Università degli studi di Padova

Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria Informatica

Progettazione e Sviluppo di un Tool per la realizzazione di mappe vettoriali in OS X e del relativo framework in iOS

Tesi di laurea magistrale

Laueando: Rossella Petrucci

Relatore:
Chiar.mo Prof.
Sergio Congiu

Tutore Aziendale:
Matteo Centro

9 luglio 2013
Anno Accademico 2012/2013

Contents

1 Introduzione **1**
1.1 Contesto Aziendale . . . 1
1.2 Obiettivi della tesi . . . 2

2 Analisi delle esigenze **3**

3 Tecnologie Utilizzate **5**
3.1 iOS . . . 8
3.1.1 iOS SDK . . . 9
3.1.2 Interfacce di iOS . . . 9
3.1.3 Architettura di iOS . . . 9
3.2 OS X . . . 18
3.2.1 Architettura di OS X . . . 18
3.2.2 Interfacce di OS X . . . 19

I Realizzazione Mappa per Pulire 2.0 **21**

4 Progettazione e sviluppo **23**
4.1 Visualizzazione dell'intera mappa in un'unica schermata . . . 23
4.1.1 Panoramica dei framework e dei tool esistenti . . . 23
4.1.2 Tiles e Zoom Level . . . 25
4.1.3 Sorgente della mappa . . . 25
4.1.4 Suddivisione della mappa in Tiles con ImageMagick . . . 26
4.1.5 Il framework CATiledLayer e la gestione delle tiles . . . 27
4.1.6 UIScrollView e gestione dello zoom . . . 30
4.2 Gestione di Pin e Callout . . . 31
4.3 Visualizzazione di un punto sulla mappa e animazione . . . 34

II Tool e Framework per la realizzazione di mappe georeferenziate **35**

5 Progettazione e sviluppo **37**
5.1 Realizzazione del tool per Mappe Vettoriali . . . 37
5.1.1 Funzionalità necessarie . . . 37
5.1.2 Conversione di Coordinate: da coordinate Geografiche a coordinate in pixel 39
5.1.3 Inserimento dei Punti di Interesse . . . 42
5.1.4 Generazione del file che descrive la mappa . . . 43
5.2 Realizzazione del Framework in iOS . . . 44
5.2.1 Sorgente della mappa:file plist . . . 44
5.2.2 Sorgente della mappa:file immagine . . . 44

6 Testing **45**
6.1 Pulire 2.0 . . . 45
6.2 Tool per mappe vettoriali e framework . . . 45

7 Conclusioni e Sviluppi Futuri **47**

Ringraziamenti **49**

Chapter 1

Introduzione

"Reinvent the phone". Reinventare il telefono. Era il 9 gennaio 2007 quando Steve Jobs mandò in estasi la folla riunita al MacWorld di San Francisco, tenendo a battesimo il primo smartphone della Mela. "Faremo la storia" disse Steve Jobs ai suoi dipendenti prima del debutto e così è stato. Così come con l'ipod qualche anno prima l'azienda di Cupertino aveva aggiunto qualcosa di nuovo alle semplici funzionalità di un lettore mp3 introducendo iTunes, un software che permetteva di scaricare musica e di sincronizzare i contenuti del proprio ipod con il mac, allo stesso modo l'iPhone non possedeva soltanto le funzionalità di un telefono ma offriva ai propri utenti qualcosa in più. Assieme all'iPhone era possibile accedere ad un mondo nuovo per i dispositivi presenti sul mercato, quello delle applicazioni. L'App Store è un indicatore del rapido incremento di interesse degli utenti per questo nuovo mondo: prima del 2008 le applicazioni software per dispositivi mobili non avevano un mercato globale. I download dagli scaffali digitali di Apple volano a un miliardo nell'aprile del 2009, a 10 miliardi nel gennaio del 2011 fino a 40 miliardi nel primo mese del 2013. Ed è proprio la crescita del bacino di utenti e la potenza di distribuzione dell'AppStore ad attrarre continuamente sviluppatori dal tutto mondo e a dar loro la possibilità di creare un software distribuibile in oltre 100 paesi nel mondo partendo da una semplice idea. Oggi le applicazioni mobile disponibili negli store sono infinite e tutti i giorni ne vengono approvate di nuove nei diversi digital market, "There is an app for that" recitava lo slogan coniato da Apple nel 2009, e questa è effettivamente la realtà ora. C'è un applicazione per ogni cosa, alcune utili altre meno, ma la cosa davvero incredibile non è tanto il numero: è il tipo di applicazione. Le applicazioni che riscuotono maggior successo sono quelle integrate nel sistema e con elevate prestazioni in grado di sfruttare al massimo tutte le potenzialità offerte dall'iOS. In uno scenario come questo affinché un prodotto abbia successo, per differenziarsi dalle molte alternative che popolano il mercato, diviene necessario puntare sugli elementi di innovazione e di originalità. E' proprio in questo contesto che si inserisce la presente tesi, la quale si propone di studiare e definire la migliore metodologia per realizzare mappe vettoriali e permettere all'utente di interagire con esse.

1.1 Contesto Aziendale

Il lavoro di tesi è stato svolto nel quadro di un tirocinio presso l'azienda Altera. Altera è una software house nata nel 1999 con lo scopo di realizzare applicazioni web ed e-commerce ritagliate su misura, le competenze specifiche riguardano principalmente la programmazione in Java, negli anni le competenze si sono estese alla programmazione Objective C su MacOSX e iOS realizzando progetti ad hoc per i mercati più disparati: dal software per il broadcast televisivo al videogame per iOS. Malgrado si tratti di un'azienda molto piccola, Altera lavora per clienti internazionali e su progetti estremamente interessanti come ad esempio il sistema di biglietteria web di The View From The Shard o il software che genera le grafiche di gran parte delle produzioni televisive italiane. Altera non ha prodotti propri ma ha sviluppato negli anni una serie di framework semilavorati da utilizzare nei propri progetti. Proprio in quest'ottica si è deciso di avventurarsi nello sviluppo di un framework per mappe vettoriali su iOS. E'necessario precisare che iOS fornisce già agli sviluppatori un framework per la gestione di mappe vettoriali che permette di gestire la visualizzazione delle

mappe e la geolocalizzazione dell'utente all'interno di esse. Allora perché intraprendere tale avventura? Il problema principale di tale framework è che, basandosi esclusivamente su mappe proprie, non risulta adatto per la gestione di particolari mappe quali ad esempio mappe indoor o di aree molto piccole. Si consideri ad esempio il caso in cui si voglia visualizzare la mappa interna di una fiera nella quale si possano individuare i padiglioni e dentro ad essi gli stand. Questo non risulta possibile utilizzando MapKit in quanto anche se si effettuasse il massimo zoom in all'interno della mappa nella zona della fiera, si potrebbe al più visualizzare l'edificio della fiera come un piccolo rettangolo ma non si avrebbe comunque nessuna informazione sugli interni. Nasce proprio da qui l'idea di realizzare un proprio framework che permetta di utilizzare non solo mappe di Apple come già il framework Mapkit permette, ma di poter gestire anche diverse sorgenti di mappa, quali ad esempio immagini. Tale framework deve quindi riuscire a gestire mappe personalizzate, permetterne la visualizzazione in formato vettoriale e consentire la localizzazione dell'utente.

1.2 Obiettivi della tesi

L'esigenza di analizzare e studiare una metodologia per realizzare mappe vettoriali e poterle poi includere in applicazioni per iOS è nata dalla richiesta sempre maggiore di sviluppare applicazioni in grado di permettere all'utente di localizzarsi all'interno di strutture o spazi aperti quali fiere, parchi divertimento e poter ricercare i luoghi d'interesse all'interno di essa. Proprio dalla richiesta di un'applicazione per la fiera Pulire 2.0 a Verona è partito tale studio. Pulire 2.0 è la più grande fiera in Italia della pulizia professionale, che ospita un'offerta merceologica completa di macchine, prodotti chimici, attrezzature e componentistica per la pulizia professionale e l'igiene degli ambienti. 30 anni di storia al servizio degli operatori fanno di Pulire la seconda manifestazione in Europa per importanza e dimensioni e una delle più importanti a livello mondiale. Obiettivo di Pulire è quello di dar vita ad una fiera interattiva su smartphone e tablet pensata sia per i visitatori che per gli espositori. In uno scenario come questo si rende quindi necessario lo sviluppo di un'applicazione con elevate prestazioni. Durante l'analisi e lo studio delle metodologie si è quindi realizzata un'applicazione per la fiera Pulire 2.0 che rispettasse le specifiche richieste dall'ente. Tuttavia, data la richiesta di diverse applicazioni che includano mappe al loro interno per fiere e parchi divertimento, la ricerca è stata ampliata al fine di poter realizzare mappe vettoriali con funzionalità di geolocalizzazione dell'utente e dei punti di interesse presenti in essa. All'interno del prossimo capitolo verranno descritte in maggior dettaglio le esigenze che hanno portato a tale studio e quali siano le funzionalità che il framework che si vuole realizzare deve fornire all'utente. Il terzo capitolo descrive brevemente le tecnologie utilizzate esponendone vantaggi e svantaggi al fine di comprendere le scelte fatte in fase di progettazione e di sviluppo. Il quarto capitolo è stato suddiviso in due parti: la prima descrive come sia stata progettata e sviluppata l'applicazione per la fiera Pulire 2.0, la quale è stata realizzata sviluppando un proprio framework per la gestione della mappa mentre la seconda descrive la fase di progettazione e di sviluppo del tool per la realizzazione di mappe vettoriali e soprattutto l'operazione di georeferenziazione della mappa. E' stato inserito un capitolo prima delle conclusioni riguardante la fase di testing che mette in luce i problemi riscontrati e le soluzioni individuate in tale fase. Infine la tesi si conclude con il capitolo che descrive le conclusioni su tale studio e i possibili sviluppi futuri di tale progetto.

Chapter 2

Analisi delle esigenze

300 espositori, oltre 10.000 visitatori, oltre 16.000 mq di esposizione. Sono questi i numeri di Pulire, una fiera nata nel 1982 con l'obiettivo di far conoscere al maggior numero di utenti l'esistenza di una produzione nazionale ad alto contenuto tecnologico. A partire da quest'anno Pulire ha deciso di offrire alla propria clientela una nuova fiera: interattiva e tecnologica, al fine di soddisfare le aspettative di una società sempre più connessa. Ad ogni visitatore ed espositore viene fornita un'applicazione per tablet e smartphone che tra le diverse funzionalità permette di visualizzare una mappa dello spazio espositivo in modo che l'utente possa localizzare i punti di interesse all'interno della stessa. In un contesto di 300 espositori, il visitatore deve riuscire a trovare facilmente gli espositori ai quali è potenzialmente interessato attraverso la pratica ricerca suddivisa per ragione sociale, categoria merceologica e marchi rappresentati. Su una superficie di 16.000 mq il visitatore deve potersi orientare in modo facile e veloce, tuttavia bisogna tenere conto che in ambienti come la fiera spesso il segnale gps potrebbe non essere preciso. Nel caso in esame la precisione del segnale gps all'interno della fiera di Verona è pari circa a 60 metri, ed essendo lo stand di dimensioni standard più piccolo di tale distanza, sarebbe inutile usufruire del servizio di geolocalizzazione all'interno della fiera. Si rende quindi necessario lo studio di una soluzione che permetta all'utente di riconoscere all'interno della mappa i luoghi più vicini alla propria posizione e potersi quindi orientare. Definite quindi le specifiche a livello di funzionalità con il cliente, si rende necessaria anche un'analisi delle caratteristiche che l'applicazione deve possedere per soddisfare l'utente durante l'utilizzo della stessa. Per comprenderle torniamo ai numeri, 16000 mq di spazio espositivo devono essere visualizzati su uno schermo da 3,5 a circa 5 pollici per iPhone e iPod e di 9,7 pollici per l'iPad, lo spazio espositivo deve essere interamente visualizzato ma bisogna fornire all'utente la possibilità di ingrandire la mappa effettuando uno zoom su di essa. Lo zoom deve però permettere all'utente di visualizzare una porzione della mappa senza perdere la qualità della mappa originale, altrimenti non riuscirebbe a distinguerne i dettagli. Proprio per questo tale ricerca si è basata sullo studio di mappe vettoriali e non di mappe raster.
Le immagini raster sono formate da un reticolo di pixel,ovvero piccolissime tessere quadrate (come un mosaico), così ravvicinate che non si distinguono l'una dall'altra e viste da distanza sufficiente, danno l'effetto di un'immagine compatta. Il principale difetto di questo tipo di immagini è che esse perdono qualità e dettaglio se vengono ingrandite presentando il tipico effetto "pixellato". Le immagini vettoriali invece sono costituite da vettori matematici, più semplicemente da una serie di tracciati e punti che formano figure geometriche. I principali vantaggi dell'utilizzo della grafica vettoriale rispetto a quella raster sono i seguenti:

- possibilità di ingrandire l'immagine arbitrariamente senza che si verifichi una perdita di risoluzione dell'immagine stessa, come in fig. 2.1, caratteristica indispensabile per visualizzare i dettagli della mappa
- possibilità di esprimere i dati in un formato che occupi (molto) meno spazio rispetto all'equivalente raster, con una riduzione dell'occupazione di RAM e memoria di massa, principalmente nelle forme geometriche o nei riempimenti a tinta piatta. Risulta, inoltre, più facile da gestire e da modificare, essendo minore la quantità di dati coinvolti in ogni singola operazione di aggiornamento. Questo rende il vettoriale particolarmente adatto per gestire grandi quantità

di dati come quelli cartografici che sono tipicamente gestiti in modalità vettoriale.

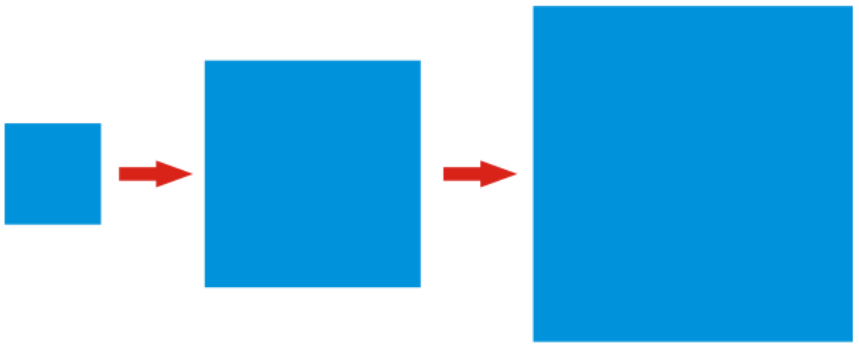

Figure 2.1: In un'immagine vettoriale a prescindere dall'aumento delle dimensioni la qualità rimane invariata

La principale differenza tra le due tipologie di mappa risiede quindi nel fatto che la grafica vettoriale, essendo definita attraverso equazioni matematiche, è indipendente dalla risoluzione, mentre la grafica raster, se viene ingrandita o visualizzata su un dispositivo dotato di una risoluzione maggiore di quella del monitor, perde di definizione. Ad esempio una linea che percorre lo schermo trasversalmente se viene rappresentata utilizzando la grafica raster viene memorizzata come una sequenza di pixel colorati disposti a formare la linea. Se si provasse ad ingrandire una sezione della linea si vedrebbero i singoli pixel che compongono la linea. Se la medesima linea fosse memorizzata in modo vettoriale la linea sarebbe memorizzata come un'equazione che parte da un punto identificato con delle coordinate iniziali e termina in un altro punto definito con delle coordinate finali. Ingrandire una sezione della linea non produrrebbe artefatti visivi o la visualizzazione dei singoli pixel componenti l'immagine, dato che la linea sarebbe visualizzata sempre con la massima risoluzione consentita dal monitor.

Chapter 3

Tecnologie Utilizzate

Benché non ci si proponga di analizzare scrupolosamente le tecnologie utilizzate durante il lavoro di tesi, è d'obbligo darne quantomeno una visione sintetica, che permetta di comprendere le scelte che sono state prese in fase di realizzazione dell'applicazione e che verranno spiegate nel prossimo capitolo. Poiché sono state realizzate due tipologie di applicazioni: per Mac e per iPod, iPad,iPhone verranno presentati i due sistemi operativi utilizzati, rispettivamente OS X e iOS descrivendo per entrambi l'SDK (System Development Kit) ovvero l'insieme di framework e di interfacce messe a disposizione da Apple per la realizzazione di applicazioni su tali dispositivi. L'SDK è quindi diverso a seconda che si stia sviluppando applicazioni per l'una o per l'altra gamma di dispositivi, tuttavia dal punto di vista dello sviluppatore l'ambiente di sviluppo e il linguaggio di programmazione restano gli stessi. L'IDE realizzato da Apple prende il nome di Xcode, mentre il linguaggio utilizzato è l'Objective C.

Xcode

Xcode è l'ambiente di sviluppo utilizzato per creare, testare, effettuare il debug della propria applicazione. Esso comprende l'applicazione Xcode, la quale si comporta da involucro per tutti gli altri strumenti necessari allo sviluppatore per realizzare le proprie applicazioni, quali ad esempio il simulatore iOS e il tool Instruments.

Le principali caratteristiche di tale ambiente di sviluppo sono:

- **Interfaccia a singola finestra**. Tale caratteristica permette di sviluppare diversi flussi di lavoro in un'unica finestra. Inoltre è possibile personalizzare la visualizzazione dei pannelli laterali dell'unica finestra nascondendoli o visualizzandoli a seconda delle esigenze.

- **Interfaccia Utente realizzabile graficamente**. Grazie a tale funzionalità è possibile costruire il design dell'interfaccia grafica relativa alla propria applicazione utilizzando un apposito editor d'interfaccia fornito da Xcode in modo grafico senza la necessità di scrivere codice. Si possono specificare molti dettagli dell'interfaccia grafica come il layout dei controlli di interfaccia e la loro connessione con la propria applicazione e la gestione dei dati.Tuttavia è bene precisare che la presenza di tale editor di interfaccia non esclude la possibilità di creare elementi dell'interfaccia (bottoni, barre di navigazione, ecc.) anche da codice. L'implementazione da codice viene maggiormente utilizzata quando si vogliono personalizzare gli elementi dell'interfaccia.

