Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Informatica

Progettazione ed implementazione di un sistema di navigazione per dispositivi mobili in contesti ospedalieri.

Relatore:
Luciano Bononi

Presentata da:
Giovanni Cozza

Correlatori:
Marco Di Felice
Emanuela Marcelli

Sessione II
Anno Accademico 2013/2014

Indice

1 Navigazione ed applicazioni *context-aware* 7
- 1.1 *Context-awareness* e dispositivi mobili . . . 7
- 1.2 Tecniche di navigazione . . . 9
  - 1.2.1 Navigazione tramite GPS . . . 9
  - 1.2.2 Tecniche di hybrid positioning . . . 10
  - 1.2.3 Navigazione inerziale . . . 11
  - 1.2.4 Indoor positioning . . . 12
  - 1.2.5 Navigazione e *augmented reality* . . . 13
- 1.3 Applicazioni di navigazione per dispositivi mobili . . . 14

2 Il progetto 19
- 2.1 La situazione del Sant'Orsola . . . 19
- 2.2 Livelli di navigazione . . . 21
  - 2.2.1 Primo livello: navigazione outdoor . . . 21
  - 2.2.2 Secondo livello: navigazione indoor . . . 24
  - 2.2.3 Terzo livello: navigazione per utenti con disabilità . . . 25
- 2.3 La ricerca della destinazione . . . 26

3 Programmazione in ambiente Android 29
- 3.1 Il sistema operativo Android . . . 29
- 3.2 Android Manifest e permessi . . . 30
- 3.3 *Activity* e *view* . . . 31
  - 3.3.1 Caratteristiche delle *activity* . . . 31
  - 3.3.2 Caratteristiche delle *view* . . . 34
  - 3.3.3 Interazioni tra *view* e *activity* . . . 35
- 3.4 I *fragment* . . . 35
- 3.5 Il database . . . 36

4 Progettazione 37
- 4.1 Sistema operativo . . . 37
- 4.2 Ricerca della destinazione . . . 38

4.3 Navigazione su mappa 40
4.3.1 Scelta della mappatura 40
4.3.2 Visualizzazione della mappa 42
4.3.3 Calcolo del percorso 42
4.4 Navigazione per immagini 43
4.4.1 Scorrimento della sequenza di immagini 44
4.5 Architettura dell'applicazione 45

5 Implementazione **47**
5.1 Interfaccia della demo 47
5.2 La *activity* principale 48
5.2.1 Inizializzazione dell'applicazione 49
5.2.2 Navigazione tra le schermate 50
5.2.3 Comunicazione con il servizio Location 51
5.2.4 Richieste al servizio Directions 52
5.3 Gestione del database 53
5.4 La sezione di ricerca 54
5.4.1 Le schermate di ricerca per padiglione e reparto 54
5.4.2 Le schermate di ricerca per servizio e trasporto 56
5.5 La schermata di informazioni 58
5.6 La schermata di navigazione su mappa 58
5.7 La schermata di navigazione per immagini 61
5.7.1 Primo algoritmo: singoli waypoint 61
5.7.2 Secondo algoritmo: coppie di waypoint 63
5.7.3 Terzo algoritmo: grafo di waypoint 66
5.7.4 Scorrimento delle immagini 68

6 Conclusioni e sviluppi futuri **71**
6.1 Conclusioni 71
6.2 Sviluppi futuri 71
6.2.1 Interfaccia 72
6.2.2 Database e server 72
6.2.3 Ricerca della destinazione 73
6.2.4 Navigazione su mappa 74
6.2.5 Navigazione per immagini 74
6.2.6 Navigazione tramite *augmented reality* 75
6.2.7 Navigazione indoor 75
6.2.8 Navigazione per utenti con disabilità 75

Introduzione

Questa tesi descrive la mia partecipazione ad un progetto del WILMA-Lab (WIreLess networks and Mobile Applications Laboratory) del Dipartimento di Informatica dell'Università di Bologna in collaborazione con il Policlinico Sant'Orsola Malpighi.

Lo scopo del progetto è lo sviluppo di una piattaforma di navigazione multi-livello che permetta agli utenti dell'ospedale di orientarsi in tutta l'area ospedaliera, sia all'interno dei padiglioni che nella zona circostante. La piattaforma comprenderebbe anche modalità di navigazione create appositamente per utenti disabili: una persona in sedia a rotelle, per esempio, verrebbe indirizzata verso le rampe o gli ascensori e non verso le scale.

Il progetto prevede che la piattaforma sia implementata all'interno di un'applicazione per dispositivi mobili scaricabile gratuitamente da Internet. In questo modo sarebbe possibile segnalarne l'esistenza su dei cartelloni all'ingresso dell'area ospedaliera e gli utenti potrebbero recuperarla in pochi secondi nel momento del bisogno.

Il mio ruolo all'interno del progetto è stato lo sviluppo di una prima versione della piattaforma, da usare come "proof of concept" per mostrarne il funzionamento e le potenzialità. A questo scopo ho partecipato ad alcune riunioni con il WILMA-Lab ed alcuni rappresentanti del Sant'Orsola; lì sono stati delineati i requisiti dell'applicazione, il contesto in cui questa verrebbe utilizzata e gli obiettivi a breve e lungo termine del progetto.

Lo sviluppo di una versione completa dell'applicazione, comprendente tutti i livelli della piattaforma, sarebbe stato impossibile per una persona sola nell'ambito di una tesi di laurea; ho mantenuto quindi l'obiettivo di progettare al meglio l'interfaccia utente e di implementare in maniera completa e funzionante almeno il primo livello, in modo da creare un'applicazione che fosse il più vicino possibile ad essere pronta per un ipotetico deployment.

Capitoli della tesi

Il primo capitolo di questa tesi è una breve introduzione che descrive la diffusione delle applicazioni *context-aware*, le principali tecniche di navigazione e il loro utilizzo in applicazioni per dispositivi mobili *general purpose*. Lo scopo è delineare il "punto di partenza" da cui sviluppare il progetto.

Nel secondo capitolo descriverò più nei dettagli il progetto stesso, elencando tutti gli obiettivi che si prefigge di raggiungere. Dividerò le *feature* della piattaforma tra quelle già implementate nella demo del progetto, quelle che potrebbero essere facilmente implementate in futuro e quelle che richiederebbero una certa quantità di ricerche e sono quindi da considerarsi obiettivi a lungo termine.

Il terzo capitolo è una breve panoramica sulla programmazione in ambiente Android, in cui descriverò alcune caratteristiche fondamentali del sistema operativo e delle librerie di supporto alla programmazione che hanno avuto una forte influenza sullo sviluppo della demo.

Nel quarto capitolo parlerò delle scelte progettuali compiute durante lo sviluppo della demo. In particolare, mostrerò le possibili interazioni tra l'applicazione e l'utente e tra l'applicazione ed alcune "entità esterne" come il database o i servizi di Google.

Nel quinto capitolo continuerò a descrivere lo sviluppo della demo, concentrandomi di più sugli aspetti implementativi. In questa sezione elencherò le classi e i metodi principali contenuti nel codice dell'applicazione, descrivendo anche alcuni algoritmi.

Il sesto ed ultimo capitolo è dedicato alle conclusioni. Al suo interno riassumerò i risultati ottenuti dalla demo e parlerò più in dettaglio dei possibili sviluppi futuri del progetto.

Capitolo 1

Navigazione ed applicazioni *context-aware*

Un tablet che cambia la disposizione degli elementi sullo schermo quando viene tenuto in orizzontale, una mappa che segue automaticamente la posizione e l'orientamento dell'utente, un telefono che aumenta o diminuisce la luminosità dello schermo in base a quella dell'ambiente circostante: questi sono alcuni esempi di applicazioni "context-aware".

Il concetto di *context-awareness* è piuttosto nuovo nel campo dell'informatica, ed indica la possibilità che un'applicazione sia in grado di ottenere dei dati riguardo il "contesto" in cui viene eseguita e di utilizzarli per migliorare le proprie prestazioni e l'esperienza offerta all'utente.

1.1 *Context-awareness* e dispositivi mobili

Poco fa ho accennato a come la *context-awareness* sia un concetto relativamente nuovo: fino a non molti anni fa, infatti, il contesto delle applicazioni era composto esclusivamente da alcune caratteristiche della macchina su cui venivano eseguite. Anche dopo la diffusione di Internet e delle reti di computer in generale, questo contesto poteva comprendere al massimo lo stato di altre macchine connesse alla prima e dei canali di comunicazione tra di esse.

La *context-awareness* vera e propria è arrivata insieme alla diffusione su larga scala dei dispositivi mobili come smartphone e tablet: questi oggetti possiedono infatti una serie di strumenti ideati proprio per allargare il contesto in cui le loro applicazioni si possono muovere.

Quasi tutti gli smartphone, per esempio, contengono un modulo GPS, un accelerometro ed una videocamera digitale. Altri dispositivi, più specializzati, possono disporre di una grande varietà di sensori: si pensi ad esempio

ad alcuni dispositivi usati in campo medico, che monitorano le condizioni di salute di un paziente e prendono automaticamente dei provvedimenti nel caso vengano rilevati cambiamenti preoccupanti.

Queste innovazioni in ambito hardware hanno portato ad una grande espansione delle possibilità di *awareness* delle applicazioni. Grazie al modulo GPS, per esempio, qualunque applicazione può sapere in qualsiasi momento la posizione geografica dell'utente; sfruttando l'accelerometro può sapere se l'utente la sta eseguendo da fermo o in movimento; grazie ai sensori di luce può rendersi conto delle condizioni di luminosità dell'ambiente, e così via.

Questo ampliamento delle possibilità, unito ad una diffusione estremamente rapida di smartphone e tablet tra il grande pubblico, ha spinto in primo piano il concetto di *context-awareness* nell'ambito dello sviluppo di applicazioni per dispositivi mobili. Attualmente, quasi tutti gli sviluppatori che operano in quest'ambito stanno facendo a gara per creare applicazioni che siano più *aware* possibile, o per aggiungere nuove funzionalità basate sulla *awareness* alle applicazioni già esistenti.

Tra tutti i campi in cui questa piccola rivoluzione sta avendo un grande impatto uno di quelli in cui il cambiamento è più evidente, e quello più rilevante per questa tesi, è il campo della navigazione.

Prima della diffusione dei dispositivi mobili era normale trovare sul web applicazioni di cartografia, in grado di mostrare una mappa che l'utente poteva scorrere a suo piacimento o al massimo di calcolare il percorso più breve tra due punti. Se l'utente avesse avuto necessità di un servizio di navigazione in tempo reale, avrebbe dovuto acquistare dei dispositivi appositi.

Negli ultimi anni quasi tutte queste applicazioni di cartografia si sono evolute e sono diventate delle applicazioni di navigazione a tutti gli effetti: gli utenti possono accedervi connettendosi ad Internet dal proprio dispositivo mobile, e l'applicazione sfrutta il modulo GPS del dispositivo per monitorare la sua posizione e fornire così un servizio di navigazione in tempo reale. L'esempio più famoso di questo cambiamento è probabilmente il famoso servizio Google Maps.

Quello delle applicazioni di navigazione è un esempio di *location awareness*, uno dei tipi più comuni di *awareness*: per fornire servizi migliori, l'applicazione sfrutta la capacità del dispositivo di conoscere la propria posizione geografica.

Questa tecnica può essere usata in una grande varietà di applicazioni, anche al di fuori di quelle di navigazione: si può avvisare l'utente se si avvicina a punti di interesse, si possono fornire previsioni metereologiche della zona in cui l'utente si trova attualmente, si può cambiare la lingua di un'applicazione a seconda della nazione in cui viene eseguita, e così via.

1.2 Tecniche di navigazione

Prima di discutere delle applicazioni di navigazione per dispositivi mobili, faccio una piccola rassegna delle tecniche su cui queste si basano. Alcune di esse, come il GPS, sono tecniche "standard", in uso da molto prima della nascita dei dispositivi mobili moderni; altre, come la navigazione inerziale, sono possibili solo grazie alle particolarità di questi dispositivi e possono fornire servizi completamente nuovi.

1.2.1 Navigazione tramite GPS

La sigla GPS sta per *Global Positioning System* ed indica un sistema di posizionamento e navigazione satellitare attualmente alla base della stragrande maggioranza dei sistemi di navigazione.

Questo sistema è basato sulla presenza di più di 30 satelliti, che orbitano attorno alla Terra e sono in grado di trasmettere segnali radio. Un dispositivo che sia in grado di ricevere ed elaborare questi segnali può di determinare la propria posizione geografica con un'accuratezza nell'ordine dei metri; l'unico requisito è che il ricevitore sia in grado di avere un contatto privo di ostacoli con almeno quattro satelliti contemporaneamente.

Inizialmente, per poter sfruttare il sistema GPS era necessario possedere dei dispositivi progettati appositamente a questo scopo. Con il passare del tempo e grazie agli avanzamenti tecnologici, questi "navigatori GPS" sono diventati oggetti di uso comune, e sono state aggiunte sempre più funzionalità ai loro software.

Attualmente qualunque navigatore è in grado di calcolare più percorsi alternativi per raggiungere ogni destinazione, di avvisare l'utente nel caso in cui superi i limiti di velocità, di trovare punti di interesse di vari tipi vicini alla sua posizione attuale, e così via.

Inoltre, negli ultimi anni i produttori di dispositivi mobili hanno aggiunto un modulo GPS a gran parte dei loro prodotti, che sono quindi in grado di ricevere ed elaborare i segnali dei satelliti GPS. Questo cambiamento ha dato il via alla diffusione delle applicazioni di navigazione e di altre applicazioni *location-aware* anche su dispositivi *general purpose*: al giorno d'oggi non è più necessario acquistare un dispositivo dedicato per avere funzionalità basiche di navigazione.

Questo non significa che l'epoca dei navigatori GPS sia finita: l'utilizzo del modulo GPS ed il mantenimento ininterrotto di una connessione ad Internet sono due operazioni molto dispendiose in termini di energia per dispositivi come smartphone o tablet, che non sono in grado di sostenerle a lungo.

Di conseguenza, se si ha necessità di utilizzare spesso e per lunghi periodi funzionalità di navigazione GPS l'acquisto di un navigatore specializzato, formato da hardware e software ottimizzati per questo scopo, può essere ancora una scelta sensata.

1.2.2 Tecniche di hybrid positioning

Il sistema di navigazione satellitare GPS descritto nella sezione precedente è stato sviluppato nel corso di decenni ed è estremamente affidabile ed accurato; a questo punto, migliorarne ulteriormente la precisione sarebbe un'operazione molto difficile e dispendiosa.

D'altro canto, se si volesse creare da zero un nuovo sistema di navigazione basato su principi diversi, anche avendo un'idea perfetta e rivoluzionaria sarebbe necessario un dispendio di risorse ancora più grande per raggiungere risultati paragonabili a quelli attualmente raggiunti dal sistema GPS.

Tenendo conto di questi due fattori, se si vuole migliorare ulteriormente la qualità dei servizi di navigazione il modo migliore è probabilmente l'utilizzo di tecniche di *hybrid positioning*. Questo termine indica l'idea di unire tra loro i dati ottenuti da diverse fonti per ottenere una stima più precisa della propria posizione; sfruttando questa tecnica è possibile, per esempio, usare la posizione calcolata dal sistema GPS come "punto di partenza" a cui applicare alcune tecniche più innovative.

Un semplice esempio di *hybrid positioning* può essere osservato proprio nell'implementazione dei servizi di navigazione all'interno dei dispositivi mobili. Nella sezione precedente ho accennato al fatto che questo tipo di applicazioni causino un grande dispendio di energie; nell'ambito di questo problema, un punto particolarmente critico è la ricerca dei satelliti.

Normalmente, un terminale GPS al momento dell'accensione deve attraversare una fase di ricerca dei satelliti della durata media di circa un minuto prima di poter essere utilizzabile; all'interno di questa fase, il dispendio di energie è particolarmente alto.

Per evitare il problema, i dispositivi mobili sfruttano un sistema chiamato *Assisted GPS*. Questa tecnica si basa sull'unione del sistema GPS con quelli di connessione alla rete telefonica o ad una rete Wi-Fi: il dispositivo, al momento dell'accensione, contatta l'Access Point o la cella radio telefonica a cui è connesso. In risposta, l'Access Point o la cella radio comunicano al dispositivo la lista dei satelliti visibili dalla loro posizione. A questo punto, il dispositivo può provare a connettersi a quei satelliti saltando la fase di ricerca: può infatti assumere che siano visibili anche dalla propria posizione,

che non può essere troppo distante a quella dell'Access Point o della cella radio.

Come ulteriore esempio di *hybrid positioning*, molti dispositivi mobili si affidano alle informazioni di posizionamento degli Access Point come "tappabuchi" nei momenti in cui il segnale GPS si indebolisce.

La posizione ottenuta in questo modo è ovviamente molto meno precisa, ma permette alle applicazioni di non interrompere completamente le funzionalità in caso di ostacoli momentanei al segnale satellitare.

1.2.3 Navigazione inerziale

Tra le varie tecniche che è possibile "ibridare" con il sistema GPS per migliorarne le prestazioni, la *navigazione inerziale* è particolarmente interessante in quanto è possibile solo grazie alla *context awareness* dei dispositivi mobili.

Questa tecnica si basa sull'utilizzo dell'accelerometro e del giroscopio presenti nel dispositivo per monitorare la velocità lineare e quella angolare dell'utente; fatto questo, è possibile utilizzare questi dati per stimare la sua posizione.

Perchè sia possibile usare la navigazione inerziale, è necessario che il dispositivo conosca la sua posizione di partenza; questo requisito può essere facilmente soddisfatto usando le normali tecniche di navigazione satellitare. Fatto questo, il sistema di navigazione inerziale modifica la sua stima della posizione dell'utente applicando la sua velocità a questa posizione di partenza.

Attualmente, i sistemi di navigazione inerziale sono in grado di funzionare da soli solo per breve tempo: a causa dell'inevitabile accumulo di piccoli errori strumentali nella rilevazione dell'accelerazione, la posizione calcolata si allontana sempre di più da quella reale con il passare dei secondi.

Per ovviare a questo problema, è necessario che il sistema si "calibri" periodicamente, ottenendo da un'altra fonte una posizione affidabile da usare come posizione di partenza e riavviando l'algoritmo.

Anche le tecniche di navigazione inerziale non possono funzionare da sole, un loro sviluppo potrebbe portare ad un risparmio di energia nella navigazione su dispositivi mobili: invece di usare costantemente il sistema GPS per monitorare la propria posizione, il dispositivo potrebbe sfruttarlo solo per calibrare periodicamente il suo sistema di navigazione inerziale, ed affidarsi ad esso per il resto del tempo.

Un altro approccio possibile sarebbe quello di usare il GPS come modalità principale di posizionamento ed usare la navigazione inerziale come "tappabuchi" nei punti in cui il segnale satellitare viene interrotto per breve tempo; per esempio, in un'applicazione di navigazione stradale potrebbe stimare la posizione di una macchina che sta attraversando un tunnel.

1.2.4 Indoor positioning

L'idea alla base della navigazione indoor, o *indoor positioning*, è piuttosto semplice: estendere i servizi di navigazione in modo che comprendano anche il posizionamento dell'utente all'interno degli edifici. Attualmente questa tecnica viene presa in considerazione solo nell'ambito di grossi edifici pubblici di cui sia disponibile la mappatura esatta.

Implementare questa idea, però, è decisamente più complesso che formularla: anche supponendo di avere a disposizione tutte le mappe necessarie, il segnale GPS non è sufficientemente preciso per essere applicato in questo ambito. Un sistema di *Indoor Positioning System*, o IPS, deve quindi fare uso di altre tecniche di posizionamento, possibilmente ibridandole con sistemi più tradizionali di posizionamento satellitare.

