fonte di informazione.
Parallelamente alla sua crescita sta iniziando una seconda grande rivoluzione grazie all'utilizzo di dispositivi mobili. Siamo partiti da un semplice telefono cellulare e ora ci troviamo nelle nostre mani uno strumento che è in grado di fornire ogni tipo di servizio. Possiamo conoscere in ogni momento la nostra posizione, possiamo inviare e ricevere e-mail, acquisire immagini e filmati, per poi pubblicarle sul nostro sito.
Queste due rivoluzioni abbattono molti limiti, permettendo di soddisfare qualsiasi bisogno con la nascita di nuove applicazioni. In un contesto come questo sono nate diverse piattaforme per dispositivi mobili, ognuna con un propria caratteristica, una propria storia e un proprio linguaggio. Una di queste si chiama Android, creata da Google e dalla Open Handset Alliance ed completamente Open Source.
Il lavoro svolto in questa tesi è stato quello di studiare l'ambiente Android e le metodologie connesse all'attività di videoscrittura collaborativa in rete, col fine di realizzare MyNotes. Questa applicazione consente la modifica concorrenziale di un breve documento di testo da parte di due utenti su dispositivi mobili.
La tesi è suddivisa nei seguenti capitoli:

- nel primo sarà descritta la storia di Android e la sua architettura;
- nel secondo sarà fornita al lettore una descrizione della collaborazione in rete;
- il terzo capitolo sarà dedicato all'analisi del progetto;

- il quarto è dedicato alla fase progettuale di MyNotes presentando i diagrammi UML;
- nel quinto capitolo saranno descritte le funzionalità più importanti dell'applicazione realizzata, con riferimenti al codice.

Capitolo 1

Introduzione ad Android

Negli anni passati gli utenti potevano accedere ad informazioni e applicazioni solo attraverso un PC. Oggi è possibile accedervi attraverso altri dispositivi, che hanno la fondamentale caratteristica di essere mobili e di dimensioni molto più ridotte. Un Personal Computer sta sempre più diventando personale, non perchè l'utente che ne fa utilizzo sia l'unico, ma perchè esso ci accompagna ovunque noi andiamo. Tutto ciò di cui abbiamo bisogno può stare completamente nella nostra tasca.
I dispositivi mobili stanno diventando dei veri e propri PC portatili, dove la funzione di telefono è solo una delle tante disponibili.
Gli sviluppatori hanno quindi la possibilità di creare e sviluppare applicazioni che sfruttino le caratteristiche di questi dispositivi. Purtroppo questi sono limitati in quantità di memoria, potenza di CPU e alimentazione, rispetto ad un PC di media potenza. Tuttavia si stima che la potenza di calcolo di un cellulare di nuova generazione sia paragonabile a quella di un PC di 8 o 10 anni fa.
In questo contesto i principali costruttori di cellulari hanno messo a disposizione degli sviluppatori i propri sistemi operativi. Ogni SO ha il proprio ambiente di sviluppo, i propri tool e il proprio linguaggio di programmazione. Purtroppo c'è da notare che nessuno di questi si è affermato come standard. Per esempio, per realizzare un'applicazione nativa per iPhone è necessario disporre di un sistema operativo Mac OS X, su cui l'iPhone si basa, oltre che ad avere la conoscenza del linguaggio Objective-C. Questo agli occhi di alcuni sviluppatori può sembrare abbastanza restrittivo.

Altro esempio, se si vuole sviluppare un'applicazione per un dispositivo Nokia, basato sul sistema operativo Symbian, è necessario utilizzare come linguaggio un dialetto del C++. Un'altro esempio è dato da Windows Mobile di Microsoft, per il quale i linguaggi di programmazione possono essere più di uno: dal Visual Basic, .net al C#. Oltre a questi ve ne sono altri, tra cui un insieme di sistemi operativi proprietari la cui conoscenza è spesso limitata ai soli vendor.
È quindi ovvio che uno sviluppatore di applicazioni debba avere una vasta conoscenza di tutti questi diversi ambienti e teconologie. Questo perchè a seconda del tipo di sistema operativo, è necessario acquisire la conoscenza dell'ambiente, della piattaforma e del linguaggio. C'è bisogno quindi di una standardizzazione, obbiettivo che Google e la Open Handest Alliance si sono prefissati con la creazione di Android[1].

1.1 Che cos'è Android

Android è un vero e proprio stack di strumenti e librerie volte alla realizzazione di applicazioni mobili[1]. L'obiettivo è quello di fornire tutto ciò di cui un operatore, un vendor di dispositivi o uno sviluppatore hanno bisogno per raggiungere i propri obiettivi. Rispetto agli ambienti citati in precedenza, Android ha la fondamentale caratteristica di essere open, dove il termine assume diversi significati:

- Android è open in quanto utilizza tecnologie open, prima fra tutte il kernel di Linux nella versione 2.6;
- Android è open in quanto le librerie e le API che sono state utilizzate per la sua realizzazione sono esattamente le stesse che gli sviluppatori utilizzeranno per creare e le proprie applicazioni. Non ci sono quindi limiti alla personalizzazione dell'ambiente se non per alcuni casi legati ad aspetti di sicurezza nell'utilizzo, ad esempio, le funzionalità del telefono;
- Android è open in quanto il suo codice è open source, consultabile da chiunque voglia contribuire a migliorarlo, documetarlo o sia semplicemente curioso di scoprirne il funzionamento. La licenza scelta dalla Open Handset Alliance è la Open Source Apache License 2.0, che permette ai diversi vendor di costruire su Android

le proprie estensioni, anche proprietarie senza legami che potrebbero limitarne l'utilizzo. In pratica non bisogna pagare nessuna royalty per l'adozione di Android sui propri dispositivi.

1.1.1 Le origini e la storia

Ogni tecnologia nasce da una esigenza. Android è nato per fornire una piattaforma aperta e per imporsi come standard per la realizzazione di applicazioni mobili.
Google non ha realizzato Android da zero, infatti nel 2005 ha acquistato la Android Inc. con i principali realizzatori che hanno poi fatto parte del team di progettazione di questa piattaforma. Nel 2007 le principali aziende nel mondo della telefonia hanno dato origine alla Open Handest Alliance (OHA), di cui fanno parte, oltre a Google:

- Motorola, Samsung, Sony-Ericsson, HTC, Asus e Toshiba come produttori di dispositivi;
- Sprint-Nextel, Vodafone, T-Mobile e altri operatori telefonici;
- Intel, Texas Instruments e NVIDIA come costruttori di componenti.

Le aziende citate sono solo alcune che compongono questa grande alleanza[1]. L'obiettivo comune è quello di creare una piattaforma open in grado di tenere il passo del mercato senza il peso di royalties che ne possano frenare lo sviluppo.
Sempre nel 2007 uscì la prima versione del Software Development Kit (SDK)[2], che ha consentito agli sviluppatori di iniziare ad esplorare la nuova piattaforma e realizzare le prime applicazioni sperimentali. Nei primi mesi del 2008 hanno anche potuto essere testate sul primo dispositivo reale, ovvero il G1 della T-Mobile.

Nello stesso anno, ma ad ottobre, è stato fatto un passo importante, rilasciando il sorgente della piattaforma con la licenza Apache[3] ed è stata annunciata la release

[1]Per una lista completa della aziende si può consultare sito ufficiale[9].

[2]Tradotto in italiano 'pacchetto di sviluppo per applicazioni', e sta a indicare un insieme di strumenti per lo sviluppo e la documentazione di software.

[3]Licenza di software libero non copyleft scritta dalla Apache Software Foundation (ASF) che obbliga gli utenti a preservare l'informativa di diritto d'autore e d'esclusione di responsabilità nelle versioni modificate[8].

Figura 1.1: Il Development Phone 1[10].

candidate dell'SDK 1.0.

Nell'Aprile 2009 venne rilasciata la versione 1.5 dell'SDK detta anche Cupcake. La principale novità è stata l'introduzione della tastiera virtuale, liberando così i produttori di hardware dal vincolo della realizzazione di un tastiera fisica. Altre importanti novità della versione 1.5 sono state la possibilità di aggiungere widget alla home del dispositivo e i live folder.

Il 16 settembre, sempre del 2009, è stato poi rilasciato l'SDK nella versione 1.6, con altre diverse importanti novità sia a livello utente sia a livello API. La più importante di queste fu il Quick Search Box, che è un'area di testo nella home del dispositivo, in cui gli utenti possono ricercare informazioni, come ad esempio i contatti o il meteo. Ciò è stato possibile grazie alla riprogettazione del framework di ricerca impostato nelle versioni precedenti. Altra importante novità introdotta fu l'integrazione di Pico, ovvero un potente sintetizzatore vocale con accenti che riflettono le principali lingue tra cui l'italiano. Dopo poche settimane dal rilascio della versione 1.6 è stato il turno della 2.0, la quale non aggiungeva grosse novità rispetto alla precedente.

Il 12 gennaio 2010 è stato rilasciato l'Android SDK 2.1 (nome in codice Eclair), dove la principale novità era l'attesa possibilità di effettuare il multi-touch sul proprio schermo. Infine il 20 Maggio 2010 al Google I/O conference è stato rilasciato l'Android SDK 2.2,

nome in codice Froyo[10].

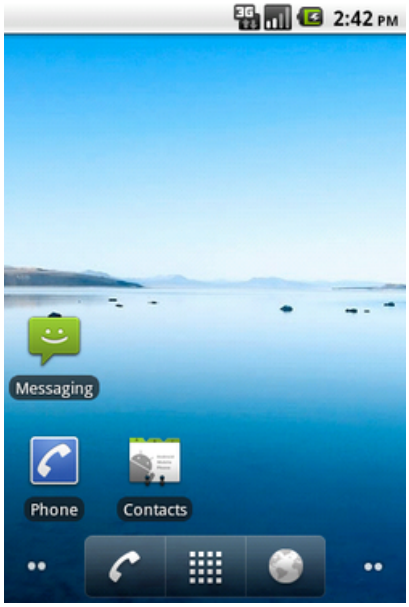


Figura 1.2: Schremata principale di Android Froyo[10].

Questa nuova versione presenta importanti aggiornamenti:

- nuovo kernel Linux 2.6.32;
- nuovo compilatore JIT[4];
- thethering Wi-fi nativo per utilizzare il dispositivo come Hotspot Wireless;
- nuova veste grafica;
- completa integrazione con Adobe Flash Player 10.1 e Adobe AIR;
- possibilità di installare le applicazioni sulla memoria SD (feature molto attesa dalla community mondiale);

[4]Compilatore just-in-time o JIT permette un tipo di compilazione, conosciuta anche come traduzione dinamica, con la quale è possibile aumentare le performance dei sistemi di programmazione che utilizzano il bytecode, traducendo il bytecode nel codice macchina nativo in fase di run-time.

- aggiornamento automatico Over-the-Air[5] delle Applicazioni.
- nuove API[6] per gli sviluppatori, tra cui le OpenGL ES 2.0;

Tutte queste novità hanno portato la piattaforma ad ottenere una maggiore performance e fluidità rispetto alla versione 2.1 Eclair.

1.1.2 Android e Java

Android fornisce un SDK in grado di facilitare lo sviluppo delle applicazioni. Siccome la fortuna di un ambiente è legato al numero di applicazioni disponibili per l'ambiente stesso, è nell'interesse di Google, al fine di promuovere la piattaforma, fornire agli sviluppatori tutti gli strumenti necessari. Android non usa un linguaggio nuovo, che gli sviluppatori sarebbero obbligati ad imparare, bensì Java, noto linguaggio di programmazione della Sun Microsystems.
Nel caso fosse stata scelta l'opzione di creare un nuovo linguaggio, Google avrebbe dovuto realizzare delle specifiche, un compilatore, un debugger, degli IDE opportuni, una documentazione e delle librerie idonee. La scelta di Java ha però generato un contrasto con quella che è la natura open di Android. I dispositivi che intendono adottare la Virtual Machine (VM) associata all'ambiente J2ME (quindi una JVM), devono pagare una royalty, caso in contrasto con la licenza di Apache citata in precedenza. Sarebbe dire che Android può essere liberamente utilizzato, però funziona solo se si dispone della VM di Sun, la quale prevede il pagamento di una royalty.
Come citato nell'introduzione, Android ha come obiettivo quello di creare delle applicazioni mobili in grado di interagire con l'utente in modo efficace. È quindi indispensabile che le diverse applicazioni in esecuzione in un dispositivo Android, vengano eseguite nel modo migliore possibile dal punto di vista dell'utente e della modalità di interazione con il dispositivo. Questo aspetto fondamentale nello sviluppo di tutte le applicazioni mobili, prende il nome di responsiveness.

[5]Per over-the-air si intende la modalità di aggiornamento del firmware di un dispositivo elettronico via etere, da satellite o ponte radio televisivo.

[6]Le Application Programming Interface API (Interfaccia di Programmazione di un'Applicazione), sono ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per un determinato compito.

Al fine di far comprendere meglio questo aspetto porto come esempio la Apple. Le applicazioni, prima di essere pubblicate per iPhone e iPod Touch, vengono testate dalla Apple stessa. Ciò viene effettuato in modo da riscontrare eventuali anomalie, evitando quindi l'installazione di programmi che potenzialmente potranno portare una cattiva interazione del dispositivo con l'utente. Questa esperienza viene comunemente definita bad experience. Android per sua natura lascia questo aspetto alla coscienza degli sviluppatori, i quali devono essere consapevoli che una cattiva applicazione potrà portare una pessima valutazione e reputazione.
A questo punto una domanda sorge spontanea: come può Android eseguire bytecode Java, senza l'utilizzo di una JVM? Semplice, Android non esegue bytecode Java, per cui non ha bisogno di una JVM. Per ottimizzare al massimo l'utilizzo delle risorse dei dispositivi, Google ha adottato una propria VM, che prende il nome di Dalvik Virtual Machine (nome di una località in Islanda) sviluppata inizialmente da Dan Bornstein. La DVM è una VM ottimizzata per l'esecuzione di applicazioni in dispositivi a risorse limitate. Esegue solo codice contenuto all'interno di file di estenzione .dex che sono ottenuti al loro volta in fase di building, dai file .class di bytecode Java.
Android permette l'utilizzo di tutte le librerie Java, ad eccezione delle Abstract Window Toolkit (AWT) e le Swing. La definizione dell'interfaccia grafica è infatti un aspetto fondamentale dell'architettura Android, in cui viene utilizzato un approccio dichiarativo, come ormai avviene nella maggior parte delle attuali piattaforme di sviluppo.

1.2 La Dalvik Virtual Machine

L'esigenza di creare delle applicazioni che siano in grado di rispondere in modo immediato all'utente è fondamentale. Con un hardware a 'risorse limitate' non si può far altro che adottare tutti i possibili accorgimenti, sia a livello di architettura, sia a livello di software, in modo da poter sfruttare al massimo le risorse disponibili.
Per avere dei dati significativi, prendiamo il dispositivo utilizzato per sviluppare l'applicativo di questa tesi, un HTC Legend, rilasciato pochi mesi fa. Le sue caratteristiche sono:

- memoria ram di 384 MB;

- memoria rom di 512 MB;
- processore Qualcomm MSM7227 a 600MHz

Se andiamo in un negozio o un sito per l'acquisto di un notebook, possiamo notare come un PC con caratteristiche medio-basse abbia comunque una RAM di almeno 2 GB e un processore Dual Core intorno ai 2 GHz.
Viste queste caratteristiche, possiamo capire il motivo per il quale è stata adottata una nuova VM diversa da quella di Sun. La DVM è ottimizzata per l'esecuzione di applicazioni in ambienti ridotti, ed è in grado di sfruttare al massimo le caratteristiche del sistema operativo ospitante. Non viene quindi sfruttata la VM di Java, in quanto si ha esigenza di risparmiare quanto più spazio possibile per la memorizzazione ed esecuzione delle applicazioni. Per esempio, se un'applicazione Java è descritta da codice contenuto all'interno di un archivio .jar di 100 KB (non compresso), la stessa potrà essere contenuta all'interno di un file di dimensione di circa 50 KB se trasformato in .dex. Questa diminuzione di quasi il 50% avviene grazie alla fase di trasformazione di bytecode java al bytecode per la DVM. Questo perché i diversi file .dex sono in grado di condividere informazioni che altrimenti nel bytecode verrebbero ripetute più volte.
La DVM non elimina il garbage collector (GC)[7] in quanto una gestione della memoria a carico del programmatore avrebbe complicato quello che è lo sviluppo delle applicazioni oltre ad aumentare la probabilità di bug e memory leak.
La DVM usa un meccanismo di generazione del codice che viene detto register based (orientato all'utilizzo di registri) a differenza di quella della JVM detto invece stack based (orientato all'utilizzo di stack). Questo meccanismo permette di ridurre di circa il 30% il numero di operazioni da eseguire. Per capire meglio come ciò possa avvenire, supponiamo di voler valutare la seguente semplice espressione: c = a + b .
Con L indichiamo l'operazione di caricamento del dato (load) e con S indichiamo l'operazione di scrittura dello stesso (store). La precedente istruzione si può tradurre nelle seguenti operazioni:

[7]per garbage collection (letteralmente raccolta dei rifiuti, a volte abbreviato con GC) si intende una modalità automatica di gestione della memoria, mediante la quale un sistema operativo, o un compilatore e un modulo di run-time, liberano le porzioni di memoria che non dovranno più essere successivamente utilizzate dalle applicazioni.