- **Assistente di editing**. Durante la fase di programmazione si è spesso costretti a lavorare su differenti parti della stessa componente, come ad esempio il layout dell'interfaccia utente e l'implementazione delle funzionalità di tale interfaccia. Questa caratteristica di Xcode permette di accedere ai contenuti di cui il programmatore necessita nel momento in cui ne deve fare uso. Ad esempio, se si sta lavorando all'implementazione di un file nell'editor principale, Xcode può aprire il corrispondente file di interfaccia in un pannello secondario dell'editor principale, senza il bisogno di aprire più editor contemporaneamente.

- **Identificazione automatica e correzione dell'errore**. Xcode verifica il codice sorgente mentre viene digitato, quando nota un'errore, l'editor evidenzia la riga di codice errata. Vengono inoltre forniti i dettagli relativi all'errore e in alcuni casi anche le possibili soluzioni per risolverli.
- **Controllo del Codice Sorgente**. Tale caratteristica permette di salvaguardare i propri file di progetto salvandoli direttamente su di un repository come Git.

Linguaggio utilizzato:Objective-C

Il linguaggio Objective-C nasce a metà degli anni '80 presso la StepStone Corporation dall'idea di Brad Cox e Tim Love di aggiungere al linguaggio C le caratteristiche dell'allora primo linguaggio ad oggetti: SmallTalk. Il rilascio dell'Objective-C avvenne nel 1986 ma non ebbe molto successo fino a quando Steve Jobs, dopo aver lasciato Apple, fondò la NeXT Computer e lo adottò come ambiente di sviluppo per il sistema operativo NeXTSTEP. Steve jobs aveva visto bene infatti oggi, grazie all'eredità di NeXTSTEP, l'Objective-C viene utilizzato per sviluppare sulla piattaforma MacOS X e iOS. Tale linguaggio, come indica il suo nome, aggiunge al C proprio gli oggetti: gli oggetti associano una porzione di dati ad una particolare operazione che può utilizzare o modificare tali dati. In Onjective-C queste operazioni sono note come metodi dell'oggetto;la porzione di dati su cui agiscono le variabili d'istanza. Riassumendo, un oggetto impacchetta una struttura dati (variabili d'istanza) ed un insieme di procedure (i metodi) all'interno di una ben definita unità di programma. Ad esempio, si supponga di dover realizzare un'applicazione che permette all'utente di disegnare immagini come linee, cerchi, rettangoli, testo, immagini bit-map e così via, creando una classe per ognuna delle forma base si crea l'oggetto in modo che l'utente possa poi manipolare le figure che desidera disegnare. Un oggetto di tipo Rettangolo, ad esempio, potrebbe avere delle variabili d'istanza che definiscono la posizione del rettangolo, la sua larghezza ed altezza oppure potrebbero indicare se è colorato, se si quale colore, ecc. Tale rettangolo potrebbe possedere anche dei metodi che servano ad esempio per impostare la posizione, le sue dimensioni, il colore, se è colorato, per definire la linea che serve a visualizzarlo. In Objective-C le variabili d'istanza degli oggetti sono interne all'oggetto: lo sviluppatore può accedervi solamente utilizzando i metodi definiti nella classe che rappresenta l'oggetto. Proprio come nel C, anche qui le variabili locali vengono nascoste dal resto del programma, un oggetto nasconde infatti sia le sue variabili d'istanza che le implementazioni dei suoi metodi.

Modello MVC

Una qualsiasi applicazione per iOS o per OS X che utilizza gli strumenti messi a disposizione dai vari framework, se progettata correttamente, è basata su un modello detto Model View Controller (MVC). MVC è un vero e proprio pattern architetturale creato negli anni '80 per consentire la presentazione multipla (in diverse forme) di un oggetto in varie interfacce grafiche senza dover modificare o adattare l'oggetto da presentare. Infatti MVC è una applicazione del pattern Observer alle interfacce utente che non solo realizza la separazione tra dati e interfaccia ma svincola anche la logica di controllo dell'applicazione dai dati. Il modello è basato sulla separazione dei compiti fra tre componenti software che interpretano tre ruoli principali:

- **Model**
  gli oggetti che costituiscono il modello rappresentano particolari conoscenze e competenze, contengono i dati di un'applicazione e definiscono la logica che regola la manipolazione dei dati. Un'applicazione MVC ben progettata ha tutti i dati importanti incapsulati in oggetti del modello. Tutti i dati che fanno parte dello stato persistente dell'applicazione devono risiedere negli oggetti del modello una volta che i dati vengono caricati nell'applicazione. Idealmente, un oggetto del modello non ha alcun collegamento esplicito con l'interfaccia utente utilizzata per presentarlo e modificarlo. Tuttavia nella pratica la separazione dall'interfaccia non è sempre la cosa migliore, infatti vi è un certo margine di flessibilità nel modello realizzato in iOS, ma in generale un oggetto del modello non dovrebbe occuparsi di come è realizzata l'interfaccia.

- **View**
  Un oggetto di tipo vista è in grado di visualizzare, e in alcuni casi di modificare, i dati del modello dell'applicazione. La view non dovrebbe essere responsabile per la memorizzazione dei dati che presenta, ma può memorizzare nella cache alcuni dati per motivi di prestazioni. Un oggetto della vista può essere responsabile della visualizzazione di solo una parte di un oggetto del modello, di un oggetto intero del modello o anche di molti oggetti del modello differenti. Le view sono disponibili in molte varietà diverse e tendono ad essere riutilizzabili e configurabili fornendo coerenza tra applicazioni diverse nello stesso sistema. Infatti il framework UIKit definisce un gran numero di oggetti di visualizzazione che possono essere riutilizzati assicurando lo stesso funzionamento in diverse applicazioni, garantendo un elevato livello di coerenza per aspetto e comportamento tra le applicazioni. Un oggetto di visualizzazione deve assicurarsi che i dati del modello siano visualizzati correttamente, di conseguenza ha bisogno di conoscere le modifiche apportate al modello.

- **Controller**
  Poiché gli oggetti del modello non devono essere legati ad interfacce specifiche, hanno bisogno di un modo generico per segnalare il cambiamento. Un oggetto di controllo funge da intermediario tra gli oggetti della view dell'applicazione e gli oggetti del modello. I controller hanno spesso il compito di fare in modo che la view abbia accesso agli oggetti del modello che devono presentare e quello di agire come canale attraverso il quale l'interfaccia sia aggiorna secondo le modifiche dei dati. Gli oggetti del controller possono anche configurare e coordinare le attività di una applicazione e gestire i cicli di vita di altri oggetti. In un tipico modello MVC, quando gli utenti immettono un valore o indicano una scelta attraverso un oggetto di visualizzazione, il valore o la scelta viene comunicato a un oggetto di controllo. L'oggetto di controllo può interpretare l'input dell'utente in alcune applicazioni specifiche o può indicare ad un oggetto del modello cosa fare con questo ingresso. Sulla base dello stesso input dell'utente alcuni oggetti del controller potrebbero anche indicare ad un oggetto di visualizzazione di modificare il proprio aspetto o il proprio comportamento. Al contrario, quando un oggetto del modello cambia, il modello comunica il cambiamento al controller, che penserà ad aggiornare di conseguenza gli oggetti di visualizzazione.

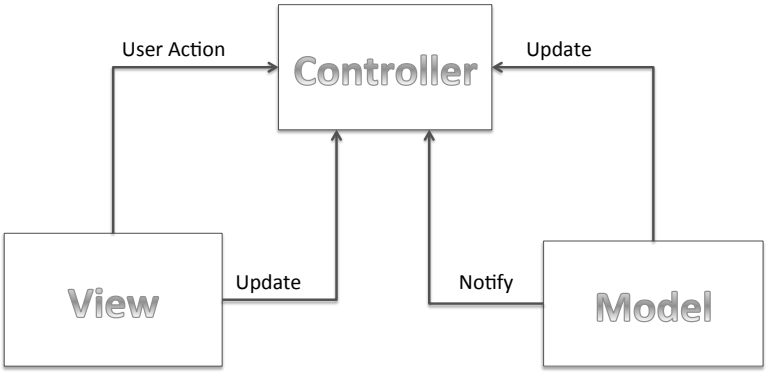

Figure 3.1: Paradigma MVC

Il modello MVC presenta alcuni aspetti positivi e negativi per lo sviluppo software. Le applicazioni progettate secondo questo modello godono di una separazione tra dati ed interfaccia molto solida essendo realizzata a livello progettuale, inoltre la separazione della logica dell'applicazione nel controllore garantisce una facile espandibilità e mantenibilità dell'applicazione. Il modello risulta vantaggioso anche nel caso in cui si sviluppino più applicazioni con elementi comuni perché la separazione dei compiti in moduli di tre tipi permette una grande riusabilità delle classi. D'altra parte, il modello risulta svantaggioso nel caso in cui non ci sia la necessità di riutilizzare parti del programma, infatti l'overhead dovuto alla progettazione piuttosto complessa e alla necessità di creare classi aggiuntive rispetto a quelle richieste da modelli più semplici può essere molto grande.

Property List File

Un file di estensione property list è una rappresentazione di una gerarchia di oggetti, una sorta di grafo, che permette all'applicazione di immagazzinare piccole quantità di dati all' interno del file system. Come da fig. 3.2, i dati all'interno del file non sono inseriti in maniera casuale ma al contrario devono possedere una struttura ben definita, ottenuta utilizzando soltanto alcuni tipi di dato: dizionari, arrays, stringhe, numeri (interi e float), date, dati binari e valori booleani. I dizionari e gli array sono tipi speciali in quanto essendo collezioni di oggetti possono contenere uno o molti tipi di dato, inclusi anche dizionari e array, permettono di creare quella gerarchia che caratterizza i file plist. Questa gerarchia creata incapsulando array e dizionari permette di ottenere un grafo di oggetti. Ogni volta che si crea un nuovo progetto, di tipo iOS od OS X, l'IDE Xcode crea di default un file info.plist, un file strutturato, che contiene le informazioni di configurazione per l'applicazione. E' tuttavia possibile creare dei file plist propri da aggiungere all'applicazione, da codice è sufficiente creare la struttura desiderata e generare poi il file plist sul quale scrivere tale struttura, oppure da Xcode è sufficiente aggiungere un nuovo file con estensione plist ed inserire manualmente i dati specificando la struttura degli oggetti. Il principale vantaggio della creazione di file plist propri risiede nella portabilità, infatti tali file possono essere scritti da un'applicazione e letti o modificati da un'altra. Inoltre lettura e scrittura non vanno ad aumentare il tempo di computazione dell'applicazione poiché grazie alla struttura del grafo si può accedere agli oggetti velocemente. Bisogna comunque tenere conto che al momento di accedere al file plist l'applicazione carica tutto il file in memoria e se i dati contenuti nel file sono molti l'applicazione potrebbe consumare gran parte della memoria del dispositivo aumentando il tempo di risposta dell'applicazione. Risulta quindi opportuno accedere al file il minor numero di volte salvando i dati in una struttura dati locale e scrivendoli poi interamente sul file un'unica volta invece che utilizzare più accessi, analogamente in fase di lettura sarà conveniente accedere ai dati nel momento in cui l'applicazione viene caricata e poi salvare tali informazioni in una struttura locale alla quale poter accedere in qualunque momento.

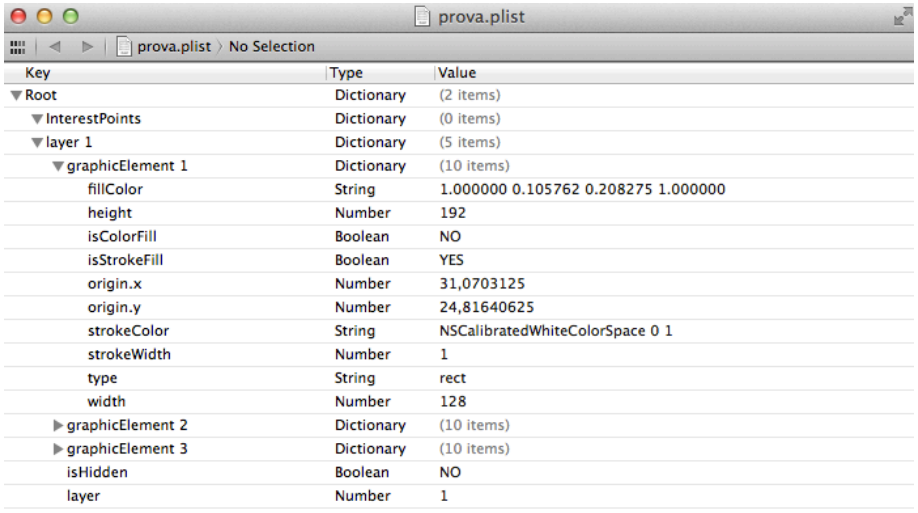

Figure 3.2: Esempio di gerarchia di oggetti all'interno di un file plist

Come verrà spiegato nel capitolo successivo, tale file sarà utilizzato per scambiare informazioni riguardanti la mappa tra l'applicazione realizzata in OS X e il framework iOS realizzato sfruttando appieno la principale caratteristica di tale estensione di file: la portabilità.

3.1 iOS

Si tratta di un sistema operativo sviluppato da Apple ed attualmente disponibile su tutti i dispositivi *mobile* dell'azienda di Cupertino: iPhone, iPad e iPod. E' possibile realizzare applicazioni

compatibili con iOS utilizzando l'iOS SDK messo a disposizione da Apple.

3.1.1 iOS SDK

L'iOS Software Development Kit (SDK) è un kit che offre al programmatore un insieme di strumenti ed interfacce necessari per sviluppare, installare, eseguire e testare le applicazioni native realizzate. Questa tipologia di applicazione viene detta nativa in quanto si utilizzano il framework del sistema iOS e il linguaggio Objective-C per svilupparle e vengono eseguite direttamente su iOS, senza l'utilizzo di virtualizzazioni o middleware tra l'applicazione e il sistema operativo. Diversamente dalle applicazioni web, le applicazioni native vengono installate fisicamente su un dispositivo e sono quindi sempre a disposizione dell'utente, anche quando il dispositivo è disconnesso sia dalla rete internet che dalla rete telefonica. Tali applicazioni risiedono accanto alle altre applicazioni di sistema e sia l'applicazione che tutti i dati utente sono sincronizzati al computer dell'utente tramite l'applicazione desktop iTunes. Oltre alle applicazioni native, è possibile creare applicazioni web utilizzando una combinazione di HTML, CSS e JavaScript. Tali applicazioni vengono eseguite all'interno del browser Safari e richiedono quindi una connessione alla rete internet per accedere al proprio web server, al contrario di quanto accade per le applicazioni native. Quindi poiché le applicazioni native si basano sul framework iOS, è necessario comprendere le tecnologie e gli strumenti che fanno parte di questo SDK in modo da effettuare le migliori scelte in fase di progettazione ed implementazione delle applicazioni e dei framework che li supportano.

3.1.2 Interfacce di iOS

Apple mette a disposizione dello sviluppatore la maggior parte delle sue interfacce di sistema racchiudendole in pacchetti speciali chiamati framework. Un framework è una directory all'interno della quale si trovano: una libreria dinamica condivisa e le risorse necessarie per il suo utilizzo, quali ad esempio file di intestazione, immagini, applicazioni di supporto, ecc. Lo sviluppatore può utilizzare i frameworks semplicemente collegandoli al progetto dell'applicazione proprio come farebbe con qualsiasi altra libreria condivisa. Tale collegamento permette di accedere alle funzioni descritte nella libreria e permette agli strumenti di sviluppo di sapere dove trovare i file di intestazione e le risorse contenute nel framework.

3.1.3 Architettura di iOS

L'architettura del sistema operativo iOS è composta da diversi livelli sovrapposti. A livello più alto iOS si comporta come un intermediario tra il sottostante livello, quello relativo all' hardware e il sovrastante, relativo all'applicazione che appare sullo schermo. La comunicazione, almeno per quanto riguarda le applicazioni native, raramente viene effettuata in modo diretto tra l'hardware e l'applicazione stessa in quanto utilizzare un ben definito sistema di interfacce (iOS Framework) permette al programmatore di non doversi preoccupare delle differenze di componenti, a livello hardware, che caratterizzano l'insieme di dispositivi mobili per i quali vengono sviluppate tali applicazioni. L'utilizzo di questo intermediario permette quindi di creare applicazioni che funzionino in maniera uniforme e stabile su dispositivi che possiedono diverse caratteristiche hardware. Tuttavia a livello più basso l'implementazione delle tecnologie che compongono iOS può essere pensata come un insieme di livelli, mostrati in figura 3.3. Ai livelli più bassi del sistema si trovano i servizi fondamentali e le tecnologie sulle quali si basano tutte le applicazioni mentre i livelli più alti contengono i servizi e le tecnologie più sofisticate. Proprio perché i livelli più alti comprendono le tecnologie più sofisticate, in fase di sviluppo di un'applicazione, sarebbe preferibile prediligere l'uso di framework di alto livello in quanto creati appositamente per fornire astrazioni orientate agli oggetti dei servizi e delle funzioni offerti dai costrutti dei livelli inferiori. Tali astrazioni, proprie dei livelli superiori, rendono la scrittura di codice più facile in quanto riducono la quantità di righe di codice da scrivere e racchiudono caratteristiche potenzialmente complesse come i socket per le connesioni e i thread. Non bisogna tuttavia pensare che tali livelli superiori si sovrappongano a quelli inferiori nascondendo questi ultimi allo sviluppatore, infatti i frameworks di livello inferiore sono a disposizione degli sviluppatori che preferiscono usarli o che vogliono utilizzare aspetti di tale strutture non esposti dagli strati superiori.