Attualmente, i sistemi di *indoor positioning* sono ancora molto imprecisi, ma con il proseguire delle ricerche è probabile che in tempi relativamente brevi si riuscirà a creare un sistema ibrido in grado di fornire informazioni con accuratezza accettabile sulla posizione dell'utente anche all'interno degli edifici.

Indoor positioning e Bluetooth

Una delle tecniche di *indoor positioning* attualmente disponibili è basata sull'utilizzo di dispositivi Bluetooth installati all'interno degli edifici in cui si vuole rilevare la posizione dell'utente.

Nel momento in cui il dispositivo personale dell'utente si avvicina ad uno di questi rilevatori Bluetooth, quest'ultimo comunica automaticamente la propria posizione. In questo modo, il dispositivo dell'utente sa che in quel momento si trova all'interno di un raggio abbastanza ristretto da quel punto.

Il difetto di questa tecnica è che per ottenere un sistema abbastanza esteso ed affidabile è necessario installare un numero abbastanza alto di rilevatori Bluetooth all'interno dei locali dell'edificio.

Per migliorare l'efficienza dell'*indoor positioning* tramite rilevatori Bluetooth, è possibile installare questi ultimi nei "colli di botiglia" dell'edificio. Per esempio, sarebbe possibile posizionare un rilevatore sopra ogni porta del-

l'edificio; in questo modo, l'utente verrebbe aggiornato sulla propria posizione ogni volta che cambia stanza.

Indoor positioning e navigazione inerziale

Le tecniche di navigazione inerziale possono tornare molto utili nel campo della navigazione indoor, soprattutto se ibridate con l'utilizzo dei rilevatori Bluetooth descritti nella sezione precedente.

In particolare, sarebbe possibile usare i segnali dei rilevatori Bluetooth come "posizioni di partenza" nell'algoritmo di navigazione inerziale, e sfruttare accelerometro e giroscopio per tracciare i movimenti dell'utente quando si sposta da un rilevatore all'altro.

Un sistema di questo genere permetterebbe di ridurre ulteriormente il numero di rilevatori necessari per questo tipo di *indoor positioning*.

1.2.5 Navigazione e *augmented reality*

Figura 1.1: Esempio di navigazione tramite *augmented reality*

Le applicazioni di *augmented reality* permettono all'utente di "guardare" il mondo attorno a lui attraverso la videocamera del suo dispositivo e di ottenere automaticamente informazioni su ciò che sta osservando.

Per esempio, l'utente potrebbe puntare la videocamera verso un monumento e sullo schermo, oltre al monumento stesso, potrebbero comparirne il nome, la data di creazione o un link alla pagina di Wikipedia che lo riguarda.

La *augmented reality* è una tecnica che può essere applicata ad una grande varietà di applicazioni, e la navigazione è uno degli ambiti in cui riscuote molto successo.

Per esempio, un'applicazione potrebbe determinare la posizione dell'utente non solo grazie al segnale GPS ma anche riconoscendo alcuni luoghi particolari (o *landmark*) quando questi vengono inquadrati dalla videocamera.

A livello di output, invece, non sarebbe più necessario visualizzare una mappa: l'applicazione potrebbe semplicemente mostrare le indicazioni sopra il paesaggio che l'utente vede attraverso il suo dispositivo.

Attualmente, le tecnologie alla base della *augmented reality* sono ancora in fase di ricerca: per sviluppare un'applicazione che implementi tutte le funzioni che ho appena descritto il dispositivo dovrebbe essere in grado di esaminare un flusso costante di immagini, confrontarle con un ampio database di *landmark*, e calcolare il modo migliore di visualizzare il percorso risultante.

Esistono alcune applicazioni sperimentali per dispositivi mobili, ma una vera e propria navigazione basata solo su *landmark* ed *augmented reality* non è attualmente disponibile.

Detto questo, è possibile sfruttare alcuni concetti compresi nella *augmented reality* per complementare altri tipi di navigazione.

Per esempio, nell'ambito della navigazione indoor si potrebbe aggiungere alle tecniche descritte nella sezione precedente la possibilità per l'utente di inquadrare alcuni *landmark* predefiniti e facilmente riconoscibili dal dispositivo (per esempio dei *QR code* posizionati sui muri) per ottenere informazioni sulla propria posizione.

1.3 Applicazioni di navigazione per dispositivi mobili

Nella prima sezione ho accennato al fatto che le applicazioni di navigazione sfruttano la *location-awareness* del dispositivo che le esegue, ovvero la sua capacità di conoscere in ogni momento la propria posizione geografica. Tuttavia, vale la pena specificare che queste applicazioni si avvalgono anche di molti altri tipi di *awareness* tipici dei dispositivi mobili moderni.

Per esempio, molte applicazioni di navigazione modificano automaticamente il livello di zoom della mappa in base alla velocità dell'utente: una persona che si muove a piedi avrà una visualizzazione molto diversa di una che sta guidando in autostrada. Inoltre, la velocità dell'utente può essere

usata anche per fare una stima del tempo necessario per raggiungere il prossimo incrocio, mentre l'orario ed il livello di luminosità dell'ambiente possono essere sfruttati per visualizzare la mappa nel modo più chiaro ed economico possibile.

In generale, le possibilità di *awareness* dei dispositivi mobili hanno fatto sì che il mondo delle applicazioni di navigazione si sia sviluppato molto velocemente, dando vita ad una serie di piattaforme, gratuite o non, tra cui gli utenti possono scegliere.

Google Maps

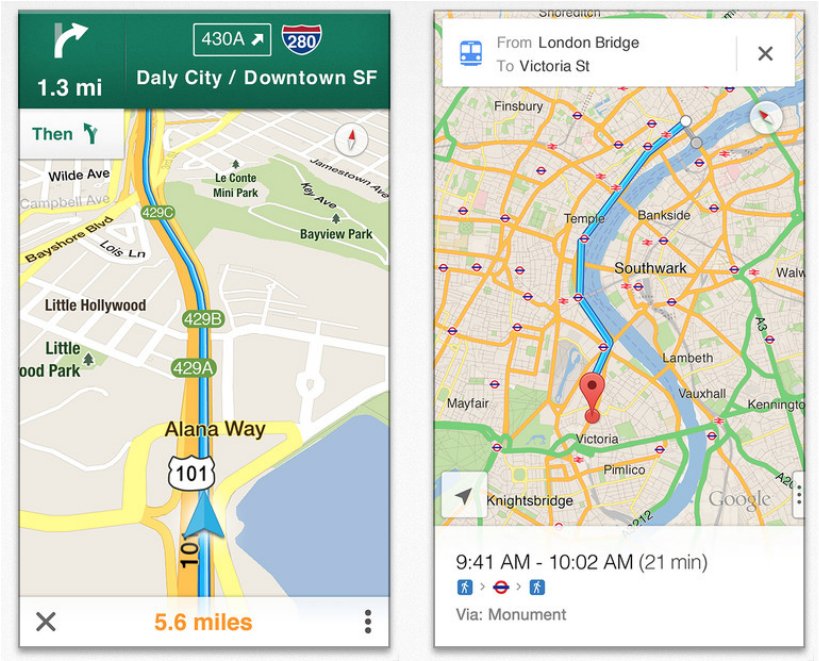


Figura 1.2: Alcune schermate dell'applicazione Google Maps per dispositivi mobili

Google Maps esisteva già da molto tempo come applicazione online di cartografia pura; con la diffusione di dispositivi mobili che usano Android come sistema operativo, Google ha fornito un'applicazione gratuita che alle funzionalità precedenti di Google Maps aggiunge quella di navigazione. Il nuovo Google Maps per dispositivi mobili è diventato in breve tempo l'applicazione di navigazione più utilizzata; questo fatto è giustificato sia dalla gratuità, sia dalla fama del "brand" Google, sia dal fatto che Google Maps è

effettivamente una delle applicazioni meglio funzionanti e contiene una vasta gamma di feature.

I punti di forza di questa applicazione sono l'affidabilità delle mappe e del servizio di navigazione, l'ampia copertura delle mappe a livello globale e la possibilità di attivare visualizzazioni alternative come Google Earth, che mostra all'utente una rappresentazione "fisica" della zona visualizzata grazie ad immagini ottenute da satelliti, o Google Street View, che mostra all'utente una serie di fotografie che rappresentano il "punto di vista" di una persona che cammina nella zona visualizzata.

Apple Maps

In risposta alla diffusione di Google Maps, la Apple ha sviluppato e pubblicato una sua applicazione di navigazione per iOS. Dopo un lancio molto controverso a causa di evidenti difetti nella funzione di navigazione e nelle mappe stesse, questa applicazione ha guadagnato terreno ed è attualmente la più utilizzata nei dispositivi Apple.

Detto questo, non ci sono differenze significative tra le feature offerte da questa applicazione e quelle di Google Maps.

Backcountry Navigator

Nonostante la presenza del "colosso" gratuito Google Maps, esiste un mercato anche per applicazioni di navigazione a pagamento che forniscano servizi aggiuntivi particolari. Background Navigator, per esempio, è progettata per essere particolarmente utile nel caso di viaggi in zone poco abitate in cui è difficile accedere ad Internet.

Questa applicazione dà all'utente la possibilità di scaricare sul proprio dispositivo la mappa della zona in cui intende recarsi; in questo modo l'applicazione potrà essere utilizzata senza accedere ad Internet, sfruttando solo il segnale GPS.

Inoltre, le mappe di questa piattaforma coprono nei dettagli anche parchi naturali, sentieri e in generale zone dove sia possibile fare escursioni.

Polaris Navigation

A prima vista, Polaris Navigation sembra una normalissima applicazione di navigazione senza caratteristiche particolari. Il suo punto di forza è la possibilità di scegliere tra diverse "fonti" da cui ottenere le mappe: l'utente può decidere in ogni momento se visualizzare le mappe di Google, quelle di OpenStreetMap, di MapQuest o di Cycle Route.

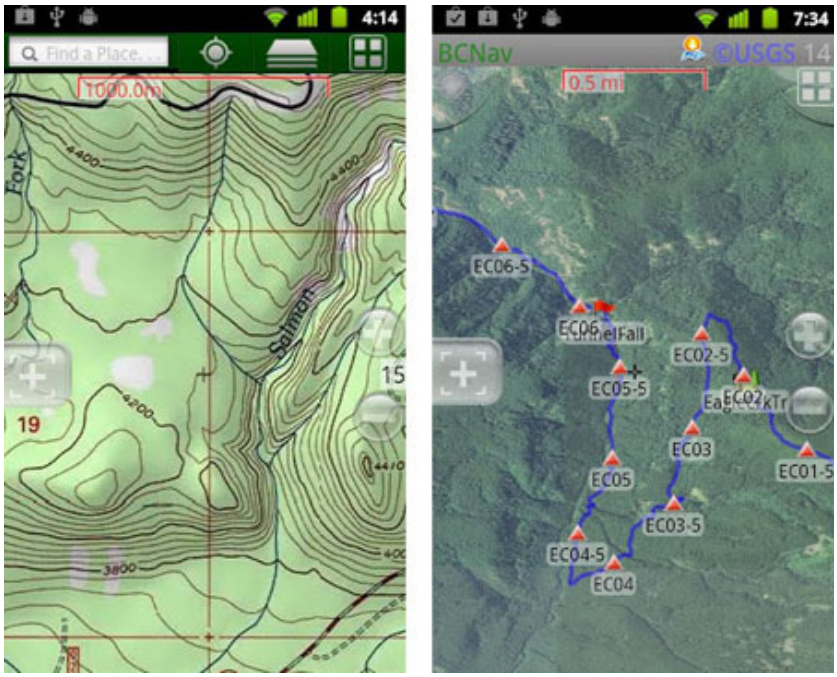


Figura 1.3: Alcune schermate dell'applicazione Backountry Navigator

Waze Social Maps

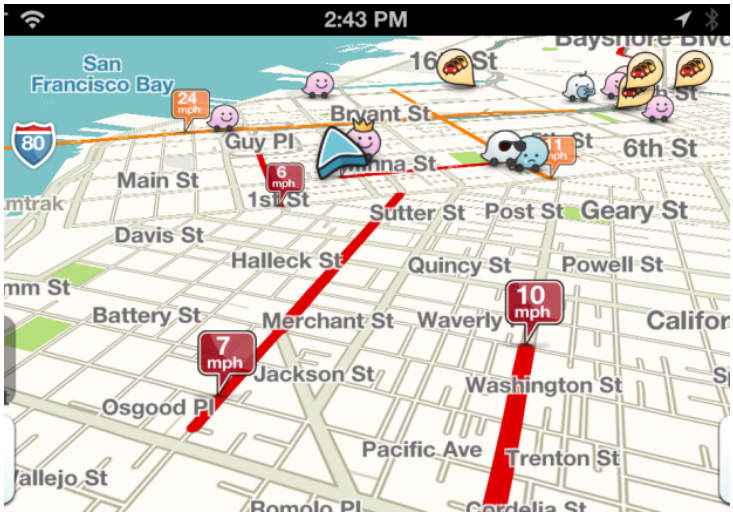


Figura 1.4: Una schermata dell'applicazione Waze Social Maps

La particolarità di questa applicazione è il fatto che sfrutta le informazioni che riceve dai vari dispositivi che la usano per ricostruire informazioni sul

grado di traffico di ogni strada. Per esempio, se molti utenti si muovono lentamente su una strada l'applicazione ne terrà conto e cercherà di escluderla dai percorsi che calcolerà successivamente.

iOnRoad Augmented Driving

Questa applicazione è un esempio di parziale utilizzo del concetto di *augmented reality* nella navigazione: per il rilevamento della posizione viene sfruttato un normale sistema di posizionamento GPS, ma il dispositivo usa le immagini rilevate dalla videocamera e i dati ottenuti da altri sensori per rilevare la presenza di oggetti in avvicinamento ed avvisare l'utente.

Per esempio, se durante la guida in autostrada l'applicazione rilevasse, tramite la videocamera, che la distanza dalla macchina successiva sta diminuendo troppo velocemente, il dispositivo emetterebbe un segnale acustico per spingere l'utente a frenare.

Oltre a questo, l'applicazione visualizza in sovraimpressione sulle immagini ottenute dalla videocamera una serie di informazioni utili, come la direzione da seguire o appunto la distanza dagli oggetti in movimento circostanti.

Capitolo 2

Il progetto

Il progetto di cui parlo in questa tesi nasce dall'idea di applicare le tecniche di navigazione descritte nel capitolo precedente in un ambito specifico, ovvero l'area ospedaliera del Policlinico Sant'Orsola-Malpighi di Bologna.

L'obiettivo finale del progetto è la creazione di una piattaforma multilivello che includa vari tipi di navigazione tra cui l'utente può scegliere in ogni momento; la struttura a livelli permetterebbe, se necessario, di effettuare lo sviluppo e il deployment di una versione della piattaforma che comprenda solo alcuni di essi e di aggiungere gli altri in seguito.

A gestire questo progetto è il WILMA-Lab (WIreLess networks and Mobile Application Laboratory) del Dipartimento di Informatica dell'Università di Bologna, in collaborazione con alcuni rappresentanti del Policlinico.

2.1 La situazione del Sant'Orsola

Il Policlinico Sant'Orsola è attualmente l'ospedale più grande d'Italia, con una media di circa 20.000 frequentatori al giorno tra pazienti, medici, personale universitario ed ospedaliero. L'area ospedaliera può essere considerata una "micro-città", formata da 27 padiglioni situati attorno ad un viale centrale lungo quasi un chilometro. Ogni padiglione contiene diversi reparti; alcuni di essi contengono anche uffici amministrativi, sale studio ed altri tipi di locali. Inoltre, alcuni reparti possono comprendere locali divisi tra vari padiglioni.

E' facile immaginare come un contesto del genere possa essere disorientante per una persona che vi arriva per la prima volta, soprattutto considerando che la maggior parte dei frequentatori sono pazienti dell'ospedale e soffrono quindi di malattie che in alcuni casi limitano le loro capacità fisiche o mentali. A questo si aggiungono altri fattori: i padiglioni sono disposti in maniera ab-

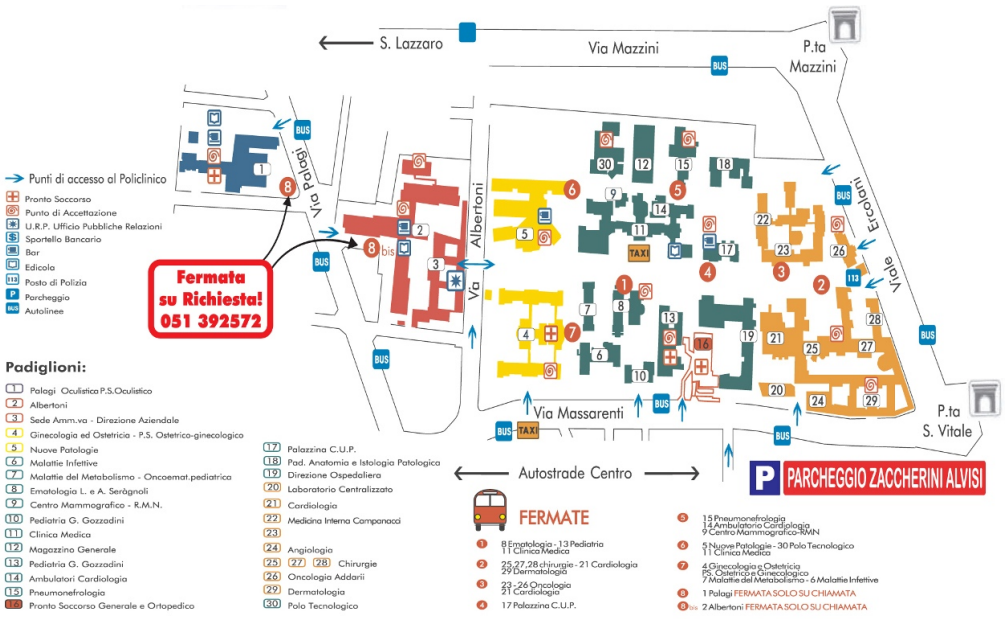


Figura 2.1: Mappa dell'area ospedaliera del Policlinico

bastanza caotica; alcuni di essi sono in una zona a parte, separata dall'area ospedaliera principale; infine, la segnaletica non sempre è sufficientemente chiara.

In una situazione di questo tipo è probabile che lo sviluppo di un metodo per orientarsi alternativo alla normale segnaletica possa migliorare l'esperienza dei frequentatori dell'ospedale; da questa considerazione è nata l'idea dello sviluppo di un'applicazione per dispositivi mobili.

Una volta completato il progetto, gli utenti potrebbero scaricare in pochi secondi l'applicazione sui propri cellulari e scoprire facilmente il percorso migliore per raggiungere qualsiasi destinazione. Per facilitare ulteriormente questo processo la piattaforma conterrebbe modalità di ricerca ad hoc: l'utente potrebbe cercare il nome di un reparto, un ufficio, una fermata dell'autobus, o magari anche solo il nome di un medico.

E' importante notare che la piattaforma non sarebbe solo mirata ai pazienti: ogni giorno entrano nell'area ospedaliera un gran numero di corrieri, fattorini ed altri operatori che hanno la necessità di raggiungere la loro destinazione il più in fretta possibile per consegnare i loro prodotti. La difficoltà di questi operatori nell'orientarsi è già stata segnalata dal personale ospedaliero, e risolvere questo problema permetterebbe di aumentare l'efficienza del Policlinico.

