ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA

SCUOLA DI INGEGNERIA E ARCHITETTURA
Corso di Laurea in Ingegneria Informatica

SVILUPPO DI APPLICAZIONI MOBILE: CONFRONTO FRA PIATTAFORME NATIVE E TECNOLOGIE WEB UTILIZZANDO IOS E PHONEGAP COME CASO DI STUDIO

Elaborata nel corso di: Sistemi Operativi LA

Tesi di Laurea di:
FORTUNATO DE MARTINO

Relatore:
Prof. ALESSANDRO RICCI

Co-relatore:
Ing. ANDREA SANTI

ANNO ACCADEMICO 2012-2013
SESSIONE I

PAROLE CHIAVE

Sviluppo applicazioni mobile

iOS

PhoneGap

HTML5

JavaScript

Ai miei genitori, ai miei fratelli,
ai miei nonni e alla mia fidanzata.

Indice

Introduzione

Negli ultimi anni il mondo del mobile computing ha avuto una vera e propria crescita esponenziale grazie soprattutto all'entrata in scena dello smartphone. In realtà, per essere più precisi, è bene sottolineare che gli smartphone esistevano già da tempo ma il loro utilizzo era in particolar modo indirizzato ai professionisti per il quale era, ma continua ad essere tutt' oggi, un valido supporto in campo lavorativo, basti pensare all'importanza della comunicazione via e-mail e non solo. Seppur comunque fossero già presenti da tempo, i primi smartphone non godevano di certo di un touch-screen sofisticato come quello odierno né in essi erano presenti funzionalità tipiche dei dispositivi che troviamo ad oggi sul mercato. Una svolta decisiva è stata segnata dall'introduzione dell'iPhone e successivamente dell'AppStore, grazie a questi la programmazione per i dispositivi mobile ha preso sempre più piede diventando un vero e proprio business. In un secondo momento alla programmazione nativa si affiancarono le tecnologie web. Questo mio lavoro di tesi si pone l'obiettivo di studiare in primis la struttura, caratteristiche e peculiarità del sistema operativo iOS e analizzare il framework PhoneGap al fine di riuscire a confrontarne i vari aspetti fondamentali anche attraverso lo sviluppo di piccole applicazioni. Così facendo, quindi scendendo nei dettagli di quelle che possono essere le differenze rilevanti, mi pongo l'obiettivo di valutarne relativi pro e contro al fine di fare una scelta del tutto personale tra iOS e PhoneGap. Il lavoro di tesi sarà diviso in 5 capitoli:

- Il primo capitolo ha il compito di redigere un quadro generale, descrivere l'evoluzione del mobile computing, esplicitare il significato dello sviluppo di applicazioni per piattaforme native e definire l'approccio cross-platform legato allo svilupo delle web app.

- Nel secondo capitolo viene descritto il sistema operativo iOS, nello specifico si evidenzia la sua struttura staficata a livelli e si presentano i vari framework e funzionalità che compongo tali livelli. In un

secondo momento viene descritta in dettaglio l'organizzazione di un' applicazione in iOS.

- Nel Terzo capitolo si descrive la struttura di PhoneGap insieme alle sue API e successivamente si fa luce su quella che viene definita organizzazione di un' applicazione web.

- Nel quarto capitolo, considerato chiave di questo lavoro di tesi, grazie allo sviluppo di piccole app si cerca di formulare quali sono i pro e i contro dell'una e dell'altra piattaforma messe a confronto, iOS e PhoneGap, così da poter giungere ad una conclusione che mi porti personalmente a scegliere una metodologia di sviluppo di un'app piuttosto che l'altra.

- Quinto e ultimo capitolo è sede di quelle che sono le mie personali considerazioni e relative conclusioni in merito al lavoro svolto per la stesura della mia tesi.

Capitolo 1

Quadro generale

1.1 Evoluzione del Mobile computing

Nell'ultimi decennio il mobile computing in generale ha avuto una vera e propria rivoluzione, stesso discorso vale anche per gli smartphone che stanno diventando sempre più, anzi lo sono già, un vero e proprio PC portatile. Guadando un pò indietro i cellulari non erano altro che un appendice del telefono fisso e quindi usati solo come mezzo di comunicazione con l'aggiunta tutt' al pi u degli sms (Short Message Service). Nel susseguirsi degli anni si aggiunsero altre funzionalità come piccoli giochi (snake), radio , calcolatrice, agenda, internet e così via fino ad arrivare al 2008 con l'uscita dell'iPhone di casa Apple, in quell'anno ci fu una vera e proprio rivoluzione. Da allora cambiò il modo di usare gli smartphone infatti con il lancio di AppStore le applicazioni sono diventate uno dei pricipali modi per comunicare, organizzare la vita, giocare e di facile utilizzo come mezzo di supporto per il lavoro. Le applicazioni non sono assolutamente paragonabili alle funzioni sopra descritte come agenda, giochi e altro, ma sono delle vere e proprie applicazioni con una loro interna complessità, organizzazione e una attenta ingegnerizzazione. Le applicazioni mobile devono saper interagie con alcune risorse presenti negli smarphone di oggi, quali, girometro, fotocamera, GPS e così via. Attualmente le aziende che decidono di sviluppare applicazioni mobile devono essere molto organizzate poiché lo sviluppo di tali applicazioni prevede l'uso di strumenti di alto livello e idonei a ogni piattaforma esistente come iOS, Android, Bada e Blackberry. Per quanto riguarda l'uso di strumenti che permettono a una software house di poter avere un solo team per lo sviluppo di applicazioni eseguibili in tutte le piattaforme esistenti verrà approfondito più avanti.

1.2 Programmazione nativa

Implementare software nativo significa sviluppare applicazione per piattaforme specifiche che abbiano un proprio ambiente di sviluppo, un proprio SDK(Software development kit) e un proprio codice. Chi sviluppa in liguaggio nativo usa ambienti di sviluppo e SDK (Software development kit) in constante aggiornamento, ció comporta, per un programmatore o per una software house, di rimanere sempre aggiornati. Voler scrivere un'applicazione per ogni tipo di piattaforma significa conoscere per ogni sistema operativo un skill set

Mobile OS Type	Skill Set Require
Apple iOS	C, Objective C
Google Android	Java (Harmony flavored, Dalvik VM)
RIM BlackBerry	Java (J2ME flavored)
Symbian	C, C++, Python, HTML/CSS/JS
Windows Mobile	.NET
Windows 7 Mobile	.NET
HP Palm webOS	HTML/CSS/JS
MeeGo	C, C++, HTML/CSS/JS
Samsung bada	C++

Ció che rende le cose complicate sono le differenze tra i vari SDK delle diverse piattaforme che contengono strumenti e interfacce necessarie per sviluppare, installare e testare le applicazioni native. Quindi ogni piattaforma usa diversi tools, build systems, API, e dispositivi con capacità diverse. In realtà una sola cosa accomuna tutti i Sistemi Operativi cioé quella di essere forniti di un Browser mobile che è accessibile attraverso codice nativo. In conclusione diventa molto chiaro il fatto che voler sviluppare un'applicazione software per ogn tipo di piattaforma diventa alquanto dispendiosa per una software house poiché tale fine richiede la presenza di più team ciascuno specializzato in una determinata piattaforma.

1.3 Web app - Cross Platform

Lo sviluppo di nuovi sistemi operativi ha portato gli sviluppatori a essere sempre aggiornati per poter costruire software per ogni piattaforma. Nasce la necessità di sviluppare applicazioni che siano eseguibile su tutte le piattaforme esistenti, quindi sviluppare un'applicazione su una piattaforma comune. Nel 2000 abbiamo visto una situazione simile nel mondo desktop,

avevamo Microsoft Windows, varie versioni di Linux e Unix e Mac di Apple. Era difficile costruire applicazioni che potevano funzionare su tutti queste piattaforme. Java Sun fu il primo a fornire una piattaforma comune su cui costruire software eseguibili su tutti i sistemi operativi. Tra il 2004 e il 2008 gli sviluppatori trovarono una frammentazione tra i browser poiché non tutti seguivano le specifiche indacate dal World Wide Web Consortium (W3C). In questo caso molte librerie JavaScript hanno contribuito a scrivere applicazioni web cross-broswer. Nel mondo mobile ci fu la medesima situazione, per le software house era un problema essere sempre aggiornate su tutte le piattaforme. Questa difficoltà portò alla creazione di framework cross-platform, tale soluzione fu indolore poiché tutte i sistemi operativi mobile accedono al broswer web tramite codice nativo, inoltre la cosa in comune che avevano i vari SO mobile era il broswer web. Nitobi, creatore di phonegap poi acquisito da Adobe, creó il primo framework cross-platform basato su tecnologie web, si potevano creare applicazioni scritte con linguaggio comune come HTML5, JavaScript e CSS3. Dopo phonegap nacquerò altri framework cross-platform come Titanium, JQuery Mobile e altri.

1.3.1 Framework

Al fine di realizzare applicazioni web pssono essere utilizzati determinati framework, nello specifico qui di seguito citiamo:

- **jQuery Mobile:** Questo framework offre la possibilità di creare siti mobile e applicazioni per piattaforme iOS , Android e così via. Basato su jQuery ed ispirato al progetto UI,sempre di jQuery, offre una buona stabilità ed un certo numero di funzionalità adeguate per sviluppare appliazioni web mobile. Questo framework fornisce un insieme di widget UI touch-friendly e un sisteme di navigazione AJAX. Per realizzare applicazioni mobile basterà usare HTML5.[10]

- **Titanium Mobile:** consente lo sviluppo rapido di applicazioni mobile (iOS, Android e altro) in JavaScript. La piattaforma contiene un layer per ogni piattaforma che consente di richiamare ed utilizzare tutti i controlli nativi dell'interfaccia grafica offerta da ognuno dei sistemi operativi supportati. Il codice dell'applicazione che si sviluppa in Titanium Mobile non contiene html, ma una vera e propria applicazione JavaScript.[7]

- **Sencha Touch:** è un framework completamente basato su standard web come HTML5, CSS3 e JavaScript permettendo di sviluppare ap-

plicazioni web per piattaforme come iOS, Android, BlackBerry e altro. Esso include una serie di interfacce grafiche ottimizzate per il touch.[8]

- **The-M-Project:** è un software open source pubblicato con licenza MIT. Esso è un framework basato su Javascript ed è basato sul pattern Model-View-Controller. [4]
- **PhoneGap:** verrà trattato ampiamente in seguito [1].

Capitolo 2

iOS

iOS é il sistema operativo creato dalla Apple e correntemente installato su dispositivi quali iPhone, iPad, e iPod. Tale Sistema Operativo, oltre a gestire tutto l'hardware del dispositivo, fornisce le tecnologie necessarie per realizzare sia applicazioni native che applicazioni web, quest' ultime realizzate utilizzando una combinazione di HTML, CSS e JavaScript eseguite all'interno del browser web. Il SO è fornito di alcune applicazioni di sistema, impossibili da cancellare, come il telefono, Mail e il broswer web Safari che fornisco all'utente finale servizi standard utili per l'uso adeguato dei vari dispositivi. Per potere sviluppare, testare e installare applicazioni native nei dispositivi Apple, iOS SDK fornisce una serie di strumenti e interfacce. Tali applicazioni vengono costruite utilizzando un framework di sistema insieme al linguaggio di programmazione Objective-C, con la possibilità di innestare qualche comando di linguaggio C. Apple fornisce una serie di framework che contengono interfacce di sistema. Un framework non é altro che una directory che contiene una libreria condivisa e delle risorse, come ad esempio immagini, helper apps, header file e così via, necessarie per sostenere tale libreria. Molte delle tecnologie usate per sviluppare applicazioni iOS le possiamo ritrovare nel sistema operativo OS X.

2.1 Objective-C

Objective-C è definito linguaggio di programmazione riflessivo orientato agli oggetti,fu sviluppato da Brad Cox nella metà degli anni ottanta presso la Stepstone Corporation. Nel 1988 la NEXT,compagnia fondata da Steve Jobs dopo che lasciò Apple, ottene la licenza da Stepstone e realizzò il proprio compilatore Objective-C e le relative librerie. Nel 1996 Apple acquistò

NEXT ottenendo la licenza Objective-C. Objective-C è un linguaggio di programmazione orientato agli oggetti, dinamico, non fortemente tipizzato derivante da SmallTalk (il primo linguaggio a oggetti) e da C con il quale mantiene una completa compatibilità. Le estensioni a oggetti con cui Objective C arricchisce il modello semantico del C sono appunto ispirate al linguaggio, già pocanzi citato, Smaltalk e in modo particolare alla relativa gestione dei messaggi. Sono le caratteristiche tipiche del runtime system che rendono di facile collocazione l'Objective-C tra i linguaggi ad oggetti dinamici. Possiamo definire il Runtime come: libreria che viene staticamente linkata ad ogni programma Objective-C. In sostanza il Runtime agisce come una sorta di macchina virtuale in cui "vivono" gli oggetti Objective-C e inoltre mette a disposizione delle API che consentono ad esempio di conoscere per ogni oggetto a quale classe appartiene, i metodi e le proprietà che possiede. Essendo Objective-C un cosiddetto superset di C allora si possono tranquillamente utilizzare tutti i metodi e funzioni del C in maniera nativa. Quindi sono supportati tutti gli elementi caratteristici della programmazione a oggetti ma al contempo non mancano concetti comunque innovativi come il meccanismo delle categorie e strumenti legati alla riflessione. La diffusione di tale linguaggio è principalmente connessa al framework OpenStep di Next e al suo successore Cocoa presente nel sistema operativo Mac OS X di Apple. Inoltre a Nex si deve il suporto dell'Objective C nel compilatore gcc di GNU. Oggi Objective C è nella maggioranza dei casi usato in concomitanza con librerie fisse di oggetti standard denominati "kit" o "framework" come Cocoa e GNUstep. Per quanto riguarda queste librerie possiamo sottolineare che esse sono spesso fornite insieme al sistema operativo, in tal modo il programmatore non è obbligato ad ereditare le funzionalità della classe base esistente ma con l'utilizzo di Objective C si ha la possibilità di dichiarare nuove classi base che non ereditino nessuna delle funzionalità già preesistenti. In origine gli ambienti di programmazione basati su Objective C offrivano la classe Object con alcune funzionalità di base ma, con l'introduzione di OpenStep, Next ha dato vita ad una nuova classe base chiamata NSObject che offre caratteristiche del tutto aggiuntive rispetto a quella di Object. La stragrande maggioranza delle classi di Cocoa ereditano da NSObject.

2.2 l'architettura iOS

In maniera simile al sistema operativo OS X anche l'architettura iOS è divisa in 4 layers, nel livello più alto iOS agisce da intermediario tra l'hardware sottostante e le applicazioni. Nei livelli più bassi del sistema sono racchiusi i servizi e le tecnologie fondamentali su cui si basano tutte le applicazioni. Queste ultime, come è giusto che sia, non comunicano direttamente con l'hardware sottostante, per evitare che le applicazioni stesse modifichino l'hardware stesso, ma comunicano con esso attraverso delle interfacce ben definite. Tale astrazione, oltre a proteggere l'hardware, facilita lo sviluppo di applicazioni che possono essere eseguite su dispositivi con diverse capacità hardware. Per lo sviluppo di applicazioni è preferibile l'uso dei framework di livello superiore, poiché questi forniscono astrazioni orientate agli oggetti per costrutti di livello inferiore, così facendo si riduce la quantità di codice scritto incapsulando funzionalità complesse. Ciò comunque non vieta l'uso di framework di basso livello. È importante specificare che oltre all'uso di framework, descritti nel paragrafo precedente, apple fornisce delle librerie standard, la maggior parte appartenenti ai livelli inferiori.

2.3 l'architettura iOS stratificata

Come accennato prima l'architettura di iOS è divisa in quattro differenti livelli di astrazione, vedi figura 1.1, ognuno dei quali racchiudono delle tecnologie che trattaremo nei prossimi paragrafi.

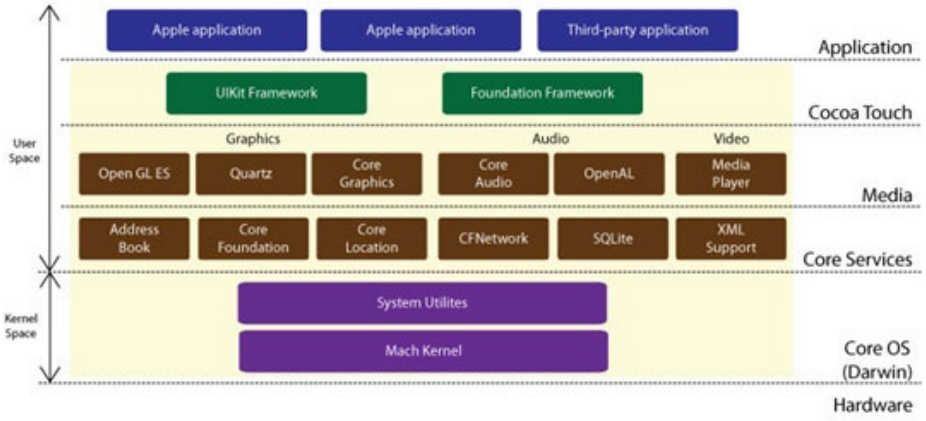

Figura 2.1: Architettura iOS

2.3.1 Core OS layer

In tale livello sono contenute le funzioni di basso livello che mi consente di interagire con l'hardware sottostante. Il programmatore ha la libertà di scegliere o meno l'utilizzo di tali tecnologie per lo sviluppo delle proprie applicazioni poiché sicuramente per il tipo di architettura sono utilizzati da altri framework di livelli sovrastanti.