- push b; // LS
- push a; // LS
- add; // LLS
- store c; // LS

Si tratta quindi del caricamento di due operandi a e b nello stack, del calcolo della loro somma e della memorizzazione del risultato in cima allo stack stesso. Se volessimo ora eseguire la stessa operazione con un meccanismo register based, otterremo:

- add a,b,c; // LLS

Otteniamo il caricamento degli operandi a e b in zone diverse di un registro e della memorizzazione del risultato nel registro stesso. I vantaggi di tale operazione sono minor tempo di esecuzione delle istruzioni al prezzo di una maggiore elaborazione in fase di compilazione o trasformazione. Quindi le operazioni di preparazione per l'esecuzione dell'operazione in un unico passaggio nel registro, possono essere eseguite in fase di building. Facendo ciò si riduce così lo sforzo a runtime ed è sicuramente un aspetto positivo nei dispositivi mobili.
Ultima importantissima caratteristica della DVM è quella di permettere un'efficace esecuzione di più processi contemporaneamente. Questo permette a ciascuna applicazione di essere in esecuzione all'interno del proprio processo Linux. C'è da ammettere però che, a fronte di molti vantaggi dal punto di vista delle prestazioni, ciò comporta alcune implicazioni dal punto di vista della sicurezza.

1.3 L'architettura di Android

Android ha un'architettura che comprende tutto lo stack degli strumenti per la creazione di applicazioni mobili di ultima generazione. Tra questi strumenti troviamo un sistema operativo, un insieme di librerie native per le funzionalità core della piattaforma, un'implementezione della VM e un insieme di librerie Java.
Gli sviluppatori hanno a disposizione un'architettura a layer, dove i livelli inferiori offrono

servizi ai livelli superiori, offrendo un più alto grado di astrazione. Esamineremo tutti i componenti di questa architettura, facendo riferimento all'immagine che rappresenta l'architettura di Android(Figura 1.3).

1.3.1 Il kernel di Linux

Il layer di più basso livello è rappresentato dal kernel Linux nella versione 2.6. La necessità iniziale era quella di disporre di un vero e proprio sistema operativo che fornisse gli strumenti di basso livello per la virtualizzazione dell'hardware sottostante, attraverso la definizione di diversi driver. Possiamo quindi notare la presenza di driver per le gestione delle periferiche multimediali, del display, della connessione Wi-Fi, dell'alimentazione e della flash memory.

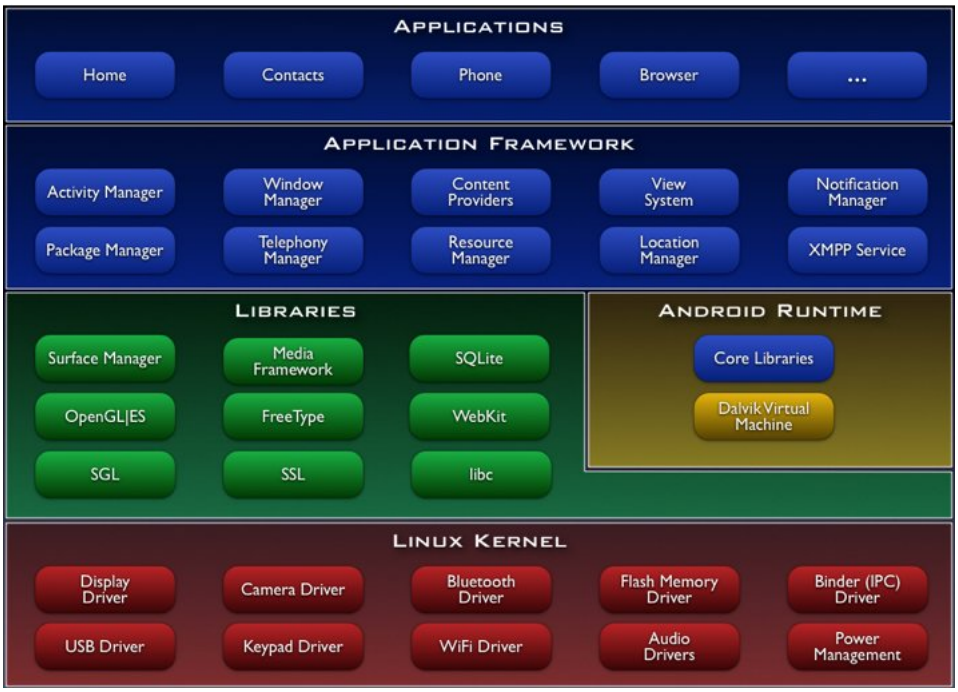


Figura 1.3: Architettura di Android[10].

È importante sottolineare anche la presenza di un driver dedicato alla gestione della comunicazione tra processi diversi (IPC). Questo è fondamentale per far comunicare componenti diversi in un ambiente in cui ciascuna applicazione viene eseguita all'interno

di un proprio processo. La scelta verso l'utilizzo di un kernel Linux è conseguenza della necessità di avere un SO che fornisse tutte le feature di sicurezza, di gestione dei processi e della memoria e di power management, e che al tempo stesso fosse affidabile e testato.

1.3.2 Librerie native

Sopra il layer costituito dal kernel di Linux 2.6, abbiamo un livello che contiene un insieme di librerie native realizzate in C e C++. Queste rappresentano il core vero e proprio di Android e fanno riferimento a un insieme di progetti Open Source, di seguito descritti.

Surface Manager

Il Surface Manager (SM) è un componente fondamentale in quanto ha la responsabilità di gestire le view, ovvero i componenti dell'interfaccia grafica. Esso deve coordinare le diverse finestre che le applicazioni vogliono visualizzare sullo schermo. È da precisare infatti, che ciascuna applicazione è in esecuzione in un processo diverso, e disegna quindi la propria interfaccia in tempi diversi. Quindi il compito del SM è prendere le diverse finestre e disegnarle sul buffer da visualizzare poi attraverso la tecnica del double buffering. In questo modo non si avranno finestre che si accavallano sul display. Il SM ha inoltre accesso alle funzionalità del display e permette la visualizzazione contemporanea di grafica 2D e 3D dalle diverse applicazioni, è quindi una componente di importanza fondamentale.

OpenGL ES

La libreria utilizzata per la grafica 3D è quella che va sotto il nome di OpenGL ES[11], una versione ridotta di OpenGL per i dispositivi mobili che permette di accedere alle funzionalità di un eventuale acceleratore grafico hardware.
Si tratta di un insieme di API multipiattaforma che forniscono l'accesso a funzionalità 2D e 3D in dispositivi embedded. L'OpenGL ES ha delle specifiche che permettono ai diversi produttori di implementarle adattandole alle proprie macchine o sistemi operativi senza pagare alcuna royalty.

SGL

La Scalable Graphics Library (SGL) è una libreria in C++ che insieme alle OpenGL costituisce il motore grafico di Android. Mentre per la grafica 3D vengono utilizzate le OpenGL ES, per la grafica 2D viene utilizzato un motore ottimizzato chiamato appunto SGL. È una libreria utilizzata principalmente dal Window Manager e dal Surface Manager all'interno del processo di renderizzazione grafica.

Media Framework

Le applicazioni Android faranno largo utilizzo di conenuti multimediali. Per fare ciò è necessario gestire i diversi CODEC per i vari formati di acquisizione e riproduzione di audio e video. Questo compononente si chiama appunto Media Framework che si basa sulla libreria open source OpenCore di PacketVideo (uno dei membri fondatori dell'OHA).
Da segnalare che OpenCore contiene come supporto agli encoder OpenMax: si tratta di un insieme di API liberamente utilizzabili, che permettorno un'astrazione delle operazioni che un dispositivo mobile è in grado di eseguire su un determinato stream di dati.
I codec gestiti dal Media Framework permettono di gestire formati importanti, tra cui MPEG4, H.264, MP3, AAC e AMR, oltre a gestire immagini come JPG e PNG.

FreeType

La gestione dei font è un altro aspetto molto importante nella definizione di un'interfaccia. Per Android è stato scelto di utilizzare il motore di rendering dei font FreeType[12]. I motivi di questa scelta, sono dati dal fatto che questo motore è di piccole dimensioni, molto efficiente, customizzabile, e soprattutto portabile. Grazie a FreeType, le applicazioni di Android saranno in grado di visualizzare immagini di alta qualità. Il suo punto di forza è quello di fornire un insieme di API semplici per ciascun tipo di font, indipendentemente dal formato del corrispondente file.

SQLite

Con Android si è deciso di utilizzare qualcosa di efficiente e piccolo, ma che avesse le caratteristiche di un DBMS relazionale. Ecco quindi i motivi della scelta di SQLite[13]. Si tratta di una libreria in-process che implementa un DBMS relazionale caratterizzato dal fatto di essere molto compatto e diretto, di non necessitare alcuna configurazione, e soprattutto di essere transazionale.
SQLite è compatto, in quanto realizzato interamente in C in modo da utilizzare solo poche delle funzioni ANSI per la gestione della memoria. Inoltre è diretto in quanto non utilizza alcun processo separato per operare ma 'vive' nello stesso processo dell'applicazione che ne fa utilizzo, da cui il termine in-process. Non necessita di alcuna procedura di installazione, quindi è adatto per sistemi embedded. SQLite non è un prodotto Android, in quanto esiste autonomamente con diversi tool di supporto.

WebKit

Ovviamente non poteva mancare un browser integrato nella piattaforma Android. Per questo proposito, si è scelto il framework WebKit[14] utilizzato anche dai browser Safari e Chrome. Si tratta di un browser engine (che dovrà essere integrato in diversi tipi di applicazioni) open source basato sulle teconologie HTML, CSS, JavaScript e DOM.

SSL

Si tratta dell'ormai famosa libreria per la gestione dei Secure Socket Layer. In tema di sicurezza, questa libreria non poteva di certo mancare.

Libc

Si tratta di un'implementazione della libreria standard C libc ottimizzata per i dispositivi basati su Linux embedded, proprio come Android.

1.3.3 Android Runtime

Come in Java, per le applicazioni Android servono tutte le classi relative all'ambiente in cui esse vengono eseguite. La differenza sta che in fase di compilazione avremo bisogno

del jar (di nome android.jar) per la creazione di bytecode Java, mentre in esecuzione il dispositivo metterà a disposizione la versione .dex del runtime che costituisce appunto la core library. Come già citato, nel dispositivo non c'è codice Java, ma solamente codice .dex, eseguito dalla DVM.

1.3.4 Application Framework

Tutte le librerie fin qui descritte, vengono poi utilizzate da un insieme di componenti di più alto livello che costituiscono l'Application Framework (AF). Si tratta di un insieme di API e componenti per l'esecuzione di funzionalità ben precise e di fondamentale importanza in ciascuna applicazione Android. È importante sapere che tutte le applicazioni per Android utilizzano lo stesso AF e come tali possono essere estese, modificate o sostituite. Da qui il motto che è possibile trovare sul sito di Android: 'All applications are equals'.

Activity Manager

In Android, il concetto di attività lo possiamo associare ad un schermata della nostra applicazione, che ne permette la visualizzazione o la raccolta di informazioni. È quindi lo strumento fondamentale attraverso il quale l'utente interagisce con l'applicazione. L'Activity Manager gestisce il ciclo di vita di ogni attività. Dovrà quindi organizzare le varie schermate di un'applicazione in uno stack a seconda dell'ordine di visualizzazione delle stesse sullo schermo.

Package Manager

Un aspetto molto importante di un sistema come Android è la gestione del processo di installazione delle applicazioni nei dispositivi. Ogni applicazione deve fornire al dispositivo che la andrà ad eseguire delle determinate informazioni. È compito del programmatore descrivere queste informazioni, attraverso un opportuno file XML di configurazione di nome AndroidManifest. In questo file si trovano informazioni di vario genere relative ad aspetti grafici (layout) dell'applicazione, a diverse activity e ad aspetti di sicurezza.

Gestire il ciclo di vita delle applicazioni nei dispositivi è il compito affidato al Package Manager.

Window Manager

Il Window Manager permette di gestire le finestre delle diverse applicazioni, gestite da processi diversi, sullo schermo del dispositivo. Esso può essere considerato come un'astrazione, con API Java, dei servizi nativi del Surface Manager descritti in precedenza.

Telephony Manager

Il TM permette una maggiore interazione con le funzionalità caratteristiche di un telefono come la semplice possibilità di iniziare una chiamata o di verificare lo stato della chiamata stessa.

Content Provider

Il CP è un componente fondamentale nella realizzazione delle applicazioni Android, poiché ha la responsabilità di gestire la condivisione di informazioni tra i vari processi. Funziona in modo simile a quello di un repository condiviso con cui le diverse applicazioni possono interagire inserendo o leggendo informazioni.

Resource Manager

Un'applicazione è composta, oltra che dal codice, anche da un insieme di file di tipo diverso, per esempio: immagini, file di configurazione o di properties per la internazionalizzazione (I18N), file di definizione del layout e molti altre. Il RM ha la responsabilità di gestire questo tipo di informazioni, mettendo a disposizione una serie di API di semplice utilizzo. Come per il codice, anche per le risorse esiste un processo di trasformazione. Le stesse si troveranno in contenuti binari ottimizzati per il loro utilizzo all'interno di un dispositivo. A queste risorse, si può accedere attraverso costanti generate in modo automatico in fase di building.

View System

L'interfaccia grafica di un'applicazione Android è composta da quelle che saranno specializzazioni della classe View. Ogni classe è caratterizzata da una particolare forma e da un diverso modo di interagire con essa, attraverso un'accurata gestione degli eventi associati. La gestione della renderizzazione dei componenti, nonché della gestione degli eventi associati è di responsabilità di un componente che si chiama View System (VS).

Location Manager

Ci sono diverse demo delle applicazioni disponibili per Android, quelle più interessanti sono appunto relative alla gestione delle mappe. Le applicazioni che gestiscono le informazioni relative alla localizzazione si chiamano Location Based Application (LBA). Queste informazioni possono essere realizzate utilizzando API messe a disposizione dal Location Manager. Quindi nei dispositivi Android è possibile accedere a funzioni legate alla location, tra cui le operazioni di georeferenziazione.

Notification Manager

Altro importante servzio è il Notification Manager. Quest'ultimo mette a disposizione un insieme di strumenti che l'applicazione può utilizzare per inviare una particolare notifica al dispositivo, il quale la presenterà all'utente. L'applicazione può quindi notificare un particolare evento al dispositivo che potrebbe, ad esempio, emettere una vibrazione, far lampeggiare i LED, visualizzare un'icona e altro ancora.

Capitolo 2

La collaborazione in rete

Sin dalle sue origini Internet è stato uno strumento di comunicazione tra persone oltre che una rete telematica per trasmettere dati. Oggi l'uso di applicazioni web e programmi che agiscono via internet per il lavoro a distanza sta diventando un approccio molto produttivo ed economico per le aziende.
In questo capitolo si descrive la collaborazione in rete e i groupware, software con il quale un gruppo di persone on-line può scambiare e creare informazioni.

2.1 I CSCW e i groupware

Il CSCW (Computer Supported Cooperative Work, lavoro cooperativo assistito dal computer) è una disciplina scientifica che studia il lavoro di gruppo e cerca di scoprire come la tecnologia possa renderlo più efficace. Questo termine, prevalentemente accademico, è stato usato per la prima volta nel 1984 per un workshop organizzato da Irene Greif (MIT) e David Cashman (DEC)[17].
Il CSCW è interdisciplinare, in quanto non si occupa solo di informatica e telecomunicazioni, ma anche di organizzazione aziendale, sociologia e psicologia. Infatti viene impiegato in molti campi, tra i quali la medicina, lo sviluppo dei software, la ricerca scientifica e molti altri.
I prodotti software che più direttamente si ispirano ai risultati e alle proposte di quest'area di ricerca sono detti groupware. Il termine groupware (dall'inglese group - gruppo, ware

- suffisso di software e hardware) si riferisce alle tecnologie (in genere basate su computer) pensate per facilitare e rendere più efficace il lavoro cooperativo da parte di gruppi di persone. Software di questo tipo permettono di condividere il lavoro e comunicare informazioni per gli scopi più diversi. Alcuni software collaborativi sono:

- i sistemi collaborativi di gestione di posta elettronica;
- i TikiWiki che consentono l'utilizzo di molteplici strumenti quali Wiki, Blog;
- le classiche applicazioni Internet di comunicazione, come le chat o i forum.

Un groupware deve necessariamente fornire ai componenti di un team di lavoro, queste funzionalità:

- comunicare, ovvero scambiarsi informazioni tra di loro;
- condividere le informazioni da loro generate e utilizzarle, lavorando contemporaneamente a parti differenti di uno stesso oggetto, tenendo traccia dei cambiamenti apportati;
- coordinarsi, in quanto ogni elemento ha dei compiti e delle attività nel gruppo, evitando conflitti e ridondanze al suo interno.

Questi programmi collaborativi possono essere classificati in base a quando i partecipanti lavorano e al luogo in cui lavorano. Quindi il lavoro può essere svolto in modo sincrono o asincrono, in uno spazio co-situato o remoto, distinguendo diverse modalità di interazione (Figura 2.1).

Figura 2.1: La matrice spazio tempo.

L'applicazione che si vuole creare è classificabile come 'Shared information spaces'. La caratteristica di questo tipo di groupware è la comunicazione implicita e asincrona tramite la condivisione di documenti, basata su un database e sul web.

2.2 Esempi di groupware

Pioniera nello sviluppo di software collaborativo è stata la Lotus Software[18], con l'insieme di applicazioni Lotus Notes: un software collaborativo con architettura client-server e con gestione integrata dell'email.
Il programma oltre ad essere un sistema di groupware (e-mail, calendario, documenti condivisi e discussioni), fornisce anche la possibilità di sviluppare applicazioni personalizzate sia client-server sia web (Figura 2.2).

