Alma Mater Studiorum · Università di Bologna

**SCUOLA DI SCIENZE**
**Corso di Laurea in Informatica**

# Mapping di Access Point Wi-Fi su Android: Valutazione

**Tesi di Laurea in Reti di Calcolatori**

**Relatore:**
**Chiar.mo Prof.**
**Vittorio Ghini**

**Presentata da:**
**Andrea Somenzi**

**Terza sessione**
**2012/2013**

*A Marco, Giorgio e Brunella*
*Senza il cui continuo ed instancabile*
*supporto sarei perduto . . .*

# Introduzione

In questa opera di tesi mi sono occupato della raccolta di informazioni sulle comunicazioni Wi-Fi in uno scenario in cui più utenti si spostano in un territorio o area utilizzando degli smartphone dotati di più interfacce di rete, tra le quali una Wi-Fi, per registrare le reti wireless attive. Intendo fornire ad un server, per ora fittizio e non ancora implementato chiamato Oracolo, informazioni riguardanti la locazione e l'area di copertura di diversi Access Point in modo che questo possa mantenere un database aggiornato di queste reti e, in implementazioni future, descritte in questo elaborato, fornire ad un utente una mappa delle wireless attive.

Essendo l'applicazione, sostanzialmente, un metodo di fornire informazioni ad un Oracolo che gestisca ed elabori suddetti dati, abbiamo deciso di chiamarla Divining Steam, in analogia ai vapori vulcanici che nell'antichità, se inalati, permettevano agli Oracoli greci di avere visioni e quindi di poter fornire un responso. Allo stesso modo i nostri vapori forniscono la materia prima all'Oracolo con la quale costruire la mappa degli Access Points per poi servire le necessità degli utenti.

É stata perciò progettata, implementata e valutata, un'applicazione che opera su un dispositivo 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 che sarà poi, in futuro, inviato all'Oracolo per essere elaborato. L'Oracolo quindi gestirà queste informazioni per creare mappe degli Access point rilevati, con diverse modalità, e per fornire utili informazioni ad un utente che voglia collegarsi ad una rete.

Questo lavoro di tesi è stato svolto in collaborazione con Alberto Paladino. Io ho maggiormente approfondito gli aspetti relativi alla valutazione dell'applicazione, in particolare ragionando sui possibili algoritmi che la parte server potrà implementare, dai più semplici ai più complessi, su come rappresentare i dati raccolti, su quali prove sperimentali eseguire per giudicare l'operato dell'applicazione e infine sul come gestire l'interfaccia utente.

Il documento di tesi è diviso in diversi capitoli:

- Nello Scenario illustreremo la situazione e le motivazioni per cui abbiamo deciso di implementare la nostra applicazione.
- Nell'Obiettivo definiamo i traguardi che vogliamo raggiungere col nostro lavoro, accennando alle varie funzionalità che sono state implementate.
- Nella sezione Strumenti, elenchiamo i componenti software e hardware che sono stati fondamentali per la riuscita di quest'opera. Trattiamo anche dei linguaggi utilizzati.
- In Progettazione, spieghiamo le componenti più importanti di Android, riferendoci anche alle scelte che abbiamo fatto prima dell'implementazione vera e propria.

- in Algoritmi Lato Server, ci occupiamo di esporre le possibili implementazioni che in futuro potranno essere sviluppate sull'Oracolo.
- in Implementazione, elenchiamo e descriviamo i metodi e le classi più importanti dell'applicazione Divining Steam.
- in Criteri di valutazione, commentiamo i risultati ottenuti dai test e definiamo le priorità dell'applicazione.
- in Sviluppi futuri e Conclusioni, illustriamo come il progetto possa essere migliorato e completato dando spunti di implementazione e riassumiamo lo stato dell'opera.

# Indice

**Introduzione** **i**

**1 Scenario** **1**

1.0.1 Lato Client . . . . . 1

1.0.2 Lato Server . . . . . 3

**2 Obiettivo** **6**

2.1 Introduzione . . . . . 6

2.1.1 Lato Client . . . . . 6

2.1.2 Lato Server . . . . . 7

2.2 Funzionalità . . . . . 7

**3 Strumenti** **10**

3.1 Software . . . . . 10

3.2 Linguaggi . . . . . 11

3.2.1 Java . . . . . 11

3.2.2 XML . . . . . 12

3.2.3 Android . . . . . 12

3.2.4 Sistemi operativi utilizzati per lo sviluppo . . . . . 14

3.3 Smartphone e Tablet utilizzati . . . . . 14

**4 Progettazione** **15**

4.1 Lato Applicazione . . . . . 15

4.1.1 Activity . . . . . 15

4.1.2 Service . . . . . 16

4.1.3 Broadcast Receivers e Intents Broadcast . . . . . 16

4.1.4 Intent . . . . . 17

4.1.5 Scelte progettuali . . . . . 18

4.2 Gestione Lato Server dell'Oracolo . . . . . 19

**5 Algoritmi Lato Server** **21**

5.1 Primo algoritmo (Organizzazione 1: Aree in punti) . . . . . 21

5.2 Secondo algoritmo (Organizzazione 2: Rete più vicina) . . . . . 22

5.3 Terzo Algoritmo (Trovare baricentro) . . . . . 22

5.4 Quarto algoritmo (Cerchi concentrici) . . . . . 23

5.5 Quinto (Poligonale) . . . . . 23

5.6 Algoritmi di inviluppo convesso in un piano . . . . . 25

5.6.1 Gift wrapping, o Jarvis march — $O(nh)$ . . . . . 25

5.6.2 Graham scan — $O(n \log n)$ . . . . . 27

5.6.3 Quickhull . . . . . 29

5.6.4 Andrew's monotone chain convex hull algorithm . . . . . 30

5.6.5 Kirkpatrick–Seidel algorithm (the ultimate planar convex hull algorithm) . . . . . 33

5.6.6 Chan's algorithm . . . . . 34

5.6.7 Dynamic convex hull . . . . . 35

5.6.8 Akl-Toussaint heuristic . . . . . 35

5.7 Sesto algoritmo (IDW e interpolazione) . . . . . 36

**6 Implementazione** **40**

6.1 Manifest e permessi . . . . . 40

6.2 onCreate . . . . . 41

6.3 onClick . . . . . 41

6.4 onStartCommand . . . . . 42

6.5 ScanWifi . . . . . 42

**7 Criteri di valutazione** **44**

7.1 Portabilità e usabilità . . . . . 44

7.2 Test . . . . . 45

7.3 Sviluppi futuri . . . . . 46

7.3.1 GPS . . . . . 47

7.4 Conclusioni . . . . . 48

**Conclusioni** **48**

**Bibliografia** **49**

# Elenco delle figure

1.1 Schema dell'oracolo . . . . . 4

5.1 Schema dell'algoritmo Jarvis march . . . . . 26
5.2 Passi 1-2 . . . . . 31
5.3 Passi 3-5 . . . . . 31
5.4 Passo finale . . . . . 32
5.5 Schema dell'algoritmo IDW . . . . . 37
5.6 Algoritmo di aging . . . . . 38

# Capitolo 1

# Scenario

### 1.0.1 Lato Client

In questo capitolo ci concentreremo sullo scenario: studieremo alcune situazioni esemplari in cui l'applicazione sarà di grande aiuto. Nonostante sia possibile ipotizzarne di numerose, esse hanno tutte un elemento in comune: la mobilità dell'utente.

Una situazione classica ad esempio, può essere quella che vede un turista che passeggia per le strade di una città. É molto probabile che avrà bisogno di informazioni sui migliori luoghi dove poter mangiare, le attrazioni turistiche o indicazioni stradali, o semplicemente voglia accedere alla rete ovunque egli sia, o quasi. Oggigiorno chiunque ha uno smartphone, esiste un'applicazione per ogni esigenza e la maggior parte richiede una connessione attiva ad Internet. Uno smartphone che non riesce ad accedere alla rete perde di fatto buona parte delle sue caratteristiche 'smart'.

Purtroppo però, i contratti UMTS abituali offrono traffico limitato e velocità solitamente inferiori a quelle di una rete ADSL casalinga media, quindi l'uten-

te sarebbe decisamente avvantaggiato nel connettersi ad un Access Point 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 poi si potrebbe passare addirittura più tempo a tentare di passare da una rete ad un'altra che non a navigare. Se poi aggiungiamo il fatto che molti tablet non hanno la possibilità di connettersi via UMTS vediamo come, purtroppo, l'idea di poter essere collegati ovunque sia ancora un'utopia.

Una situazione totalmente differente potrebbe invece essere la decisione di un Comune o di un altro ente di mappare tutte le reti wireless presenti in tale zona, per i motivi più disparati. Google stessa ad 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 Wifi, 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. Che lo scopo, e quindi l'esigenza primaria, sia più simile al primo o al secondo esempio, l'idea comune è quella di

avere un'applicazione che possa svolgere in modo efficace entrambi questi compiti. Parleremo delle funzionalità dell'oracolo in modo più dettagliato nel prossimo capitolo.

L'applicazione sviluppata è compatibile con tutti gli smartphone Android dalla versione 2.2 in poi, ed è pensata per esser utilizzata da persone non necessariamente al corrente del funzionamento interno dell'applicazione. Essa ha infatti un'interfaccia spartana e quindi molto semplice, ma al tempo stesso esaustiva, grazie alla quale anche una persona poco pratica non dovrebbe avere problemi nel suo utilizzo.