Accelerate Framework: Il framework contiene interfacce per l'esecuzione di DSP (digital signale processor), calcoli matematici e elaborazione delle immagini; più precisamente esso contiene una serie di API C per l'utilizzo di vettori, matrici, elaborazione di segnali digitali e elaborazione di grandi numeri. Nello specifico l'API VDSP (Accelerate/Accelerate.h) fornisce funzioni matematiche per applicazioni come l'elaborazione audio e video, la diagnostica delle immagini mediche, l'elaborazione del segnale radar, analisi sismica e l'elaborazione di dati scientifici. Tale framework fù introdotto nell'iOS 4.0

Core Bluetooth Framework: Consente agli sviluppatori di interagire con Bluetooth 4.0 LE (low energy), le interfacce Objective-C di questo framework permettono la scansione degli accessori, il collegamento e lo scollegamento dei device, la scrittura e lettura dei parametri all'interno di un servizio e molto altro.

External Accessory Framework: è introdotto in iOS 3.0 e fornisce un canale per comunicare con accessori collegati ad un dispositivo iOS-

based. Tali accessori possono essere collegati tramite il connettore dock a 30 pin, tramite il mini connetore a 9 pin di ultima generazione oppure in modalità wireless. Gli svilppatori possono utilizzare questo framework per poter ottenere informazioni dai vari accessori collegati,avendo anche la possibilità di manipolarli tramite comandi opportuni. Per poter interagire con un accessorio è neccessario che il produttore di accessori costruisca un supporto per comunicare con iOS, tale supporto deve contenere almeno un protocollo di comando per inviare e ricevere dati dall'applicazione all' accessorio e viceversa.

Generic Security Services Framework: fornisce un insieme di servizi legati alla sicurezza. Le interfacce contenute in questo framework sono specificate in IETF RFC 2743 e RFC 440, inoltre sono presenti alcuni servizi aggiunti come la gestione delle credenziali.

Security Framework: mette a disposizione un framework per garantire la sicurezza dei dati che un'applicazione gestisce. Contiene altre interfacce per la gestione dei certificati, chiavi pubbliche o private e la generazione di numeri pseudocasuali crittografati. Inoltre memorizza i certificati e le chiavi nel portachiavi che é un archivio sicuro per i dati sensibili dell'utente. Il framework in questione contiene la libreria Crypto che pervede un supporto per la crittografia simmatrica, HMAC (keyed-hash message authentication code) e digest(funzione compatibile con la libreria OpenSSl non disponibile in iOS). È possibile condividere il portachiavi tra più applicazioni create così da facilitare l'interazione tra applicazioni della stessa suite.

System: comprende il kernel, i driver e le interfacce Unix di basso livello. I driver di questo livello forniscono l'interfaccia tra l'hardware e i frameworks di sistema. Il nucleo di iOS,come quello di Mac OS X, include il kernel e la base UNIX del sistema operativo noto come Darwin, sistema operativo open source pubblicato da Apple che fornisce solo il kernel e lo spazio base di utenti, strumenti e servizi tipici dei sistemi UNIX. Il Darwin SO (iOS e Mac OS X) è costituito dal kernel XNU Darwin che contiene tre componenti di base strutturati in livelli e sono: Mach layer, BSD layer e IØKits. Il Mac microkernel è responsabile di ogni aspetto del sistema, gestisce il sistema della memoria virtuale, il file di sistema, threads e la comunicazione fra i processi. Lo strato BSD è basato sul kernel BSD (FREE BSD) e fornisce API e servizi; il componente BSD quindi prevede file systems, interfaccia / protocolli di rete, modello di sicurezza UNIX, chiamate di sistema, Modello di processo BSD, FreeBSD kernel API, molte delle API POSIX, e il supporto del kernel per pthread (thread POSIX). IØKits è un framework object-oriented che serve per lo sviluppo di driver e il caricamen-

to dinamico di moduli. Quindi le applicazioni possono fare direttamente uso di Mach e BSD. iOS quindi fornisce una serie di interfacce per accedere a molte funzioni di basso livello del sistema operativo. un'applicazione accede a queste funzioni tramite la libreria Libsystem.

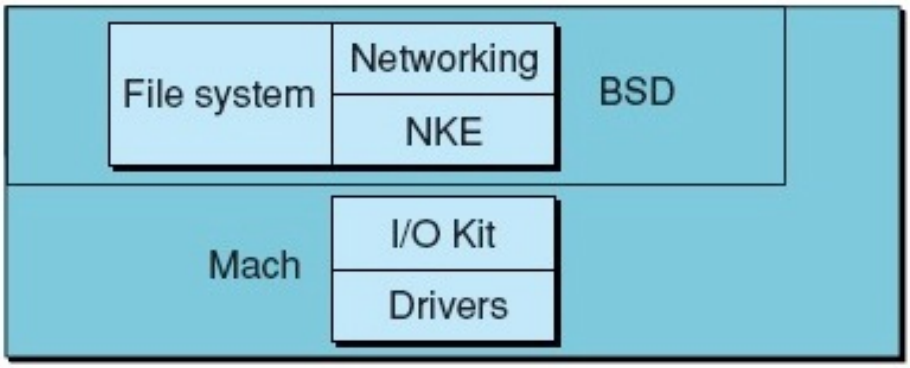

Figura 2.2: Architettura del Kernel

2.3.2 Core Service layer

Il Core Service layer contiene una serie di servizi che tutte le applicazioni possono utilizzare. Anche se alcune applicazioni non utilizzano questi servizi, molte parti del sistema sono costruite su di essi. Tale livello, come il Core OS layer, si occupa di offrire appunto i servizi fondamentali del sistema operativo.

High-Level Features

iCloud: Integrato a partire da iOS 5 e Mac OS X, nasce come sostituto di MobileMe. iCloud è un sistema di archiviazione di documenti e contenuti multimediali in una posizione centrale (server remoti) accessibili dai vari dispositivi in possesso dell'utente. l'utente ha la possibilità di visualizzare o modificare documenti da qualsiasi dispositivo senza dover sincronizzare o trasferire file. Questa tecnologia offre uno strato di sicurezza per l'utente, anche se un dispositvo dovesse andare perduto i documenti archiviati non si perdono se sono depositati in iCloud. Esistono due modi in cui le applicazione possono trarre vantaggio da iCloud:

- **iCloud document storage:** si utilizza questa funzione per memorizzare documenti e dati nell'account iCloud dell'utente.
- **iCloud key-value data storage:** si utilizza questa funzione per condividere piccole quantitá di dati tra le istanze dell'applicazione.

La maggior parte delle applicazioni utilizzano il primo modo per archiviare dati. Quindi si predilige l'archiviazione iCloud poiché l'utente si preoccupa maggiormente della condivisione dei suoi documenti tra i vari dispositivi al fine di visualizzarli e gestirli con non troppa difficoltà. Al contrario il secondo, è un metodo non visibile all'utente, che l'applicazione usa per condividere piccole quantità di dati con le altre istanze di se stessa.

Automatic Reference Counting: è una funzione del compilatore che semplifica il processo di gestione dei tempi di vita degli oggetti Objective-C. Questa funzione sostituisce il tradizionale sistema di gestione delle memoria usata nelle precedenti versioni di iOS. Non bisogna più ricordare se trattenere o rilasciare un oggetto, perché ARC valuta i requisiti di durata di ogni oggetto inserendo il metodo appropriato in fase di compilazione.

Block Objects: sono simili alle funzioni C tradizionali che possono essere incorporate nel nostro codice C e Objective-C. Usando i Block Objects è possibile costruire codice dinamicamente.In altri linguaggi vengono chiamati closures ma, anche se sono estramamente potenti, ancora non sono molto usati dai programmatori.

Data Protection: consente all'applicazione di proteggere dati sensibili degli utenti sfruttando la crittografia integrata sui dispositivi. Se l'applicazione indica un file come protetto, il sistema memorizza il file sul disco in formato codificato. Finché il dispositivo rimarrà boccato sarà impossibile accedere al contenuto del file, una volta che il dispositivo sarà sbloccato dall'utente verrá creata una chiava di decodifica per consentire all'applicazione di accedere. In iOS 5 sono stati aggiunti dei livelli di protezione che consentono all'applicazione di accedere al file anche se il dispositivo è bloccato, facendo attenzione alla gestione dei dati che si desidera proteggere.

File-Sharing Support: rende i file contenuti nelle applicazioni e nella directory Documenti accessibili tramite iTunes 9.1 e versioni successive. l'utente può quindi spostare i file dentro/fuori la directory secondo le necessità tramite iTunes, però non si può condividere i dati con altre applicazioni dello stesso dispositivo.

Grand Central Dispatch: è una tecnologia BSD-level, introdotta in iOS 4.0, utilizzata per gestire l'esecuzione dei tasks dell'applicazione e i threads. Per quanto concerne la programmazione concorrente GCD for-

nisce una soluzione rivoluzionaria per il miglioramento delle performance, utilizzando a tal scopo quattro astrazioni principali:

- Block objects
- Dispatch queues
- Sincronizzazione
- Eventi

In-App Purchase: introdotto in iOS 3.0 offre la possibilità di vendere contenuti e servizi all'interno delle singole applicazioni. Questa funzionalità è implementata utilizzando lo Store kit framework (che vedremo in seguito) che fornisce le infrastrutture necessarie per eleborare le transazioni finanziare per l'acquisto all'interno dell'iTunes store, da parte dell'utente. In iOS 6.0 è stato aggiunto il supporto per l'hosting di contenuti, l'acquisto di contenuti all'interno delle applicazioni e cosa molto importante è stata risolta una vulnerabilità, scoperta in iOS 5.1, che rendeva l'applicazione maggiormente predisposta ad intrusioni da parte di attacker (utente malintenzionato).

SQLite: la libreria SQLite consente di incorporare un database SQL nelle applicazioni senza dover eseguire separatamente un server remoto all'interno di un processo server. Quindi è possibile creare e gestire database locali contenuti nei file delle applicazioni.

XML Support: il framework foudation fornisce la classe NSXML Parser che analizza in maniera event-driven i documenti XML notificando ad un delegate gli elementi (attributi, elementi, blocchi CDATA e cosí via) che incontra durante la sua elaborazione. Fornisce anche un supporto aggiuntivo con la libreria open source libXML2 che consente sia di analizzare o scrivere dati XML arbitrari in modo rapido sia di trasformare contenuti XML in HTML.

Core Services Frameworks

Accounts Framework: fornisce l'accesso agli account utente memorizzati nel database. Usando questo framework non c'è bisogno di essere responsabili della gestione e memorizzazione delle credenziali. l'utente può evitare di digitare il proprio nome utente e password concedendo all'applicazione di utilizzare i suoi account di accesso.

Address Book Framework: fornisce l'accesso alla rubrica memorizzata sul dispositivo dell'utente. un'applicazione utilizza questo framework

per recuperare i contatti con cui può effettuare,ad esempio, una chiamata via internet. In iOS 6 per accedere ai contatti si richiede all'utente l'autorizzazione esplicita.

Ad Support Framework: fornisce un identificatore che viene utilizzato dalle applicazioni per scopi pubblicitari.

CFNetwork Framework: basato su socket BSD fornisce un insieme di interfacce in linguaggio C molto performanti che utilizzano paradigmi orientati agli oggetti per lavorare con protocolli di rete. Tale framework semplifica il controllo sui vari stack di protocolli e le operazioni per comunicare con server FTP e HTTP.

Core Data Framework: è una tecnologia che gestisce il modello dei dati di un'applicazione Model-View-Controller, gestisce anche la persistenza dei dati e il grafo degli oggetti. Si basa principalmente su concetti estrapolati dal mondo dei database così da portare la gestione dei file nelle applicazioni ad un livello superiore. Si può evitare di definire strutture dati di programmazione poiché è possibile utilizzare gli strumenti grafici di Xcode per costruire uno schema che rappresenta il modello dei dati. Inoltre Core Data Framework offre meccanismi per gestire aggiornamenti significativi delle applicazioni.

Core Foundation Framework: è un framework molto importante per lo sviluppo di applicazioni, è un API C che contiene gli elementi costruttivi per le applicazioni, come ad esempio stringhe, array e dizionari. Il Core Foundation framework è strettamente correlato al Foundation Framework, che fornisce interfacce Objective-C per le stesse caratteristiche. Se si vuole mescolare oggetti Foundation e tipi Core Foundation è possibile usufruire del Toll-free brindging. Toll-free brindging significa che è possibile intercambiare tipi Core Foundation e Foundation nei metodi e anche le funzioni di entrambi i metodi.

Core Location Framework: cosente di determinare la posizione corrente del dispositivo. Il framework utilizza l'hardware a disposizione come il GPS, cellulare e Wi-Fi radio per ottenere la longitudine e latitudine correnti. È possibile usare questa tecnologia nelle prorpie applicazioni per fornire la posizione corrente dell'utente. In iOS 4.0 è stata introdotto un servizio di monitoraggio della posizione a bassa potenza che sfrutta i ripetitori cellulari per tener traccia delle variazioni di posizione dell'utente.

Core Media Framework: introdotto in iOS 4.0 fornisce vari tipi di supporto di basso livello utilizzati dal AV Foundation Framework . La maggior parte delle applicazioni non devono mai utilizzare questa infrastruttura, ma l'uso di essa è previsto solo per quei sviluppatori che hanno bisogno di un

controllo più preciso per la creazione, gestione e riproduzione di contenuti audio e video.

Core Motion Framework: fornisce un insieme di interfacce per l'accesso a tutti i dati di movimento del dispositivo. Quindi esso supporta l'accesso ai dati, grezzi e elaborati, del girometro usando una serie di interfacce basate su blocchi. Per i dispositivi provvisti di giroscopio è possibile recuperare dati che forniscono informazioni sull'atteggiamento e la velocitá di rotazione del dispositivo. Tale framework è molto utile per determinate applicazioni, come ad esempio i giochi, che utilizzano il movimento per migliorare l'esperienza complessiva dell'utente.

Core Telephony Framework: serve per ottenere informazioni sui provider di servizi cellulari di un utente. Uilizzando questo framework le applicazioni basate su eventi di chiamate cellulari (ad esempio il VoIP) possono essere notificate quando questi eventi si verificano.

Event Kit Framework: si utilizza questo framework per ottenere l'accesso agli eventi sul calendario, fornendo anche la possibilit'a di crearli. Inoltre fornisce la possibilità di associare allarmi ad un evento. In iOS 6 è stato aggiunto il supporto per la creazione e l'accesso al promemoria sul dispositivo dell'utente. In particolare per poter accedere al calendario e al promemoria è necessaria l'autorizzazione dell'utente, quindi le applicazioni devono essere pronte anche per un eventuale negazione da parte dell'utente.

Foundation Framework: fornisce un insieme di classi di utilità di base e inoltre introduce diversi paradigmi che definiscono funzionalità non appartenenti al linguaggio Objective-C. Il Foundation Framework ha diversi obbiettivi:

- Fornisce un insieme di classi di utilità di base.
- Rendere più facile lo sviluppo del software
- Supporto alle stringhe Unicode, alla persistenza degli oggetti e alla distrubuzione degli oggetti
- Fornire un livello di indipendenza del SO, per migliorare la portabilità

Tale framework fornisce un supporto alle seguenti funzionalità:

- Tipi di dati di base (stringhe, array e così via)
- Bundle
- Data e gestione del tempo

- Raw e gestione dei blocchi dati
- Gestione delle preferenze
- URL e manipolazione del flusso
- File e loop di esecuzione
- Bonjour
- Gestione delle porte
- Internazionalizzazione
- Sostegno cache

Mobile Core Services Framework: introdotto in iOS 3.0 definisce i tipi di basso livello utilizzati in Uniform Type Identifiers (UTIs).

Newsstand Kit Framework: si utilizza questo framework per sviluppare il lato client di una applicazione Newsstand (Edicola).

Pass Kit Framework: introdotto in iOS 6 implementa il supporto per i passes scaricabili utilizzando servizi web e nuovi formati di file. Le aziende possono creare pass che rappresentano elementi quali coupon, carte d' imbarco e così via. I pass usano un formato di file speciale crittograficamente firmato prima di essere consegnato nei dispositivi mobile.

Quick Look Framework: fornisce un'interfaccia per visualizzare i file che non sono supportati. Questo framework è usato principalmente dalle applicazioni che scaricano file dalla rete o che lavorano con file provenienti da fonti sconosciute. Una volta scaricato il file è possibile utilizzare controller di visualizzazione, fornito da tale framwork, per visualizza il contenuto direttamente nell'interfaccia utente.

Social Framework: fornisce una semplice interfaccia per l'accesso agli account social media dell'utente. Esso soppianta Twitter Framework che era stato introdotto in iOS 5. Le applicazioni utilizzano questo framework per inviare aggiornamenti di stato e immagini sull'account dell'utente. Questo framework lavora con Accounts framework per garantire che l'accesso all'account dell'utente sia stato approvato.

Store Kit Framework: fornisce il supporto per l'acquisto di contenuti e servizi dall'interno delle applicazioni iOS. Ad esempio si utilizza questa funzionalità per consentire di aquistare contenuti exstra dell'applicazione. Questo framework si concentra sugli aspetti finanziari delle transazioni, in modo tale che avvengano in modo sicuro e corretto.

