Sistemi di localizzazione e navigazione outdoor e indoor su dispositivi mobile per la fruizione di beni museali e artistici

Riccardo Del Chiaro, Franco Yang  
*Dipartimento di Ingegneria dell’Informazione, Corso di laurea in Ingegneria Informatica, Università di Firenze*

# Obiettivi

L’obiettivo principale dell’applicazione è quello di rendere fruibili, in maniera più naturale possibile, le opere d’arte situate nei vari musei, attraverso la navigazione sia outdoor che indoor. La filosofia seguita è appunto quella della “interazione naturale”, che cerca di rendere più semplice e immediato possibile il dialogo con l’utente, ottenuta attraverso una sensorizzazione dell’ambiente d’interesse. In tal modo lo sforzo richiesto all’utente è ridotto al minimo e l’azione per ricercare le informazioni rilevanti è delegata al dispositivo piuttosto che all’utente, che di fatto non deve far altro che avvicinarsi ad un’opera o, nel peggiore dei casi, inquadrare un QR code con la fotocamera del dispositivo mobile.

Inoltre l’applicazione si prende carico di profilare le preferenze di un utente, attraverso un’analisi statistica sui musei e/o opere maggiormente visitate. Il tutto per eventuali suggerimenti sulle possibili mete da fruire in futuro.

# State of Art

La parte che presenta più problematiche e che sicuramente è più aperta a futuri sviluppi è la navigazione indoor, problema non ancora del tutto risolto a livello mondiale (come riscontrabile da altri progetti e da ricerche online). Esistono, infatti, molte soluzioni che variano per costi e complessità di realizzazione.

Una possibile soluzione sarebbe quella di usare una sorta di “mini-gps” consistente di una antenna che riesce a rilevare la posizione precisa entro un edificio di qualsiasi dispositivo mobile. È chiaro come una soluzione del genere sia costosa sia per la progettazione che per la realizzazione e tuttavia non esente da problematica quali interferenze con i muri.

Soluzioni più economiche consistono nella sensorizzazione degli edifici, anche di tipologie diverse in modo da sfruttare più elementi ed arrivare a una localizzazione più accurata possibile. La nostra soluzione fa parte proprio di questa categoria ed in particolare si è deciso di usare beacon e qr code. La motivazione principale di questa scelta sta nei costi relativamente ridotti e nella buona accuratezza (se usati con le giuste librerie) dei dati ricevuti.

Come facilmente reperibile su internet, una problematica importante riguarda l’uso dei beacon e alla loro suscettibilità a interferenze e quindi a errori dovuti all’ambiente circostante. Problematiche che Apple ha in parte risolto (per i propri dispositivi) attraverso la libreria di iBeacons, si può infatti provare sperimentalmente la notevole accuratezza dei dati ricevuti su un dispositivo iOS in particolar modo sulla distanza. Accuratezza che Apple sembra essere riuscita a raggiungere attraverso l’elaborazione dei dati “raw” ricevuti dai beacon e dall’uso di algoritmi efficienti per l’attenuazione delle interferenze fra beacon e ambiente circostante (comprendente anche altri beacon, rifrazioni dei segnali ecc.). Chiaramente tale accuratezza è fondamentale per funzioni quali localizzazione attraverso triangolazione e meno rilevante qualora si necessiti solamente della prossimità.

Sfortunatamente per la piattaforma Android non esiste una libreria simile a quella di iBeacons e le conseguenze sono immediatamente riscontrate nel fatto che le misurazioni delle distanze oltre ad 1 metro risultano ben poco accurate e quasi casuali. Altra possibile causa di tali imprecisioni, oltre alla mancanza di una libreria, potrebbe essere il vasto panorama di dispositivi che supportano Android e che quindi presentano diverse case produttrici per cui a livello di hardware, bluetooth in particolare, presentano tarature diverse. Una soluzione potrebbe essere l’implementare un algoritmo che tari ogni dispositivo facendo porre questo a distanze prefissate dal beacon per poi misurare la potenza del segnale ricevuto e di conseguenza rimappare i valori.