Restando però in tema open source, un software collaborativo di prossimo sviluppo è Google Wave.
Google Wave è definito come 'uno strumento personale di comunicazione e collaborazione'. Si tratta di un misto tra un'applicazione web, una piattaforma e un protocollo di comunicazione pensato per riunire e-mail, messaggistica istantanea, wiki e social

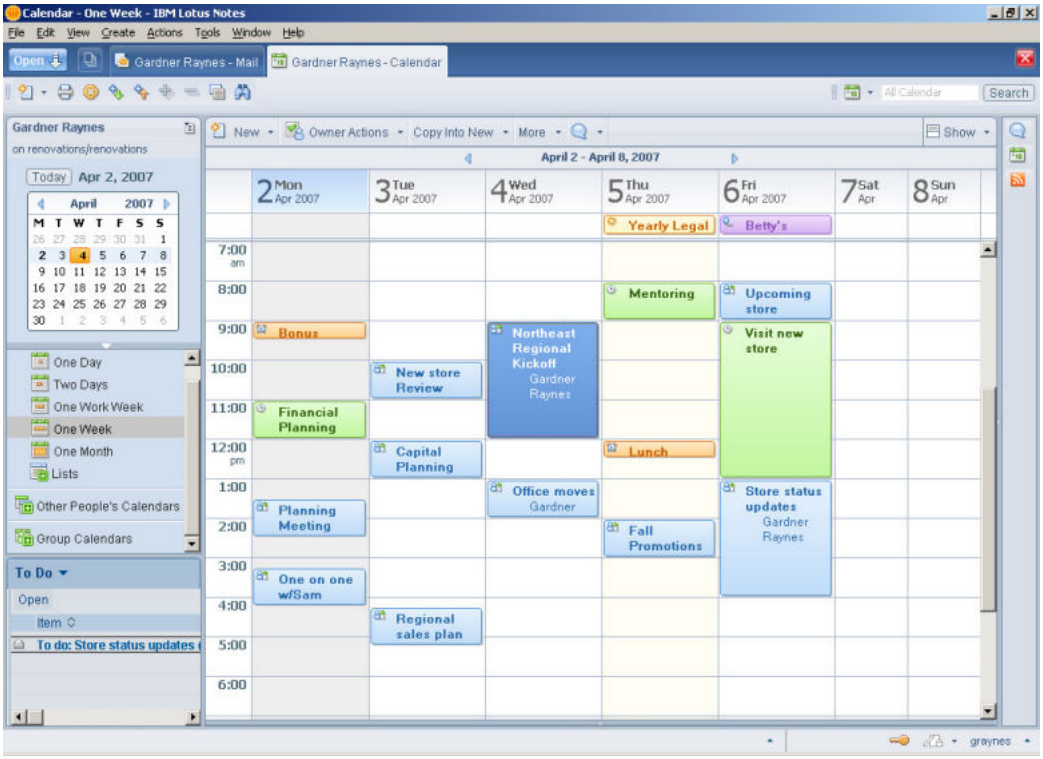


Figura 2.2: Calendario del programma Lotus Notes[18].

network. Questo software è orientato alla collaborazione e al real-time, supportato da estensioni che possono fornire ad esempio un solido controllo ortografico e grammaticale, la traduzione automatica tra 40 diverse lingue e diverse altre estensioni (Figura 2.3).

Google Wave è una piattaforma open source, che consente di creare una rete di interscambio dati battezzata 'WaveNetwork'. Inoltre permette l'integrazione con altre applicazioni Google, come Google Maps, Gmail, Google docs, Google Calendar. Wave infatti integra in un'unica schermata on-line servizi di social network, messaggistica istantanea, documenti, foto, video, audio sharing, feed Rss, mappe e altro.
Gli internauti potranno quindi scambiarsi e condividere in modalità 'one to many' (uno a molti) contenuti in formato digitale (mappe, testi, immagini, filmati) e conservare questi dati in tempo reale, realizzando un gruppo di lavoro digitale. Quindi un utente avrà la possibilità:

- di elaborare documenti condivisi;
- di gestire un''onda' comunicativa UGC[1] in tempo reale;
- di gestire interazioni e comunicazioni con gruppi ristretti.

[1]User Generated Content, contenuto generato dagli utenti.

Figura 2.3: Schermata principale di Google Wave[19].

Capitolo 3

La fase di analisi

In questo capitolo verrà presentata la modalità di sviluppo dell'applicazione MyNotes. Saranno definite i requisiti del progetto e la sua possibile collocazione nel Market di Android. Verranno analizzati i rischi e i costi di sviluppo e le metriche per stabilire la qualità del software, che dovranno essere rispettate.
Viene inoltre presentato il modello di processo attuato per lo sviluppo del progetto.

3.1 Situazione attuale

I servizi che oggi sono disponibili sulle piattaforme dei dispositivi mobili, sono in continuo sviluppo.
È possibile usare il proprio dispositivo come agenda personale, dove tener sotto controllo gli impegni quotidiani. È possibile conoscere le condizioni climatiche del proprio luogo di residenza. È possibile puntare il proprio dispositivo verso un monumento per collegarsi a Wikipedia e conoscerne la storia.
Queste sono alcune delle tante funzioni a cui possiamo fare affidamento per i nostri bisogni, ed essendo questo mercato appena nato, le migliori applicazioni devono ancora essere create. Ciò che oggi è disponibile per i PC, potrà essere sviluppato domani per uno smartphone.
Nel campo della collaborazione in rete, esistono già dei validi programmi usati per i lavori di gruppo nelle aziende. A questo proposito si vuole realizzare un prototipo di un

editor collaborativo di testo in rete, il cui nome è MyNotes, per poter valutare la sua realizzazione su dispositivi mobili. Il prototipo sarà completamente open source e verrà implementato per piattaforma Android.
Si ha l'opportunità di posizionare il prodotto in un campo non ancora esplorato. Fornire quindi un servizio analogo a Google Docs[1] che sia in grado di soddisfare il bisogno di gestire i propri file anche su dispositivi mobili.
Il prodotto nella sua versione definitiva potrebbe essere posizionato nell'Android Market, nella sezione delle applicazioni di Produttività. Tuttavia non è stato fatto uno studio approfondito di tale sezione, ma è stata individuata e analizzata solo un'alternativa: Gdocs (Figura 3.1). Questa applicazione gratuita permette solo la gestione dei documenti presenti sul proprio account Google. Non offre quindi la possibilità a più utenti di collaborare per eventuali modifiche. Inoltre l'apertura e la gestione dei file sono limitate alle estensioni .doc e .pdf.

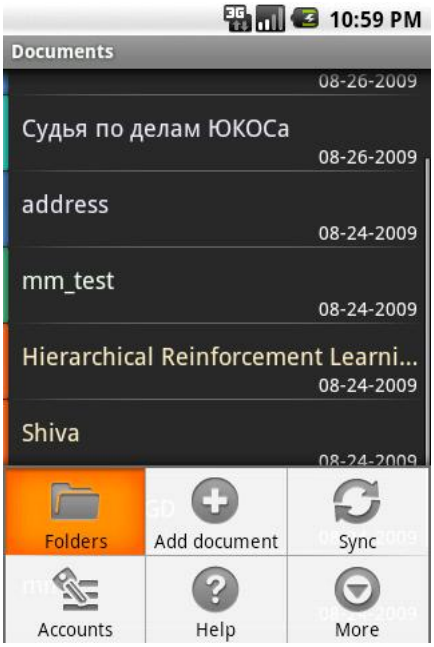


Figura 3.1: Schermata principale di Gdocs[20].

[1]Google docs è un servizio in rete di Google, dove un utente iscritto ha circa 1Gb di spazio dove creare e salvare i suoi documenti, ad esempio: testi, presentazioni e fogli di calcolo.

La concorrenza quindi non è presente in questa sezione.

3.2 Il progetto MyNotes

Si è deciso di sviluppare un'applicazione di base che gestisce una connessione tra i dispositivi tramite tecnologia Bluetooth. Gli utenti saranno guidati verso la creazione di una Personal Area Network, dove potranno interagire nello scambio dei dati. Una volta avviata la connessione, il programma gestirà le connessioni e avvierà la condivisione di un testo.
L'applicazione MyNotes sarà anche un editor di testo, che offrirà modalità di creazione, salvataggio e esportazione delle note nella memoria del dispositivo. La piattaforma Android richiesta è la versione 2.1 detta anche Eclair, o superiore, per eseguire l'applicazione correttamente.
Il sistema dovrà fornire le seguenti caratteristiche:

- fornire la possibilità all'utente di condividere i testi;
- gestione dei file creati;
- gestione delle connessioni di rete degli utenti collegati;
- invio delle modifiche da parte degli utenti sul testo condiviso.

3.2.1 Specifiche dei requisiti

In questa sezione si elencheranno i requisiti dividendoli in requisiti funzionali e requisiti non funzionali. I requisiti funzionali descrivono tutte le funzionalità ed i servizi forniti dall'applicazione, mentre i requisiti non funzionali ne descrivono proprietà e vincoli. Obiettivo dell'analisi di questi requisiti sono:

- misurare le funzionalità che l'utente riceve e richiede;
- misurare i risultati dello sviluppo e/o la manutenzione del software indipendentemente dalla tecnologia utilizzata;
- fornire una misura che sia coerente tra progetti e produttori differenti.

Requisiti funzionali

1. Visualizzazione dell'attività principale. L'applicazione guiderà l'utente a creare nuove note, oppure visualizzerà le note create in precedenza.
2. Quando l'utente sceglie di creare una nuova nota, verrà avviata l'attività di creazione/modifica di una nota.
3. È possibile effettuare modifiche sulla nota creata ad esempio annullare, copiare, incollare, selezionare.
4. La nota può essere salvata in qualsiasi momento nel database dell'applicazione.
5. La nota può essere esportata in qualsiasi momento nella memoria SD del dispositivo in formato .txt, nell'apposita cartella dell'applicazione.
6. La nota può essere cancellata in qualsiasi momento dal database dell'applicazione.
7. Dall'attività principale è possibile rendere il dispositivo disponibile ad una condivisione di una nota, inviata da un altro utente.

Requisiti non funzionali

1. Tolleranza in caso di disconnessione da parte di uno dei due utenti che stanno modificando la nota. Entrambi gli utenti potranno salvare la nota condivisa, evitando quindi la perdita del lavoro svolto.
2. In caso di improvvisa interruzione dell'applicazione, i dati salvati devono essere recuperabili dal database dell'applicazione.

3.2.2 Il modello di processo

Per realizzare l'applicazione MyNotes, si è scelto di seguire un modello di processo software iterativo ed incrementale, il Rational Unified Process[2]. Grazie a questa scelta, si è potuto organizzare il processo di produzione in quattro fasi:

[2]Estensione dello Unified Process, è un modello di processo software iterativo sviluppato da Rational Software (oggi parte di IBM).

- Inception (Concezione): consiste nel concepimento dell'editor di testo, in particolare l'analisi della fattibilità, delle risorse e delle problematiche relative allo sviluppo di una rete tra due dispositivi;

- Elaboration (Elaborazione): consiste nello sviluppare un piano di realizzazione del software, definendo la sua architettura e individuando gli elementi che potrebbero causare rischi elevati, in modo da poterli gestire in anticipo;

- Construction (Costruzione): consiste nella realizzazione in maniera incrementale di varie versioni dell'applicazione, introducendo in ogni versione nuove funzionalità;

- Transition (Transizione): fase finale del lavoro, nel quale viene effettuato il debuggin del sistema con la correzione degli errori stabilendo infine, se il prodotto è conforme a ciò che è stato stabilito nella fase di concezione.

L'utilità del processo RUP è data dalla sua evoluzione ad ogni passo che viene effettuato. Il ciclo può ripetersi molte volte, con un diverso tono di attenzione, qualora fosse necessario.

Figura 3.2: Le quattro fasi dell'Unified Process

3.2.3 Principali obiettivi

Quando si realizza un editor di testo collaborativo, è necessario tenere in considerazione i seguenti obiettivi:

- Modifica flessibile: gli utenti devono avere una completa flessibilità nella modifica del documento, devono poter essere in grado di utilizzare tutte le funzionalità messe a disposizione senza che l'uso di esse comprometta il lavoro svolto;

- Collaborazione: questa caratteristica indica che ogni utente deve conoscere le azioni degli altri partecipanti;

- Controllo della sessione: un editor collaborativo dovrebbe consentire agli utenti di creare o lasciare sessioni di editazioni per un tempo arbitrario;
- Comunicazione: il sistema deve fornire lo stesso meccanismo di comunicazione per gli utenti che stanno collaborando su un documento;
- Facilità di cancellazione: il sistema di editazione deve consentire un facile supporto per la cancellazione degli aggiornamenti.

3.3 Analisi del rischio

Come spesso avviene in tutti i progetti software, ci si può ritrovare a dover affrontare dei ritardi o al fallimento del progetto stesso. Analizzare i rischi vuol dire controllare che il prodotto sia di buona qualità, che sia realizzato nei tempi prestabiliti, che non usi risorse eccessive e che rispetti i requisiti forniti.
I maggiori rischi valutati e che si potranno riscontrare sono qui di seguito elencati.

- Pianificazione inefficiente: riguarda soprattutto la pianificazione dei tempi di lavoro, dovuto ad un'analisi scorretta dei tempi necessari allo sviluppo dell'applicazione;
- Dispositivi di sviluppo: riguarda in particolare alla obsolescenza dei sistemi di sviluppo; per sviluppare alcune applicazioni Android sono necessari dei dispositivi e relative piattaforme aggiornate;
- Perdita di dati: il rischio derivante alla perdita dei dati è stato arginato dal salvataggio giornaliero di essi, sfruttando hard disk remoti e servizi on-line.

A questi rischi si aggiungono poi quelli riguardanti la creazione dei sistemi collaborativi.

3.3.1 Difficoltà tecniche su dispositivi mobili

Ci sono due tipi di editor di testi collaborativi: quelli in real-time (RTCE[3]) e quelli non real-time. I real-time collaborative editing è una modalità di editing sincronizzata

[3]http://en.wikipedia.org/wiki/Collaborative_real-time_editor

(o simultanea), il che significa che gli utenti possono modificare lo stesso file contemporaneamente. Mentre nei sistemi collaborativi non in tempo reale la collaborazione è asincrona quindi gli utenti non potranno modificare lo stesso file contemporaneamente.
Visti alcuni limiti di risorse che si hanno a disposizione in una piattaforma mobile e dalla complessità nel gestire un editor in tempo reale a causa del networking lag, si è deciso di sviluppare l'applicativo fornendo una collaborazione asincrona. Le complessità di sviluppo degli editor in tempo reale è data dal tempo che passa tra la spedizione di un pacchetto di dati e la sua ricezione. Questo problema dipende dalla velocità della comunicazione che influisce negativamente, creando un problema fondamentale che è quello di dover ricostruire le modifiche applicate al testo. Gli utenti non possono ottenere le modifiche del documento in modo istantaneo, quindi il server dovrà ricostruire la sequenza di azioni effettivamente svolte dai vari utenti.
Questo editor collaborativo di base è stato creato come modello client-server. Chi lancia la condivisione della nota avrà il ruolo di server, il quale dopo aver stabilito una connessione invierà i dati all'utente connesso. Le modifiche del testo a livello locale non potranno essere effettuate fino a quando non viene restituito al server una risposta.
Ovviamente questo è un approccio molto meno potente, ma garantisce una collaborazione di base ad un costo computazionale relativamente basso. È utile applicarlo su piattaforme mobili in cui ci sono limitate risorse di elaborazione. Un'applicazione che usa questo modello è NetSketch[4].

3.3.2 Risorse

Le risorse necessarie richieste dai sistemi collaborativi su piattaforma mobile si possono riassumere in:

- Comunicazione: la comunicazione è necessaria per l'interazione fra gli utenti e come gia detto è asincrona;
- Eventi: gli utenti possono ricevere notifiche di eventi;
- Nickname dispositivo: serve ad indenticare l'identità di chi interagisce con oggetti condivisi;

[4]Programma di disegno collaborativo disponibile su iPhone e iTouch.

- Operazioni di modifica: possono essere locali e remote. Un'operazione è locale se è generata localmente, un'operazione è remota se è generata da un utente remoto e notificata ad un client attraverso l'uso di un meccanismo di conferma.

3.4 Analisi dei costi

3.4.1 Stime di Costo Software

Per quanto riguarda la valutazione del costo delle risorse umane, si è scelto d'applicare il noto modello di stima dei costi: l'algoritmo di COCOMO II (COnstructive COst MOdel)[5]. Per questo studio si è utilizzata l'ultima versione del software per COCOMO II, fornito dall'USC-CSSE[6].
Di seguito si farà riferimento a voci visualizzate nell'interfaccia grafica dell'applicativo disponibile alla pagina del sito consultato[7].
Si è ipotizzato uno sviluppo di 3000 linee di codice, non riutilizzato o modificato da progetti precedenti, lasciando invariati gli altri campi. Per quanto riguarda i costi del prodotto, si richiede che la sua affidabilità, la grandezza del database e la sua complessità, siano molto elevate. Nei costi del personale si ritiene opportuno avere dei programmatori con capacità, continuità e esperienze di programmazione molto alte. Infine per quanto riguarda i costi del progetto si prevede l'uso di tools esterni e di un piano di sviluppo.

Il risultato ottenuto è stato uno sforzo complessivo di 4 mesi/persona per un totale di 2000 SLOC[8]. Risultato in linea con i tempi che sono stati necessari per lo sviluppo dell'applicazione.

[5]Pubblicato nel 2001 nel libro Software Cost Estimation with COCOMO II[15].

[6]Center for Systems and Software Engineering della University of Southern California e fondato nel 1993 dallo stesso Boehm[16].

[7]http://csse.usc.edu/tools/COCOMOII.php

[8]Source lines of code (dall'inglese Linee di codice sorgente) è una metrica software che misura le dimensioni di un software basandosi sul numero di linee di codice sorgente.

