rocessatiplementare questa interfaccia.si classe i cui oggetti vogliono essere analizzati dal ProximityManager per una analisi 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, in ottica di un progetto più ampio basato su una idea di “Smart Art”.

# 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 potrebbe essere quella di tentare di creare un nuovo standard delle telecomunicazioni per la triangolazione indoor, rendendo possibile una localizzazione accurata tramite antenne poste internamente all’edificio e di una adeguata tecnologia in ogni dispositivo. È chiaro come   
una soluzione del genere sia costosa sia per la progettazione che per la realizzazione, che tutti i vecchi dispositivi non sarebbero probabilmente compatibili e che tuttavia non sarebbe comunque esente da problematica quali interferenze con antenne poste in stanze vicine e ostacoli che potrebbero alterare il segnale.

Soluzioni più economiche consistono nella sensorizzazione degli edifici con tecnologie giá disponibili, anche più di una contemporaneamente, in modo da sfruttare più elementi ed arrivare ad una localizzazione più accurata possibile. La nostra soluzione fa parte proprio di questa categoria ed in particolare si è deciso di utilizzare beacon e QR code. La motivazione principale di questa scelta sta nei costi relativamente ridotti, nella ampia diffusione di queste tecnologie lato client e nella buona dei dati ricevuti. Infatti la gran parte dei dispositivi mobili moderni ha ormai una radio bluetooth capace di connettersi ai beacon, ed una fotocamera con la quale è possibile scannerizzare QR code. La principale problematica della localizzazione con triangolazione bluetooth risulta essere la calibrazione e la definizione della relazione tra i valori RSSI con la distanza in metri dal dispositivo trasmettente. Per capire infatti la distanza in metri da un dispositivo trasmettente bluetooth si utilizzano infatti 3 valori fondamentali:

* Broadcasting power: è la potenza di uscita dell’apparecchio trasmettente (beacon).
* RSSI: è la potenza del segnale vista dal dispositivo ricevente (dispositivo mobile).
* Measured Power: è un valore calibrato di fabbrica che indica quale dovrebbe essere la potenza ricevuta del segnale ad un metro di distanza dal dispositivo trasmettente.

Questi 3 valori purtroppo non bastano per garantire una buona accuratezza sulla distanza. Infatti, nonostante la formula utilizzata suggerita dalla teoria dei segnali, il predittore della distanza basato sulla potenza del segnale ricevuto (RSSI) può essere ottenuta effettuando una ricalibrazione della potenza basata su una tabella distanza/valori RSSI. [[[1]](#footnote-1)]

Questo perché ogni dispositivo mette in campo molte variabili non facilmente predicibili, e di fatto ricevono i segnali in modo molto differente. Ogni dispositivo può infatti avere un chipset bluetooth differente, oltre che una antenna diversa, una posizione diversa della antenna stessa all’interno del dispositivo, e dei materiali differenti (alluminio, policarbonato, vetro ecc.. ) che possono alterare in modo diverso i segnali ricevuti.

Pensiamo quindi che sarebbe senz’altro necessario uno standard ed una sorta di certificazione che assicuri che un terminale è stato calibrato per questo tipo di tecnologia. In questo modo, se la tecnologia si diffondesse, i produttori di telefoni potrebbero fare le calibrazioni e farsele certificare da un ente addetto, in modo da poter esporre pubblicamente la certificazione ed i parametri necessari alla calibrazione delle formule che legano la distanza con i valori RSSI per un preciso dispositivo.

Un’altra problematica importante, oltre alla corretta calibrazione, riguarda l’uso dei segnali radio in generale 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 e probabilmente con una più evoluta gestione dello stack bluetooth.

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, ma sicuramente questi tipi di lavori non potranno mai partire sull’ecosistema Android (o su qualsiasi altra piattaforma) se prima non nasce uno standard mondiale per i beacon (lo standard di apple, iBeacons, e proprietario).