L'app, che si è deciso di chiamare 'Divining Steam', si occupa 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 successivamente i dati sarà poi in grado di fornire indicazioni precise all'utente riguardo a quale rete Wi-Fi connettersi in base alla potenza di segnale di tutte quelle trovate o alternativamente, se non fossero disponibili reti nelle vicinanze, fornirà una mappa che indicherà come muoversi per trovare reti.

### 1.0.2 Lato Server

Quello che si vuole realizzare è un server centralizzato che gestisca un input e un output verso dispositivi mobili, al fine di mappare le reti wireless in una determinata area geografica. L'input, gestito tramite 'Divining Steam', consiste in elenchi di reti trovate per ogni punto geografico in cui è stata eseguita la scansione. L'output consisterà in un qualche tipo di visualizzazione per l'utente e di

meccanismo per fare sì che un device possa scegliere se usare la Wi-Fi o un collegamento UMTS entro un determinato percorso, al fine di avere una connettività continua e una spesa energetica ed economica minore. Quello che ci interessa è definire come gestire il database interno all'oracolo senza fornire, almeno per il momento, una reale implementazione.

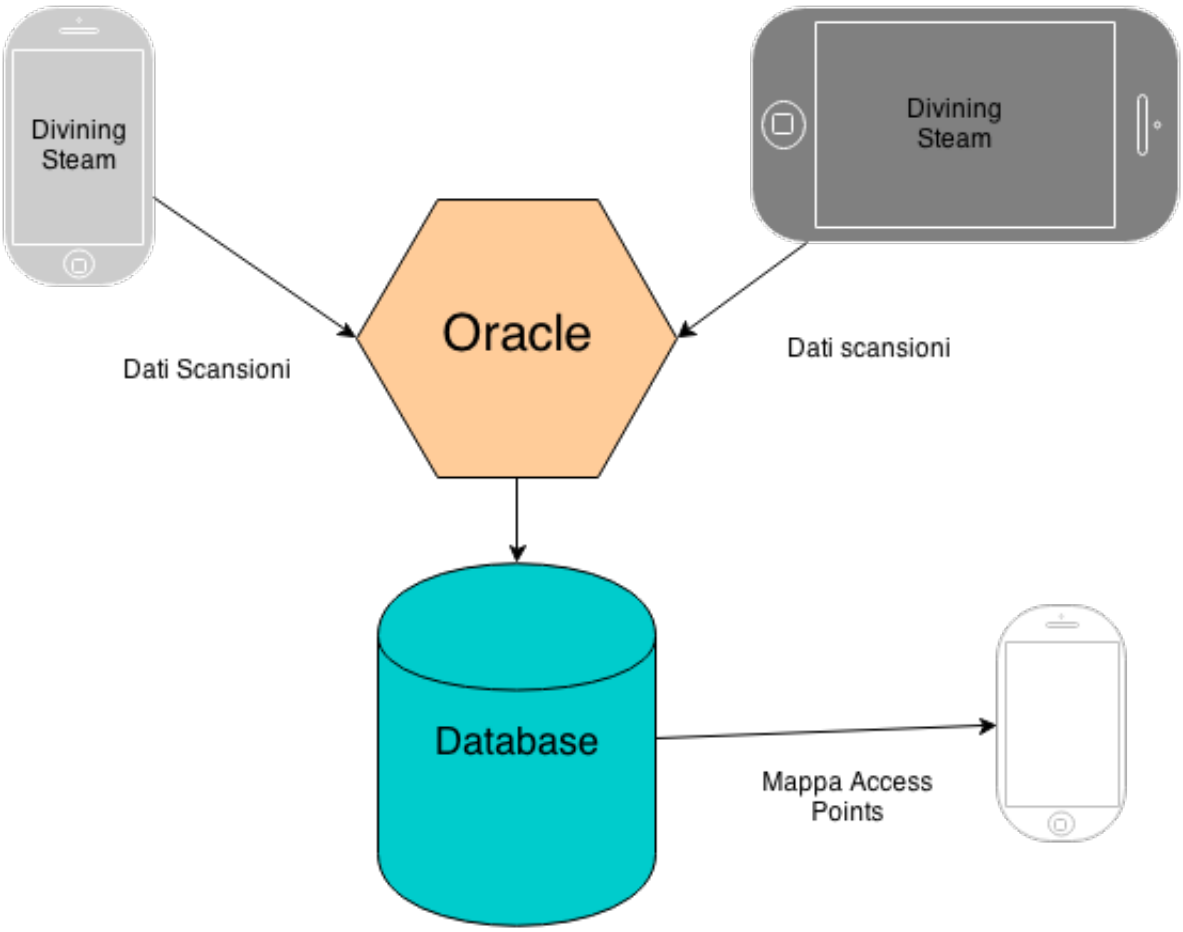


Figura 1.1: Schema dell'oracolo

Dato che i dati arriveranno all'oracolo in formato XML, anche il database interno sarà gestito nello stesso modo.

Struttura file XML in ingresso:

```
<?xml version="1.0" encoding="UTF-8"?>

<oracle>
	<session>
		<timestamp>Data</timestamp>
		<scan>
		<id></id>
			<mac></mac>
			<ssid></ssid>
			<capabilities></capabilities>
			<frequency></frequency>
			<powerlevel></powerlevel>
			<powerliteral></powerliteral>
		</scan>
	</session>

	<session>
		<timestamp>Data</timestamp>
		<scan>
			<id></id>
			<mac></mac>
			<ssid></ssid>
			<capabilities></capabilities>
			<frequency></frequency>
			<powerlevel></powerlevel>
			<powerliteral></powerliteral>
		</scan>
	</session>
</oracle>
```

# Capitolo 2

# Obiettivo

## 2.1 Introduzione

### 2.1.1 Lato Client

Come abbiamo accennato nel capitolo precedente, l'oracolo ha come funzione finale quella di suggerire all'utente quale sia l'Access Point migliore a cui collegarsi, o fornire una mappa tramite la quale possa decidere un percorso ottimale per rimanere coperto da reti wireless. Nonostante la risposta in output dell'applicazione possa sembrare una così semplice informazione però, il ricavarla è risultato di una serie di operazioni, di varia complessità.

L'applicazione infatti ha il compito di creare un database locale con tutti i dati estraibili dai vicini Access Point e associarvi la posizione GPS del dispositivo. Tramite una serie di algoritmi, che verranno discussi in un'apposita sezione di questo trattato, un server appositamente predisposto, il nostro Oracolo, riceverà questi dati e costruirà una mappa a tutti gli effetti, contenente le posizioni approssimate (in modo più o meno preciso in base all'algoritmo usato) degli Access Point e la loro area di copertura, anch'essa approssimata.

É inoltre necessario specificare che una ricezione migliore delle onde Wi-Fi emesse da un 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 ancora meglio uno Speed-Test per controllare le performance della linea.

### 2.1.2 Lato Server

Si desidera implementare un algoritmo che, ricevuti un gran numero di file XML da vari device con sopra installata 'Divining Steam', li aggreghi insieme per popolare un database sempre aggiornato contenente i vari rilevamenti e, sopratutto, riesca ad inferire la locazione più o meno esatta di ogni Wireless Access Point (AP) e una o più aree di copertura (a seconda dell'affidabilità del segnale) che rispecchino più o meno fedelmente l'area reale di copertura.

Suddetto algoritmo funzionerà su un server centrale chiamato 'Oracolo', che funzionerà da database condiviso per tutti i device. I dispositivi non faranno calcoli (se non per la fornitura dei dati) e questa parte, di conseguenza, sarà totalmente server-side a causa della complessità notevole degli algoritmi, ma anche a causa della diversità dei molteplici device Android sul mercato, che non sono sempre dispositivi dotati di una sufficiente capacità di calcolo.

## 2.2 Funzionalità

L'applicazione è stata sviluppata per soli dispositivi con sistema operativo Android, in quanto gli altri sistemi operativi mobili al momento della progettazione

non erano interessanti ai fini del progetto:

- iOS di Apple non sarebbe stato idoneo come S.O. poichè le API per sfruttare la connettività Wi-Fi sono 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 totalmente closed source, mentre Android è Open Source. Android inoltre è il più diffuso ed è presente anche su dispositivi relativamente economici.

- 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 su Smartphone.
- Inizio e stop scansione a comando:
  L'interfaccia è stata mantenuta il più semplice possibile per evitare di confondere anche l'utente meno pratico. La schermata dell'app è infatti semplicemente composta di due text-view, una per mostrare su schermo i risultati dell'ultima scansione effettuata, e una per eventuali errori o comunicazioni; inoltre abbiamo un tasto per avviare e interrompere la scansione in base alle proprie esigenze. Parleremo meglio del comportamento in seguito alla pressione del tasto in un successivo capitolo.
- Scansione ad intervalli regolari:
  In base al valore numerico contenuto dentro la text-box dell'app, è possibile impostare l'intervallo con cui verranno effettuate le successive scansioni. Un intervallo ragionevole è tra i 1000 e i 10000 millisecondi.

- 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 cose più complesse come navigare in Internet e quindi sfruttare il Wi-Fi stesso per scansionare e contemporaneamente sfruttare la connessione attiva. Approfondiremo successivamente anche questo importante aspetto.