Software Size Sizing Method: Source Lines of Code

	SLOC	% Design Modified	% Code Modified	% Integration Required	Assessment and Assimilation (0% - 8%)	Software Understanding (0% - 50%)	Unfamiliarity (0-1)
New	2000						
Reused		0	0				
Modified							

Software Scale Drivers

Precedentedness	Nominal	Architecture / Risk Resolution	Nominal	Process Maturity	Nominal
Development Flexibility	High	Team Cohesion	High		

Software Cost Drivers

Product		Personnel		Platform	
Required Software Reliability	Very High	Analyst Capability	Nominal	Time Constraint	Nominal
Data Base Size	Very High	Programmer Capability	Very High	Storage Constraint	Nominal
Product Complexity	Very High	Personnel Continuity	Very High	Platform Volatility	Nominal
Developed for Reusability	Nominal	Application Experience	High	**Project**	
Documentation Match to Lifecycle Needs	Nominal	Platform Experience	High	Use of Software Tools	Very High
		Language and Toolset Experience	High	Multisite Development	Nominal
				Required Development Schedule	High

Figura 3.3: Calibrazione dei parametri su COCOMO II.

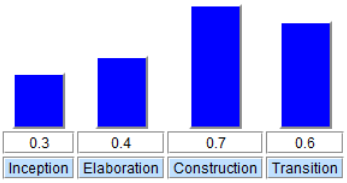
Results

Software Engineering

Effort = 4 Person-months
Schedule = 7 Months
Cost = $0

Total Equivalent Size = 2000 SLOC

Staffing by Phase



Phase Distribution

Phase	Effort (Person-months)	Schedule (Months)	Average Staff	Cost (Dollars)
Inception	0.3	1.0	0.3	$0
Elaboration	1.1	3.0	0.4	$0
Construction	3.6	5.0	0.7	$0
Transition	0.6	1.0	0.6	$0

Software Effort Distribution for RUP/MBASE (Person-Months)

Phase/Activity	Inception	Elaboration	Construction	Transition
Management	0.0	0.1	0.4	0.1
Environment/CM	0.0	0.1	0.2	0.0
Requirements	0.1	0.2	0.3	0.0
Design	0.1	0.4	0.6	0.0
Implementation	0.0	0.1	1.2	0.1
Assessment	0.0	0.1	0.9	0.1
Deployment	0.0	0.0	0.1	0.2

Your output file is http://csse.usc.edu/tools/data/COCOMO_June_20_2010_06_04_33_158438.txt

Created by Ray Madachy at the Naval Postgraduate School. For more information contact him at rjmadach@nps.edu

Figura 3.4: Risultati dell'analisi su COCOMO II.

3.4.2 Il grafico di Gantt

Come strumento di supporto alla gestione del progetto, si è deciso di adottare il diagramma di Gantt[9].
Un diagramma di Gantt permette di rappresentare graficamente un calendario delle attività. Tale operazione è utile al fine di pianificare, coordinare e tracciare specifiche attività in un progetto dando una chiara illustrazione dello stato d'avanzamento del progetto rappresentato.
Qui di seguito viene illustrato il grafico del progetto MyNotes, dove possiamo notare la suddivisione in sotto-fasi della fase di Concezione e di Elaborazione.

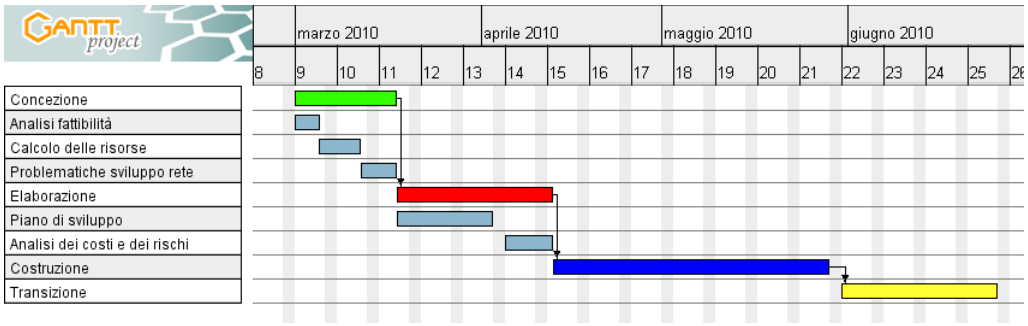


Figura 3.5: Grafico di Gantt con le fasi di creazione del progetto.

3.5 Metriche per la qualità

Per qualità del software, si intende la misura in cui un prodotto software soddisfa un certo numero di aspettative rispetto sia al suo funzionamento sia alla sua struttura interna. Stabilire la qualità di un software può essere un compito molto arduo, in quanto si basa su un insieme di parametri di qualità significativi, che devono essere poi misurati. Inoltre la qualità dipende dalle tecnologie e dalle metodologie con cui il software è realizzato. I parametri sono classificati in due famiglie: parametri esterni e parametri interni.
I primi si riferiscono alla qualità del software così come è percepita dai suoi utenti, e in-

[9]Grafico così chiamato in ricordo dell'ingegnere statunitense che si occupava di scienze sociali e che lo ideò nel 1917, Henry Laurence Gantt

cludono correttezza, affidabilità, robustezza, efficienza, usabilità. I secondi si riferiscono alla qualità del software così come è percepita dagli sviluppatori, e includono verificabilità, manutenibilità, riparabilità, evolvibilità, riusabilità, portabilità, leggibilità. Non raramente esiste una correlazione fra questi due aspetti (banalizzando: il software mal scritto tende anche a funzionare male).
La valutazione della qualità del progetto MyNotes si baserà su questi parametri. Di seguito sono riportate le valutazioni in base ai parametri esterni:

- Correttezza: un programma deve soddisfare la sua specifica, quindi i requisiti sono il fondamento su cui misurare la qualità e il software si dice corretto se rispetta i suoi requisiti;
- Affidabilità: un programma non deve presentare malfunzionamenti, quindi tanto più sono rari, tanto più il programma è affidabile;
- Robustezza: la robustezza di un sistema è la misura in cui il sistema si comporta in modo ragionevole in situazioni impreviste, non contemplate dalle specifiche. Situazioni di questo tipo possono essere dati di input scorretti, fallimenti di componenti software o hardware esterni al sistema e interagenti con esso, e così via;
- Efficienza: con questo parametro si vuole misurare se un sistema usa memoria, CPU e altre risorse in modo proporzionato ai servizi che svolge, ovvero senza sprechi;
- Integrità: un programma, durante la sua esecuzione, non deve danneggiare dati e/o risorse presenti nel sistema;
- Usabilità: un sistema viene definito facile da usare se un essere umano lo reputa tale. Questa è, quindi, una qualità soggettiva, che dipende sia dal contesto, sia dall'esperienza in cui il programma viene sottoposto. Può essere d'aiuto un'interfaccia utente semplice e amichevole di un'applicazione, ma anche in questo caso è la formazione e la cultura dell'utente a giudicare tale caratteristica.

Tutti questi parametri sono legati alle caratteristiche operative del programma. I parametri interni invece, sono legati alla capacità di subire modifiche. L'adattabilità a nuovi ambienti è trascurata, in quanto il programma sarà disponibile solo su piattaforma Android. I parametri da tenere in considerazione del primo tipo, sono:

- Manutenibilità: con questo parametro si indica che possono essere apportate modifiche, al sistema realizzato, con facilità;
- Riparabilità: un sistema è riparabile se la correzione degli errori è poco faticosa. Nel software la riparabilità si persegue attraverso la modularizzazione;
- Evolvibilità: il software, a differenza di altri prodotti ingegneristici è facilmente modificabile. È consigliabile tenere traccia dei cambiamenti effettuati, in quanto le specifiche vanno aggiornate. Infatti questo aspetto può rendere i cambiamenti futuri difficili da compiere.

Questi ultimi due parametri indicano, in breve, la flessibilità del software.

Capitolo 4

MyNotes: la fase progettuale

L'UML è un linguaggio di modellazione che consente di costruire modelli object-oriented per rappresentare e costruire domini di diverso genere. L'obbiettivo è quello di descrivere il comportamento e la struttura di un sistema software.
Questo modello è strutturato secondo un insieme di viste che rappresentano diversi aspetti, sia a scopo di analisi che di progettazione, mantenendo la tracciabilità dei concetti impiegati nelle diverse viste.
In questo capitolo viene descritta la modalità di realizzazione del progetto, applicando le regole di ingegneria del software. Quando si realizza un prodotto o un sistema software è importante svolgere una serie di passi, ovvero un percorso che aiuti il programmatore ad ottenere risultati di qualità in un tempo prefissato.
In questo caso, è stato utile seguire un percorso iterativo ed incrementale, al fine di valutare al meglio la convenienza e la logica dell'applicazione.

4.1 Diagramma dei casi d'uso

Gli Use Case Diagram (letteralmente tradotto diagrammi dei casi d'uso) sono diagrammi dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso.
Possiamo considerarli come uno strumento di rappresentazione dei requisiti funzionali

di un sistema. Di seguito viene presentato il diagramma dei casi d'uso dell'applicazione MyNotes, con gli utenti che operano come principali attori.

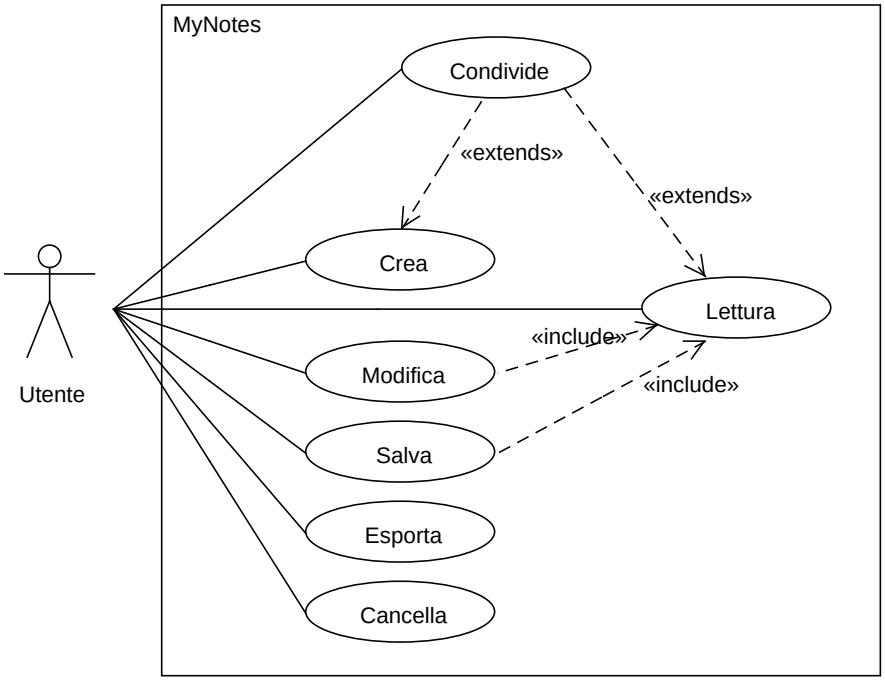


Figura 4.1: Il diagramma dei casi d'uso

4.1.1 Scenari dei casi d'uso

Gli scenari rappresentano le interazioni che intercorrono tra uno o più attori e il sistema. Di seguito vengono qui rappresentati solo alcuni scenari relativi all'utilizzo dell'applicazione MyNotes: la creazione di una nota, la condivisione di una nota e l'esportazione di una nota sulla memoria SD.

Caso d'uso	L'utente avvia una condivisione di una nota
Breve descrizione	Fase che riguarda la creazione di una connessione con un altro dispositivo per condividere una nota scelta dall'utente
Attori primari	Utente, sistema
Attori secondari	Altro utente
Precondizioni	I dispositivi devono essere muniti di tecnologia Bluetooth
Sequenza degli eventi	1. Il caso inizia dopo che l'utente avrà scelto di condividere la propria nota. 2. Se la rete Bluetooth è stata attivata, il sistema procede con la ricerca dei dispositivi. 3. Altrimenti, il sistema richiederà all'utente di attivare la rete Bluetooth; a) se l'utente rifiuta, il sistema termina l'avvio della condivisione di una nota; b) se l'utente accetta, il sistema avvia la ricerca di altri dispositivi. 4. Il sistema visualizza la lista dei dispositivi raggiungibili. 5. L'utente sceglie il dispositivo al quale connettersi: a) se il dispositivo scelto è stato accoppiato precedentemente, il sistema avvia la condivisione di una nota; b) altrimenti il sistema avvia l'operazione di accoppiamento dei dispositivi. 6. Il sistema stabilisce una connessione con un altro utente: a) aggiorna il titolo del programma, visualizzando il nome del dispositivo con cui si è connessi; b) invia all'utente collegato la nota condivisa.
Post-condizioni	Nessuna
Sequenza alternativa	Nessuna

Figura 4.2: Scenario: l'utente avvia la condivisione di una nota.

Caso d'uso	L'utente crea una nota
Breve descrizione	Fase iniziale che riguarda la creazione delle note
Attori primari	Utente, sistema
Attori secondari	Nessuno
Precondizioni	Nessuna
Sequenza degli eventi	1. Il caso d'uso inizia non appena l'utente apre il programma. 2. Se sono state create delle note, il sistema visualizzerà le note disponibili. 3. Altrimenti, il sistema visualizzerà un messaggio guidando l'utente alla creazione una nuova nota: a) all'utente verrà chiesto di premere il tasto menu; b) all'utente verrà chiesto di premere il tasto crea nota.
Post-condizioni	Il sistema resta in attesa di ricevere nuovi comandi
Sequenza alternativa	Nessuna

Caso d'uso	L'utente esporta la nota
Breve descrizione	Fase che riguarda l'esportazione della nota nella memoria SD
Attori primari	Utente, sistema
Attori secondari	Nessuno
Precondizioni	Deve essere presente nel dispositivo una scheda di memoria SD
Sequenza degli eventi	1. Il caso d'uso inizia non appena l'utente sceglie di esportare la nota, dal menu a comparsa. 2. Il sistema individua il percorso dove salvare la nota, sulla memoria. 3. Se il file era stato esportato in precedenza, il sistema sovrascrive il file. 4. Altrimenti il file verrà salvato normalmente.
Post-condizioni	Il file viene salvato
Sequenza alternativa	Nessuna

Figura 4.3: Scenari: l'utente crea una nuova nota (in alto) ed esporta una nota nella memoria SD.

4.2 Diagramma delle classi

Il diagramma delle classi (class diagram) è uno dei principali diagrammi UML. Viene utilizzato per descrivere tipi di entità con le loro caratteristiche, e le eventuali relazioni fra questi tipi. Si utilizza quindi il concetto di classe del paradigma object-oriented e altri correlati (per esempio la generalizzazione, che è una relazione concettuale assimilabile al meccanismo object-oriented dell'ereditarietà).
Di seguito viene presentato il diagramma delle classi relativo all'applicazione (Figura 4.4).

4.3 Diagramma di sequenza

Un Sequence Diagram è un diagramma utilizzato per descrivere uno scenario. Uno scenario è una determinata sequenza di azioni in cui tutte le scelte sono state già effettuate; in pratica nel diagramma non compaiono scelte, né flussi alternativi. Questi diagrammi sono strettamente collegati agli Activity Diagram, in quanto da essi possono derivare uno o più Sequence Diagram.
Se per esempio l'Activity Diagram descrive due flussi di azioni alternativi, da questi si potrebbero ricavare due scenari, e quindi due Sequence Diagram alternativi. In pratica, il Sequence Diagram descrive le relazioni che intercorrono in termini di messaggi tra Attori, Oggetti di business, Oggetti od Entità del sistema che si vuole rappresentare.
Lo scenario descritto dal diagramma di sequenza (Figura 4.5) riguarda la condivsione di una nota.

4.4 Diagramma di attività

L'Activity Diagram è un diagramma che definisce le attività da svolgere per realizzare una data funzionalità. Può essere utilizzato in fase di design, per dettagliare un determinato algoritmo. In teoria ad ogni Use Case Diagram dovrebbe corrispondere un Activity Diagram. Dettagliatamente, un Activity Diagram definisce una serie di attività o flusso, anche in termini di relazioni tra di esse, che è responsabile per la singola attività e i punti di decisione (Figura 4.6).

