Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

Un sistema per il Mapping di Access Point Wi-Fi in ambito Android

Tesi di Laurea in Reti di Calcolatori

Relatore:
Chiar.mo Prof.
Vittorio Ghini

Presentata da:
Simone Violi

Seconda Sessione
2013/2014

A papà Claudio e mamma Paola
Che mi hanno sempre sostenuto e spronato
A Elisabetta
Che ha riempito la mia vita
A Patrizia
Che mi fa sentire come un figlio...

Indice

Elenco delle figure

Introduzione

In questa opera di tesi mi sono occupato della raccolta e memorizzazione di informazioni sulle comunicazioni Wi-Fi, su uno scenario in cui diversi utenti possono spostarsi in territorio aperto o area urbana, utilizzando dispositivi mobili come smartphone o tablet dotati di più interfacce di rete, tra le quali una Wi-Fi, per rilevare le reti wireless attive. Tramite un client intendo fornire ad un server, che gestisce un database chiamato "Oracolo", informazioni riguardanti la locazione geografica approssimativa e l'area di copertura di diversi Access Point, in modo da costruire e mettere a dsposizione uno strumento utile e centralizzato, che a sua volta distribuisca queste informazioni ad altri client disclocati nel territorio.

Il nome "Oracolo" nasce dalla possibilità che hanno i client di poter consultare, in qualsiasi momento e posizione geografica, un database che fornisca informazioni utili sui dispositivi Wi-Fi circostanti, che ricorda figurativamente l' oracolo presente nelle credenze delle antiche civiltà elleniche, fonte di consigli e profezie infallibili. Questo seppur meno "infallibile" , permette ai client di ottenere dati usufruibili sugli Access Point vicini, che precedentemente sono stati rilevati da altri client e forniti al server stesso.

É stata perciò progettata, implementata e valutata, un'applicazione che opera su dispositivi Android, che a comando dell'utente, acquisisce informazioni sugli

Access Point nelle vicinanze ad intervalli regolari o definiti dall'utente. Questi dati vengono descritti e salvati in un file XML locale al dispositivo, che verrà poi inviato al server per essere elaborato. L'Oracolo quindi gestirà queste informazioni in un database e le fornirà per successive consultazioni.

Questo lavoro di tesi è la continuazione del lavoro iniziato da Andrea Somenzi e Alberto Paladino; in particolare il mio lavoro si è concentrato sulla realizzazione delle comunicazioni fra client e server, la realizzazione della fase di consultazione dell' oracolo sul client e l' implementazione del server stesso, che memorizza o scarta i dati ricevuti in un database e li fornisce su richiesta, tramite l'utilizzo di un certo algoritmo di ricerca geografica.

Il documento di tesi è diviso in diversi capitoli:

- In "Scenario" vengono illustrate le possibili situazioni prese in analisi e le scelte implementative per l' applicazione.
- In "Obiettivo" vengono definiti i goal prefissati dal progetto di Mapping Access Point e più precisamente da questo lavoro di tesi.
- In "Funzionalità" vengono elencati gli attributi che caratterizzano questa applicazione.
- In "Strumenti" vengono citati e descritti linguaggi, componenti software e hardware utilizzati per la realizzazione di questo progetto.
- In "Progettazione" viene trattata l'analisi teorica che è stata alla base delle scelte implementative.

- In "Implementazione" vengono descritte le scelte implementative effettuate da ambo i lati client e server.
- In "Criteri di valutazione" vi è un' analisi sui risultati ottenuti dai test.
- In "Sviluppi futuri" vengono accennati possibili sviluppi implementativi lato client e lato server, con una possibile nuova analisi delle priorità.
- In "Conclusioni" viene riassunto l'operato svolto e tracciata una valutazione dei traguardi raggiunti e raggiungibili in futuro.

Capitolo 1

Scenario

In questo capitolo si prendono in analisi diversi scenari in cui l'utilizzo di questa applicazione potrebbe essere di aiuto, ed hanno tutti un elemento in comune: la mobilità dell'utente. Tali esempi sono stati menzionati nel documento di tesi di Somenzi e Paladino ed in quanto esplicativi, sono stati qui riportati per fare maggiore chiarezza.

Primo scenario: utilizzo continuativo della rete dati

Un primo esempio classico può essere rappresentato da un turista che passeggia per le strade di una città. Questo molto probabilmente avrà bisogno di informazioni sui migliori luoghi dove poter mangiare, le attrazioni turistiche, indicazioni stradali, ottenere quindi informazioni sui cosiddetti punti di interesse o semplicemente voler accedere alla rete in qualsiasi momento. Oggigiorno chiunque sia in possesso di uno smartphone utilizza un'applicazione per ogni esigenza e la maggior parte di queste richiede una connessione attiva ad internet. Uno smartphone che non riesce ad accedere alla rete perde di fatto buona parte delle sue caratteristiche

di mobilità.

Purtroppo però i contratti UMTS abituali offrono traffico limitato e velocità solitamente inferiori a quelle di una rete ADSL casalinga media, quindi l'utente sarebbe decisamente avvantaggiato nel connettersi ad un access point Wi-Fi vicino, piuttosto che usare la connessione dati del proprio cellulare (sia essa una connessione GPRS, UMTS o LTE) sia per i motivi sopra elencati, sia anche per motivi economici (navigare con la connessione dati all'estero è solitamente molto costoso) e perché il consumo di batteria risulterebbe notevolmente superiore rispetto alla navigazione sotto rete Wi-Fi.

Per non parlare poi della difficoltà del passare continuamente da wireless a UMTS e viceversa durante uno stream o una chiamata voip, interrompendo il collegamento e rischiando di non riuscire più a ripristinarlo. In casi estremi si potrebbe impiegare addirittura più tempo a tentare di passare da una rete ad un'altra che non a navigare. Se si aggiunge il fatto che molti tablet non hanno la possibilità di connettersi via UMTS, l'idea di poter essere collegati ovunque rimane ancora oggi un'utopia.

Secondo scenario: catalogazione reti disponibili

Una situazione totalmente differente potrebbe invece riguardare la decisione di un Comune o di un altro ente di mappare tutte le reti wireless presenti nella propria zona per vari motivi. Google stessa per esempio, sfrutta le proprie automobili che utilizza per Google Street View anche per tracciare tutte le reti wireless rilevate, così da potervi associare una posizione GPS e popolare un database, di dimensioni tra l'altro decisamente consistenti, contenenti tra tutti i dati anche le

coppie <Rete Wi-Fi, Posizione>.

Questi dati, ottenuti in scala globale, permettono a Google di riuscire a localizzare con un'elevata precisione anche gli utenti che sono connessi ad internet da dispositivi che non dispongono di antenna GPS.

L'esigenza primaria è quella di avere un'applicazione che possa svolgere in modo efficace entrambi i compiti esplicitati nel primo e nel secondo esempio, che si traduce quindi nello scopo di questo progetto. Si parlerà delle funzionalità dell'oracolo in modo più dettagliato nei prossimi capitoli. Ora si andranno ad illustrare gli scenari da un punto di vista più sistemico, identificandone i due aspetti client e server ed i corrispettivi ruoli che assumono in questo lavoro di tesi.

1.1 Lato Client

Il client sviluppato è compatibile con tutti gli smartphone Android dalla versione 2.2 alla 4.2 ed è pensato per esser utilizzato anche da utenti non al corrente del funzionamento interno dell'applicazione, tramite l'utilizzo di una interfaccia molto semplice ed immediata ma al tempo stesso esaustiva, utilizzabile anche da una persona poco pratica del funzionamento interno.

L'applicazione a disposizione dell' utente si dovrà occupare di fornire le informazioni relative alle reti Wi-Fi vicine tenendo conto dei due scenari sopra esposti; ovvero se ha la necessità di scansionare le reti disponibili e memorizzarle (prima in locale e successivamente sul server) per poi un riutilizzo futuro, o se vuole semplicemente connettersi alla prima rete Wi-Fi più vicina già conosciuta. Se si trova

nel primo scenario il client si preoccuperà di scansionare tutte le reti wireless intorno al device, registrando tutti i dati tecnici degli access point captati. Queste informazioni verranno poi inviate ad un server, il cosiddetto 'Oracolo', il quale elaborando i dati li memorizzerà in un database. Nel secondo scenario invece il client chiederà al server in base alla propria posizione geografica un elenco di reti disponibili nelle vicinanze o in assenza di tale una serie di informazioni che possano permettere l'utente di potersi muovere in direzioni coperte da segnali Wi-Fi. Alcuni di questi strumenti sono trattati solo in maniera teorica e lasciati a future implementazioni.