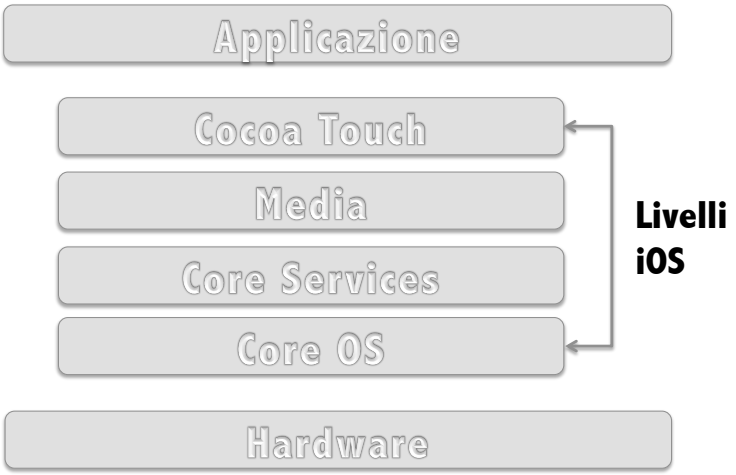

Figure 3.3: Architettura a livelli del sistema operativo iOS

Cocoa Touch Layer

Il livello Cocoa Touch si trova sulla sommità dello stack iOS e contiene i frameworks che vengono più comunemente utilizzati dagli sviluppatori di applicazioni iOS, come il multitasking, l'input basato sul tocco, le notifiche push, e molti servizi di alto livello del sistema. A partire dalla versione 4.0 di iOS le applicazioni realizzate utilizzando tale kit di sviluppo non vengono terminate quando l'utente preme il tasto home, come invece avveniva per le applicazioni realizzate con versioni precedenti, ma vengono eseguite in background. La gestione del passaggio da applicazione attiva all'esecuzione in background è lasciata al multitasking, supporto definito da UIKit. Se le applicazione messe in stato di background continuassero a restare attive, il consumo di batteria sarebbe elevato, quindi al fine di preservare la durata della batteria del dispositivo la maggior parte delle applicazioni viene sospesa dal sistema dopo il passaggio a stato di background. La sospensione dell'esecuzione vuol dire che l'applicazione non esegue alcun codice anche se rimane in memoria. Resta in memoria ma non esegue, è come se fosse "congelata" al momento in cui è stata sospesa, in questo modo può riprendere velocemente e senza consumare batteria se viene rilanciata. Non sempre però le applicazioni eseguono uno stato di sospensione se messe in background, ma continuano ad eseguire. I principali motivi per cui non vengono sospese dal sistema sono:

- L'applicazione richiede una quantità limitata di tempo per terminare un'operazione importante
- L'applicazione potrebbe sostenere servizi specifici, i quali richiedono un tempo di durata fissa e intervalli regolari di esecuzione in background
- L'applicazione può utilizzare notifiche locali per generare avvisi all'utente se l'applicazione è in esecuzione

E' importante sottolineare che, indipendentemente da come si comporti l'applicazione quando si trova in background, il supporto multitasking non richiede lavoro aggiuntivo da parte del programmatore. E' infatti il sistema ad inviare delle notifiche all'applicazione per indicarle se avviene un passaggio di stato da attivo a background e viceversa, ed è poi l'applicazione stessa che in base a tali notifiche effettua il salvataggio dei dati utente. Tra le principali funzionalità messe a disposizione a partire da iOS 5 per migliorare la gestione degli elementi e la loro interconnessione si trovano: layout automatico e storyboard. Introdotto con iOS 6, layout automatico migliora il modello precedentemente utilizzato per disporre gli elementi di una interfaccia utente, infatti diventa possibile definire delle regole in modo più intuitivo e con un maggior numero di relazioni disponibili per la disposizione degli elementi. Le relazioni tra gli elementi sono oggetti Objective-C chiamati vincoli. Un oggetto, solitamente, ha informazioni più dettagliate in merito alla sua dimensione normale, il suo posizionamento all'interno della superview, e il suo posizionamento rispetto al suo punto di

vista di pari livello. Tuttavia nel caso in cui si abbaia la necessità di personalizzare tali vincoli, un controller può sostituire questi valori con valori non standard. Storyboard, introdotto con iOS 5, costituisce il nuovo metodo consigliato per progettare l'interfaccia utente dell'applicazione. A differenza dei file .xib precedentemente usati, questa nuova tecnologia permette di progettare l'intera interfaccia utente all'interno di un unico ambiente grafico in modo da poter visualizzare tutte le view, i controller e soprattutto le relazioni tra essi. Le relazioni tra le view, ovvero gli effetti di transizione che permettono il passaggio da una view ad un'altra possono essere definite tramite un comodo editor grafico integrato nell'IDE Xcode oppure manualmente da codice. In questo modo è possibile controllare non solo il contenuto ma anche il flusso dell'interfaccia utente. E' possibile utilizzare un unico storyboard oppure utilizzarne diversi per poter gestire le varie parti dell'interfaccia utente. La gestione degli storyboard da parte di Xcode avviene in fase di compilazione: il contenuto dello storyboard viene suddiviso in parti distinte che possono essere caricate singolarmente al fine di ottenere prestazioni migliori. L'applicazione non si interfaccia direttamente con le singole parti in cui è stato suddiviso lo storyboard ma ha a disposizione varie classi della libreria UIKit per accedere a tali contenuti da codice. Tale livello mette a disposizione svariati frameworks, tra questi verranno descritti in maggior dettaglio quelli che sono stati maggiormente utilizzati in fase di progettazione e sviluppo:

1. **Core Location e Map Kit Framework**

Figure 3.4: Visualizzazione della mappa, annotations e callout

Ogni dispositivo iOS possiede la capacità di determinare in quale luogo del mondo si trovi, grazie all'utilizzo del framework *Core Location*. iOS include inoltre un altro framework, *Map Kit*, il quale permette di creare facilmente mappe interattive che possono mostrare qualunque luogo, incluso ovviamente quello in cui si trova l'utente. Il primo dei due framework utilizza per l'individuazione della posizione dell'utente:

- *GPS*, tra le tre tecnologie la più accurata, in quanto riceve segnali ad onde da diversi satelliti per determinare la posizione corrente ma non è disponibile sulla prima generazione di iPhone, iPod touch o iPad che dispongono della sola connessione Wi-Fi.
- *cell ID location*, fornisce un'approssimazione della posizione corrente dell'utente basata sulla posizione fisica del dispositivo rilevata dalla cella telefonica alla quale il dispositivo si è attaccato l'ultima volta. Tuttavia poiché ogni cella ricopre una vasta area, il margine di errore nel rilevamento della posizione risulta quindi maggiore. Inoltre poiché tale tecnologia richiede una connessione radio alla cella, gli unici dispositivi che possono utilizzare tale tecnologia risultano essere l'iPhone e gli iPad dotati di una connessione 3G.

- *Wi-Fi Positioning Service (WPS)* , utilizza il MAC address del dispositivo per accedere al più vicino access point Wi-Fi e poter così indovinare la propria posizione facendo riferimento ad un database che contiene gli indirizzi noti dei server providers e le loro aree di servizio. Tale tecnologia risulta imprecisa e l'errore può essere anche di molti chilometri.

E' importante tenere presente, in fase programmazione, che tutte le tre modalità proposte richiedono un'enorme consumo di batteria quindi non bisogna richiedere la posizione più di quanto non sia strettamente necessario. Inoltre il framework *Core Location* permette di specificare l'accuratezza desiderata per la rilevazione della posizione, settando tale parametro al minimo non è possibile prevenire un utilizzo massiccio della batteria. Tali tecnologie sono comunque nascoste dall'applicazione, in questo modo non è necessario dire al framework quale metodologia utilizzare ma basta specificare l'accuratezza che si desidera ottenere in fase di rilevazione e il framework stesso in base a tale parametro decide quale tecnologia soddisfa meglio le specifiche.

Grazie a tale framework si ottengono le coordinate relative alla posizione dell'utente ma per visualizzarle è necessario utilizzare una mappa. A tale scopo Apple ha introdotto a partire da iOS 3.0 il framework *Map Kit*, il quale utilizza alcuni servizi di back-end di cui usufruisce anche l'applicazione Maps di Apple per la visualizzazione di mappe. Questo permette di affermare che tali servizi siano robusti in quanto vengono messi alla prova spesso all'interno delle app native, testate ogni giorno da milioni di utenti. La struttura del framework è composta da una classe principale, MKMapView, che serve a rappresentare la mappa e visualizzarla sullo screen in modo che sia interattiva, come ci si aspetterebbe da una qualunque applicazione moderna sulle mappe. È possibile utilizzare questa mappa per fornire indicazioni o punti di interesse. Tale classe è stata realizzata in modo tale da fornire funzioni per visualizzare una mappa, permettere lo scrolling e lo zoom attraverso il pinch gesture, annotare una mappa con immagini personalizzate oppure contenuti, aggiungere overlay e molto altro ancora. In iOS 4.0, la visualizzazione della mappa di base ha ottenuto il supporto per le annotazioni trascinabili e i moduli personalizzati. Le annotazioni trascinabili consentono di riposizionare una annotazione, a livello di codice o tramite interazioni con l'utente, dopo che è stata posta sulla mappa. Da iOS 6.0 è possibile creare una applicazione di routing, il cui compito è quello di fornire indicazioni agli utenti. Quando l'utente richiede indicazioni relative alla navigazione, l'applicazione Mappe consente ora all'utente di scegliere l'applicazione da cui ricevere quelle direzioni. Inoltre, tutte le applicazioni possono chiedere l'applicazione Mappe per fornire indicazioni stradali e la visualizzazione di più punti d'interesse. E' possibile inoltre impostare da codice quale tipologia di mappa si voglia visualizzare: ibrida, satellitare oppure standard. I dati relativi alla mappa da mostrare venivano forniti dal servizio Google Mobile Maps (GMM) fino ad iOS 5, da iOS 6 in poi Apple utilizza invece mappe proprie. Grazie agli aggiornamenti continui le Mappe di Apple utilizzate a partire da iOS 6 hanno risolto diversi problemi e ora sono considerate, almeno in USA, migliori di quelle di Google. Gli sviluppatori di app hanno confrontato le API di MapKit di Apple con la nuova Google Maps SDK: anche se rimangono diverse funzioni da migliorare le API della Mela sono risultate quelle preferite e consigliate per lo sviluppo di terze parti. Servendosi quindi di mappe proprie, MapKit non permette ancora al programmatore di utilizzare mappe diverse da quelle fornite. in fig. 3.5 è riportata la struttura della classe MkMapView, essa è conforme alla classe NSObject in quanto fornisce un delegato il quale permette di gestire ulteriori metodi che si occupano del caricamento della mappa, di individuare la zona della mappa da visualizzare nel caso in cui le coordinate del punto centrale cambino e gestire le annotazioni della mappa. Poiché permette di creare un delegato, la classe MKMapView è conforme anche alla classe UIResponder, la quale definisce un interfaccia per gli oggetti che rispondono e gestiscono alcuni eventi quali ad esempio il pinch gesture sulla mappa. Infine essendo una sottoclasse di una UIView pertanto fornisce molti degli elementi comuni di una view, quali ad esempio bottoni e label.

2. **UIKit Framework**
   UIKit è sicuramente il più utilizzato ed il più grande framework dell'iOS SDK in quanto è il responsabile di tutte le funzioni dell'interfaccia utente, dalla creazione delle finestre ai

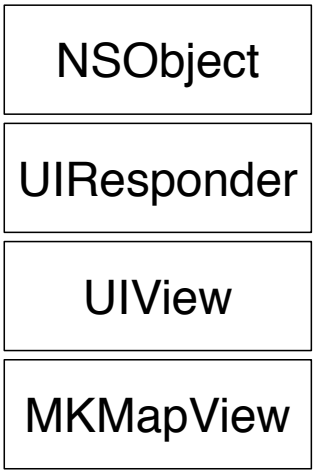

Figure 3.5: Struttura di MKMapView

minimi componenti dell'interfaccia fino alla gestione degli input. UiKit fornisce quindi allo sviluppatore un'insieme di classi necessarie per costruire e gestire l'interfaccia utente. Esso fornisce l'oggetto applicazione, la gestione degli eventi, finestre, viste e controlli appositamente progettati per l'interfaccia touch screen. In un dispositivo touch, altra funzionalità indispensabile ed introdotta a partire da iOS 3.2, è il riconoscimento di gesti come "swipe" e "pinch" oppure il semplice tocco. Riconosciuto il gesto, è possibile collegare ad esso un'azione da eseguire quando si verifica il gesto, si pensi ad esempio all'azione di zoom causata dal gesto di "pinch". Il compito di riconoscimento viene affidato ad un particolare oggetto che registra gli eventi di tocco ed applica le euristiche definite dal sistema per riconoscere il tipo di tocco effettuato. Nel caso in cui vi siano più view sovrapposte è anche possibile riconoscere su quale view sia avvenuto il touch in base alla priorità della view. UIKit include una classe UIGestureRecognizer che definisce il comportamento di base per tutti gli oggetti di riconoscimento gesti. È possibile tuttavia definire sottoclassi personalizzate o utilizzare una delle sottoclassi definite in UIKit per gestire qualsiasi dei seguenti gesti standard:

- Single tap o double tap (Tocco singolo o qualsiasi numero di tocchi successivi)
- Pinch In e Out (Zoom In e Zoom Out)
- Panning (Trascinamento)
- Swipe (in qualsiasi direzione da destra a sinistra o viceversa)
- Rotazione (le dita in movimento in direzioni opposte attorno ad un punto)
- Pressione Prolungata

In fig. 3.6 viene presentata la gerarchia delle classi di questo framework. Tale schema fornisce una visione completa dell'ereditarietà applicata a questo framework in quanto permette di individuare la superclasse dell'oggetto che si desidera utilizzare. Ad esempio, è immediato scoprire che un oggetto UIButton deriva direttamente dalla sua superclasse UIControl, che a sua volta eredita dalla classe UIView che deriva da UIResponder che infine ha come superclasse la classe radice (root class) da cui derivano tutti gli oggetti dell'UIKit, ovvero NSObject. Al fine di comprendere meglio i termini che verranno poi utilizzati nei prossimi capitoli per descrivere la fase di progettazione e sviluppo delle applicazioni, verrà di seguito fornita una descrizione di alcuni degli elementi principali che compongono tale framework.

- *UIView*
  Una view è l'area rettangolare visibile sullo schermo del dispositivo, essa rappresenta un contenitore per altri oggetti (bottoni, campi di testo, slider, ecc.) e permette di gestire

il touch dell'utente sullo schermo. Le view hanno una struttura gerarchica, esiste una superview che può contenere diverse subviews che vengono visualizzate sopra alla view padre.

- *UIWindow*
  E' una sottoclasse di UIView. Ogni applicazione possiede una sola window (finestra) , una finestra è uno spazio geometrico sullo schermo, che viene creata automaticamente quando l'applicazione viene eseguita.
- *TextView e ImageView*
  Si tratta di semplici classi derivate dalla UIView che permettono di visualizzare rispettivamente del testo modificabile e delle immagini.
- *Barre di Navigazione*
  La barra di navigazione permette all'utente di passare da una view all'altra, seguendo il flusso dell'interfaccia utente definito dal programmatore. In questo modo l'utente visualizza le varie schermate come fossero pagine di un libro e utilizza la barra di navigazione per cambiare view oppure per attivare particolari funzioni della view corrente.
- *Alert Sheets*
  Il ruolo fondamentale di tali oggetti è quello di notificare all'utente l'avvenimento di un qualche evento oppure chiedere conferma di qualche operazione tramite la comparsa di una finestra al centro dello schermo.
- *Tabelle*
  Tali oggetti permettono di visualizzare lite di files, messaggi o altri tipi di collezioni. Si può rendere possibile la selezione di uno o più elementi della lista, la rimozione oppure l'aggiunta di nuovi elementi. Sono quindi oggetti molto flessibili che permettono allo sviluppatore di definire come ogni cella debba essere rappresentata e definirne il comportamento.
- *Barre di stato*
  La barra di stato è un elemento che fa parte del sistema operativo e compare nel lato più alto dello schermo, sulla quale è possibile visualizzare elementi come l'ora, la potenza del segnale, la potenza della rete, la carica della batteria. Tale barra è modificabile nella posizione e nell'aspetto grafico ed è possibile aggiungere immagini alla barra per notificare qualche evento oppure qualche operazione in corso.
- *Classe UIControl*
  Permette di gestire le seguenti componenti all'interno della view:
  - *UIButton*
    Semplici bottoni
  - *UITextField*
    Campi di testo modificabili
  - *UISwitch*
    Una componente che permette di definire due stati diversi ON-OFF
  - *UISlider*
    Una componente che permette di cambiare un valore numerico semplicemente spostando verso destra o verso sinistra una rotella su di una linea
  - *UISegmentedControl*
    Una componente che permette di definire diverse azioni all'interno di un'unica area a seconda della zona in cui si clicca