System Configuration Framework: serve per determinare che tipo di connessione, Wi-Fi o cellulare, è in uso sul dispositivo. Fornisce inoltre funzioni che determinano la raggiungibilità di un server host e servizi di rilevamento degli errori.

2.3.3 Media Layer

Lo strato di supporto che contiene i grafici, audio e tecnologie video orientata verso la creazione della migliore esperienza multimediale disponibile su un dispositivo mobile. Di particolare attenzione è una recente tecnologia sviluppata chiamata AirPlay che permette lo streaming audio verso la Apple Tv oppure verso altoparlanti AirPlay di terze parti. Il supporto AirPlay è integratoin AV Foundation e Core Audio Framework, ciò implica che qualunque contenuto audio riportato usando uno di questi due framework viene in automatico reso idoneo per quel che riguarda la distribuzione tramite AirPlay.

Graphics Technologies: creare un'applicazione in maniera efficace significa utilizzare immagini prerenderizzate insieme a delle views standard e controlli fatti dal framework UIKit lasciando che il sistema faccia il disegno. Tuttavia ci possono essere delle situazioni, che ovviamente dipendono dal tipo di applicazione che si vuole progettare, in cui bisogna andare oltre alla grafica semplice; utilizzando le seguenti tecnologie per gestire i contenuti grafici dell'app:

- **Core Graphics** (Quartz) gestisce il 2D vector native e il rendering
- **Core Animation** fornisce un supporto avanzato per l'animazione di views e altri contenuti
- **Core Image** offre il supporto avanzato per la manipolazione di audio e video
- **OpenGL ES and GLKit** fornisce supporto per il rendering 2D e 3D utilizzando interfacce con accelerazione hardware
- **Core Text provides** fornisce un layout di testo sofisticato e un motore per il rendering
- **Image I/O** contiene intefacce per la lettura e scrittura della maggior parte dei formati di immagine

- **The Assets Library framework** consente di accedere alla libreria foto e video dell'utente

Le tecnologie sopra elencate verranno approfondite più avanti. Grazie alla Graphics Technologie si elimina il problema di adattare manualmente i contenuti grafici di un'applicazione sui vari dispositivi Apple che abbiano un display retina o no, visto ceh qualsiasi contenuto grafico viene scalato automaticamente.
Audio Technologies: include diversi metodi di riproduzione e registrazione dei contenti audio. I framework che fanno parte di questra tecnologia, ordinati dal livello alto a livello basso, sono:

- **Media Player framework**
- **AV Foundation framework**
- **OpenAL**
- **Core Audio frameworks**

I framework di alto livello sono i più usati dagli sviluppatori perch' e sono facili da usare. Essi verranno approfonditi più avanti.

Video Technologies tale tecnologia consente la riproduzione di contenuti video, la possibilità di registrare video attraverso app.

- **UIImagePickerController Class**
- **Media Player framework**
- **AV Foundation framework**
- **Core Media**

Media Layer Frameworks

Assets Library Framework fornisce un 'interfaccia basata sulle query per recuperare foto e video dal dispositivo. Con questo framework è possibile accedere alle stesse funzionalità che normalmente vengono gestite mediante l'applicazione foto.

AV Foundation framework: fornisce una serie di classi Objective-C per la riproduzione di contenuti audio, con la possibilità di avere il controllo sui vari aspetti di riproduzione di qualsiasi contenuto con durata diversa.

In iOS 3 è stato aggiunto il supporto per la registrazione e la gestione delle informazioni dei contenuti audio. In iOS 4 sono stati aggiunti dei servizi:

- Media asset management
- Media editing
- Movie capture
- Movie capture
- Track management
- Metadata management for media items
- Metadata management for media items
- Sincronizzazione dei suoni
- un'interfaccia Objective-C che fornisca gli attributi di un file audio (formato dei dati, frequenza di campionamento e così via)

l'ultimo aggiornamento, in iOs 5, include il supporto per lo streaming audio e video attraverso AirPlay.

Core Audio frameworks il supporto nativo per l'audio è fornito da questa famiglia di framework, che offre la possibilità alle applicazioni di riprodurre, registrare e mixare contenuti audio. Inoltre grazie a questi framework è possibile attivare la vibrazione sui dispositivi che lo supportano.

Core Graphics Framework: è una API che si basa su C. Questo framework viene utilizzato in particolar modo per la gestione del disegno basato su trasformazioni, gestione del colore, gradienti e sfumature, creazioni immagini, creazione di documenti in pdf e così via.

Core Image Framework: fornisce un potente set di filtri che permettono all'utente di modificare video e immagini. Tale framework si può utilizzare per effetuare operazioni semplici come ritoccare e correggere le foto o per operazioni avanzate come il rilevamento del volto. Questo framework sfrutta la CPU disponibile e la potenza di elaborazione della GPU per assicurare che le operazioni siano veloci ed efficienti.

Core MIDI Framework: fornisce un metodo standard per comunicare con dispositivi MIDI tra cui tastiere e sintetizzatori.

Core Text Framework: contiene un insieme di interfacce basate su C che permettono la disposizione del testo e la gestione del font. Esso fornisce

un motore completo di layot di testo che è possibile utilizzare per gestire il posizionamento del testo sullo schermo.

Core Video Framework: fornisce il supporto per il Core Media framework.

Image I/O Framework: fornisce interfacce per importare ed esportare i dati e metadati delle immagini. Questo framework supporta tutti i tipi di immagini standard disponibili in iOS.

GLKit Framework: contiene una serie di classi objective-C utili per semplificare la creazione di un'applicazione OpenGL ES 2.0. Fornisce il supporto per le quattro aree chiave di sviluppo delle applicazioni.

Media Player framework fornisce l'accesso alla libreria iTunes da parte dell'utente, offrendo la possibilità di riprodurre brani musicali, effettuare una ricerca di canzoni e così via. In iOS 5 è stato aggiunto il supporto per la visulizzazioni delle informazioni, del brano attualmente in esecuzione, nel lock screen e nel multitaskin controls.

OpenAL (Open Audio Library) è uno standard cross-platform per un rendering efficiente di audio posizionale a tre dimensioni.

OpenGL ES Framework: fornisce gli strumenti per il disegno 2D e 3D. È un framework basato su C che lavora a stretto contatto con l'hardware del dispositivo per fornire un frame rate elevato per le applicazioni di giochi in full-screen

Quartz Core Framework: contiene le interfacce Core Animation, che è una tecnologia di animazione e compositing avanzato che utilizza un percorso di rendering ottiizzato per realizzare animazioni complesse e effetti visivi.

2.3.4 Cocoa Touch Layer

Cocoa Touch è un insieme di strutture orientate agli oggetti che forniscono un ambiente di runtime per le applicazioni in esecuzione su iOS. Questo livello definisce infrastrutture applicative di base per creare elementi di interfaccia utente touch-based e accedere all'hardware-specific come l'accelerometro, giroscopio, fotocamera e il magnetometro. Inoltre fornisce supporto per tecnologie chiave come il multitasking, notifiche push, e molti servizi di alto livello.

High-Level Features

Auto Layout: questa tecnologia migliora il modello "springs and struts", quindi definisce le regole riguardanti la disposizione degli elementi dell'interfeccia utente rendendola più intuitiva rispetto al modello precedente.

Storyboards: soppianta i file .nib, tale tecnologia permette di progettare un'intera interfaccia grafica in un unico posto, facilitandone la creazione. Lo Storyboards permette di definire i segues, che sono delle transizioni da un view controller ad un altro, definendole visivamente in Xcode.

Document Support: la classe UIDocument appartenente al framework UIKit fornisce un contenitore per tutti i dati associati ai documenti dell'utente. Fornisce inoltre un supporto integrato per la lettura
scrittura asincrona dei dati di file, un sicuro salvataggio dei dati, salvataggio automatico dei dati e il supporto per file flat.

Multitasking: le applicazione create utilizzando iOS SDK 4.0 o successive non vengono terminate quando l'utente preme il tasto Home, ma vengono messi in background. Un applicazione in background rimane in memoria senza eseguire alcun codice preservando il consumo di batteria, inoltre questo comportamento consente all'applicazione di riprendere rapidamente quando viene rilanciata. Alcune applicazioni possono essere autorizzate a continuare l'esecuzione in background per i seguenti motivi:

- un'applicazione può richiedere un tempo finito per completare un compito importante
- un'applicazione può dichiarare di supportare servizi specifici che richiedono un regolare tempo di esecuzione in background.
- un'applicazione può utilizzare le notifiche locali per generare avvisi utente.

Il supporto multitasking definito da UIKit aiuta il passaggo dell'applicazione da e per lo stato di background senza problemi.

Printing: permette alle applicazioni di inviare contenuti in modalità wireless su stampanti vicine. Tale tecnologia utilizza una coda gestita con algoritmo FCFS. La stampa wireless è disponibile su dispositivi che supportano il multitasking. UIKit si occuperà di tutto il lavoro pesante gestendo l'interfaccia di stampa, la pianificazione e l'esecuzione di lavori sulla stampante.

UI State Preservation: tecnologia che serve per salvare lo stato dell'interfaccia utente quando l'applicazione viene mandata in background,

quando l'applicazione verrà rilanciata utilizza lo stato salvato per ripristinare l'interfaccia facendo sembrare che l'applicazione non ha mai smesso di funzionare. Tale supporto è integrato in UIKit che fornisce l'infrastruttura per salvare e ripristinare l'interfaccia dell'app.

Apple Push Notification Service: offre un modo per notificare all'utente nuove informazioni, anche nel caso in cui l'applicazione non sia in esecuzione.

Local Notifications: esso insieme a Apple Push Notification Service completano il meccanismo di notifiche push così che le applicazioni possano generare le notifiche a livello locale senza la necessità di appoggiarsi a servere esterni. Le notifiche locali sono indipendenti dalle applicazioni perché se una notifica è prevista il sistema consegnerà la notifica senza bisogno che l'applicazione vada in esecuzione.

Gesture Recognizers: sono oggetti che si collegano alla views e servono per riconoscere i pi' u comuni tipi di gesti come swipes e pinches. Quindi si ha la possibilità in un punto della views di associare un gesto ad una azione.

Peer-to-Peer Services: tecnologia formita dal framework Game Kit che consente di utilizzare la connettività peer-to-peer per avviare sessioni di comunicazioni con dispositivi vicini via bluetooth.

Standard System View Controllers: molti dei framework nello strato Cocoa Touch contengono view controller per presentare interfacce di sistema standard. Ogni qualvolta si vede eseguire una determinata operazione vi è la necessità di utilizzare una view controller ralativo al framework corrispondente. Per meglio intenderci, consideriamo delle classiche operazioni effettuate dell'utente:

- Visualizzare o modificare le informazioni di contatto: utilizzare i view controller nel Address Book UI framework
- Creare o modificare il calendario, utilizzare il view controller nel Event Kit UI framework
- Comporre una e-mail o un messaggio SMS: utilizzare i view controller nel Message UI framework

External Display Support: tecnologia che permette di collegare un display esterno su alcuni dispositivi iOS.

Cocoa Touch Frameworks

Address Book UI Framework: questo framework semplifica il lavoro che serve per visualizzare le informazioni dei contatti nelle applicazioni e garantisce che le applicazioni utilizzino la stessa interfaccia così da avere una coerenza con la piattaforma.

Event Kit UI Framework: fornisce le classi necessarie per creare, modificare e visualizzare gli eventi utilizzando un view controller.

Game Kit Framework: fornisce varie funzionalità di rete attraverso un insieme di classi costruite sul top di Bonjour. Tale framework consente di usare la connettività peer-to-peer non solo su giochi ma anche su altri tipi di app. In iOS 4.1 è stato introdotto Game Center che consente ai dispositivi di connettersi a tale servizio per poter giocare on-line e scambiarsi informazioni quali record, posizioni in classica e altro.

iAd Framework: consente di pubblicare annunci sulle app, essi vengono inseriti nella view in formato standard in basso o a schermo intero. Le view create per gli annunci interagiscono con il servizio annunci di Apple per gestire la presentazione degli annunci e la risposta al touch.

Map Kit Framework: fornisce un'interfaccia per l'incorporamento delle mappe nelle view delle app, inoltre questo framework fornisce il supporto per le annotazioni sulla mappa, aggiungendo sovrapposizioni e fornisce la possibilità di fare ricerche reverse-geocoding per ricavare informazioni dal segnaposto posizionata su una determinata coordinata sulla mappa.

Message UI Framework: mette a disposizione delle view controller specializzate per presentare interfacce standard per comporre email e SMS.

Twitter Framework: supporta un'interfaccia utente per la generazione di tweet e il supporto per la creazione di URL per accedere al servizio di Twitter.

UIKit Framework: fornisce le classi necessarie per costruire e gestire interfacce utente di un'applicazione per iOS. Definisce le struttura per il comportamento delle app, inclusa la gestione degli eventi. Gli oggetti UIKit si possono aggiungere all'interfaccia in tre modi:

- Utilizzando lo strumento Interface Builder
- Creando, posizionando e configurando gli oggetti scrivendo righe di codice
- Implementando gli oggetti da UIView creando sottoclassi che ereditano da UIView

2.4 Organizzazione di un'applicazione

Per sviluppare una buona applicazione è necessario avere per prima cosa un'idea chiara di ciò che si vuole creare e poi tradurla in una serie di azioni che richiedono una forte pianificazione. Quindi è importante scegliere un processo di sviluppo al fine di garantire un ottimo risultato finale. Per ciò che concerne lo sviluppo di applicazioni iOS ci sono alcuni design pattern e tecniche fondamentali che è necessario conoscere prima di poter scrivere il codice, indipendentemente dal tipo di applicazione che si vuole sviluppare. l'infrastruttura di base di un'applicazione iOS è costruita da oggetti appartenenti al framework UIKit, tali oggetti forniscono il supporto per la gestione degli eventi, la visualizzazione di contenuti sullo schermo e l'interazione con il resto del sistema. È importante sapere che un dispositivo iOS esegue più applicazioni contemporaneamente, ma solo un'applicazione sarà in foreground, quindi che ha il permesso di presentare un interfaccia utente e rispondere agli eventi di touch, le altre applicazioni invece rimangono in background. Una gestione della memoria accurata ed efficiente è importante per un applicazione iOS perchè tali dispositivi non hanno a disposizione una memoria come quella di un computer desktop. Una caratteristica importante per un applicazione mobile è l'interazione con alcune risorse quali l'accelerometro, il girometro, fotocamera, gps , e tante altre. Quindi è importante coordinare tali interazioni con le altre parti del sistema. Infine bisogna conoscere bene gli strumenti con cui sviluppare in modo agevole le applicazioni e questi sono: Xcode inteface buildere e il simulatore.

2.5 Concetti fondamentali

2.5.1 I fondamentali Design Patter

Come già anticipato precedentemente ci sono alcuni modelli di progettazione, quindi Design pattern, che è necessario apprendere. In iOS i framework del sistema forniscono infrastrutture importanti per lo sviluppo di un'applicazione ed è l'unico modo per accedere all'hardware sottostante. Tali framework sono basati su vari Design Patter quindi la loro comprensione è un importante primo passo per sviluppare un'applicazione in modo agevole. I Design Pattern più importanti sono:

- **Model-View-Contreoller:** macro patter che riguarda la struttura dell'intera app, permette di dividere un insieme di oggetti che collaborano tra di loro in gruppi distinti in base al ruolo che svolgono.

- **Delegation:** facilita la comunicazione fra gli oggetti, evita il legame tra un elemento generico, riutilizzabile, dell'interfaccia utente e il comportamento specifico.
- **Target-action:** evita il legame tra un elemento generico, riutilizzabile, dell'interfaccia utente e il comportamento specifico.
- **Notification:** Consiste nella comunicazione senza forte accoppiamento fra gli oggetti.

Il Model View Control è fondamentale per lo sviluppo di una buona app, consente la separazione degli oggetti in tre ruoli: Model, View e Control e definisce anche in che modo tali oggeti comunicano.

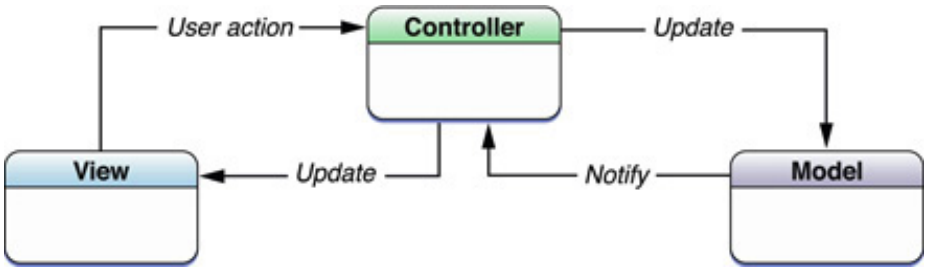

Figura 2.3: Schema Model-View-Control [6].