2.2 Livelli di navigazione

Il progetto si divide in tre "livelli di supporto" per la navigazione, con complessità crescente di progettazione e sviluppo.

2.2.1 Primo livello: navigazione outdoor

Il primo livello della piattaforma comprende le funzionalità di navigazione outdoor all'interno dell'area ospedaliera mediante tre modalità: navigazione su mappa, navigazione per immagini e navigazione tramite realtà aumentata.

E' possibile sviluppare le prime due modalità in tempi relativamente brevi e utilizzando poche risorse, dato che sfruttano in gran parte tecnologie già affermate e disponibili pubblicamente. Le tecnologie di realtà aumentata sono invece ancora in fase di sviluppo e di evoluzione, ed una loro integrazione nella piattaforma richiederebbe più tempo e risorse.

Navigazione su mappa

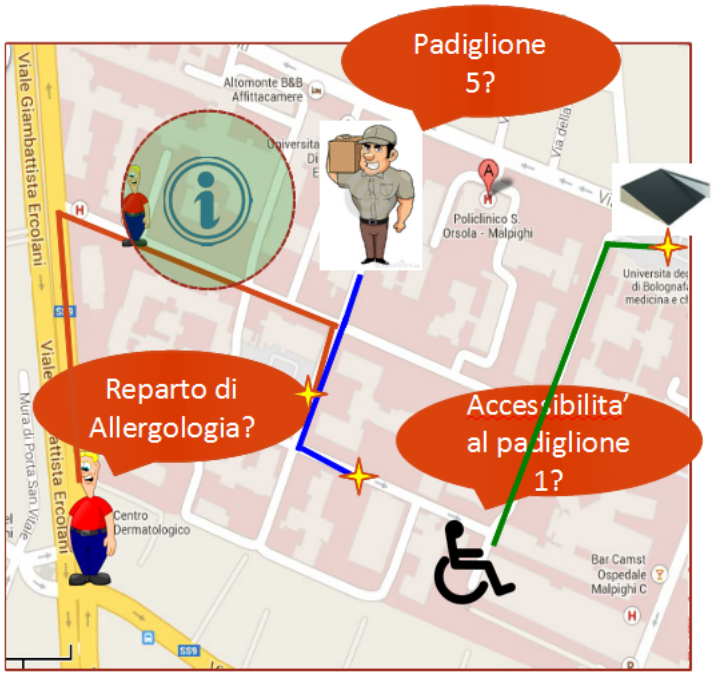


Figura 2.2: Esempio di navigazione su mappa

Questa è la modalità di navigazione più semplice e diffusa, usata di default da tutte le applicazioni di navigazione *mainstream* come Google Maps.

In generale, il dispositivo visualizza sullo schermo una mappa della zona in cui l'utente si trova e segnala graficamente su di essa in qualche modo la propria posizione. L'utente seleziona la sua destinazione, generalmente digitando un indirizzo, e l'applicazione disegna sulla mappa il percorso da seguire per raggiungerla.

Molte applicazioni di navigazione hanno inoltre una feature di auto-correzione della rotta: se l'utente si allontana dal percorso stabilito dall'applicazione questa se ne accorge e calcola un nuovo percorso, usando la nuova posizione dell'utente come punto di partenza.

Nell'ambito di questo progetto, queste semplici feature di navigazione su mappa sono sufficienti per la piattaforma, con alcune piccole correzioni.

La prima modifica è il fatto che non c'è necessità di rappresentare nella mappa zone al di fuori dell'area ospedaliera del Policlinico Sant'Orsola e delle immediate vicinanze: questo rende possibili alcune opzioni di sviluppo, come una nuova mappatura ad hoc della zona, che non sarebbero considerabili per aree più grandi.

Inoltre, l'utente deve poter scegliere la sua destinazione da una lista di "punti di interesse" prestabiliti e divisi in varie categorie. Per esempio, se un paziente non si ricordasse il nome del reparto in cui si vuole recare, deve poter scorrere una lista di tutti i reparti in cui cercarlo.

Navigazione per immagini

Figura 2.3: Esempio di rappresentazione di strade mediante immagini

La modalità di navigazione per immagini è pensata per gli utenti che, per qualunque motivo, abbiano difficoltà ad orientarsi con una mappa, o che lo trovino semplicemente scomodo.

Questo tipo di navigazione prevede che il percorso verso la destinazione non sia rappresentato come una linea su una mappa ma come una serie di fotografie, ciascuna delle quali rappresenti in modo chiaro e riconoscibile una parte del percorso.

In questo modo l'utente, una volta selezionata la destinazione, può guardarsi attorno fino a trovare la strada ritratta nella prima immagine e seguirla; all'incrocio successivo si dirigerà verso la strada ritratta nella seconda immagine, e così via. Lo scorrimento delle immagini può essere fatto manualmente dall'utente, oppure l'applicazione può passare automaticamente alla fotografia successiva in base agli spostamenti del dispositivo.

Come nel caso della navigazione su mappa, è possibile aggiungere all'applicazione una funzione di auto-correzione della rotta che ricostruisca la sequenza di immagini nel caso in cui l'utente si sia allontanato troppo dal percorso.

L'implementazione di questa modalità di navigazione è resa possibile dal fatto che il progetto riguarda un'area relativamente piccola: prima di poterla utilizzare è infatti necessario disporre di un database contenente le fotografie di tutte le strade comprese nell'area ospedaliera, ciascuna ritratta da diverse direzioni.

Navigazione tramite realtà aumentata

La navigazione tramite realtà aumentata prevede che l'applicazione utilizzi costantemente la videocamera del dispositivo per "guardare" l'ambiente attorno all'utente. Grazie al riconoscimento di alcuni elementi particolari del paesaggio, detti *landmark*, l'algoritmo di navigazione potrebbe capire la direzione in cui l'utente sta guardando e visualizzare direttamente sul paesaggio ripreso dalla videocamera il percorso da seguire.

Ci sono due metodi possibili per implementare questa modalità di navigazione. Il primo metodo è "allenare" l'algoritmo in modo che riesca a riconoscere i *landmark* "naturali" presenti nell'area ospedaliera, come le facciate degli edifici, alcuni incroci particolari e così via. Il secondo metodo è disporre dei *landmark* "artificiali" nella zona, segnalando all'utente di inquadrarli per ricevere indicazioni dall'applicazione.

Il primo metodo è molto più complicato e costoso dal punto di vista delle tecnologie e dello sviluppo dell'applicazione, ma genera un'esperienza migliore dell'utente e non richiede modifiche dell'area ospedaliera. Il secondo

Figura 2.4: Esempio di navigazione tramite realtà aumentata

metodo non richiede l'utilizzo di tecnologie sperimentali, ma è più macchinoso da usare e impone all'ospedale di disporre i *landmark* artificiali in tutta la zona.

In ogni caso, non mi soffermerò molto su questa modalità di navigazione, in quanto non è compresa tra le feature della demo.

2.2.2 Secondo livello: navigazione indoor

Il secondo livello della piattaforma consiste nell'estendere le funzionalità di navigazione del primo livello anche all'interno dei singoli padiglioni, aggiungendo alla piattaforma un *Indoor Positioning System*.

Nella sezione del primo capitolo dedicata agli IPS ho elencato una serie di problemi che devono essere affrontati da questo tipo di sistemi, ed alcuni dei metodi attualmente in uso o in fase di studio per combatterli.

Per il momento, il laboratorio ha preso in considerazione sia tecniche basate su tecnologie wireless, come il posizionamento di dispositivi Bluetooth all'interno dei locali, che tecniche basate sul riconoscimento di piccoli *landmark* artificiali tramite la videocamera.

Sono inoltre in corso delle ricerche sulla effettiva possibilità di includere in questo livello della piattaforma algoritmi di navigazione inerziale che sfruttino

Figura 2.5: Esempio di navigazione indoor

l'accelerometro ed il giroscopio presenti all'interno di smartphone e tablet. Una volta risolte le problematiche di calibrazione degli strumenti, queste tecniche potrebbero tornare molto utili per lo sviluppo di un IPS.

In ogni caso, uno sviluppo completo di questo livello richiederebbe quasi sicuramente un intervento fisico abbastanza invasivo all'interno di tutti i padiglioni e lo sviluppo completo di tecnologie attualmente in fase di studio: può essere quindi considerato un obiettivo più a lungo termine del progetto.

Per questo motivo, la demo di cui tratta questa tesi non comprende funzionalità di navigazione indoor.

2.2.3 Terzo livello: navigazione per utenti con disabilità

Il terzo livello del progetto è ancora in fase di ideazione, e riguarda l'inserimento nella piattaforma di modalità di navigazione ad hoc pensate per utenti con vari tipi di disabilità.

Alcuni esempi potrebbero essere l'aggiunta di una guida vocale alla navigazione, simile a quella già presente nei dispositivi GPS ma pensata apposta per utenti non vedenti o ipovedenti; la possibilità di calcolare percorsi ad hoc per persone con difficoltà deambulatorie di varia entità; l'aggiunta di un'in-

terfaccia basata quasi interamente su immagini per persone con difficoltà nella lettura, e così via.

Dato che questo livello è ancora in fase di ideazione, le difficoltà di sviluppo di ciascuna di queste idee non sono ancora state valutate con precisione. In generale non dovrebbero richiedere l'utilizzo di tecnologie particolarmente avanzate, ma potrebbero richiedere un certo dispendio di risorse.

Per esempio, lo sviluppo di una modalità di navigazione per utenti con difficoltà di deambulazione richiederebbe una mappatura completa di tutti gli elementi problematici in questo senso presenti nell'area ospedaliera, e lo sviluppo di un algoritmo di ricerca del percorso in grado di riconoscerli ed evitarli. Questo porterebbe quasi sicuramente all'impossibilità di usare dei servizi di mappatura e di ricerca del percorso già esistenti.

In ogni caso, anche questo livello della piattaforma non è trattato estensivamente da questa tesi.

2.3 La ricerca della destinazione

A prescindere da quali modalità di navigazione vengano effettivamente incluse nella piattaforma, un punto chiave del suo sviluppo è l'interfaccia che permette all'utente di scegliere il "punto di interesse" che vuole raggiungere.

All'interno di questa sezione si svolgerà infatti la maggior parte delle interazioni tra l'utente ed il programma, e lo sviluppo di un'interfaccia piacevole e facilmente utilizzabile avrà un grande peso nel determinare se i potenziali utenti decideranno di usare o meno la piattaforma.

La prima cosa da fare è decidere quali siano i tipi di punti di interesse da includere nel database. Per fare ciò, è necessario pensare a chi saranno gli utenti dell'applicazione e a quali saranno le loro necessità.

Un paziente, per esempio, potrebbe entrare nel Policlinico sapendo il nome del reparto in cui deve recarsi, o magari solo il nome di un esame che deve fare o del medico che deve vedere. I corrieri ed i fattorini potrebbero conoscere solo il numero del padiglione a cui devono effettuare la consegna. Il personale dell'ospedale generalmente non avrà bisogno dell'applicazione, ma in alcuni casi potrebbe voler cercare un reparto o un padiglione in cui non era mai andato.

Tipi di punti di interesse

La lista esatta dei tipi di punti di interesse da includere nell'applicazione non è ancora definitiva ma deve includere almeno alcune categorie di base, già implementate nella demo:

- Padiglioni: l'utente ha la possibilità di cercare un padiglione di cui conosce il numero;
- Reparti: se l'utente inserisce il nome di un reparto, l'applicazione scopre in che padiglione si trova e dà indicazioni per raggiungerlo;
- Mezzi di trasporto: l'applicazione permette agli utenti di cercare anche fermate dell'autobus, parcheggi, punti in cui si possono trovare dei taxi, eccetera;
- Servizi: questa categoria di punti di interesse comprende i vari servizi offerti dal Policlinico: punti di accettazione, bancomat, bar, e tutto ciò che non rientra nelle altre categorie.

Altri tipi di ricerca non implementati nella demo ma che potrebbero essere inclusi in versioni future sono:

- Medico: l'applicazione potrebbe fornire all'utente informazioni sul reparto (o i reparti) in cui un certo medico lavora, ed indirizzarlo verso di esso (o uno di essi);
- Esame: se l'utente è all'ospedale per sottoporsi ad un esame, l'applicazione potrebbe indirizzarlo verso la struttura in cui questo esame viene effettuato.

Ricerca del punto di interesse

Perchè l'interfaccia di ricerca della destinazione sia in pari con gli standard delle applicazioni moderne, deve includere sia una funzionalità di ricerca manuale in cui l'utente può scorrere la lista completa dei punti di interesse di un certo tipo, sia la possibilità di restringere la lista in base ad una serie di parametri.

Il tipo di ricerca più semplice e più usato è quello per nome: l'utente deve poter inserire il nome di un reparto o di un padiglione per selezionarlo. Nel caso di altri tipi di punti di interesse, invece, può aver senso una ricerca per tipo: l'utente può aver bisogno della lista di un certo tipo di servizi o di mezzi di trasporto (per esempio, tutti i bagni pubblici e tutte le fermate di taxi).

Inoltre, è utile che i campi di ricerca testuale abbiano una funzione di autocompletamento, ovvero che mostrino all'utente i risultati della sua ricerca man mano che lui immette del testo nel campo di ricerca.

Supponiamo per esempio che l'utente abbia bisogno di raggiungere il reparto di cardiologia. Nel momento in cui avrà scritto "card" nel campo di ricerca, l'applicazione gli dovrà mostrare una lista di tutti i reparti il cui nome contiene la stringa "card". In questo modo, l'utente potrebbe notare il reparto di cardiologia tra essi e selezionarlo senza bisogno di finire di scrivere la parola.

Questa funzione permette agli utenti di risparmiare solo pochi secondi per ogni ricerca, ma è comunque molto apprezzata dal punto di vista dell'usabilità.

Capitolo 3

Programmazione in ambiente Android

Prima di descrivere l'implementazione vera e propria del progetto, mi soffermerò su alcune caratteristiche della programmazione in ambiente Android e sulle API che ho utilizzato durante lo sviluppo. Gli argomenti di cui parlerò in questo capitolo sono stati fattori molto importanti nello sviluppo della demo della piattaforma ed hanno motivato le scelte implementative elencate nei capitoli seguenti.

3.1 Il sistema operativo Android

Android è un sistema operativo open-source basato su kernel Linux per dispositivi mobili, nato nel 2007 ed attualmente alla versione 4.4, chiamata "Kit-Kat". E' sviluppato dalla società Android Inc, fondata nel 2003 ed acquisita da Google nel 2005 ed è attualmente il sistema operativo più usato su smartphone e tablet, in diretta competizione con i sistemi della Apple.

Nelle sue ultime versioni, Android è basato su un kernel Linux 3.x e su una serie di middleware, librerie ed API scritte in C o C++ e distribuite su più livelli. Al livello superiore di questo stack c'è l'"Application Framework", composto da una serie di "manager" che gestiscono concorrentemente gli aspetti principali delle applicazioni in uso: le *activity*, le *view*, la telefonia, la gestione della memoria permanente, le notifiche, e così via.

Qualsiasi applicazione Android, anche la più semplice, interagisce esplicitamente o implicitamente con questo framework, che include anche librerie Java e librerie Apache Harmony.

Le applicazioni Android sono java-based; è possibile scrivere programmi in linguaggio C o C++, ma questi devono comunque essere inseriti all'interno di un programma Java per poter essere utilizzati dal sistema.

Per eseguirle, il sistema sfrutta la Dalvik Virtual Machine, una macchina virtuale alternativa alla Java Virtual Machine classica e progettata da Google per ottimizzare la poca memoria presente nei dispostivi mobili. La DVM nasconde al sistema operativo sottostante la gestione della memoria e dei thread, ed è possibile eseguirne più istanze contemporaneamente.

3.2 Android Manifest e permessi

La questione della privacy e dell'utilizzo delle informazioni personali è un argomento molto discusso negli ultimi tempi, soprattutto nell'ambito dell'utilizzo di dispositivi mobili e di applicazioni che devono essere sempre connesse ad Internet. Non tutti gli utenti, per esempio, si sentono sicuri sapendo che informazioni sulla loro posizione geografica vengano costantemente trasmesse in rete.

Il problema non è di facile risoluzione: perchè alcuni tipi di applicazione possano funzionare è inevitabile che l'utente debba fornire loro alcune sue informazioni personali. Allo stesso tempo, una volta che questi dati sono stati inviati via Internet è impossibile garantire effettivamente all'utente che non saranno conservati o utilizzati in qualche modo.

Android gestisce questo problema tramite l'Android Manifest: un file XML in cui ogni applicazione dichiara alcune sue proprietà, tra cui una serie di permessi che chiede all'utente per poter essere eseguita. Subito prima dell'installazione di un'applicazione, il sistema legge il suo Android Manifest ed avvisa l'utente dei permessi che questa richiede; l'utente ha così la possibilità di interrompere l'installazione se lo ritiene opportuno.

I tipi di permessi che è possibile chiedere sono tantissimi: si va dal monitoraggio della posizione geografica, alla possibilità di far partire chiamate o inviare SMS, all'accesso della rubrica dei contatti, e così via. Non tutti questi permessi riguardano le informazioni personali: è necessario chiedere il permesso anche per modificare alcune impostazioni globali del sistema, per usare la videocamera per ottenere immagini o video, per disabilitare temporaneamente lo spegnimento automatico dello schermo, e così via. E' necessario un permesso anche solo per fare in modo che l'applicazione invii o riceva dati via Internet.

Per assicurare la "correttezza" delle applicazioni, il sistema operativo è strutturato in modo tale che queste siano fisicamente impossibilitate ad usare

le feature per cui non hanno chiesto il permesso: questo dà all'utente la certezza di essere consapevole dei comportamenti del suo dispositivo.

3.3 *Activity* e *view*

I componenti principali di un'applicazione Android sono le *activity* e le *view*. Le *activity* contengono il codice Java che definisce il comportamento dell'applicazione, mentre le *view* sono file XML che ne definiscono l'aspetto grafico. Bisogna però notare che il confine tra questi due concetti non è sempre ben definito: è possibile modificare l'aspetto grafico dell'applicazione con del codice contenuto in una *activity*, o incorporare nelle *view* alcuni aspetti del suo comportamento.

Per dare un'idea di come questi due componenti interagiscono, si può dire che ogni schermata di un'applicazione è generata dall'accoppiamento di una *activity* ed una *view*. Quando l'utente passa ad una nuova schermata, il sistema carica una nuova *activity* che a sua volta carica la *view* da rappresentare sullo schermo.

La distinzione tra *activity* e *view* serve principalmente a semplificare la programmazione di applicazioni che devono funzionare su vari tipi di dispositivi con schermi molto diversi. E' infatti possibile associare ad ogni *activity* un numero arbitrario di *view*, ciascuna corrispondente ad alcune condizioni: sarà poi il sistema a decidere durante l'esecuzione quale *view* utilizzare.

L'esempio più semplice per mostrare l'efficacia di questo concetto è un'applicazione che modifica in tempo reale la disposizione degli elementi della schermata a seconda che l'utente tenga il telefono in posizione orizzontale e verticale: a livello di programmazione, questo significa che lo sviluppatore ha previsto due *view* per la *activity* corrispondente.

3.3.1 Caratteristiche delle *activity*

La divisione del codice in *activity* serve a dare al sistema la possibilità di caricare in memoria solo una parte dell'applicazione. Ogni *activity* viene infatti caricata solo nel momento in cui l'utente accede per la prima volta alla schermata corrispondente; quando si passa ad una nuova schermata, la *activity* viene "messa in pausa" in modo che possa essere facilmente recuperabile.