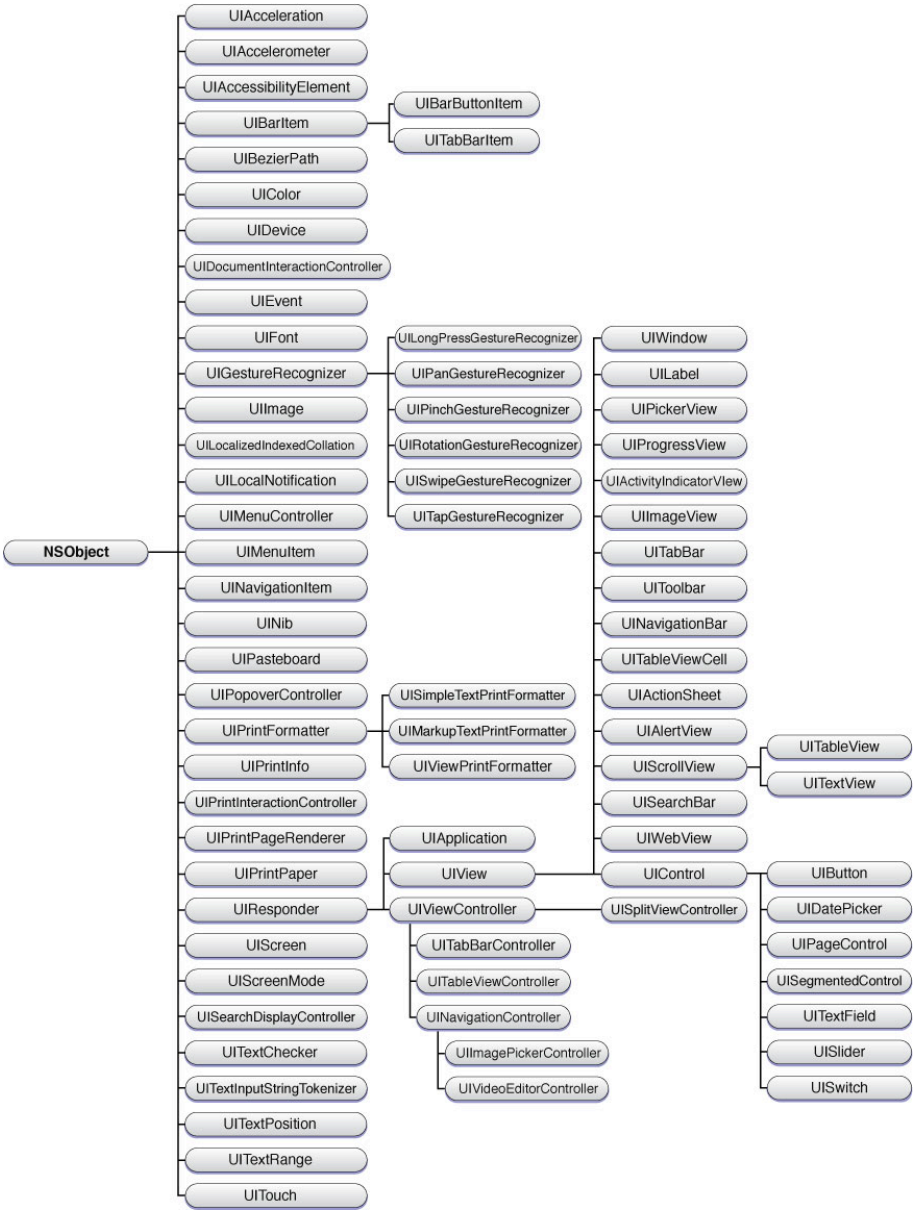

Figure 3.6: Struttura del framework UIKit

Media Layer

Il ruolo di tale livello è quello di fornire tecnologie di tipo audio, video, animazioni e grafica all'applicazione al fine di creare la migliore esperienza multimediale possibile su un dispositivo mobile. Le tecnologie offerte dal Media Layer sono state progettate per rendere più facile la creazione e la gestione degli elementi multimediali all'interno di un'applicazione nativa. Le tecnologie audio fornite da tale livello sono state progettate per aiutare a fornire una ricca esperienza audio agli utenti. Si parla di ricca esperienza in quanto vi è la possibilità di riprodurre e registrare audio di alta qualità. Per quanto riguarda la riproduzione di contenuti video, vengono fornite diverse tecnologie al programmatore, che può utilizzare quella più consona alle proprie esigenze. Alcune di queste permettono, sui dispositivi con l'hardware appropriato, di registrare video ed incorporarli nell'applicazione. Nella scelta della tecnologia da utilizzare bisogna ricordarsi che i frameworks di livello superiore possono semplificare notevolmente il lavoro per supportare le più comuni funzionalità. Come già detto, creare applicazioni che possiedono una elevata qualità a

livello di grafica, permette di essere poi più competitivi sul mercato. Il più semplice ed efficace metodo per la realizzazione di un'applicazione è quello di utilizzare le immagini prerenderizzate, gli elementi standard ed i controlli del framework UIKit e lasciare al sistema il compito di effettuare il rendering dell'interfaccia. In questo modo è possibile ottenere applicazioni standard con una grafica semplice in linea con le applicazioni native realizzate da Apple. Tuttavia se si desidera ottenere applicazioni con una grafica personalizzata è possibile utilizzare le seguenti tecnologie per la gestione dei contenuti grafici dell'applicazione. E' importante sottolineare che tale livello mette a disposizione svariati frameworks per lo sviluppo, tra questi verranno descritti in maggior dettaglio quelli che sono stati utilizzati in fase di progettazione e sviluppo:

1. **Core Graphics Framework**
   Tale framework, noto anche con il nome di Quartz 2D, gestisce il rendering nativo vettoriale 2D. Quartz è lo stesso motore di disegno vettoriale che viene utilizzato da OS X. Tale motore offre il supporto per il disegno basato sul percorso, rendering con anti-aliasing, gradienti, immagini, colorazione, trasformazioni tra coordinate spaziali e la completa gestione di documenti PDF: creazione, visualizzazione e parsing.

2. **Core Animation Framework**
   Fornisce supporto per realizzare animazioni grafiche di elementi, immagini ed altri contenuti. Core Animation, sebbene sia contenuto nel QuartzCore framework, è integrato in molti altri framework di iOS, comprese le classi di UIKit come UIView, offrendo animazioni per molti comportamenti standard di sistema (ad esempio si pensi alle animazioni che si possono effettuare tra la visualizzazione di una view e la successiva). Oltre ad utilizzare le animazioni standard offerte dal motore Quartz è inoltre possibile utilizzare l'interfaccia Objective-C per creare animazioni personalizzate. Si possono quindi realizzare interfacce utente dinamiche senza che le prestazioni dell'applicazione ne risentano, come accadrebbe utilizzando API grafiche di basso livello come OpenGL.

3. **Core Image Framework**
   Il ruolo principale di tale framework è di permettere la gestione avanzata di video ed immagini. Vengono messi inoltre a disposizione filtri per operazioni semplici quali ad esempio ritocco e correzione di foto oppure per la gestione di operazioni più avanzate come il riconoscimento facciale e delle features. Il principale vantaggio dovuto all'utilizzo di tali filtri risiede nel fatto che tali filtri non modificano mai direttamente le immagini di partenza, ma lavorano su una copia di esse, preservando così lo stato delle immagini originali. Inoltre la velocità e l'efficienza di tali operazioni è dovuta all'utilizzo della CPU disponibile e della potenza di elaborazione della GPU da parte del framework.

4. **Core Text Framework**
   Tale framework viene utilizzato per la gestione del layout ed il rendering di testo all'interno dell'applicazione. Il motore di layout di Core Text è stato pensato per effettuare le operazioni di disposizione del testo in maniera semplice. Per quanto riguarda i font, tale framework è progettato per la gestire i font Unicode nativamente, unificando diverse strutture font di OS X in un'unica interfaccia di programmazione completa.

5. **Image I/O Framework**
   Fornisce interfacce per la lettura e la scrittura della maggior parte dei formati di immagini. Se inizialmente Image I/O faceva parte del framework Core Graphics, è stato successivamente reso un framework singoo al fine di consentire agli sviluppatori di utilizzarlo in modo indipendente.

6. **Assets Library Framework**
   Tale framework consente di accedere alle foto e ai video presenti nella libreria fotografica del dispositivo. L'accesso viene gestito da un sistema basato su query per il recupero di foto e video all'interno del file system del dispositivo. E' inoltre possibile salvare nuove foto e video all'interno della libreria fotografica.

Core Services Layer

Tale livello contiene molti dei servizi fondamentali per la realizzazione di applicazioni, anche se alcuni di essi non vengono mai utilizzati direttamente, tuttavia molte parti del sistema sono costruite su tali servizi. E' importante sottolineare che tale livello mette a disposizione svariati frameworks per lo sviluppo, tra questi verranno descritti in maggior dettaglio quelli che sono stati utilizzati in fase di progettazione e sviluppo:

1. **Core Data Framework**
   Introdotto a partire da iOS 3.0, il framework CoreData è una tecnologia pensata per la gestione del modello di dati di una applicazione Model-View-Controller. Se l'applicazione possiede una piccola struttura non è necessario utilizzare CoreData, il quale è stato realizzato per gestire modelli di dati altamente strutturati. Infatti è possibile definire le strutture dati non solo via codice ma anche utilizzando un tool grafico integrato in Xcode, riducendo così la quantità di codice che il programmatore dovrebbe altrimenti scrivere. Tramite tale tool è possibile creare uno schema del modello dati dell'applicazione. Inoltre CoreData mette a disposizione anche le seguenti funzionalità:

   - Conservazione dei dati in un database SQLite per migliorare le performance
   - Un NSFetchedResultsController, una classe che si occupa di gestire i risultati delle query sulle entity del modello
   - Supporto per la propagazione di cambiamenti e per assicurare che le relazioni tra gli oggetti siano consistenti
   - Supporto per il filtraggio, l'organizzazione dei dati in memoria

3.2 OS X

Si tratta di un sistema operativo sviluppato da Apple ed attualmente disponibile su tutti i computer dell'azienda di Cupertino. Dal punto di vista dello sviluppatore di applicazioni, Apple mette a disposizione l' OS X SDK e l'IDE Xcode. Utilizzando l'ambiente di sviluppo Xcode e i framework di sistema, si possono sviluppare un'ampia varietà di applicazioni software per Mac, incluse le seguenti:

- **Apps** Le applicazioni aiutano a realizzare funzioni che vanno dalla creazione di contenuti e l'organizzazione di dati alla connessione con altre persone e al divertimento. OS X fornisce inoltre un vario sistema di tecnologie che possono essere utilizzate per estendere le funzionalità della propria applicazione e aumentare la qualità dell'esperienza dell'utente nell'utilizzo dell'applicazione.
- **Frameworks e librerie** che permettono la condivisione di codice tra diverse applicazioni.
- **Strumenti da riga di comando e Demoni**. I primi permettono agli utenti più esperti di manipolare i dati da riga di comando del terminale dell'applicazione. I demoni (*daemon* in inglese) vengono tipicamente eseguiti in background e agiscono come server per processare le richieste provenienti dai client.
- **Plug-ins delle applicazioni e i pacchetti**. I primi che estendono le funzionalità di altre applicazioni e i secondi che contengono il codice e le risorse che l'applicazione può caricare in modo dinamico quando viene eseguita.
- **Plug-ins di sistema**, quali ad esempio le unità audio, le estensioni del kernel, il kit I/O dei drivers del dispositivo, i pannelli delle preferenze, screen saver; estendono le funzionalità di sistema.

3.2.1 Architettura di OS X

Come iOS, anche l'architettura de sistema operativo OS X è composta da diversi livelli, come si può vedere in figura 3.7. I livelli più bassi forniscono i servizi fondamentali ai quali si affidano tutti i software su OS X. I layer sovrastanti, esattamente come per iOS, forniscono invece servizi più sofisticati e tecnologie che sono costruite sui servizi sottostanti oppure li complementano.

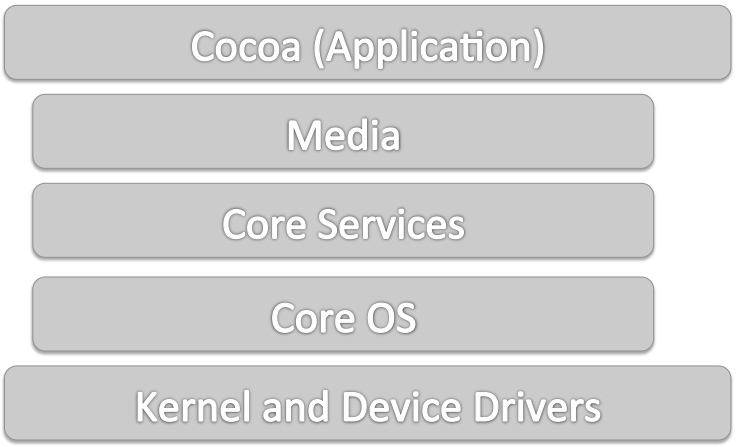

Figure 3.7: Architettura a livelli del sistema operativo iOS

Di seguito una breve descrizione dei vari livelli:

- Il livello **Cocoa Application** include tutte le tecnologie per realizzare l'interfaccia utente dell'applicazione, rispondere ad eventi dell'utente come il click all'interno di una finestra, e per gestire il comportamento dell'applicazione in generale.

- Il livello *Media Layer* comprende le tecnologie necessarie per riprodurre, registrare e la modifica di contenuto multimediale sia video che audio, e per realizzare animazioni grafiche 2D e 3D
- Il livello *Core Services* contiene molti dei servizi fondamentali e delle tecnologie che vanno dalla comunicazione di rete a basso livello e dall'ARC (Automatic Reference Counting) alla manipolazione di stringhe e alla formattazione dei dati.
- Il livello Core OS definisce le interfacce di programmazione che sono collegate all'hardware e quelle di rete, incluse le interfacce per eseguire funzioni ad elevate prestazioni di calcolo sulla CPU o sulla GPU del computer.
- Il livello Kernel and Device Drivers consiste dell'ambiente kernel Mach, i drivers del dispositivo, librerie di sistema e altre componenti di basso livello. Inoltre il livello include i supporti per i file di sistema, rete, sicurezza, comunicazione tra processi, linguaggi di programmazione, drivers del dispositivo, ed estensioni per il kernel.

3.2.2 Interfacce di OS X

La maggior parte dei frameworks di iOS sono presenti anche in OS X, anche se tuttavia esistono differenze tra il framework di iOS e il suo corrispetivo di OS X nel come tali framework siano implementati ed utilizzati. Senza fornire una panoramica dei framework di OS X, che molti aspetti sono simili a quelli di iOS si è deciso di descrivere le basilari differenze nell'utilizzo e nell'implementazione, in modo da non risultare ridondanti nella descrizione delle tecnologie utilizzate. Bisogna precisare che alcuni framework presenti in iOS, per quanto iOS sia nato più tardi rispetto ad OS X, non sono ancora presenti in OS X. Ad esempio il framework per la gestione delle mappe, MapKit, in OS X non è ancora presente anche se è stato annunciato al WWDC 2013 che tale framework sarà presente nel nuovo sistema operativo Mavericks OS X. L'espansione dei framework di iOS è maggiore rispetto a quella di OS X in quanto gli sviluppatori prediligono il primo per la portata del mercato di applicazioni.

- **UIKit vs AppKIt**
  In iOS il framework UIKit fornisce l'infrastruttura necessaria per la realizzazione di applicazioni grafiche, gestire il ciclo di eventi, ed eseguire altre operazioni relazionate all'interfaccia utente. Tale framework è completamente distinto dal corrispettivo di OS X: il framework AppKit. Una delle principali differenze risiede ad esempio nel sistema di coordinate, se infatti in iOS l'origine delle coordinate si trova sull'angolo sinistro in alto dello schermo, in OS X l'origine è posta invece nell'angolo in basso a sinistra. L'ascissa resta quindi invariata, ma l'ordinata è opposta. Le altre differenze risiedono soprattutto nella differenza di utilizzo delle applicazioni per iOS e OS X, nelle prime l'utente può interagire con l'applicazione toccando lo schermo con le dita, mentre nella seconda tipologia di applicazioni l'interazione è basata sul click del mouse oppure sull'input da tastiera. Questo comporta ad esempio l'esistenza di diverse funzioni per l'individuazione degli eventi (click del mouse, tocco, swipe, ecc.). Inoltre esistono classi di view specifiche per uno solo dei due framework, che non trovano il corrispettivo.
- **Foundation Framework**
  Una versione del framework Foundation è disponibile sia in OS X che in iOS, entrambi forniscono il supporto per gestire valori, stringhe, collezioni, threads, e molti tipi di dato tra i più comuni. Esistono tuttavia alcune tecnologie che non sono incluse in iOS. Una delle principali tecnologie che ad esempio non è ancora presente in iOS è il *bindings*, una collezione di tecnologie per implementare al meglio il paradigma *model-view-controller*: nel quale il modello incapsula i dati dell'applicazione, la view si occupa di visualizzare e modificare tali dati, e il controller esegue una funzione di mediatore tra model e view.

Part I

Realizzazione Mappa per Pulire 2.0

Chapter 4

Progettazione e sviluppo

Pulire 2.0, la più grande fiera in Italia della pulizia professionale, diventa per la prima volta una fiera tecnologica grazie all'utilizzo di due applicazioni, una per gli espositori e l'altra per i visitatori. Tramite le nuove tecnologie si pone l'obiettivo di far sì che la fiera non si esaurisca nei 3 giorni di manifestazione, ma possa essere preparata molto prima e continuare molto dopo. Tra le varie funzionalità offerte, rientra anche la visualizzazione della mappa dello spazio espositivo, che deve permettere all'utente di interagire con essa per potersi orientare all'interno della fiera. Le principali caratteristiche che la funzionalità di mappa deve possedere sono:

- Visualizzazione dell'intera mappa della fiera (16.000 mq di spazio espositivo) in un'unica schermata e la possibilità di effettuare zoom in/zoom out senza perdere qualità immagine
- Visualizzazione del pin quando l'utente clicca su uno stand e del rispettivo callout e implementazione delle funzionalità relative a pin e callout come il framework Mapkit di iOS indica
- Il click sulla lista degli espositori deve fornire la possibilità di visualizzare con un'animazione un pin sullo stand in cui si trova quell'espositore, con il rispettivo callout

Per ognuna delle caratteristiche sono state visionate e studiate molte strade prima di raggiungere quella definitiva e di ottenere il prodotto poi utilizzato durante la fiera. Di seguito verranno descritte le funzionalità descritte sopra e le soluzioni individuate per realizzarle al meglio.

4.1 Visualizzazione dell'intera mappa in un'unica schermata

Il primo problema da fronteggiare è stato comprendere come visualizzare una mappa di 16.000 mq all'interno di uno schermo di pochi centimetri.