2.5.2 l'imporatanza della gestione della memoria

La gestione della memoria per un'applicazione è importantissima poiché essendo in escuzione su dispositivi mobile, quindi con memoria limitata rispetto a un PC, è necessario un corretto utilizzo. In qualsiasi app, o programma in generale, è importante garantire che gli oggetti non necessari vengano deallocati, garantendo al contempo che oggetti in uso non vengano deallocati e non siano lazy nella creazione di oggetti. Objective-C offre due ambienti per la gestione della memoria che sono:

- **Reference counting:** (conteggio dei riferimenti), permette di determinare la durata degli oggetti

- **Garbage Collection:** si passa la responsabilita di determinare la durata degli oggetti ad un sistema automatico (non disponibile su iOS)

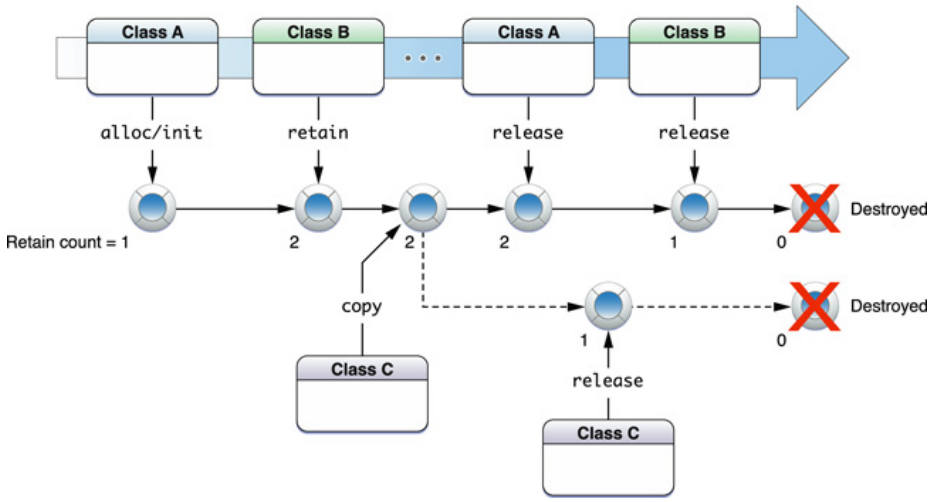

Figura 2.4: Reference Counting [6].

2.6 Il cuore dell'App

un'applicazione in iOS è costituita da un insieme di oggetti UIKit che gestiscono il ciclo degli eventi (Event loop) e le interazioni con iOS. Attraverso una combinazione di sottoclassi, pattern dalegate e altre tecniche, è possibile modificare i comportamenti di dafault definiti da UIKit per implementare al meglio un'app. Oltre alla modifica degli oggetti UIKit bisogna definire una serie di oggetti che definisce il più grande insieme di oggetti in un app. Tale insieme è costituito da i data object, a differenza degli oggetti UIKit, la definizione è interamente a carico del programmatore. In aggiunta bisogna fornire una serie di oggetti di interfaccia utente che però sono, grazie alle classi UIKit, di facile definizione. Il cuore dell'applicazione è l'oggetto UIApplication che fornisce un punto centralizzato di controllo e di coordinamento per le applicazioni in esecuzione su iOS, quindi ha il compito di ricevere gli eventi dal sistema inviandoli alla parte di codice scritta per la gestione.

2.6.1 Main Event loop

l'event loop viene creato, non appena l'applicazione viene mandata in esecuzione dalla funzione UIApplicationMain iniziando subito l'elaborazione degli eventi, il risultato di tale elaborazione è l'aggiornamento della view e dello stato. Stabilisce una connessione con le componenti di sistema che sono responsabili della consegna degli eventi. l'applicazione riceve gli eventi attravero una sorgente situata nel main thread's run loop. Gli eventi vengono gestiti singolarmente quindi sono inseriti in un coda di tipo FIFO. Una volta che l'interfaccia utente è stata caricata, l'applicazione verrà guidata dagli eventi esterni. l'oggetto application (UIApplication) recupera il primo oggetto nella coda e lo converte in un oggetto evento (UIEvent) e affida la sua elaborazione a altri componenti specifici. Una volta conclusa l'elaborazione di un evento, UIApplication preleva un altro oggetto dalla coda. Tale ciclio (loop) continuerà così fino alla terminazione dell'app. Quando l'applicazione viene avviata vengono definiti anche un set di oggetti responsabili di disegnare l'interfaccia utente e gestire gli eventi. Tali oggetti includono finestre e varie view. Quando l'oggetto application preleva un oggetto dalla coda degli eventi, lo invia alla finestra dove si è verificato l'evento. A sua volta la finestra invia l'evento alla view che è il gestore più approrpriato per questo evento. La view spesso inizia subito una serie di azioni che modificano l'aspetto dell'app, lo stato e i duoi stessi dati. Una volta completate tali operazioni il controllo torna a UIApplication che si accinge a recuperare il prossimo evento dalla coda. Se la prima view a cui viene consegnato l'evento non gestisce tale evento, si può passare l'evento ad altre view tramite responder chain.

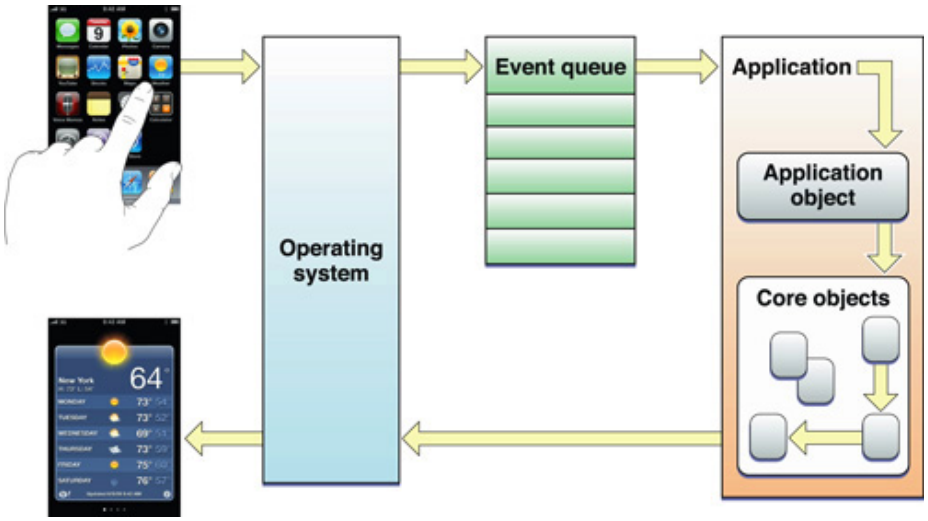

Figura 2.5: Event loop [6].

2.6.2 Model View Control

Per capire come gli oggetti UIKit lavorano è importante conoscere gli oggetti che compongono un'applicazione in iOS. La figura mostra gli oggetti comunemente presenti in un'app. Come si vede dal grafico le applicazioni iOS sono organizzate intorno al design pattern Model-View-Controller. Model-View-Controller assegna agli oggetti di un'applicazione un ruolo: model, vista o controller. Oltre a definire i ruoli definisce in che modo tali oggetti comunicano tra loro. Ogni tipo di oggetto è separato dagli altri da confini astratti e comunicano attraverso tal confini. Come si vede in figura il controller object comunica con il model object e con view object, la view e il model non devono mai comunicare fra loro.

- **Model Objects:** incapsulano i dati specifici di un'applicazione e definiscono la logica con cui manipolano ed elaborano i dati. Gran parte dei dati che fanno parte dello stato persistente dell'applicazione (se tale stato persistente viene memorizzato in file o database) devono risiedere nel model dopo che i dati vengono caricati nell'app. Quando un oggetto del model cambia (ad esempio vengono ricevuti dati tramite una connessione di rete) l'oggetto controller viene notificato, esso a sua volta aggiornerà gli oggetti vista

- **View Objects:** è in grado di rispondere alle azioni dell'utente, ha la scopo di visualizzare i dati di del Model Object consentendone la modifica. View object apprende subito i cambiamenti dei dati del model attraverso il controller object comunicando le modifiche introdotte dall'utente.
- **Controller Objects:** funge da intermediario tra view objects e model objects. Possono anche eseguire la configurazione e funzioni di coordinamento, inoltre gestisce i cicli di vita di altri oggetti.

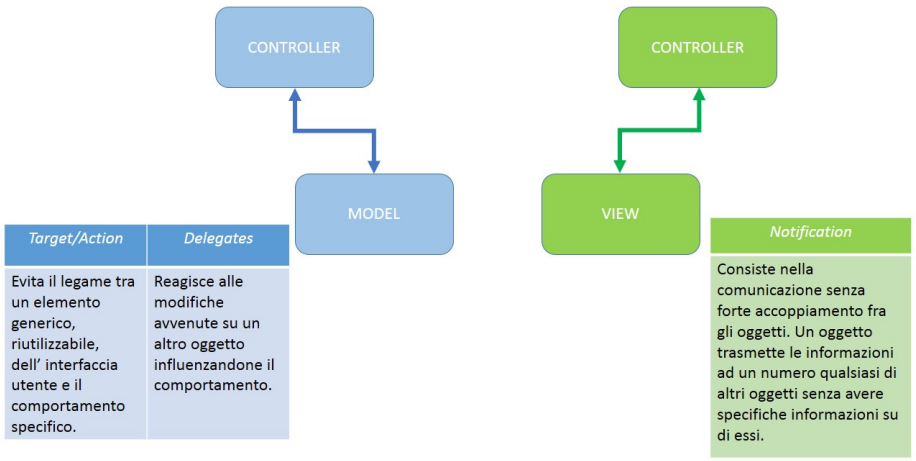

Figura 2.6: Comunicazione fra gli object e i relativi design patter usati [6].

I benefici che porta tale modello sono:

- riusabilità degli oggetti
- interfacce tendono ad essere ben definite
- applicazioni che adottano un modello MVC sono facilmente estensibili.

Molte tecnologie e architettura di Cocoa sono basate su Model-View-Controller, quindi iOS presuppone che tutte le applicazioni siano costruite utilizzando tale modello di progettazione.
Il ruolo degli oggetti, raffigurati, in un'applicazione iOS:

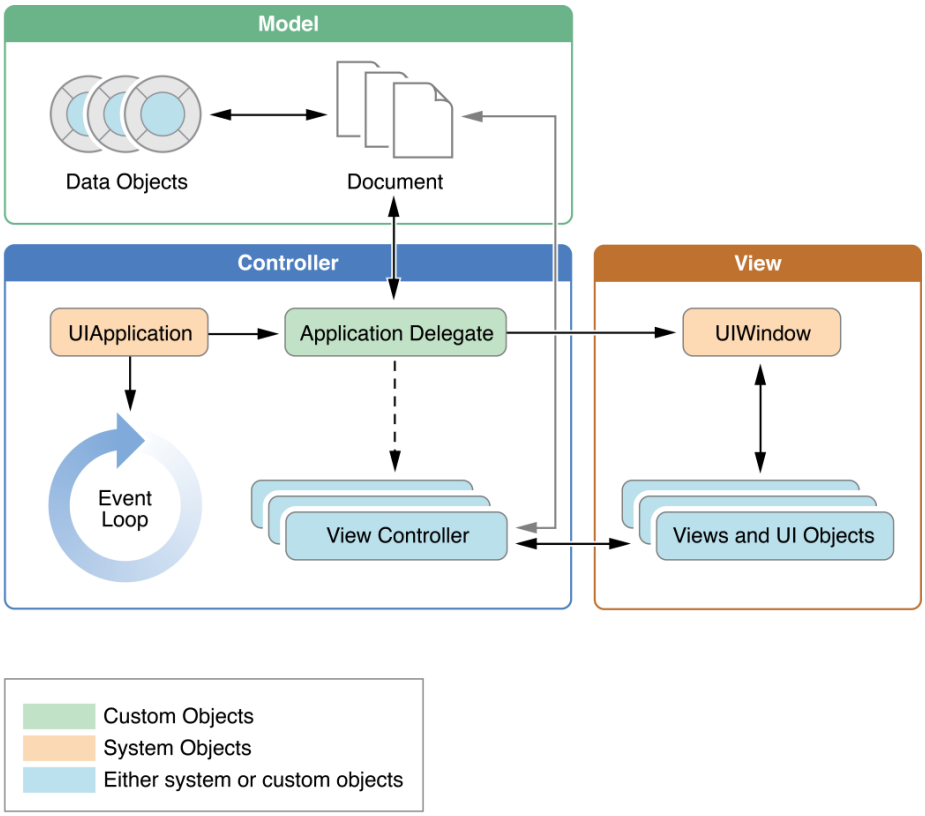

Figura 2.7: Oggetti chiave nel Model-View-Controller [6].

- **UIApplication object:** questo oggetto controller gestisce il ciclo di eventi (event loop) e coordina altri comportamenti di alto livello. Si utilizza questo oggetto senza la creazione di sottoclassi. Esso lavora in tandem con l'oggetto delegate dove risiede la logica dell'app.
- **Event Loop:** è semplicemente un ciclo di esecuzione: coordina la ricezione di eventi provenienti da diverse sorgenti collegati all 'Event loop.
- **Documents and data model objects:** memorizza i contenuti dell'app. Ad esempio una' applicazione bancaria potrebbe memorizzare un database che contiene le transazioni finanziarie, mentre un'applicazione painting potrebbe memorizzare un oggetto immagine o anche la sequenza di comandi di disegno che hanno portato alla creazione di

quell'immagine. In quest'ultimo caso, un oggetto immagine è ancora un oggetto dati perché è un contenitore per i dati di immagine. Le applicazioni possono utilizzare oggetti documenti (UIDocument) per gestire alcuni o tutti gli oggetti del modei object. Gli oggetti non sono necessari ma offrono un modo conveniente per raggruppare i dati appartenenti ad un singolo o pacchetti di file.

- **Delegate object:** è un oggetto personalizzato creato al momento del lancio dell'app, in genere dalla funzione UIApplicationMain. Il compito di questo oggetto è di gestire le transizioni di stato all'interno dell'app, ad esempio esso è responsabile del lancio e manipolazione delle transizioni da e verso il background.

- **View Controller object:** si occupa della presentazione dei contenuti dell'applicazione sullo schermo. Esso gestisce una singola view e la sua collezione di subviews. Quando viene presentato, il view controller rendel visibile le view installandole nella finestra dell'app. La classe UIViewController fornisce funzionalità di default per il caricamento delle view, per la rotazione delle view in risposta a rotazioni dei dispositivi e molti altri comportamenti standard.

- **UIWindow object:** coordina la presentazione di una o più view su uno schermo. Per effettuare modifiche sul cotenuto di un app, si utilizza un view controller per modificare le view che compaiono nella finestra corrispondente; non si sostituisce mai la finestra stessa.

- **View, control, and layer objects :** forniscono la rappresentazione visiva del contenuto dell'app. Una view è un oggetto che disegna un contenuto in un'area rettangolare designata e risponde agli eventi all'interno di quell'area. I controller sono un tipo specializzato di view responsabile dell'implementazione di oggetti familiari tipo bottoni, campi di testo e toggle switches (interruttori a levatta). Layer objects sono in realtà dei data object che rappresentano un contenuto visivo. Le view utilizzano i layer objects intensamente dietro le quinte per rappresentare il loro contenuto. Inoltre è anche possibile aggiungere layer objects per implementare animazioni complesse.

2.6.3 Model Data

Il modello dati di un'applicazione comprende le strutture dati e la business logic necessarie per mantenere i dati in uno stato consistente. I dati vanno

mantenuti lontano dall'interfaccia utente poiché rende facile implementare un'applicazione universale in grado di funzionare sia su iPad che su iPhone, rendendo facile riutilizzare alcune parti di codice. Si possono usare alcune tecnologie utili per definire tipi di modello di dati.

- **Custom Data Model:** un oggetto personalizzato combina alcuni semplici dati (stringhe, numeri, date, URL e così via) con la business logic necessari per gestire i dati e garantire la consistenza dei dati stessi. I Custom Data sono in grado di memorizzare una combinazione di valori scalari e puntatori ad altri oggetti. Ad esempio il framework Foundation definisce le classi per molti tipi di dati semplici e memorizza collezioni di altri oggetti. Naturalmente, quando si definiscono oggetti personalizzati è possibile incorporare valori scalari direttamente nelle implementazioni di classe. Infatti un data object può includere una miscela di scalari e tipi di oggetto per le sue variabili. Nel listato è definita una classe di esempio per la raccolta di immagini.La classe contiene una serie di immagini e un elenco di indici di un array che rappresenta gli elementi selezionati, contiene inoltre una stringa per il titolo e una variabile booleana per indicare se la raccolta di immagini è modificabile.

```
@interface PictureCollection : NSObject {
   NSMutableOrderedSet* pictures;
   NSMutableIndexSet* selection;

   NSString* title;
   BOOL editable;
}

@property (nonatomic, strong) NSString * title;
@property (nonatomic, readonly) NSOrderedSet* pictures;

// Method definitions...

@end
```

- **Structured Data Model Using Core Data:** Core data è un framework che gestisce la persistenza dei dati e il grafo degli oggetti. Aiuta a salvare il Model object (Model-View-Controller) in un file con la possibilità di recuperarli di nuovo. Fornisce un'infrastruttura per la gestione di tutte le modifiche apportate al Model, dando la possibilità di annullare, ripetere e mantenere le relazioni reciproche tra gli oggetti. Esso consente di mantenere solo un sottoinsieme di

oggetti in un dato momento del model. Si ha la possibilità di definire le principali caratteristiche delle classi del modello comprese le relazioni tra loro in un editor basato su GUI. Tale definizione fornisce una ricchezza di funzionalità di base tra cui l'impostazione dei valori di default. Consente di mantenere insiemi disgiunti di modifiche agli oggetti, questo è utile ad esempio quando l'utente apporta modifiche in una view scartandola senza inflenzare i dati visualizzati in un'altra view. Ha una infrastruttura dati per il data storage versioning (archivio dati) e la migrazione, consentendo di aggiornare facilemente una versione old di file dell'utente con una versione attuale. Inoltre consente di memorizzare i dati in iCloud.