Per sperimentare in maniera immediata la differenza del comportamento dei beacon sulle varie piattaforme è possibile scaricare da AppStore o da GooglePlay l’applicazione demo di estimote che comprende le varie funzioni di prossimità e distanza.

Altro problema dipendente dall’accuratezza dei beacon è la difficoltà nel determinare la direzione verso la quale il dispositivo sta puntando, per cui nel caso si avessero due opere adiacenti sulla stessa parete non è ben definibile a quale delle due l’utente sia interessato. Soluzione parziale al problema potrebbe essere l’uso combinato della bussola (presente ormai in tutti i dispositivi) e di algoritmi probabilistici che determinino quale sial il beacon più vicino.

Ovviamente i beacon non sono l’unico strumento utilizzabile, vi sono molti altri sensori di prossimità più o meno accurati che andrebbero testati, tuttavia, come già accennato, i beacon risultano semplici da usare e di costi relativamente ridotti.

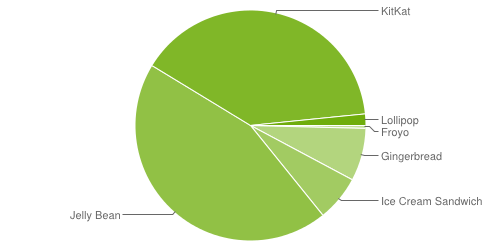
# Contesto

L’applicazione è stata pensata per funzionare in un contesto ben preciso:

* L’utente finale possiede un dispositivo mobile (smartphone, tablet) con sistema operativo Android.
* L’utente che offre il servizio nel proprio edificio sensorizza esso con Beacon (Estimote) e/o QR code.
* L’utente che offre il servizio si appoggia su un server con le seguenti caratteristiche:
  + Database per il retrieving delle informazioni sia sui musei che sulle opere.
  + Possiede uno strato di software che effettui query sul database e che risponda all’applicazione con un formato JSON interpretabile.

In questo caso si è utilizzato uno strato di software lato server in php per effettuare query su un database MySQL, traducendo poi le risposte del database in JSON.

## Utente finale

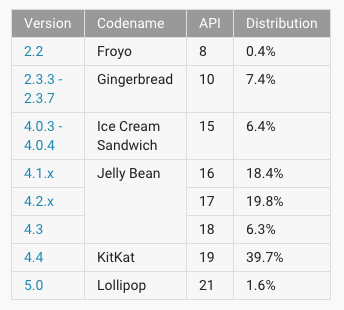
Data la grande diffusione su una ampia fascia sociale dei dispositivi mobili basati su sistema operativo ANDROID, abbiamo deciso di limitare e focalizzare i nostri sforzi su questa precisa piattaforma.

Per offrire un’esperienza di utilizzo più coinvolgente ed allo stesso tempo semplice ed immediata abbiamo pensato di utilizzare le ultime versioni dell’SDK di Android fornito da Google, che consente di utilizzare strumenti più avanzati sia per la programmazione che per lo sfruttamento di risorse dei dispositivi moderni, come l’accelerazione dell’interfaccia grafica utilizzando le risorse di calcolo grafico della GPU.

Non tutte le features fornite dagli aggiornamenti dell’SDK sono retrocompatibili con le vecchie versioni del sistema operativo Android, per questo siamo stati costretti a scegliere in anticipo quali versioni di android escludere dal supporto del nostro software e quali invece integrare.  
Giudicando le analisi ufficiali fornite da google e confrontandole con i reali guadagni nell’utilizzare una versione minima richiesta più moderna, siamo giunti alla conclusione che avremmo potuto escludere le versioni di android precedenti alla 4 (ed eventualmente anche la 4, includendo così il supporto solo dalla 4.1 Jellybeam, se si fossero stati riscontrati dei reali vantaggi nell’escludere la versione 4).