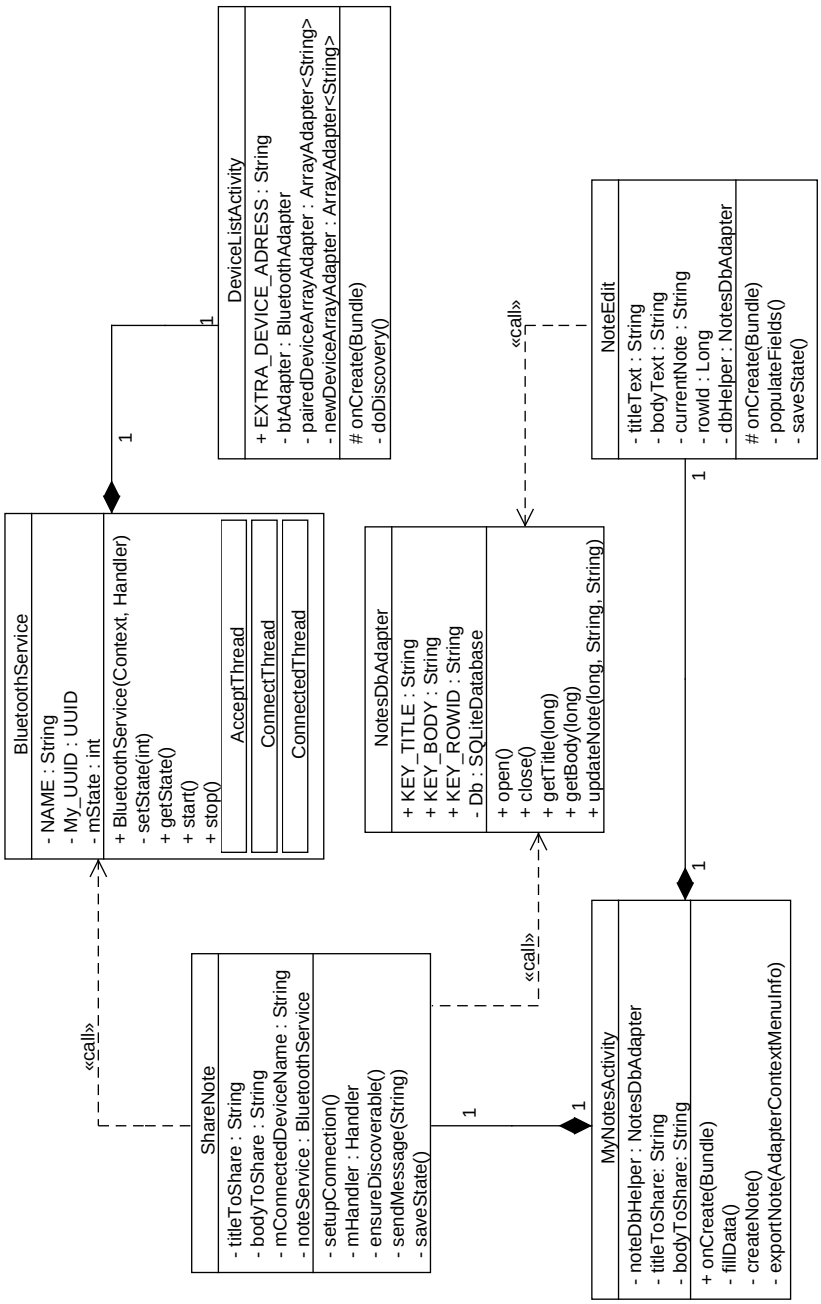


Figura 4.4: Il diagramma delle classi

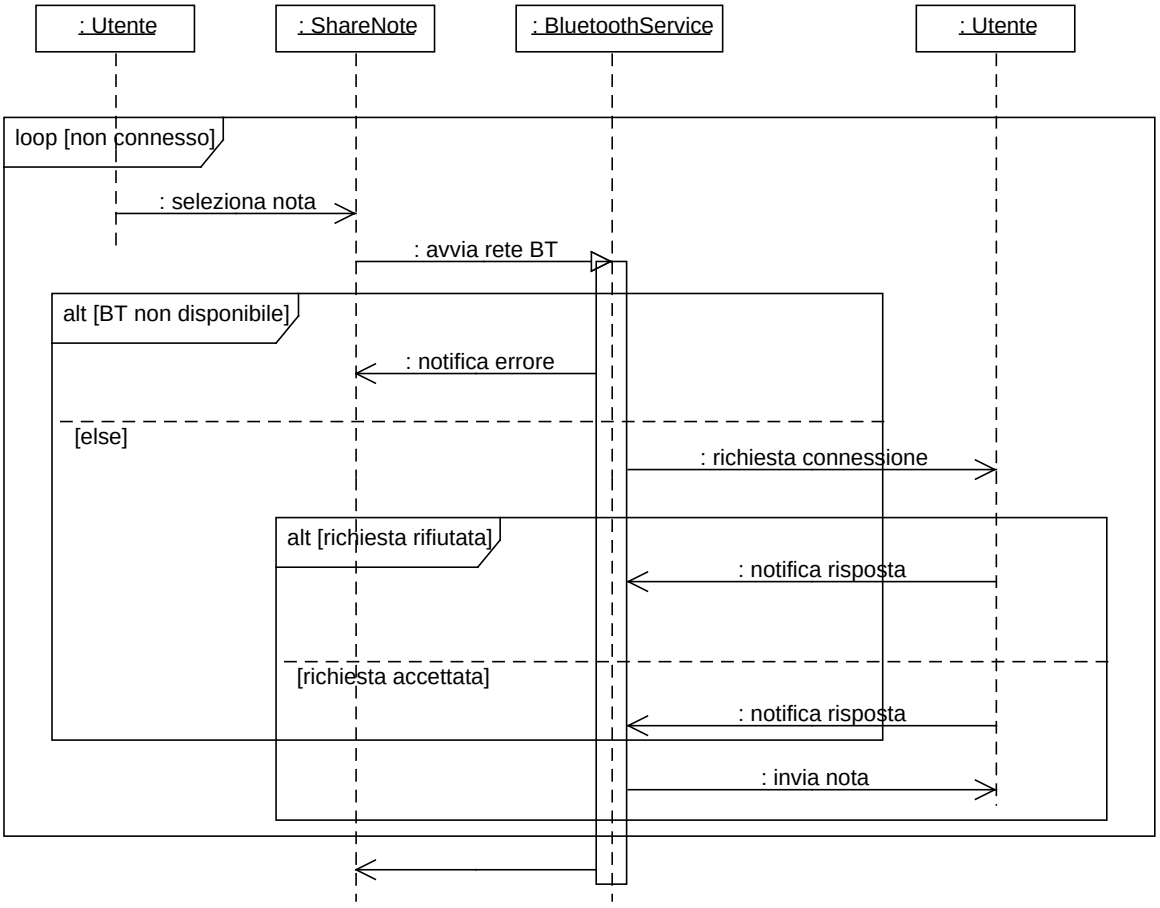


Figura 4.5: Il diagramma di sequenza

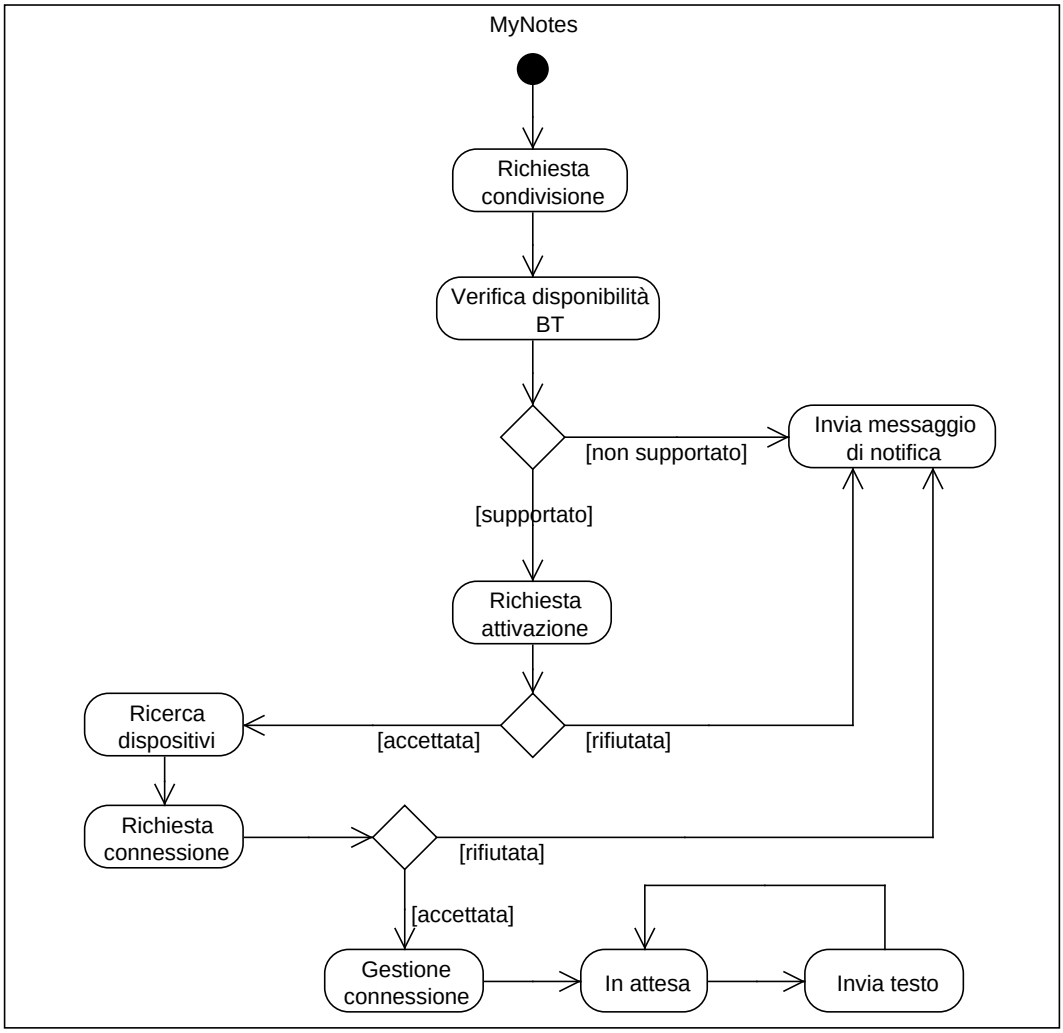


Figura 4.6: Il diagramma delle attività

4.5 Diagramma di stato

Lo State Chart Diagram è un diagramma utilizzato per descrivere il comportamento di entità o di classi in termini di stato (macchina a stati). Il diagramma mostra gli stati che sono assunti dall'entità o dalla classe in risposta ad eventi esterni. I diversi tipi di stati che un'entità o una classe può assumere, partendo dallo stato iniziale fino a quello finale, rappresentano il ciclo di vita della macchina.
Il diagramma di stato presente qui di seguito, rappresenta lo stato dell'utente all'avvio della condivisione di una nota.

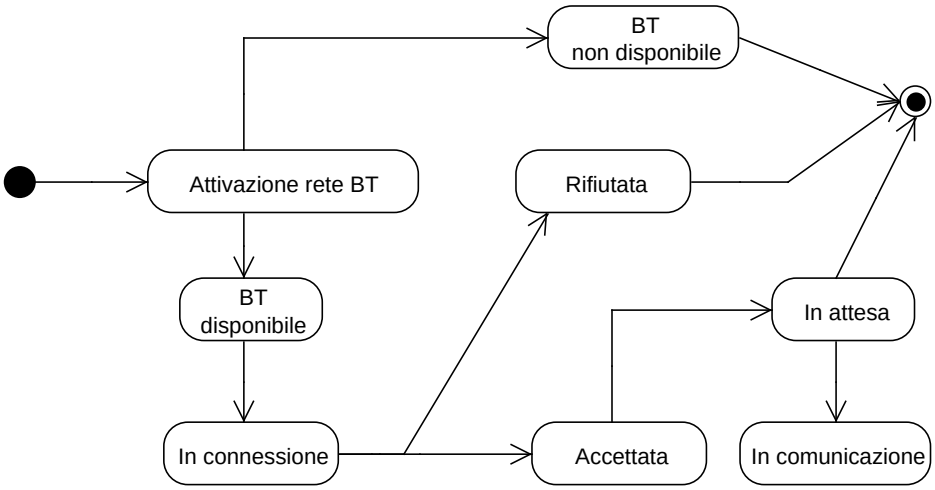


Figura 4.7: Il diagramma di stato

Capitolo 5

Funzionalità dell'applicazione

In questo capitolo viene presentato il codice sviluppato, con la relativa interfaccia grafica, per illustrare il funzionamento dell'applicazione MyNotes. Verranno descritte le capacità del programma, le sue caratteristiche, gli strumenti necessari alla sua installazione e alla sua esecuzione.
Inoltre è presente una sezione dedicata a parti cruciali del codice, con i relativi commenti. Questo, oltre a fornire ai lettori una completa visione del codice prodotto, servirà a comprendere meglio l'Android SDK.

5.1 L'installazione

Per un corretto funzionamento dell'applicazione, si richiede che il dispositivo Android, posseduto dall'utente, sia munito principalmente di tecnologia Bluetooth. L'applicazione richiede inoltre la piattaforma Android 2.1 (detta anche Eclair).
Nel processo di installazione del programma, all'utente verrà richiesto il permesso di consentire all'applicazione, di utilizzare alcuni servizi necessari alla sua esecuzione:

- Archiviazione: ovvero modificare/eliminare i contenuti della scheda SD;
- Comunicazione di rete: ovvero consentire connessioni Bluetooth;
- Strumenti di sistema: gestione del Bluetooth.

Nel caso un cui, l'utente non vorrà accettare le suddette condizioni di utilizzo, potrà annullare l'installazione (Figura 5.1).

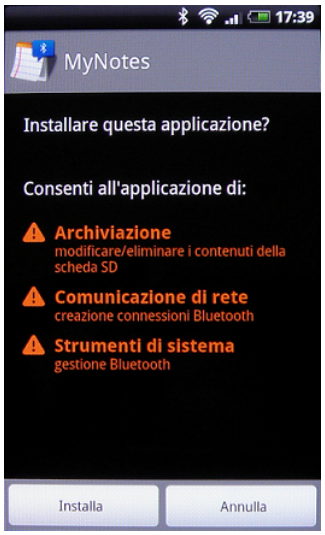


Figura 5.1: Fase di installazione del programma su un dispositivo Android.

Una volta installato e avviato, è il programma stesso a gestire l'eventuale indisponibilità di una memoria SD. Ciò non pone limiti all'utilizzo dell'applicazione, in quanto, come descritto prima, si dispone di un database interno. Mentre per quanto riguarda l'avvio di un connessione Bluetooth, il programma chiederà all'utente se vuole avviare o no il segnale Bluetooth, dopo aver rilevato la sua esistenza nel dispositivo (Figura 5.2).

Nella fase di test del programma sono stati utilizzati due dispositivi Android, con piattaforma 2.1 Eclair.

5.2 Informazioni generali

Dal menu delle applicazioni disponibili, si ricerca l'applicazione MyNotes (Figura 5.3) e si effettua il lancio.

Prima di passare ad una accurata descrizione, è bene sapere che un'applicazione Android si divide sostanzialmente in attività (Activity). Le attività rappresentano sostanzialmente una possibile interazione dell'utente con l'applicazione stessa. Per capire

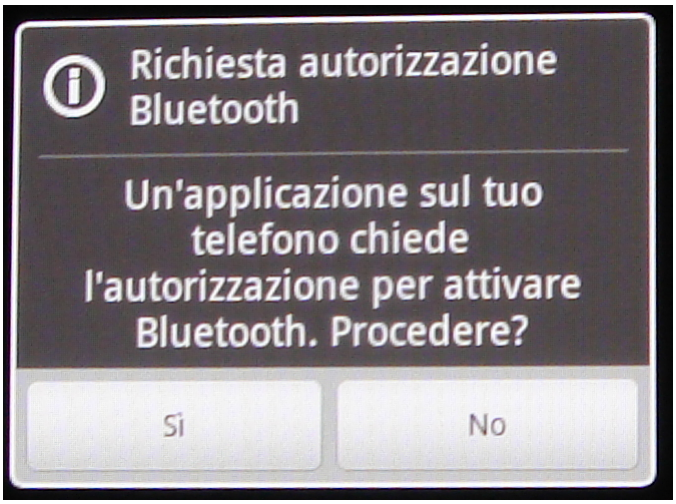


Figura 5.2: Richiesta all'utente dell'attivazione del Bluetooth, per consentire al programma di procedere alla condivisione.

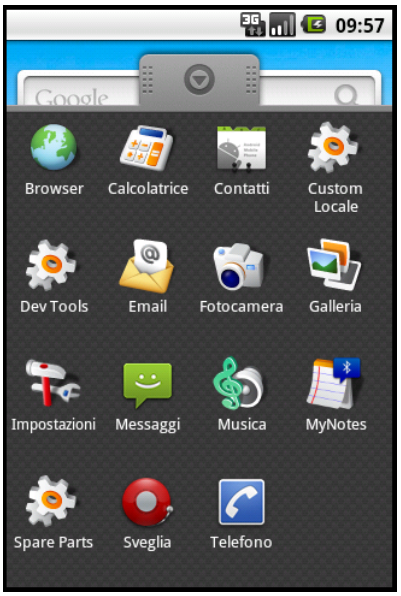


Figura 5.3: Menu principale di Android con la lista dei programmi disponibili, tra i quali MyNotes.

meglio di cosa stiamo parlando, possiamo associare l'attività al concetto di schermata. In una schermata potremo avere a disposizione componenti di sola visualizzazione insieme ad altri che permettono l'interazione con l'utente. Queste ultime, nel linguaggio di programmazione Android, vengono definite View, cioè viste.
MyNotes è suddivisa principalmente in tre attività, ognuna con le proprie funzionalità e collegate fra loro:

- l'attività principale chiamata 'MyNotesActivity', è il main dell'applicazione;
- l'attività di creazione/modifica di una nota, chiamata 'EditNote';
- infine l'attività di creazione/modifica di una nota in modo condiviso, 'ShareNote'.

Queste attività saranno approfondite meglio nelle seguenti sezioni, non prima di aver definito cosa è necessario per eseguire e far funzionare correttamente il programma.

5.3 La struttura del codice

In questa sezione verranno presentate, tutte le classi che compongono l'applicazione MyNotes. Prima di ciò, è necessario affrontare alcuni aspetti importanti riguardo il meccanismo di gestione del ciclo di vita di un'attività.
Ogni attività che si vuole creare in Android, deve essere necessariamente la specializzazione, diretta o indiretta, della classe Activity del package android.app. La ragione di tutto questo è la necessità, da parte del dispositivo, di poter gestire il ciclo di vita dei componenti in esecuzione. La gestione avviene attraverso l'invocazione di opportuni metodi di callback, di cui il componente deve essere provvisto.
Come descritto in precedenza, le attività di un'applicazione Android, hanno un proprio ciclo di vita. Esse possono assumere differenti stati, quando sono in esecuzione:

- ACTIVE: l'attività è in cima allo stack, ovvero è visibile e riceve gli eventi da parte dell'utente;
- PAUSED: l'attività non è attiva, ma è ancora visibile; non sarà sensibile agli eventi generati dall'utente, e verrà eliminata, per ragioni di spazio, solo se ne necessario;

- STOPPED: l'attività non è attiva né visibile, ciò la rende candidata ad essere eliminata dallo stack di memoria;

- INACTIVE: un'attività si trova in questo stato quando viene eliminata, oppure prima di essere creata.