- **Document-Based Data Model:** è un modo conveniente per gestire i file di un'applicazione scritti sul disco. In questo tipo di modello si utilizza un oggetto document per rappresentare il contenuto di un singolo file o pacchetto di file sul disco. l'oggetto document è responsabile per la lettura/scrittura dei contenuti del file e lavora con il view controller dell'applicazione per rappresentare i contenuti del documento sullo schermo. Nella figura seguente si vedono i rapporti tipici tra documenti, file e oggetti del data model dell'app. Ogni documento è autonomo e non interagisce direttamente con altri documenti. Il documento interagisce solo con un singolo file e crea la rappresentazione di tutti i dati presenti nel file. Essendo il contenuto di un file unico lo sarà anche la struttura dati associata.

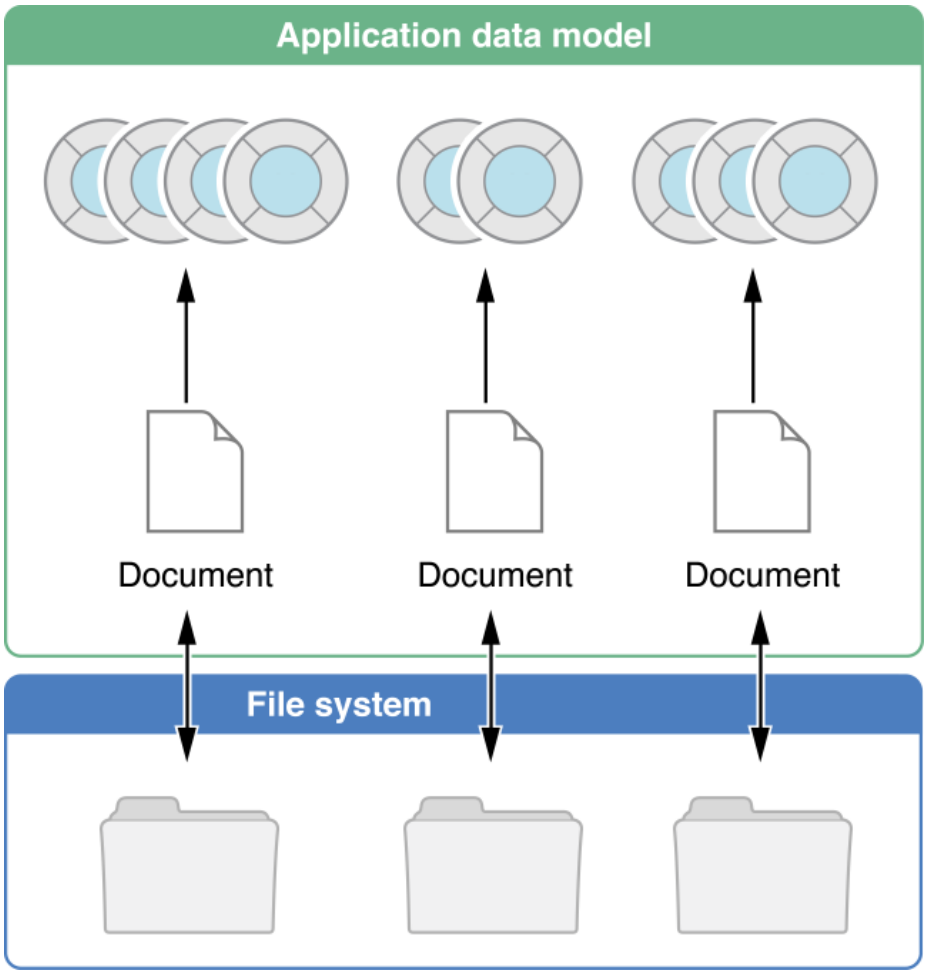

Figura 2.8: Utilizzo di documenti per la gestione del contenuto del file [6].

2.7 Multitasking

è fondamentale dire che solo un'applicazione è in esecuzione in foreground o in background, poiché le risorse sui dispositivi iOS sono limitate. un'applicazione in background si deve comportare in modo diverso rispetto che in foreground. Il sistema operativo limita le applicazioni in background così da preservare la durata della batteria migliorando l'uso dell'applicazione in foreground. Il SO ha il compito di notificare un'applicazione quando passa da background a foreground così da dare la possibilità all'applicazione di

modificare il suo comportamento. Mentre l'applicazione è in foreground il sistema invia, nel momento in cui avvengono, eventi touch ad esso per l'elaborazione, UIKit si occupa di fornire eventi agli oggetti personalizzati. È molto importante per ciò che concerne l'implementazione di un'app:

- rispondere in modo appropriato alle transizioni di stato
- avere un comportamento adeguato quando l'applicazione passa in background
- registrarsi alle notiche che segnalano le modifiche di sistema alle esigenze dell'app
- chiedere al sistema un'autorizzazione se l'app, mentre si trova in background, ha bisogno di eseguire ancora

2.7.1 Gli stati dell'applicazione

In ogni momento dell'esecuzione di un'app, essa si può trovare in uno degli stati elencati di seguente:

- **Not running:** l'applicazione non è stata avviata o è stata in esecuzione ma è stata terminata dal sistema.
- **Inattivo:** l'applicazione è in foreground, ma non è in fase di ricevere eventi.
- **Attivo:** l'applicazione è in foreground ed è pronta a ricevere gli eventi.
- **Background:** l'applicazione ìn back ground con codice in esecuzione.
- **Suspended:** l'applicazione è in background ma non è in esecuzione.

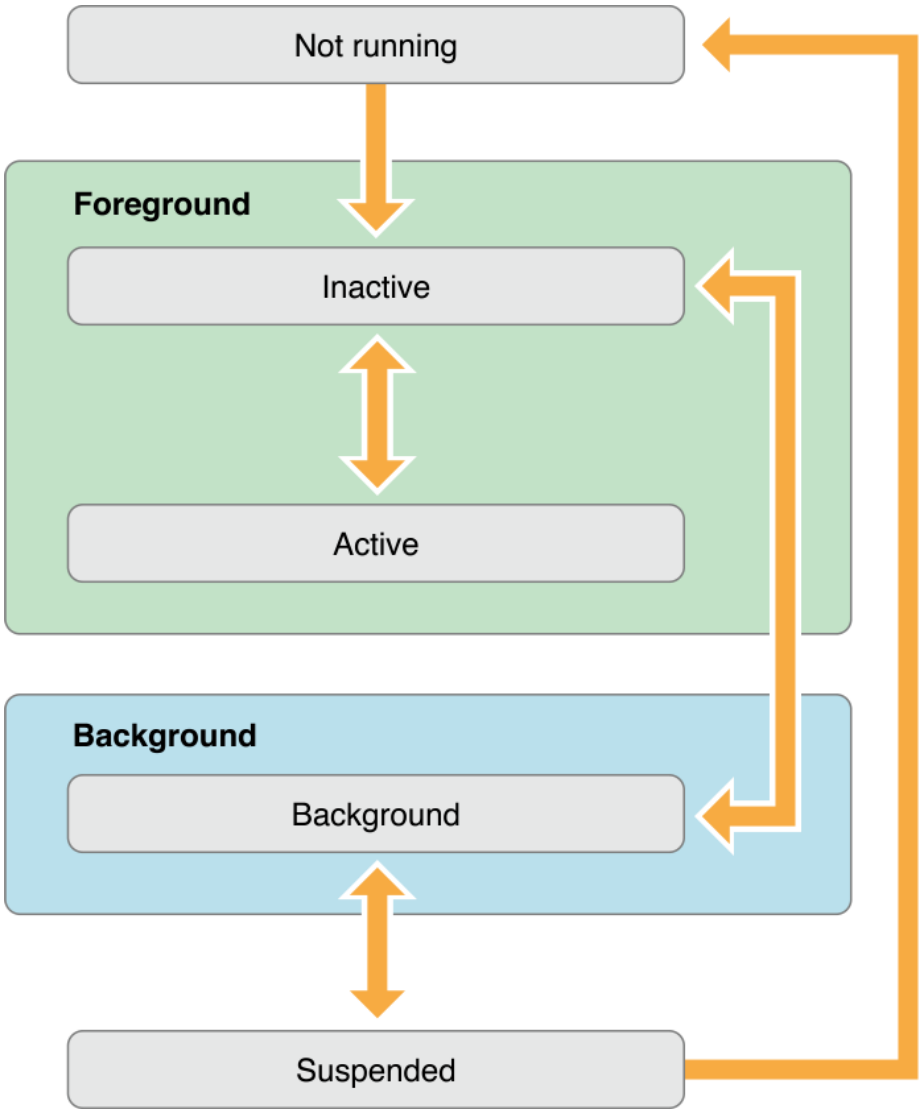

Figura 2.9: Stati in un'applicazione iOS [6].

La maggior parte delle transizioni di stato sono accompagnate da una corrispondente chiamata ai metodi del proprio oggetto delegate. Quando viene lanciata un'applicazione si muove dallo stato not running allo stato foreground o background, tale transizione attraversa brevemente lo stato inactive. Durante il ciclo di avvio il sistema crea un processo e un main

thread per l'applicazione e chiama su quest' ultimo la funzione main. Tale funzione passa il controllo a UIKit che si accupa di inizializzare l'applicazione preparandola al funzionamento. La figura seguente mostra la sequenza di eventi che si verificano quando l'applicazione viene lanciata in foreground.

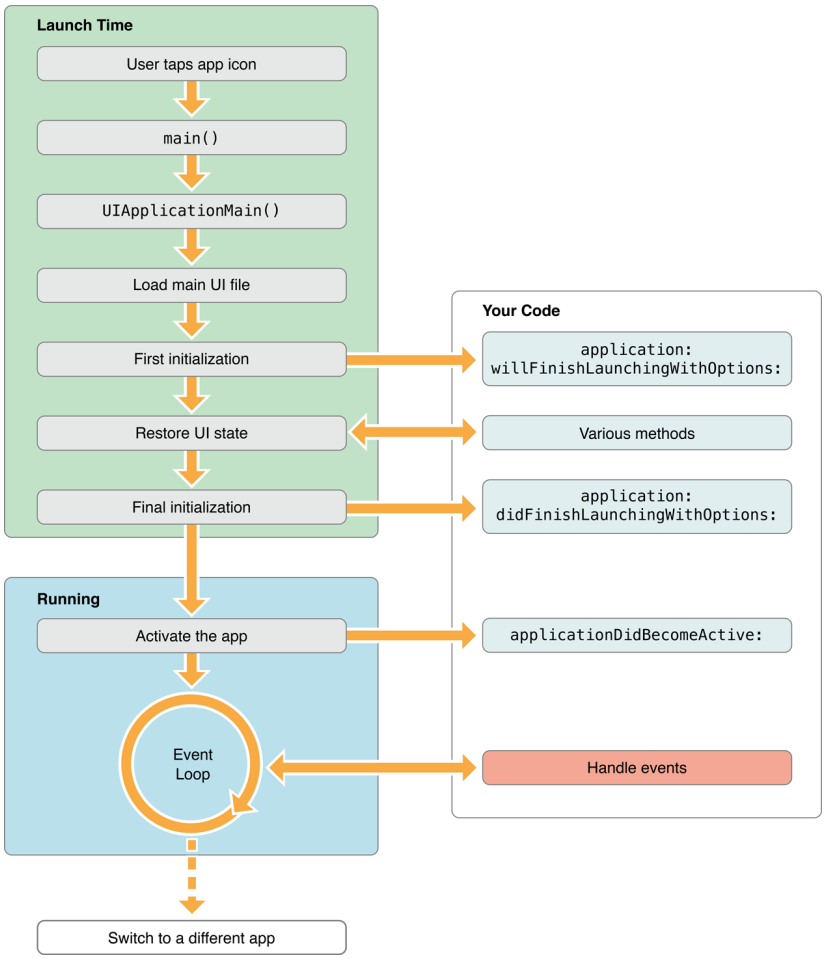

Figura 2.10: Avvio di un'applicazione in foreground [6].

Se l'applicazione viene lanciata in background il ciclo di lancio cambia leggermente, invece di essere resa attiva entra in background per gestire l'evento e poi viene sospesa. Il sistema carica lo stesso i file dell 'interfaccia utente ma non la rende visibile.

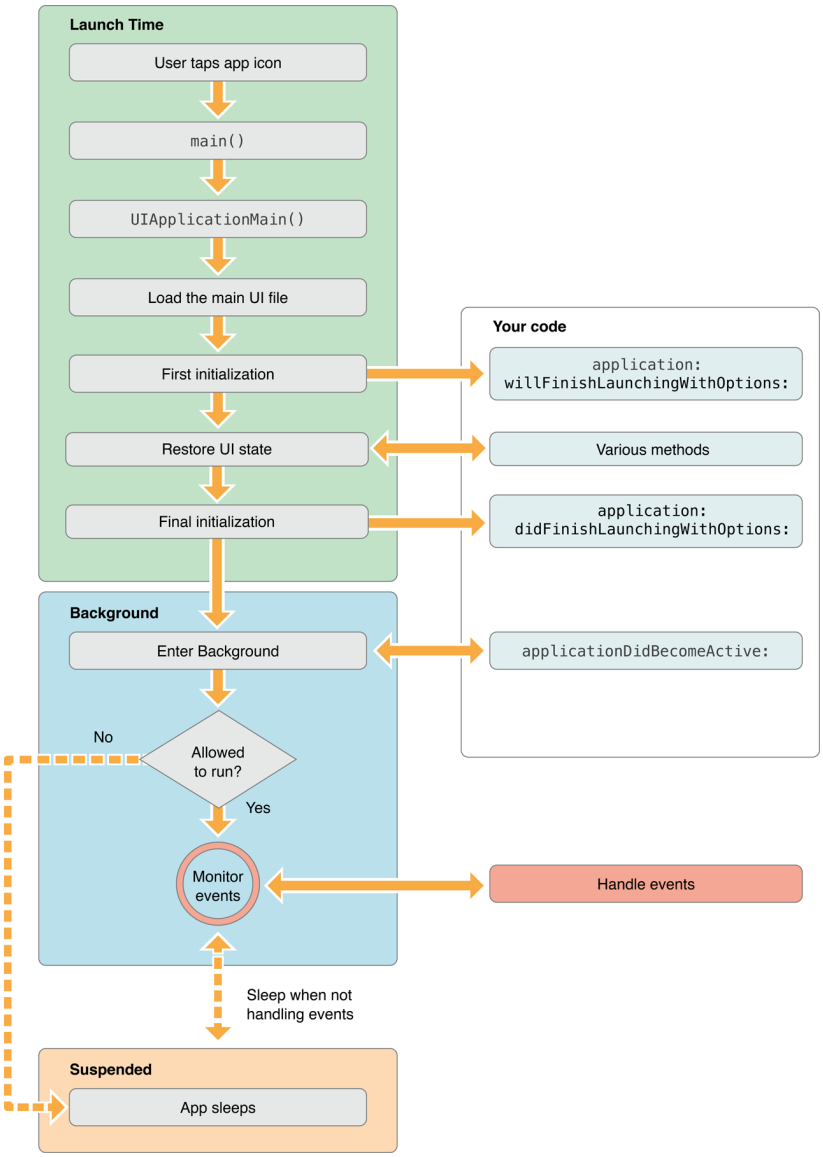

Figura 2.11: Avvio di un'applicazione in background [6].

La funzione main, come in qualsiasi applicazione basata su C, è il punto di ingresso principale per un'applicazione iOS. Tale funzione però viene usata in minima parte in quanto il suo compito è quello di consegnare il controllo al framework UIKit. Ogni nuovo progetto con Xcode è dotato di

una funzione main di default la quale implementazione non deve essere mai cambiata.

```
#import <UIKit/UIKit.h>

int main(int argc, char *argv[])
{
    @autoreleasepool {
        return UIApplicationMain(argc, argv, nil, NSStringFromClass([MyAppDelegate class]));
    }
}
```

2.7.2 Interruzioni

Quando si verifica un'interruzione allert-based, ad esempio una chiamata in arrivo, il sistema sposta l'applicazione temporaneamente in stato inactive così da permettere di chiedere all'utente come procedere. l'applicazione rimane nello stato inactive fino a quando l'utente non respinge l'alert, poi ritorna o in foreground o in background. La figura seguente mostra il flusso degli eventi di un'applicazione quando riceve un'interruzione allert-based.

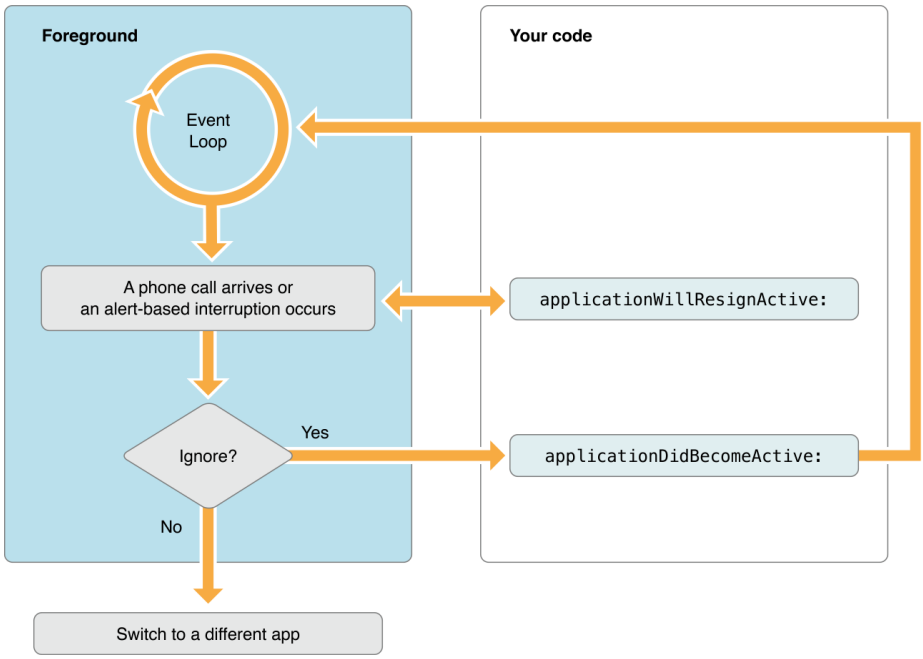

Figura 2.12: Gestione interruzioni [6].