La versione 4 di android offre infatti una serie di nuove feature che rivoluzionano completamente il sistema [[[1]](#footnote-1)], offrendo una nuova modalitá più efficace e dinamica per la gestione della interfaccia grafica (i fragment, disponibili fino a quel momento solo su android 3 che era riservato a device con grossi schermi, come tablet) oltre che un nuovo tema di base più moderno ed il supporto all’accelerazione hardware tramite GPU per il disegno dell’interfaccia grafica.  
Tutte queste feature (e probabilmente molte altre) si sono rivelate fondamentali nella costruzione della nostra applicazione, che fa largo utilizzo di fragment e della accelerazione hardware per disegnare le mappe degli edifici tramite le librerie di disegno standard di android per le interfacce grafiche.

Inoltre con android 4 viene riunificato l’ecosistema che era stato diviso in due con l’annuncio di android 3 (disponibile ed ottimizzato solo per tablet). Android 4 infatti supporta pienamente sia dispositivi con piccoli schermi (smartphone) che dispositivi con schermi più grossi (tablet o mini-pc).  
Abbiamo così rinunciato, al momento dell’analisi descritta, a dare il supporto per il nostro software a circa il 7.8% del totale degli utenti dell’ecosistema android [[[2]](#footnote-2)].



## Utente erogatore del servizio

L’utente erogatore del servizio è colui che decide di utilizzare la tecnologia descritta per offrire un servizio ai propri client/utenti. Potrá quindi porre beacons Bluetooth e QR code nel proprio edificio (museo, nello specifico).

[ TODO ]

[ PARLARE DEI BEACONS, CON SPECIFICHE TECNICHE, COSTI, E ANALISI DI MERCATO RIGUARDANTE I COSTI DEI BEACONS ANCORA NON PROPRIAMENTE BASSI PER VIA DELLA ANCORA SCARSA DIFFUSIONE, CHE SIAMO SICURI CRESCERÁ NEL TEMPO ANDANDO AD ABBASSARE I PREZZI ALL’UTENTE FINALE DEI BEACONS]

Una delle prime cose che si notano (anche con una scarsa esperienza) nello sviluppare software, è che quando si può risolvere un problema lo sforzo necessario a risolvere la sua astrazione (e quindi risolvere contemporaneamente molti problemi simili) è spesso irrisorio, e porta anche il vantaggio di costringersi a lavorare con una migliore progettazione ed organizzazione del progetto.

L’estendibilitá del software è un presupposto fondamentale per il ciclo di vita del software stesso, per questo motivo il nostro approccio nello sviluppo è stato quello di essere il più astratti possibili nei limiti del buon senso, mantenendo un giusto equilibrio tra estendibilitá e sforzo necessario ad implementare tale proprietá.

# Soluzioni tecniche adottate

Per produrre questo applicativo abbiamo proceduto con un approccio TOP-DOWN, focalizzando la nostra attenzione su come avrebbe dovuto essere l’esperienza di utilizzo del prodotto per l’utente finale sul proprio dispositivo mobile. L’obiettivo, come abbiamo detto, è proprio offrire un’esperienza meno invasiva ma comunque efficace da parte del supporto elettronico verso l’utente, in modo che la propria esperienza con l’arte sia il meno possibile disturbata da esso.

L’interfaccia grafica è quindi fondamentale per ottenere questo risultato, poiché deve essere semplice, familiare e intuitiva.

Siamo così partiti dalla sua progettazione, immaginando il prodotto finito nella sua interessa, disegnando come sarebbero dovuti apparire tutti gli elementi grafici. Abbiamo così prodotto un mockup dell’applicazione che abbiamo utilizzato poi come linea guida durante tutto il processo di creazione del software.

Le ultime linee guida di Google per lo sviluppo di interfacce grafiche ci sono venute in aiuto. Esprimono proprio quel bisogno di semplicitá ed immediatezza che stavamo cercando.

## Material Design

Il Material Design vuole essere un linguaggio visivo che sintetizzi, con pochi concetti, i principi classici per un buon design[[[3]](#footnote-3)]. Può quindi essere una chiave intuitiva di lettura per una interfaccia grafica di un software. Semplice ed immediata per un nuovo utente, oltre che familiare ed intuitiva per un utente che ha già avuto modo di utilizzare le ultime versioni dei software Google e Android.