I metodi di callback, che descrivono tutti i possibili passaggi di stato di una Activity, sono i seguenti:

- onCreate(): si tratta dell'operazione invocata in corrispondenza della creazione dell'attività, contenente le principali operazioni di inizializzazione;

- onStart(): se il metodo onCreate() termina con successo, l'attività esiste e si prepara in seguito, alla propria visualizzazione; il sistema invoca quindi il metodo onStart();

- onResume(): viene invocato se l'attività ha ottenuto o meno il focus, quindi è in cima allo stack per poter essere visualizzata nel display;

- onPause(): questo metodo viene invocato quando si passa da un'attività ad un'altra; si pensi per esempio alla pressione del tasto Back, del dispositivo, il quale provvederà ad immettere l'attività che si stava visualizzando nello stato di pausa;

- onRestart(): metodo invocato quando deve essere ripristinata un'attività, precedentemente fermata;

- onStop(): metodo invocato prima della eliminazione dell'attività;

- onDestroy(): metodo invocato per eliminare l'attività dallo stack, mettendola nello stato INACTIVE.

Per capire meglio tutti i possibili passaggi di stato di una Activity e i relativi metodi di callback, è utile consultare il seguente diagramma (Figura 5.4).

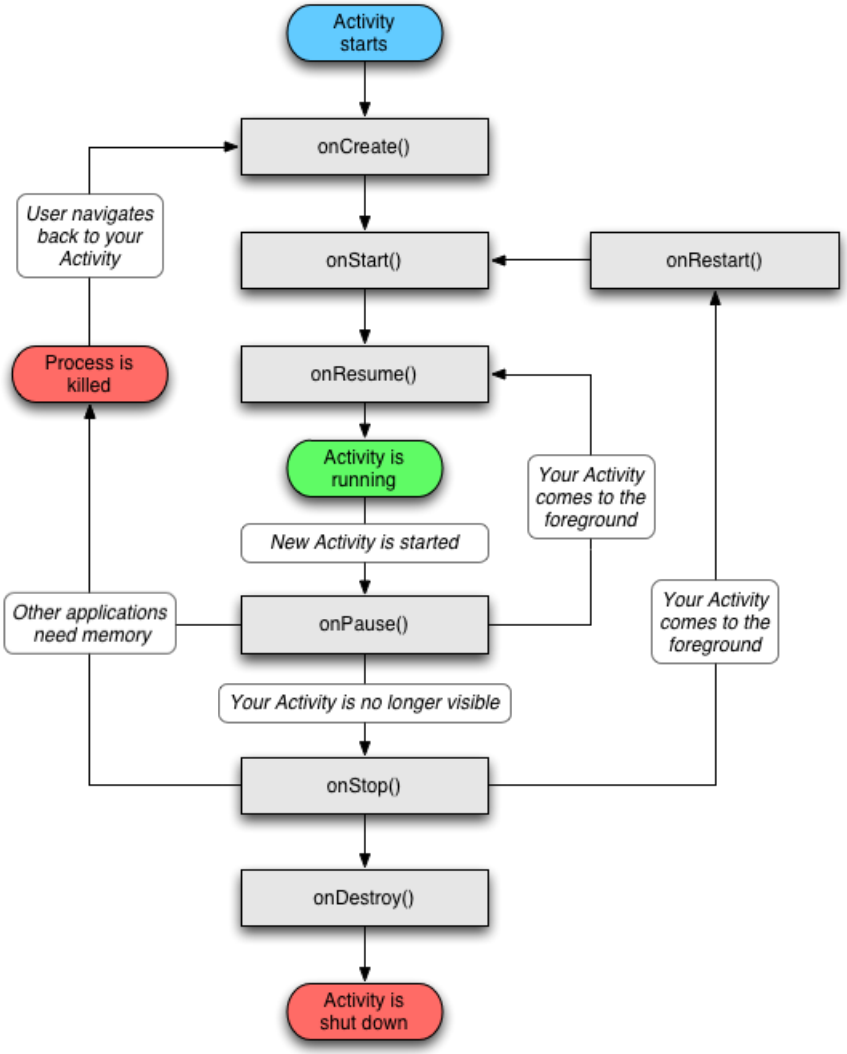


Figura 5.4: Rappresentazione del ciclo di vita di un'attività.

5.3.1 MyNotesActivity

L'attività principale del programma è la prima schermata con cui l'utente dovrà interagire. Al primo avvio del programma, ed in seguito in caso di note non ancora create, verrà visualizzato un messaggio, invitando l'utente a creare delle nuove note.
Accedendo al 'Menu' dell'applicazione, tramite l'apposito tasto disponibile sul dispositivo, è possibile creare una nuova nota, rendersi disponibili ad un collegamento via Bluetooth, visualizzare una breve informazione sul programma e ovviamente uscire da esso.

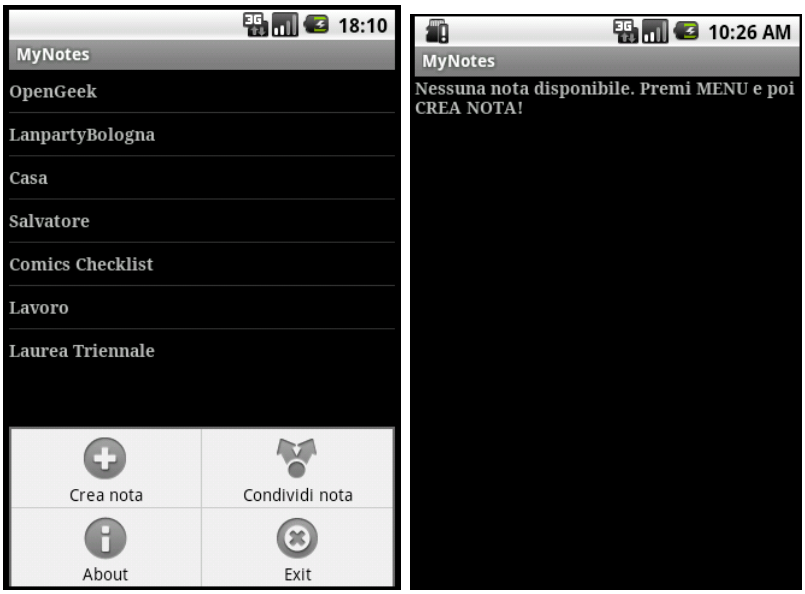


Figura 5.5: L'attività principale con o senza note disponibili.

Di seguito ecco alcune parti del codice, che richiedono un ulteriore approfondimento. Come tutte le attività, il metodo principale è onCreate().
Possiamo notare come al suo interno, la prima istruzione sia l'invocazione dell'analogo metodo della classe padre, cosa obbligatoria se non si vuole incorrere in un'eccezione. Il parametro Bundle fornisce un modo per ottenere il riferimento ad un eventuale stato che l'attività aveva prima di essere eliminata dal sistema.

```
@Override
/** Metodo chiamato al lancio dell'applicazione. */
```

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Imposto il layout dell'applicazione
    setContentView(R.layout.notes_list);
    // Creo e apro il database dove verranno salvate le note create
    noteDbHelper = new NotesDbAdapter(this);
    noteDbHelper.open();
    // Riempio i campi con le eventuali note disponibili
    fillData();
    registerForContextMenu(getListView());
}
```

Il metodo onCreateOptionMenu() fornisce all'attività un menu, alla quale accedere alle opzioni fornite dall'applicazione. Anche qui è da notare l'invocazione dell'analogo metodo della classe padre, seguito poi dall'aggiunta delle opzioni che si vogliono fornire.

```
@Override
/** Metodo usato per la creazione del menu dell'applicazione. */
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
    menu.add(0, INSERT_ID, 0, R.string.menu_insert).setIcon(R.
        drawable.ic_menu_add);
    menu.add(0, LAUNCH_EDIT_ID, 0, R.string.menu_launch_edit)
          .setIcon(R.drawable.ic_menu_share);
    menu.add(0, ABOUT_ID, 0, R.string.menu_about).setIcon(R.drawable
        .ic_menu_info_details);
    menu.add(0, EXIT_ID, 0, R.string.menu_exit).setIcon(R.drawable.
        ic_menu_close);
    return true;
}
```

Quest'ultimo metodo andrà sempre 'accoppiato' con il metodo onMenuItemSelected(), il quale permetterà di identificare l'opzione scelta dal menu e di lanciare l'azione desiderata.

```
@Override
```

```
/** Metodo usato per identificare l'opzione scelta dal menu. */
public boolean onMenuItemSelected(int featureId, MenuItem item) {
    switch(item.getItemId()) {
    case INSERT_ID:
      // Creo una nuova nota
      createNote();
      return true;
    case LAUNCH_EDIT_ID:
      // Avvio l'attivit\`a di editazione condivisa
      Intent i = new Intent(this, ShareNote.class);
      startActivity(i);
        return true;
    case ABOUT_ID:
      showDialog(DIALOG_ABOUT);
      return true;
    case EXIT_ID:
      // Chiudo l'applicazione
        finish();
        return true;
    }
    return super.onMenuItemSelected(featureId, item);
}
```

Il metodo fillData() è usato per visualizzare le note disponibili, in una lista. Questa lista in seguito sarà visibile sulla schermata, permettendo all'utente di accedere ad ogni singolo elemento.
Inizialmente verranno caricati i dati dal database, ed inseriti in un 'cursore' (Cursor sono una classe nativa di Android, usata per la gestione dei dati); infine l'attività inizierà la gestione del cursore stesso con startManagingCursor(). Prendendo come riferimento solo il titolo, si crea un array delle note disponibili e un array di identificatori delle note. Infine viene creato un adattatore e impostata la visualizzazione.

```
/** Metodo usato per visualizzare le note disponibili. */
private void fillData() {
    // Caricamento dei dati dal database e creazione delle righe
    Cursor notesCursor = noteDbHelper.fetchAllNotes();
    notesCursor = noteDbHelper.fetchAllNotes();
```

```
        startManagingCursor(notesCursor);

        // Creo un array delle note disponibili da visualizzare,
        // prendendo come riferimento solo il titolo
        String[] from = new String[]{NotesDbAdapter.KEY_TITLE};

        // Creo poi un array di identificatori delle note
        int[] to = new int[]{R.id.text};

        // Creo un cursore come adattatore e imposto la visualizzazione
        SimpleCursorAdapter notes =
              new SimpleCursorAdapter(this, R.layout.notes_row,
                  notesCursor, from, to);
        setListAdapter(notes);
    }
```

Il metodo createNote() è un metodo invocato a seguito della scelta da parte dell'utente, di creare una nuova nota.
Da esso verrà lanciata un'altra importante attività, ovvero NoteEdit. Verrà creato un Intent, ovvero una dichiarazione da parte di un'attività, di voler lanciare uno o più componenti, per poter collaborare utilizzando informazioni note a runtime. Infine verrà lanciato un altro metodo nativo delle classi Android, startActivityForResult(), il quale bloccherà l'attività principale, fino a quando l'attività che è stata lanciata, non termina, attendendo il risultato prodotto.
I risultati generati dalle attività devono essere gestite da un altro metodo nativo delle classi Android, onActivityResult(), le cui funzionalità saranno spiegate più avanti.

```
    /** Metodo usato per il lancio dell'attivit\'a di creazione di una
       nota. */
    private void createNote() {
        Intent i = new Intent(this, NoteEdit.class);
        startActivityForResult(i, ACTIVITY_CREATE);
    }
```

Il metodo exportNote() è anch'esso lanciato da menu, precisamente da un evento di long-click su una nota scelta (Figura 5.6). Viene usato per salvare la nota sulla scheda

SD del dispositivo, se quest'ultima è disponibile.
Prendendo come parametro l'id della nota selezionata, si procederà entrando in un blocco try-catch. Nel blocco try verrà impostato il percorso della cartella nella SD-Card; se è possibile scrivere sulla SD, si procede con l'esportazione, prendendo titolo e corpo della nota.
Qualora la scheda SD non dovesse essere presente nel dispositivo, o inaccessibile, nel blocco cacth sarà gestito l'errore generato, il quale verrà segnalato all'utente.

Figura 5.6: Menu di selezione dopo evento di long click su una nota.

```
/**
 * Metodo usato per salvare la nota sulla scheda SD del dispositivo,
 * se quest'ultima \'e disponibile.
 * @param info l'id della nota da salvare
 */
private void exportNote(AdapterContextMenuInfo info) {
  try {
  // Imposto il percorso della cartella nella SD-Card
      File root = Environment.getExternalStorageDirectory();
      // Se \'e possibile scrivere sulla SD, si procede con l'
         esportazione
```

```
        if (root.canWrite()){
          // Prendo titolo della nota
          File f = new File(root, noteDbHelper.getTitle(info.id) + ".
              txt");
            FileWriter fw = new FileWriter(f);
            // Prendo il corpo della nota
            BufferedWriter out = new BufferedWriter(fw);
            out.write(noteDbHelper.getBody(info.id));
            // Avviso il successo del salvataggio
            Toast.makeText(this, "Nota salvata", Toast.LENGTH_SHORT).
                show();
            out.close();
            }
        } catch (IOException e) {
          // In caso di errore nel salvataggio sar\`a visualizzato un
              avviso
          Toast.makeText(this, "Nota non salvata", Toast.LENGTH_SHORT)
              .show();
        }
}
```

Concludiamo l'esplorazione dell'attività principale, con altri due metodi fondamentali delle attività di Android, onListItemClick() e onActivityResult().
Il primo metodo, non è altro che un ascoltatore della lista delle note disponibili. Alla selezione dell'utente sulla nota scelta, esso provvederà al lancio dell'attività NoteEdit, permettendo la modifica della nota. Il risultato generato alla chiusura dell'attività NoteEdit, verrà in seguito gestito da onActivityResult().
Infatti quest'ultimo metodo, gestisce i risultati ottenuti dalle attività lanciate, invocando il metodo della classe padre ed infine effettuando un refresh sulle note disponibili.

```
@Override
/** Metodo usato per il lancio dell'attivit\`a di edit per una nota.
    */
protected void onListItemClick(ListView l, View v, int position,
    long id) {
     super.onListItemClick(l, v, position, id);
     Intent i = new Intent(this, NoteEdit.class);
```

```
        i.putExtra(NotesDbAdapter.KEY_ROWID, id);
        startActivityForResult(i, ACTIVITY_EDIT);
    }

    @Override
    /** Metodo usato dopo la chiusura di una attivit\'a lanciata,
        gestendo i risultati generati. */
    protected void onActivityResult(int requestCode, int resultCode,
        Intent intent) {
      super.onActivityResult(requestCode, resultCode, intent);
      fillData();
    }
```

5.3.2 NoteEdit

Questa è l'attività lanciata dalla schermata principale, il cui compito è quello di fornire all'utente i campi utili alla creazione e/o alla modifica di una nota (Figura 5.7).

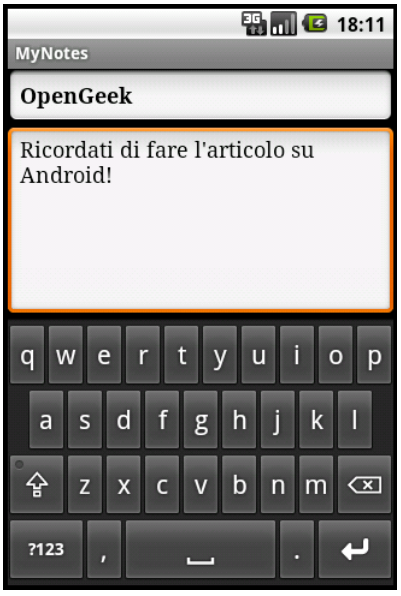


Figura 5.7: Attività NoteEdit così come si presenta all'utente.

Dal metodo onCreate() viene impostata la schermata con la quale interagire. In seguito viene invocato il metodo populateFields(), che preso l'id della nota scelta, caricherà negli appositi campi, il corpo e il titolo per iniziare le modifiche.
Nel caso in cui è stato scelto di creare una nuova nota, i campi saranno vuoti.

```
/** Caricamento del titolo e del corpo di una nota che si vuole
   modificare. */
private void populateFields() {
    if (rowId != null) {
        Cursor note = dbHelper.fetchNote(rowId);
        startManagingCursor(note);
        titleText.setText(note.getString(
                    note.getColumnIndexOrThrow(NotesDbAdapter.
                        KEY_TITLE)));
        bodyText.setText(note.getString(
                note.getColumnIndexOrThrow(NotesDbAdapter.KEY_BODY))
                    );
        String title = titleText.getText().toString();
        String body = bodyText.getText().toString();
        currentNote = title + "<title>" + body;
    }
}
```

Nel caso in cui l'attività non sia più in cima allo stack, e quindi non abbia il focus da parte del dispositivo, verrà chiamato il metodo onSaveInstanceState(). Questo metodo permette il completo ripristino dell'attività.

```
@Override
/** Metodo usato per il salvataggio dell'attivit\'a. */
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putLong(NotesDbAdapter.KEY_ROWID, rowId);
}
```

L'ultimo metodo che merita un'approfondimento è il metodo saveState(). Questo metodo permette di salvare la nota che si è creata, o che si sta modificando, anche nei momenti in cui l'attività EditNote dovesse essere messa in pausa o fermata.