1.2 Lato Server

Quello che si vuole realizzare è un server centralizzato che gestisca dati in input e output da e verso dispositivi mobili, al fine di mappare le reti wireless in una determinata area geografica. Il server dovrà quindi gestire i dati in maniera differente a seconda delle necessità del client, ovvero se questo sia nella sua fase di acquisizione tramite scansione o acquisizione tramite ricerca su oracolo. Se si rientra nella prima casistica, l'input fornito dal client è costituito da elenchi di reti scansionate in un certo punto geografico, organizzati in documenti XML. Il server in questo caso non fornisce un vero e proprio output verso il client se non una conferma di avvenuta ricezione, ma si occuperà di gestire i dati memorizzandoli nel database o scartandoli se sono già presenti. Se si è in fase di acquisizione tramite consultazione dell' oracolo, il client fornisce al server la sua posizione in termini di coordinate geografiche in un documento XML. Il server restituirà un elenco di reti vicine al dato punto come risultato di una ricerca su database tramite l'algoritmo implementato.

Schema riassuntivo del progetto:

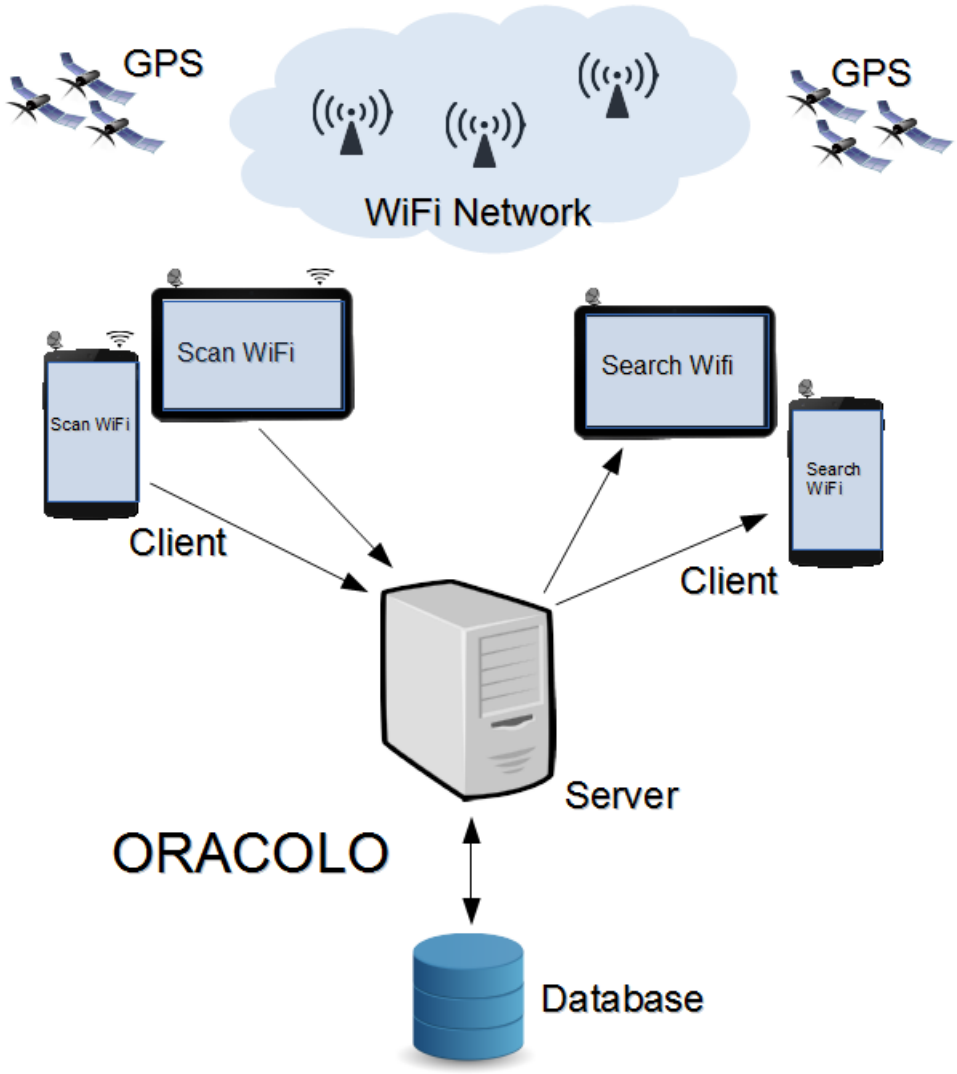

Figura 1.1: Schema dell'oracolo

Capitolo 2

Obiettivo

Ora verranno descritti gli obiettivi identificati, prendendo in analisi il punto di vista dell' utente che andrà ad utilizzare il client e quello del server che dovrà cercare di rispondere alle necessità del client.

2.1 Lato Client

Come già accennato, l'applicazione lato client ha sostanzialmente due compiti, raccogliere dati e inviarli all'oracolo e ricevere dati precedentemente salvati da questo. Il primo compito consiste nel creare un database locale con tutti i dati scansionati dai vicini access-point e associarvi la posizione GPS del dispositivo. Successivamente questi dati verranno inviati ad un server predisposto, l' Oracolo, che riceverà questi dati e costruirà un database contenente le posizioni approssimate degli Access Point, in base all'algoritmo usato e la loro area di copertura, anche questa approssimata. Il secondo compito consiste nel richiedere all' Oracolo un elenco di Access Point vicini alla propria posizione già precedentemente rilevati, inviando come parametro di ricerca la propria locazione geografica. Inoltre necessario specificare che una ricezione migliore delle onde Wi-Fi emesse dal Access Point non è obbligatoriamente sinonimo di connessione ad internet migliore.

Purtroppo questa non è una condizione verificabile a priori senza doversi connettere prima a tutti gli access point trovati e senza fare una prova di navigazione, o test di velocità di connessione per controllare le performance della linea.

2.2 Lato Server

Il server quindi avrà il compito di catalogare le scansioni effettuate dai client in un database centralizzato e fornire a questi, in un secondo momento, un elenco il più accurato possibile di Access Point in base alla loro posizione. Il livello di accuratezza dipenderà dagli algoritmi utilizzati per inferire sulla loro posizione in base alla locazione del client o alla potenza del segnale nel momento in cui lo ha rilevato. Tutte le operazioni di computazione verranno quindi delegate al server evitando questi oneri al client, garantendo quindi un consumo ridotto di batteria. Inoltre la centralizzazione dei calcoli garantisce compatibilità fra i dispositivi mobili, che riceveranno o invieranno solo dati testuali XML.

Capitolo 3

Funzionalità

Al momento della progettazione dell' applicazione si è scelto lo sviluppo per soli dispositivi con sistema operativo Android, in quanto gli altri sistemi operativi mobili non erano interessanti ai fini del progetto, quindi si è deciso di mantenere tale idea anche in questo progetto di tesi. Le motivazioni di tali scelte furono:

- iOS di Apple non sarebbe stato idoneo come S.O. poichè le API per sfruttare la connettività Wi-Fi erano private e quindi non ufficialmente disponibili.
- Windows Phone di Microsoft non è stato preso in considerazione a causa della poca diffusione del sistema.

Inoltre entrambi questi sistemi operativi sono abbondantemente closed source a differenza di Android che è Open Source. Android inoltre è il più diffuso ed è presente anche su dispositivi relativamente economici.

Il server Oracolo è interamente scritto in PHP e il database che raccoglie i dati è un documento XML. La scelta del linguaggio PHP è stata effettuata in quanto sono già presenti numerose librerie native per la manipolazione di stringhe XML e la formulazione di query XPATH oppure sono facilmente reperibili su internet. Inoltre vi è un' ottima interoperabilità con diverse tecnologie come Java e .NET nel

caso si vogliano implementare in futuro funzionalità che si appoggiano a queste. Infine va menzionato che l'interprete php è open source.

L' applicazione a disposizione dell' utente dispone delle seguenti funzionalità, alcune di queste precedentemente implementate dal lavoro di Somenzi e Paladino e da me modificate o riadattate per poter aggiungere nuove caratteristiche:

- Interfaccia con layout auto-orientabile:
  Per questioni di praticità, è stata implementata l'interfaccia in modo da poter essere utilizzata sia tenendo il telefono in modalità Portrait (verticale), sia in modalità Landscape (orizzontale), decisamente più comoda in caso di utilizzo dell'oracolo su Tablet anzichè Smartphone. Ora a tale funzionalità è è stata aggiunta la possibilità di poter scorrere lo schermo tramite uno scroll.

- Menù di scelta modalità:
  Ora all'avvio dell'applicazione l'utente potrà scegliere la modalità che vorrà effettuare, tra "Chiedi All' Oracolo" o "Cerca Wi-Fi" tramite le semplice pressione del pulsante relativo. Prima era presente la sola modalità di scansione che si presentava immediatamente sullo schermo all' avvio.