- 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 dello smartphone o del tablet.

# Capitolo 3

# Strumenti

Come ogni progetto di questo tipo abbiamo utilizzato diverse tecnologie, qui le dividiamo in hardware e software e ne spieghiamo le particolarità e i motivi per cui le abbiamo utilizzate.

## 3.1 Software

Come IDE (Integrated Development Environment) abbiamo utilizzato Eclipse (versione v22.2.1) che, abbinato alla suite di plugins ADT (Android Developer Tools) messa a disposizione da Google stessa, ha semplificato il processo di installazione e configurazione dell'ambiente.

ADT velocizza la creazione di progetti Android, l'inizializzazione e preparazione di macchine virtuali per poter testare sul proprio computer le proprie applicazioni. Nonostante quest'ultimo aspetto sia assolutamente interessante e potente però, lo abbiamo sfruttato quasi esclusivamente nelle prime fasi di sviluppo, in quanto non è possibile utilizzare le funzioni relative al Wi-Fi nel simulatore. E' possibile infatti collegare via USB uno o più dispositivi Android al computer, e

fare il deployment direttamente sullo smartphone in un ambiente reale.

## 3.2 Linguaggi

### 3.2.1 Java

Il linguaggio utilizzato per scrivere il codice è un dialetto di Java, uno dei più diffusi e popolari linguaggi di programmazione. Uno dei suoi punti di forza maggiore è la portabilità: il codice Java compilato che viene eseguito su una piattaforma non deve essere ricompilato per essere eseguito su una piattaforma diversa. Il prodotto della compilazione è infatti in un formato chiamato bytecode che può essere eseguito da una qualunque implementazione di un processore virtuale detto JVM (Java Virtual Machine). Nel caso di Android, al posto della JVM usiamo una DVM (Dalvik Virtual Machine). Java è un linguaggio di programmazione ad oggetti che prende gran parte della sua sintassi dal C e dal C++, mantenendo però un approccio decisamente più object-oriented.

In aggiunta alle librerie standard, il programmatore può utilizzare un numero arbitrario di librerie di terze parti. Queste librerie, contenute in vari package, vengono utilizzate per importare determinati metodi o attributi comuni per semplificare e uniformare i programmi e renderli maggiormente leggibili ai programmatori.

A differenza del programma Java standard, in un'applicazione Android non c'è il classico metodo main da dove solitamente un programma comincia a inizializzarsi. Nella programmazione Android infatti, l'ecosistema è pensato per essere Event Driven, ovvero guidato dagli eventi che avvengono nel sistema, chiamati da parte dell'hardware o di altri componenti. Questo paradigma comporta che il program-

matore debba sviluppare delle routine il più possibile indipendenti. Un vantaggio è che il sistema operativo potrà ottimizzare le risorse, ad esempio rinunciando a caricare componenti (software e hardware) non supportati o non fondamentali perché inutilizzati.

Un'altro elemento molto utile è che i vari componenti possono condividere le loro funzionalità, sempre che l'utente ne conceda il permesso. Ad esempio, è possibile condividere risorse create da un'applicazione con un'altra, sia essa di terze parti o meno. In questo modo il programmatore può concentrarsi unicamente sul proprio software senza dover sviluppare anche i componenti extra, perchè già creati da altri sviluppatori.

### 3.2.2 XML

XML (eXtensible Markup Language) è un linguaggio di markup ideato per scambiare, immagazzinare e dare una struttura ai dati, non per visualizzarli. A quello scopo vengono utilizzati altri linguaggi di markup atti allo scopo come HTML. I tag di XML non sono predefiniti, l'utente può e devedefinire i propri tag in modo che siano autodescrittivi. XML è una Recommendation del W3C. Queste sue caratteristiche sono state determinanti per la scelta di usarlo come mezzo per immagazinare e gestire i dati delle wireless.

### 3.2.3 Android

Android è il sistema operativo per dispositivi mobili (ma anche fissi, come le smartTV) più usato al mondo, con più di un miliardo di telefoni e tablet in tutto il mondo che lo utilizzano. Android è un sistema operativo aperto, libero e gratuito. Privilegia la libertà di utilizzo (il controllo) a discapito della facilità di uso

a differenza ad esempio di iOS, ed è il più diffuso in assoluto sia come numero di dispositivi, arrivando ad un invidiabile 71% di quota di mercato, ma anche come tipologia di dispositivi, che vanno dallo smartwatch alla smartTV, passando per console di videogiochi e macchine fotografiche; a differenza di Windows Phone e iOS che invece si limitano a smartphone e tablet. Windows Phone, inoltre, scarseggia come utenza in confronto a IOS e Android.

Android si basa sul kernel Linux. Inizialmente è stato sviluppato da Android, Inc., supportato finanziariamente e poi comprato da Google nel 2005. Nel 2007 viene rivelato al mondo insieme alla fondazione della Open Handset Alliance, un consorzio di aziende dell'hardware, software e delle telecomunicazioni votato alla definizione di standard aperti per i dispositivi mobili.

Il 22 Ottobre 2008 esce il primo smartphone con Android, HTC Dream.

Android è sviluppato per gestire tutti i vari dispositivi che si possono trovare su uno smartphone, come schermi touch, accelerometri, giroscopi, GPS, sensori di prossimità, bluetooth, schede Wireless etc... Inoltre gestisce lo schermo secondo due layout uno verticale, detto portrait e uno orizzontale, detto landscape, in base a come lo schermo è orientato.

Il codice di Android è stato rilasciato da Google sotto la licenza Apache, permettendo la libera modifica e distribuzione da parte dei produttori di hardware e dai developer e decretandone in buona parte il successo.

Inoltre Android gestisce i servizi in background in modo che possano funzionare a prescindere da quello che l'utente fa nel mentre, implementando un vero e

proprio multitasking e dando la possibilità di far partire un'applicazione e farla andare senza che questa influisca, o sia influenzata, da altri servizi o applicazioni.

### 3.2.4 Sistemi operativi utilizzati per lo sviluppo

- Windows 7 home premium 64bit.
- OS X 10.9.1

## 3.3 Smartphone e Tablet utilizzati

- HTC Desire Z, Android 2.3.3, Display 3.7 pollici
- Asus Transformer, Android 4.1, Display 11 pollici

# Capitolo 4

# Progettazione

## 4.1 Lato Applicazione

Un'applicazione Android è caratterizzata da diversi componenti standard. Prima di spiegare meglio come abbiamo proseguito, di seguito sono brevemente spiegate le principali funzioni del sistema operativo che abbiamo utilizzato.

### 4.1.1 Activity

Un'activity è un componente che fornisce un'interfaccia utente sotto forma di singola schermata, con la quale l'utente può interagire per usare l'app. É importante notare che ogni activity risiede in una 'finestra' dedicata in cui crea un'interfaccia utente specifica per quella activity, che può occupare tutto lo schermo o essere più piccola e 'levitare' su altri componenti sottostanti.

Un'activity può avere vari stati. Essa infatti può essere attivata, creata, fermata, ripresa o distrutta, ed ognuno di questi stati può essere gestito in Android.

### 4.1.2 Service

Un servizio (service) è un componente che si occupa di eseguire operazioni a lungo termine in background, senza fornire un'interfaccia utente.
Un servizio può essere inizializzato ed avviato da dentro l'app, per poi continuare a funzionare a prescindere da quale applicazione sia attiva al momento. Qualunque applicazione inoltre può effettuare comunicazioni infraprocesso con il servizio. In sistemi chiusi come iOS, questa cosa non può avvenire.

Un servizio può venire avviato in qualunque punto del codice tramite la funzione startService(). Una volta partito un servizio può funzionare indefinitivamente, anche se l'applicazione che lo ha lanciato viene messa in background o addirittura terminata. Per questo un servizio deve essere esplicitamente fermato dall'applicazione o deve fermarsi autonomamente quando ha completato il suo compito.

Un servizio è legato (bound) ad una applicazione quando un altro componente lo lega a sè chiamando bindService(). Un servizio bound offre un' interfaccia client-server che permette ai componenti di mandare messaggi via comunicazione infraprocesso e rimane in esecuzione finchè un componente è legato ad esso. Più componenti possono legarsi al servizio, ma nel momento in cui tutti si slegano da esso, il servizio si distrugge.

### 4.1.3 Broadcast Receivers e Intents Broadcast

I Broadcast Receivers sono uno dei componenti più utilizzati nell'architettura Android. Questi componenti sono in ascolto di determinati messaggi, chiamati Intents Broadcast.

Gli Intents Broadcast sono un particolare tipo di Intent che vengono spediti attraverso il metodo sendBroadcast(). Vengono utilizzati per notificare determinati eventi alle applicazioni del sistema che sono in ascolto, in modo che possano reagire di conseguenza. Ogni applicazione nativa e non, può inviare questo tipo di messaggio. Android fa un uso molto esteso di Broadcast Intents: informazioni sulla batteria scarica, cambiamenti della connessione, chiamate e sms in arrivo sono tutti esempi di Intents Broadcast.

### 4.1.4 Intent

Un intent è più in generale, un oggetto di applicazione creato allo scopo di mandare messaggi da un componente ad un altro, in modo che il primo possa richiedere al secondo l'attivazione di una sua specifica azione. Nella nostra app usiamo gli intent principalmente in tre casi:

1. Per far partire(istanziare) un'activity, passando un intent ad una funzione startActivity(). L'Intent descrive quale activity far partire e trasporta i dati necessari.