```
/** Metodo usato per il salvataggio della nota sul database. */
private void saveState() {
    // Prendo titolo e corpo della nota
    String title = titleText.getText().toString();
    String body = bodyText.getText().toString();

    // Se i campi sono vuoti la nota non viene salvata
    if (title.length() == 0 && body.length() == 0){
      Toast.makeText(this, "Nota non valida", Toast.LENGTH_SHORT).
          show();;
    }
    else {
      // Se il titolo \`e vuoto, viene impostato un titolo di
          default
      if (title.length() == 0) {
          title = "No title";
          }
      if (rowId == null) {
        long id = dbHelper.createNote(title, body);
        if (id > 0) {
          rowId = id;
        }
      } else {
        dbHelper.updateNote(rowId, title, body);
        }
    }
    currentNote = title + "<title>" + body;
}
```

5.3.3 ShareNote

Questa attività è la schermata con il quale sarà possibile avviare una connessione e di seguito condividere una nota (Figura 5.8).

Il metodo onCreate() a differenza degli omonimi implementati nelle altre attività effettua, oltre alle operazioni di inizializzazione, il controllo della disponibilità del modulo di trasmissione Bluetooth. Se il dispositivo utilizzato dispone del modulo, allora

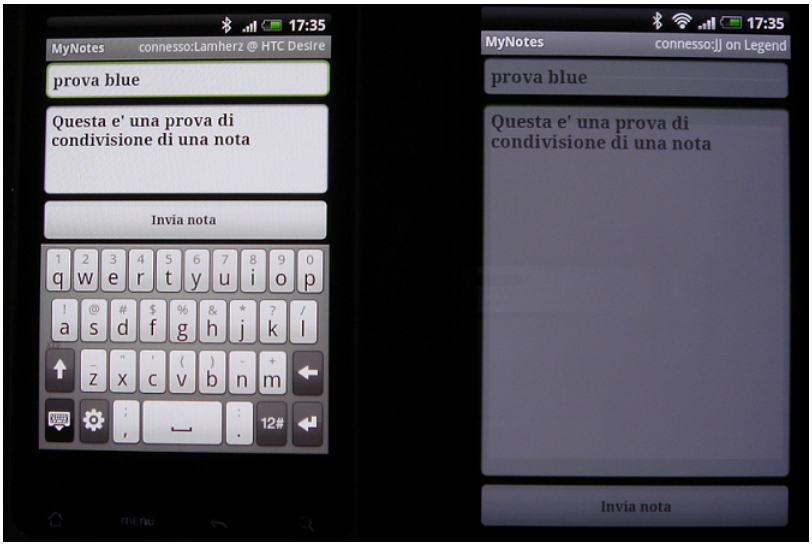


Figura 5.8: Attività ShareNote vista attraverso due dispositivi con piattaforma Android.

sarà possibile effettuare una condivisione della nota; altrimenti il programma riporterà l'utente alla schermata principale, segnalando la mancanza del modulo.

```
@Override
/** Metodo chiamato al lancio dell'attivit\'a. */
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Apro il database
    dbHelper = new NotesDbAdapter(this);
    dbHelper.open();

    // Impostazione del layout
    requestWindowFeature(Window.FEATURE_CUSTOM_TITLE);
    setContentView(R.layout.main_share_note);
    getWindow().setFeatureInt(Window.FEATURE_CUSTOM_TITLE, R.layout.
       custom_title);

    // Impostazione del titolo
    appTitle = (TextView) findViewById(R.id.title_left_text);
    appTitle.setText(R.string.app_name);
```

```
        appTitle = (TextView) findViewById(R.id.title_right_text);

        // Impostazione del Bluetooth locale
        bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

        // Se il Bluetooth non viene trovato, il terminale non lo
            supporta
        if (bluetoothAdapter == null) {
            Toast.makeText(this, "Bluetooth non disponibile", Toast.
                LENGTH_LONG).show();
            finish();
            return;
        }
    }
```

I metodi onStart() e onResume() hanno una implementazione differente in questa attività. Il primo verificherà che il servizio Bluetooth sia attivo. Nel caso in cui il servizio sia disattivato, sarà visualizzata una schermata di dialogo che richiedera l'attivazione da parte dell'utente. Dopo la sua attivazione, sarà invocato il metodo setupConnection(). Il metodo onResume() effettua un ulteriore verifica dell'attivazione del servizio Bluetooth, in modo da garantire il completo ripristino non solo dell'attività, ma anche del servizio di connessione.

```
    @Override
    /** Metodo usato per il lancio dell'attivit\'a. */
    public void onStart() {
        super.onStart();

        // Se il BT non \'e attivato, si effettua una richiesta di
            attivazione.
        // setupConnection() sar\'a poi chiamata nel metodo
            onActivityResult()
        if (!bluetoothAdapter.isEnabled()) {
            Intent enableIntent = new Intent(BluetoothAdapter.
                ACTION_REQUEST_ENABLE);
            startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
        // Altrimenti verr\'a impostata la sessione di condivisione
```

```
            della nota
        } else {
            if (noteService == null)
              setupConnection();
        }
    }

    @Override
    /** Metodo usato per il ripristino dell'attivit\'a. */
    public synchronized void onResume() {
        super.onResume();

        // Questo controllo serve nel caso in cui il BT non \'e stato
            abilitato nel metodo
        // onStart(), quindi l'applicazione va in pausa;
        // onResume() sar\'a chiamato quando sar\'a ritornato
            ACTION_REQUEST_ENABLE.
        if (noteService != null) {
            // Solo se lo stato \'e STATE_NONE, sappiamo che l'
                applicazione non \'e partita
            if (noteService.getState() == BluetoothService.STATE_NONE) {
              // Inizializzazione dell'applicazione
              noteService.start();
            }
        }
    }
```

Il metodo usato per avviare la connessione è setupConnection().
Nella prima fase vengono recuperati il titolo e il corpo della nota da condividere, che verranno poi inseriti nei loro campi. In seguito verrà inizializzato il bottone di invio dei dati con il proprio ascoltatore dell'evento. Infine si avvia la classe BluetoothService per gestire la connessione.

```
    /** Metodo usato per avviare una connessione Bluetooth. */
    private void setupConnection() {

        // Recupero i dati della nota da condividere
        Intent intent = getIntent();
```

```
String pkg = getPackageName();
titleToShare = intent.getStringExtra(pkg + ".myTitle");
bodyToShare = intent.getStringExtra(pkg + ".myBody");

// Inizializzazione dei campi per il testo con un listener per
   le chiavi di ritorno
titleOutEditText = (EditText) findViewById(R.id.shared_title);
bodyOutEditText = (EditText) findViewById(R.id.shared_body);

// Prendo il titolo e il corpo della nota scelta
titleOutEditText.setText(titleToShare);
bodyOutEditText.setText(bodyToShare);

// Inizializzazione del bottone di invio con un ascoltatore
sendButton = (Button) findViewById(R.id.button_send);
sendButton.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        // Invio titolo e corpo della nota modificata
      String note = titleOutEditText.getText().toString() + " <
         title> "
            + bodyOutEditText.getText().toString().trim();
      sendMessage(note);
      onBlock(false);
    }
});

...

// Inizializzazione della classe BluetoothService per gestire la
    connessione BT
noteService = new BluetoothService(this, serviceHandler);
```

In fase di condivisione, le modifiche effettuate da un utente sul documento, escluderanno l'utente con cui si è connessi ad effettuarne in contemporanea. Per implementare garantire quindi ordine nei momenti di modifica del documento, è stato implementato il metodo onBlock(), il quale attiverà/disattiverà la schermata dell'utente che non starà lavorando sul documento.

```
/** Metodo usato per fermare l'attivit\'a di modifica all'utente che
    ha modificato la nota. */
public void onBlock(boolean flag) {
  if (flag == false) {
      titleOutEditText.setEnabled(false);
      bodyOutEditText.setEnabled(false);
      sendButton.setEnabled(false);
  }
  else {
      titleOutEditText.setEnabled(true);
      bodyOutEditText.setEnabled(true);
      sendButton.setEnabled(true);
  }
}
```

L'invio e il ricevimento di una nota viene trasmesso in byte, ed è quindi utile avere un metodo che provvede al riconoscimento del titolo e del corpo del documento. Il metodo onRead() viene chiamato dal gestore degli eventi mHandler, e dopo aver identificato il titolo e il corpo della nota, riempirà i campi della schermata.

```
/**
 * Metodo usato per leggere la nota inviata da un utente.
 * @param note la nota da leggere e da spezzare in titolo e corpo
 */
public void onRead(String note) {
// Preparo degli array che conterranno la nota, il suo titolo e il
   suo corpo
ArrayList<String> myNote = new ArrayList<String>();
ArrayList<String> myTitle = new ArrayList<String>();
ArrayList<String> myBody = new ArrayList<String>();
StringTokenizer token = new StringTokenizer(note, " ");
titleToShare = "";
bodyToShare = "";

while (token.hasMoreTokens() == true) {
  myNote.add(token.nextToken());
}
```

```
// Salvo negli array il titolo e corpo della nota
for (int i = 0; i < myNote.size() ; i++) {
  String word = myNote.get(i);
  if (word.equalsIgnoreCase("<title>")) {
    for (int j = i+1; j < myNote.size() ; j++) {
      myBody.add(myNote.get(j));
    }
    break;
  }
  else {
    myTitle.add(word);
  }
}

for (int i = 0; i < myTitle.size() ; i++) {
  String word = myTitle.get(i);
  titleToShare = titleToShare + word + " ";
}

for (int i = 0; i < myBody.size() ; i++) {
  String word = myBody.get(i);
  bodyToShare = bodyToShare + word + " ";
}

    // Inserisco titolo e corpo della nota nei loro campi
  titleOutEditText.setText(titleToShare);
  bodyOutEditText.setText(bodyToShare);
}
```

Nel caso in cui i dispositivi non sono mai stati accoppiati precedentemente, è necessario fornire all'utente un metodo veloce per attivare la visibilità del proprio dispositivo. Evitando quindi l'uscita dall'applicazione, tramite il menu è possibile attivare la visibilità del proprio dispositivo in modo da essere rilevabile da altri.

```
/** Metodo usato per attivare la visualizzazione del proprio
   dispositivo, verso altri. */
private void ensureDiscoverable() {
    if (mBluetoothAdapter.getScanMode() !=
```

```
            BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE) {
            Intent discoverableIntent = new Intent(BluetoothAdapter.
                ACTION_REQUEST_DISCOVERABLE);
            discoverableIntent.putExtra(BluetoothAdapter.
                EXTRA_DISCOVERABLE_DURATION, 300);
            startActivity(discoverableIntent);
        }
    }
```

Il metodo sendNote() si occupa dell'invio della nota modificata all'utente con sui si è connessi. Prima di procedere all'invio, viene effettuato un controllo sullo stato della connessione dei dispositivi. Infine sarà comunicato alla classe BluetoothService di inviare la nota in byte.

```
    /**
     * Metodo che invia titolo e corpo della nota.
     * @param message una stringa che rappresenta il titolo da mandare.
     */
    private void sendNote(String message) {
        // Controlla che i dispositivi siano connessi prima di fare
            altre operazioni
        if (noteService.getState() != BluetoothService.STATE_CONNECTED)
            {
            Toast.makeText(this, "Nessuna connessione", Toast.
                LENGTH_SHORT).show();
            return;
        }

        // Prendo il messaggio e dico alla classe BluetoothService di
            scriverlo
        byte[] send = message.getBytes();
        noteService.write(send);
    }
```

La creazione di un thread in Android è un'operazione semplice. I problemi si hanno però quando si tratta di thread con la responsabilità di procurare delle informazioni da visualizzare all'interno di componenti grafici contenuti in una Activity. Per risolvere

questo problema Android mette a disposizione un piccolo framework per semplificare l'interazione tra thread diversi di una stessa applicazione.
In questo caso si è utilizzato un Handler, classe disponibile nel package android.os. Questa classe viene utilizzata nell'applicazione per ricevere e gestire i messaggi contenenti informazioni da utilizzare per l'aggiornamento dei componenti della UI.
Il primo passo consiste nella creazione dell'Handler che dovrà elaborare le eventuali informazioni, provenienti dalla classe BluetoothService. Queste informazioni sono contenute all'interno di un oggetto di tipo Message e dovranno essere elaborate. Facendo quindi un override del metodo handleMessage(), che viene invocato in corrispondenza della ricezione di un Message, possiamo gestire:

- i cambiamenti di stato della connessione;
- i casi di scrittura e lettura di un messaggio;
- gli avvisi e le notifiche;
- il messaggio contenente il nome del dispositivo al quale si è connessi.

```
// Gestore delle informazioni tornate dalla classe BluetoothService
private final Handler serviceHandler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {
        // Caso del cambio di stato dell'applicazione
        case MESSAGE_STATE_CHANGE:
            switch (msg.arg1) {
            case BluetoothService.STATE_CONNECTED:
                mTitle.setText(R.string.title_connected_to);
                mTitle.append(mConnectedDeviceName);
                break;
            case BluetoothService.STATE_CONNECTING:
                mTitle.setText(R.string.title_connecting);
                break;
            case BluetoothService.STATE_LISTEN:
            case BluetoothService.STATE_NONE:
                mTitle.setText(R.string.title_not_connected);
```

```
                break;
            }
            break;
        // Caso di scrittura del messaggio
        case MESSAGE_WRITE:
          bodyToShare = (String)bodyOutEditText.getText().toString()
              .trim();
          titleToShare = (String)titleOutEditText.getText().toString
              ().trim();
            break;
        // Caso di lettura di un messaggio
        case MESSAGE_READ:
          byte[] writeBuf = (byte[]) msg.obj;
            // Recupero i dati della nota in formato stringa, e li
               imposto nel layout
            String receivedNote = new String(writeBuf).trim();
            onRead(receivedNote);
          onBlock(true);
            break;
        // Caso di un messaggio contenente il nome del terminale
        case MESSAGE_DEVICE_NAME:
            // salva il nome del terminale connesso
            mConnectedDeviceName = msg.getData().getString(
               DEVICE_NAME);
            Toast.makeText(getApplicationContext(), "Connected to "
                           + mConnectedDeviceName, Toast.
                               LENGTH_SHORT).show();
            break;
        // Caso riguardante un messaggio di notifica
        case MESSAGE_TOAST:
            Toast.makeText(getApplicationContext(), msg.getData().
               getString(TOAST),
                           Toast.LENGTH_SHORT).show();
            break;
        }
    }
};
```

Anche durante la condivisione di una nota è fornita la possibilità di salvare i dati nel database dell'applicazione, analogamente a quanto avviene nell'attività NoteEdit, tramite il metodo savaState().

5.3.4 La classe BluetoothService

BluetoothService è la classe che gestisce il lavoro riguardante le connessioni Bluetooth tra i dispositivi.
Per gestire i thread delle connessioni sono state create tre classi:

- AcceptThread: gestisce il thread di ascolto per le connessioni in entrata;
- ConnectThread: gestisce il thread della connessione tra due dispositivi;
- ConnectedThread: gestisce la trasmissione dei dati durante la connessione.

In fase di creazione, il costruttore della classe provvede ad:

- istanziare il modulo di trasmissione Bluetooth presente nel dispositivo;
- impostare lo stato della connessione;
- impostare il gestore dei messaggi dei thread.

```
/**
 * Costruttore della classe BluetoothService.
 * Prepara una nuova connessione tramite BT.
 * @param context  l'UI dell'attivit\`a
 * @param handler  un gestore dei messaggi da inviare all'UI
 */
public BluetoothService(Context context, Handler handler) {
    adapter = BluetoothAdapter.getDefaultAdapter();
    connectionState = STATE_NONE;
    connectionHandler = handler;
}
```

Il metodo setState() viene invocato per impostare lo stato della connessione.
Esso fornisce allo Handler un messaggio di un eventuale cambiamento, che verrà poi

comunicato all'attività ShareNote. Il metodo getState() serve a richiedere lo stato della connessione.

```
/**
 * Metodo che imposta lo stato della connessione.
 * @param state un intero che definisce lo stato della connessione
 */
private synchronized void setState(int state) {
    connectionState = state;
    // Fornisco al gestore il nuovo stato, in modo che l'attivit\'a
       UI si aggiorni
    connectionHandler.obtainMessage(ShareNote.MESSAGE_STATE_CHANGE,
       state, -1)
                .sendToTarget();
}

/** Metodo che ritorna l'attuale stato della connessione. */
public synchronized int getState() {
    return connectionState;
}
```

Il metodo start() è invocato per avviare il servizio di connessione.
Vengono cancellati eventuali thread avviati in precedenza (ConnectThread e ConnectedThread), per poi avviare AcceptThread. Lo stato del dispositivo viene impostato nella modalità 'in ascolto', in attesa di una connessione.

```
/**
 * Avvio del servizio di connessione. Viene avviato AcceptThread,
    per
 * incominciare una sessione di ascolto nella modalit\'a server.
    Questo metodo viene
 * chiamato dal metodo onResume().
 */
public synchronized void start() {

    // Cancello qualsiasi thread che cerca di creare una connessione
    if (connectThread != null) {connectThread.cancel();
       connectThread = null;}
```

```
        // Cancello qualsiasi thread che sta gestendo una connessione
        if (connectedThread != null) {connectedThread.cancel();
            connectedThread = null;}
        // Avvio un thread di ascolto tramite BluetoothServerSocket
        if (acceptThread == null) {
            acceptThread = new AcceptThread();
            acceptThread.start();
        }
        setState(STATE_LISTEN);
    }
```

Il metodo connect() prendendo come parametro un dispositivo al quale connettersi, avvia una connessione Bluetooth.
I thread avviati verranno cancellati in modo da avviare il nuovo thread (ConnectThread) per connettersi al dispositivo scelto.