Sfortunatamente per la piattaforma Android non esiste ancora niente del genere e le conseguenze sono immediatamente riscontrate nel fatto che le misurazioni delle distanze oltre ad 1 metro risultano ben poco accurate e quasi casuali.

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.

Date queste problematiche ci siamo limitati ad utilizzare i beacons bluetooth come sensori di prossimitá (con due possibilitá: sei vicino o sei lontano). Ovviamente i beacon non sono l’unico strumento utilizzabile per questo tipo di lavoro, vi sono molti altri sensori di prossimità più o meno accurati che andrebbero testati (come per esempio la tecnologia NFC). Tuttavia i beacon risultano semplici da usare poiché non richiedono alcuno sforzo da parte dell’utente (con NFC si deve avvicinare il sensore posto sul proprio telefono ad almeno 20cm circa di distanza con il sensore trasmettente), i costi sono relativamente ridotti e la tecnologia è già fruibile con la quasi totalita dei dispositivi, al contrario di NFC che ancora oggi, in alcuni dispositivi mobili top gamma, continua a non essere sempre inserito ed in ogni modo nei dispositivi di fascia media o low cost raramente è presente.

# 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.

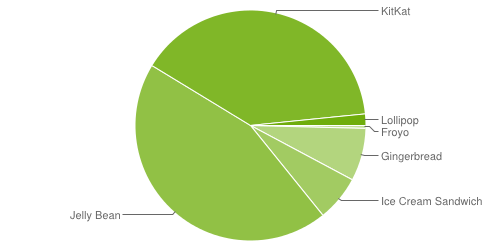
In questo caso si è utilizzato uno strato di software lato server in php per effettuare query su un database MySQL, traducendo poi le risposte del database in JSON.

## 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.

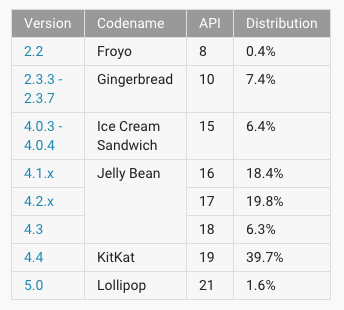
Non tutte le features fornite dagli aggiornamenti dell’SDK sono retrocompatibili con le vecchie versioni del sistema operativo Android, per questo siamo stati costretti a scegliere in anticipo quali versioni di android escludere dal supporto del nostro software e quali invece integrare.  
Giudicando 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 [[[2]](#footnote-2)], 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 [[[3]](#footnote-3)].



## 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. Dovrá quindi porre i beacons Bluetooth e QR code nel proprio edificio (museo, nello specifico) e fornire una mappa indoor in un formato appropriato.

Il database

[ 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[[[4]](#footnote-4)]. 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: preliminari

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.
* **Test su localizzazione GPS, beacon bluetooth Estimote e QR code.**

Una volta completata questa fase preliminare di test, abbiamo potuto procedere per la progettazione vera e propria del software con la sicurezza di poter integrare tutte le tecnologie richieste.

## Progettazione del software: Interfaccia Grafica

I due fondamentali moduli per la gestione dell’interfaccia grafica sono:

**Main Activity**: classe java, activity principale del programma, ed anche il suo punto di ingresso per l’esecuzione. Il suo compito principale è quello di inizializzare tutti i componenti grafici tramite file esterni xml, inizializzare il FragmentHelper e tutti i gestori di eventi dello slidingBarPanel. Si può quindi dire che il suo ruolo sia quello di gestire lo SlidingBarPanel e quello di inizializzare il programma delegando esternamente ad altri componenti il compito di gestire il resto dell’interfaccia grafica (il primo delegato è il FragmentHelper). Inoltre gestisce le animazioni ed i cambi , [ TODO ]

**FragmentHelper:** classe java il cui compito è quello di facilitare la gestione dell’interfaccia grafica e dei fragment, prendendosi carico di tutti i dettagli tecnici riguardanti le trasformazioni dell’interfaccia da un punto ad un altro dell’applicativo, fornendo dei semplici metodi di accesso che possono essere richiamati ovunque. Il FragmentHelper interpreta infatti il ruolo di un mediatore, si interpone tra l’interfaccia grafica (mainActivity e fragments) ed il resto dei componenti specifici del software. Il FragmentHelper implementa il design pattern Singletone, in modo che sia uno strumento utilizzabile da chiunque e ovunque all’interno del codice java.