- Modalità "Cerca Wi-Fi" con scansione a comando:
  L'interfaccia è stata mantenuta il più semplice possibile per far risaltare i dati dal contesto presentativo, fintanto che l'app rimane in fase di sviluppo. La schermata dell'app è composta da due $TextView$, una per mostrare su schermo i risultati dell'ultima scansione effettuata e una per eventuali errori o comunicazioni (ora anche messaggi ricevuti dal server). Sotto a queste è presente un campo dove poter inserir l'intervallo di scansione ed un tasto per avviare e interrompere la scansione in base alle proprie esigenze. Infine è presente una casella da spuntare se si vogliono inviare i dati al server al termine della scansione.

- Scansione ad intervalli regolari:

  In base al valore numerico contenuto dentro una *InsertBox*, è possibile impostare l'intervallo con cui verranno effettuate le successive scansioni. Un intervallo ragionevole è tra i 1000 e i 10000 millisecondi. Ora si è scelto per motivi di praticità di permettere all'utente di impostare il valore direttamente in secondi. Il valore di default è impostato a 3 secondi.

- Scansione in background come servizio:

  Per l'utente deve essere possibile fare anche altre cose con lo smartphone mentre la scansione è in atto, siano esse cose semplici come mandare sms o telefonare, ma anche navigare in internet e quindi sfruttare il Wi-Fi stesso per scansionare e contemporaneamente sfruttare la connessione attiva.

- Salvataggio scansioni su file XML:

  L'applicazione salva tutte le informazioni finora ottenute in un file, concatenando tutti i dati ricevuti ad ogni scansione. Questo file viene salvato in locale sulla scheda di memoria del dispositivo mobile, per un futuro o immediato upload al server a seconda della scelta dell' utente.

- Upload Dati al Server:

  Prima di effettuare una scansione l'utente potrà scegliere se effettuare immediatamente l' upload al server appena questa è terminata, mettendo una spunta su una *CheckBox* presente su schermo sopra il pulsante di avvio scansione. In caso contrario i dati rimarranno salvati su un file in locale. Tale operazione è effettuata in maniera asincrona, evitando quindi il blocco dello schermo.

- Modalità "Chiedi All' Oracolo" con richiesta a comando.

  Ora è possibile ricevere dati precedentemente inviati all' Oracolo alla pres-

sione di un pulsante. E' necessario che il dispositivo GPS sia attivo per poter effettuare una ricerca sul server, pertanto nessuna ricerca partirà fintanto che il dispositivo non abbia completato la sua sincronizzazione con il GPS. Un messaggio verrà visualizzato a video per notificare l' utente. Per motivi di prestazioni e per una maggiore semplicità di debug attualmente si è scelto di non effettuare richieste continuative in background ma solo "On Demand". Le motivazioni saranno discusse successivamente.

A queste funzionalità andrebbero aggiunte quelle fornite dal server che verranno invece esposte nei capitoli successivi nel dettaglio.

Capitolo 4

Strumenti

Per lo sviluppo di questo progetto sono state utilizzate diverse tecnologie a seconda dell' ambito di lavoro. Tali tecnologie sono separate in Software ed Hardware (smartphone e tablet) e sono qui sotto elencate.

4.1 Software

L' ambiente di sviluppo IDE (Integrated Development Environment) utilizzato è stato principalmente Eclipse Juno (versione v4.2.2) con la suite di plugins ADT (Android Developer Tools) messa a disposizione da Google stessa (versione v22.2.1), per lo sviluppo del client Android.

La suite ADT velocizza la creazione di progetti Android, la creazione e preparazione di macchine virtuali per poter testare sul proprio computer le proprie applicazioni. Nonostante quest'ultimo aspetto sia assolutamente interessante e potente è sfruttato quasi esclusivamente nelle prime fasi di sviluppo, in quanto non è possibile utilizzare le funzioni relative al Wi-Fi nel simulatore. E' possibile inoltre collegare via USB al computer uno o più dispositivi Android attivando la modalità debug fornita dal S.O. ed effettuare il deployment direttamente sullo

smartphone in un ambiente reale.

Per lo sviluppo del lato server è stato utilizzato l'editor di testo Open Source Notepad++ per Windows che è leggero e veloce ed ha la possibilità di essere esteso con numerosi plugins a seconda del linguaggio utilizzato.

Il server Apache utilizzato in fase di test e sviluppo è stato configurato con l'ausilio di XAMPP v3.2.1 per Windows.

4.2 Linguaggi

4.2.1 XML

XML (eXtensible Markup Language) è un linguaggio di markup ideato per scambiare, immagazzinare e dare una struttura ai dati. Questo, nato negli anni '90 e diventato Recommendation del W3C (World Wide Web Consortium) nel febbraio del 1998, è un metalinguaggio che permette la definizione di altri linguaggi tramite l'aggiunta o estensione (da cui il termine "extensible") di nuovi tag alla grammatica che lo definisce . A differenza di altri linguaggi di markup come HTML, il suo scopo non è definire una grammatica per la visualizzazione (descrizione e formattazione) di dati in ambito web, ma di poter creare documenti strutturati che possano essere usati anche in ambiti differenti dal web. Col tempo infatti XML è diventato quasi uno standard per la comunicazione fra sistemi e applicazioni diverse. Queste sue caratteristiche sono state determinanti nella la scelta di usarlo come strumento unico di scambio informazioni tra client e server e come database nell' Oracolo stesso.

4.2.2 Android

Android è il sistema operativo per dispositivi mobili (e meno mobili, come le smartTV) più usato al mondo, con più di un miliardo di smartphone e tablet in tutto il mondo che lo utilizzano. Sviluppato inizialmente da Android Inc. successivamente acquisito da Google nell' agosto del 2005, nasce sulla base di un kernel Linux, ed è oggi un sistema operativo aperto, libero e gratuito con licenza Apache. Viene utilizzato in numerosi ambiti diversi oltre ai già citati tablet e smartphone; infatti lo si può trovare nei portatili e netbook, smartbook, eBook reader, fotocamere e smart TV e recentemente anche in lettori CD\}DVD.

Le applicazioni in Android sono implementate con l' ausilio di framework scritti in JAVA nella parte dinamica dell' applicazione e scritti in XML nella parte statica. Queste parti vengono combinate in fase di compilazione in un codice intermedio, simile al bytecode di Java, generando un file con estensione .apk che verrà eseguito successivamente su una macchina virtuale chiamata Dalvik Virtual Machine (DVM) che sostituisce la JVM di Java.

Oltre alle peculiarità del S.O., un fattore incisivo per la scelta di tale ambiente di sviluppo sono le risorse rese disponili agli sviluppatori da parte del team di Android. Infatti è consultabile tutta la documentazione ufficiale e sono disponibili online numerose guide per lo sviluppo.

4.2.3 PHP

Il linguaggio utilizzato sul server è PHP (PHP Hypertext Preprocessor). Nato nel 94 come raccolta di script CGI per la creazione di pagine web dinamiche che negli anni venne esteso e riscritto, dapprima in C, per poi diventare un linguaggio con un buon supporto al paradigma orientato agli oggetti nella sua versione 5. La sua sintassi è simile a quella del C e ne deriva in parte i suoi costrutti pur rimanendo un linguaggio ad alto livello. Tale linguaggio viene interpretato con l' ausilio di un file di configurazione, il *php.ini*, che fornisce le impostazioni e configurazioni dei vari moduli con cui l'interprete è stato compilato. La quantità elevatissima di API e documentazione disponibile nel web sono strumenti che hanno contribuito la sua diffusione nel tempo come linguaggio server side.

4.2.4 Sistemi operativi utilizzati per lo sviluppo

- Windows 7 Ultimate 64bit.

4.3 Smartphone e Tablet utilizzati

- Smartphone Huawei Ascend g510 Android 4.1 Schermo 4.5"
- Smartphone Huawei Ascend y330 Android 4.2.2 Schermo 4"
- Smartphone Samsung Galaxy Next Android 2.2.1 Schermo 3.14"
- Tablet Lenovo s6000 Android 4.2 Schermo 10.1"

Capitolo 5

Progettazione

5.1 Lato Applicazione

Per poter descrivere meglio gli aspetti dell' applicazione è necessario menzionare le funzionalità native del linguaggio Android; ciò aiuterà la comprensione del lavoro iniziato dai miei predecessori e da me ampliato. In particolare qui sotto verranno velocemente descritti i componenti Activity e Service, i loro metodi di comunicazione e come questi risultano essere attori principali dell' applicazione.