4.1.1 Panoramica dei framework e dei tool esistenti

Essendo una mappa, si è subito pensato di utilizzare il framework Mapkit di iOS per poter visualizzare ed interagire con la mappa della fiera proprio come nell'applicazione nativa Mappe di un qualunque dispositivo dell'azienda di Cupertino. Tuttavia questa strada non è risultata percorribile in quanto le mappe utilizzabili in tale framework possono solo essere Mappe di Mapkit. A questo punto non potendo utilizzare le mappe di MapKit si è pensato di ricorrere all'utilizzo delle mappe di Google, considerate a partire da iOS 6 le principali rivali delle mappe di Apple. Google mette a disposizione un'interessante applicazione, Google Maps Floor Plans, che permette di effettuare l'upload dell'immagine relativa alla mappa interna della fiera (appunto floor plan) e di inserirla come un nuovo livello identificando sulla cartina l'edificio della fiera e ponendovi sopra l'immagine. Tuttavia si è dovuta abbandonare questa strada in quanto tale servizio di Google risulta al momento disponibile solo in alcuni paesi, tra i quali non è ancora presente l'Italia. I paesi in cui il servizio di Google è invece disponibili sono al momento:

- Belgio

- Canada
- Danimarca
- Francia
- Germania
- Giappone
- Inghilterra
- America
- Svezia

Tuttavia oltre a MapKit esistono una serie di framework non nativi per iOS ed open source che utilizzano mappe personalizzate e non solo quelle di MapKit, quali ad esempio:

- *Route-Me*
  Route-Me è una libreria open source che può essere utilizzata in modo nativo in una qualunque applicazione iOS ed è stata progettata per rendere le stesse funzionalità del framework nativo MapKit di iOS, aggiungendo la possibilità di utilizzare una qualunque mappa come source. La mappa che si vuole visualizzare deve provenire da una delle seguenti sorgenti:
  - Server del progetto Open Street Map, si tratta di file in formato .osm che ricoprono la regione d'interesse
  - CloudMade, che fornisce servers per ospitare dati di tipo Open Street Map
  - Microsoft Virtual Earth
  - Open Aerial Map
  - Mappe di Yahoo!

  Ognuna di queste sorgenti di dati possiede differenti restrizioni e regole per ottenere la licenza di utilizzo delle proprie mappe. In particolare, le mappe di Yahoo! risultano inutilizzabili all'interno di Route-Me a causa dei termini di licenza in quanto il codice d'accesso alle mappe viene fornito solo per dimostrazioni. In tutti questi casi si tratta quindi di utilizzare mappe già realizzate e non di visualizzare una mappa custom. Tale framework mette inoltre a disposizione il servizio Mapnik, uno strumento per il rendering di mappe dalla qualità grafica elevata grazie all'anti-aliasing, posizionamento delle etichette intelligenti e simboli SVG scalabili. Il più famoso utilizzo di Mapnik è quello di renderizzare lo stile del layer principale all'interno delle mappe di OpenStreetMap, solitamente viene utilizzato in applicazioni python per distribuire le proprie mappe attraverso internet. E' destinato a essere eseguito in un ambiente multithread e mira principalmente, ma non esclusivamente, allo sviluppo basato sul web.

- *MapBox*
  MapBox è una variante di Route-Me che possiede molte funzionalità e la documentazione a proposito di tale framework è molto vasta. Tuttavia Route-Me utilizza Quartz, la tecnologia dietro UIKit, al contrario di Mapkit che utilizza OpenGL ES, e questo causa una riduzione di velocità nel caricamento della mappa e durante la fase di zoom in e zoom out. Anche MapBox utilizza come sorgente mappe di tipo Open Street Map oppure permette di creare mappe custom utilizzando il tool TileMill. Le mappe web generate da TileMill possono usare tooltips sovrapposti, popup cliccabili, grafici e immagini interattive, marker SVG, tessiture complesse e layer multipli. Una volta realizzate possono poi essere esportate in differenti formati come: .png, .pdf, .svg e MBTiles. TileMill supporta sia di dati vettoriali (CSV, shapefile, KML, GeoJSON) o raster (GeoTiff), e può anche collegarsi a grandi sorgenti di dati come OpenStreetMap, PostgreSQL e SQLite. Tilemill fa uso della libreria di restituzione di mappe Mapnik (usato da OpenStreetMap) e usa CartoCSS come linguaggio di stile.

- *CloudMade API*
  Il principale vantaggio di Cloude Made rispetto a Google Maps, Yahoo Maps e altri competitors su larga scala, è quello di permette ai propri utenti di gestire lo schema dei colori e i livelli visibili all'interno di mappe che possono essere personalizzabili. Allo stesso modo di OpenStreetMap, CloudMade utilizza Mapnik per il rendering delle proprie mappe, mentre il resto del software si allontana dal sistema OSM. Tuttavia CloudeMade non è open source come MapBox o Route-Me e il costo al momento è di un centinaio di dollari all'anno.

Riassumendo: MapKit, il framework nativo di iOS per la gestione delle mappe, non può essere utilizzato con mappe non appartenenti al pacchetto fornito da MapKit, mentre per quanto riguarda gli altri framework, eliminando quelli non open source, risultano poco personalizzabili e i tool per realizzare la mappa non sono pensati per mappe di interni. Inoltre in tutte le soluzioni bisogna affidarsi ad un web server esterno sul quale viene caricata la mappa. Sebbene in questo modo l'applicazione risulti più leggera, lo svantaggio è che deve essere presente una connessione ad internet per scaricare la mappa dal server esterno. Si è quindi cercata una soluzione personalizzabile e facilmente gestibile, senza dover ricorrere a servizi esterni. Si è pensato quindi di realizzare un proprio framework basato sulle principali funzionalità che i framework sopra descritti offrono.

4.1.2 Tiles e Zoom Level

La visualizzazione di una mappa di grandi dimensioni all'interno di uno schermo notevolmente più piccolo viene realizzata da tali framework suddividendo la mappa in piccole immagini tipicamente di 256x256 pixel dette tiles. Ogni tile è quindi una piccola immagine che contiene informazioni topografiche di un'area rettangolare della mappa, affiancando le tiles si ottiene l'illusione di visualizzare l'intera immagine senza tagli. Per visualizzare più dettagli all'interno della mappa si utilizza generalmente il livello di zoom: alti livelli di zoom aumentano la dimensione fisica della mappa visualizzata sullo schermo e ovviamente la quantità di dettagli visibili, come si può vedere in fig. 4.7. Organizzare tutti questi piccoli tasselli affinché il mosaico della mappa risulti ordinato è necessario utilizzare un sistema d coordinate. Ogni tile possiede una coordinata z che descrive il livello di zoom e le coordinate x ed y che descrivono invece la posizione della tile all'interno della griglia di tasselli per quel zoom level. Ad esempio la prima tile nel sistema di coordinate appena definito, si trova allo 0/0/0. Il livello di zoom 0 permette di visualizzare l'intera mappa, al livello successivo z0 viene suddivisa in quattro quadrati equivalenti in modo tale che 1/0/0 e 1/1/0 coprano la parte superiore mentre 1/0/1 e 1/1/1 coprano la parte inferiore della mappa. I livelli di zoom sono correlati tra loro da potenze di 4, ad esmpio z0 contiente 1 sola tile, z1 ne contiene 4, z2 ne contiene 16 e così via. Questa relazione esponenziale aumenta la quantità di dettagli visibili ad ogni livello di zoom ed aumenta anche la quantità di memoria richiesta per il salvataggio delle tiles in memoria. Per comprendere quanto sia importante il consumo di memoria dovuto a tale metodologia si pensi ad esempio ad un livello di zoom z15, esso richiederebbe 1.1 bilioni di tiles per coprire l'intera mappa, mentre ad un livello z17, solo due livelli di zoom di più, sarebbero necessarie 17 bilioni di tiles.

4.1.3 Sorgente della mappa

Nel caso dei framework sopra presentati, la sorgente delle tiles è una mappa di Google Maps o servizi alternativi come OpenStreetMaps. Come utilizzare quindi la suddivisione in tiles senza poter utilizzare le sorgenti standard e i framework adibiti al supporto delle funzionalità di mappa? L'iOS SDK mette a disposizione tra le svariate tecnologie, un framework che permette di suddividere immagini di grandi dimensioni in tiles e fornisce allo sviluppatore tutte le funzioni per la gestione e il rendering di tali tiles. Disponendo di un'immagine prototipo della mappa in formato .png si è deciso di utilizzare tale strada per la visualizzazione della mappa con un'elevata qualità e di realizzare poi un framework strutturato come MapKit che consideri come sorgente non più una mappa di Google Maps o altre fonti ma una semplice immagine png suddivisa in tiles.

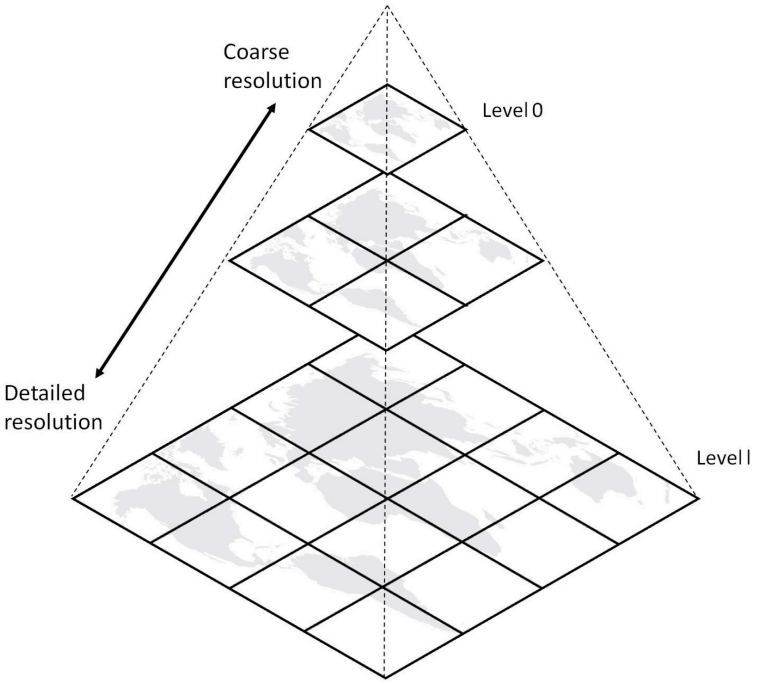

Figure 4.1: Suddivisione di una mappa in tiles a seconda del livello di zoom

4.1.4 Suddivisione della mappa in Tiles con ImageMagick

Per la suddivisione in tiles si è deciso di utilizzare ImageMagick, una suite di programmi open source che permette di creare, gestire, comporre e convertire immagini bitmap. Può leggere e scrivere immagini in una varietà di formati (circa 100) inclusi DPX, EXR, GIF, JPEG, JPEG 2000, PDF, PhotoCD, PNG, Postscript, SVG e TIFF. ImageMagick permette di agire sulle immagini solamente da riga di comando, non esiste infatti alcun tool grafico come in Photoshop o in Gimp. Può sembrare più scomodo ma in realtà è una opzione molto potente per elaborare molte immagini in poco tempo o includere l'elaborazione delle immagini in programmi web o script bash. Lo script bash utilizzato per ottenere la suddivisione in tiles è riportato in fig. 4.1. E' stata definita una funzione tile che imposta il livello zoom dell'immagine e la ritaglia in quadratini di 256x256 pixel (le tiles), salvandole in un nuovo file. Tale file viene nominato utilizzando le coordinate x ed y della tile all'interno della griglia e il livello di zoom in modo tale che sia più facile riconoscere le varie tiles al momento di doverle caricare da codice.

Listing 4.1: Script Bash per la suddivisione in tiles

```
#!/bin/bash
file=$1
function tile() {
convert $file -scale ${s}%x -crop 256x256 \
-set filename:tile "%[fx:page.x/256]_%[fx:page.y/256]" \
+repage +adjoin "${file%.*}_${n}_%[filename:tile].png"}
s=12.5 n=125
tile
s=25 n=250
tile
s=50 n=500
tile
s=100 n=1000
tile
```

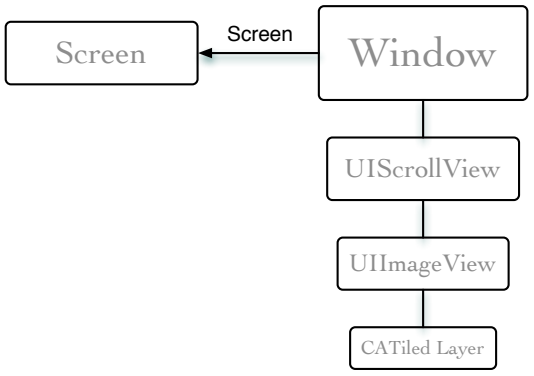

Figure 4.2: Gerarchia delle view

Figure 4.3: Screenshot relativo alla visualizzazione della mappa a livello di zoom 0

4.1.5 Il framework CATiledLayer e la gestione delle tiles

CATiledLayer è il framework che viene utilizzato per la visualizzazione e la gestione delle tiles. Il funzionamento di tale tecnologia può sembrare qualcosa di magico in quanto la maggior parte della sua implementazione è racchiusa in una sorta di scatola nera nascosta al programmatore. Per quanto riguarda l'utilizzo di tale framework gli sviluppatori hanno pareri contrastanti e probabilmente tale confusione è dovuta principalmente al fatto che fino ad iOS 4 CATiledLayer causasse continui crash dell'applicazione sulla quale veniva eseguito senza apparenti spiegazioni. Tuttavia dopo iOS 4, Apple si è premunita di sistemare i bug di tale framework e renderlo così più efficiente, ma ormai gli sviluppatori frustati dai continui crash l'avevano già abbandonato. Solitamente viene utilizzato dagli sviluppatori per:

- Disegnare immagini troppo grandi per essere inserite in un'unica UIImageView (una qualunque immagine di dimensioni maggiori a 2096x2096)
- Ridurre l'occupazione di memoria causato da immagini di grandi dimensioni
- Aggiungere la possibilità di realizzare uno zoom infinito come in Google Maps preservando la qualità dell'immagine ad ogni livello di zoom
- Implementare correttamente lo zoom su oggetti di tipo CGPath se essi si trovano all'interno di una UIScrollView (senza CATiledLayer essi risultano sfuocati quando viene effettuato lo zoom)

Utilizzare CATiledLayer per visualizzare l'intera mappa richiede che l'immagine in formato .png dell'intera mappa sia collocata su di una UIImageView in quanto essa è stata progettata appositamente per la gestione del rendering di immagini. Inoltre, al fine di poter implementare le funzioni di zooming necessarie per la visualizzazione di maggiori dettagli, l'UIImageView dovrà essere collocata all'interno di una UIScrollView. La struttura delle view risulta essere quindi come in fig. 4.2: all'interno della window è stata inserita la UIScrollView, la quale avrà come subview la

UIImageView, la quale a sua volta avrà come subview il CATiledLayer che si occuperà di caricare la giusta tiles a seconda delle coordinate e del livello di zoom. In fig. 4.3 è possibile vedere il risultato di tale struttura: quando il livello di zoom è a livello 0 l'immagine viene interamente visualizzata sullo schermo, la UIScrollView non mostra le scrollbars e il CATiledLayer carica l'intera immagine. E' necessario sottolineare che in realtà non viene mai utilizzata l'intera immagine, nemmeno a livello di zoom 0, in quanto altrimenti essendo un file in formato .png la qualità della mappa ne risentirebbe. Come si ottiene allora l'immagine in fig. 4.3 se non si dispone dell'intera immagine? Per rispondere a tale domanda è necessario comprendere come venga implementato tale livello e come esso gestisca il caricamento delle tiles.

CATiledLayer è in realtà una sottoclasse di CALayer, CALayers sono semplicemente classi che rappresentano un rettangolo sullo schermo e ne visualizzano il contenuto. Questa è anche la definizione di UIView, perciò cosa differenza una UIView da un layer? Ogni UIView contiene di default un layer radice sul quale disegnare. E' possibile accedere a tale layer radice utilizzando la seguente riga di codice:

```
CALayer *myLayer = myView.layer;
```

Esistono diversi tipi di oggetti CALayer che permettono di implementare specifiche funzionalità oltre a quelle definite nella classe CALayer. E' quindi possibile cambiare il tipo di layer utilizzato da una iOS view semplicemente sovrascrivendo il metodo layerClass della view e ritornando la classe oggetto che si vuole utilizzare nel seguente modo:

```
+ (Class) layerClass
    {
         return [CATiledLayer class];
    }
```

Per poter poi disegnare utilizzando il layer creato bisogna implementare il codice relativo al disegno all'interno della funzione **drawRect**, una funzione propria della classe UIView. Tale funzione viene chiamata quando la view appare sullo schermo e i suoi contenuti vengono aggiornati. Prima di chiamare l'implementazione del metodo drawRect scritta dal programmatore l'oggetto view configura automaticamente l'ambiente di disegno in modo tale che la funzione drawRect possa iniziare a disegnare immediatamente. La parte di configurazione effettuata dalla UIView comprende la creazione di un contesto grafico per l'ambiente di disegno corrente. Il contesto grafico fornisce l'area sulla quale è possibile disegnare e ne gestisce le principali funzionalità: colore, larghezza della linea, trasformazioni, ecc. Il contesto può essere ad esempio un'immagine Bitmap, il layer di una UIView, un file PDF. E' poi possibile accedere a tale contesto grafico dal metodo drawRect chiamando la funzione definita nell'UIKit framework: UIGraphicsGetCurrentContext. Il caricamento delle tiles viene quindi completamente gestito all'interno della funzione drawRect in quanto non appena viene individuata la tile da utilizzare, in base alle coordinate x, y ed al livello di zoom (scale), essa deve essere disegnata all'interno del contesto grafico della UIView (nel nostro caso tale contesto è il CATiledLayer) al quale si può accedere soltanto all'interno del metodo drawRect tramite la funzione UIGraphicsGetCurrentContext.

```
- (void)drawRect:(CGRect)rect
{
    CGContextRef context = UIGraphicsGetCurrentContext();

    // get the scale from the context by getting the current transform
        matrix, then asking
    // for its "a" component, which is one of the two scale components. We
        could also ask
    // for "d". This assumes (safely) that the view is being scaled equally
         in both dimensions.
    CGFloat scale = CGContextGetCTM(context).a;

    CATiledLayer *tiledLayer = (CATiledLayer *)[self layer];
    CGSize tileSize = tiledLayer.tileSize;
```

```
    // Even at scales lower than 100%, we are drawing into a rect in the
         coordinate system
    // of the full image. One tile at 50% covers the width (in original
         image coordinates)
    // of two tiles at 100%. So at 50% we need to stretch our tiles to
         double the width
    // and height; at 25% we need to stretch them to quadruple the width
         and height; and so on.
    // (Note that this means that we are drawing very blurry images as the
         scale gets low.
    // At 12.5%, our lowest scale, we are stretching about 6 small tiles to
          fill the entire
    // original image area. But this is okay, because the big blurry image
         we're drawing
    // here will be scaled way down before it is displayed.)
    tileSize.width /= scale;
    tileSize.height /= scale;

    // calculate the rows and columns of tiles that intersect the rect we
         have been asked to draw
    int firstCol = floorf(CGRectGetMinX(rect) / tileSize.width);
    int lastCol = floorf((CGRectGetMaxX(rect)-1) / tileSize.width);
    int firstRow = floorf(CGRectGetMinY(rect) / tileSize.height);
    int lastRow = floorf((CGRectGetMaxY(rect)-1) / tileSize.height);

    for (int row = firstRow; row <= lastRow; row++) {
        for (int col = firstCol; col <= lastCol; col++) {
            UIImage *tile = [self tileForScale:scale row:row col:col];
            CGRect tileRect = CGRectMake(tileSize.width * col, tileSize.
                 height * row,
                                         tileSize.width, tileSize.height);

            // if the tile would stick outside of our bounds, we need to
                 truncate it so as
            // to avoid stretching out the partial tiles at the right and
                 bottom edges
            tileRect = CGRectIntersection(self.bounds, tileRect);

            [tile drawInRect:tileRect];
        }
    }
}
```