Oltre a questi due componenti principali, abbiamo delegato le funzionalitá secondarie (come animazioni) e le funzionalitá specifiche di ogni fragment ad un modulo esterno (package **gui**).

In particolare il package gui.customSlidingHeader contiene i fragment che possono essere posti nel contenitore del FragmentSlidingHeader di cui abbiamo parlato, mentre in gui.customList ci sono i fragment ce gli ahe possono essere posti nel FragmentListContainer, oltre che gli adapter necessari per riempire i dati delle liste e delle descrizioni che devono comparire nel fragment.

## Progettazione del software: Outdoor Engine e Localizzazione Outdor.

Avendo deciso di utilizzare le API di Google Maps V2, il lavoro per l’outdoor engine è stato per lo più di integrazione. Abbiamo utilizzato le API per visualizzare la mappa sul nostro MapFragment, delegando ad una classe esterna Map la gestione della mappa e delle API di google. In questo modo abbiamo incapsulato in Map tutte le funzionalitá offerte da google maps, per poi costruire dei semplici metodi di accesso per le funzionalitá che ci erano utili, interconnessi al resto della nostra applicazione.

In particolare Map integra e traduce per google maps le informazioni sulla posizione dei musei derivate dal database, inoltre gestisce la geolocalizzazione e la prossimitá con un museo con l’aiuto delle classi esterne contenute nel package OutdoorProximity.

Per ottenere le indicazioni stradali dalla posizione GPS corrente fino ad un museo, abbiamo utilizzato le Google Directions Api. Per farlo eseguiamo una query sul server google in cui specifichiamo partenza e arrivo in coordinate geografiche, google ci risponde con un oggetto JSON complesso costituito da:

* Route: contiene una serie di leg che uniscono tutti i waypoints (punti attraverso i quali si impone il passaggio del percorso) partendo dal punto di partenza arrivando alla destinazione. Se non ci sono waypoints ci sará una sola Leg.
* Leg: singolo percorso tra 2 waypoints, o tra origine e primo waypoints, o tra ultimo waypoints e destinazione, o tra origine e destinazione in assenza di waypoints.
* Step: singolo tratto rettilineo compreso tra un possibile bivio ed il successivo. Ogni step contiene una polyline, cioè un insieme di punti sulla mappa che approssimano il percorso stradale.

Per gestire tutto questo abbiamo utilizzato una gerarchia di classi che ricalcano gli oggetti JSON, ed una classe GMapRouteManager che ci aiuta ad utilizzare questi oggetti con un unico punto di accesso per gestire questa funzionalitá.  
Il Manager effettuerá la query verso i server google, e provvederá a restituire al chiamante un oggetto Navigation, che contiene tutti gli oggetti sopra elencati giá tradotti da JSON a Java.

Nel nostro caso specifico sará la classe Map a richiamare il GMapRouteManager al momento in cui l’utente richiede di navigare verso un museo grazie all’interazione col MapFragment. Il GMapRouteManager effettuerá la richiesta ai server google, traducendo i dati e restituendoli all’oggetto della classe Map, il quale provvederá a dialogare con le Google Maps Api per disegnare, linea per linea, il percorso contenuto nell’oggetto Navigation fornito dal GMapRouteManager.