2. Per far partire un servizio chiamando startService(). L'intent descrive quale servizio far partire e trasporta i dati necessari.

3. Per consegnare un broadcast. Il sistema attiva diversi broadcast per vari eventi del sistema, ed è possibile inviare un broadcast ad altre applicazioni passando un intent a sendBroadcast(), sendOrdererdBroadcast() o sendStickyBroadcast().

Esistono due tipi di intent, espliciti ed impliciti. Gli intent espliciti specificano il componente da attivare dal nome della classe. Generalmente si usano per attivare un componente nella propria applicazione, dato che si conoscono il nome

della classe, dell'activity o del servizio che si vuole far partire. Gli intent impliciti invece non nominano un componente specifico ma invece dichiarano un'azione generale da compiere, il che permette ad un componente di un'altra applicazione di intercettarlo.

### 4.1.5 Scelte progettuali

Vengono ora definite le scelte progettuali di come strutturare il database, come gestire e trasformare i dati, quali dati inferire da quelli iniziali.

É importante specificare che, poichè il lavoro è svolto a livello applicazione, quando ci riferiamo ai dati ottenuti dalla scansione in 'Divining Steam', è in realtà il sistema operativo al livello sottostante a fornirci questi dati. In pratica, l'affidabilità dei dati ottenuti dalla scansione nella nostra app, sarà dovuta unicamente all'operato di Android.

- Intervallo scansione

  Viene lasciata all'utente la scelta dell'intervallo fra una scansione e l'altra, con un valore minimo di 1000ms e un valore massimo da definire sperimentalmente. Altri metodi di scansione dinamica (in base alla velocità, in base a quanto un area è già stata coperta) o statica, vengono lasciate ad implementazioni future. Per ora lo scan ha dato prova di funzionare sia se usato da pedoni, sia a bordo di autovetture a bassa velocità.

- Coordinate GPS

  Ragionando sul fatto che è meglio avere dati parziali che nessun dato, e che i dispositivi mobili non dovrebbero fare calcoli (ma solo ricevere e mandare informazioni), abbiamo deciso di registrare i dati di tutte le scansioni, anche

di quelle in cui il gps non sia attivo o non ancora allineato. Sarà poi compito dell'Oracolo scartare i valori con latitudine e longitudine uguali a zero, dando così la possibilità a dispositivi sprovvisti di GPS, quantomeno, di segnalare la presenza di Wi-Fi in una determinata area, perdendo però, ovviamente, qualsiasi tipo di accuratezza. Si è rilevato che il GPS ha una precisione fino ad un metro. Non si ritiene necessario che le scansioni avvengano secondo un determinato modello ma si ritiene che scansioni casuali, per quanto in sufficiente numero, bastino per i nostri scopi.

- Wi-Fi

  Ostacoli e distribuzione della rete vanno a modificare le aree di copertura dell'Access Point, rendendole quasi mai dei cerchi perfetti o delle ellissi, ma più simili a dei poligoni.

  Per ogni punto geografico è stata segnata una potenza relativa di ogni Access Point rilevato; poi interpolando le coordinate con la relativa potenza e distinguendo le reti tramite SSID e MAC, vengono create delle forme geometriche che indicativamente mostrino l'area coperta per ogni AP.

- Trasferimento informazioni

  Si pensava di mandare i file all'oracolo tramite lettura della scheda di memoria del device collegandolo ad un qualsiasi computer e accedendo così al file XML. Lasciamo ogni metodo o ulteriore analisi ad implementazioni future.

## 4.2 Gestione Lato Server dell'Oracolo

L'Oracolo non è stato ancora implementato, ma abbiamo studiato un buon numero di possibili algoritmi per agevolare un suo sviluppo futuro. Parleremo

esaustivamente di questi algoritmi nel prossimo capitolo.

# Capitolo 5

# Algoritmi Lato Server

In questo capitolo sono elencati e spiegati tutti gli algoritmi presi in considerazione per l'Oracolo.

## 5.1 Primo algoritmo (Organizzazione 1: Aree in punti)

Una prima idea di algoritmo è la seguente: l'oracolo unisce tutti i file XML, scartando le scansioni con coordinate nulle o potenza di segnale inferiore a -80db. Crea poi un nuovo file XML per ogni wireless trovata, in cui va a mettere i dati della rete wireless e tre insiemi di coordinate, uno che contenga i punti dove la Wi-Fi ha un segnale buono (da -0db a -50db), uno dove il segnale è discreto (da -50 a 65), e uno dove il segnale è basso (da -65 a -80). Ciò non è ancora soddisfacente, nè come approssimazione, nè come inferenza sui dati (ovvero vorremmo poter estrapolare dati migliori). Però ci permetterebbe quantomeno di organizzare i dati.

## 5.2 Secondo algoritmo (Organizzazione 2: Rete più vicina)

Invece che trovare per ogni rete tutti i punti nella quale trasmette, possiamo definire per ogni punto rilevato (inteso come coordinate geografiche) le reti che sono state trovate in quel punto con la loro relativa potenza (sempre scartando coordinate nulle o valori di potenza inferiori a -80db). Unendo tutte le scansioni effettuate in quel punto (sostanzialmente in quel metro quadrato, spostandosi o meno), vengono rilevate quindi le coordinate geografiche attuali e il server potrebbe ritornare le informazioni relative al punto con coordinate più simili (vicine) trovate, entro un certo scarto. In questo modo potremmo non solo conoscere le reti alle quali possiamo accedere in quel determinato punto, ma anche la più vicina (ovvero quella con potenza maggiore).

Ciò ci rende però ciechi rispetto agli spostamenti: ad esempio potremmo chiederci: in che direzione è il AP? o essere appena fuori un'area densamente coperta e non trovare nessuna rete (anche se pochi passi in una direzione sarebbero sufficienti). Seppure un passo avanti rispetto al precedente algoritmo, non ci soddisfa ancora.

## 5.3 Terzo Algoritmo (Trovare baricentro)

Abbiamo deciso di trovare il centro dei punti semplicemente calcolandone il baricentro (ovvero facendo una media delle latitudini e delle longitudini). Non è un metodo di grandissima accuratezza nè è perfetto, ma ottiene risultati apprezzabili.

## 5.4 Quarto algoritmo (Cerchi concentrici)

Trovato il centro tramite #3 possiamo creare tre cerchi concentrici. Uno che usa come raggio la distanza fra il centro e il punto più vicino al limite dei punti buoni, uno al limite dei punti medi e il limite dei punti bassi.

## 5.5 Quinto (Poligonale)

Si è pensato di utilizzare algoritmi di Convex Hull (Inviluppo convesso) da applicare alle coordinate geografiche convertite in punti del piano. Un inviluppo convesso è una serie di punti in un piano euclideo che corrisponde alla forma più piccola contenente la serie di punti. Un modo di visualizzare pragmaticamente la cosa è quello di pensare come ai punti nello spazio come una serie di chiodini piantati sul piano, e l'inviluppo convesso come un elastico tirato in modo da contenerli e poi rilasciato. Una serie di punti è definita convessa se contiene i segmenti che connettono ogni paio dei suoi punti. Un inviluppo di una serie di punti X è definita come:

1. il minimo (e unico) insieme convesso contenente X
2. L'intersezione di tutti gli insiemi convessi contenenti X
3. l'insieme di tutte le combinazioni convesse di punti in X
4. L'unione di tutti i simplessi con vertice in X

Una combinazione convessa è una combinazione lineare (operazione che viene svolta nell'ambito dell'algebra lineare del tipo $a_1v_1 + ... + a_nv_n$) di elementi (nel nostro caso punti) fatta con coefficienti non negativi a somma 1. Cioè $\lambda_1x_1 + \lambda_2x_2 + ... + \lambda_mx_m$ dove $\lambda_1 + \lambda_2 + ... + \lambda_m = 1$ e quindi una combinazione

lineare positiva e affine. Viene definita tale perchè l'insieme di tutte le combinazioni convesse di un certo insieme di punti, al variare dei coefficienti, coincide con l'inviluppo convesso di quell'insieme. Esempi di combinazioni convesse sono la media ponderata o il valore atteso.

Per esempio, quando l'insieme è costituito da soli due punti, allora la combinazione convessa, espressa nella forma $\lambda_x + (1 - \lambda)y, \lambda \in [0, 1]$ , esprime tutti i punti contenuti nel segmento compreso tra $x$ e $y$.

In matematica, il simplesso n-dimensionale è il politopo n-dimensionale col minor numero di vertici. Il simplesso di dimensione zero è un singolo punto, il simplesso bidimensionale un triangolo e quello tridimensionale un tetraedro. Il simplesso n-dimensionale ha n + 1 vertici. Come tutti i politopi, il simplesso ha facce di ogni dimensione: queste sono tutte a loro volta simplessi. Per la sua semplicità, il simplesso è generalmente ritenuto il blocco base con cui costruire spazi n-dimensionali più complicati tramite un processo detto triangolazione.

Un inviluppo convesso di un numero finito di punti S è l'insieme di tutte le combinazioni convesse dei suoi punti. In una combinazione convessa ad ogni punto xi in S viene assegnato un peso o coefficiente ai, in modo che i coefficienti siano tutti non negativi e a somma 1. Questi coefficienti (o pesi) vengono poi usati per fare una media ponderata dei punti.