5.1.1 Activity

Una activity come suggerisce il nome è una funzionalità o un' attività che un utente può svolgere. Questa attività viene presentata sotto forma di singola schermata dedicata, con la quale l'utente può interagire. Le sua funzionalità dinamiche sono espresse in una classe JAVA mentre i suoi aspetti presentativi sono descritti in un documento XML.
Tali activity possiedono cicli di vita differenti tra loro e godono di forme di gerarchie definite. L' idea principale quindi è di avere due attività distinte che svolgano

il compito di inviare dati all'oracolo e di riceverli su richiesta e di una terza che funga da menù introduttivo di scelta per l' utente.

5.1.2 Service

Un service è un componente che viene avviata da un' altra componente all'interno dell' applicazione (una activity) che si occupa di eseguire operazioni a lungo termine in background, senza fornire un'interfaccia utente. Un servizio generalmente assume due forme che sono frutto del legame logico tra la componente chiamante e il servizio stesso. Quando il ciclo di vita di un servizio avviato è indefinito ed indipendente dal ciclo di vita della componente chiamante, si ha un servizio "started". Si ha invece un servizio "bounded", se al momento dell' avvio viene creato un legame (bound) tra la componente chiamante e il servizio per mezzo di un' interfaccia, in modo che questi possano comunicare in qualsiasi momento tramite comunicazione interprocessuale.
La necessità di avere un servizio che effettui una scansioni in background ripetute si sposa bene con la prima forma. Entrambe le fasi (richiesta all'oracolo e ricerca) infatti possono essere strutturate in tale maniera con l'activity che assume il ruolo di chiamante e delegante del servizio che svolge effettivamente le azioni di richiesta e scansione. Come accennato prima , per ottenere una maggiore chiarezza in fase di debugging, la fase di richiesta all'oracolo attualmente è implementata come attività "on Demand" , in quanto fonte di maggiori criticità in termini temporali.

5.1.3 Intent e Intent Filters

Un intent è un oggetto con lo scopo di inviare messaggi contenenti le azioni che si stanno per svolgere (intenti) fra componenti diverse nell' applicazione. Questi possono essere utilizzati per avviare un' activity od un servizio o inviare un

broadcast. Possono essere suddivisi per categorie, per azioni o per il tipo di dati che trattano attraverso un meccanismo di filtraggio adottato dal systema stesso, che seglie implicitamente l'intent più idoneo se questo non è stato specificato manualmente.

5.1.4 Broadcast Intents and Broadcast Receivers

Come già menzionato gli intent oltre ad essere utilizzati per avviare servizi ed attività, vengono impiegati come strumento di comunicazione fra altre componenti all'interno dell'applicazione, ovvero per inviare broadcast. La componente che vuole inviare un messaggio creerà un Broadcast Intent contenente la tipologia del messaggio (la natura dei dati) e il messaggio stesso (i dati veri e propri). La componente che deve ricevere tale messaggio deve disporre di un Broadcast Receiver in ascolto per poter catturare il messaggio ed interpretarlo.

Risulta evidente quanto questo strumento risulti fondamentale nella comunicazione fra activities e fra activity e service, perciò si farà ricorso a questo più volte nella realizzazione di entrambe le fasi.

5.2 Lato Server

Per poter rispondere alle diverse necessità del client, il server dovrà essere organizzato in moduli diversi che fungano da raccoglitori, manipolatori e fornitori di dati. La natura del linguaggio PHP permette tramite il suo paradigma una strutturazione in moduli abbastanza indipendenti, i quali potranno essere chiamati dal client mediante HTTP Request in base alle proprie necessità. Inoltre si potranno integrare in futuro ulteriori moduli per aggiungere funzionalità senza dover metter mano a quelli già esistenti.

5.2.1 Raccogliere i dati

La catalogazione delle reti dovrà avvenire tramite una comunicazione unidirezionale tra client e server. Il client invierà dati tramite HTTP POST al server che intercetterà questi dati e si assumerà l' incarico di memorizzarli in un documento XML dopo averli strutturati in un certo modo, trasparente al client stesso. Al termine potrà comunicare al client l' avvenuta ricezione di tali dati.

5.2.2 Organizzare i dati

Se il compito di popolare il database spetta ai diversi client sparsi nel territorio, il come popolarlo spetta al server. Questo che come già specificato è costituito da un documento XML, dovrà contenere non solo le specifiche di ogni access point ma anche le informazioni della posizione geografica del client al momento della rilevazione. Risulta evidente quanto la presenza di più voci relative allo stesso access point con coordinate differenti non siano dati superflui o ridondanti ma forniscano un ulteriore elemento di riconoscimento dell' effettiva area di copertura del segnale rilevato.

5.2.3 Distribuire i dati

Così come riceve i dati, il server nelle sue vesti da Oracolo dovrà restituirli una volta consultato. In base ad uno specifico parametro di ricerca fornito dal client, come le coordinate geografiche, il server dovrà restituire tutti gli access point esistenti entro una certa zona attorno al client. Questa zona o "raggio" dovrà essere di dimensioni ragionevoli, per permettere al client di poter effettivamente usufruire di tali informazioni, pur sapendo che la potenza del segnale registrata al momento della scansione potrebbe non rispecchiare una effettiva vicinanza o lontananza dal dispositivo. Sarebbe utile inoltre che in mancanza di dati suffi-

cienti per dare una risposta al client, il server inferisca sulle posizioni degli access point più prossime alla locazione del client, in modo da poter fornire una sorta di percorso indicativo, che possa aiutarlo nella scelta di quale direzione prendere per entrare in una zona coperta da segnale Wi-Fi. Per poter fare questo genere di inferenze però è necessario conoscere la destinazione dell' utente, nel caso che questo sia in movimento.

5.3 Scelte progettuali

Vengono ora specificate le scelte progettuali adottate nelle sue due componenti client e server, che ricalcano in parte le scelte prese inizialmente da Somenzi e Paladino. Come già menzionato nel loro lavoro i tesi, la qualità del servizio di scansione delle reti Wi-Fi dipende dalle funzionalità fornite dal sistema operativo Android , in quanto il client sfrutta a livello applicativo tali funzionalità e non le va a ridefinire.

5.3.1 Lato Applicazione

Precedentemente si sono menzionate le caratteristiche fondamentali del linguaggio android. In base a tali conoscenze si è proceduto alla scrittura del codice, analizzando le caratteristiche che dovrà avere l'applicazione per essere funzionale, come la necessità di avere il segnale GPS attivo o la possibilità di cambiare l' intervallo con cui effettuare la scansione o altre componenti sotto menzionate.

- Intervallo scansione

  Viene lasciata all'utente la scelta dell'intervallo fra una scansione e l'altra, con un valore minimo di 1000 ms e un valore massimo a 10000 ms. Si è deciso di impostare come valore di default 3000 ms in seguito a numerosi

test eseguiti a piedi ed in autovettura. Ora tale valore può essere specificato dall' utente direttamente in secondi e sarà l' applicazione ad effettuare la conversione.

- Coordinate GPS

  Che il client sia in fase di scansione o di richiesta all oracolo, vi è comunque la necessità che il dispositivo mobile abbia il GPS attivo e sincronizzato. Poiché tale situazione potrebbe non essere verificata in fase di scansione, è stato applicato l'approccio precedentemente ipotizzato dai miei predecessori, ovvero che è meglio possedere dati parziali che non avere nessun dato. L' applicazione infatti se è nella sua fase di scansione continuerà a funzionare e ad inviare dati al server anche se sprovvisti di coordinate. Tali dati verranno poi scartati dal server in quanto attualmente non utili per la costruzione di una mappa di access point, ma utili in fase di testing per capire la mole di dati media che passa nella comunicazione client-server. Questi potrebbero comunque essere sfruttati inferendo sulla posizione in base al tempo trascorso dalla ultima rilevazione effettuata e la successiva, ma queste valutazioni sono attualmente solo ipotetiche e non prese in seria analisi. Per quanto concerne la fase di richiesta all' Oracolo, fino a quando non si è stabilita una sincronizzazione col segnale GPS nessun dato può essere ricevuto, in quanto nessuna richiesta può essere effettuata e verrà quindi segnalato all'utente tramite un messaggio di avviso.

- Wi-Fi

  Ogni volta che viene effettuata una scansione, vengono recuperati tutti i dati possibili relativi agli access-point rilevati in quel preciso istante, ovvero indirizzi MAC, SSID, capabilities d' accesso e potenze del segnale in forma numerica e testuale, poi tali informazioni vengono ampliate con l'aggiunta delle coordinate GPS del dispositivo. Prima le coordinate erano legate ad un blocco di reti Wi-Fi scansionate nello stesso istante, ora invece ogni singolo access-point è completo di informazioni sulle coordinate ed inviato al server direttamente come tale, poichè nel server verranno comunque catalogati e memorizzati in questa maniera.