Map

Map Fragment

Database  
Manager

Google Maps Api V2

GMapRouteManager

Navigation

Direction Request

Proximity  
Manager

Google Directions Api

Degno di nota è il Proximity Manager. Questa classe è il punto di accesso per la funzionalitá di prossimitá outdoor.  
Il suo funzionamento è dipendente da altre classi ed interfacce satelliti:

**ProximityObject**: interfaccia java che rappresenta un generico oggetto localizzabile tramite latitudine e longitudine. Qualsiasi classe i cui oggetti vogliono essere processati dal ProximityManager per una analisi di prossimitá devono implementare questa interfaccia.

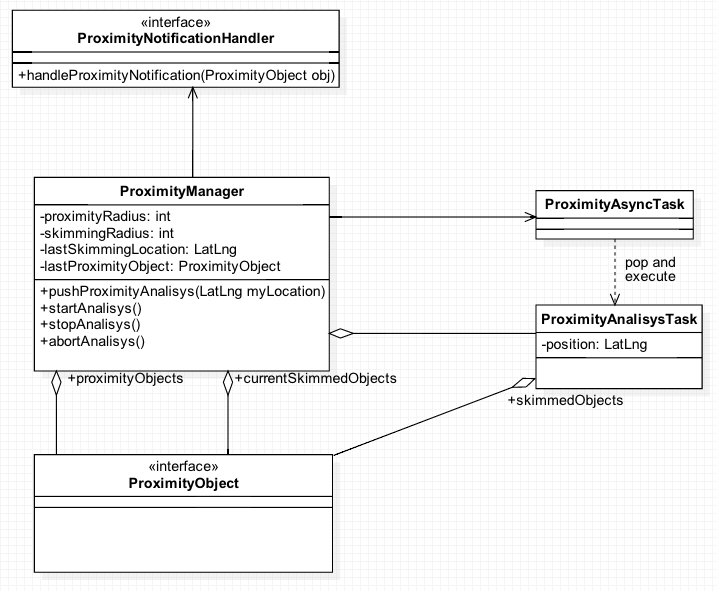
**ProximityNotificationHandler:** interfaccia java che rappresenta l’oggetto che ha interesse a ricever gli esiti delle analisi eseguite dal manager. Ogniqualvolta una analisi di prossimitá da un risultato degno di nota (ovvero quando c’è un cambiamento, cioè quando viene rilevato un nuovo oggetto vicino oppure viene rilevato che non c’è più un oggetto nelle vicinanze) il ProximityManager richiamerá i relativi gestori di eventi del ProximityNotificationHandler che è in ascolto.

**ProximityAnalisysTask:** rappresenta un unico task che ha il compito di analizzare le distanze tra la posizione corrente ed i musei, in modo da avvertire il manager se esiste un ProximityObject vicino alla posizione corrente. Con posizione corrente si intende la posizione corrente al momento in cui il ProximityAnalisysTask è stato inviato al manager. Infatti un ProximityAnalisysTask viene messo in una coda dal ProximityManager ogni qualvolta che un oggetto esterno richiede di eseguire una analisi su una nuova posizione corrente aggiornata, e viene fatta una sorta di screenshot della situazione corrente nel manager (tutti i riferimenti ai ProximityObject da analizzare al momento in cui si richiede la nuova analisi vengono memorizzati nel nuovo ProximityAnalisysTask stesso). Nella pratica verrá creato un task nuovo ogni volta che il GPS rileverà una nuova posizione corrente.

**ProximityAsyncTask:** il ProximityManager lancerá questo task su un thread separato, il suo compito è quello di prendere il prossimo elemento dalla lista di AnalisysTask contenuta nel manager, eseguirla (sul thread separato in modo da non bloccare la graphic user interface) e restituire al manager il risultato. Quando non ci sono AnalisysTasks in coda, il proximityAsyncTask starà inattesa per 1 secondo senza utilizzare le risorse, per poi ripartire e controllare se ci sono nuovi task in coda.