Abbiamo così deciso di seguire queste linee guida per uniformarci ad uno standard che riteniamo comodo e quanto mai efficace. Comodo per l’utente finale… ma purtroppo non possiamo dire lo stesso per lo sviluppatore, che, se vuole utilizzare questo standard allo stato attuale, supportando terminali precedenti alla ultima release di Android (Lollipop, 5.0), non avrà alcuno strumento o aiuto da parte di Google nell’SDK.

Il Material Design è infatti una idea nuova e gli aiuti per implementare i suoi principi in modo semplice nelle applicazioni Android sono stati introdotti con le ultime API (level 21). Non è possibile quindi utilizzare la maggior parte di questo supporto sui terminali con versione Android installata precedente alla 5.0, cioè più del 98% dei dispositivi!

Essendo affascinati da questa filosofia di design, abbiamo ricercato sulla rete strumenti che ci potessero aiutare ad implementare il Material Design sulla nostra applicazione mantenendo la retrocompatibilitá con android 4. Ci siamo così imbattuti in alcuni progetti open source che ci hanno aperto la possibilitá di utilizzare alcuni componenti di base del design ideato da Google senza dover reinventare la ruota partendo da zero. Convinti della possibilitá di poterlo quindi implementare in qualche modo con le nostre mani e con l’aiuto della comunità open source, dopo alcuni test tecnici, abbiamo iniziato a progettare e disegnare il mockup seguendo le linee guida di Google.

[ … TODO parlare del mockup e/o immagini … ]

Per la concretizzazione del nostro mockup su applicazione reale, sono stati fondamentali gli aiuti fornitici indirettamente dalla comunità open source. In particolare abbiamo basato la nostra interfaccia su due componenti grafici personalizzati (custom view) completamente open source e reperibili sulla rete:

* Sliding Up Panel library: https://github.com/umano/AndroidSlidingUpPanel.
* Getbase Floating Action Button: <https://github.com/futuresimple/android-floating-action-button>.

Sliding Up Panel è stato utilizzato come scheletro dell’interfaccia della nostra applicazione, mantenendola sempre attiva sulla finestra principale e modificando i “fragment” contenuti da essa a seconda degli eventi scaturiti dall’interazione con l’utente e con l’ambiente sensorizzato.

I componenti fondamentali delle GUIs (Graphic User Interface, interfaccia grafica) in Android 4.0 e successivi, sono proprio le Activity ed i Fragment. Le Activity sono molto simili al concetto di finestra, che ormai qualsiasi utente tecnologico conosce, il Fragment è invece un concetto un po’ più tecnico. Sono “pezzi” di interfaccia grafica con un loro proprio ciclo vitale che possono essere “incollati” uno alla volta o più di uno in contemporanea su una o più Activity, come adesivi che possono essere staccati e riattaccati in vari punti della finestra in qualsiasi momento.

Abbiamo quindi basato l’applicazione su un’unica Activity nella quale andiamo a scambiare i fragment attivi con quelli non attivi secondo cosa richiede l’utente.

L’Activity contiene così solo lo scheletro dell’interfaccia, con lo SlidingUpPanel che fa da contenitore per 3 fragment. Deve quindi solo preoccuparsi di gestire i componenti grafici comuni a tutta l’interfaccia grafica, lasciando che siano i singoli fragment a gestire i propri peculiari componenti. Lo SlidingUpPanel, infatti, gestisce 2 contenuti dei quali il secondo può scorrere sopra il primo partendo dal basso fino a coprirlo completamente:

1. **Main Content**: il contenuto principale, lo sfondo. Contiene il fragment principale della nostra applicazione. In particolare contiene la vista della mappa, sia che ci troviamo all’esterno di un edificio sia che ci troviamo all’interno (visualizzando ovviamente in questo caso la mappa dell’edificio, se disponibile).
2. **Sliding Content**: il contenuto scorrevole. La custom view SlidingUpPanel gestisce lo scorrimento di questo contenuto quando l’utente effettua un trascinamento della sua parte più alta (barra dello slider) che è sempre visibile. Lo abbiamo gestito in modo che contenga due fragment:
   1. Fragment sliding header: contiene il contenuto della barra dello slider.
   2. Fragment List Container: contiene il content secondario, nella pratica saranno delle liste o delle descrizioni dei contenuti selezionati.