- Salvataggio Dati in locale

  Poichè potrebbe non essere possibile effettuare una connessione al server in fase di scansione, i dati rilevati verranno salvati su file XML in locale, che al termine della scansione successiva potranno essere inviati al server se l'utente lo desidera. I file una volta avvenuto l' upload verranno cancellati dalla memoria SD locale per liberare spazio. Tali file sono salvati in una cartella locale chiamata ".Oracle" e denominatinominati con "scan_" più un codice progressivo costituito dalla data in questo formato "ddMMyyyyhhmmss" (giorno(2),mese(2),anno(4),ore(2),min(2),sec(2)). L' ordine dei file attualmente non è influente ai fini del progetto, ma solo un metodo di differenziazione ed organizzazione. La struttura interna dei file in locale è simile a quella precedentemente implementata, con la differenza che tali verranno creati solo al termine della scansione.

La struttura di tali file è la seguente:

```
<?xml version="1.0" encoding="UTF-8"?>
<oracle>
	<session>
		<timestamp>Data</timestamp>
		<scan>
			<id></id>
			<gpsData>
				<lat></lat>
				<lng></lng>
			</gpsData>
			<mac></mac>
			<ssid></ssid>
			<capabilities></capabilities>
			<frequency></frequency>
			<powerlevel></powerlevel>
			<powerliteral></powerliteral>
		</scan>
		<scan>
			<id></id>
			<gpsData>
				<lat></lat>
				<lng></lng>
			</gpsData>
			<mac></mac>
			<ssid></ssid>
			<capabilities></capabilities>
			<frequency></frequency>
			<powerlevel></powerlevel>
			<powerliteral></powerliteral>
		</scan>
	</session>
</oracle>
```

- Trasferimento dati in remoto

  I file XML precedentemente creati in locale verranno letti e trasferiti al server mediante HTTP POST Content-Type: text/XML e una volta terminata la connessione verranno cancellati dallo storage locale per liberare spazio. In caso di errore o mancata connessione, rimarranno nella memoria e verranno spediti al tentativo successivo.

5.3.2 Gestione Lato Server dell'Oracolo

Il lato server è stato realizzato prendendo in analisi alcuni elementi teorizzati da Somenzi e Paladino, ed effettuando alcune considerazioni in merito all' efficacia o efficienza di tali algoritmi.

Problematiche

Quello che si è cercato di realizzare è un algoritmo semplice che potesse in tempo breve dare una risposta soddisfacente al client, pur non precludendosi miglioramenti futuri per poter dare una migliore risposta in termini di precisione. Per prima cosa è necessario interrogarsi se sia più importante fornire risultati corretti in tempi lunghi (o fuori tempo) o dare risultati imprecisi e talvolta insufficienti ma usufruibili. La risposta chiaramente è una via di mezzo che permetta all' utente di avere sempre una risposta utile in tempi altrettanto utili. Nella realizzazione di un algoritmo simile è necessario tenere conto di fattori diversi, alcuni ponderabili ed altri meno. Tra questi fattori il più importante risulta essere la inconsistenza dei dati rispetto alla realtà attuale, ovvero la differenza che passa tra ciò che è stato scansionato e memorizzato nel passato con ciò che esiste attualmente in fase di "consultazione dell' Oracolo". A concorrere nella formulazione di tale inconsistenza ci sono fattori diversi come la potenza del segnale reale, la posizione

reale dell' access-point, la posizione del dispositivo al termine della chiamata (che può essere diversa da quella in cui è stata iniziata la ricerca sull' Oracolo). Molti degli algoritmi presi in analisi non riescono a dare una risposta significativa a tali problematiche e la formulazione di una risposta utile al client risulta essere assai meno prevedibile.

Prima soluzione: Ricerca circoscritta

Da queste considerazioni quindi si è partito col presupposto che il client deve quantomeno ottenere una risposta, seppur incompleta, che magari possa essere migliorata nel tempo.

Quindi come primo algoritmo si è sfruttata la possibilità di avere memorizzate nel database molteplici locazioni del medesimo access-point. Infatti cercando nel database entro un certo raggio geografico abbastanza circoscritto e possibile ottenere diversi risultati, alcuni relativi ai medesimi dispositivi e restituirli singolarmente in quanto ipoteticamente sufficientemente vicini. Naturalmente questa soluzione dipende strettamente dal numero di rilevamenti (o scansioni) effettuate su tali dispositivi, infatti quante più scansioni sono state effettuate contenenti tali access-point, tanto più è probabile dare risposte significative ai client che interpellano l' Oracolo. La comodità di tale soluzione risiede nel fatto che è possibile dare una risposta anche con l' ausilio di un solo punto, seppur come precedentemente specificato, tale risposta potrebbe essere non soddisfacente.

Seconda soluzione: Sviluppo polinomiale

Uno sviluppo polinomiale in tal senso, come ipotizzato da Somenzi e Paladino, risulta essere una migliore approssimazione all' aumento significativo dei punti. Infatti il calcolo di intersezione tra tali punti fornisce una mappatura più accurata della zona di copertura di un access-point a discapito però di un maggiore

costo computazionale quindi tempo di risposta al client. Tali calcoli dovranno per giunta essere effettuati ad ogni richiesta in quanto i dati nel database potrebbero essere stati ampliati da nuove scansioni, cambiando di fatto la geometria del poligono calcolato precedentemente. Inoltre gli algoritmi che sviluppano un inviluppo convesso necessitano di più punti e risultano essere del tutto inutili in mancanza di tali.

Di fronte a queste analisi sono giunto alla conclusione che lo sviluppo polinomiale possa essere realizzato come strumento migliorativo di una ricerca massimale precedentemente realizzata e per tanto attualmente non vi è nessuna implementazione.

Terza Soluzione: inferenza sui dati

Tale strumento inoltre potrebbe essere utilizzato come metodo di inferenza in mancanza di dati significativi sugli access point. Si potrebbe infatti cercare di costruire una sorta di tragitto interpolando i punti piu prossimi al client già presenti nel database fornendo quindi una indicazione di massima all' utente, permettendogli così di potersi muovere sul territorio in direzione di tali locazioni. Naturalmente nemmeno in questo caso verrebbe garantita la reale presenza di segnale Wi-Fi in quanto potrebbe essere spento o non raggiungibile l' access-point. Tale soluzione potrebbe essere implementata in futuro.

Capitolo 6

Implementazione

In questo capitolo verranno discusse le modifiche effettuate sul client e spiegate le implementazioni effettuate sul server.

6.1 Lato Client

6.1.1 Classi, Servizi ed Activities

L' app a disposizione dell' utente è sviluppata con l'utilizzo di diverse activities qui descritte brevemente:

- *MainActivity*

  Produce il menù iniziale per la scelta delle due modalità da effettuare, "Cerca Wi-Fi" e "Chiedi all' Oracolo". La scelta porta alla chiusura di questa activity e l' apertura di quella scelta.

- *ScanActivity*

  Si occupa della gestione degli aspetti relativi alla scansione delle Wi-Fi. Essa avvia un servizio alla pressione di un pulsante e riceve da tale servizio i dati scansionati, li salva in un file e su richiesta dell' utente li spedisce al server.

- *ScanService*

  Questo è il servizio avviato dalla *ScanActivity*. Si occupa della gestione della geolocalizzazione e della scansione effettiva dei dispositivi Wi-Fi circostanti. Il processo di scansione si ripete ad un intervallo costante specificato dall'utente e si perpetua indefinitamente fino alla pressione del pulsante Ferma presente anche esso nell'activity.

- *SearchActivity*

  Si occupa della consultazione dell' Oracolo. Permette di connettersi al server ed effettuare una richiesta sulla base della propria posizione geografica. Visualizza a schermo l'elenco dei risultati ottenuti.

- *Class wifiList*

  Questa classe tramite i suoi metodi permette di manipolare i dati sugli access point scansionati.

- *Class XMLParser*

  Questa classe permette di decifrare un documento XML e costruire una struttura dati DOM che sia facilmente navigabile, per poter permettere all' utente di visualizzare i dati ottenuti dal server.

- *Class XmlToFile*

  Si occupa della creazione di documenti XML a partire da dati testuali rsultanti dalle scansioni degli access point. Tali dati verranno poi memorizzati in un file locale.

6.1.2 Permessi su AndroidManifest.xml

Il documento manifest è un documento che specifica i permessi di cui l'app necessita sul dispositivo ed altre informazioni relative all' app stessa, tra cui:

- versione dell' applicazione: 0.6
- versione delle sdk supportate: min=9 max=20
- permessi sotto specificati.
- nomi dei servizi e activity ed intent-filter relativi all'azione che andranno a svolgere.

Elenco Permessi

- CHANGE_WIFI_STATE

  Permette all' applicazione di cambiare lo stato di connettività del dispositivo mobile relativo al Wi-Fi.