Il ProximityManager implementa inoltre un meccanismo di “scrematura” degli oggetti prima di passarli al nuovo AnalisysTask in modo da velocizzare l’esecuzione di questo: se ci fossero centinaia o migliaia di musei sul database sarebbe inutile analizzarli tutti quanti ogni qual volta la posizione del GPS varia anche di poco. Abbiamo quindi pensato di mantenere su una lista separata i riferimenti a tutti i musei entro un certo raggio (sull’ordine dei kilometri) in modo che, fino a che la nostra posizione non esce da questo cerchio, effettueremo la ricerca dei musei vicini solo tra quelli contenuti nel cerchio, cioè quelli preventivamente scremati. Ogni volta che ci avviciniamo al bordo di questo cerchio, verrà effettuata una nuova scrematura prendendo come origine del cerchio il punto delle coordinate geografiche in cui siamo usciti dal precedente cerchio di scrematura.

Questo è stato fatto perché non possiamo essere certi che in un futuro il numero di musei contenuti su un database sia piccolo.



## Progettazione del software: Indoor Engine

La progettazione del motore per la gestione delle mappe indoor è stata ben più impegnativa, poiché non abbiamo potuto utilizzare nessuna libreria esistente.

Siamo partiti con dei Test sui Canvas (classe di Android per gestire il disegno vettoriale) per assicurarci della fattibilitá della cosa, implementando dei gestori di eventi per gestire le gestures sulle ImageView (come il classico pinch-to-zoom per ingrandire ed il drag per spostare la visuale).

Lavorando sulle matrici di trasformazione per oggetti 2D siamo riusciti a costruire due tipologie di oggetti disegnabili differenti:

* Oggetti che reagiscono allo zoom scalando la propria dimensione in modo proporzionale ad uno scale factor. Questi oggetti rappresentano il background, la mappa.
* Oggetti che reagiscono allo zoom senza scalare la propria posizione, ma traslando il loro centro geometrico in modo che risulti sempre posizionato nello stesso punto relativo sul background (sulla mappa).  
  Questi oggetti rappresentano i markers, ovvero degli indicatori posti in punto preciso della mappa che indicano un punto di interesse.

Abbiamo cioè simulato ciò che Google maps fa con le sue mappe, i markers non cambiano mai dimensione quando compiamo lo zoom, mentre lo sfondo (la mappa) è ingrandito. Naturalmente i markers indicheranno sempre lo stesso punto nonostante che esso sia stato traslato a causa dello zoom.

Su questa base abbiamo costruito il package **building**, che contiene le classi per la gestione e per il disegno di una generica mappa indoor di un edificio, ed il package **spot**, che contiene la gerarchia di classi per la gestione di ciò che deve essere disegnato sopra alla mappa avendo quel comportamento traslatorio agli eventi di zoom.

Questi due package sono strettamente legati e non possono esistere indipendentemente. Data la maggiore complessità della gerarchia degli **Spot**, partiremo descrivendo prima com’è costituito un edificio indoor per il disegno della mappa.

**Building:** è il contenitore dell’edificio, contiene tutti gli oggetti e le informazioni necessarie per disegnare l’edificio. In particolare è una aggregazione di Floor (cioè di “piani” dell’edificio), inoltre contiene uno SpotManager e può gestire il disegno degli spot e del cammino tra due spot, ma approfondiremo questo argomento in un secondo momento.

**Floor:** è un contenitore per le stanze dell’edificio, inoltre contiene anche questo una serie di SpotManager che vedremo più tardi. Ogni edificio può avere un solo Floor attivo contemporaneamente, e sullo schermo verrà disegnato un unico Floor alla volta.

**Room:** è l’entitá più complessa e ricca di funzionalitá. Identifica il concetto di stanza di un edificio. Deve gestire il disegno dei muri, delle porte e dei pavimenti. Per farlo deve essere definito un perimetro della stanza utilizzando i Vertex. La Room contiene una lista ordinata di Vertex e disegnerá il perimetro della stanza seguendo le coordinate indicate da questa lista. A seconda del tipo di Vertex in cui la procedura di disegno si imbatte, essa disegnerá un muro piuttosto che una porta o una apertura (spazio aperto).