Quando si verifica un'interruzione si ha un perdita di controllo dell'app, essa continua a funzionare in foreground ma non riceve eventuali eventi di touch dal sistema. In risposta a un'interruzione l'applicazione deve eseguire queste operazioni:

- Terminare task periodici e timer
- Interrompere tutte le richieste dei metadata in esecuzione
- Non avviare nuovi tasks
- Mettere in pausa riproduzione di un film
- Entrare in uno stato di pausa se l'applicazione è un gioco
- Sospendere eventuali code di spedizioni
- Interrompere OpenGL ES

Una volta riattivata l'app, dopo aver fatto fronte all'interruzione, verranno riattivate le operazioni elencate pocanzi.

2.7.3 Applicazione in Background

L 'applicazione passa in background alla pressione del tasto Home, sleep/-wake o il sistema lancia una nuova app, quindi l'applicazione in foreground passa allo stato inactive quindi in background. Le transizioni vengono gestite da applicationWillResignActive e applicationDidEnterBackground metodi che indicano al delegate che l'applicazione passa rispettivamente in stato di active e background come si vede in figura.

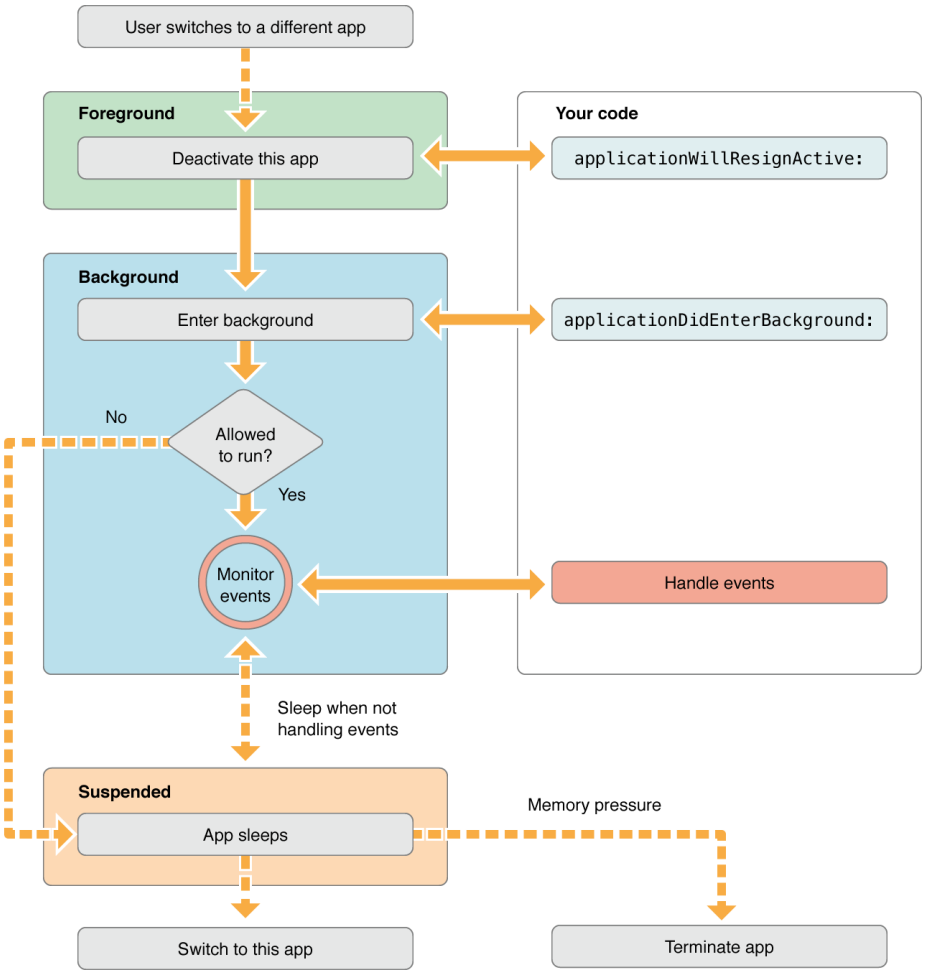

Figura 2.13: Passaggio da foreground a background [6].

Ogni applicazione in background deve liberare memoria, il sistema cerca di mantenere le applicazioni in memoria ma appena la memoria si esaurisce le applicazioni in background vengono terminate. Le applicazioni che consumano più memoria sono le prime a essere terminate. Le applicazioni dovrebbero rimuove i riferimenti a oggetti non appena non sono più necessari. Rimuovendo tali riferimenti il compilatore ha la capacità di rilasciare gli oggetti subito. Se i invece si utilizza la memoria cache per memorizzare alcuni oggeti così da migliorare le prestazioni, bisogna attendere prima la transizione in background per poi eliminare i riferimenti a tali oggetti. Ad esempio si potrebbero eliminare i riferimenti di alcuni oggetti, quali:

- Oggetti immagine
- Grandi supporti o file di dati che è possibile caricare sucessivamente
- Altri oggeti che l'applicazione può ricreare successivamente.

2.7.4 Applicazione in Foreground

L' applicazionetornando in foreground riavvierà le attività che si erano interrotte quando passò in background. I passaggi che si verificano quando l'applicazione si sposta in foreground sono rappresentati in figura, il metodo applicationWillEnterForeground annulla tutto ciò che era stato fatto con il metodo applicationDidEnterBackground, invece il metodo applicationDidBecomeActive continua a svolgere i compiti che si fanno in fase di lancio. un'applicazione in stato suspended deve poter ricevere alcune notifiche,ad esempio modifiche di orientamento, cambiamento di tempo e così via, che gestirà non appena torna in foreground, tali notifiche per non andare perse vengono messe in code di sistema e verranno consegnate non appena l'applicazione torna in foreground. Se le notifiche dovessero essere tante, per non sovraccaricare l'applicazione il sistema fonde gli evendi per consegnare una singola notifica.

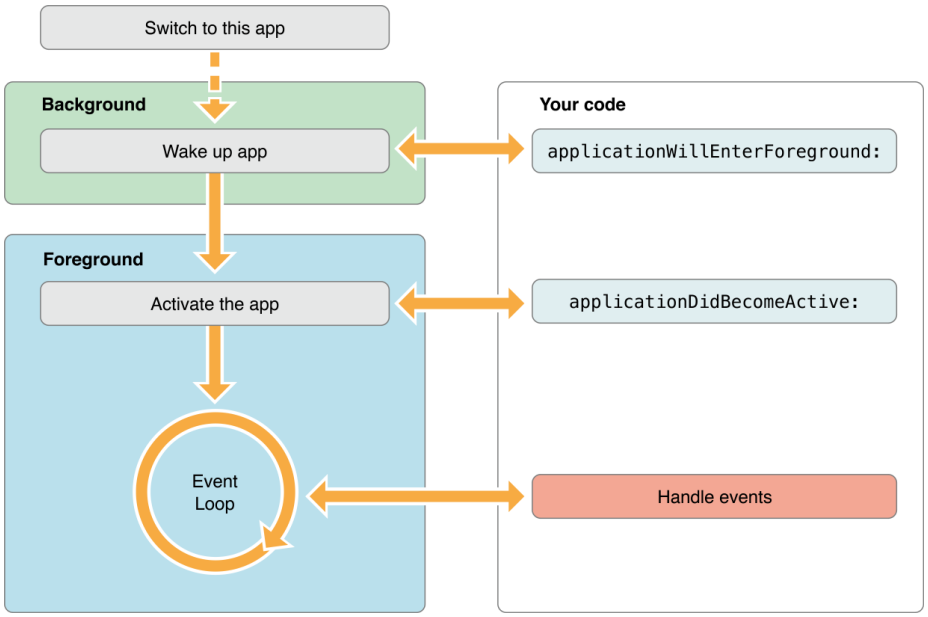

Figura 2.14: Passaggio da background a foreground [6].

2.7.5 Terminazione App

Nel momento in cui l'applicazione è in fase di terminazione, sia in foreground che in background, il sistema chiama il metodo applicationWillTerminate che indica al delegate quando l'applicazione sta per terminare. Il metodo effettua operazioni di pulizia, salva i dati utente o informazioni di stato dell'applicazione così da ripristinare l'applicazione allo stato attuale nel lancio successivo, ha 5 secondi per fare il tutto altrimenti verra "ucciso" e rimosso dalla memoria. l'applicazione deve essere preparata nel caso in cui l'utente "uccide" l'applicazione in modo esplicito utilizzando l'interfaccia multitasking, infatti l'applicazione trovandosi in background il sistema chiama il metodo applicationWillTerminate.

2.7.6 Il Main Run Loop

Il main run loop di un'applicazione è responsabile dell'elaborazione di tutti gli eventi scaturiti dall'utente. l'oggetto UIApplication configura il main run loop in fase di lancio e lo usa per elaborare eventi e aggiornare le view in base agli eventi. Come suggerisce il nome il main run loop viene seguito

dal main thread dell'applicazione così da assicurare che gli eventi relativi all'utente vengano elaborati serialmente nell'ordine in cui sono stati ricevuti. In figura è rappresentata l'architettura del main run loop. Il sistema genera eventi dovuti all'interazione dell'utente con un dispositivo, tali eventi vengono consegnati attraverso una porta speciale istituita da UIKit. Gli eventi messi in coda verranno spediti uno a uno al main run loop, poi l'oggetto UIApplication riceve l'evento e prende la decisione su ciò che deve essere fatto.

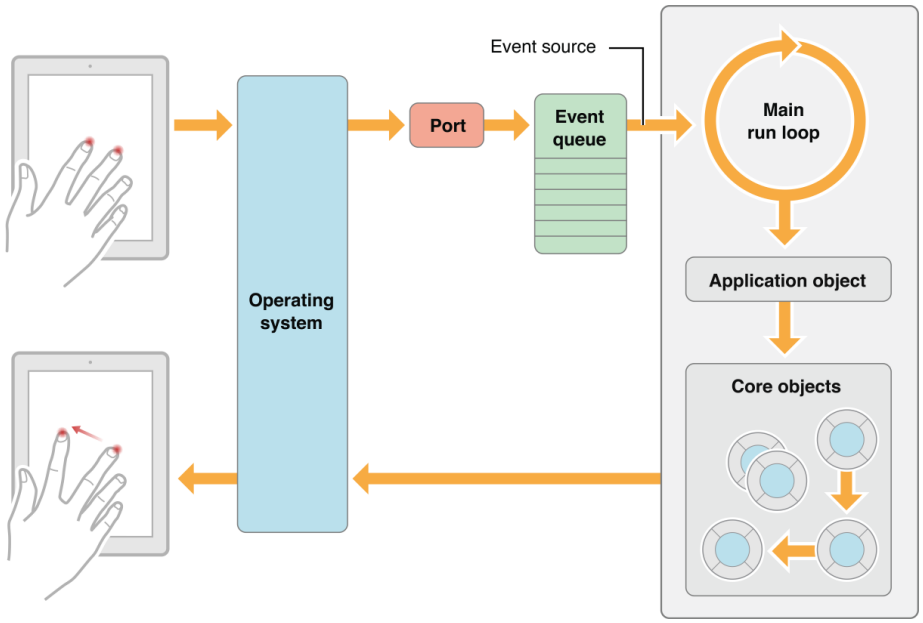

Figura 2.15: Elaborazione eventi del Main Run Loop [6].

In iOS 4 e versioni successive, il multitasking consente all'applicazione di continuare l'esecuzione in background anche se l'utente sta usando un'altra app, preservando la durata della batteria.

2.8 Concorrenza

Siamo in un era dove la tecnologia non si ferma neanche davanti alle limitazioni fisiche dovuti al calore limitando la velocità dei processori. Infatti i produttori per poter aumentare il rendimento dei chip, hanno aumentato

i processori su ogni chip. Aumentando il numero di core su ogni chip si potevano eseguire più istruzioni al secondo. Nasce un problema: come si possono sfruttare i core in più?

iOS adotta un approccio di progettazione asincrona per risolvere il problema della concorrenza. Le funzioni asincrone esistono da molto tempo, utiliizzate per avviare le attività che possono richiedere molto come la lettura dei file sul disco. Quando una funzione asincrona viene chiamata fà un certo lavoro per avviare un task. Tale lavoro comporta l'acquisizione di un thread in background, l'avvio di un task desiderato su esso e l'invio di una notifica al chiamante attraverso una funzione di callback.

2.8.1 Grand Central Dispatch

Una delle tecnologie per l'avvio di task in modo asincrono è il Grand Central Dispstch. Esso comprende funzionalità e librerie, e fornisce miglioramenti per il supporto per l'esecuzione di codice concorrente su hardware multicore in iOS e OS X. GCD si occupa di creare i thread necessari e di pianificare i task su quei thread. Tale tecnologia prende il codice di gestione dei thread scritta in un'applicazione e muove tale codice a livello di sitema, quindi in poche parole la responsabilità nell'esecuzione e gestione dei thread passa dal livello applicativo a quello del sistema operativo. Gestisce code di tipo FIFO a cui l'applicazione può inviare i propri task in forma di block object. I Blocchi presenti in dispatch queues vengono eseguiti su un pool di thread, gestiti dal sistema operativo. GCD offe tre tipi di code:

- **Main:** task eseguiti in modo seriale sul main thread dell'app
- **Concurrent:** i task sono prelevati con ordine FIFO, ma vengono eseguiti contemporaneamente e possono finire in qualsiasi ordine
- **Serial:** task eseguiti uno alla volta in ordine FIFO

Grazie a questa tecnologia tutto quello che un programmatore deve fare è definire i task che desidera eseguire e aggiunderli in una dispatch queue.

2.8.2 Dispatch Queues

I Dispatch Queues rappresentano un meccanismo basato su C per l'esecuzione di attività personalizzate, tale meccanismo esegue compiti sia in serie che in concomitanza ma sempre e comunque con ordine FIFO. Di particolare importanza sono i vantaggi introdotti grazie al suo utilizzo:

- Fornisce un'interfaccia di programmzione semplice;
- Offre la gestione automatica del pool di thread;
- è molto più efficiente della memoria;
- l'invio asincrono a una dispatch queue non provoca problemi di deadlock;
- I serial dispatch queue offrono un'alternativa efficiente per lock e altre forme di sincronizzazione.

I task inviati a una dispatch queue devono essere incapsulati all'interno di una funzione o block object. È importante ricordare che i dispacth queues fanno parte di quella che viene definita "Tecnologia Grand Central Dispatch".

2.8.3 Dispatch Sources

I Dispatch Sources rappresentano un meccanismo basato su C utilizzato al fine di effettuare l'elaborazione in modo asincrono di specifiche tipologie di eventi di sistema. Tale meccanismo agisce incapsulando informazioni su un particolare tipo di evento di sistema e lancia un block object specifico o una funzione a una Dispatch Queue ogni volta che si verifica l'evento. Viene utilizzato questo meccanismo per monitorare i seguenti tipi di eventi:

- Timer
- Signal handler
- Descrittori di eventi correlati
- Elaborazione di eventi correlati
- Eventi della porta Mach
- Eventi trigger

è bene specificare che i Dispatch Sources fanno parte di ciò che viene definito "Grand Central Dispatch".

2.8.4 Operation Queues

Le Operation Queues vengono implementate dalla classe NSOperationQueue, eseguono compiti sempre rispettando l'ordine FIFO e prendono in considerazione determinati fattori al fine di stabilire l'ordine di esecuzione dei vari task. Primo tra questi fattori è il fatto che un determinato task dipende a sua volta dal completamento di altri task, tali dipendenze si configurano quando vengono definiti i vari task. È importante sottolineare che i task presenti in un'Operation Queue devono essere istanze della classe NSOperation. Attribuiamo ad un oggetto operation la definizione di Objective-C che incapsula il task che si desidera eseguire insieme a tutti i dati ovviamente neccessari per eseguirlo. Tali oggetti appena citati generano "key-value observg notification (KVO)" che è un modo utile per monitorare lo stato di avanzamento di un task. Le Operation Queues hanno la particolarità di eseguire più operazioni contemporaneamente ma al contempo è possibile utilizzare le varie dipendenze al fine di garantire che l'esecuzione venga effettuata in modo seriale quando se ne verifichi la necessità.