- ACCESS_WIFI_STATE

  Permette all' applicazione di accedere alle informazioni relative alle reti Wi-Fi scansionate.

- WRITE_EXTERNAL_STORAGE

  Consente all' applicazione di accedere in scrittura ad una memoria esterna al dispositivo.

- ACCESS_FINE_LOCATION

  Consente di ottenere informazioni sulla locazione precisa di sorgenti di segnale come GPS.

- ACCESS_COARSE_LOCATION

  Come sopra permette di accedere a sorgenti di segnale ma con locazione approssimata.

- INTERNET

  Abilità l'applicazione all'utilizzo di socket di rete.

6.1.3 Cenni sul codice

In questa sezione verranno descritte brevemente le funzioni principali che compongono l'applicazione, alcune di esse precedentemente implementate da Somenzi e Paladino e da me parzialmente modificate o adattate.

MainActivity

public class MainActivity extends Activity implements OnClickListener

Questa activity ha il compito di mostrare a schermo un menù a scelta fra le due diverse modalità "Cerca Wi-Fi" (presente nella ScanActivity) e "Chiedi All' Oracolo" (presente nella SearchActivity). Questa implementa l'interfaccia *onClickListener*, che permette con i suoi metodi *onClick*(), di avviare tramite la pressione di un pulsante a scelta, una delle due activity *ScanActivity* e

SearchActivity . Alla pressione del pulsante viene creato un nuovo intent che indica all' applicazione che si vuole avviare una nuova activity, che sucessivamente viene effettivamente avviata.

Intent intentSearch = new Intent(getBaseContext() , SearchActivity.class);
startActivity(intentSearch);

Qualsiasi altro metodo dell' activity viene ereditato dalla superclasse *Activity*.

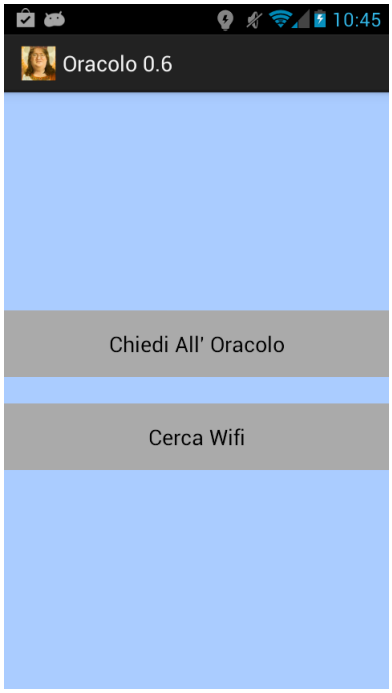

Figura 6.1: MainActivity

ScanActivity

All' avvio di tale activity lo schermo si troverà in una situazione simile alla figura 6.2.
Viene chiesto all' utente di inserire l' intervallo in cui ripetere la scansione per prima cosa, poi una volta chiusa l'interfaccia di inserimento testo, gli si presenterà l' activity con i suoi componenti, che partendo dall' alto verso il basso sono:

- *textBox* che contiene i risultati di scansione.
- *textBox* per i messaggi di errore o comunicazioni varie.
- *insertBox* per permettere all'utente di specificare l' intervallo di scansione.
- *checkBox* per scegliere se inviare i dati o meno al server dopo la scansione.
- *ProgressBar* viene visualizzata solo quando si sta effettuando l'upload dei dati al server.
- *Button* per l'avvio o arresto della scansione.

public class ScanActivity extends Activity implements OnClickListener

Alla creazione dell' activity viene eseguita la funzione *ActivateGPS*() già precedentemente implementata dai miei colleghi, che permette di abilitare la localizzazione GPS sul dispositivo se questa non è già attiva. Così come nell'activity precedente (ed in quella *SearchActivity*) si implementa l'interfaccia *OnClickListener* che con l'uso del metodo *onClick*, alla pressione del pulsante, verrà prima letto il valore dell' intervallo specificato dall' utente se presente, poi avviato il servizio *ScanService*che effettuerà la scansione. Premendo nuovamente il pulsante si arresterà la scansione. I dati scansionati con l'utilizzo di un oggetto *writer* della classe *xmlToFile* verranno prima tradotti in XML, poi

scritti in un file nel formato come prima specificato , con l'utilizzo della funzione $writeToFile$. $writeToFile("scan" + format, xmlResult)$; Se la checkBox è stata attivata dall' utente , tali dati verranno inviati al server uno ad uno. La funzione $sendToServer()$ si occuperà di leggere la cartella contenente i file, inviarne il contenuto con una HTTP POST Request, effettuato tramite l'ausilio di un *AsyncTask* che in background si connetterà al server tramite $connect()$. Al termine dell' upload i file se correttamente inviati verranno cancellati. La comuncazione tra l'activity e il servizio di scansione avviene con l'ausilio di un *BroadcastReceiver* che cattura i risultati inviati dal servizio e li visualizza sulla prima TextBox verde.

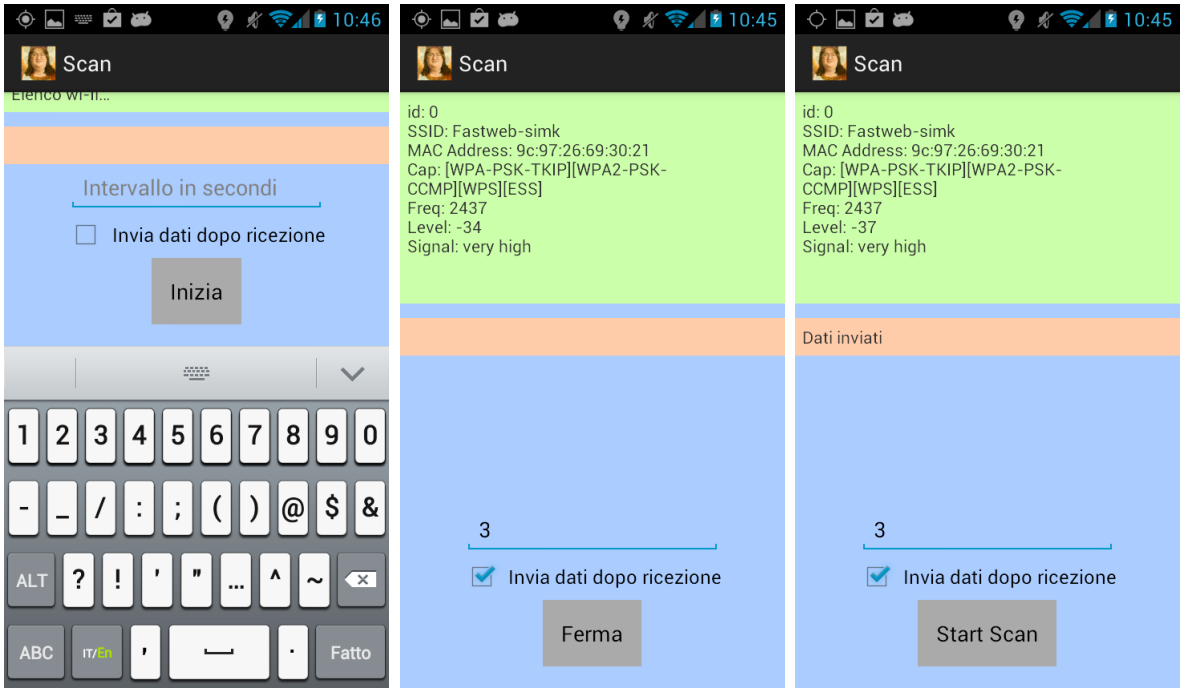

Figura 6.2: ScanActivity

ScanService

public class ScanService extends Service implements LocationListener

Il servizio implementa *LocationListener*, ciò permette di ereditarne i suoi metodi, tra cui *onLocationChanged* che permette di svolgere azioni al variare dello stato dell' oggetto *LocationManager*. All' avvio del servizio vengono identificate le coordinate GPS con l'utilizzo della funzione *localize*(). Successivamente viene creato un task che permette la scansione in un thread differente ad un intervallo specificato nell' activity. Periodicamente con l'utilizzo della funzione $scanWifi()$ viene effettuata la scansione delle reti disponibili (dopo aver controllato che il servizio di sitema Wi-Fi sia abilitato). I dati ricevuti vengono passati a $wifiToString()$ che li organizza in una struttura e crea una stringa di testo da dare come risposta all'activity con l'aggiunta delle coordinate del dispositivo. Il codice di questo servizio è rimasto parzialmente invariato rispetto alla stesura iniziale dei miei colleghi, a parte la creazione della struttura dati contenenti i singoli access point, i quali ora sono arricchiti della geoposizione del dispositivo mobile al momento della scansione singolarmente e non più a gruppo.