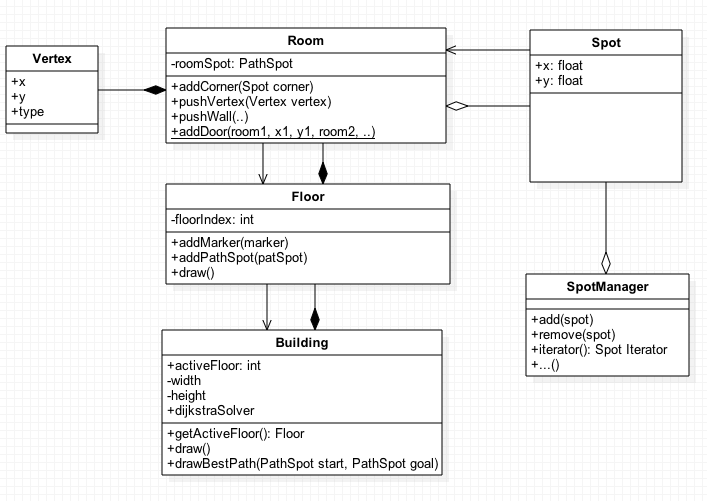
Room inoltre contiene una lista di Spot. Ogni spot infatti è localizzato in una Room. In questo modo possiamo a priori sapere dove si trovano gli spot (in che stanza) e possiamo ricavare una lista di spot a partire da ogni stanza comunicandoci che cosa è contenuto, funzionalità molto interessante poiché le estensioni della classe spot identificheranno i concetti di marker e punti di interesse. Sarà quindi possibile, a partire da ogni stanza, avere una lista dei marker o punti di interesse contenuti.

Ogni stanza ha inoltre uno Spot speciale, chiamato RoomSpot, che deve identificare approssimativamente il centro della stanza. Sarà utilizzato per semplificare la costruzione dei cammini all’interno della stanza.

**Vertex:** rappresenta un vertice di una stanza con delle coordinate spaziali x y in metri. Ogni edificio è disegnato dentro ad una cornice delle dimensioni specificate dall’edificio (width e height), per cui si prende come punto (0, 0) l’origine degli assi, ovvero il primo punto a partire da sinistra in alto dell’immagine. Ogni Vertex ha un tipo che specifica come deve essere disegnato il poligono tra il vertex corrente ed il successivo.

**Spot:** classe base della gerarchia di Spot, identifica un punto nello spazio definito in metri (come per i Vertex). Al contrario del Vertex, uno spot sa sempre in che stanza è contenuto.

**SpotManager:** contenitore e gestore generico per Spot generici. Uno Spot non deve obbligatoriamente essere gestito da uno SpotManager, cioè uno Spot può esistere anche se nessuno SpotManager lo contiene. Oltretutto uno Spot non ha coscienza di dove sia contenuto (non ha un riferimento allo SpotManager che lo contiene, ma solo alla stanza). Quindi uno Spot può essere contenuto da una sola stanza, ma può essere contenuto da più SpotManager.