2.8.5 Performance

Le tecnologie descritte pocansi non garantiscono il miglioramento dell'efficienza dell'app. Il programmatore è responsabile nell'utilizzare le code in modo efficace per un'app, senza imporre un carico eccessivo alle altre risorse dell'app. Ad esempio creare 10.000 oggetti operatione, presenti in un operation queue, l'applicazione per allocare una quantità di memoria non banale porterebbe a paging e quindi a una riduzione delle prestazioni. Se l'introduzione della concorrenza rende l'applicazione meno efficiente si possono utilizzare degli strumenti per verificare potenziali cause come:

- Shark
- Instruments
- Analysis Tools
- Monitoring Tools
- Hardware Analysis Tools
- Additional Command-Line Tools

Capitolo 3

Phonegap

Phonegap è un framework open source sviluppato dall'azienda canadese Nitobi e succesivamente acquistata da adobe nell'ottobre 2011. Tale framework viene utilizzato per sviluppare applicazioni native attraverso tecnologie web. Quanto appena detto significa che le applicazioni mobile vengono scritte in HTML5, CSS3 e JavaScript.

3.1 Struttura

Le applicazioni che utilizzano questo framework come già detto vegono scritte in HTML, CSS3 e JavaScript, il prodotto finale di un'applicazione phonegap è un archivio di applicazioni che possono essere distribuite attraverso i vari Store quali App Store, Android Market e così via. Di conseguenza per un'applicazionei OS l'output è un file IPA (iOS Application Archive), per le applicazioni Android l'output è un file APK (Android Package) ecc... Questi sono gli stessi formati di packaging utilizzati dalle applicazioni native per poterle distribuire sui relativi store.

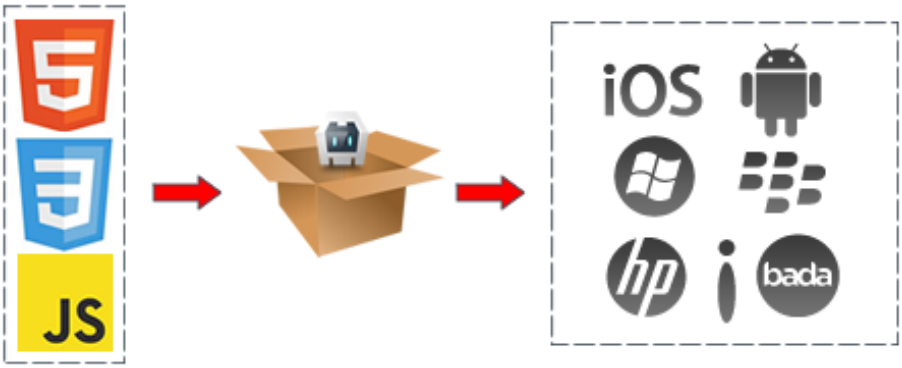

Figura 3.1: Phonegap

Le applicazioni sviluppate usando phonegap sono applicazioni ibride. Cosa significa applicazioni ibride? Significa che le applicazioni non sono scritte puramente in HTML5 e JaVaScript, né esclusivamente in linguaggio nativo. Entrando nello specifico è importante dire che parti dell'applicazione, quindi interfaccia utente, logica e comunicazione con il server sono basate su HTML e JavaScript. Invece la parte che comunica e controlla il dispositivo è scritta in linguaggio puramente nativo. Da questo punto di vista quindi possiamo dire che phonegap fornisce un ponte tra la web app e la piattaforma nativa, esso consente all'API Javascript di accedere e controllare il dispositivo stesso. È fondamentale dire che PhoneGap userà il linguaggio nativo della piattaforma per accedere alle risorse hardware e software in modo da aggiungere le funzionalità di base al motore JavaScript e renderle così facilmente utilizzabili dall'applicazione come fossero tradizionali metodi di libreria.

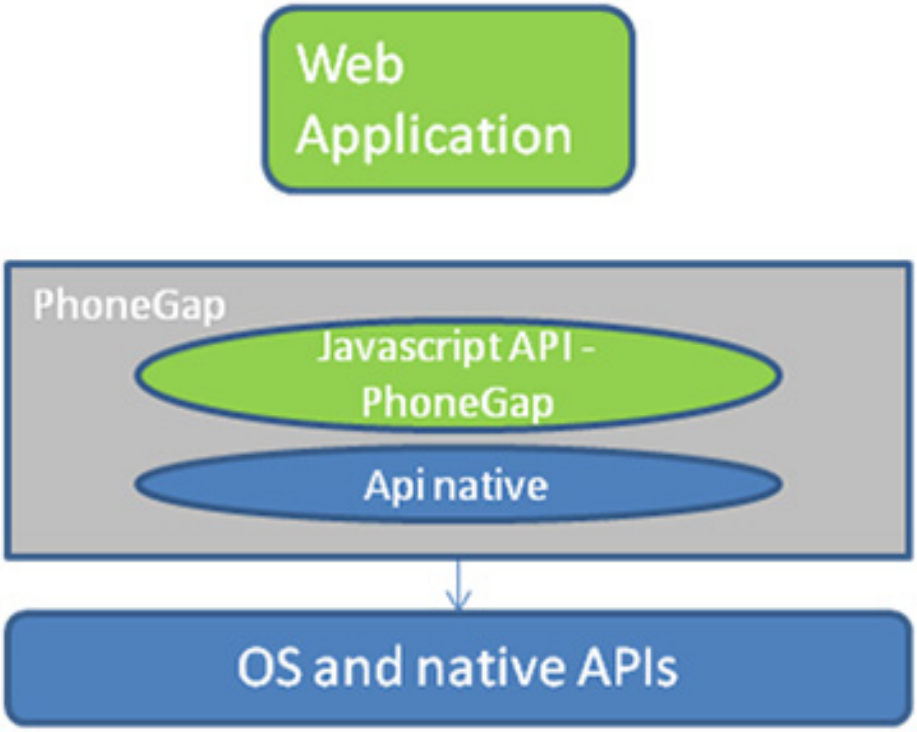

Figura 3.2: Interfacciamento di Phonegap

3.1.1 API Phonegap

Il framework Phonegap è principalmente una libreria JavaScript che permette alle applicazioni, scritte in HTML5 e JavaScript, di accedere alle funzioni del dispositivo. Le API del framework Phonegap sono:

- **Accelerometer:** cattura il moto del dispositivo nelle tre direzioni x, y e z.
- **Camera:** consente alle applicazioni di accedere alla fotocamera del device.
- **Capture:** fornisce l'accesso all'audio, immagini e video effettuati grazie alla capacità di cattura del dispositivo
- **Compass:** ottiene la direzione che il dispositivo sta indicando.

- **Connection:** consente di accedere alle informazioni di connessione (cellulare, wifi) del dispositivo.
- **Contacts:** consente l'accesso ai contatti del dispositivo.
- **Device:** descrive l'hardware e il software del dispositivo.
- **Events:** gestisce e controlla gli eventi.
- **File:** é un API basata sul W3C API FILE, serve per leggere, scrivere e navigare nel file system.
- **Geolocation:** fornisce l'accesso al sensore GPS del dispositivo.
- **Globalization:** ottiene informazioni ed esegue operazioni specifiche per le impostazioni locali dell'utente e la timezone.
- **InAppBrowser:** é un broswer web che viene visualizzato in un'applicazione quando viene aperto un URL.
- **Media:** offre la possibilità di registrare e riprodurre i file audio su un dispositivo.
- **Notification:** gestisce le notifiche del dispositivo.
- **Splashscreen:** consente agli sviluppatori di mostrare / nascondere la schermata iniziale dell'applicazione.
- **Storage:** consente di accedere alle opzioni di archiviazione dei dispositivi.

Caratteristiche e funzionalità variano tra i diversi dispositivi e piattaforme, ad esempio alcuni dispositivi Android e IPhone hanno la bussola altri invece no.

3.2 Organizzazione di un'applicazione Web

Le innumerevoli API spaziano da aspetti prettamente ludici, come la creazione di disegni all'interno del canvas (tela), alla visualizzazione di video e musica senza alcun plug-in esterno (Flash o Silverlight) fino ad aspetti più significativi come la capacità di lavorare con i thread, drag e drop e così via. Quindi è possibile costruire applicazioni in grado di "mimare" per complessità e interattività quelle standalone, che normalmente si progettano per

Figura 3.3: Caratteristiche per diverso Sistema Operativo mobile

sistemi operativi quali Windows, Linux, Mac e per SO mobile iOS, Android e così via. Le tecnologie per costruire applicazioni web sono:

- **HTML5:** usato come linguaggio di markup e layer strutturale, definisce la struttura del documento.
- **CSS3:** usato come linguaggio di visualizzazione e layer di presentazione, è lo strumento fondamentale per definire come un documento HTML deve essere visualizzato da un broswer.
- **JavaScript:** usato come linguaggio di programmazione e layer di interazione, è il linguaggio di programmazione più usato e più importante per progettare e sviluppare applicazioni web.
- **Framework Phonegap:** o similari, usato per facilitare la programmazione del web e per permettere una compatibilità cross-platform e cross-broswer.

DI seguito verrano descritte le funzionalità più importanti per lo sviluppo di applicazioni mobile.

3.2.1 HTML5

Definiamo HTML5 come linguaggio di markup per la strutturazione delle pagine web. HTML5 introduce un valore aggiunto alle originarie funzionalità del classico HTML, inteso appunto come semplice strumento di markup in particolar modo importante in PhoneGap per la completezza delle interfacce di programmazione che mette a disposizione. Per essere più precisi possiamo dire che le novità introdotte da HTML5 rispetto a HTML4 hanno il fine primario di migliorare il disaccoppiamento tra la struttura definita dal markup, le caratteristiche di resa(tipo di carattere, colori) definite dalle direttive di stile e i contenuti di una pagina web definiti dal testo vero e proprio. HTML5 fornisce anche il supporto per la memorizzazione locale di ingenti quantità di dati scaricati dal web browser al fine di consentire l'utilizzo di applicazioni web anche in assenza di collegamento internet. Inoltre è di particolare importanza dire che con l'avvento di HTML5:vengono rese più stringenti le regole per la strutturazione del testo in capitoli, paragrafi e sezioni; vengono introdotti elementi di controllo per i menù di navigazione; vengono migliorati ed ampliati gli elementi di controllo per i moduli elettronici; vengono introdotti specifici elementi utilizzati per il controllo di contenuti multimediali; vengono addirittura eliminati determinati elementi dimostratisi scarsi o aventi nessun utilizzo effettivo; viene reso possibile il supporto di Canvas che permette di utilizzare JavaScript al fine di creare animazioni e grafica bitmap; viene introdotta la "geolocalizzazione" grazie all'espansione di SO quali Android e iOS; viene introdotto un sistema più efficiente, Web Storage, in alternativa ai classici cookie, con risultato finale il notevole risparmio di banda; vengono standardizzati programmi quali JavaScript, chiamati Web Workers, con la possibilità di utilizzare alcuni siti offline; viene effettuata la sostituzione di luogo e complesso doctype con un semplice < !DOCTYPE html > .

3.2.2 CSS3

Il CSS3 è l'ultima versione del CSS (Cascading Style Sheets o Fogli di stile) che definiamo essere un linguaggio informatico utilizzato al fine di definire la formattazione di documenti HTML, XHTML e XML in siti web e relative pagine web. Le regole vigenti per la composizione del CSS sono contenute in un insieme di direttive emanate dal 1996 dal W3C. L' introduzione del CSS è divenuta necessaria per riuscire a separare i contenuti dalla formattazione e permettere così facendo una programmazione più chiara e di facile utilizzo sia per gli autori di pagine HTML che semplicemente per gli utenti,

garantendo al contempo la possibilità di riutilizzare il codice e di rendere la sua manutenibilità più semplice.

3.2.3 JavaScript

JavaScript è un linguaggio di scripting orientato agli oggetti utilizzato comunemente per la creazione di siti web, quindi ci si riferisce ad esso quando si parla di Programmazione Web. JavaScript ha delle caratteristiche peculiari tra cui possiamo citare: é un linguaggio interpretato infatti il codice non viene compilato ma interpretato;

- la sintassi è relativamente simile a quella di C, C++ e Java;
- definisce le funzionalità caratteristiche di quelli che definiamo "linguaggi di programmazione ad alto livello" (strutture di controllo, cicli ecc.) e consente l'utilizzo del paradigma object oriented;
- è in realtà comunque un linguaggio debolmente orientato agli oggetti in quanto, ad esempio, il meccanismo dell'ereditarietà è più simile a quello del Self e del Newton Script che a quello del linguaggio Java che, come sappiamo, è un linguaggio fortemente orientato oggetti.

Vi sono comunque altri aspetti di particolare interesse che caratterizzano questo linguaggio, infatti possiamo dire che, in JavaScript lato client, il codice viene direttamente mandato in esecuzione sul client e non sul server. L'aspetto vantaggioso di questo tipo di approccio è che, nonostante ci possa essere la presenza di script particolarmente complessi, il web server non viene sovraccaricato dalle richieste dei clients. Al contempo però nel caso di script con codice sorgente particolarmente grande il tempo per il download può rivelarsi più lungo, inoltre può presentarsi l'inconveniente presenza di un ulteriore svantaggio: ogni informazione che comporta un accesso a dati memorizzati in un database remoto deve necessariamente essere rimandata ad un determinato linguaggio che effettui in modo esplicito la transazione al fine di restituire i risultati ad una o più variabili di JavaScript, operazioni di questo tipo richiedono inoltre il caricamento della pagina stessa. Successivamente questi limiti in particolare sono stati superati grazie all'entrata in scena di AJAX.

3.3 L' elemento Canvas

Esso innesta in un documento HTML un'area di disegno al cui interno possiamo compiere molte operazioni legate alla gestione e manipolazione grafiche. Quindi ci fornisce la possibilità di disegnare oggetti grafici e singoli pixel, e applicare trasformazioni quali rotazioni, traslazioni. Volendo fare una puntualizzazione per quanto riguarda il codice possiamo dire che non cambia niente quindi l'elemento canvas viene definito come qualsiasi altro elemento della struttura.

3.4 Elementi Multimediali

Arricchiscono una pagina di contenuti multimediali quali brani audio e filmati video senza l'utilizzo di plug-in esterni quali Flash, QuickTime, Silverlight. I tag audio e video sono elementi DOM e sono simili ai tag img. Con l'ausilio di JavaScript è possibile ascoltare gli eventi ad esempio quando viene premuto play viene eseguito il codice relativo all'evento.

3.5 Drag and Drop

Il DnD è una potente funzionalità riguardante le interfacce utente, che permette di trascinare oggetti sorgenti (immagini, testo e così via) e rilasciarli su altri oggetti destinatari (immagini, testo, aree vuote e così via). In generale è caratterizzata da tre azioni che consistono:

- cliccare su un oggetto
- trascinare l'oggetto in un'altra posizione(drag)
- rilasciarlo (drop)

Sia con JavaScript che con le API di HTML5, il DnD è attuabile con qualsiasi elemento renderizzato nella pagina web e le operazioni di trascinamento e rilascio sono effettuabili sia all'interno della finestra del broswer sia con le finestre di altre applicazioni esterne.

3.6 Web Worker

Le API Web Workers permettono di eseguire parti di codice scritte in JavaScript, esse richiedono un'elevata complessità di calcolo in background quin-

di eseguibili in thread separati.Tali API facilitano il lavoro a quegli svilppatori che progettano applicazioni sofisticate, che senza thread causerebbe uno stallo della pagina web.

3.7 Storage dei dati lato client

La progettazione di un'applicazione web richiede la possibilità di memorizzare dati localmente nel dispositivo in uso dell'utente. Esistono tre tecnologie per poter memorizzare i dati:

- **Cookie:** rappresenta un frammento di dato che può essere scambiato, durante una comunicazione, tra una sorgente e una destinazione.
- **Web storage API:** tecnologia standard (W3C) per lo storage locale dei dati che per renderli persistenti si avvale di array associativi che mappano chiavi a valori.
- **Database API:** serie di tecnologie che consente di memorizzare i dati in modo sofisticato ed efficienti usando meccanismi propri dei database.

3.8 Cross-origin messaging

Cross-origin messaging è un meccanismo che fornisce la possibilità di scambiare dei dati tra documenti di pagine web, aventi un origine differente, in modo sicuro e affidabile evitando i subire attacchi di cross-site scripting (vulnerabilità che affligge i siti web dinamici).

3.9 Offline

HTML5 offre la possibilità di progettare applicazioni che possono essere utilizzate offline, ovvero senza una connessione di rete, grazie ad uno spazio di storage definito ‘applicazion cache’ dove sono memorizzati tutti i file del codice e delle risorse di tale applicazione che sono definite su file apposito "cache manifest". Se la connessione di rete è disponibile verrà fatto un controllo automatico e asincrono per verificare un eventuale cambiamento del cache manifest. Se tale modifica dovesse essere avvenuta, quindi se le risorse sono cambiate devono essere scaricate e memorizzate per un update

nella relativa application cache in modo tale da utilizzare l'applicazione aggiornata

3.10 Microdata

I microdata permettono di attribuire agli elementi HTML delle ´informazioni´ aggiuntive come dei metadati che definiscono al livello semantico lo scopo per il quale sono stati impiegati. In pratica "etichettano" gli elementi HTML mediante delle proprietà interpetrabili dai broswer, moti di ricerca e software in grado di elaborarle.