La funzione tileForScale si occupa invece di recuperare all'interno del progetto il file relativo alla tile che deve essere disegnata. L'aver nominato il file di ogni tile specificando il livello di zoom e le coordinate x ed y della tile all'interno della griglia di suddivisione in tiles permette di riconoscere velocemente l'immagine tra tutte le tiles presenti.

```
- (UIImage *)tileForScale:(CGFloat)scale row:(int)row col:(int)col
{
    // we use "imageWithContentsOfFile:" instead of "imageNamed:" here
         because we don't
    // want UIImage to cache our tiles
    //
    NSString *tileName = [NSString stringWithFormat:@"%@_%d_%d_%d",
         _imageName, (int)(scale * 1000), col, row];
    NSString *path = [[NSBundle mainBundle] pathForResource:tileName ofType
         :@"png"];
    UIImage *image = [UIImage imageWithContentsOfFile:path];
    return image;
}
```

4.1.6 UIScrollView e gestione dello zoom

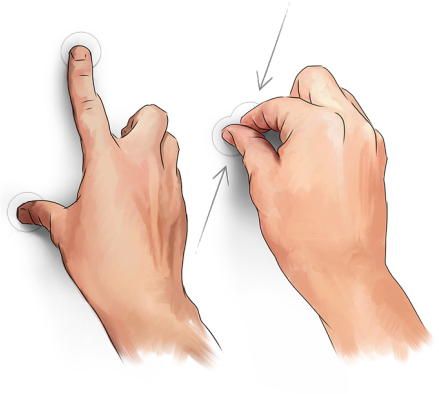

Figure 4.4: Pinch Gesture

Per quanto riguarda invece la gestione dello zoom la classe UIScrollView permette di implementare velocemente e facilmente lo zoom azionato dal pinch dell'utente sul display, come mostrato in fig. 4.4. E' sufficiente infatti impostare un delegato per la UIScrollView che sia conforme al protocollo UIScrollViewDelegate. Tale oggetto delegato dovrà poi semplicemente implementare la funzione viewForZoomingInScrollView: ritornando la view sulla quale deve essere effettuato lo zoom.

```
- (UIView *)viewForZoomingInScrollView:(UIScrollView *)scrollView
{
    return imageView;
}
```

Per avvicinarsi sempre di più alle funzionalità offerte dal framework Mapkit, si è deciso inoltre di inserire lo zoom in azionato da due tocchi consecutivi con un solo dito e lo zoom out azionato dal tocco singolo con due dita. Per inserire tali funzionalità è necessario:

- Creare un oggetto di tipo UIGestureRecognizer per ognuna delle gesture che si vuole implementare. Tale oggetto si occuperà di chiamare una specifica funzione dichiarata al momento della sua creazione, quando verrà rilevato il tocco da parte dell'utente.
- Prima di aggiungere tale oggetto alla view sulla quale si vuole rilevare il tocco, bisogna impostare i numeri di tocchi, con quante dita viene azionato il tocco e infine la priorità. Supponiamo infatti di aver implementato sia il tocco singolo che i due tocchi consecutivi, allora l'applicazione rileverà sempre prima il singolo tocco non chiamando mai la funzione relativa al doppio tap.
- Le funzioni chiamate dalla gesture corrispondente, se devono implementare lo zoom in o lo zoom out devono conoscere il punto cliccato ed efettuare lo zoom solo in quella zona della mappa. A tale scopo è stata creata una apposita funzione che si occupa di individuare l'area della view sulla quale effettuare lo zoom. Tale funzione richiede come parametri il punto cliccato, che è possibile ricavare dalla funzione associata alla gesture e il nuovo livello di zoom calcolato a partire dal livello corrente moltiplicato per una costante. Più la costante aumenta e più lo zoom in sarà evidente. Si è verificato che un valore di 1.5 moltiplicato per il livello di zoom della scrollview al momento del tap risulta il valore corretto. Individuata l'area grazie alla funzione descritta, è poi possibile utilizzare la funzione zoomToRect:(CGRect)rect animated:(BOOL)animated che effettua lo zoom in modo tale che il contenuto della view diventi l'area definita dal rect, aggiustando il livello di zoom se necessario.

```
- (CGRect)zoomRectForScale:(float)scale withCenter:(CGPoint)center
{
    CGRect zoomRect;

    // the zoom rect is in the content view's coordinates.
    //    At a zoom scale of 1.0, it would be the size of the
          imageScrollView's bounds.
    //    As the zoom scale decreases, so more content is visible, the
          size of the rect grows.
    zoomRect.size.height = [scrollView frame].size.height / scale;
    zoomRect.size.width  = [scrollView frame].size.width  / scale;

    // choose an origin so as to get the right center.
    zoomRect.origin.x    = center.x - (zoomRect.size.width  / 2.0);
    zoomRect.origin.y    = center.y - (zoomRect.size.height / 2.0);


    return zoomRect;
}
```

4.2 Gestione di Pin e Callout

Conclusa quindi la visualizzazione della mappa e la gestione dello zoom bisogna fare in modo che la mappa supporti le stesse funzionalità che l'utente si aspetterebbe di trovare aprendo una qualunque applicazione che si basi sul framework MapKit. Mapkit utilizza degli oggetti chiamati annotations per evidenziare specifiche coordinate sulla mappa e poter fornire all'utente informazioni aggiuntive su di esse. Tipicamente per identificare un annotation viene inserita una immagine nel punto che si vuole evidenziare fornendo la possibilità di visualizzare una nuvoletta di testo con ulteriori informazioni (callout). E' possibile inserire immagini personalizzate oppure utilizzare i pin, una sottoclasse degli annotation che utilizza come immagine quella di una puntina dalla testa rossa. L'utilizzo dei pins risulta importante nella realizzazione della mappa per Pulire 2.0 in quanto si vuole permettere all'utente di selezionare uno stand e scoprire maggiori informazioni sull'espositore. La checklist da seguire per aggiungere un annotation alla propria mappa è la seguente:

1. Definire l'oggetto di tipo annotation appropriato utilizzando una delle seguenti opzioni:
   - Se si vuole ottenere un annotation standard la classe MKPointAnnotation permette di implementare un semplice annotation e utilizzandone le proprietà è possibile specificare il titolo e il sottotitolo che si vuole far apparire all'interno del callout
   - Se invece si vuole ottenere un annotation personalizzato bisogna creare un oggetto che sia conforme al protocollo MKAnnotation.
2. Definire l' annotation view che si occupi di visualizzare l'oggetto annotation creato sullo screen. Se l'annotation è identificato da un'immagine statica allora è sufficiente creare un'istanza della classe MKAnnotationView ed assegnare l'immagine alla proprietà immagine. Se al contratrio si vuole utilizzare un pin annotation standard bisogna creare un'istanza della classe MKPinAnnotationView
3. Implementare la funzione mapView:viewForAnnotation: nel delegato della mapView
4. Infine aggiungere l'oggetto annotation alla mapView utilizzando la funzione addAnnotation: oppure addAnnotations:.

Annotations e callouts sono oggetti definiti per essere utilizzati in combinazione con oggetti di tipo MKMapView, un oggetto che definisce una view, la quale permette di visualizzare la mappa fornita da MapKit e non permette invece di utilizare sorgenti di mappe che siano immagini. La soluzione individuata è stata quella di creare un framework che permettesse di aggiungere

e gestire gli annotations e i callouts nello stesso modo in cui lavora MapKit ma basandosi su un'immagine invece che su un oggetto di tipo MKMapView. Anziché interagire direttamente con il CATiledLayer si è pensato di realizzare una classe MapsView, sottoclasse di una semplice UIView, che si occupasse di gestire tutte le funzioni di aggiunta, rimozione e gestione dei pins e dei callouts senza interferire con la visualizzazione della mappa. Aggiungendo poi tale classe alla classe CATiledLayer come sottoclasse, è possibile riuscire a visualizzare pin e callout sulla mappa. Ma come visualizzare gli oggetti di tipo annotation su una imageView e nn su una mappa? Poiché la visualizzazione degli annotation richiede che essi siano contenuti all'interno di una annotationView, la quale è indipendente dall'oggetto di tipo MKMapView e può quindi essere utilizzata anche se non si dispone di tale oggetto. Si è pensato quindi di utilizzare la stessa checkList fornita da MapKit cambiandone solo l'ultimo punto ovvero implementando la funzione addAnnotation all'interno della classe mapsView. Tale funzione inizializza un oggetto di tipo annotationView, e la posiziona nel punto in cui si vuole compaia il pin, aggiungendola poi come subview della mapsview, in questo modo la gerarchia delle view diventa quella in fig. 4.5. Aggiunti i pin alla mappa, si è notato un problema quando si effettuava uno zoom sulla mappa, infatti il pin aumentava di dimensione assieme alla mappa diventando così enorme quando il livello di zoom era massimo. Per risolvere tale problema si è pensato di

```
  -(void)scrollViewDidZoom:(UIScrollView *)scrollview
{
    //scalo la/le pinview in modo che il/i pin non resti sempre uguale/i
    for (MAPAnnotationView *pinview in mapview.subviews)
    {
            pinview.transform = CGAffineTransformMakeScale(1.2/scrollView.
                 zoomScale, 1.2/ scrollView.zoomScale);
    }
}
```

Per quanto riguarda i callouts è stato invece utilizzato un framework di terze parti SMCalloutView il quale è stato realizzato con il proposito di poter utilizzare i callout su sorgenti di qualsiasi tipo e non solo MKMapView. Tale framework viene utilizzato anche all'interno di MapBox, uno dei framework precedentemente descritti come alternativa a MapKit, e offre quindi le stesse funzionalità di un qualunque callout di MapKit. Come si può vedere dalla fig. 4.6 infatti il callout appare esattamente come un qualunque callout di MapKit sia per quanto riguarda la grafica, sia per quanto rigurda le funzionalità. Cliccando sulla freccia è possibile accedere al dettaglio. Per utilizzare tale framework è sufficiente copiare i file SMCalloutView.h e SMCalloutView.m all'interno del proprio progetto. La funzione principale da chiamare è presentCalloutFromRect, la quale si occupa di aggiungere il callout alla view specificata (nel nostro caso l'annotationView).

A questo punto la mappa è completa, sono presenti tutte le componenti e sono state realizzate tutte le funzionalità richieste. Il problema più difficile da risolvere in fase di sviluppo e che merita un'analisi approfondita all'interno di questa sezione è stato gestire e riconoscere dove venisse compiuto il touch dell'utente per identificare quale elemento venisse toccato. Inoltre a seconda dell'elemento toccato era necessario lanciare una diversa azione a seconda dei casi:

- Se sulla mappa non è presente alcun elemento (non ci sono callout o pin), allora effettuando un solo tap, viene chiamata una funzione che invia una query al database, il quale contiene per ogni espositore le coordinate in pixel sulla mappa dello stand in cui si trova. Poiché con un dito non si riesce ad essere precisi nel click, si è pensato di calcolare un intorno del punto cliccato e richiedere al database di restituire tutti gli espositori le cui coordinate sono contenute in tale intorno. A questo punto si sceglie l'espositore le cui coordinate risultano avere distanza minore dal punto cliccato. Viene quindi inserito un pin non sul punto cliccato ma sulle coordinate restituite dal database, in questo modo il pin appare sempre nel centro dello stand cliccato e non in posizioni intermedie. E' necessario precisare che in realtà le coordinate non risiedono su un database esterno in quanto altrimenti ogni volta che l'utente effettua un tap sulla mappa dovrebbe essere presente la connessione internet altrimenti non vedrebbe nulla e resterebbe ad aspettare la comparsa di un pin invano. Proprio per questo motivo si è deciso di realizzare un modello di database interno all'applicazione utilizzando CoreData, tale modello è stato strutturato esattamente come il database esterno presente sul server. In questo modo

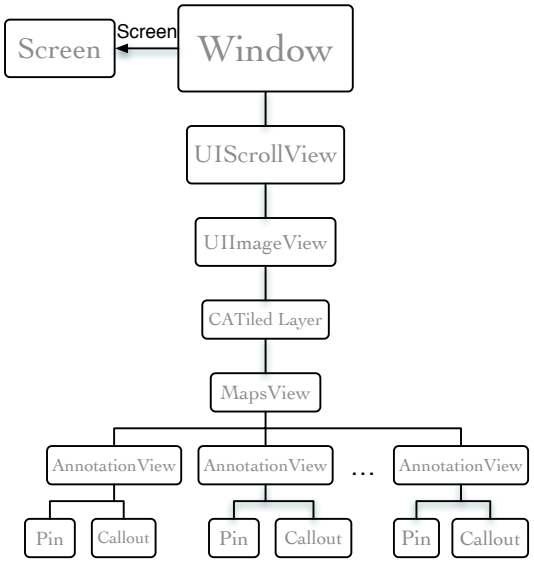

Figure 4.5: Gerarchia delle view

Figure 4.6: Screenshot relativo alla visualizzazione della mappa con pin e callout aperto

il database interno all'applicazione risulta essere una replica dello schema del database del server e viene popolato la prima volta che l'applicazione viene caricata scaricando i dati dal database esterno, poi il modello interno all'applicazione resta sempre uguale a meno che non avvenga qualche aggiornamento sul database esterno, a quel punto tale aggiornamento viene notificato all'applicazione che ripopolerà nuovamente il modello interno con i dati aggiornati.

- Se invece sulla mappa è già presente un pin, cliccando su un qualunque altro punto della mappa compare il pin relativo al nuovo punto cliccato e scompare il precedente, in modo da lasciare libera la mappa e non riempirla di pin.
- Se sulla mappa c'è un pin, cliccando su di esso compare il rispettivo callout e cliccando sulla freccina del callout è possibile accedere alla pagina di dettaglio dell'espositore. Per far scomparire il callout è possibile semplicemente cliccare nuovamente sul pin oppure cliccare su un qualunque punto della mappa.

Come riesce l'applicazione a rilevare su quale view avviene il click? Per risolvere tale problema bisogna implementare la funzione hitTest, la quale ritorna la subview che viene toccata. L'hit-test consiste nel verificare se il tocco è contenuto dentro il frame della view che si trova sopra tutte le altre sullo stack, se il punto è contenuto allora controlla ricorsivamente tutte le subviews di tale view. La view che si trova alla base dello stack diventa la hit-test. Individuata quindi la hit-test view iOS invia l'evento di touch a tale view per la sua gestione. Per comprenderne meglio il suo funzionamento, si supponga ad esempio di possedere la gerarchia di view in fig. **??** e che la view toccata sia la E, come si comporta la funzione di hitTest?

- Il tocco avviene all'interno della view A, quindi vengono testate le subviews B e C.
- Il tocco non è contenuto all'interno del frame della view B, ma è contenuto nel frame di C, quindi vengono testate D ed E.
- Il tocco non è contenuto nel frame di D, ma si trova in quello di E. A questo punto la view E, che è la view alla base dello stack, contiene il tocco e diventa quindi la hit-test view.

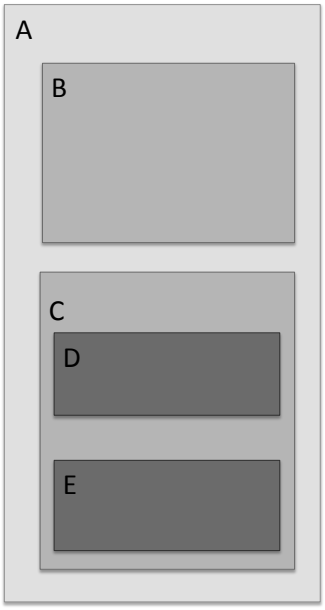

Figure 4.7: Esempio di gerarchia di view

```
- (UIView *) hitTest:(CGPoint)point withEvent:(UIEvent *)event
{
    if (_annotationViews.count > 0) //If there are pins on map
    {
        UIView *pinV = [_annotationViews[0] hitTest:[_annotationViews[0]
              convertPoint:point fromView:self] withEvent:event]; //Check if
               the tap is on the pin
        return pinV ?: [super hitTest:[super convertPoint:point fromView:
              self] withEvent:event];
    }
    else
        return [super hitTest:[super convertPoint:point fromView:self]
              withEvent:event];
}
```

4.3 Visualizzazione di un punto sulla mappa e animazione

Tale funzionalità deve permettere all'utente di visualizzare un punto specifico all'interno della mappa, in particolare deve mostrare lo stand nel quale si trova l'espositore selezionato. Le coordinate relative allo stand in cui si trova l'espositore vengono individuate inviando una semplice query al database. Una volta note le coordinate bisogna creare un oggetto di tipo annotation, aggiungerlo alla annotationView che verrà inserita come subview della MapsView. Per aiutare l'utente ad orientarsi all'interno della mappa si è deciso di aggiungere una animazione che effettuasse uno zoom in sul punto in cui inserire il pin. Nel caso in cui invece si voglia visualizzare un gruppo di espositori, è il caso in cui si effettua una ricerca per tipo, allora l'animazione non è più uno zoom in ma aggiunge ogni pin con un certo delay. Cliccando poi su un qualunque pin compare il callout che permette di accedere al dettaglio di quell'espositore.