L'inviluppo convesso si può esprimere come

$$\sum_{i=1}^{|S|} \alpha_i x_i | (\forall i : \alpha \leq 0) \wedge \sum_{i=1}^{|S|} \alpha_i = 1$$

Rimanendo in due dimensioni, in questo modo creiamo dei poligoni. Ogni punto che non è l'inviluppo convesso di uno degli altri punti sarà un vertice del nostro poligono.

Dati dei punti nello spazio, creare un inviluppo convesso significa costruire una rappresentazione (poligono) univoca ed efficiente della forma che contiene suddetti punti. La complessità computazionale viene solitamente stimata in termini di $n$, il numero di punti in ingresso, e in termini di $h$, che è il numero di punti sull'inviluppo convesso.

## 5.6 Algoritmi di inviluppo convesso in un piano

Essendo i seguenti algoritmi composti da ordinamenti, i quali hanno complessità computazionale $\Omega(n \log n)$, ci aspettiamo che anche nel migliore dei casi non si possa scendere al di sotto di $\Omega(n \log n)$.

### 5.6.1 Gift wrapping, o Jarvis march — $O(nh)$

Primo algoritmo ideato e anche il più semplice, ma non il più efficiente. Da utilizzare quando il numero di punti non è troppo elevato.
L'algoritmo parte dal punto più a sinistra e sceglie il punto successivo in modo che tutti gli altri punti siano a sinistra di un ipotetico segmento tracciato fra i due punti.

Questo punto può essere trovato in $O(n)$ tempo confrontando gli angoli polari di tutti i punti rispetto a $p$. Continuando ad iterare fintanto che il punto $ph$ non è uguale al punto $P0$ ci permette di avere il nostro poligono in $h$ passaggi.

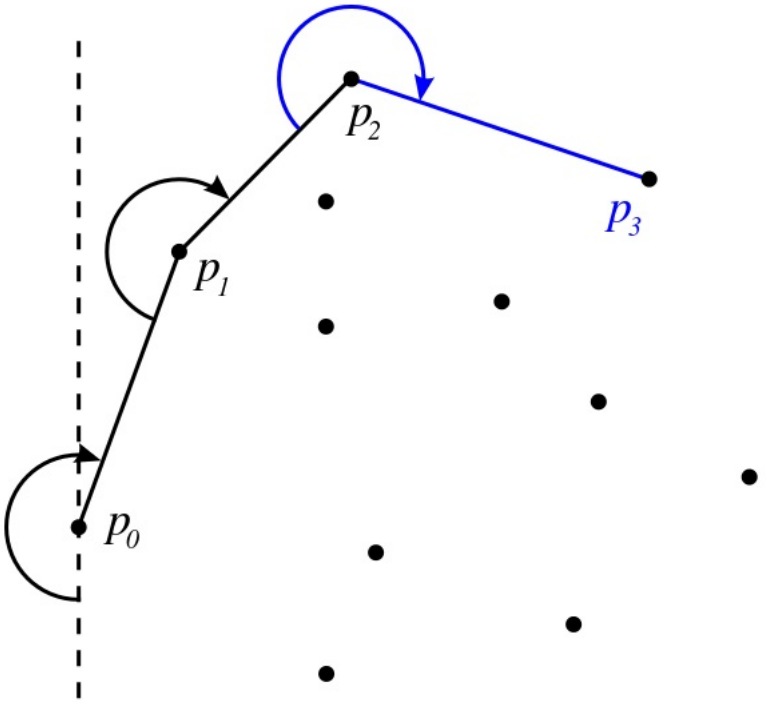


Figura 5.1: Schema dell'algoritmo Jarvis march

La complessità è quindi $O(nh)$ e quindi Jarvis è sensibile al numero di punti in output. Per numeri piccoli, quindi, riusciamo ad avere risultati migliori di $O(n \log n)$ , mentre se il numero di vertici supera $\log n$, è il caso di cambiare algoritmo

Pseudocodice

```
jarvis(S)
   pointOnHull = leftmost point in S
   i = 0
   repeat
      P[i] = pointOnHull
      endpoint = S[0]       // initial endpoint for a candidate edge on the hull
      for j from 1 to |S|
         if (endpoint == pointOnHull) or (S[j] is on left of line from P[i] to
             endpoint)
            endpoint = S[j] // found greater left turn, update endpoint
      i = i+1
      pointOnHull = endpoint
   until endpoint == P[0]    // wrapped around to first hull point
```

### 5.6.2 Graham scan — $O(n \log n)$

Per prima cosa si trova il punto con la coordinata $y$ più bassa. Se esso esiste in più punti, allora si troverà il punto con la coordinata $x$ e la coordinata $y$ più basse. Chiameremo questo punto $P$. Questo primo passaggio, come in Jarvis, necessita di $O(n)$ passaggi. I punti poi verranno ordinati in maniera crescente in base all'angolo essi e il punto $P$ hanno con l'asse $X$. Qualsiasi algoritmo di ordinamento va bene.

L'algoritmo procede poi considerando i vari punti ordinati in sequenza, e per ognuno di loro determinando se, rispetto agli altri punti, si sta eseguendo una curva verso destra o verso sinistra. Se si sta voltando a destra, il penultimo è interno e non deve esser considerato. Il processo continua finchè gli ultimi tre punti non descrivono una svolta a destra. Ogni volta che si trova una svolta a sinistra l'algoritmo passa al punto successivo. Per determinare se abbiamo una svolta a sinistra o a destra possiamo usare della semplice aritmetica.
Per tre punti $(x_1, y_1)$,$(x_2, y_2)$ e $(x_3, y_3)$ basta computare il prodotto vettoriale dei vettori che connettono i punti e $(x_1, y_1, 0)$ e $(x_2, y_2, 0)$ e i punti $(x_1, y_1, 0)$ e $(x_3, y_3)$, che è dato dall'espressione $(x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1)$ . Se il risultato è 0, i punti sono co-lineari, se è positivo i tre punti descrivono una curva a sinistra, se è negativo (o in senso anti-orario) abbiamo una curva a destra (o in senso orario). Quando si ritorna al punto di partenza l'algoritmo termina e i punti del poligono vengono ritornati in output.

Come complessità abbiamo un ordinamento di complessità $O(n \log n)$ e, mentre la complessità del ciclo che determina le curve può sembrare $O(n2)$ , in realtà è $O(n)$, perchè ogni punto è considerato al massimo solo due volte. Quindi il tempo di ordinamento domina su quello necessario per determinare le varie curve e trovare i punti che descrivono il poligono e quindi la complessità è di $O(n \log n)$.

Pseudocodice:

```
#Per prima cosa definiamo che
# Three points are a counter-clockwise turn if ccw > 0, clockwise if
# ccw < 0, and collinear if ccw = 0 because ccw is a determinant that
# gives the signed area of the triangle formed by p1, p2 and p3.
```

```
function ccw(p1, p2, p3):
    return (p2.x - p1.x)*(p3.y - p1.y) - (p2.y - p1.y)*(p3.x - p1.x)
#Poi mettiamo il risultato in un array di punti
let N          = number of points
let points[N+1] = the array of points
sAP points[1] with the point with the lowest y-coordinate
sort points by polar angle with points[1]

# We want points[0] to be a sentinel point that will stop the loop.
let points[0] = points[N]

# M will denote the number of points on the convex hull.
let M = 1
for i = 2 to N:
    # Find next valid point on convex hull.
    while ccw(points[M-1], points[M], points[i]) <= 0:
          if M > 1:
                M -= 1
          # All points are collinear
          else if i == N:
                break
          else
                i += 1

    # Update M and sAP points[i] to the correct place.
    M += 1
    sAP points[M] with points[i]
```

### 5.6.3 Quickhull

Il quickhull è un altro metodo per generare un inviluppo convesso dati dei punti nel piano. Usa l'approccio dividi e conquista in maniera simile al quicksort.

In media la sua complessità è $O(n * \log(n))$ , ma nel caso peggiore è $O(n2)$ . In circostanze normali l'algoritmo funziona sostanzialmente bene, ma il tempo di processing diventa lento nel caso in cui ci sia un alto grado di simmetria o punti posti su una circonferenza. L'algoritmo può essere descritto come segue:

1. Trova i punti con coordinate $x$ massime e minime, questi sicuramente sono parti dell'inviluppo
2. Usare l'ipotetica linea fra i due punti per dividere l'insieme di punti in due insiemi distinti, che saranno processati ricorsivamente.
3. Determinare il punto, su un lato della linea, con la massima distanza dalla linea. Questo punto, insieme agli altri due trovati prima formerà un triangolo.
4. I punti dentro a quel triangolo non possono fare parte dell'inviluppo e possono quindi essere ignorati o scartati.
5. Ripetere i precedenti due passi sulle due linee formate dal triangolo
6. Continuare fino a che non ci sono più punti, la ricorsione è finita e i punti rimanenti sono l'inviluppo convesso cercato.

### 5.6.4 Andrew's monotone chain convex hull algorithm

Questo algoritmo costruisce un inviluppo convesso a partire da punti su un piano con complessità $O(n \log n)$. Per prima cosa ordina i punti in base alla coordinata $x$ e poi in base alla coordinata $y$ in caso di valori di $x$ uguali e poi costruisce gli inviluppi superiore ed inferiore dei punti in tempo $O(n)$.