Quindi per passare visivamente, ad esempio, dalla lista dei musei alla descrizione di un museo, rimuoveremo il fragment della lista dal fragment\_list\_container, inserendo al suo posto un fragment dedicato alla visualizzazione dei dettagli dei musei in generale, al quale diremo di visualizzare i dettagli di uno specifico museo.

Con questo tipo di gestione evitiamo cambi di interfaccia, l’utente non si preoccuperá di dove si trova poiché i fragment verranno posti nel posto giusto al momento giusto in modo automatico, e con un semplice trascinamento dello Sliding Header sará possibile passare dalla visualizzazione della mappa alla visualizzazione dei dettagli su ciò che abbiamo vicino (opera o museo).

Risultato finale in Main Activity

Contenitore del  
Fragment List Container

Contenitore del  
Fragment Sliding Header

Sliding Content

Contenitore del  
Main Fragment

Main Content

## Progettazione del Software

Completato il mockup siamo passati alla progttazione del software suddividendo inizialmente il contesto in:

1. Lato Client
   1. Gestione dell’interfaccia grafica e interazione con l’utente.
   2. Gestione del motore outdoor (mappe, localizzazione e prossimitá).
   3. Gestione del motore indoor (mappe indoor, localizzazione e prossimitá indoor).
   4. Interfacciamento con un generico database che risponde con formato JSON.
2. Lato Server
   1. Progettazione base di dati.
   2. Gestione del database con query e traduzione delle risposte in JSON.

Per la parte lato client, data la nostra totale ignoranza in merito all’SDK Android, abbiamo iniziato a studiare l’ecosistema sulla documentazione ufficiale Google, in modo da comprendere i principi fondamentali.

Una volta completato questo primo ciclo siamo passati ad un approccio empirico, dato che le conoscenze acquisite non sarebbero bastate data la quasi totale inesperienza sul campo: abbiamo iniziato a fare dei test per ogni singola area che sapevamo avremmo dovuto sviluppare, per capire quali fossero le strade percorribili, oltre che per fare esperienza ed ottenere manualitá con questi, per noi, nuovi strumenti.

I test fondamentali che abbiamo condotto sono:

* **Test sulla gestione dei Fragment e dell’interfaccia grafica**: inoltre test sulla integrazione dei componenti grafici open source (in particolare in questa fase abbiamo scelto di integrare SlidingUpPanel).
* **Test sulla visualizzazione delle mappe outdoor**: siamo partiti con l’idea di utilizzare Open Street Map, ma abbiamo riscontrato i grossi limiti di questa piattaforma dovuti anche ad una quasi inesistente documentazione. Siamo così ricaduti sull’utilizzo delle Google Maps Api v2 per Android.
* **Test sulla visualizzazione delle mappe indoor**: non avendo trovato alcun componente che potesse aiutarci nella visualizzazione di una mappa indoor personalizzata, siamo partiti con l’idea di costruire un nostro motore proprietario per la visualizzazione di mappe indoor. Google Maps fornisce infatti un motore per mappe indoor, ma obbliga che le nostre mappe siano fornite al server centrale Google, obbligo che non vogliamo noi stessi dover dare ai clienti erogatori del servizio. Abbiamo quindi iniziato a fare test con librerie grafiche (come openGL ES) ricadendo su un soluzione più consona ai nostri obiettivi, ovvero le librerie di disegno grafico di Android, che sono accelerate dalla GPU dalla versione 4.0 presentando quindi sia il vantaggio della semplicitá sia quello delle performance e leggerezza.

1. https://developer.android.com/about/versions/android-4.0.html [↑](#footnote-ref-1)
2. https://developer.android.com/about/dashboards/index.html [↑](#footnote-ref-2)
3. http://www.google.com/design/spec/material-design/introduction.html [↑](#footnote-ref-3)