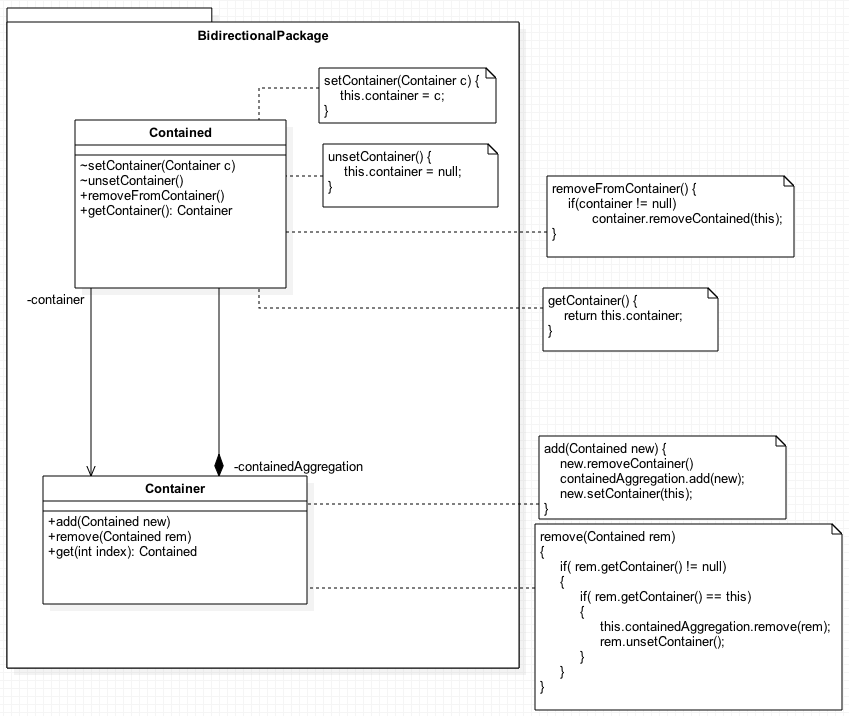
Per disegnare la mappa sarà quindi sufficiente richiamare il todo draw() di Building, il quale provvederà a delegare la richiesta a tutti i suoi contenuti. Ovviamente non è sufficiente richiamare draw() per fare apparire sullo schermo la mappa, infatti questo metodo riceve in ingresso un oggetto della classe Canvas di Android, e farà in modo che tutto ciò che verrà disegnato sarà diesgnato proprio sopra questo Canvas.

L’utilizzatore della classe Building dovrà quindi gestire la parte di proiezione del canvas sullo schermo. Nel nostro caso abbiamo delegato questa operazione alla classe IndoorMap e IndoorMapFragment, che provvedono a proiettare su una ImageView di android il canvas su cui è stato disegnato l’edificio. Queste classi provvedono anche ad implementare i gestori degli eventi della ImageView in modo da gestire lo zoom e il drag della mappa.

Per gestire la bidirezionalitá delle relazioni *contenitore – contenuto* presenti tra Building – Floor, Floor – Room e Room – Spot, si è utilizzato sempre lo stesso modello. Poiché il modello è risultato funzionale ed efficace è stato creato una gerarchia di classi per implementare automaticamente questo modello. Le entitá fondamentali sono Container e Contained, e la loro unione (ContainerContained) fanno si che sia possibile gestire catene di Contenitori e Contenuti senza dover reimplementare lo stesso pattern.

Spieghiamo brevementeil funzionamento: il contained offre imetodi setContainer ed unsetContainer che, senza effettuare alcun controllo, modificano il riferimento privato del contained verso un container a piacimento (nullo nel caso di unsetContainer). Questi metodi sono package protected, ovvero accessibili solo al package. Al contrario, i metodi Add e Remove del Container sono pubblici ed aggiungono un oggetto Contained all’elenco interno degli oggetti contenuti. Allo stesso tempo effettuano un controllo sull’oggetto da aggiungere e modificano in modo appropriato il loro contenutore. Questo è possibile perché il Container ed il Contained devono stare nello stesso package, in modo che il Container possa accedere ad i metodi package protected.

Nonostante i dibattiti sul concetto di classe friend e metodo friend, siamo dell’idea che in questo caso sarebbe stato più consono l’utilizzo di tali concetti per poter offrire al solo Container l’accesso ai metodi del Contained che adesso hanno visibilitá package protected.



Tornando ai dattagli del motore indoor, possiamo adesso analizzare la gerarchia di classi degli Spot e SpotManager.

1. http://altbeacon.github.io/android-beacon-library/distance-calculations.html [↑](#footnote-ref-1)
2. https://developer.android.com/about/versions/android-4.0.html [↑](#footnote-ref-2)
3. <https://developer.android.com/about/dashboards/index.html> [↑](#footnote-ref-3)
4. http://www.google.com/design/spec/material-design/introduction.html [↑](#footnote-ref-4)