Quando il sistema esaurisce lo spazio disponibile, per esempio perchè l'utente ha aperto e poi messo in pausa troppe *activity*, comincia a rimuovere dalla memoria alcune delle *activity* in pausa. Se l'utente decide di riaprirle, il sistema deve caricarle di nuovo in memoria come se le stesse aprendo per

la prima volta. Questo comportamento per cui una *activity* viene caricata in memoria, portata in primo piano, messa in pausa, e così via è definito il "ciclo di vita" della *activity*.

A livello implementativo, ogni *activity* corrisponde ad una classe Java che erediti la superclasse *Activity* ed il suo ciclo di vita è definito da una serie di metodi che questa classe deve implementare.

Oltre ai metodi che definiscono il suo ciclo di vita, ogni *activity* contiene altri metodi che ne definiscono il comportamento in risposta agli eventi che accadono durante la sua esecuzione; per esempio, questi metodi potrebbero determinare le reazioni della *activity* alla pressione di un bottone, ad uno swipe sullo schermo, alla scoperta di una nuova rete wireless, alla ricezione di un messaggio e così via.

Metodi del ciclo di vita della *activity*

I metodi che compongono il ciclo di vita di una *activity* vengono chiamati automaticamente dal sistema quando è necessario. Tra questi, il più importante è forse *onCreate*, che viene chiamato nel momento in cui la *activity* viene creata e caricata in memoria; in questo metodo il programmatore deve occuparsi dell'inizializzazione dei dati e della *view* corrispondente alla *activity*.

Il metodo *onPause* viene chiamato nel momento in cui la *activity* non è più in cima allo stack del sistema, ma è ancora visibile; questo accade, per esempio, se viene caricata una nuova *activity* che occupa solo parte dello schermo. Quando la *activity* torna in primo piano, il sistema chiama il suo metodo *onResume*. All'interno di questi due metodi lo sviluppatore può bloccare e riattivare alcuni comportamenti della *activity* che non hanno senso se questa è in background.

Quando la *activity* scompare interamente dallo schermo, rimpiazzata da un'altra, viene invocato invece il metodo *onStop*; in questo metodo è possibile bloccare l'utilizzo di risorse dispendiose per il sistema, come il posizionamento GPS. Quando la *activity* verrà visualizzata di nuovo sarà il metodo *onStart* ad essere invocato, ed al suo interno si potranno riattivare le funzionalità bloccate in *onStop*.

E' importante per il programmatore tener conto del fatto che una *activity* bloccata con il metodo *onStop* potrebbe venir rimossa dalla memoria del sistema in qualsiasi momento; questo significa che è necessario salvare in memoria permanente gli aspetti importanti dello stato dell'esecuzione per evitare di perderli.

Perchè ciò sia possibile, il sistema chiama il metodo *onSaveInstanceState* prima di rimuovere dalla memoria l'istanza della *activity*; in questo metodo il programmatore può salvare le caratteristiche dell'esecuzione in un oggetto di tipo *Bundle* (un insieme di proprietà arbitrarie) che verrà passato al metodo *onCreate* nel momento in cui la *activity* dovesse essere ricaricata in memoria.

Questo significa anche che è necessario includere nel metodo *onCreate* delle istruzioni per ripristinare lo stato dell'esecuzione, nel caso in cui questo venisse invocato per ripristinare una *activity* che era stata messa in pausa e poi eliminata dalla memoria.

Metodi di risposta ad eventi

Escludendo ciò che riguarda il suo ciclo di vita, una *activity* generalmente contiene dei metodi che vengono chiamati in risposta a degli "eventi", che possono essere invocati da elementi dell'interfaccia o da altre classi.

Gli elementi dell'interfaccia in grado di generare eventi vengono detti "widget"; per esempio, un bottone genera un evento quando viene premuto, mentre un campo testuale genera degli eventi quando l'utente ne modifica il contenuto. Questi eventi provocano una reazione solo se la *activity* in esecuzione sta "ascoltando" il widget che li genera.

Perchè una *activity* ascolti un widget, è necessario fare in modo che questa implementi un'interfaccia appropriata. Per esempio, se si vuole che una *activity* ascolti un bottone questa dovrà implementare l'interfaccia *OnClickListener* ed il relativo metodo *onClick*. Fatto questo, il metodo *onClick* verrà invocato automaticamente ogni volta che il bottone sarà premuto; il programmatore deve quindi specificare al suo interno la reazione della *activity* alla pressione di quel pulsante.

Diversi tipi di widget richiedono diversi tipi di interfacce. Per esempio, è possibile fare in modo che una *activity* reagisca ogni volta che l'utente cambia il contenuto di una particolare casella di testo editabile usando l'interfaccia *TextWatcher*; che reagisca a swipe sullo schermo usando un'interfaccia della famiglia dei *GestureListener*; che reagisca al fatto che alcuni elementi vengano portati in primo piano usando *OnFocusChangeListener*; e così via.

E' importante notare che non sono solo i widget a generare eventi: alcuni servizi online, per esempio, trasmettono informazioni alle applicazioni generando eventi. Le *activity* che vogliono sfruttare questi servizi devono essere in grado di rispondere agli eventi che generano, proprio come se fossero causati da azioni dell'utente.

3.3.2 Caratteristiche delle *view*

Una *view* è una rappresentazione XML del layout di una schermata dell'applicazione; gli elementi XML rappresentano le parti di questo layout (bottoni, scritte, immagini e così via) mentre gli attributi rappresentano le loro proprietà ed il modo in cui si dispongono nello spazio.

Esistono anche degli "elementi contenitore" che hanno il compito di raggruppare gli elementi di base e stabilire la loro posizione; per esempio, una serie di elementi contenuti in un *LinearLayout* di orientamento verticale verranno disposti in colonna.

Esistono moltissime tecniche per stabilire la posizione degli elementi; in particolare, è importante definire un modo in cui lo "spazio in eccesso" deve essere distribuito in modo che la schermata occupi tutto lo schermo a prescindere dalla sua dimensione in pixel.

Esistono vari metodi per ottenere questo risultato, ma il principale è quello di assegnare un "peso" a ciascun elemento: in questo modo tutto lo spazio non utilizzato viene suddiviso tra i vari elementi in maniera proporzionale al loro peso.

Combinando questa tecnica con l'utilizzo di vari *LinearLayout* annidati, è possibile creare layout di qualsiasi forma che siano in grado di adattarsi dinamicamente alle dimensioni dello schermo.

Anche se la *view* è strutturata bene, in alcuni casi è comunque necessario cambiare totalmente la disposizione degli elementi, o gli elementi stessi, a seconda delle dimensioni dello schermo o del suo orientamento. Icone che funzionerebbero bene in uno schermo con risoluzione bassa, per esempio, potrebbero risultare sgradevoli in uno schermo più moderno, anche se si usasse un buon metodo di ripartizione dello spazio vuoto.

In questi casi è necessario creare diverse *view* per ogni *activity*, specificando nel loro nome le condizioni secondo cui il sistema deve scegliere quale caricare. Per esempio, un'applicazione potrebbe disporre le sue *view* in due directory chiamate *layout* e *layout-720dp*: il sistema caricherebbe automaticamente quelle contenute nella seconda directory se la risoluzione del lato lungo dello schermo fosse pari o superiore a 720 pixel, e quelle contenute nella prima altrimenti.

Allo stesso modo, è possibile creare delle *view* da caricare quando lo schermo viene messo in posizione orizzontale mettendole in una directory con un nome del tipo *layout-land*.

3.3.3 Interazioni tra *view* e *activity*

Il principio dietro alla divisione delle applicazioni in *activity* e *view* è la separazione della rappresentazione grafica del programma dalla logica del suo funzionamento. Di fatto, però, questa separazione non è completa: dall'interno di una *activity* è possibile accedere ad un oggetto Java corrispondente alla *view* attualmente rappresentata sullo schermo invocando il metodo *getView*.

Una volta ottenuto questo oggetto, lo sviluppatore può accedere a tutti gli elementi contenuti nella *view* ed invocare una serie di metodi su di essi; può cambiare il testo visualizzato in un campo testuale, lo sfondo di un bottone, ed anche proprietà direttamente collegate al layout come il peso di un elemento. E' anche possibile creare dinamicamente nuovi elementi o eliminarne alcuni dal layout.

Questo comportamento è una trasgressione del principio di suddivisione dei compiti tra *activity* e *view*, ma è di fatto necessario per permettere alle applicazioni di modificare il proprio layout in tempo reale in base alle azioni dell'utente o ad altri eventi esterni.

Per esempio, una schermata potrebbe contenere un bottone che fa partire una query in un database i cui risultati devono essere rappresentati all'interno di una *ListView* del layout. In questo caso la *activity*, nel metodo *onClick* corrispondente a quel bottone, deve recuperare la *ListView* in questione e modificarla inserendovi i dati da visualizzare.

Altri esempi potrebbero essere situazioni in cui la pressione di un bottone fa comparire o scomparire altri bottoni, o in cui la ricezione di risposte a query via internet permette di visualizzare nuovi elementi.

3.4 I *fragment*

La semplice programmazione per *activity* in alcuni casi può rivelarsi un po' limitante. Esistono infatti molte applicazioni che dedicano una parte dello schermo ad elementi fissi, presenti in tutte le schermate; questi elementi dovrebbero essere replicati in tutte le *activity* e le *view*.

Per evitare questa inutile replicazione di codice e rendere la programmazione più intuitiva, Android ha introdotto il concetto di *fragment*. Un *fragment* è essenzialmente una mini-*activity* inserita all'interno di un elemento di una *view*; durante l'esecuzione, il comportamento di quel particolare elemento non è dettato dalla *activity* attualmente in primo piano ma dal codice del *fragment* stesso.

Una volta definito il layout ed il comportamento di un *fragment*, questo può essere incluso nei layout di diverse *activity*. Inoltre, ogni *activity*

può cambiare dinamicamente i *fragment* che la compongono facendo delle richieste al *FragmentManager*.

L'esistenza dei *fragment* apre una nuova questione nella programmazione Android: è possibile programmare un'applicazione come un insieme di *activity*, come un'unica *activity* composta da vari *fragment*, o come una via di mezzo.

Per esempio, un tipico utilizzo dei *fragment* è l'implementazione di una barra di navigazione dell'applicazione. Sarebbe possibile implementare la barra di navigazione come un *fragment* da includere nelle *view* di tutte le *activity*, oppure implementare l'intera applicazione come un'unica *activity* contenente il menu di navigazione ed un elemento in cui caricare le varie schermate, rappresentate come *fragment*, durante l'esecuzione.

Non esiste una scelta giusta: dipende tutto da quanto sia il comportamento comune tra le varie schermate e dalle preferenze personali del programmatore.

3.5 Il database

Nella programmazione Android ci sono vari metodi per gestire la rappresentazioni di dati su memoria permanente: ogni applicazione, per esempio, può gestire un certo numero di "preferenze" che rimangono salvate tra un'esecuzione e l'altra. E' anche possibile intervenire direttamente sul file system del dispositivo, leggendo e scrivendo file di vari tipi. Un'altra soluzione è utilizzare un database SQLite.

Android contiene infatti un'implementazione di SQLite, e qualsiasi applicazione può facilmente creare un suo piccolo database persistente. Le librerie Android contengono diverse classi che permettono agli sviluppatori di eseguire facilmente query SQL per la creazione, l'interrogazione e la modifica di un database.

Combinando le funzioni di query del database e di lettura da file, è anche possibile creare dei file testuali con una serie di query SQL da fare eseguire all'applicazione. Questa tecnica è molto comoda soprattutto in fase di creazione o modifica dei database, dato che lo sviluppatore dell'applicazione e chi fornisce i dati non sempre sono la stessa persona.

Capitolo 4

Progettazione

Il mio compito principale all'interno del progetto è stato lo sviluppo di una versione dimostrativa della piattaforma che implementasse almeno le funzionalità di base, in modo che potesse essere usata come "proof of concept".

Questa demo si concentra sul primo livello di navigazione della piattaforma: comprende un'interfaccia grafica, la funzione di ricerca dei punti di interesse e le modalità di navigazione su mappa e per immagini. L'unica feature del primo livello non inclusa nella demo è la navigazione tramite realtà aumentata, che richiederebbe uno studio più approfondito delle tecnologie relative.

Oltre a queste funzioni la demo contiene anche una schermata in cui l'utente può consultare le mappe degli interni dei padiglioni; questa funzione è stata pensata come "tappabuchi" in attesa di un futuro sviluppo della navigazione indoor. L'applicazione, inoltre, seleziona automaticamente la mappa da mostrare all'utente all'apertura della schermata in base al padiglione più vicino alla sua posizione attuale.

4.1 Sistema operativo

La prima scelta progettuale effettuata nello sviluppo della piattaforma è stata quella di implementarla sotto forma di applicazione per sistemi operativi Android. Questa decisione è stata dettata principalmente da due fattori: Android è attualmente il sistema operativo di gran lunga più diffuso sui dispositivi mobili di tutto il mondo; inoltre, sono disponibili gratuitamente in rete una quantità enorme di librerie, servizi e documentazione per lo sviluppo su di esso.

Tra gli obiettivi a lungo termine del progetto potrebbe essere inserita anche un'estensione della piattaforma ai sistemi operativi Apple, principali concorrenti di Android. Vista la complessità maggiore dello sviluppo su questi sistemi, però, la questione è stata rimandata al futuro.

Il fatto stesso di sviluppare la piattaforma come applicazione Android mette a disposizione degli sviluppatori alcuni strumenti interessanti; in particolare, una volta completato lo sviluppo è possibile inserire l'applicazione nel Google Play Store.

Questo rende molto semplice la fase di deployment: il Play Store è il metodo standard di ottenere nuove applicazioni su dispositivi Android, e quello di gran lunga più semplice. Si può quindi supporre che basti segnalare ai potenziali utenti il nome dell'applicazione e la sua presenza sul Play Store per dare a chiunque la possibilità di trovarla e scaricarla.

Inoltre, una volta installata l'applicazione su un dispositivo, questo si occupa di scaricare ed installare automaticamente eventuali aggiornamenti nel momento in cui questi vengono resi disponibili online. Questa funzione dei sistemi Android rende molto semplice la modifica o l'aggiunta di feature anche cruciali all'applicazione anche dopo il deployment iniziale.

4.2 Ricerca della destinazione

La sezione di ricerca della destinazione è quella dalla struttura più semplice: alcune schermate permettono all'utente di selezionare un punto di interesse da una lista contenuta all'interno di un database. Nello sviluppo di questa parte della piattaforma, la scelta più importante è dove posizionare il database.

Ci sono fondamentalmente due possibilità: fare in modo l'applicazione crei e gestisca un piccolo database su ogni dispositivo che la esegue oppure creare un unico database all'interno di un server installato nell'area del Sant'Orsola. Entrambe le scelte hanno dei vantaggi e degli svantaggi.

Fare in modo che l'applicazione gestisca il database in locale è sicuramente il sistema migliore dal punto di vista dell'efficienza: le query verrebbero effettuate in breve tempo e senza bisogno di connettersi alla rete.

Il lato negativo di questa opzione sta nel fatto che nel caso di modifica ai dati (per esempio per via dello spostamento di un reparto) sarebbe necessario modificare il database all'interno di tutti i dispositivi che hanno installato l'applicazione. Android gestisce automaticamente gli aggiornamenti e l'applicazione ha comunque bisogno di essere connessa ad Internet per

avviarsi, quindi il rischio che gli utenti si ritrovino ad usare una versione non aggiornata è molto basso; il processo rimane comunque un po' macchinoso.

Mantenere tutti i dati in un database centralizzato offre il vantaggio di renderne molto semplice il mantenimento e di ridurre la necessità di aggiornamenti dell'applicazione. D'altro canto, la piattaforma avrebbe bisogno di effettuare tutte le query via Internet, aumentando sensibilmente il tempo necessario per ciascuna di esse.

Questo diventa rilevante soprattutto durante la ricerca del punto di interesse: in questa fase l'applicazione deve effettuare un gran numero di query. Prima di implementare una soluzione di questo tipo sarebbe quindi necessario valutare se i benefici che ne verrebbero tratti superano i difetti; questo non sarebbe possibile senza una prova sul campo, in cui misurare i disagi effettivi provocati all'esperienza degli utenti dalla minor velocità delle query.

E' anche possibile pensare ad una "via di mezzo" in cui il database è situato sul server e l'applicazione, ogni volta che viene avviata, fa una query in cui richiede tutti i dati e li salva in locale. Tutte le query successive opererebbero sulla copia in locale dei dati e sarebbero quindi efficienti, e non sarebbe necessario diffondere le modifiche al database tramite aggiornamenti.

Attualmente la demo mantiene il database in locale, instanziandolo di nuovo ogni volta che viene avviata. Questa soluzione è stata scelta solo per semplicità di implementazione e comodità in fase di testing; non sono state ancora compiute vere e proprie scelte progettuali in questo senso.

Struttura del database

La struttura del database usato nella demo è estremamente semplice: è formato da quattro tabelle contenenti la lista dei punti di interesse divisi per tipo, più una quinta tabella che serve a specificare quali reparti sono contenuti in ogni padiglione.

Le tabelle "trasporti" e "servizi" contengono un campo in cui è specificato il tipo del mezzo di trasporto o del servizio in questione. Per esempio, un mezzo di trasporto potrebbe essere la fermata di un autobus o un parcheggio, mentre un servizio potrebbe essere un punto di accettazione o una sala studio.

Per dare un esempio di utilizzo del database: se si volesse estrarre una lista dei reparti contenuti al secondo piano del padiglione 15, bisognerebbe usare la query "SELECT reparti.nome FROM reparti, repartipadiglioni WHERE reparti.id = repartipadiglioni.reparto AND repartipadiglioni.padiglione = 15 AND repartipadiglioni.piano = 2".

Se invece si volesse ottenere la lista di tutte le fermate degli autobus, basterebbe usare la query "SELECT * FROM trasporto WHERE tipo=fermata".

4.3 Navigazione su mappa

I requisiti di questa sezione della piattaforma, se ci si limita al primo livello di navigazione, sono tre: avere una mappatura dell'area ospedaliera, essere in grado di rappresentarla graficamente sullo schermo e poter calcolare e mostrare sulla mappa il percorso più breve tra due punti.

Implementare da zero queste tre funzionalità sarebbe un compito molto dispendioso in termini di tempo e risorse; si è quindi valutata in primo luogo la possibilità di ottenerle sfruttando delle librerie esterne.

In generale, il tradeoff tra implementare da zero una funzionalità e ottenerla da una libreria mette semplicità di implementazione e risparmio contro la versatilità del risultato. Implementando da zero una feature è infatti possibile aggiungerle comportamenti particolari non sempre offerti da librerie e servizi che la implementano in modo "standard".

La scelta dev'essere quindi effettuata in base a quanti comportamenti "particolari" si richiedano a ciascuna di queste feature e a quanto questi comportamenti siano importanti per il buon funzionamento dell'applicazione.

Questo tradeoff si è presentato diverse volte durante l'implementazione della sezione di navigazione su mappa della piattaforma.