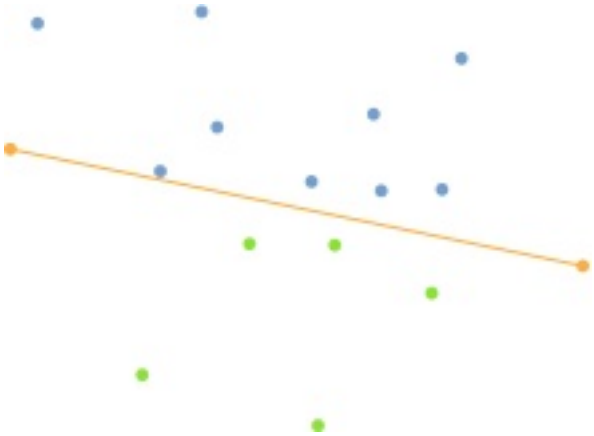

Figura 5.2: Passi 1-2

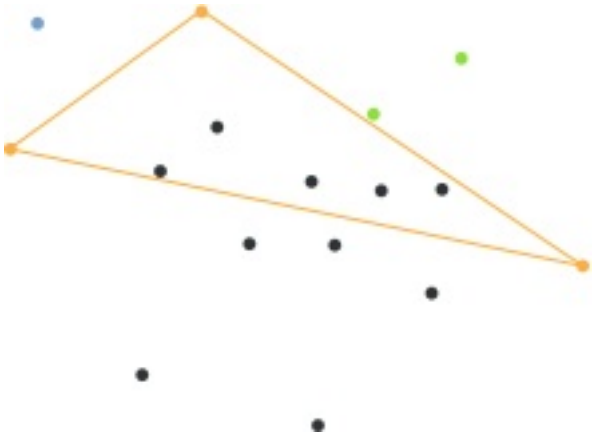

Figura 5.3: Passi 3-5

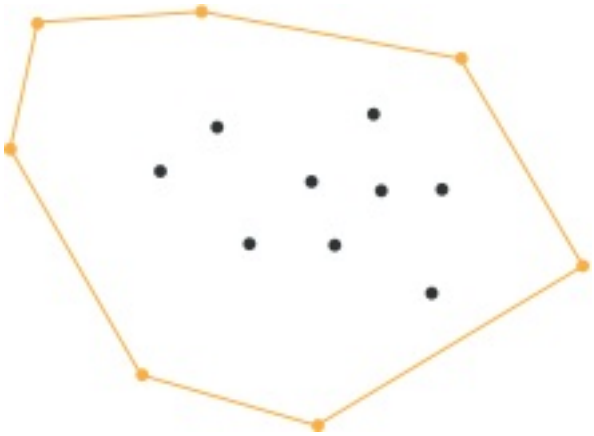

Figura 5.4: Passo finale

Pseudocodice:

```
Input: a list P of points in the plane.

Sort the points of P by x-coordinate (in case of a tie, sort by y-coordinate).

Initialize U and L as empty lists.
The lists will hold the vertices of upper and lower hulls respectively.

for i = 1, 2, ..., n:
    while L contains at least two points and the sequence of last two points
            of L and the point P[i] does not make a counter-clockwise turn:
        remove the last point from L
    append P[i] to L

for i = n, n-1, ..., 1:
    while U contains at least two points and the sequence of last two points
            of U and the point P[i] does not make a counter-clockwise turn:
        remove the last point from U
    append P[i] to U

Remove the last point of each list (it is the same as the first point of the
```

```
   other list).
Concatenate L and U to obtain the convex hull of P.
Points in the result will be listed in counter-clockwise order.
```

### 5.6.5 Kirkpatrick–Seidel algorithm (the ultimate planar convex hull algorithm)

Questo algoritmo, definito dagli stessi autori come "l'algoritmo definitivo per l'inviluppo convesso" ha complessità $O(n \log h)$, dove ha è il numero di punti dell'inviluppo. Quindi questo algoritmo dipende sia dall'input sia dall'output per quanto riguarda la complessità.

Questo algoritmo funziona sostanzialmente in maniera inversa dal *quickhull*, dato che invece di dividere e conquistare, come definito dai propri creatori, "sposa prima di conquistare" (marriage-before-conquest ). Il *quickhull* divide l'input in base alla posizione e poi fonde i vari punti esterni per creare il poligono. Questo algoritmo divide l'input come prima trovando il punto mediano delle coordinate di ascissa x dei punti, ma poi ribalta l'ordine dei passaggi successivi. Prima trova gli angoli dell'inviluppo convesso, che hanno un'intersezione con la linea verticale definita dal punto mediano delle $x$, il che richiede $O(n)$, poi i punti alla destra e alla sinistra della linea mediana che non possono contribuire all'inviluppo vengono scartati e l'algoritmo procede ricorsivamente nei punti rimanenti. Più in dettaglio, l'algoritmo fa una ricorsione separata per la parte superiore dell'inviluppo e per la parte inferiore. Nella parte superiore i punti che non contribuiscono, e che quindi vengono scartati, sono quelli sotto al bordo superiore, mentre sotto vengono scartati quelli sopra al bordo inferiore. Quindi sostanzialmente ad ogni iterazione troviamo un pezzo del poligono. Che è composto da $h$ punti. Quindi la complessità

è $O(n \log h)$

### 5.6.6 Chan's algorithm

L'algoritmo è l'ultimo sviluppato e ha una complessità di $O(n \log h)$. L'algoritmo combina un algoritmo a complessità $O(n \log n)$ come $Grahamscan$ per esempio, con il $Jarvismarch$, in modo da ottenere un buon tempo di computazione in $O(n \log h)$.

È un algoritmo notevole perchè è molto più semplice da implementare di $Kirkpatrick\breve{}Seidel$ e si estende naturalmente alle 3 dimensioni qualora dovesse essercene bisogno.

Inizialmente supponiamo che il valore di $h$ è conosciuto e definiamo un parametro $m = h$. Questa assunzione non è realistica, ma sarà rimossa in futuro. Partiamo partizionando arbitrariamente l'insieme dei punti al massimo $1 + n/m$ sottoinsiemi $Q$ con al massimo $m$ punti ognuno. Poi computiamo l'inviluppo convesso di ogni sottoinsieme $Q$ con un qualsiasi algoritmo $O(n \log n)$ , che, lavorando su insiemi con $m$ punti e non $n$, questa fase ha complessità $O(n/m)O(m \log m) = O(n \log m)$.

La seconda parte consiste nell'eseguire la $Jarvis'sMarch$ quando gli inviluppi già calcolati per velocizzare l 'elaborazione. Quindi per ogni punto $pi$, cerchiamo un punto $pi+1$ in modo che tutti gli altri punti siano alla destra della linea $pipi+1$. Questo occupa $O(\log m)$.

Usando le due fasi descritte sopra, possiamo calcolare l'inviluppo convesso di $n$ punti in tempo $O(n \log h)$, assumendo di conoscere il valore di $h$. Se definiamo $m < h$ , possiamo fermare la computazione dopo $m + 1$ passaggi, spendendo solo $O(n \log m)$ , senza però calcolare l'intero inviluppo. Possiamo quindi definire $m$ piccolo, per poi aumentarlo fintanto che $m > h$, nel qual caso viene ritornato il

poligono come risultato. Bisogna quindi scegliere $m$ adeguatamente per evitare troppe iterazioni quando $m < h$ ed evitare di sforare di molto nel caso di $m > h$.

$$\sum_{t=0}^{O(\log\log h)} O(n \log 2^{2^t}) = O(n) \sum_{t=0}^{O(\log\log h)} O(2^t) = O(n \cdot 2^{1+O(\log\log h)}) = O(n \log h)$$

### 5.6.7 Dynamic convex hull

L'inviluppo convesso dinamico serve in tutti quei casi in cui si vuole mantenere e aggiornare un poligono (inviluppo convesso) seguendo dinamicamente il cambiamento dell'input, come punti che compaiono, punti che si spostano o punti che scompaiono.

Usando un algoritmo sopra descritto è banale trovare un inviluppo significativo dei punti in ingresso. Ma mantenerlo dinamicamente non lo è affatto, dato che appena inseriamo un nuovo punto il nostro poligono potrebbe degenerare in un triangolo, mentre toglierne uno potrebbe togliere senso al poligono, facendolo collassare (un triangolo che diventa un segmento).

Dato che si suppone che l'Oracolo almeno per i primi tempi non funzioni in real-time, e che i dati cambino a distanza di settimane o mesi, si lascia la decisione, supportata da dati empirici che ancora non possiamo avere, di implementare o meno questi metodi successivamente.

Fonte: `http://www.brics.dk/DS/02/3/BRICS-DS-02-3.pdf`

### 5.6.8 Akl-Toussaint heuristic

Volendo si potrebbe usare questo algoritmo euristico per velocizzare la computazione prima di applicare un algoritmo. L'idea è di escludere i punti che in ogni caso non farebbero parte del poligono. Trovati i quattro punti con le più alte e basse coordinate nell'asse $x$ e $y$. Tutti i punti dentro a questo quadrato non

sono parte dell'inviluppo. Questo costa $O(n)$, ma può velocizzare in gran parte un algoritmo successivo.