SearchActivity

public class SearchActivity extends Activity implements OnClickListener

L'interfaccia si presenta simile a quella presente nella *ScanActivity* (fig 5.2) ma semplificata in quanto deve occuparsi della sola ricezione e visualizzazione dal server nelle sue vesti da oracolo. Si è scelto di non utilizzare una seconda *textBox* per i messaggi di errore o comunicazioni ma di mostrali in un unica *textBox*, in quanto questi sono strettamente legati all' ottenimento di un output da mostrare a schermo. Come nella *ScanActivity* all'avvio si inizializza la geolacalizzazione con *ActivateGPS*() e viene implementato *OnClickListener* per avviare la ricezione

alla pressione del pulsante. *OnClick*() permette di effettuare le operazioni di localizzazione (con *Localize*()) e richiesta al server sulla base della propria posizione geografica con *connect*(). I dati ottenuti verranno tradotti a XML a linguaggio comune con l'utilizzo della classe $XMLParser$ e mostrati a schermo nella relativa $textBox$.

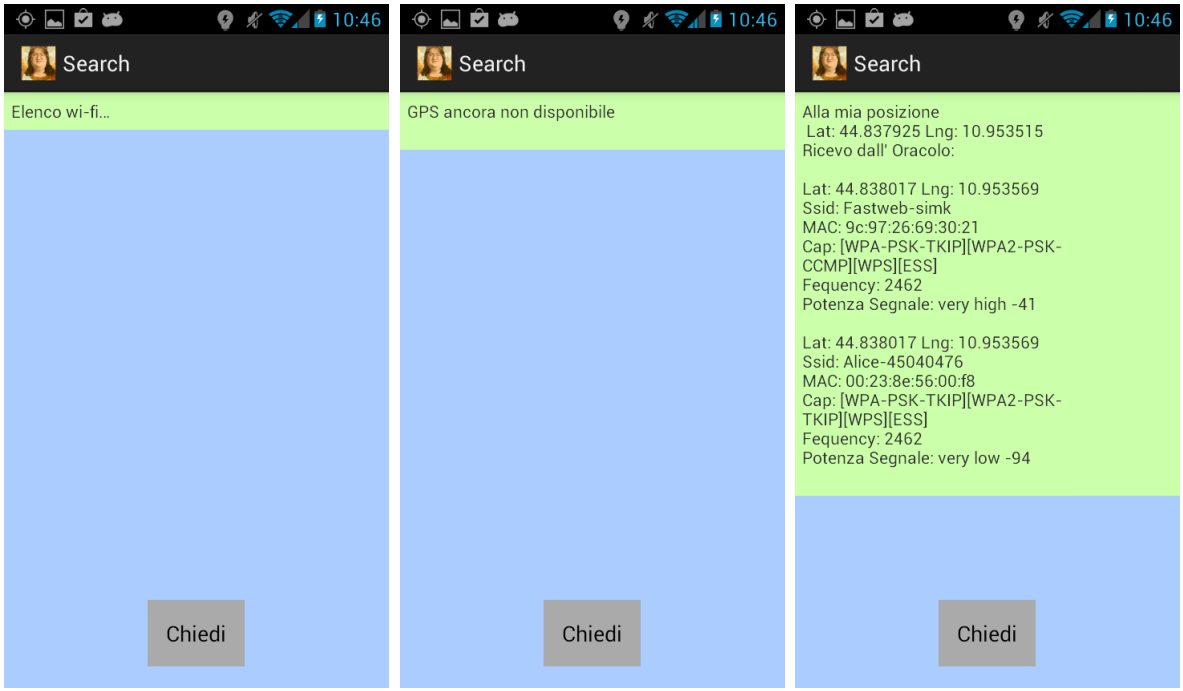

Figura 6.3: SearchActivity

6.2 Lato Server

6.2.1 Moduli e risorse

Il server come già accennato è organizzato in moduli, da me definiti "Controller" poichè svolgono l' attività di controllare e gestire determinate azioni che sono raggruppate logicamente. Qui sono elencati brevemente moduli ed altre risorse con i rispettivi compiti:

- *index.php* Modulo a cui vengono reindirizzate tutte le richieste. Si occupa di avviare gli altri moduli se esistenti o rifiutare le richieste se malformate.
- $wificatcherController.php$ si occupa della ricezione dei dati inviati dal client e del opolamento del database.
- $wifidispatcherController.php$ si occupa di fornire al client le posizioni degli access point vicini.
- *searchOracle.php* è un modulo contentente funzioni varie che operano sul database, utilizzate dagli altri moduli.
- *oracolo.xml* è il documento XML che costituisce il database.

6.2.2 Cenni sul codice

Ora si andranno a descrivere più dettagliatamente le operazioni svolta dai moduli.

index.php

Il file *index.php* svolge un ruolo simile a quello di .htaccess su un server apache, ovvero quello di interpretare le richieste ricevute, scartare quelle malformate (incomplete o che riferiscono a risorse sconosciute)e avviare moduli specificati nella richiesta. Una richiesta valida è nel formato:

```
http:\\indirizzo\nomerisorsa\parametri
```

Dove *nomerisorsa* è il nome del modulo (es: $wificatcher$) che nel server è seguito da Controller e dall' estensione .php (es: $wificatcherController.php$) e *parametri* varia a seconda del modulo ma che rimane nel formato:

```
\parametro1\parametro2\...\parametron
```

Attualmente nessun modulo necessita di parametri poichè la comunicazione client -> server avviene tramite messaggi XML. In caso di richiesta malformata restituisce un messaggio di errore http 404.

$wificatcherController.php$

Questo modulo sfrutta librerie per la manipolazione di stringhe XML come $SimpleXMLElement$ e $DOMElement$. Il modulo sulla base di una richiesta HTTP POST, legge i dati dalla stringa in input e apre il database. Se l'input è xml corretto comincia il parsing della stringa, ricavando i nodi relativi agli access point e costruendo un DOMDocument con questi che andrà a costituire l'output

da salvare all' interno del database. Ad ogni nodo viene controllato il su campo che costituisce le coordinate e si controlla che queste non sia uguali a 0.0, in tal caso si scartano. Questo si verifica quando il client spedisce dati senza che abbia effettuato la geolocalizzazione. La scelta di non scartare tali dati direttamente dal client è stata già precedentemente specificata. Se il procedimento è andato a buon fine restituisce un messaggio testuale, visualizzabile sul client.

wifidispatcherController.php

Operando anche essa sulla base di una richiesta HTTP POST , viene letto il messaggio XML fornito dal client contenente le sole coordinate GPS in questo formato:

```
<gpsData>
        <lat></lat>
        <lng></lng>
</gpsData>
```

Tali coordinate costituiranno il centro di una ellisse terrestre di raggio prefissato, corrispondente alla portata di ricezione dell' antenna Wi-Fi del dispositivo, attualmente bloccata per debugging a 25 metri, entro il quale cominciare la prima ricerca di access point. La funzione *coordsRegion*() fornisce due coppie di coordinate minime e massime che costituiscono gli estremi di questa ellisse. Tali coppie verranno utilizzate per cercare nel database tutti gli access point che rientrano in un un' area compresa dentro queste coppie con l'utilizzo della funzione *searchGpsDataInOracle*(). I nodi restituiti verranno puliti dai nodi duplicati corrispondenti agli stessi access point con l'utilizzo di *eliminateDuplicate*(). Infatti è possibile (anzi sarebbe meglio) che nel database siano presenti molteplici nodi corrispondenti gli stessi access point in modo da avere una mappatura piu

fine delle loro zone di copertura, ma tali nodi dovranno comparire una sola volta agli occhi del client. Al termine viene restituito al client il documento XML comprensivo degli access point rilevati nello stesso formato con cui è stato ricevuto dal *wificatcherController.php*.

Questo modulo andrebbe esteso con la possibilità di valutare la potenza del segnale relativa di ogni access point inferendo sulla loro posizione come accennato precedentemente, ovvero attraverso la mappatura di zone di copertura con l'utilizzo di poligoni. Tale metodo necessita di un elevato numero di punti, ed in presenza di una scarsa rappresentanza nel database risulta inutile. Proprio in virtù di queste considerazioni, nel capitolo "Sviluppi Futuri", viene menzionata questa possibilità implementativa.

Capitolo 7

Criteri di valutazione

I principali criteri di valutazione per l'applicazione sono stati la portabilità, l'usabilità, la validità dei dati. Per il server oltre a questi anche la velocità di risposta. In questa sezione verranno trattate le problematiche che si potrebbero riscontrare in questi scenari.