4.3.1 Scelta della mappatura

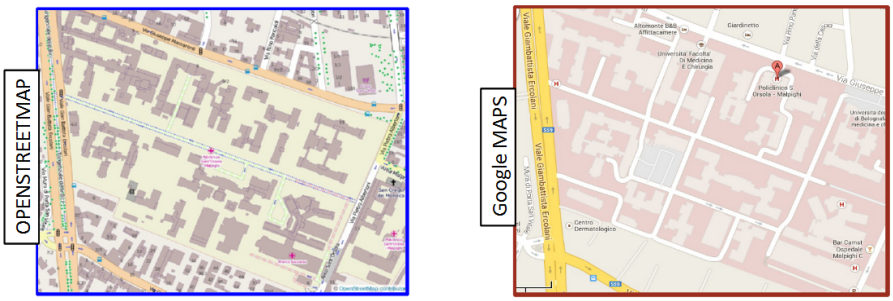


Figura 4.1: Mappature di Google Maps ed OpenStreetMap

Costruire da zero una mappatura dell'intera area ospedaliera sarebbe un compito costoso e superfluo, in quanto esistono già diverse mappature disponibili pubblicamente e facilmente integrabili in applicazioni di navigazione. Per quanto riguarda la mappatura, la questione è stata quindi semplicemente a quale di queste mappature affidarsi.

Tra quelle prese in considerazione, le due opzioni più convincenti sono state Google Maps ed OpenStreetMap. Entrambi sono progetti molto famo-

si, ed entrambi forniscono una mappatura sufficientemente precisa dell'area ospedaliera del Policlinico.

Le librerie di Google sono affidabili, molto ben documentate, relativamente versatili e, soprattutto, integrano automaticamente la mappatura e la visualizzazione grafica della mappa. Dal punto di vista della facilità di sviluppo e della qualità estetica del risultato finale, questa è l'opzione decisamente migliore.

Il rovescio della medaglia è il fatto che la mappatura è inaccessibile agli sviluppatori e non può quindi essere modificata o estesa in nessun modo. Questo diventa un problema perchè le mappa di Google, benchè rappresentino nei dettagli le strade asfaltate, non contengono alcune strade esclusivamente pedonali che vengono regolarmente utilizzate dai frequentatori dell'ospedale.

Inoltre, i termini di utilizzo delle API di Google Maps impongono un limite piuttosto ambiguo sulla possibilità di essere usate per lo sviluppo di "applicazioni di navigazione in tempo reale". Questo limite non si applica alla semplice ricerca e visualizzazione di un percorso sulla mappa, ma potrebbe essere interpretato come un divieto di aggiungere all'applicazione funzioni che ricalcolino il percorso quando l'utente se ne allontana.

Le mappe di OpenStreetMap, al contrario di quelle di Google, sono completamente aperte e basate sul crowdsourcing: chiunque ha la possibilità di farvi delle correzioni o delle aggiunte, utilizzando alcuni tool molto semplici e disponibili gratuitamente.

Questo risolve il problema dell'assenza di alcune strade dalla mappatura dell'area ospedaliera, e rende questa scelta molto adatta anche all'implementazione del terzo livello della piattaforma: sarebbe facile aggiungere alle mappe alcuni punti marcati esplicitamente come "punti difficili" per utenti con vari tipi di disabilità.

Il rovescio della medaglia è un aumento della complessità di implementazione: il progetto OpenStreetMap mette a disposizione degli utenti alcuni tool che permettono di rappresentare graficamente delle mappe, ma nessuno di questi raggiunge la completezza e la facilità di utilizzo delle API di Google.

Dopo aver valutato i pro e i contro, per la demo è stato deciso di usare le API e le mappature di Google Maps, in quanto l'obiettivo primario era sviluppare nel minor tempo possibile un'applicazione dimostrativa funzionante.

Durante il proseguimento dello sviluppo, però, si valuterà la possibilità di spendere le risorse necessarie per passare alle mappature di OpenStreetMap o ad altre mappature aperte e modificabili.

4.3.2 Visualizzazione della mappa

La rappresentazione grafica di una mappa è un compito molto complesso e pieno di problemi, soprattutto se si tiene conto di tutti gli elementi necessari per fornire all'utente un'interfaccia al passo coi tempi (controllo dello zoom, spostamento tramite swipe, e così via). Per questo motivo è stato scelto di automatizzare il più possibile questa fase sfruttando le API di Google Maps, decisione che ha condizionato anche la scelta della mappatura.

Queste API hanno due possibili modalità di utilizzo, ed anche in questo caso ciascuna ha i suoi pro ed i suoi contro. E' possibile sfruttare collegamenti tra l'applicazione in fase di sviluppo e l'applicazione Maps, oppure inserire delle mappe direttamente nelle schermate della propria applicazione.

Usando il primo approccio sarebbe possibile, per esempio, inserire all'interno del layout dell'applicazione un bottone che, quando premuto, apra l'applicazione Maps del dispositivo centrando la mappa nel punto desiderato.

Usando il secondo approccio si inserisce un elemento contenente la mappa all'interno di una *view* della propria applicazione; in questo modo si evita di costringere l'utente a fare avanti-indietro tra due applicazioni diverse per completare un singolo task, cosa che risulterebbe scomoda e controintuitiva.

La scelta è meno semplice di quello che sembra: il secondo approccio è chiaramente migliore in termini di usabilità dell'applicazione, ma le API fornite da Google per implementarlo non contengono tutte le funzionalità incluse nell'applicazione Maps.

In particolare, l'applicazione Maps contiene già una funzionalità di navigazione molto ben fatta e facilmente attivabile dall'esterno. E' sufficiente passare all'applicazione durante la sua apertura le coordinate della destinazione e Maps si occupa di trovare il percorso, visualizzarlo e modificarlo automaticamente in base agli spostamenti dell'utente.

Se si vogliono replicare queste funzionalità all'interno dell'applicazione di navigazione ospedaliera, è necessario implementarle da zero.

Nonostante questo, la scelta è ricaduta sul secondo approccio: la schermata di visualizzazione della mappa è uno dei punti cardine dell'interfaccia, ed è la schermata in cui l'utente trascorrerà probabilmente la maggior parte del tempo di utilizzo del programma. Spostare questa schermata in un'altra applicazione peggiorerebbe di molto la semplicità d'uso della piattaforma.

4.3.3 Calcolo del percorso

Avendo deciso di non sfruttare l'applicazione Maps, è stato necessario trovare un metodo per implementare il calcolo del percorso tra la posizione

dell'utente e la sua destinazione. Anche in questo caso la scelta è ricaduta su uno dei servizi offerti da Google: il Google Directions Service.

La API di questo servizio è progettata in modo da essere indipendente dalla piattaforma che la utilizza, può essere attivata tramite una semplice richiesta HTTP e restituisce il risultato in un formato testuale.

Il principale punto a favore di questa scelta, oltre alla semplicità di utilizzo, è che i risultati prodotti da questo servizio sono sicuramente compatibili con la mappatura di Google utilizzata nella demo.

Nonostante questo, non esistono funzioni automatiche che visualizzino il percorso su una mappa creata tramite le API di Google Maps: è necessario implementare a parte un sistema che legga l'output del Directions Service e lo trasformi in un formato compatibile con queste API.

Inoltre, è necessario implementare da zero anche un eventuale sistema di auto-correzione del percorso: l'applicazione deve tenere d'occhio la posizione dell'utente e, qualora questo dovesse allontanarsi dalla strada giusta, fare una nuova richiesta al Directions Service.

Un problema relativo a questo servizio di Google è il fatto che la licenza gratuita metta un limite al numero di richieste giornaliere ed al numero di richieste al secondo. Questo limite riguarda tutte le richieste effettuate con la stessa API key, ovvero tutte quelle effettuate dallo stesso programma, anche se da dispositivi diversi.

I limiti non sono tali da dare problemi in fase di testing ma potrebbero rendere di fatto inutile l'applicazione se il numero di utenti, e quindi di richieste, dovesse salire troppo. Per superare questo problema è sufficiente acquistare una licenza commerciale per l'utilizzo del servizio.

4.4 Navigazione per immagini

La sezione di navigazione per immagini è implementata interamente all'interno dell'applicazione, non esistendo delle librerie pubbliche che forniscano questo tipo di funzionalità.

Prima di scendere nei dettagli dell'algoritmo usato, a livello di progettazione questa sezione pone un semplice problema: è necessario disporre di una grossa quantità di immagini che ritraggano tutte le strade dell'area ospedaliera.

In questo caso, a differenza di quando si parlava del database dei punti di interesse, la soluzione più logica è mantenere tutte le immagini all'interno di un server e fare in modo che l'applicazione scarichi solo quelle di cui ha bisogno. Salvare tutte le immagini su ogni singolo dispositivo, infatti,

aumenterebbe di gran lunga le dimensioni dell'applicazione, e di conseguenza il tempo necessario per scaricarla.

Questo significa che anche se si decidesse di non usare il server per i punti di interesse, sarebbe in ogni caso necessario predisporne uno per supportare questa modalità di navigazione.

Nonostante queste considerazioni, attualmente non esiste alcun server dedicato a questa piattaforma; di conseguenza, la demo mantiene le immagini necessarie a questa modalità nella memoria del dispositivo. Questo non crea particolari problemi in quanto la versione dimostrativa contiene una quantità molto limitata di immagini, quelle necessarie a rappresentare un paio di percorsi usati come esempio.

4.4.1 Scorrimento della sequenza di immagini

Una scelta progettuale di minore entità ma comunque importante per l'usabilità dell'applicazione è la decisione delle modalità secondo cui l'utente può scorrere le immagini della sequenza durante lo spostamento.

Idealmente l'applicazione dovrebbe essere in grado di scorrere la sequenza automaticamente, monitorando la posizione dell'utente e reagendo ai suoi spostamenti; durante le prime fasi dello sviluppo ho però ritenuto opportuno implementare un algoritmo più semplice di scorrimento manuale.

Attualmente l'applicazione utilizza un metodo misto: nel momento in cui l'utente accede alla schermata di navigazione per immagini gli viene mostrata l'immagine del punto più vicino a quello in cui lui si trova attualmente. Fatto questo, l'applicazione non interviene più e l'utente deve scorrere automaticamente la sequenza.

In questo modo un utente potrebbe, per esempio, seguire una parte del percorso usando la navigazione su mappa e poi passare a quella per immagini, trovandosi automaticamente al punto giusto della sequenza. Dopodichè, sarebbe l'utente stesso a scorrere all'immagine successiva nel momento in cui raggiungerebbe il punto mostrato in quella attuale.

Ho implementato questo metodo di scorrimento perchè ho ritenuto che un maggior controllo da parte dell'utente possa aumentare l'usabilità di questa feature dell'applicazione; durante lo sviluppo della piattaforma sarebbe comunque molto semplice aggiungere una modalità di scorrimento completamente automatico della sequenza di immagini.

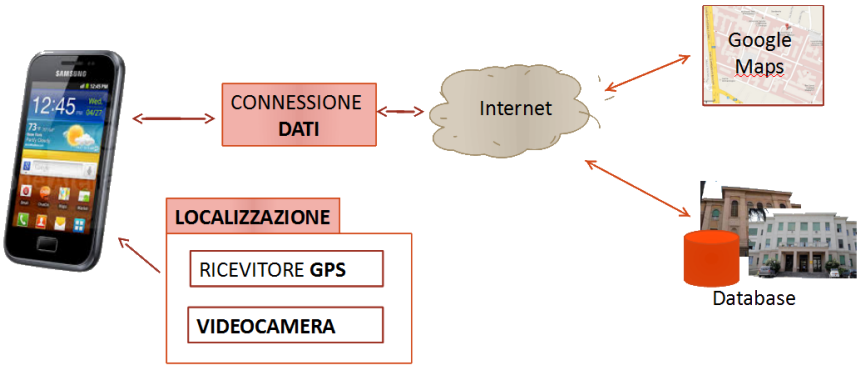


Figura 4.2: Architettura della piattaforma

4.5 Architettura dell'applicazione

La piattaforma è implementata come un'applicazione per sistemi operativi Android chiamata "Navigatore Sant'Orsola". Questa applicazione funziona su tutte le versioni di Android a partire dalla 4.0 e richiede i permessi di invio dati via Internet, di scrittura sulla memoria permanente del dispositivo e di monitoraggio della posizione dell'utente via GPS e Wi-Fi. Inoltre, l'applicazione ha bisogno che il dispositivo sia costantemente connesso ad Internet durante il suo utilizzo.

Il funzionamento dell'applicazione è descritto in Figura 4.2.

L'applicazione riceve dati sulla posizione dell'utente dal modulo GPS del dispositivo o, nel caso della navigazione con *augmented reality*, dalla videocamera; inoltre, riceve via Internet dal servizio di Google Maps i dati necessari per la rappresentazione della mappa.

Quando l'utente sceglie una destinazione, l'applicazione usa la connessione del dispositivo per mandare una richiesta al servizio Google Directions; la risposta viene poi usata per visualizzare il percorso.

Se l'utente sceglie di utilizzare la navigazione per immagini il dispositivo calcola la sequenza di immagini da visualizzare e le scarica dal server del Sant'Orsola, di nuovo usando la propria connessione ad Internet.

La versione demo implementa tutti i comportamenti appena descritti ad eccezione della comunicazione con il database: punti di interesse ed immagini vengono mantenuti all'interno del dispositivo.

Capitolo 5

Implementazione

Durante l'implementazione della versione dimostrativa della piattaforma ho cercato di mantenermi il più fedele possibile alle scelte progettuali descritte nel capitolo precedente, e di implementare nella maniera più completa possibile il ristretto numero di feature scelte per questo progetto.

La cosa principale che manca alla demo per potersi considerare una prima "versione alfa" della piattaforma sono i dati: attualmente la demo opera su un piccolo database contenente la posizione di una ventina tra padiglioni, reparti ed altri punti di interesse. Anche per quanto riguarda la navigazione ad immagini, l'applicazione contiene solo una decina di fotografie. Inoltre, tutte le mappe dei padiglioni sono al momento dei *placeholder*.

5.1 Interfaccia della demo

La prima parte della demo ad essere implementata è stata l'interfaccia grafica, uno dei punti cruciali dello sviluppo: l'applicazione contiene diverse feature e deve essere utilizzabile dal maggior numero possibile di persone, anche da chi non abbia esperienza nell'uso di dispositivi mobili o di applicazioni di navigazione.

Questo perchè l'insieme degli utenti a cui la piattaforma è mirata è composto in gran parte dai pazienti dell'ospedale; si può ragionevolmente assumere che gran parte di loro non faccia un uso "esperto" del proprio dispositivo e che molti di loro abbiano bisogno di usare la piattaforma solo una volta per raggiungere un singolo punto di interesse.

Fatte queste considerazioni, ho deciso di strutturare l'interfaccia della demo seguendo lo schema "classico" delle applicazioni Android, in modo da dare all'utente una sensazione di familiarità.

L'applicazione è formata da tre "schermate principali" tra cui l'utente può navigare usando dei tab posti in cima allo schermo: una schermata di navigazione, una di ricerca della destinazione ed una di consultazione delle informazioni sull'interno dei padiglioni.

Il tipo di navigazione si può cambiare nel menu di opzioni dell'applicazione; a seconda di quello attualmente selezionato, il tab di navigazione porterà alla schermata di navigazione su mappa, a quella di navigazione per immagini o a quella di navigazione per realtà aumentata (non implementata nella demo).

Il tab di ricerca porta ad una schermata in cui è possibile selezionare il tipo di ricerca che si vuole effettuare: la demo comprende le opzioni di ricerca per padiglione, per reparto, per mezzo di trasporto o per servizio. La schermata di ricerca contiene anche i bottoni della ricerca per medico e per esame, ma queste due feature non sono implementate nella demo.

Una volta scelto il tipo di ricerca, l'utente viene portato ad una nuova schermata in cui può scegliere il punto di interesse desiderato. Quando anche questa scelta sarà stata effettuata, l'applicazione andrà automaticamente alla schermata di navigazione.

La schermata informativa sui padiglioni al momento contiene solo una mappa del padiglione selezionato. L'applicazione cambia automaticamente il padiglione visualizzato in questa sezione quando rileva il fatto che l'utente si sia avvicinato ad uno di essi; è comunque possibile selezionare manualmente il padiglione di cui si vuole vedere la mappa.

Attualmente tutte le immagini che dovrebbero contenere le mappe dei padiglioni sono dei placeholder, dato che non mi è stato possibile reperire le mappe vere e proprie in formato digitale.

5.2 La *activity* principale

Considerato il tipo di interfaccia da implementare, ho deciso di strutturare l'applicazione come un'unica *activity* composta da molti *fragment*.

Il layout della *activity* comprende la barra di navigazione dei tab ed un *FrameLayout* (un elemento vuoto) che occupa tutto il resto dello schermo, chiamato *fragment container*; durante l'esecuzione, la *activity* riempie questo elemento contenitore con il *fragment* corrispondente alla schermata in cui l'utente si trova.

Sarebbe stato possibile implementare ogni schermata come una *activity* a parte, usando i tab per passare dall'una all'altra, ma ho ritenuto che una

struttura a *fragment* fosse più adeguata per implementare questo tipo di applicazione.

Questa architettura significa che la *activity* principale, chiamata appunto *Main Activity*, deve gestire tutte le informazioni che rimangono rilevanti anche durante i cambi di schermata; per esempio, deve passare alla schermata di navigazione la destinazione selezionata nella schermata di ricerca.

Per poter comunicare con i *fragment*, la *activity* implementa alcune interfacce definite all'interno dei *fragment* stessi. In questo modo, all'interno del codice dei *fragment* sarà possibile invocare i metodi contenuti in queste interfacce per comunicare con la *activity*.

Nell'esempio di prima in cui la schermata di ricerca comunica la destinazione a quella di navigazione, all'interno del *fragment* di ricerca viene invocato un metodo della *activity* per comunicarle il punto di interesse selezionato. La *activity*, in risposta a questa invocazione, calcola il percorso verso la destinazione ed invoca a sua volta un metodo del *fragment* di navigazione per visualizzarlo.

Oltre alla gestione dei *fragment*, la *Main Activity* deve contenere anche il codice per tutte le attività che vengono svolte "dietro le quinte": l'inizializzazione dell'applicazione e di tutti i servizi, la gestione del database, la ricerca del percorso e la sua elaborazione, e così via.

Questo compito comprende anche la gestione del ciclo di vita della *activity* stessa e la comunicazione con entità esterne, come i servizi Google Directions e Google Location.

5.2.1 Inizializzazione dell'applicazione

L'inizializzazione dell'applicazione avviene all'inizio del ciclo di vita della *Main Activity*, ovvero nel suo metodo *onCreate*.

All'interno di questo metodo vengono instanziati tutti i *fragment* che compongono l'applicazione; in questo modo gli oggetti saranno già pronti nel momento in cui l'utente dovesse cambiare schermata. Fatto questo, viene fatta una richiesta al *Fragment Manager* per caricare nel layout della *activity* il *fragment* corrispondente alla prima schermata, ovvero quella di navigazione su mappa.

La seconda parte dell'inizializzazione è quella della barra contenente i tab che permettono di navigare tra la schermate dell'applicazione. Questa è rappresentata da un oggetto di tipo *Action Bar*, mentre i singoli bottoni sono oggetti di tipo *Tab*.

Una volta creata la *Action Bar*, è necessario che la *activity* implementi l'interfaccia *Tab Listener*; in questo modo sarà in grado di reagire agli eventi generati dalla pressione dei tab da parte dell'utente. I due metodi principali di questa interfaccia sono *onTabSelected* e *onTabReselected*, che permettono alla *activity* di distinguere il caso in cui l'utente scelga un tab diverso da quello in cui si trova da quello in cui clicchi di nuovo sul tab già selezionato.

A questo punto, l'inizializzazione dell'interfaccia grafica è completa. Tutto ciò che rimane da fare è inizializzare il database ed i vari servizi online di cui l'applicazione fa uso.