Part II

Tool e Framework per la realizzazione di mappe georeferenziate

Chapter 5

Progettazione e sviluppo

Il principale obiettivo della realizzazione del tool in OS X è quello di permettere agli utenti di realizzare mappe vettoriali e georeferenziarle per poterle poi utilizzare all'interno del framework precedentemente realizzato. La georeferenziazione è la tecnica che permette di associare ad un dato, in formato digitale, una coppia di coordinate che ne fissino la posizione sulla superficie terrestre.

5.1 Realizzazione del tool per Mappe Vettoriali

Se per la mappa realizzata per Pulire 2.0 non risultava necessario includere una conversione di coordinate da geografiche a pixel per problemi di localizzazione dell'utente tuttavia generalmente tale funzionalità, implementata anche da MapKit, risulta molto utile nella realizzazione di mappe per parchi divertimento. In tale ottica si è quindi deciso di realizzare un proprio tool di disegno di mappe vettoriali.

5.1.1 Funzionalità necessarie

Descrizione delle funzionalità fondamentali che il Tool deve fornire per disegnare la mappa:

- disegnare elementi grafici
- aggiungere testo
- colorazione, riempimento
- aggiunta di Layer con la possibilità di rendere cliccabili alcune zone della mappa e di cambiarne colore cliccando sulla zona stessa.

Disegnare gli elementi grafici

Riuscire a realizzare elementi grafici di elevata qualità è importante al fine di ottenere un'applicazione di alto livello. Per poter disegnare tali elementi bisogna utilizzare la costruzione per cammini messa a disposizione dalla classe NSBezierPath. I cammini sono composti da primitive di comando e uno o più punti. Il comando comunica all'oggetto cammino come interpretare i punti associati. Una volta assemblati, l'insieme di punti crea una serie di linee che formano la figura desiderata. Tutte le figure hanno un cammino diverso ovviamente, ma alcune delle loro proprietà come colore di riempimento, colore della linea, bounds sono proprietà comuni a tutti gli elementi grafici. Si è quindi deciso di creare un generico oggetto che rappresentasse l'elemento grafico e poi per ogni diversa figura (rettangolo, cerchio, linea, testo) realizzare una classe, la quale eredita dalla classe generica le proprietà comuni ed implementa poi lo specifico cammino di ogni singola figura. Come sempre il codice relativo al disegno deve risiedere all'interno della funzione drawRect della view sulla quale si vogliono visualizzare gli elementi grafici. In base quindi a quale elemento grafico si vuole ottenere è necessario creare un'istanza di un oggetto della classe di quell'elemento e disegnarne il cammino. Per rendere più chiaro tale concetto si veda di seguito il codice relativo ad esempio al disegno di un rettangolo:

```
  @implementation MAPRect

- (NSBezierPath *)bezierPathForDrawing
{
    NSBezierPath *path = [NSBezierPath bezierPathWithRect:[self bounds]];
    [path setLineWidth:[self strokeWidth]];
    return path;

}

@end
```

```
   MAPRect * rect = (MAPRect *)element;
                NSBezierPath *rectPath = [rect bezierPathForDrawing];
                if (![element.strokeColor isEqualTo:[NSColor blackColor]])
                {
                    NSColor *color = element.strokeColor;
                    [rect setStrokeColor:color];
                    [[rect strokeColor] set];
                }
                else
                {
                    [[NSColor blackColor] set];
                }

                [rectPath stroke];
```

Spostare gli elementi grafici

Per conoscere quale elemento è stato selezionato dall'utente per essere spostato viene utilizzata la funzione mouseDown, la quale viene chiamata ogni qual volta l'utente preme il tasto sinistro del mouse all'interno della view che implementa tale metodo. Rilevato il punto cliccato, si verifica se uno degli elementi grafici presenti sulla view contenga o meno tale punto, in caso affermativo l'elemento grafico viene evidenziato disegnando dei quadratini grigi sugli angoli della figura. Individuato l'elemento è necessario spostare solamente l'origine dell'elemento senza tuttavia modificare le alte proprietà quali ad esempio altezza, larghezza e colore. Gli spostamenti dell'utente vengono seguiti utilizzando la funzione mouseDragged, la quale è stata propriamente implementata in modo tale che l'elemento grafico venga ridisegnato sulla view mentre l'utente tiene premuto il tasto al fine di seguirne i movimenti come in un qualunque tool di disegno. La funzione che si occupa di tale spostamento semplicemente sposta di un certo offset il rettangolo che incapsula la figura e richiama il metodo drawRect della view ogni qualvolta vi è uno spostamento maggiore di 4 pixel in x o in y. Tale accorgimento si è reso utile in quanto altrimenti l'applicazione non riusciva a ridisegnare in tempo il rettangolo e comportando quindi rallentamenti in fase di spostamento. La scelta della distanza di 4 pixel si è dimostrata ottima in quanto non rallenta l'applicazione e permette di spostare figure anche con un minimo spostamento del mouse.

Cambiare dimensioni degli elementi grafici

Come già spiegato nella precedente sezione, ogni qual volta viene selezionata una figura compaiono agli angoli dei rettangoli grigi, tali rettangoli hanno lo scopo di permettere all'utente di cambiare le dimensioni dell'elemento selezionato. Come prima la funzione mouseDown intercetta il click del mouse, controlla se esso è avvenuto all'interno dell'immagine oppure se è avvenuto su uno dei rettangolini, in tal caso cambia le dimensioni del rettangolo che incapsula la figura e richiama il metodo drawRect per visualizzarla aggiornata all'interno della view.

Cambiare il colore degli elementi grafici

Per quanto riguarda le proprietà degli elementi grafici come colore e spessore della linea, non è necessario creare un nuovo pannello manualmente ma è sufficiente creare un oggetto di tipo NSColorPanel, il quale istanzia un nuovo pannello per la scelta dei colori. Accedendo alle proprietà di tale pannello è poi possibile conoscere il colore selezionato dall'utente ed applicarlo all'elemento grafico selezionato.

5.1.2 Conversione di Coordinate: da coordinate Geografiche a coordinate in pixel

Generalmente utilizzando mappe fornite da MapKit, il framework permette di effettuare tramite la chiamata di una semplice funzione la conversione delle coordinate da quelle in latitudine-longitudine a quelle in pixel e viceversa. Poiché si è deciso di realizzare un tool per mappe e il relativo framework si è reso necessario lo studio di una strategia per implementare tale funzionalità.

Descrizione del problema

Si tratta di trovare una strategia per creare in modo efficiente una mappa geolocalizzata, ovvero una mappa nella quale sia possibile identificare un legame diretto tra la posizione del pixel in coordinate x-y e una posizione geografica in coordinate latitudine–longitudine. In questo modo è possibile riportare coordinate geografiche direttamente sull'immagine.

Soluzione individuata

Per prima cosa bisogna procurarsi la mappa geografica relativa al luogo che si desidera mappare. Utilizzando ad esempio Google Maps è possibile individuare l'area di interesse e cliccando sulla mappa ottenere le coordinate in latitudine-longitudine dei punti cliccati su di essa. Al fine di ottenere numeri facilmente comparabili bisogna convertire i valori della latitudine Nord e della longitudine Est dal sistema sessagesimale (gradi, minuti e secondi) a quello decimale. Ad esempio, se si considera Castel Sant'Angelo a Roma si ottiene:

$$lat = 41°54'10" = 41 + 54/60 + 10/3600 = 41,903$$
$$long = 12°27'59" = 12 + 27/60 + 59/3600 = 12,466$$

Per conoscere la conversione in pixel di un punto sulla mappa se le due mappe sono proporzionate è sufficiente calcolare le coordinate in latitudine-longitudine di tre estremi della mappa, ad esempio gli angoli in alto al fine di calcolare l'ampiezza della mappa sia in lunghezza che in larghezza espressa in gradi. Per ottenere lunghezza e larghezza è sufficiente calcolare le distanze tra due estremi utilizzando la formula di Pitagora:

$$distanza = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Tale formula viene utilizzata in geometria per calcolare la distanza tra due punti su di un piano. Essendo la terra sferica e non piana si potrebbe obiettare che tale formula non è quella corretta. Tuttavia nel caso di mappe per fiere o parchi divertimento le distanze ricoperte sono molto piccole, infatti ad esempio se si selezionano i due angoli opposti della mappa del parco divertimenti Gardaland, le coordinate risultano (45,456481, 10.707475) e (45,458483, 10,715768). Questi sono i punti a massima distanza sulla mappa eppure le due coordinate non differiscono nemmeno di un grado ma di poche unità. E' quindi possibile assumere in approssimazione che la distanza tra due punti all'interno della mappa sia uguale alla distanza tra due punti su di un piano. Tale approssimazione, oltre a ridurre il tempo di computazione dell'algoritmo per il calcolo delle distanze, viene utilizzata anche da MapKit nel calcolo di distanze piccole, nel caso invece di distanze maggiori il framework di iOS utilizza formule più complesse che prendono in considerazione che la terra è sferica e non piatta. Alcuni esempi di queste formule sono: la formula di Haversine e la formula di Vicenty. A questo punto avendo le coordinate degli estremi della mappa, non si deve fare altro che calcolare le

dimensioni della mappa in pixel (altezza e larghezza) e fare le dovute proporzioni. Per comprendere come vengano effettuate le proporzioni, si veda la fig. 5.1: sulla sinistra vi è la mappa in coordinate latitudine-longitudine con altezza H e larghezza W, mentre sulla destra la mappa in pixel di altezza h e larghezza w. Si supponga che sulla mappa a sinistra venga individuato un punto e si voglia sapere dove posizionare tale punto all'interno della mappa in pixel sulla destra. A questo punto è sufficiente utilizzare la seguente proporzione per ottenere ad esempio la coordinata x del punto (analogo per la coordinata y):

$$W : B = w : b$$

dove B indica la distanza del punto dal bordo della mappa. A questo punto l'unica incognita risulta essere b, che rappresenta la coordinata x del punto.

```
# coordinate da individuare sulla mappa in pixel
lat = XXXXXXX
lon = XXXXXXX

# larghezza e lunghezza della mappa in gradi
deltaLat = upperLeftLat - bottomLeftLat
deltaLon = upperRightLon - upperLeftLon

# larghezza e lunghezza della mappa in pixel
imgWidth, imgHeight = oggettoMappa.size

# coordinate calcolate
Y = imgHeight - int(float((lat - bottomLeftLat) * imgHeight) / float(
     deltaLat))
X = int(float((lon - upperLeftLon) * imgWidth) / float(deltaLon))
```

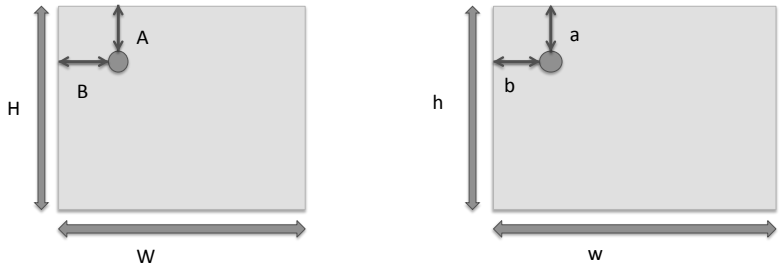

Figure 5.1: Mappa in coordinate (lat, long) a sinistra e mappa in coordinate (x,y) a destra. Dato il punto di coordinate (A, B) determinare le rispettive coordinate in pixel nella mappa di destra.

Nel caso le due mappe siano tra loro proporzionate tale conversione restituisce il risultato corretto, ma cosa succede nel caso ad esempio delle mappe di parchi divertimento che vengono disegnate senza rispettare le proporzioni? Tale algoritmo non riesce ad effettuare la giusta conversione e bisogna quindi adattarlo a tale caso. Innanzitutto bisogna aumentare la precisione, perciò bisogna ottenere un maggior numero di punti sulla mappa geografica ed individuare poi tali punti all'interno della mappa in pixel per effettuare la conversione con una precisione maggiore. Tali punti non devono essere presi casualmente altrimenti non si riesce ad effettuare una conversione efficiente in relazione a tali punti. Bisogna quindi prendere i punti sulla mappa geografica in modo tale da poter costruire all'interno del rettangolo che racchiude la zona da mappare, una sorta di griglia della quale si conoscono le coordinate dei punti relativi alle intersezioni di righe e colonne e ai bordi del rettangolo. In poche parole è come se si costruisse un foglio di carta millimetrata sulla zona d'interesse, di tale foglio sono note larghezza e lunghezza di ogni quadratino della griglia, se si riuscisse a costruire un foglio di carta millimetrata anche sulla mappa in coordinate pixel, allora sarebbe sufficiente conoscere in quale quadratino della mappa geografica cade il punto, individuare il corrispettivo quadratino nella mappa in pixel ed effettuare le solite proporzioni per individuare la x e la y. Se le due mappe sono proporzionate si è semplicemente ridotto ad uno spazio piccolo

la conversione e le formule di proporzione restano tuttavia le stesse. Quindi nel caso ci sia proporzionalità non ha molto senso suddividere il rettangolo delle due mappe in griglie, ma nel caso di mappe non proporzionate tale strategia si rivela efficace. Suddividendo la mappa geografica in piccoli quadratini (a seconda di quanto si vuole essere precisi), è possibile ricostruire la griglia della mappa in pixel, la quale non essendo proporzionata avrà quadratini di dimensioni diverse tra loro. A questo punto preso un qualunque punto all'interno di un quadratino della griglia la conversione delle coordinate di tale punto in coordinate pixel si riduce all'individuazione di un punto all'interno di un piccolo quadrato e non più dell'intera mappa perciò la precisione risulta maggiore. Come si può allora realizzare la griglia della mappa in pixel se ogni quadratino può avere una dimensione diversa, e come conoscere larghezza e altezza di ogni quadratino? Per conoscere l'ampiezza si è inserita all'interno del tool per realizzare mappe la possibilità di visualizzare la mappa di Google Maps della zona d'interesse e visualizzare affianco l'immagine disegnata della mappa. Costruita la griglia sulla zona geografica con il numero di righe e colonne specificato dall'utente in base alla precisione desiderata, la costruzione della griglia sulla mappa in pixel deve permettere all'utente di inserire manualmente dei marker per ogni punto individuato sulla griglia della mappa geografica. In questo modo ogni punto della griglia ha un suo corrispondente nella mappa in coordinate pixel ed è quindi possibile realizzare la griglia anche sulla mappa in pixel. Tale griglia non è composta da quadratini tutti della stessa dimensione, altrimenti sarebbe proporzionata, ma ogni quadratino può avere dimensione diversa. L'applicazione dovrà quindi salvare per ogni punto le coordinate in latitudine-longitudine e quelle in pixel corrispettive memorizzando a quale riga e quale colonna appartenga per poter ricavare la griglia al momento di effettuare la conversione. Nel caso di un quadratino completamente sproporzionato costruire la griglia con linee parallele diventa quindi impossibile. Si è pensato quindi di non ricostruire la griglia disegnandola sull'immagine ma semplicemente memorizzare le coppie di punti corrispondenti e salvarle come sempre nel file plist da generare. Il lavoro dell'applicazione per la realizzazione della mappa a questo punto è conclusa, sarà poi il framework in iOS ad occuparsi della conversione avendo a disposizione le coppie di punti e la loro posizione all'interno della griglia disegnata sulla zona geografica. Il framework quando riceve dal GPS un punto in coordinate latitudine-longitudine utilizza il seguente algoritmo per effettuare la conversione:

- Il GPS restituisce la posizione in coordinate (latitudine, longitudine), tale punto che sarà chiamato **P** avrà coordinate
$$(P_{lat}, P_{long})$$
- Viene individuato il quadratino all'interno del quale si trova tale punto. Note le coordinate degli angoli del quadratino in latitudine-longitudine, si trovano i corrispettivi in pixel semplicemente accedendo al dizionario che contiene le coordinate in pixel ed utilizzando come chiave l'indice di riga e quello di colonna di tali punti.
- Note quindi le coordinate in (latitudine, longitudine) del quadratino sulla mappa geografica e note le corrispettive coordinate del quadratino sulla mappa in pixel,si tratta ora di individuare la posizione del punto all'nterno del quadratino sulla mappa in pixel. Si tenga presente che sulla mappa in pixel in realtà non si dispone di un quadrato ma di un quadrilatero in quanto i lati non è detto siano paralleli. Tuttavia è comunque possibile effettuare l'individuazione del punto utilizzando le solite proporzioni già viste pre il caso di quadratini proporzionati.