7.1 Portabilità e usabilità

L' app lato client è compatibile sui dispositivi con sistema operativo Android a partire dalla 2.3 fino alla 4.2. Inoltre è stata mantenuta la semplicità dell'interfaccia ideata dai miei predecessori, in modo tale da non causare problemi a possibili utenti senza conoscenze informatiche. Il server è stato ideato in modo da poter essere esteso facilmente in futuro con l'aggiunta di moduli e la modifica parziale o totale delle funzioni senza dover snaturarne la comunicazione col client. L'applicazione nella sua totalità rimane comunque in fase di sperimentazione e quindi non esente da bug o imperfezioni. I punti critici riscontrati risiedono nella sincronizzazione tra dati sul server e dati sul client, per questo motivo si è scelto di effettuare una connessione al server solo dopo aver raccolto i dati e terminata la scansione, almeno fino a quando l'applicazione rimane in fase di debugging.

7.2 Test

Dal lato applicativo sono stati effettuati test di scansione e ricezione dati, in particolare sono state effettuate le seguenti prove:

1. Scansioni continuative di lunga durata o brevi scansioni ripetute nel tempo non creano errori o blocchi di nessun tipo.

2. In caso di mancata connessione al server il timeout entra in funzione correttamente, i dati rimangono salvati in locale. Viene visualizzato un messaggio di testo all'utente per notificarlo di tale evento.

3. In automobile, a velocità basse e con intervalli di scansioni anche bassi, non sono state rilevate anomalie.

4. In caso di upload al server di grandi quantità di dati (ordine di decine di MB), questi vengono correttamente inviati, a patto che il server sia correttamente configurato per accettare dati HTTP Post Request sufficientemente grandi (controllare file di configurazione *php.ini*). Il tempo di upload è direttamente proporzionale alla quantità di dati inviati.

5. In caso di invio dati al server senza aver effetuato una sincronizzazione col segnale GPS, tali dati verranno correttamente scartati dal server singolarmente. Tale operazione è completamente trasparente al client.

Capitolo 8

Sviluppi futuri

Un primo miglioramento per l'applicazione a disposizione dell' utente potrebbe riguardare la sua interfaccia, che per ora anche se funzionale è volutamente spartana in quanto il client nel suo stato attuale è impostato per fornire dati chiari, anche se non particolarmente piacevole all' occhio. Tali dati per esempio potranno essere esposti invece che con delle $TextBox$, direttamente dentro delle $ListView$ espandibili.

Una buona idea potrebbe essere quella di integrare nella status bar di Android un' icona indicativa che il servizio sia in funzione, permettendo di fermare, mettere in pausa o far ripartire il servizio stesso con l' ausilio di pulsanti. Alternativamente o in aggiunta, si potrebbe considerare di creare un widget che a sua volta permetta di gestire la scansione.

La scansione stessa potrebbe essere resa automatica o si potrebbe dare la possibilità all'utente di scegliere la modalità con cui effettuarla. Se si desidera effettuare la scansione automaticamente bisognerebbe rendere dinamico l'intervallo fra una scansione e l'altra in base alla velocità relativa dell'utente o in base alla densità

di reti in una specifica zona.

La connessione col server potrebbe essere implementata in un servizio a parte tramite l'utilizzo di un thread diverso o con un AsyncTask indipendente anche nella sua fase di richiesta all' Oracolo, così come già implementato nella fase di scansione. Tale possibilità è stata momentaneamente disabilitata in quanto è ancora nello stadio di sviluppo.

Il server seppur implementato può essere migliorato aggiungendo funzioni che stimino le zone di copertura degli access point sulla base degli algoritmi polinomiali ipotizzati sul documento di tesi di Somenzi e Paladino.

Il server in mancanza di dati sufficienti da restituire al client, come per esempio il caso in cui questo si trovi in una zona completamente assente da segnale Wi-Fi, dovrebbe costruire un percorso plausibile al primo access point più vicino, inferendo quanto più possibile sulla vicininza del client a tali access point.

Il database può essere arricchito di informazioni effettuando inferenze continuative sul suo contenuto. Per esempio al raggiungimento di un certo numero di istanze di uno stesso access point con locazioni differenti, per esempio 10 occorrenze, si può calcolare la zona di copertura effettiva (ma approssimata) di tale access point, sempre con l'ausilio degli algoritmi polinomiali, salvarla direttamente nel database *Oracolo.xml* stesso, in una zona riservata ai dati "precisi", in maniera tale da poter essere consultata successivamente dai client senza che il server debba nuovamente effettuare dei calcoli, guadagnando sul tempo di risposta.

8.1 Valutazioni sul GPS

Come già discusso nel documento di tesi di Somenzi e Paladino, l'intero sistema è fortemente dipendente dalla localizzazione GPS, ne consegue che le maggiori problematiche dipendono da essa. Una di queste è che in ambienti chiusi è difficile trovare segnale, che diventa affidabile solo dopo molto tempo o non diventa mai realmente affidabile (con scarto di più di 10 metri, dato sicuramente non trascurabile).
Per questi motivi, anche in condizioni ottimali può succedere che le prime due scansioni avvengano senza aver rilevato le coordinate geografiche del dispositivo. Un' altra problematica è che alcuni dispositivi, come per esempio certi tablet, sono sprovvisti di GPS, ma dispongono comunque di una connessione Wi-Fi.

Conclusioni

Lo scopo del progetto era quello di creare un' applicazione che fornisse la mappatura delle reti Wi-Fi di un territorio ed eventuali ulteriori servizi, come fornire la rete Wireless più vicina, o dato un determinato percorso, verificare che esso sia coperto da reti.
Inoltre era necessario costruire un server che potesse permettere la ricezione di tali dati da una scansione, la loro memorizzazione su un server condiviso e la possibilità di fornirli su richiesta, per poi poter essere visualizzati o riutilizzati in base alle diverse necessità.

Di questo ecosistema, si è continuato lo sviluppo precedentemente iniziato da Somenzi e Paladino della parte client, anche con l' aggiunta della possibilità di ricezione di dati da parte del server, si è iniziato lo sviluppo della parte server, la creazione del database e delle funzionalità di comunicazione tra client e server. Questo obiettivo è stato parzialmente raggiunto tenendo in considerazione importanti requisiti come la portabilità, l'affidabilità e le prestazioni, ed è stato ideato un possibile sviluppo futuro per poterne migliorare l' efficienza.
Rimangono tuttavia aspetti determinanti relativi alla disposizione di access point sul territorio e alla localizzazione GPS che ne minano per ora l' efficacia per un suo utilizzo effettivo.

Bibliografia

[1] “A Toolkit for Automatically Constructing Outdoor Radio Maps”, Indiana University: http://www.cs.indiana.edu/surg/Publications/itcc2005_toolkit.pdf
Usato per IDW e ultimo algoritmo

[2] “WiFi Mapping Software: Footprint”: http://www.alyrica.net/wifi_mapping
Fornisce un possibile script in python implementabile sul server per generare poligoni dati i punti.

[3] “Wireless Cyber Assets Discovery Visualization”: http://securedecisions.com/wp-content/uploads/2011/06/Wireless-Cyber-Assets-Discovery-Visualization.pdf
Cita diverse fonti di strumenti che generano mappe dati punti geografici.

[4] "Vogella Expert Android and Eclipse development knowledge": http://www.vogella.com/tutorials

[5] "Android Developers Italia": http://www.anddev.it

[6] "Stack Overflow": http://stackoverflow.com

[7] "Wikipedia":
http://en.wikipedia.org

[8] "Php.net":
http://php.net/manual/it/index.php

[9] "Android API":
http://developer.android.com/reference/packages.html

[10] "Android SDK: Asynchronous HTTP Requests":
http://mobiledevtuts.com/android/android-http-with-asynctask-example/

[11] "Android XML Parsing Tutorial":
http://www.androidhive.info/2011/11/android-xml-parsing-tutorial/

[12] "Immagini utilizzate per lo schema iniziale con licenza di riuso":
http://www.google.com/imghp?hl=it

Ringraziamenti

Ringrazio Elisabetta che con il suo affetto e la sua grinta mi ha spronato al raggiungimento di questo obiettivo.

Ringrazio Patrizia la quale ha fatto in modo che l' affetto fosse maggiore della grinta.

Ringrazio i miei genitori che hanno sempre creduto in me. Nonostante tutto.

Ringrazio la mia famiglia e chiunque mi abbia supportato e sopportato.

Ringrazio il mio relatore Vittorio Ghini, per la disponibilità e il supporto datomi.

Ringrazio Sommy e Palla coi quali ho iniziato e portato avanti questo lavoro di tesi e di aver vissuto con loro momenti fatti di alti e di bassi, ma sempre proficui per la mia crescita personale. Ringrazio anche le loro famiglie, per avermi accolto e sfamato

.