```
    /**
     * Metodo che avvia ConnectThread per avviare una connessione
        tramite un dispositivo remoto.
     * @param device il dispositivo bluetooth da connettere
     */
    public synchronized void connect(BluetoothDevice device) {

        // Cancello qualsiasi thread che cerca di creare una connessione
        if (mState == STATE_CONNECTING) {
            if (mConnectThread != null) {mConnectThread.cancel();
                mConnectThread = null;}
        }
        // Cancello qualsiasi thread che sta gestendo una connessione
        if (mConnectedThread != null) {mConnectedThread.cancel();
            mConnectedThread = null;}
        // Avvio il nuovo thread per connettermi col dispositivo dato
        mConnectThread = new ConnectThread(device);
        mConnectThread.start();
        setState(STATE_CONNECTING);
    }
```

Il metodo connected() gestisce la connessione stabilita, prendendo come parametro il socket e il dispositivo. La procedura è la seguente:

- vengono cancellati i thread non più utili;
- viene avviata la classe interna ConnectThread;
- viene notificato il nome del dispositivo a cui si è connessi all'utente;
- imposta lo stato del dispositivo a 'connesso'.

```
/**
 * Metodo che avvia ConnectedThread() per iniziare a gestire una
    connessione BT.
 * @param socket il socket BT con il quale \'e stata fatta la
    connessione
 * @param device il dispositivo BT con cui si \'e connessi
 */
public synchronized void connected(BluetoothSocket socket,
   BluetoothDevice device) {

    // Cancello il thread che ha completato la connessione
    if (connectThread != null) {connectThread.cancel();
       connectThread = null;}
    // Cancello qualsiasi thread che sta gestendo una connessione
    if (connectedThread != null) {connectedThread.cancel();
       connectedThread = null;}
    // Cancello gli altri thread perch\'e voglio stabilire solo una
       connessione
    if (acceptThread != null) {acceptThread.cancel(); acceptThread =
        null;}
    // Avvio il thread per gestire la connessione e le trasmissioni
    connectedThread = new ConnectedThread(socket);
    connectedThread.start();

    // Invio il nome del dispositivo connesso all'attivit\'a UI
    Message msg = connectionHandler.obtainMessage(ShareNote.
       MESSAGE_DEVICE_NAME);
```

```
        Bundle bundle = new Bundle();
        bundle.putString(MyNotesActivity.DEVICE_NAME, device.getName());
        msg.setData(bundle);
        connectionHandler.sendMessage(msg);

        setState(STATE_CONNECTED);
    }
```

Il metodo write() comunica al ConnecteThread la nota da inviare agli utenti in modo asincrono.
Prima dell'invio viene effettuato un controllo sullo stato della connessione.

```
    /**
     * Metodo che scrive al ConnectedThread in modo asincrono.
     * @param out i byte da scrivere
     */
    public void write(byte[] outNote) {
        // Creo un oggetto temporaneo
        ConnectedThread r;
        // Sincronizzo una copia della ConnectedThread
        synchronized (this) {
            if (connectionState != STATE_CONNECTED) return;
            r = connectedThread;
        }
        // Eseguo la scrittura asincrona
        r.write(outNote);
    }
```

Il fallimento di una connessione viene notificata all'utente tramite il metodo connectionFailed(). Dopo aver impostato lo stato del dispositivo 'in ascolto', invia la notifica all'attività ShareNote. Questo metodo viene invocato dalla classe ConnectThread.
Invece la perdita di una connessione viene notificata dal metodo connectionLost(), che effettua la stessa procedura. Viene però invocato dalla classe ConnectedThread.

```
    /** Metodo che indica che la connessione \'e fallita e crea un
       avviso. */
    private void connectionFailed() {
        setState(STATE_LISTEN);
```

```
        // Mando un messaggio di notifica all'attivit\'a
        Message msg = connectionHandler.obtainMessage(ShareNote.
            MESSAGE_TOAST);
        Bundle bundle = new Bundle();
        bundle.putString(MyNotesActivity.TOAST, "Impossibile connettersi
            ");
        msg.setData(bundle);
        connectionHandler.sendMessage(msg);
    }

    /** Metodo che indica che la connessione si \'e persa e crea un
       avviso. */
    private void connectionLost() {
        setState(STATE_LISTEN);

        // Mando un messaggio di notifica all'attivit\'a
        Message msg = connectionHandler.obtainMessage(ShareNote.
            MESSAGE_TOAST);
        Bundle bundle = new Bundle();
        bundle.putString(MyNotesActivity.TOAST, "Connessione persa");
        msg.setData(bundle);
        connectionHandler.sendMessage(msg);
    }
```

Come parte finale della classe, troviamo i thread usati per creare e gestire le connessioni tra i dispositivi, che sono gestiti da classi interne.
La classe AcceptThread definisce il thread di ascolto per le connessioni. Si comporta come un client da lato server ed è attivo fino a quando la connessione non è accettata o cancellata. Al lancio del thread viene effettuato un controllo per verificare che il dispositivo non sia già connesso. Nel caso in cui è stata accettata una connessione, verrà avviato ConnectedThread per la sua gestione.

```
    /**
     * Questo \'e il thread di ascolto per le connessioni.
     * Si comporta come un client da lato server. Funziona fino a quando
         la connessione
```

```
 * non \'e accettata (o fino alla sua cancellazione).
 */
private class AcceptThread extends Thread {
    // Server socket locale
    private final BluetoothServerSocket mmServerSocket;

    public AcceptThread() {
        BluetoothServerSocket tmp = null;

        // Creo un nuovo server socket in ascolto
        try {
            tmp = adapter.listenUsingRfcommWithServiceRecord(NAME,
               MY_UUID);
        } catch (IOException e) {}
        mmServerSocket = tmp;
    }

    public void run() {
        setName("AcceptThread");
        BluetoothSocket socket = null;

        // Ascolto il server socket se non si \'e connessi
        while (connectionState != STATE_CONNECTED) {
            try {
                // Questa \'e una chiamata di blocco e restituisce
                   una connessione effettuata
              // con successo o un'eccezione
                socket = mmServerSocket.accept();
            } catch (IOException e) {
                break;
            }

            // Se una connessione \'e stata accettata...
            if (socket != null) {
                synchronized (BluetoothService.this) {
                    switch (connectionState) {
                    case STATE_LISTEN:
                    case STATE_CONNECTING:
```

```
                        // Caso normale, avvio il thread di
                           connessione.
                        connected(socket, socket.getRemoteDevice());
                        break;
                    case STATE_NONE:
                    case STATE_CONNECTED:
                        // Caso in cui la connessione non \'e pronta
                            o si \'e gi\'a connessi.
                      // Chiudo il nuovo socket.
                        try {
                            socket.close();
                        } catch (IOException e) {}
                        break;
                    }
                }
            }
        }

        /** Metodo che cancella il thread creato. */
        public void cancel() {
          try {
                mmServerSocket.close();
            } catch (IOException e) {}
        }
    }
```

ConnectThread è un thread attivo finché si cerca di creare una connessione in uscita con un dispositivo. E' usato dall'applicazione quando viene lanciata in modalità server, ovvero quando l'utente ha scelto di condividere una propria nota, con un altro utente. Il costruttore creerà un socket per effettuare la connessione ed al lancio del thread verrà cancellata la ricerca di altri dispositivi, in modo da stabilire la connessione. Se la connessione è fallita, verrà riavviata la modalità di ascolto. Infine verrà lanciato ConnectedThread per la gestione della connessione.

```
    /**
```

```
 * Questo thread \‘e attivo fich\’e si cerca di creare una
    connessione in uscita con un
 * dispositivo. Rimane attivo sia se la connessione ha successo o
    fallisce.
 * E’ usato dall’applicazione quando viene lanciata in modalit\‘a
    server, ovvero
 * l’utente ha scelto di condividere una propria nota, con un altro
    utente.
 */
private class ConnectThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final BluetoothDevice mmDevice;

    public ConnectThread(BluetoothDevice device) {
        mmDevice = device;
        BluetoothSocket tmp = null;

        // Prendo un socket BT (BluetoothSocket) per effettuare una
           connessione con
        // il dispositivo BT dato.
        try {
            tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
        } catch (IOException e) {}
        mmSocket = tmp;
    }

    public void run() {
        setName("ConnectThread");

        // Cancello sempre la ricerca, perch\’e rallenta la
           connessione
        adapter.cancelDiscovery();
        // Effettuo la connessione col socket (BluetoothSocket)
        try {
            // Questa \‘e una chiamata di blocco e restituisce solo
          // una connessione di successo o un’eccezione
            mmSocket.connect();
        } catch (IOException e) {
```

```
                connectionFailed();
                // Chiudo il socket
                try {
                    mmSocket.close();
                } catch (IOException e2) {}
                // Avvio il servizio di riavvio della modalit\`a di
                   ascolto
                BluetoothService.this.start();
                return;
            }

            // Ripristino il thread di connessione, in quanto ho finito
            synchronized (BluetoothService.this) {
                connectThread = null;
            }

            // Avvio il thread col dispositivo connesso
            connected(mmSocket, mmDevice);
        }

        public void cancel() {
            try {
                mmSocket.close();
            } catch (IOException e) {}
        }
    }
```

Infine troviamo la classe ConnectedThread. Questo thread è attivo durante una connessione con un dispositivo remoto e gestisce tutte le trasmissioni in entrata e in uscita.
Il costruttore della classe provvederà alla creazione del socket e dei flussi di dati in entrata e uscita. All'avvio il thread resterà in ascolto dei flussi di input; nel caso in cui la connessione verrà persa, sarà lanciata una notifica. Il metodo write(), quando sarà invocato a seguito della pressione del tasto invio dall'interfaccia, scriverà al flusso di output (OutStream), verso il dispositivo connesso.

```
    /**
```

```
 * Questo thread \'e attivo durante una connessione con un
    dispositivo remoto.
 * Esso gestisce tutte le trasmissioni in entrata e in uscita.
 */
private class ConnectedThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final InputStream mmInStream;
    private final OutputStream mmOutStream;

    public ConnectedThread(BluetoothSocket socket) {
        mmSocket = socket;
        InputStream tmpIn = null;
        OutputStream tmpOut = null;

        // Prendo il socket (BluetoothSocket) con i flussi di input
           e output
        try {
            tmpIn = socket.getInputStream();
            tmpOut = socket.getOutputStream();
        } catch (IOException e) {}

        mmInStream = tmpIn;
        mmOutStream = tmpOut;
    }

    public void run() {
        byte[] buffer = new byte[1024];
        int bytes;

        // Resto in ascolto dei flussi di input (InputStream) finch
           \'e sono connesso
        while (true) {
            try {
                // Leggo dal flusso di input (InputStream)
                bytes = mmInStream.read(buffer);
                // Invio i bytes ottenutito all'UI
                connectionHandler.obtainMessage(ShareNote.
                   MESSAGE_READ, bytes, -1, buffer)
```

```
                        .sendToTarget();
                } catch (IOException e) {
                    connectionLost();
                    break;
                }
            }
        }

        /**
         * Metodo che scrive al flusso di output connesso (OutStream).
         * @param buffer i byte da scrivere
         */
        public void write(byte[] buffer) {
            try {
                mmOutStream.write(buffer);
                // Condivido il messaggio inviato con l'UI
                connectionHandler.obtainMessage(ShareNote.MESSAGE_WRITE,
                    -1, -1, buffer)
                        .sendToTarget();
            } catch (IOException e) {}
        }

        public void cancel() {
            try {
                mmSocket.close();
            } catch (IOException e) {}
        }
    }
```

5.3.5 Le classi di supporto

Per concludere questo capitolo, si descriverà il funzionamento delle classi di supporto dell'applicazione. Come più volte accennato, l'applicazione è munita di un proprio database, utilizzato dalle attività NoteEdit e ShareNote.
L'altra classe di supporto, usata principalmente all'avvio delle connessioni, è DeviceLis-

tActivity. Questa è una semplice attività che fornisce all'utente un elenco di dispositivi raggiungibili, con la quale poi stabilire una connessione.

DeviceListActivity

Questa classe crea una lista dei dispositivi trovati nell'area dopo uno scan. Quando un dispositivo è stato scelto dall'utente, l'indirizzo MAC del dispositivo viene inviato indietro all'attività che lo richiede, come risultato dell'intent.
Precisamente, questa classe durante tutta la fase dello scan, visualizzerà una barra di progresso. Se sono stati trovati dei dispositivi, provvederà alla visualizzazione di essi in una finestra. Se sono stati rilevati dei dispositivi precedentemente accoppiati, essi verranno visualizzati nella parte alta della lista.
Dopo che l'utente avrà scelto il dispositivo al quale connettersi, verrà gestita la fase di accoppiamento, oppure, se ciò è già avvenuto, comunicherà alla classe BluetoothService di avviare la connessione.

NotesDbAdapter

Questa classe fornisce una connessione ad un semplice database di Android per salvare le note create dall'utente. È possibile definire le operazioni CRUD (Create, Read, Update, Delete) e fornire una visibilità delle note in una lista.
La sua funzione principale di questa classe è quella di garantire all'utente la possibilità di creare/modificare le proprie note.

Conclusione

I vantaggi dell'editing di gruppo su dispositivi mobili

La collaborazione di gruppo ha una notevole importanza nel campo aziendale.
L'utilizzo di software collaborativi permette alle aziende di migliorare la propria organizzazione interna, aumentando la velocità delle comunicazioni e riducendo i costi di gestione. Lo sviluppo dei medesimi software su dispositivi mobili è un'occasione per controllare le attività di gruppo, anche quando si è al di fuori del luogo di lavoro.
Nel caso dell'elaborazione dei documenti di testo, l'accesso al gruppo di lavoro tramite dispositivo mobile, sia esso uno smartphone o un tablet, sarà il prossimo passo dell'evoluzione del concetto di collaborazione.

Il prodotto ottenuto

Creare un editor di testo collaborativo per dispositivi mobili non è molto semplice.
I maggiori problemi che si possono riscontrare sono legati alle risorse disponibili sul dispositivo, che possono limitare l'implementazione delle funzioni per la trasformazione del testo, oltre allo sviluppo della connessione client-server. Per questi motivi, il programma MyNotes effettua una mutua esclusione dell'editing su un utente, quando lo stesso ha completato e inviato le modifiche effettuate ad un'altro utente. L'attività di una condivisione della nota è invece eseguibile solo su rete Bluetooth.
I problemi relativi al mantenimento della coerenza dei dati, durante la fase di condivisione, risultano quindi arginati.

Sviluppi futuri

L'editor implementato va considerato come un punto di partenza verso un applicativo con maggiori funzionalità.
In futuro non è da escludere un miglioramento delle connessioni sfruttando Internet e la rete Wifi, in modo da potenziarne le caratteristiche mobili. Oltre alla rete si potrà implementare la gestione dei testi condivisi, in modo da garantire un'utilità pari ad un groupware disponibile su dispositivi fissi.
L'implementazione di nuove funzionalità come ad esempio la gestione delle tabelle, di un database o di presentazioni potranno rendere l'applicativo interessante per l'impiego in campo aziendale o per gruppi di lavoro privato.

Bibliografia

[1] Carli M. . *Android guida per lo sviluppatore*. Apogeo, 2010.

[2] Horstmann C. *Concetti di informatica e fondamenti di Java*. Apogeo, 2010.

[3] IBM Staff. *Rational Unified Process: Best Practices for Software Development Teams*. `http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf` [ultimo accesso: giugno 2010].

[4] Arlow e Neustadt. *UML e Unified Process*. McGraw Hill, 2003.

[5] Pressman. *Ingegneria del software*. McGraw Hill, 2004.

[6] Wikipedia. *Man-hour*. `http://en.wikipedia.org/wiki/Man_hours` [ultimo accesso: giugno 2010].

[7] GNU. *GSL - GNU Scientific Library*. `http://www.gnu.org/software/gsl/` [ultimo accesso: giugno 2010].

[8] The Apache Software Foundation. *Apache Software Foundation - Projects*. `http://projects.apache.org/` [ultimo accesso: giugno 2010].

[9] Open Handset Alliance. *Open Handset Alliance*. `http://www.openhandsetalliance.com/` [ultimo accesso: giugno 2010].

[10] Android Developers. *Android Developers*. `http://developer.android.com/index.html` [ultimo accesso: giugno 2010].

[11] OpenGL ES. *OpenGL ES*. http://www.khronos.org/opengles/ [ultimo accesso: giugno 2010].

[12] The FreeType Project. *The FreeType Project*. http://freetype.sourceforge.net [ultimo accesso: giugno 2010].

[13] SQLite. *SQLite Home Page*. http://www.sqlite.org [ultimo accesso: giugno 2010].

[14] The WebKit Open Source Project. *The WebKit Open Source Project*. http://webkit.org [ultimo accesso: giugno 2010].

[15] Stan Malevanny. *Case Study: Software Project Cost Estimates Using COCOMO II Model*. 2005.

[16] Center for Systems and Software Engineering. *CSSE Website*. http://csse.usc.edu/csse/research/COCOMOII/cocomo_main.html [ultimo accesso: giugno 2010].

[17] Wikipedia. *Computer Supported Cooperative Work*. http://it.wikipedia.org/wiki/Computer_Supported_Cooperative_Work [ultimo accesso: giugno 2010].

[18] IBM Staff. *IBM Lotus Software Italia*. http://www-01.ibm.com/software/it/lotus/ [ultimo accesso: giugno 2010].

[19] Google. *GoogleWave*. http://wave.google.com/about.html [ultimo accesso: giugno 2010].

[20] Gdocs for Android. *Gdocs for Android*. http://sites.google.com/site/gdocsforandroid/ [ultimo accesso: giugno 2010].

Ringraziamenti

I miei maggiori ringraziamenti vanno al professor Messina, per avermi accolto come tesista e per avermi guidato con pazienza e con i suoi consigli durante lo svolgimento della tesi. Desidero ringraziare la mia famiglia, per avermi dato la possibilità di intraprendere la carriera universitaria. Desidero ringraziare i miei amici per essermi stati vicini in ogni momento. Ed infine desidero ringraziare la mia ragazza, la persona più importante nella mia vita.