Fonte: `http://en.wikipedia.org/wiki/Convex_hull_algorithms`

## 5.7 Sesto algoritmo (IDW e interpolazione)

Per prima cosa dobbiamo inferire una locazione più o meno accurata di ogni AP per poter fare un modello minimamente accurato. Inoltre dispositivi sprovvisti di GPS possono geolocalizzarsi una volta che la mappa viene creata sapendo solamente la potenza che captano dal AP più vicino.

Sicuramente non ci è possibile avere una copertura uniforme (ostacoli, itinerari degli operatori non uniformi) e quindi l'algoritmo dovrà essere in grado di gestire buchi nell'informazione. Per risolvere questo problema, potremmo ricorrere al metodo di interpolazione chiamato IDW (Inverse Distance Weighting) per costruire la mappa. In questo modo potremmo utilizzare i nostri punti sparpagliati per interpolare ogni punto (con un metro di approssimazione). I punti interpolati poi possono essere utilizzati in un algoritmo di creazione mappa.

L'IDW si basa sull'assunzione che l'interpolazione dei punti debba essere influenzata maggiormente dai punti a loro più vicini, e sempre meno influenzata mano a mano che aumenta la distanza dal punto. La superficie di interpolazione è una media pesata dei punti sparpagliati e del peso associato a ad ogni punto (che diminuirà all'aumentare della distanza).

Un modo semplice di calcolare la superficie interpolante usando IDW consiste

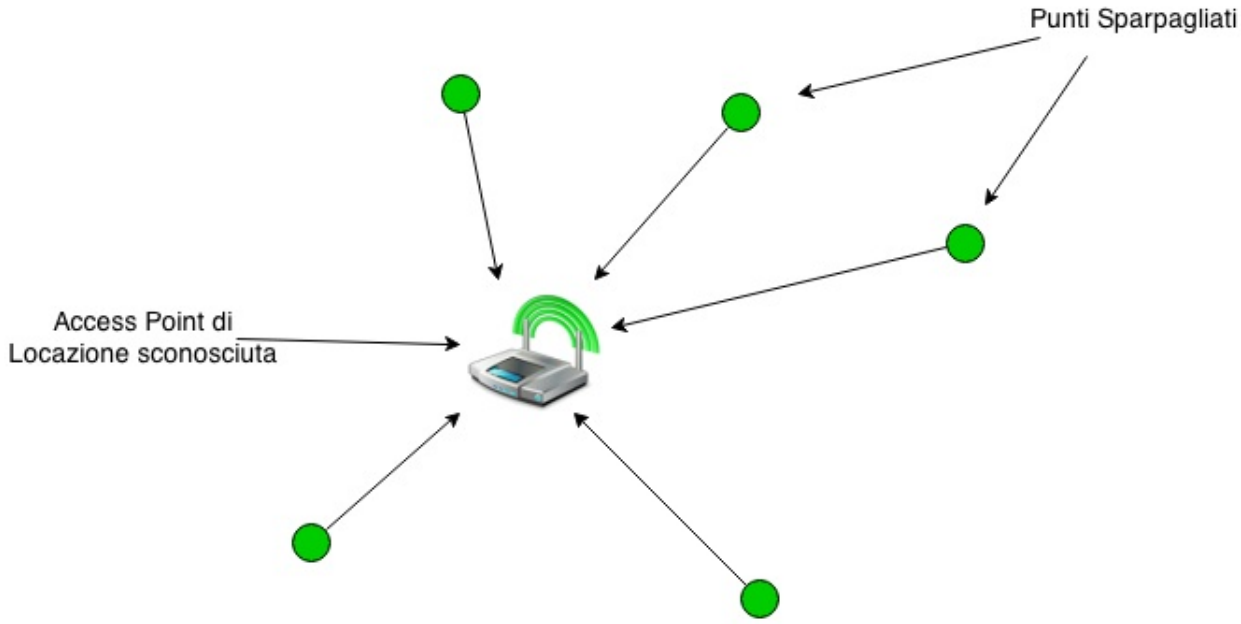


Figura 5.5: Schema dell'algoritmo IDW

nella seguente funzione:

$$F(x,y) = \sum_{i=1}^{n} w_i f_i$$

dove $n$ è la grandezza dei punti sparsi, $f_i$ è il valore corrispondente ad ogni punto, e $w_i$ è il peso associato ad ogni punto. Il peso associato ad ogni punto può essere calcolato come

$$w_i = \frac{h_i^{-p}}{\sum_{j=i}^{n} h_j^{-p}}$$

dove $p$ è un numero intero positivo, tipicamente $p = 2$ e $h_i$è la distanza dal punto di interpolazione al punto sparso. Per definizione, la somma di tutti i pesi deve dare 1.

La limitazione di questo algoritmo sta nel fatto che usando come input solo dei punti sparsi, la superficie interpolata è una semplice media dei dati ed è prona a

sbagliare nei punti estremi. Questo significa che la potenza di segnale dei punti interpolati non potrà essere minore della più piccola potenza ricavata, anche se realmente quel punto non è coperto! Dobbiamo quindi generare punti "artificiali" dove la rete non è rilevata con una potenza fittizia di -100db. Ovviamente, questi punti dovranno essere posizionati entro un certo raggio dal AP, oltre il quale sicuramente il segnale sarà troppo debole, per essere utili. Il paper suggerisce 50 metri dall'Access Point, ma questa distanza potrebbe aumentare o diminuire via prove sperimentali.

Un ulteriore algoritmo di aging può essere introdotto per scartare punti e AP quando questi dopo un certo periodo non ricevano aggiornamenti o ricevano aggiornamenti negativi (nei punti in cui ci dovrebbe essere segnale invece non c'è nulla).

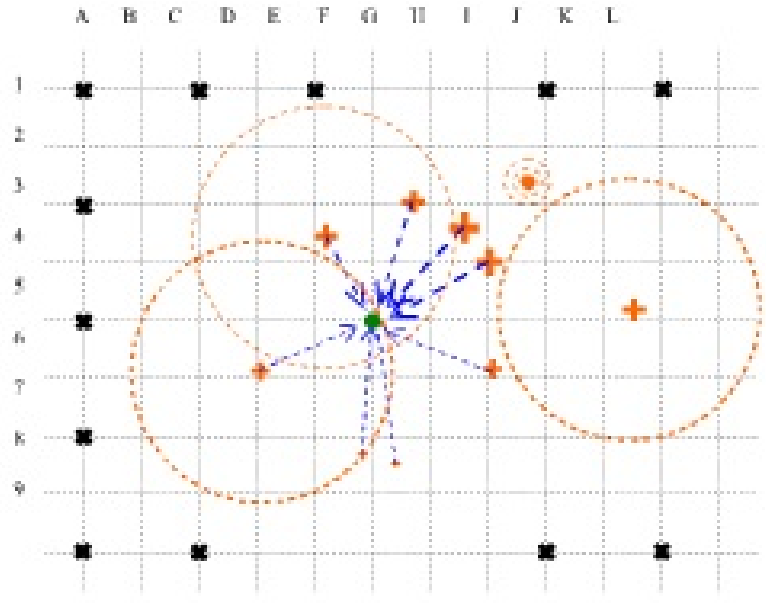

Figura 5.6: Algoritmo di aging

In questo caso i punti neri sono quelli fittizi dove non si prende il segnale, quelli rossi corrispondono ai punti trovati con il loro corrispettivo peso. Abbiamo deciso di non seguire la via dell'interpolazione. Per i nostri scopi è fin troppo complessa e dispendiosa per quanto riguarda il tempo macchina, e non ci interessa davvero sapere quanti DB riceviamo ad ogni metro dell'area coperta, ma solo i limiti di quest'area. Abbiamo quindi illustrato il metodo descritto in toolkit per mostrare una possibile implementazione, ma dubitiamo sia utile, sopratutto nei primi tempi, ricorrere a strumenti tanto potenti e dispendiosi, sopratutto prima di test e prove empiriche adeguate.

Fonte: A Toolkit for Automatically Constructing Outdoor Radio Maps, Indiana University

# Capitolo 6

# Implementazione

In questo capitolo verrà trattato brevemente, ma esaustivamente, come le varie funzionalità dell'applicazione siano state implementate.

## 6.1 Manifest e permessi

Ogni applicazione Android deve avere un file AndroidManifest.xml nella sua cartella root. Il Manifest raccoglie le informazioni essenziali per quanto riguarda l'applicazione, informazioni che il sistema Android deve avere per poter eseguire il codice dell'applicazione.

Le principali cose che il manifest definisce sono le seguenti:

- Nomina il package Java dell'applicazione. Il nome del package serve come identificatore univoco per l' applicazione.
- Descrive i componenti dell'applicazione, le activity, i servizi, i broadcast receiver e i content provider che fanno parte dell'applicazione.
- Nomina le classi che implementano ognuna delle componenti e definisce le loro capabilities (ad esempio quali intent possono gestire). Questo fa capi-

re ad Android cosa i vari componenti sono e sotto quali condizioni vanno lanciati.

- Determina quali processi ospitano le varie componenti dell'applicazione
- Dichiara quali permessi l'applicazione deve avere in modo da poter accedere a parti protette delle API e interagire con altre app.
- Dichiara i permessi che altri devono avere per interagire con i componenti dell'applicazione.
- Dichiara il livello minimo delle API Android che l'applicazione richiede e quelle ottimali (API target).