5.2.2 Navigazione tra le schermate

Gran parte dei metodi della *Main Activity* servono ad implementare lo spostamento tra le schermate dell'applicazione. I cambiamenti di schermata avvengono in risposta ad alcuni eventi e consistono in una richiesta al *Fragment Manager* in cui viene cambiato il *fragment* visualizzato nel layout della *activity*.

Gli eventi che producono cambiamenti di schermata si possono dividere in quattro tipi: la pressione di un tab, il cambiamento del tipo di navigazione, l'utilizzo di widget presenti in alcune schermate e la pressione del tasto "indietro".

I tab servono a spostarsi tra le tre "sezioni principali" dell'applicazione: la navigazione, la ricerca dei punti di interesse e la visualizzazione delle mappe dei padiglioni.

Se l'utente seleziona il tab di una sezione diversa da quella in cui si trova, l'applicazione lo porta subito alla schermata principale di quella sezione. Inoltre, se l'utente si trova in una delle "sotto-schermate" di ricerca (per esempio quella di ricerca per padiglioni) e preme il tab "cerca", viene riportato alla schermata principale della sezione di ricerca.

Il cambiamento del tipo di navigazione genera un cambiamento di schermata solo se l'utente si trova nella schermata di navigazione; in caso contrario, la *activity* memorizza il nuovo tipo di navigazione in modo da poter aprire la schermata giusta quando l'utente tornerà in quella sezione.

Il fatto che alcuni widget possano produrre un cambiamento di schermata è rilevante perchè significa che la *Main Activity* deve rimanere in ascolto anche degli eventi generati dai *fragment* che rappresentano le singole schermate.

Questo significa che all'interno dei *fragment* è possibile chiamare la *activity* per segnalarle di cambiare schermata. Per esempio, se l'utente si trova

nella sezione di ricerca e seleziona una destinazione, il *fragment* notifica questo evento alla *activity* che porta immediatamente l'utente nella sezione di navigazione.

L'utilizzo del tasto "indietro" è abbastanza limitato in questa applicazione: le tre sezioni principali sono allo stesso livello, e non è quindi possibile usarlo per spostarsi tra di esse.

L'unico caso in cui il tasto "indietro" ha effetto è quando l'utente si trova in una delle sotto-schermate di ricerca e vuole tornare a quella di selezione del tipo di ricerca.

5.2.3 Comunicazione con il servizio Location

I dispositivi mobili possiedono una serie di tecniche per tenere d'occhio la propria posizione geografica; oltre al GPS, per esempio, possono sfruttare alcune informazioni relative alla rete Wi-Fi a cui sono connessi.

Se non si hanno richieste particolari, però, il modo più semplice di monitorare la posizione dell'utente è utilizzare il servizio Google Location. Uno dei compiti della *Main Activity* è proprio quello di comunicare con questo servizio, attivandolo durante l'inizializzazione ed ascoltando le sue notifiche.

A questo scopo, durante l'inizializzazione del sistema la *activity* costruisce un oggetto di tipo *Location Client* ed effettua una richiesta di connessione ai Google Play Services. Una volta ricevuta risposta positiva alla richiesta di connessione, l'applicazione crea una *Location Request*. All'interno di questo oggetto è possibile specificare alcuni parametri della richiesta da effettuare al servizio. I due più importanti sono la "priorità" e l'"intervallo".

Una priorità alta fornisce informazioni più accurate ma consuma più energia elettrica, mentre una priorità bassa punterà al risparmio del livello di batteria a scapito della quantità di informazioni. L'intervallo è invece il lasso di tempo che il LocationClient lascerà trascorrere dopo la ricezione di un aggiornamento sulla propria posizione prima di chiederne un altro. Anche questo parametro ha una forte influenza sul consumo di energia dell'applicazione.

In questa particolare applicazione, la richiesta che viene effettuata dà la priorità alla precisione del risultato piuttosto che al risparmio energetico.

Una volta effettuata la richiesta, la *activity* riceve una chiamata del metodo *onLocationChanged* ogni volta che il servizio rileva uno spostamento del dispositivo. Questo significa che la *activity* deve implementare l'interfaccia *LocationListener* che contiene questo metodo.

Durante il ciclo di vita della *activity*, la connessione al servizio Location viene interrotta nel metodo *onStop* e riattivata nel metodo *onStart*. Que-

sto permette all'applicazione di interrompere il monitoraggio della posizione, attività dispendiosa in termini di energia, quando passa in secondo piano.

5.2.4 Richieste al servizio Directions

Per ottenere un percorso dal Google Directions Service è necessario conoscere latitudine e longitudine dei punti di partenza e di arrivo ed includere questi dati in una richiesta HTTP ad un URL predefinito.

In un'applicazione di navigazione, il punto di partenza corrisponde alla posizione attuale dell'utente; le sue coordinate sono quelle ottenute tramite il servizio Location descritto nella sezione precedente.

Il punto di arrivo è invece uno dei punti di interesse contenuti nel database dell'applicazione; le sue coordinate si possono quindi recuperare con una query al database stesso.

Formato della richiesta

Una volta ottenuti i punti di partenza e di arrivo, è necessario effettuare una richiesta HTTP ad un URL che segua il formato *"http://maps.googleapis.com/maps/api/directions/output?parameters"* , dove "output" rappresenta il tipo di output desiderato (JSON o XML) e "parameters" è una serie di parametri separati dal carattere "&".

Tutte le richieste devono comprendere obbligatoriamente i due parametri "origin" e "destination", ovvero il punto di partenza e di arrivo del percorso. E' possibile rappresentare questi punti sia specificando la loro longitudine e latitudine che tramite una stringa che rappresenti il loro indirizzo.

E' inoltre possibile inserire nelle richieste alcuni parametri opzionali. Il parametro "mode", per esempio, permette di specificare il tipo di navigazione richiesto: a piedi, in bicicletta, in automobile o usando i mezzi di trasporto pubblici. Altri parametri permettono di evitare strade a pagamento, di chiedere strade alternative, di scegliere il formato di output e così via.

Usando una serie di parametri "waypoint" è anche possibile specificare una serie di punti intermedi per cui il percorso deve passare.

Formato dell'output

L'output prodotto da una richiesta al Directions Service può essere in formato JSON o XML, e contiene una lunga serie di informazioni sul percorso calcolato dal servizio. A differenza delle richieste, nell'output tutti i punti sono rappresentati solo dalla loro longitudine e latitudine.

Il servizio divide ogni percorso in una serie di segmenti; per ciascuno di essi specifica le due estremità, la lunghezza, il tempo di percorrenza ed alcune altre informazioni opzionali. Infine, l'output contiene lunghezza e tempo di percorrenza dell'intero percorso.

Perchè questo output sia utilizzabile dall'applicazione è necessaria una classe che faccia il parsing del file JSON ed estragga latitudine e longitudine di tutti i punti intermedi del percorso. Fortunatamente il sito Android Developers mette a disposizione degli sviluppatori una classe, chiamata *JSONParser*, adibita a svolgere proprio questo compito: il suo metodo *decodePoly* prende come input una stringa che rappresenti il risultato in JSON di una richiesta al Directions Service e restituisce una lista di oggetti *LatLng* che rappresentano i punti del percorso.

Richieste asincrone

Dato che una richiesta al Directions Service deve attendere generalmente almeno un paio di secondi prima di ricevere una risposta, il procedimento descritto in questa sezione è stato implementato come *task asincrono* della *activity* principale.

Questo significa che la classe *Main Activity* contiene una piccola classe interna chiamata *Directions Fetcher Task*, che eredita le proprietà della classe *AsyncTask*. Quest'ultima classe fa parte delle librerie standard di Android, e permette di eseguire parte del codice in background senza bloccare la *activity*.

In questo caso l'applicazione, quando ha bisogno di calcolare un percorso, sfrutta un task per eseguire la richiesta HTTP e la decodifica del risultato in backrgound; nel frattempo, il thread principale mostra all'utente un messaggio di attesa.

5.3 Gestione del database

Nella versione dimostrativa della piattaforma, tutti i dati sui punti di interesse dell'area ospedaliera sono contenuti in un piccolo database SQL. Fortunatamente, Android mette a disposizione alcune classi che permettono di gestire questo tipo di operazioni in maniera estremamente semplice.

In particolare, l'applicazione include le operazioni che riguardano il database nella classe *DBManager*, che eredita le funzionalità della classe di libreria *SQLiteOpenHelper*. Quest'ultima permette allo sviluppatore di evitare tutte le questioni tecniche di creazione e mantenimento del database: una volta instanziato un oggetto *DBManager* è possibile cominciare subito a fare delle query.

Inizializzazione ed aggiornamento del database

Quando l'utente avvia l'applicazione, questa deve costruire il database dei punti di interesse. Per rendere più semplici eventuali modifiche del database, le query di creazione e di riempimento delle tabelle non sono comprese nel codice Java ma sono contenute in due file testuali chiamati *creadb* e *popoladb*.

La classe *DBManager* contiene un metodo, chiamato *executeSQLscript*, che fa il parsing di questi file, cercando di eseguire ogni riga come se fosse una query SQL. Questo metodo viene chiamato durante l'inizializzazione della *Main Activity*.

In questo modo è possibile che una persona modifichi il contenuto del database senza bisogno di mettere le mani sul codice Java.

Usando alcune funzionalità della classe *SQLiteOpenHelper*, sarebbe possibile evitare di creare da zero il database ad ogni esecuzione ed implementare un meccanismo che permetta di aggiornarlo automaticamente in caso ne sia disponibile una nuova versione.

Non ho incluso questa feature nella demo perchè il database stesso è solo un meccanismo temporaneo: in una ipotetica versione definitiva della piattaforma, i dati sarebbero probabilmente contenuti sul server del Policlinico.

5.4 La sezione di ricerca

La sezione di ricerca della destinazione è composta da una schermata di selezione del tipo di ricerca e da un certo numero di schermate, ciascuna delle quali implementa uno dei tipi di ricerca. Attualmente sono implementati quattro tipi di ricerca, ma sarebbe facile aumentare il numero delle schermate in nuove versioni della demo o nella versione definitiva.

La schermata di selezione del tipo di ricerca è implementata nella classe *SearchFragment*. Questo *fragment* ha un semplice layout a griglia che si espande fino ad occupare tutto lo spazio disponibile; ogni elemento della griglia è composto da un'icona ed una *label* che indicano il tipo di ricerca che rappresenta.

5.4.1 Le schermate di ricerca per padiglione e reparto

Queste due schermate hanno un layout molto simile, e sono implementate dalle classi *SearchPadiglioneFragment* e *SearchRepartoFragment*.

L'elemento più importante di questi due *fragment* è la lista dei padiglioni o dei reparti, che occupa gran parte dello schermo. Questa lista è implementata tramite un elemento di tipo *ListView*, che alla creazione del *fragment* è vuoto

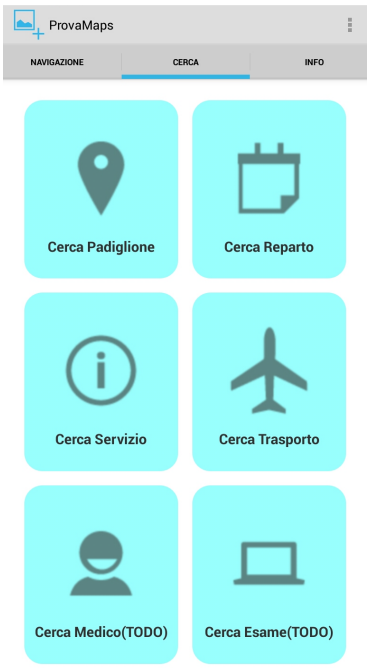


Figura 5.1: La schermata di selezione del tipo di ricerca

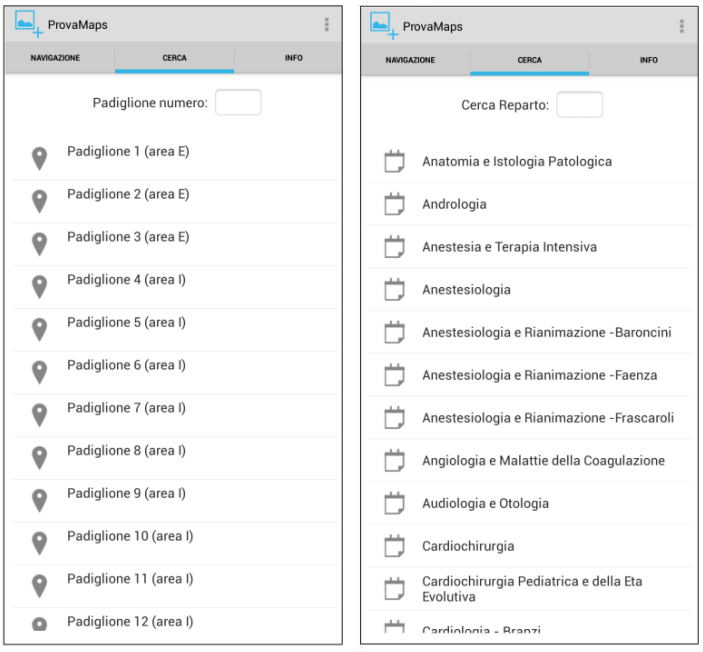


Figura 5.2: Le schermate di ricerca per padiglione e reparto

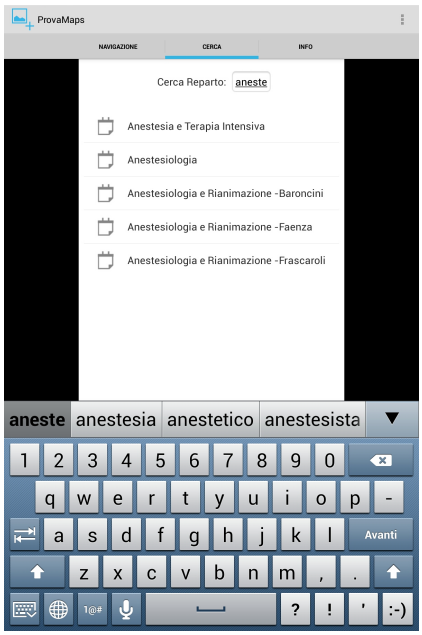


Figura 5.3: Esempio di autocompletamento di una ricerca

e che viene man mano riempito dal *fragment* stesso con il risultato di varie query al database.

Per permettere all'utente di trovare subito un padiglione di cui conosce il numero o un reparto di cui conosce il nome, entrambe le schermate contengono un campo testuale. I due *fragment* rimangono in ascolto di qualsiasi modifica del contenuto di questo campo e reagiscono all'evento con una nuova query al database, limitando i risultati in base al testo immesso dall'utente.

Per esempio, se nella schermata di ricerca per reparto l'utente scrive la parola "anestesiologia", la lista dei reparti sullo schermo verrà limitata a quelli che contengono quella parola nel loro nome.

La ricerca viene ripetuta ogni volta che l'utente aggiunge o cancella una lettera nel campo testuale; in questo modo la lista dei reparti viene aggiornata in tempo reale man mano che l'utente cambia il parametro della sua ricerca.

5.4.2 Le schermate di ricerca per servizio e trasporto

Queste due schermate sono implementate dalle classi *SearchServizioLayout* e *SearchTrasportoLayout* e sono molto simili alle due descritte nella sezione precedente. La differenza è negli strumenti che l'utente ha a disposizione per effettuare la sua ricerca.

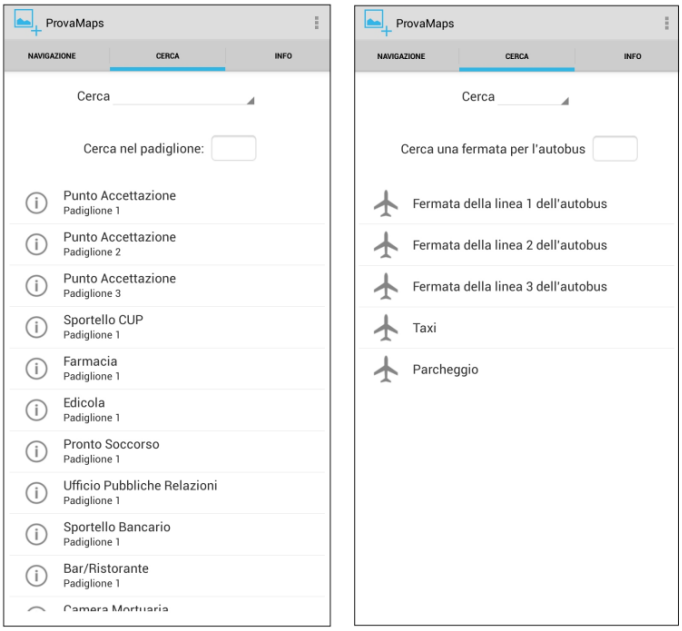


Figura 5.4: Schermate di ricerca per servizio e per mezzo di trasporto

Nel caso della ricerca per servizio, l'utente può selezionare da un menu a tendina il tipo di servizio che sta cercando ed allo stesso tempo immettere in un campo testuale il padiglione a cui questo servizio è legato.

Per esempio l'utente potrebbe voler cercare un qualsiasi punto di accettazione, oppure il punto di accettazione del padiglione 12, oppure qualsiasi servizio che sia legato al padiglione 12.

Nella schermata di ricerca per trasporto l'utente ha di nuovo un menu a tendina da cui può scegliere il tipo di trasporto; inoltre, se sta cercando una fermata di autobus può immettere in un campo testuale il numero di linea dell'autobus che desidera.

Per esempio l'utente potrebbe cercare un qualsiasi parcheggio, una qualsiasi fermata dell'autobus o una fermata dell'autobus da cui passi la linea 19.

Come per gli altri tipi di ricerca, i *fragment* reagiscono a qualunque input dell'utente effettuando una nuova richiesta al database e modificando in tempo reale la lista dei servizi o dei mezzi di trasporto visualizzati sullo schermo.

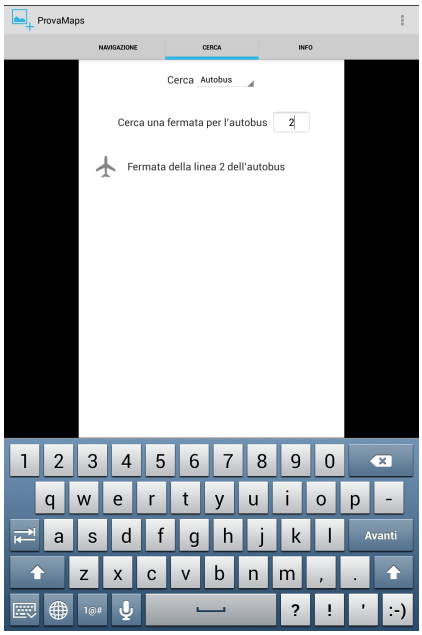


Figura 5.5: Esempio di ricerca di una fermata dell'autobus

5.5 La schermata di informazioni

Questa schermata è stata pensata come un modo per fornire all'utente vari tipi di informazioni sul padiglione che sta visitando; per il momento contiene solamente uno spazio in cui visualizzarne la mappa, ma sarebbe molto semplice modificarne il layout aggiungendo altre informazioni. Il *fragment* che la implementa è un oggetto della classe *InfoFragment*.

La particolarità di questa sezione dell'applicazione è il modo in cui viene selezionato il padiglione visualizzato: ogni volta che l'utente apre la schermata, l'applicazione calcola quale dei padiglioni sia il più vicino alla posizione attuale del dispositivo.

Una volta aperta la schermata di informazioni, l'utente può comunque cambiare manualmente il padiglione visualizzato tramite un menu a tendina. Un altro menu gli permette di selezionare il piano di cui visualizzare la mappa.