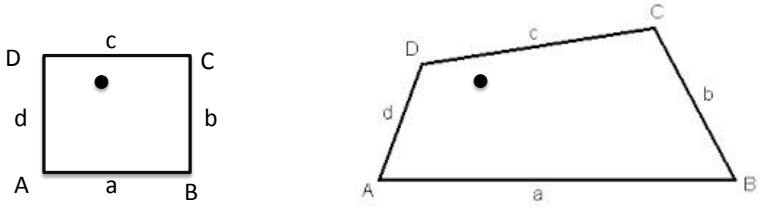

Figure 5.2: A sinistra quadratino della griglia della mappa geografica, a destra quadrilatero corrispettivo a tale quadratino

Visualizzare Google Maps con WebView

Per realizzare la conversione è necessario ottenere le coordinate dei punti del rettangolo che individua la zona geografica di interesse e gli eventuali punti della griglia nel caso in cui le due mappe non siano proporzionate tra loro. In OS X non è ancora presente un framework per la gestione di mappe, per fare in modo che l'utente non sia costretto ad individuare manualmente tali punti e doverli poi inserire all'interno dell'applicazione, si è utilizzata una webView sulla quale viene visualizzato Google Maps, e sulla quale si può quindi cercare la zona geografica che si desidera. Il caricamento di Google Maps viene effettuato utilizzando un file in formato .html, che si trova all'interno del progetto, tale file contiene uno script in JavaScript il quale sfruttando le API di Google carica la mappa. A questo punto bisogna però realizzare il rettangolo della zona di interesse, crearne la griglia in caso di non proporzionalità e rilevare le coordinate dei punti della griglia possibilmente facendo in modo che l'utente non debba inserire tali dati manualmente. Le possibilità per creare il rettangolo sarebbero state due: permettere all'utente di cambiare le dimensioni del rettangolo disegnandolo sopra, oppure permettere all'utente di spostare la mappa al di sotto del rettangolo ed effettuare uno zoom in modo che sia completamente contenuta all'interno del rettangolo. Si è pensato di utilizzare la seconda modalità, la quale permette così di muoversi all'interno di Google Maps senza che il rettangolo si muova assieme alla mappa. Per implementare tale modalità è bastato aggiungere una subview alla webview e portarla in primo piano in modo tale che il rettangolo fosse fissato sulla webview. A questo punto attraverso un semplice form l'utente può inserire il numero di colonne e di righe che vuole aggiungere al rettangolo per aumentare la precisione di conversione. Per ottenere quindi le coordinate dei punti della griglia in (latitudine, longitudine) senza farle inserire all'utente si è utilizzato nuovamente il file .html implementando una funzione che avesse come parametri le coordinate (x, y) in pixel della webView e restituisse le coordinate in (latitudine, longitudine) corrispondenti a tale punto. Nella versione implementata inserisce inoltre sul punto da mappare un marker che permetta di controllare la correttezza della funzione.

```
1  function getPointCoordinates(x, y)
2   {
3      var coordinates = overlay.getProjection().fromContainerPixelToLatLng(new google
           .maps.Point(x, y));
4
5          console.log(coordinates.jb + ", " + coordinates.kb);
6
7      marker = new google.maps.Marker({ position: coordinates, map: map});
8
9              return coordinates.jb + ", " + coordinates.kb;
10
11  }
```

Listing 5.1: Funzione che restituisce le coordinate in formato (latitudine, longitudine)

Visualizzare il disegno della mappa con IKImageView

Per visualizzare la mappa che è stata disegnata è necessario utilizzare una imageView, fortunatamente OS X mette a disposizione l'IKImageView, un kit completo che permette di gestire facilmente ed efficientemente la visualizzazione e la modifica delle immagini.

5.1.3 Inserimento dei Punti di Interesse

Per introdurre la possibilità di visualizzare pin e callout quando l'utente clicca su specifiche zone della mappa, si pensi ad esempio agli stand della fiera, è necessario che l'applicazione in OS X fornisca al framework iOS le coordinate di tali punti sulla mappa. Si è quindi inserita la possibilità di aggiungere dei marker sulla mappa e per ogni marker aggiunto, inserire la descrizione che si vuole sia visualizzata all'interno del rispettivo callout. Poichè non è presente il framework MapKit l'aggiunta dei Pin è stata implementata aggiungendo una nuova imageView, contenente l'immagine di un pin in formato png, e centrata nel punto cliccato dall'utente. Vengono quindi salvate in un dictionary tutte le informazione necessario: coordinate del punto, titolo e sottotitolo. In questo

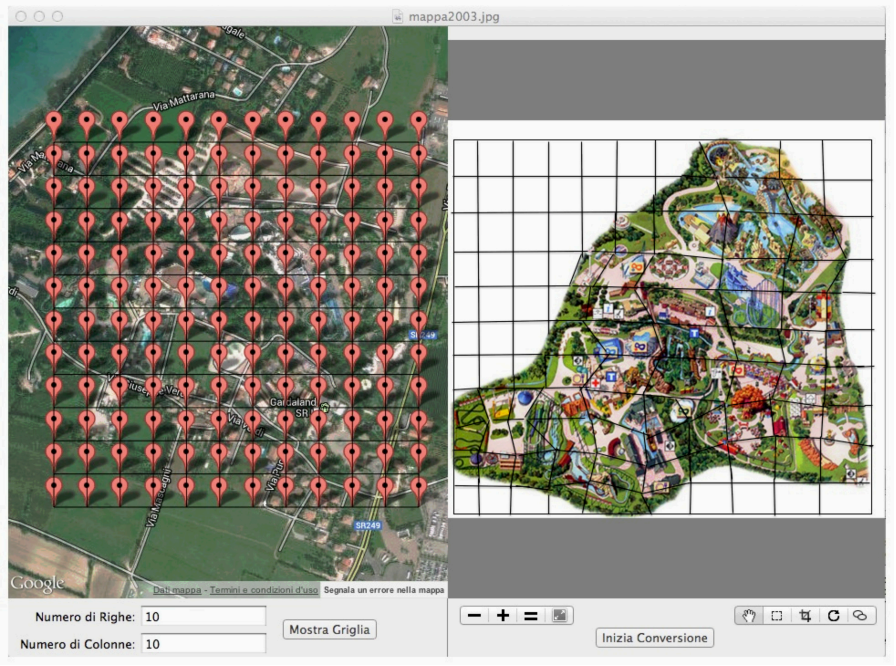

Figure 5.3: Screenshot relativo al tool per la conversione di coordinate integrato nell'applicazione

modo il framework iOS dispone di tutte le informazioni necessarie per visualizzare il pin ed il callout quando l'utente clicca sulla zona interessata.

5.1.4 Generazione del file che descrive la mappa

Terminato il disegno della mappa e la conversione delle coordinate è possibile produrre un file in formato .plist il quale contiene una descrizione della mappa ordinata e dettagliata. Nel caso in cui la mappa sia stata disegnata utilizzando l'applicazione, allora nel file plist viene salvato un dizionario che possiede come chiave il nome dell'elemento e come valore un dizionario delle proprietà di tale elemento grafico. Tali proprietà serviranno poi al framework realizzato in iOS che le utilizzerà per ridisegnare l'immagine. Le proprietà definite nel dizionario per ogni elemento grafico sono:

- Colore di riempimento
- Colore della linea che individua il bordo
- Dimensioni dell'elemento:altezza, larghezza, coordinate del punto di origine già convertite dall'applicazione nel sistema di coordinate di iOS
- Spessore della linea
- Tipologia di elemento grafico, ad esempio rect, circle, text, ecc.

In questo modo è possibile ricreare la stessa immagine prodotta con l'applicazione OS X all'interno di una applicazione per iOS. Se invece l'immagine non è stata creata con l'applicazione in quanto si è utilizzata l'applicazione solamente per la conversione di coordinate, allora il file plist conterrà solamente il nome del file dell'immagine e due dizionari. Tali dizionari servono a memorizzare le coordinate in (latitudine, longitudine) e in pixel. Per individuare velocemente il punto in coordinate (latitudine, longitudine) e il suo corrispettivo si è scelto di utilizzare, per le coppie di punti correlate, la stessa chiave in modo da velocizzare la ricerca. La chiave è costituita da indice di riga e indice di colonna della griglia. Il valore è invece un punto espresso nel sistema di coordinate indicato dal dizionario.

5.2 Realizzazione del Framework in iOS

Tale framework è lo stesso framework realizzato per l'applicazione Pulire 2.0 con l'unica differenza che la sorgente può non essere un'immagine.

5.2.1 Sorgente della mappa:file plist

Nel caso in cui l'applicazione in OS X sia stata utilizzata per disegnare la mappa, allora non si possiede un'immagine della mappa ma la descrizione di ogni elemento grafico presente sulla scena è interamente contenuta all'interno del file plist. In tale caso quindi l'applicazione legge dal file plist solo la prima volta che viene lanciata tutte le informazioni riguardanti gli oggetti presenti sulla scena e le salva in locale. In questo modo non deve accedere al plist più volte e l'applicazione risulta più veloce in fase di caricamento della mappa. Ottenute tali informazioni la view principale implementa il metodo drawRect ed utilizza per ridisegnare gli oggetti le stesse funzioni utilizzate da OS X quando l'utente le disegna. La struttura della funzione è la stessa ma non bisogna dimenticare che i framework OS X e iOS non utilizzano gli stessi oggetti. Questo è stato uno dei principali problemi nella lettura del file Plist in quanto incapsulando in stringhe certi tipi di oggetto propri di OS X, il framework iOS non sapeva come interpretarli e tradurli. Per comprendere meglio si pensi ad esempio all'oggetto NSColor di OS X che memorizza il colore dell'elemento grafico. Richiedendo a tale oggetto il colore, esso non restituisce le componenti RGB, ma restituisce una stringa contenente il nome dello spazio dei colori, ad esempio se si utilizzano solo colori della scala di grigi restituisce NSCalibratedWhiteColorSpace. Tale stringa non viene riconosciuta dal framework iOS in quanto la descrizione dei colori di un elemento grafico in iOS non specifica lo spazio dei colori. In questo caso si è quindi pensato di incapsulare nella stringa da inserire nel file plist solamente le componenti RGB e $\boldsymbol{\alpha}$.
Al momento della lettura del file plist tali informazioni verranno incapsulate nell'oggetto di iOS adibito per la memorizzazione dei colori: UIColor.

5.2.2 Sorgente della mappa:file immagine

Questo è il caso in cui la mappa è stata creata con un qualunque altro tool (AutoCAD, Photoshop) e l'applicazione è stata utilizzata per effettuare la sola fase di geolocalizzazione e/o di inserimento dei punti di interesse. In tal caso il file plist contiene solo i dati utili ad effettuare la conversione. Esattamente come nel caso precedente le informazioni relative alla conversione vengono memorizzate in locale quando l'applicazione viene lanciata senza accedere continuamente al file plist. Per visualizzare la mappa non è possibile utilizzare il file caricato dall'utente nell'applicazione di OS X, in primis per la qualità dell'mmagine e anche perché esso non può essere memorizzato nel file plist interamente. Esso deve quindi essere caricato all'interno del framework e suddiviso in tiles per migliorare la qualità al momento della visualizzazione della mappa.

Chapter 6

Testing

6.1 Pulire 2.0

La fase di testing dell'applicazione realizzata può essere suddivisa in due fasi: la prima è stata realizzata durante l'implementazione dell'applicazione mentre la seconda è stata realizzata in fase di utilizzo dell'applicazione nei giorni di esposizione. Per quanto riguarda la prima fase essa è servita a testare l'applicazione prima che fosse inserita nello store e quindi utilizzata dagli utenti in modo da poter migliorare alcuni aspetti e risolvere eventuali bug. E' possibile utilizzare alcuni servizi esterni per distribuire la versione beta della propria applicazione affinché essa possa essere provata non solo dagli sviluppatori, i quali possono utilizzare l'applicazione sul proprio dispositivo in quanto dispongono di un particolare certificato che permette di installare l'applicazione sul proprio dispositivo, ma anche da altri utenti. In tale contesto disporre di un servizio per la distribuzione della versione beta è risultato fondamentale per permettere all'ente di visionare l'applicazione prima del lancio in modo da segnalare eventuali richieste di modifiche prima della distribuzione dell'applicazione. Non solo ma in tal modo è stato possibile fare in modo che il flusso di sviluppo fosse continuamente seguito dall'ente e quindi modificato in itinere in base alle esigenze. Il servizio utilizzato è TestFlight, il quale è un servizio gratuito disponibile sia per iOS che per android, che realizza tale servizio. E' sufficiente registrarsi al sito, creare un team, invitare i membri del team, registrare i dispositivi dei membri ed infine caricare sulla piattaforma di TestFlight la propria applicazione e distribuirla. Tale servizio simula il comportamento dell'AppStore vero e proprio ma permette di ottenere dei feedback da parte dei membri del team e monitorare l'attività dei tester. Superata tale fase e distribuita l'applicazione sull'AppStore, la seconda fase di testing è avvenuta durante i tre giorni della fiera su visitatori ed espositori, i quali hanno utilizzato sul campo l'applicazione. Se l'applicazione si trova sullo store è possibile accedere ad una statistica del numero di download, degli eventuali crash della propria applicazione ed ottenerne così un quadro del comportamento. Non sono stati registrati crash in fase di utilizzo e non sono stati segnalati dagli utenti. E' tuttavia necessario specificare che ogni applicazione prima di essere caricata sullo store richiede un periodo di circa 15 giorni per essere testata da Apple perciò se l'applicazione mostra crash o altri bug evidenti non viene accettata e quindi distribuita. Perciò superata la fase di approvazione da parte di Apple l'applicazione raramente può avere bug che causino il crash dell'applicazione durante l'utilizzo.

6.2 Tool per mappe vettoriali e framework

Al contrario di Pulire 2.0 l'applicazione realizzata non è ancora entrata in produzione perciò non è ancora stata testata su casi reali. Alcuni dei principali problemi riscontrati sono stati warning relativi alla memoria in fase di testing della visualizzazione della mappa realizzata con il tool e visualizzata grazie al framework iOS. Tali problemi non venivano riscontrati sul simulatore in fase di programmazione, ma sono stati registrati in fase di testing su dispositivo (iPod terza generazione). Si è notato che tale warning in fase di zoom in o zoom out era dovuto alla richiesta di disegnare nuovamente la mappa mentre l'utente cliccava su di essa per aumentarne o diminuirne il livello

di zoom. La soluzione è stata diminuire il massimo fattore di scala che si può applicare alla scrollView. Riducendo tale fattore ad un valore di 2.5 il warning è sparito in quanto lo zoom in massimo effettuabile sulla mappa viene ridotto e con esso vengono diminuito il numero di chiamate alla funzione drawRect per ridisegnare la mappa in base al livello di zoom in modo da mantenerne la qualità. Tale valore permette comunque di visualizzare ogni dettaglio della mappa senza tuttavia appesantirne la computazione. La funzionalità di georeferenziazione della mappa si è verificata corretta durante la fase di testing su casi d'esempio.

Chapter 7

Conclusioni e Sviluppi Futuri

Per quanto riguarda la prima parte, il framework realizzato è stato effettivamente utilizzato all'interno del progetto di Pulire 2.0 per il rendering e la gestione della mappa relativa allo spazio espositivo della fiera. Non sono stati riscontrati crash in fase di utilizzo su nessuna tipologia di dispositivo utilizzato durante il periodo di fiera o problemi di rallentamento dell'applicazione dovuti a warning della memoria. Il framework ha quindi rispettato le aspettative risultando robusto a qualunque tipologia di dispositivo.
Per quanto riguarda invece la seconda parte, il framework utilizzato per iOS si basa su quello sviluppato per Pulire 2.0 con la differenza di prendere in input un file in formato .plist invece che un'immagine e di realizzare la conversione di coordinate non implementata per la fiera.La fase di testing è stata pertanto ridotta alla sola parte di conversione delle coordinate e al rendering della mappa caricandola da file plist. I problemi di memoria riscontrati su dispositivi con velocità di processore minore (Processore iPod di terza generazione: Apple A4 chip - Processore iPhone5: - Apple A6 chip) sono stati rilevati e risolti. La conversione di coordinate è stata testata ed è risultata corretta, si potrebbe tuttavia trovare una metodologia per velocizzare le operazioni di conversione lato framework iOS. Per quanto riguarda l'applicazione OS X per realizzare le mappe, sarebbe interessante riuscire ad individuare una strategia per creare la griglia della mappa disegnata dall'utente senza dover far inserire all'utente manualmente i marker. Inoltre sarebbe interessante migliorare la gestione delle view quando l'utente aumenta o diminuisce le dimensioni della finestra, in particolar modo il comportamento a volte anomalo delle scrollView all'interno delle window. La grafica poi potrebbe essere migliorata rendendo più chiare all'utente le funzionalità di ogni componente presente sulla window (bottoni, textfield, ecc..).
Infine, per quanto riguarda l'esperienza di tirocinio, essa si è rivelata molto positiva portando all'acquisizione di competenze che spaziano anche al di fuori del singolo progetto realizzato: una conoscenza del linguaggio di programmazione Objective-C, della piattaforma di sviluppo iOS e di quella OS X, e la capacità di gestire un progetto di discrete dimensioni, curarne la progettazione del software, l'implementazione ed il testing.

Ringraziamenti

- In primis ringrazio Matteo, Susanna e Alessandro che in questi mesi di tirocinio presso Altera mi hanno permesso di imparare moltissimo all'interno di un ambiente lavorativo sempre divertente e dinamico.
- Ringrazio il professor Sergio Congiu per la disponibilità dimostrata in questi mesi nel seguirmi in questo percorso di tirocinio.
- Ringrazio la mia famiglia: Mamma, Papà e Marco che mi hanno sempre sostenuta e motivata ad andare avanti in questo percorso.
- Ringrazio Giorgio per esserci sempre stato e aver sempre creduto in me.
- Ringrazio Francesca per le sue serate canto, ballo, corsa, ciaccole che mi hanno tenuta attiva in questi mesi.
- Ringrazio Lucia (la mia patatina) e Federica, le mie uniche compagne di viaggio donne in questo percorso di studi con le quali ho condiviso momenti indimenticabili.
- Alberto, Ale, Alvi, Bea, Bedo, Daniele, Emma, Federico, Marty, Sisko, Vanuz grazie per le serate giochi in scatola, gli spritz, le pause caffè. In particolare un grazie a Sebastian che in questi mesi mi ha sostenuta e incoraggiata a non mollare. Sei un migliore amico perfetto e poi sai anche cucinare! Grazie anche a Deba per i Rosssss urlati nei corridoi del dei quando ancora era uno studente e ora che sei fuori dal mondo universitario grazie per tutte le serate con Giò. Grazie a tutti voi altrimenti il dei non sarebbe stato lo stesso.

Bibliography

[1] David Mark , Jack Nutting , Jeff LaMarche , Fredrik Olsson, *Beginning iOS 6 Development: Exploring the iOS SDK*, Apress

[2] David Chisnall, *Objective-C Phrasebook, 2nd Edition*, Addison-Wesley Professional

[3] Neil Smyth *iPhone iOS 6 Development Essentials*

[4] Apple Inc. (2012), iOS Technology Review, Apple Inc., Cupertino.

[5] *MAC OS X developers OSX Technology Overview website*, Apple Inc., Cupertino.