## 6.2 onCreate

In questa porzione di codice, comune a tutte le applicazioni Android, vengono effettuate le prime inizializzazioni di variabili, dell'interfaccia e del GPS. Qualora il GPS fosse ancora disattivato, 'Divining Steam' porterà l'utente alla schermata di abilitazione dei servizi di localizzazione tramite la funzione $ActivateGPS()$. Viene inoltre dichiarato un $Intent$ legato alla classe java$ScanService$. La $onCreate$ è nella classe $MainActivity.java$.

## 6.3 onClick

Questa funzione viene chiamata nel momento in cui viene premuto il pulsante Start Scan (o Stop Scan).
Nel primo caso, viene avviato un *service* (con l'intent dichiarato nella $onCreate$). Viene inoltre inizializzato un buffer dove verranno salvati temporaneamente i risultati della scansione. Viene anche impostato il timer prendendo il valore dalla

*EditBox* che, nel caso sia vuota, vale 1000(ms).
Nel secondo caso invece, il servizio avviato in precedenza viene interrotto e viene chiamata la funzione *writeToFile* che si occuperà di scrivere i risultati ottenuti in XML, identificato da un *timestamp* sulla memoria esterna del device.

É importante specificare che i risultati vengono convertiti in XML prima della scrittura su disco tramite la classe da noi implementata $XmlToFile$ e che spiegheremo successivamente. La *onClick* è relativa al bottone principale nel nostro layout, e risiede nella classe $MainActivity.java$.

## 6.4 onStartCommand

Questo metodo viene invocato all'avvio del service descritto precedentemente. Si occupa di chiamare la funzione $scanWifi$ con un intervallo in millisecondi, definiti in base all'input dell'utente nell'interfaccia. Una volta ricevuti i dati dallo scan (di tipo $WifiList$), vengono trasmessi via Intent Broadcast alla $MainActivity$, che li intercetta tramite la classe $MyReceiver$, di tipo $BroadcastReceiver$, e procede come indicato sopra. La *onStartCommand* è nella classe $ScanService.java$

## 6.5 ScanWifi

Questa funzione attiva la modalità Wi-Fi del dispositivo e avvia la scansione delle reti circostanti sfruttando le funzionalità del sistema operativo, che gestisce queste scansioni in modo indipendente. La $ScanWifi()$ restituisce in output una stringa contenente i risultati della scansione (convertiti dal tipo $WifiList$ di Android in stringa, dalla funzione $wifiToString$) uniti alla latitudine e longitu-

dine trovata dalla *OnLocationChanged*, funzione che si occupa di aggiornare la posizione dell'utente in base ai suoi spostamenti.

# Capitolo 7

# Criteri di valutazione

I criteri di valutazione per l'applicazione sono stati la portabilità, l'usabilità, la validità dei dati. In questa sezione ne parleremo brevemente e accenneremo alle problematiche che si potrebbero riscontrare in questi scenari.

## 7.1 Portabilità e usabilità

'Divining Steam' funziona su tutti i dispositivi Android con almeno la versione 2.3 installata. É stata testata in ogni sua funzionalità sia su tablet, che su smartphone, senza che si siano verificati crash o anomalie.
L'app funziona sia in modalità orizzontale che verticale e il layout è stato ottimizzato in modo da adattarsi automaticamente a qualunque dimensione e risoluzione di display. Abbiamo deciso di adottare XML come linguaggio di scambio dati, in modo da poter garantire interoperabilità.

É stata mantenuta la semplicità nell'ideazione dell'interfaccia, in modo tale da non causare problemi a possibili utenti senza conoscenze informatiche.

## 7.2 Test

Non sono stati notati rallentamenti causati dall'app neanche nei casi in cui il dispositivo fosse piuttosto datato o le scansioni durassero svariati minuti. Non sono stati rilevati surriscaldamenti nè interferenze con altri processi o applicazioni. Anche in caso di utilizzo prolungato 'Divining Steam' ha dato i risultati sperati. Sono stati effettuate più nel dettaglio diverse prove:

1. É stato verificato che la connessione Wi-Fi non subisce rallentamenti apprezzabili, anche nel caso di chiamata VOIP.

2. In seguito a numerosi tragitti a piedi, non è stato riscontrato alcun dato errato.

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

4. Per scrupolo, è stata lasciata l'applicazione aperta per 20 minuti filati a scansionare con intervalli bassi, e anche in questo caso non ci sono stati problemi.

# Sviluppi futuri e Conclusioni

## 7.3 Sviluppi futuri

Future implementazioni possono consistere in un miglioramento dell'Applicazione. L'interfaccia utente per ora è funzionale ma decisamente spartana e si potrebbe lavorare per renderla più appetibile agli occhi.

Una buona idea potrebbe essere quella di far sì che mentre l'oracolo sta scansionando, nella taskbar di Android venga indicato che il servizio è in funzione e, tramite pulsanti, permettere di fermare, mettere in pausa o far ripartire il servizio. Alternativamente o in aggiunta, si potrebbe considerare di creare un widget che a sua volta permetta di gestire la scansione.

La scansione potrebbe essere resa automatica o si potrebbe dare la possibilità all'utente di scegliere se averla manuale o automatica. Nel caso di scansione automatica bisognerebbe far si che l'intervallo fra una scansione e l'altra sia dinamico in base alla velocità relativa dell'utente o in base alla densità di reti in una specifica zona.

Una necessaria futura implementazione, è quella del lato server. L'Oracolo resta da implementare, con gli algoritmi suggeriti nella relativa sezione di questo

trattato.

Una volta che l'Oracolo sarà in funzione, l'applicazione potrebbe usare i dati delle wireless trovate per geolocalizzare il dispositivo, anche se questo è sprovvisto di GPS o se questo è dentro edifici o se per qualsiasi altro motivo il GPS non riesce ad inizializzarsi.

Una volta che l'oracolo sarà stato creato, i relativi algoritmi implementati e un tipo di output definito e realizzato, l'interfaccia utente dell'Applicazione dovrà essere modificata per gestire questo output dall'oracolo, sotto forma di Mappa e informazioni relative alla posizione.

Infine è da definire un metodo per inviare e ricevere dati dall'*Oracolo*. Sarebbe il caso di definire una comunicazione indipendente dall'applicazione e dal server, in modo da poterne cambiare l'architettura e riuscire comunque a comunicare.

### 7.3.1 GPS

Il sistema di localizzazione GPS ha presentato due problemi. La prima è che in ambienti chiusi è difficile trovare segnale, il segnale diventa affidabile solo dopo molto tempo o non diventa mai realmente affidabile (scarto di più di 10 metri, che per il nostro scopo non è una cosa da poco).
Per questi motivi, anche in condizioni ottimali può succedere che le prime due scansioni avvengano senza aver rilevato le coordinate geografiche del dispositivo.
La seconda problematica è che alcuni dispositivi, come per esempio certi tablet, sono sprovvisti di GPS.

Una soluzione al primo problema potrebbe essere quella di non avviare lo scan fintanto che il gps non sia totalmente inizializzato, ma al costo di perdere tutte le prime scansioni. Una soluzione al secondo problema, sarà fornita una volta che l'*Oracolo* sarà implementato.

## 7.4 Conclusioni

Lo scopo del progetto era quello di creare un servizio 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.
Per fare ciò, è necessaria la creazione di un database condiviso, gestito da un server chiamato *Oracolo*, un'applicazione per dispositivi mobili che fornisca i dati necessari all'*Oracolo* e infine, un metodo per visualizzare in modo ragionevole queste informazioni.

Di questo ecosistema, si è deciso di concentrarsi sulla parte client per dispositivi Android. Questo obiettivo è stato raggiunto tenendo in considerazione importanti requisiti come la portabilità, l'affidabilità e le prestazioni, ed è stato sperimentato in diverse situazioni senza intoppi. Rimangono, tuttavia, aspetti marginali che, facoltativamente, potranno essere implementate in futuro.

# 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/wiki/Convex_hull_algorithms

# Ringraziamenti

Ringrazio chiunque in qualche modo non mi abbia rispettato, mi abbia ferito, abbandonato, ignorato o calpestato. Sono state opportunità preziose per migliorarmi.

Ringrazio chiunque mi abbia supportato, ha creduto in me e mi è stato vicino. Mi avete dimostrato che c'è un motivo per continuare a lottare.

Ringrazio ogni artista, musicale e non, che con le sue opere mi ha arricchito, spronato, meravigliato e confortato.

Ringrazio Palla, Martina, Sha e Laura, Lorenza, Il Mara, Simone per essere persone 'meglio' e provare che le persone 'meglio' esistono.

Ringrazio Taty e Giglio per aver 'spaccato' insieme.

Ringrazio chi mi ha sostenuto economicamente e ha creduto in me, come la mia famiglia, o chi mi permette di fare un gran bel lavoro, come Gianni.

Ringrazio il relatore Vittorio Ghini, per la disponibilità, la chiarezza e il supporto datoci.

Ringrazio tutti gli altri famigliari che hanno sempre creduto in me.

Ringrazio i miei allievi presenti e passati, per l'affetto che mi hanno sempre dimostrato.

Ringrazio la famiglia di Palla per avrmi accolto senza riserve e Palla per avermi messo a disposizione i suoi beni materiali(Mac, Wi-Fi e Divani) durante la tesi.

Ringrazio il Metal e la musica in generale, per esistere e avermi definito negli anni.