5.6 La schermata di navigazione su mappa

Le API di Google Maps permettono di automatizzare quasi tutte le funzionalità richieste da questa schermata usando una classe di libreria chiamata *MapFragment*. All'inizio dello sviluppo l'applicazione usava questa soluzione,

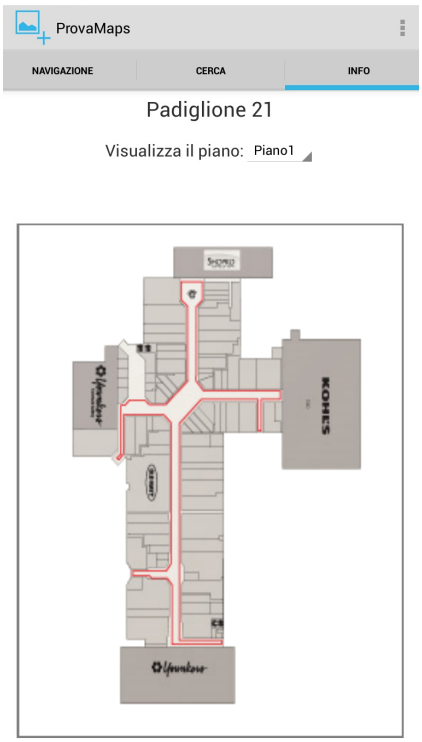


Figura 5.6: La schermata di informazioni sui padiglioni

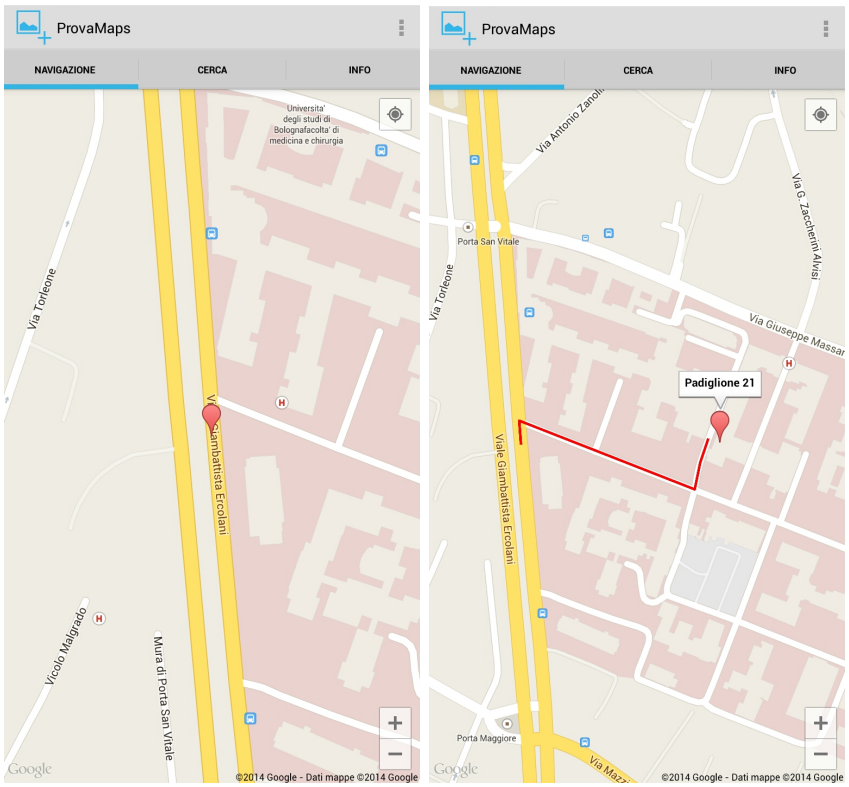
instanziando un *MapFragment* durante l'inizializzazione della *Main Activity* ed usando il metodo *drawPolyline* su di esso per visualizzare i percorsi.

Questo approccio è quello più semplice e fornisce automaticamente una vasta gamma di feature che permettono all'utente di interagire con la mappa: è possibile scorrere la mappa tramite swipe sullo schermo, cambiare il livello di zoom, tornare automaticamente alla propria posizione e così via.

Lo svantaggio dell'utilizzo di un *MapFragment* sta nel fatto che non è possibile modificare i suoi comportamenti di base, alcuni dei quali sono decisamente svantaggiosi per il funzionamento dell'applicazione. In particolare, nel momento in cui passa in background il *fragment* "dimentica" i marker e le *polyline* che gli sono stati assegnati.

Questo significa che se l'utente durante il percorso accedesse ad un'altra schermata (per esempio per controllare le informazioni su un padiglione) e poi tornasse alla sezione di navigazione, sulla mappa non sarebbe visualizzato nessun percorso. L'utente dovrebbe scegliere di nuovo la sua destinazione ogni volta che effettuasse un cambio di schermata.

Dal punto di vista dell'usabilità, questo comportamento del *MapFrag-*



(a) Senza una destinazione (b) Con una destinazione

Figura 5.7: Schermata di navigazione su mappa

ment non è accettabile in un'applicazione di questo tipo: per questo motivo la versione attuale della demo implementa la schermata di navigazione su mappa con un *fragment* personalizzato chiamato *MapNavFragment*.

La *view* corrispondente a questo *fragment* è composta da un solo elemento di tipo *MapView*, che quindi si estende fino ad occupare tutto lo schermo. Questo è un elemento predefinito delle librerie di Google Maps, e fornisce di fatto tutte le funzionalità principali del *MapFragment*; l'unica differenza è che lo sviluppatore, per usarlo, deve creare una classe *fragment* personalizzata in cui specificare il suo comportamento in risposta agli eventi del suo ciclo di vita.

Nel caso del *MapNavFragment*, ho aggiunto del codice al metodo *onResume*: ogni volta che il *fragment* viene riportato in primo piano, ottiene dalla *Main Activity* la posizione attuale del dispositivo e, se sono presenti, la destinazione dell'utente ed il percorso per arrivarci. Fatto questo, visualizza le informazioni sulla mappa e la centra sulla posizione attuale dell'utente.

Questa semplice modifica è tutto ciò che serve per fare in modo che il *frag-*

ment continui a visualizzare il percorso anche se l'utente naviga nelle altre sezioni dell'applicazione durante il cammino. Tutte le altre feature del *MapFragment* (possibilità di muovere la mappa con degli swipe, manipolazione dello zoom, e così via) sono fornite automaticamente dall'elemento *MapView*.

5.7 La schermata di navigazione per immagini

Il layout di questa schermata, implementata dalla classe *ImgNavFragment*, è composto solo da un elemento *ImageView* in cui l'applicazione inserisce di volta in volta l'immagine da visualizzare sullo schermo; la sua implementazione, però, è la parte più complessa di questa demo.

Il nucleo di questa implementazione è un algoritmo che prende in input una serie di punti rappresentante un percorso ottenuto dal servizio Directions e deve ricavarne in qualche modo una serie di immagini che sia in grado di guidare l'utente senza bisogno di una mappa.

Nel corso dello sviluppo ho sperimentato diversi algoritmi di complessità crescente, migliorando man mano i risultati ottenuti. L'algoritmo attualmente incluso nella demo ragguinge un risultato accettabile, ma è ancora migliorabile.

Nelle sezioni seguenti descriverò i tre algoritmi principali che ho sperimentato nel corso della demo, ne spiegherò il funzionamento e, nel caso dei primi due, mostrerò i problemi che mi hanno spinto ad abbandonarli.

Un problema di cui non mi sono occupato nel corso dello sviluppo della tesi è come ottenere di fatto le immagini necessarie: ho programmato gli algoritmi usando dei placeholder. E' importante notare che l'acquisizione delle immagini va rimandata a dopo aver scelto un algoritmo definitivo, in quanto algoritmi diversi hanno bisogno di fotografie diverse.

5.7.1 Primo algoritmo: singoli waypoint

Come primo passo dello sviluppo, ho deciso di partire da un algoritmo fin troppo simplicistico. Come prima cosa ho esaminato i dati di input, ovvero la serie di waypoint ottenuti dalla richiesta al Directions Service. Ciascuno di questi waypoint rappresenta un punto in cui il percorso cambia direzione; si può quindi assumere che gran parte di essi siano in corrispondenza di incroci.

L'idea alla base del primo algoritmo è semplicemente assegnare una fotografia ad ogni incrocio compreso nell'area ospedaliera, e far corrispondere

```
for ciascuno dei waypoint dell'input do
    for ciascuno degli incroci dell'area ospedaliera do
        confronta la distanza tra waypoint ed incrocio;
    end
    prendi l'incrocio con distanza minore dal waypoint;
    if la distanza è minore di 5m then
        prendi la fotografia corrispondente all'incrocio;
        if la sequenza non contiene la fotografia then
            inserisci la fotografia nella sequenza;
        end
    end
end
```

Algorithm 1: Primo algoritmo per la navigazione ad immagini

ogni waypoint del percorso ad uno di questi incroci. Queste fotografie sono contenute in un piccolo database interno all'applicazione.

Dato che l'area ospedaliera è comunque piuttosto limitata sia per dimensioni che per numero di strade ed incroci, è possibile usare tecniche di "forza bruta" senza preoccuparsi più di tanto dell'efficienza: l'algoritmo prende ciascuno dei waypoint contenuti nel percorso e lo confronta con tutti gli incroci presenti nel suo database, scegliendo il più vicino. Fatto questo, inserisce nella sequenza di immagini quella corrispondente all'incrocio appena trovato.

Una volta ripetuto questo procedimento per tutti i waypoint del percorso, si ottiene una lista di immagini corrispondenti al percorso stesso.

Questo algoritmo presenta diversi problemi: tanto per cominciare, non tutti i waypoint del percorso corrispondono a degli incroci. Il servizio Directions inserisce un waypoint in ogni punto in cui il percorso cambia direzione oltre una certa soglia; questo può accadere in corrispondenza di incroci, curve della strada o a volte, in maniera apparentemente casuale, anche durante dei rettilinei. In alcuni casi, due richieste identiche al servizio producono due risultati leggermente diversi per numero e posizione dei waypoint; per esempio, può capitare che solo in una delle due risposte il percorso cambi leggermente angolazione a metà di un rettilineo, generando un waypoint.

Per rimediare a questo problema ho aggiunto all'algoritmo alcuni meccanismi. Il primo è l'aggiunta di un controllo su una *distanza massima*: se l'incrocio più vicino ad un waypoint è più distante di cinque metri, il sistema assume che quel waypoint corrisponda ad una curva o ad un rettilineo e non gli assegna nessuna immagine.

Come seconda cosa, ho fatto in modo che l'algoritmo non aggiunga due

volte la stessa immagine alla sequenza. Capita spesso, infatti, che in corrispondenza di alcuni incroci il percorso effettui una "curva" rappresentata da diversi waypoint; senza questo controllo, in corrispondenza di quella curva l'algoritmo inserirebbe nella sequenza diverse copie della stessa immagine.

Una volta aggiunti questi accorgimenti al codice, l'algoritmo è stato in grado di produrre con affidabilità una sequenza di immagini corrispondente alla serie di incroci attraversati dal percorso di input.

Difetti

Una volta implementati i miglioramenti del caso, si può dire che l'algoritmo funzioni. Tuttavia, il risultato prodotto non è soddisfacente dal punto di vista dell'usabilità: nel momento in cui l'utente raggiunge un incrocio, gli viene semplicemente mostrata un'immagine dell'incrocio successivo.

Se la strada è relativamente sgombra e l'utente la sta guardando dall'angolazione giusta, è possibile che l'immagine visualizzata gli sia di aiuto; in gran parte dei casi, però, non è sufficiente ad indicargli dove andare.

Perchè il servizio di navigazione per immagini sia effettivamente utilizzabile, è necessario mostrare all'utente un'immagine chiara della strada che deve prendere in quel preciso momento, ritratta dall'angolazione giusta.

5.7.2 Secondo algoritmo: coppie di waypoint

```
for ciascuno dei waypoint dell'input, eccetto l'ultimo do
    prendi il waypoint in considerazione ed il suo successore;
    if la distanza tra i due waypoint è maggiore di 5m then
        for ciascuno dei segmenti di strada dell'area ospedaliera do
            confronta la distanza tra i due waypoint e le due estremità
            del segmento;
            if la distanza di entrambi i punti dalle estremità è minore
            di 5m then
                prendi l'immagine corrispondente al segmento;
                aggiungi l'immagine alla sequenza;
                interrompi il ciclo e passa al waypoint successivo;
            end
        end
    end
end
```

Algorithm 2: Secondo algoritmo per la navigazione ad immagini

In questa seconda versione dell'algoritmo i waypoint che compongono l'input non vengono più presi in considerazione singolarmente, ma a coppie. Ad ogni coppia di waypoint viene fatto corrispondere un "segmento di strada" rappresentato da una fotografia.

Come nell'algoritmo precedente, le immagini sono contenute in un database; in questo caso, però, ad ogni immagine corrisponde una coppia di punti. Il sistema è fatto in modo che invertendo l'ordine dei punti si ottenga una fotografia diversa; in questo modo si avranno due immagini diverse per ogni segmento di strada, a seconda del lato da cui lo vuole guardare.

Durante la sua esecuzione l'algoritmo scorre la lista di waypoint considerandoli a coppie (il primo ed il secondo waypoint insieme, poi il secondo e il terzo, e così via). Ogni coppia di waypoint viene confrontata con tutti i segmenti presenti nel database; se viene trovato un segmento sufficientemente simile alla coppia di waypoint in considerazione, l'immagine corrispondente viene aggiunta alla sequenza.

Perchè due coppie di punti vengano considerate "sufficientemente simili", la distanza tra i due punti di partenza e tra i due punti di arrivo deve essere inferiore ad un certo limite. Questo limite è attualmente settato a 5 metri, ma è facilmente modificabile.

Nella sezione precedente ho accennato al fatto che in alcuni casi le curve vengono rappresentate come una sequenza di quattro o cinque waypoint molto vicini tra loro. L'algoritmo scarterebbe comunque i waypoint intermedi senza bisogno di accorgimenti particolari, ma per migliorare leggermente l'efficienza ho fatto in modo che i segmenti più corti di un certo limite vengano scartati prima ancora di venire confrontati con le coppie di punti nel database.

Difetti

Questo algoritmo è teoricamente in grado di produrre una sequenza di immagini corrispondente alla sequenza di strade che compongono il percorso di input, ma ha due problemi che non possono essere superati senza modifiche sostanziali all'algoritmo stesso.

Il primo problema nasce dall'irregolarità dell'output del Google Directions Service. Nella sezione precedente ho accennato al fatto che a volte, in maniera completamente casuale, le parti di percorso corrispondenti a dei rettilinei vengono interrotte uno o più waypoint intermedi.

Come esempio si può considerare il percorso rappresentato in Figura 5.8. Al suo interno, il tratto di strada compreso tra i punti 4 e 5 è un rettilineo

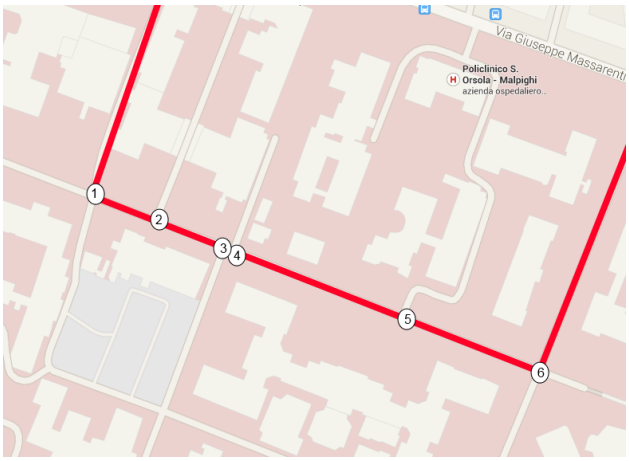


Figura 5.8: Esempio di segmentazione di un tratto di strada

senza curve nè incroci; nonostante questo, il servizio Directions potrebbe decidere di inserire tra di essi un ulteriore waypoint.

Se questo dovesse succedere, il sistema non riuscirebbe ad assegnare un'immagine a questa parte di percorso: il database contiene un riferimento alla coppia di punti (4,5), ma non alle due coppie che verrebbero create a causa di questo nuovo waypoint.

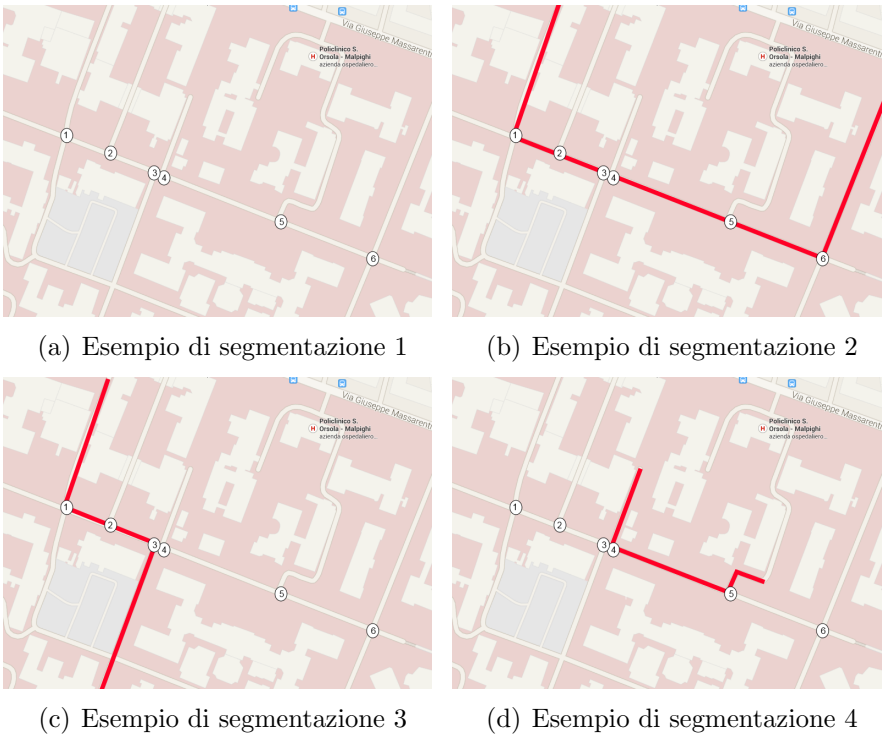
Il secondo problema è molto più grave, dato che non genera errori occasionali ma rende proprio impossibile l'implementazione vera e propria dell'algoritmo. Per mostrarlo, il modo migliore è usare un esempio.

Prendiamo il tratto di strada che in Figura 5.9(a) va dal punto 1 al punto 6. Questo rettilineo può essere compreso, interamente o in parte, in molti percorsi. Il percorso rappresentato in Figura 5.9(b), per esempio, lo comprende interamente; quello rappresentato in Figura 5.9(c) comprende il tratto dal punto 1 al punto 3; quello rappresentato in Figura 5.9(d) comprende il tratto dal punto 4 al punto 5; e così via.

La schermata di navigazione per immagini deve essere in grado di rappresentare tutti questi percorsi; questo vuol dire che il database deve contenere immagini corrispondenti ai segmenti (1,5), (1,3) e (4,5). Non finisce qui: se si considerano tutti gli altri possibili percorsi, quel singolo tratto di strada dovrebbe essere rappresentato da un totale di più di 100 immagini.

Allargando il raggio d'azione dell'algoritmo, il numero di fotografie necessarie al suo funzionamento sale esponenzialmente, arrivando ad un livello improponibile se si considera l'ipotesi di comprendere l'intera area ospedaliera nel database.

Questi due problemi mostrano che confrontare alla cieca coppie di way-



(a) Esempio di segmentazione 1 (b) Esempio di segmentazione 2

(c) Esempio di segmentazione 3 (d) Esempio di segmentazione 4

Figura 5.9: Esempio di possibili segmentazioni di un tratto di strada

point con segmenti rimane un approccio troppo simplicistico. Se si vuole arrivare ad un'implementazione della navigazione per immagini accettabile sia dal punto di vista dell'efficienza che del numero di immagini necessarie, l'algoritmo deve usare un po' di logica.

In particolare, è necessario che l'applicazione sia in grado di comprendere che alcuni segmenti di strada sono composti da segmenti più piccoli e, sfruttando questo principio, di generare la lista di immagini in maniera più flessibile.

5.7.3 Terzo algoritmo: grafo di waypoint

Grazie all'utilizzo dei servizi Google Maps e Google Directions, l'applicazione non ha avuto bisogno di implementare un sistema di mappatura o rappresentazione dell'area ospedaliera durante lo sviluppo della sezione di navigazione su mappa. Tuttavia, per superare i problemi descritti nella sezione precedente è stato indispensabile includere nella sezione di navigazione per immagini un sistema, per quanto minimale, di rappresentazione interna delle strade del Policlinico.

```
inizializza una sequenza di nodi vuota;
for ciascuno dei waypoint dell'input do
    for ciascuno dei nodi del grafo do
        confronta la distanza tra waypoint e grafo;
    end
    prendi il nodo più vicino al waypoint;
    if la distanza è minore di 5m then
        aggiungi il nodo alla sequenza di nodi;
    end
end
for ogni nodo della sequenza, eccetto l'ultimo do
    prendi il nodo in considerazione ed il suo successore;
    if esiste un arco tra i due nodi then
        aggiungi l'immagine corrispondente all'arco alla sequenza;
    else
        calcola il percorso più breve tra i due nodi; aggiungi le
        immagini degli archi così ottenuti alla sequenza;
    end
end
```

Algorithm 3: Terzo algoritmo di navigazione per immagini

Nella terza versione dell'algoritmo di navigazione per immagini, l'insieme delle strade dell'area ospedaliera è rappresentato tramite un grafo. I nodi del grafo corrispondono a tutti gli incroci o curve delle strade del Policlinico, mentre gli archi corrispondono ai segmenti di strada che uniscono questi incroci e curve.

Ad ogni nodo è assegnata un'etichetta e ad ogni arco corrispondono due immagini, che rappresentano il segmento di strada corrispondente fotografato dalle due estremità.

Una volta costruito questo grafo, l'algoritmo procede in maniera simile alla sua prima versione: l'applicazione prende ciascuno dei waypoint che compongono il percorso fornito da Google Directions e lo confronta con tutti i nodi contenuti nel suo grafo, cercandone uno sufficientemente vicino.

Una volta ripetuto questo procedimento per tutti i waypoint, l'applicazione ha trasformato la lista di punti geografici in una lista di nodi del suo grafo. Fatto questo, calcola il percorso più veloce all'interno del grafo per percorrere questa lista di nodi, ottenendo così una lista di archi.

Compiuto anche questo passo, è sufficiente prendere l'immagine corrispondente a ciascuno di questi archi per ottenere la lista di immagini da mostrare all'utente.

Perchè questo algorimo possa funzionare è necessario che l'applicazione possa calcolare il percorso più veloce tra due nodi all'interno di un grafo. Esistono diversi algoritmi a questo scopo, come l'algoritmo A* o quello di Dijkstra; in questo modo, però, si aggiunge un ulteriore livello di complessità all'algoritmo.

Questo problema è mitigato dal fatto che nella maggior parte parte dei casi i segmenti del percorso di Google Directions corrispondono già a singoli archi del grafo, o al massimo a percorsi di due o tre archi. Nel caso medio, di conseguenza, il tempo necessario per calcolare questo percorso è molto breve.

Risultati

Questo terzo algoritmo di navigazione per immagini ha prodotto affidabilmente buoni risultati in tempi accettabili in fase di test. Anche il numero di immagini necessarie per il suo funzionamento è ragionevole: è necessario dividere tutte le strade nei loro segmenti più piccoli ed assegnare due immagini a ciascuno di essi.

Come vantaggio aggiuntivo, questa divisione delle strade in segmenti produce una serie di immagini più dettagliata e quindi più utile per l'utente.

Un altro aspetto interessante di questo algoritmo è che avendolo a disposizione sarebbe in teoria possibile eliminare del tutto l'utilizzo di Google Directions: per trovare un percorso è sufficiente trovare i nodi più vicini ai punti di partenza e destinazione e calcolare il percorso più breve tra di essi all'interno del grafo.

Questo grafo potrebbe quindi essere il primo passo verso una mappatura ad hoc dell'area ospedaliera che incorpori le informazioni necessarie per tutti i tipi di navigazione della piattaforma.

5.7.4 Scorrimento delle immagini

Durante la progettazione della sezione di navigazione per immagini ho notato che per offrire all'utente la migliore esperienza possibile è necessario implementare sia un modo per scorrere manualmente le immagini sia un algoritmo che le scorra automaticamente.

Scorrimento manuale

Considerando le caratteristiche del sistema Android e dei dispositivi mobili, il metodo più comodo di scorrere manualmente una sequenza di immagini è tramite "swipe". Questo termine indica il gesto di appoggiare il dito sul-

lo schermo e "trascinarlo" leggermente in una direzione, utilizzato in molte situazioni dai dispositivi dotati di touchscreen.

Per fare in modo che una schermata dell'applicazione sia in grado di rilevare gli swipe, è necessario creare al suo interno una sottoclasse che implementi l'interfaccia *GestureListener*. Esistono alcune classi di libreria che facilitano il compito agli sviluppatori; per esempio, nella demo ho utilizzato la classe *SimpleOnGestureListener*.

Questa classe contiene una serie di metodi vuoti, corrispondenti ai vari tipi di interazioni tra l'utente e lo schermo; se lo sviluppatore vuole che la sua applicazione reagisca ad una di queste interazioni, è sufficiente che sovrascriva uno di questi metodi. Per esempio, uno swipe causa l'invocazione automatica del metodo *onFling*.

In conclusione, all'interno della schermata di navigazione per immagini è implementato un *SimpleOnGestureListener* il cui metodo *onFling* fa in modo che venga visualizzata l'immagine successiva (in caso di swipe verso destra) o la precedente (in caso di swipe verso sinistra).

In questo modo l'utente può scorrere velocemente e comodamente la sequenza di immagini man mano che cammina lungo il percorso.

Scorrimento automatico

Per implementare un sistema di scorrimento automatico delle immagini è necessario fare un'aggiunta al metodo *onLocationChange* della *Main Activity*, che viene chiamato dal sistema ogni volta che il Location Service rileva uno spostamento del dispositivo.

La tecnica di scorrimento più semplice è verificare all'interno di questo metodo quale dei nodi compresi nel percorso sia il più vicino all'attuale posizione dell'utente e visualizzare l'immagine corrispondente. Attualmente l'applicazione utilizza questo metodo, con due piccoli accorgimenti.

Il primo accorgimento serve a permettere all'utente anche lo scorrimento manuale delle immagini: lo scorrimento automatico avviene solamente se la schermata di navigazione per immagini è in background. In questo modo l'utente può camminare visualizzando un'altra schermata e, quando passa alla sezione di navigazione per immagini, l'applicazione gli mostra quella più vicina alla sua posizione attuale. Fatto questo, l'utente può scorrere la sequenza a piacimento senza che l'applicazione lo ostacoli cambiando automaticamente l'immagine visualizzata.

Il secondo accorgimento serve ad alleggerire l'applicazione: lo scorrimento automatico della sequenza di immagini non avviene ad ogni invocazione del metodo *onLocationChange*, ma solo se è passato un certo periodo di tempo

dallo scorrimento precedente. Attualmente questo timeout è settato a due secondi, ma può essere facilmente modificato; sarebbe anche possibile renderlo una variabile modificabile dall'utente tramite il menu "opzioni".

Capitolo 6

Conclusioni e sviluppi futuri

6.1 Conclusioni

All'interno di questa tesi ho descritto gli eventi che hanno portato allo sviluppo di una versione dimostrativa della piattaforma di navigazione ospedaliera per dispositivi mobili, a partire dai fattori che hanno spinto alla creazione del progetto fino ad arrivare alla progettazione ed all'implementazione della demo stessa.

Il risultato di questo processo è stato lo sviluppo di un'applicazione comprendente buona parte delle feature incluse nel primo livello della piattaforma. Il fatto che la demo fosse stata pianificata per essere usata come "proof of concept" significa che l'implementazione di alcune di queste feature non è ancora ad un livello tale da poter essere considerata completa; tuttavia, tutte le modalità della applicazione sono funzionanti.

L'applicazione permette all'utente di scegliere la sua destinazione tra una serie di punti di interesse realmente presenti nel Policlinico, ed è in grado di calcolare il percorso per raggiungerla e mostrarlo sia nella modalità di navigazione su mappa che in quella di navigazione per immagini.

6.2 Sviluppi futuri

Lo sviluppo della piattaforma può proseguire in due direzioni. La prima è il completamento delle feature incluse nella demo; in questo modo si arriverebbe rapidamente ad avere una versione della piattaforma pronta al deployment, anche se incompleta.

L'altra possibilità sarebbe concentrarsi sullo sviluppo delle feature mancanti, come la navigazione indoor o la realtà aumentata. Per queste sezioni della piattaforma il deployment è molto più lontano, ma implemen-

tarle permetterebbe di ottenere un'applicazione più completa e molto più innovativa.

Sarebbe anche possibile seguire una via di mezzo: completare le feature già esistenti per ottenere una versione utilizzabile della piattaforma, ed in seguito sviluppare le altre modalità ed aggiungerle sotto forma di aggiornamenti dell'applicazione.

6.2.1 Interfaccia

L'interfaccia grafica usata nella demo è funzionale ed abbastanza intuitiva. Durante lo sviluppo dell'applicazione vera e propria, però, sarebbe ideale modificare alcuni elementi per renderla più piacevole esteticamente.

Non si tratterebbe di grandi modifiche: basterebbe perfezionare i parametri dei vari elementi testuali che compongono il layout delle schermate ed inserire delle icone più piacevoli e funzionali nella schermata di selezione del tipo di ricerca.

Quello che manca veramente all'interfaccia grafica della demo è la compatibilità con le varie famiglie di dispositivi su cui l'applicazione potrebbe essere eseguita.

Per ovvi motivi, durante lo sviluppo ho potuto testare l'applicazione solo sul mio dispositivo personale, un tablet con uno schermo da dieci pollici. Tuttavia, la piattaforma è mirata principalmente agli smartphone, che hanno schermi con caratteristiche molto diverse sia in quanto a dimensioni che in quanto a risoluzione.

Perchè l'applicazione sia utilizzabile su tutti i tipi di dispositivi mobili, è necessario che includa diverse versioni dell'interfaccia grafica, ciascuna specializzata per una situazione particolare. Per esempio, nella versione mirata agli schermi piccoli sarebbe necessario ingrandire la sezione del layout dedicata agli elementi testuali perchè questi rimangano leggibili. Sarebbe inoltre necessario preparare diverse versioni delle icone, sempre divise in base alla risoluzione.

6.2.2 Database e server

Nel quarto capitolo della tesi ho parlato del fatto che alcuni elementi dell'applicazione (sicuramente le fotografie delle strade e le mappe dei padiglioni, forse anche il database sui punti di interesse) dovrebbero essere spostati su un server dedicato.

Una volta completata l'installazione del server, la banca dati dovrebbe essere poi preparata. Sarebbe necessario procurarsi informazioni dettagliate

sulle coordinate esatte degli ingressi di tutti i padiglioni, scattare le fotografie di tutte le strade, determinare esattamente quali servizi includere, e così via.

Una volta completati questi passi, il mantenimento del server sarebbe un'operazione relativamente semplice; sarebbe però necessario tenere sotto controllo i cambiamenti alla situazione dell'ospedale per fare in modo che il database li rifletta tempestivamente.

La difficoltà di questo compito varierebbe in base a quali tipi di ricerca venissero implementati: se si aggiungesse la modalità di ricerca per medico, per esempio, sarebbe necessario tenere d'occhio gli spostamenti di tutti i medici del Policlinico.

A livello dell'applicazione, sarebbe necessaria una riscrittura completa della classe *DBManager*: invece di instanziare e gestire un database interno, questa classe dovrebbe occuparsi di inviare le query al server tramite la connessione del dispositivo e di decodificare i risultati.

Dato che durante lo sviluppo della demo il server non esisteva, non ho approfondito la questione. Esistono comunque delle librerie che permettono di gestire in maniera efficace e relativamente semplice processi di questo tipo.

6.2.3 Ricerca della destinazione

Le schermate di ricerca della destinazione, così come sono presentate nella demo, sono già abbastanza funzionali. Prima che questa sezione della piattaforma possa dirsi completa, però, è necessario finalizzare le decisioni sui tipi di ricerca da implementare; queste decisioni non possono essere prese senza aver consultato i rappresentanti del Policlinico.

Alcune questioni relative alle tipologie di punti di interesse, infatti, non possono essere risolte senza una conoscenza approfondita dell'area ospedaliera. Per esempio: la ricerca di "servizi" ha senso, o alcuni di questi servizi meriterebbero una sezione a parte? Ha senso collegare i servizi ai padiglioni? Avrebbe senso assegnare una sezione solo alle fermate degli autobus ed includere gli altri mezzi di trasporto nei "servizi" generici?

Allo stesso modo, non è possibile implementare tipologie di ricerca più "particolari" senza un confronto con le persone che dovranno effettivamente occuparsi del database.

Per esempio, la sezione di ricerca per medico permetterebbe agli utenti di essere diretti verso l'ufficio di un medico di cui sanno solo il nome, o verso il padiglione in cui lavora. Perchè questa sezione possa essere utile, però, è necessario che il database contenga una lista completa dei medici e che questa venga costantemente aggiornata.

Anche in questo caso, ci sono delle questioni a cui solo il personale ospedaliero può dare risposta: è possibile costruire una lista del genere? Ha senso collegare un medico ad un reparto? Quali sarebbero i costi dei frequenti controlli ed aggiornamenti del database, e le possibilità di errore?

Detto questo, anche in questo caso si potrebbe considerare l'opzione di completare l'implementazione delle modalità di ricerca indispensabili e rimandare le considerazioni sugli altri tipi di ricerca a dopo il deployment. In questo modo si potrebbe tener conto anche delle opinioni degli utenti nel decidere quali di queste modalità siano più urgenti.

6.2.4 Navigazione su mappa

Per quanto riguarda la navigazione su mappa, è necessario prendere una decisione definitiva riguardo alla mappatura da utilizzare. Se si decidesse di passare alle mappe di OpenStreetMap o in generale ad una mappatura diversa da quella attualmente utilizzata, sarebbe necessaria una modifica a buona parte del codice dell'applicazione.

Nel caso venisse mantenuta l'implementazione basata su Google Maps, questa sezione dell'applicazione potrebbe considerarsi grossomodo completa. Sarebbe possibile fare qualche piccolo miglioramento estetico ed aggiungere qualche opzione in più all'applicazione, come la possibilità di visualizzare sulla mappa dei marker in corrispondenza di tutti i padiglioni.

6.2.5 Navigazione per immagini

Escludendo la parte di inserimento dati, l'unica cosa che manca per considerare completa questa sezione è una decisione definitiva su come funzioni lo scorrimento della sequenza di immagini, in particolare su come dovrebbero interagire lo scorrimento automatico e quello manuale.

Sarebbe anche possibile pensare a diverse modalità di scorrimento ed aggiungere al menu opzioni la possibilità di scegliere quale utilizzare.

Se venisse deciso di includere la mappatura nell'applicazione stessa, per esempio scaricando da OpenStreetMap la mappa che rappresenta la zona del Sant'Orsola ed inserendola nel database dell'ospedale, sarebbe appropriato eliminare il "grafo" di cui fa uso l'algoritmo ed incorporare la creazione della sequenza di immagini nella fase di ricerca del percorso.

6.2.6 Navigazione tramite *augmented reality*

Per sviluppare la parte finale del primo livello di navigazione sarebbe necessario uno studio approfondito delle tecniche di *augmented reality* attualmente in uso e di quanto tempo, denaro e risorse sarebbero necessari per poterle utilizzare.

Fatto questo, bisognerebbe scegliere quali e quante *feature* di realtà aumentata inserire all'interno dell'applicazione: si potrebbe usare questa tecnica solo per la rappresentazione del percorso, in maniera simile alla navigazione per immagini, oppure sfruttare i *landmark* anche per il riconoscimento della posizione.

Queste decisioni dovrebbero essere prese in collaborazione con i rappresentanti del Policlinico, dato che potrebbero comportare alcuni interventi, per quanto minori, sulle strutture dell'ospedale.

6.2.7 Navigazione indoor

Anche in questo caso, prima di implementare il secondo livello della piattaforma sarebbe necessario un confronto con i rappresentanti dell'ospedale per decidere quali tecniche utilizzare. Inoltre, in questo caso si parlerebbe di modifiche che potrebbero essere anche piuttosto invasive.

Una volta deciso un approccio, sarebbe necessario selezionare una zona, per esempio un singolo padiglione, da usare per la fase di sviluppo e di testing. Solo a questo punto si potrebbe passare all'implementazione vera e propria del sistema di *indoor positioning*.

Non potendo contare su tecnologie già ampiamente utilizzate e testate come nel caso della normale navigazione outdoor, i tempi di sviluppo di questo livello della piattaforma potrebbero rivelarsi molto lunghi, soprattutto se si decidesse di ibridare più tipi di tecnologie per ottenere risultati migliori.

6.2.8 Navigazione per utenti con disabilità

Come prima cosa, è necessario decidere quali siano esattamente le feature che comporranno questo livello della piattaforma. In alcuni casi si tratterebbe di funzionalità che riguardano solo l'interfaccia dell'applicazione (come la navigazione vocale per non vedenti), in altri sarebbe necessaria una modifica della mappatura e della ricerca del percorso (come per la navigazione per utenti con disabilità motorie).

In generale, per facilitare lo sviluppo delle funzionalità di questo livello sarebbe molto utile passare ad un qualche tipo di mappatura personale del-

l'area ospedaliera, in modo da poter modificare liberamente le mappe e la funzione di ricerca del percorso.

Bibliografia

[1] Anind K. Dey, Gregory D. Abowd: *Towards a better understanding of context and context-awareness*

[2] Albrecht Schmidt: Context-aware computing: context-cwareness, context-aware User Interfaces, and implicit interaction

[3] Sito ufficiale del governo USA sul Global Positioning Systme : *gps.gov*

[4] Jimmy LaMance, Javier DeSalas, Jani Jarvinen: Assisted GPS: A low-infrastructure approach

[5] Paul D. Groves: *Principles of GNSS, inertial and multisensor integrated navigation systems*

[6] IEEE Sensors Council: Inertial sensor technology trends

[7] Anja Bekkelien: *Bluetooth indoor positioning*

[8] Silke Feldmann, Kyandoghere Kyamakya, Ana Zapater, Zighuo Lue: An indoor Bluetooth-based positioning system: concept, implementation and experimental evaluation

[9] Augmented Reality ON: *What is augmented reality*

[10] Martin Jaros: Augmented reality navigation

[11] Sito ufficiale del Policlinico Sant'Orsola-Malpighi, *aosp.bo.it*

[12] Android Open Source: Codenames, tags and build numbers

[13] Android Developers, *developer.android.com*

[14] Google Developers, developers.google.com