3.11 Eventi

Per quanto riguarda l'ambito della programmazione lato client gli eventi rappresentano l'elemento cardine attraverso cui avere la possibilità di gestire l'iterazione di un utente con la finestra del browser,con il documento contenuto in essa e con qualsiasi elemento renderizzato all'interno del medesimo documento. Con il termine "evento" si intende identificare la manifestazione di un qualche accadimento(pressione di un tasto, clic del mouse, fine dell'upload del documento, aggiunta di un elemento e così via)occorso su un target(finestra, documento o elemento) che il browser rileva e notifica a un programma il quale a sua volta, attraverso un opportuno codice, decide di conseguenza le azioni da intraprendere. Focalizzando la nostra attenzione all'ambito della programmazione di web application è utile sottolineare che i browser adottano il modello di gestione degli eventi così come è stato formalizzato dalla specifica del W3C denominata Document Object Model (DOM) Level 3 Events Specification. Tale specifica definisce in maniera dettagliatauna serie di regole e interfacce per la realizzazione di una serie di eventi, generici e al contempo indipendenti dalla piattaforma e dal linguaggio al fine di:

- registrare dei gestori per gli eventi;
- descrivere come gli eventi "fluiscono" attraverso attraverso quelli che definiamo nodi di una struttura ad albero di elementi;
- ottenere informazioni contestuali riguardanti il tipo di evento occorso.

Avendo l'obiettivo finale di riuscire ad utilizzare con cognizione di causa il sistema di eventi occorre comprendere dettagliatamente specifici termini:

- **EVENT TYPE:** tipo di evento) è un oggetto istanza di un determinato tipo di evento che ha un nome,condizioni di attivazione e caratteristiche proprie che lo distinguono da altre tipologie di eventi. Per meglio chiarire tale definizione facciamo un esempio, l'evento di nome "clic" è un oggetto di tipo MouseEvent con proprietà specifiche quali "clientX, clientY" e così via.

- **EVENT TARGET:** (destinatario dell'evento) è l'oggetto verso il quale è indirizzato l'evento stesso.

- **EVENT HANDLER:** (gestore dell'evento) è il codice che viene registrato o associato a un determinato oggetto e che a sua volta viene invocato dal browser quando il relativo tipo di evento accade sull'oggetto al fine di far compiere specifiche operazioni.

- **EVENT FLOW:** (flusso di un evento) è definita tale la modalità di propagazione di un evento ossia il percorso che l'oggetto evento compie attraverso i nodi della struttura di un documento per poi giungere al relativo target. Tale percorso appena citato può a sua volta essere diviso in varie fasi: capturing phase, at-target phase,bubbling phase. Partendo dalla "capturing phase" l'oggetto evento si propaga attraverso i nodi antenati del target, quindi dalla default view, per arrivare fino al nodo genitore del target. Ogni gestore di evento registrato per tali nodi può gestire il relativo evento prima che giunga al target, successivamente in quella che definiamo "at-target phase" l'oggetto evento giunge al nodo destinatario e l'eventuale handler associato si occupa di gestirlo. Nell'ultima fase "bubbling phase" l'oggento evento continua la sua propagazione attraverso i nodi antenati del target, nello specifico dal nodo genitore del target fino alla defaultView.

- **DEFAULT ACTION:** (azione di default) è la logica di comportamento associata di default a un oggetto evento.

3.12 Assenza di un ambiente di sviluppo

Phonegap pur essendo un framework non fornisce IDE o ambienti di sviluppo speciali per la codifica. Se si volesse sviluppare un'applicazione Phonegap per iOS bisognerebbe utilizzare Xcode oppure se si volesse sviluppare un'applicazione Phonegap per Android bisognorebbe utilizzare Eclipse e Android SD.

Capitolo 4

Confronto

Nel presente capitolo si cercherà di centrare l'attenzione su quelle che possono essere differenze o eventuali similitudini per quel che concerne lo sviluppo di una serie di semplici applicazioni con cui andare ad effettuare il test di alcuni degli aspetti di maggiore interesse per lo sviluppo di mobile app. Quanto detto è stato effettuato tramite l'utilizzo della piattaforma iOS e il relativo confronto dello sviluppo delle stesse semplici applicazioni utilizzando, in un secondo momento, il framework PhoneGap al fine di evidenziarne pro e contro. A tale scopo nello svolgimento del mio lavoro di tesi ho sviluppato appunto una serie di semplici applicazioni utilizzando le due diverse metodiche al fine di evidenziarne aspetti caratteristici sia dell'una che dell'altra. Comunque per effettuare lo sviluppo delle applicazioni native ho fatto uso del software Xcode e per testarle ho utilizzato il relativo simulatore sempre di XCode mentre per le web app ho usato eclipse per Android e per testarle ho usato un dispositivo Android.

4.1 Supporto

In tale sezione mi soffermerò a descrivere in dettaglio il supporto che iOS e Phonegap danno alle diverse tipologie di dispositivi quindi diverse risoluzione dello schermo e diversa versione del SO installata.

4.1.1 Supporto di iOS

Quando si parla d iOS è naturale pensare alla stretta connessione con Xcode per quanto riguarda lo sviluppo delle applicazioni . Xcode ha la particolarità di facilitare in maniera sostanziale la creazione di applicazioni sui diversi

dispositivi di casa Apple quali iPhone, iPad, iPod, fornendo in tal modo il supporto necessario per quanto riguarda la diversa dimensione del display e i differenti SO. Se ci si sofferma sulla nuova versione di Xcode, la 4.5, bisogna sottolineare che supporta l' architettura armv7 che depreca l'iPhone a partire dal modello 3g e precedenti. Inoltre c' è da dire che Apple ha ufficialmente divulgato la notizia in relazione alla quale fa presente la sua accettazione solo di nuove applicazioni e update inviate all'App Store e costruite su dispositivi iOS con display retina, quindi tutti gli iPad e gli iphone a partire dal 4 in poi considerando il 4 stesso compreso. Al fine di voler sviluppare applicazioni che supportano una vasta gamma di versioni del Sistema Operativo, iOS offre la possibiltà di utilizzare controlli a runtime per impedire l'esecuzione di funzioni non presenti nel sistema operativo in uso. Quindi per verificare se una determinata funzione è disponibile o meno per una data versione di iOS si può procedere facendo un semplice confronto come si vede nel listato seguente.

Listing 4.1: Verifica se la funzione è disponibile

```
if (UIGraphicsBeginPDFPage != NULL)
{

  UIGraphicsBeginPDFPage();

}
```

Pertanto, se si verifica che la funzione è disponibile per la versione corrente di iOS presente sul dispositivo in uso, viene semplicemente eseguita. Per ciò che concerne invece il supporto per i diversi display, ad esempio la visualizzazione di un'immagine, l'app deve contenere in totale sei file separati della stessa immagine, di cui tre file per iphone e ipod e tre file per ipad, ovviamente ognuno avente risoluzione dell'immagine adatta al determinato tipo di dispositivo che si intende utilizzare. In particolare ogni file deve essere salvato con un criterio specifico che è : < ImageName > < device-modifier > . < filenameextension > per immagini da visualizzare su display non retina invece per i display ad alta risoluzione (retina) viene aggiunto "@2x", volendo fare un esempio quindi sarebbe "mela-iphone@2x.png". Invece con l'uscita del nuovo iPhone con schermo 4 inch c' è la necessita di aggiungere 568h,quindi ad esempio in tal caso diventerà "mela-iphone-568h@2x.png. La classe UIImage si occuperà di caricare l'immagine corretta per il determinato tipo di dispositiva e relativa risoluzione del display.

Listing 4.2: Caricare immagine

```
UIImage *anImage = [UIImage imageNamed:@"mela"];
```

4.1.2 Supporto di Phonegap

Come è stato già in precedenza anticipato, una volta realizzata un'app facendo uso del framework PhoneGap l'app in questione si adatta a qualunque piattaforma supportata. Per quel che riguarda il supporto per la risoluzione del display una soluzione molto valida è quella di ricorrere all'utilizzo di una media query nel file .css . In tal caso la media query applica una specifica immagine se la condizione si rivela varitiera.

Listing 4.3: Esempio per iPad

```
/* iPads (landscape) */
@media only screen
   and (min-device-width : 768px)
   and (max-device-width : 1024px)
   and (orientation : landscape) {

   background: url(pictures/image_land.png);
}

/* iPads (portrait)  */
@media only screen
   and (min-device-width : 768px)
   and (max-device-width : 1024px)
   and (orientation : portrait) {

   background: url(pictures/image_port.png);

}
```

Nell'esempio riportato viene verificato se lo schermo del dispositivo (lo screen) ha una specifica risoluzione e se la condizione è vera viene caricata quella determinata immagine.

4.1.3 Confronto zero

In relazione a quello che può essere il confronto fra iOS e PhoneGap possiamo fare alcune precisazioni in merito. PhoneGap ha la peculiarità di facilitare lo sviluppo di applicazioni non solo su varie versione di un Sistema Operativo ma anche lo sviluppo per tutte le piattaforme supportate. iOS fornisce facilmente supporto per lo sviluppo di applicazioni per le sue varie versioni. Per quanto riguarda iOS possiamo dire che gli sviluppatori potrebbero anche evitare di implementare i vari controlli al fine di verificare se una determinata funzione è presente su una specifica versione del SO considerato perché appunto iOS, a differenza di Android, ha il maggior numero di dispositivi aggiornati alla versione più recente del suo Sistema Operativo Mobile. Infatti tale aggiornamento riguarda il ben 93% dei devices in uso. Per concludere possiamo dire che sia PhoneGap che iOS non presentano alcun problema relativo al supporto delle risoluzioni e dimensioni dei vari display.

4.2 Organizzazione di un'app in generale

In questo paragrafo tramite lo sviluppo di una mia piccola e semplice applicazione , da prendere come esempio, si ha l'intenzione di cercare di mettere in evidenza in maniera più chiara e comprensiva possibile quella che è l'organizzazione strutturale a livello generale dell'app vista sia dal punto di vista della piattaforma iOS che dal corrispondente punto di vista del framework preso come termine di paragone PhoneGap. L' applicazione in esame esegue la semplice operazione algebrica di somma, vengono forniti due valori che l'utente, in maniera semplice ed intuitiva, non deve fare altro che digitare nelle applicazioni osite textField posizionate una di fianco all'altra e successivamente eseguendo un semplice touch, sul relativo button adempito a tale funzione, viene immediatamente visualizzato il valore che rappresenta l'effettiva somma dei due valori forniti in precedenza. Vista la sua funzionalità e fine ultimo ho denominato tale piccola app "SumApp". Di seguito, cercando di avere il più neutrale e oggettivo punto di vista possibile, mi soffermerò a delineare in maniera più dettagliata gli aspetti differenti che caratterizzano entrambe le metodiche, utilizzate al fine di sviluppare la stessa applicazione , con l'intenzione così di stilare quelli che, a mio avviso, possono essere considerati pro e contro dell'una o dell'altra metodologia che a priori lo sviluppatore a suo piacimento decide di mettere in atto.

4.2.1 Organizzazione in iOS

Concentrando l'attenzione su quello che definiamo lo sviluppo di un'app in iOS è di facile notazione che l'azienda informatica statunitense Apple fa largo utilizzo del designer pattern Model View Controller. A tal proposito, infatti, ho personalmente notato che creando un progetto base, tramite l'impiego di Xcode, cattura in maniera preponderante la nostra attenzione tale utilizzo del sopracitato MVC. Soffermandoci su quella che è la vera e propria organizzazione strutturale dell'app chiariamo di seguito alcune caratteristiche e aspetti chiave che ne costituiscono le proprietà tipiche dell'app in iOS. UIViewController è una classe Controller che ha il compito di coordinare le interazioni tra l'interfaccia utente (View) ed i dati dell'applicazione stessa (Model). Per ciò che concerne lo sviluppo dell'app in dettaglio si parte creando la View, questo lo possiamo fare o scrivendo tante righe di codice o optando per l'utilizzo di Interface Builder. In particolare è bene chiarire alcune specifiche di questo software appena citato: esso ha la caratteristica di fornire un gradito aiuto per creare l'interfaccia in modo semplice e veloce trascinando molto semplicemente gli oggetti, che si vogliono usare, nelle posizioni desiderate. Il passo successivo a quello del posizionamento degli oggetti è quello di aggiungere le azioni e le outlet, per "azioni" si intende ciò che succede nel momento in cui l'utente preme il pulsante, mentre per "outlet" si intendono i relativi riferimenti agli elementi di input e di output dell'interfaccia.

Listing 4.4: Parte del codice in ViewController.h

```
@property (weak, nonatomic) IBOutlet UITextField *num1;

@property (weak, nonatomic) IBOutlet UITextField *num2;


-(IBAction)button:(id)sender;
```

Dopo basta solo collegare grazie con interface builder gli elementi grafiici e le variabili di istanza. Il controller (ViewController.m) conterà la logica della nostra piccola applicazione .

```
-(IBAction) button:(id)sender{

   risu.text=[NSString stringWithFormat:@"%d",
              [num1.text integerValue]+[num2.text integerValue]];

}
```

4.2.2 Organizzazione in PhoneGap

Per quanto riguarda l'organizzazione dell'app facendo uso del framework Phonegap possiamo definirla alquanto semplice. Infatti per una semplice applicazione come questa sottoposta ad esame o comunque anche per applicazioni più alte a livello di complessità ciò di cui lo sviluppatore ha solo ed esclusivamente bisogno è una pagina HTML e uno script JavaScript. In particolare, soffermandoci sul caso da me studiato, la pagina HTML contiene la struttura della View.

```
<div class="somma" >
       <form name="modulo">
           <div class="left">
               <input type="text" name="x" /> <br />
           </div>
           <div class="right">
               <input  type="text" name="y" /><br />
           </div>
           <br />
           <div class="label">
               <p id="Info">
               Risultato
               </p><br>
           </div>
           <input class="button" type="button" value="SOMMA" onclick="somma()" />
       </form>
```

Invece lo script JacaScript, index.js, contiene la logica dell'app.

```
function somma()
{

    var a, b, c;
    //lc = document.getElementById("Info");

    a = parseFloat(window.document.modulo.x.value);
    b = parseFloat(window.document.modulo.y.value);

    c = a + b;

    lc.innerHTML=c;
    //window.document.modulo.ris.value = c;

}
```

Anticipando un piccolo confronto con iOS, sul quale mi soffermerò più in dettaglio nel paragrafo successivo, possiamo subito notare che in Phonegap la realizzazione dell'interfaccia grafica comporta un tempo maggiore, rispetto all'utilizzo di iOS, poiché utilizzando eclipse l'interfaccia in Phonegap viene realizzata scrivendo righe di codice e via via per visualizzarne i risultati bisogna di volta in volta avviare il simulatore, cosa che appunto al contrario non accade in iOS.

4.2.3 Un primo confronto

Dopo aver soffermato l'attenzione sull'organizzazione in generale dell'app dal punto di vista di iOS e della stessa applicazione dal punto di vista di Phonegap, si hanno allora a disposizione elementi necessari al fine di procedere con l'esposizione di quelle che, a mio avviso, possono essere considerate differenze che caratterizzano l'utilizzo dell'uno o dell'altro tipo di sviluppo. Come già accennato precedentemente in Phonegap la realizzazione dell'interfaccia grafica è più dispendiosa in termini di tempo impiegato rispetto alla realizzazione dell'interfaccia in iOS; nel complesso l'organizzazione dell'app in Phonegap è da ritenere più semplice visto l'utilizzo di una sola pagina HTML e un solo script JavaScript ma al contempo tale aspetto non risulta più così tanto evidente quando si ha a che fare con applicazioni di maggiore e più sostanzioso grado di complessità per le quali l'uso di esclusivamente due soli file sorgenti si rivela non solo inadatto ma anche errato; in iOS è evidente immediatamente la reale separazioni delle varie parti grazie all'uso del Model-View-Controller, quindi questo aspetto implica una organizzazione più ottimale sia se si tratta di applicazioni semplici che di applicazioni più complesse avendo in tal caso la possibilità di estendere o riutilizzare le varie parti in modo semplice. Inoltre per concludere c' è anche da sottolineare che sviluppare un'app utilizzando Phonegap vuol dire anche avere un'interfaccia grafica che cerca in qualche modo di "avvicinarsi e più assomigliare" alla relativa piattaforma per la quale l'app è destinata ma nn sarà mai effettivamente uguale, a tale scopo ci sono delle librerie in uso che offrono la possibilità di "scimmiottare" le interfacce desiderate.

L' esecuzione

Procedendo con la simulazione dell'esecuzione della mia semplice applicazione ho notato che in iOS l'app risulta, in maniera evidente, essere molto più fluida e più veloce rispetto a quella sviluppata in PhoneGap, è doveroso specificare però che tale differenza è stata riscontrata tramite il test fatto sul simulatore, per quanto riguarda l'app in iOS, e il corrispondente test fatto sul dispositivo Android per quanto riguarda l'app sviluppata con PhoneGap. Ineffetti anche soltanto facendo un semplice touch sul button si nota la più lenta risposta del button alla pressione, utilizzando PhoneGap. Da quanto detto risulta essere di facile comprensione che sia aumentando il grado di complessità dell'app che il numero degli oggetti presenti nell'interfaccia grafica si può verificare la provocazione di un rallentamento dell'app in generale (bisogna specificare che sto usando il simulatore quindi in previ-

sione di questo penso che usando un dispoitivo mi dia lo stesso rellentamento o latenza), e di conseguenza una diminuzione delle relative prestazioni specialmente con le attuali architetture multitasking. Tale rallentamento ovviamento è stato notato utilizzando il simulatore di Xcode, come detto sopra, pertanto in base a tale osservazione presumo che possa verificarsi una simile latenza anche eseguendo l'app direttamente sul dispositivo.

4.3 Interazione con i sensori

In questo paragrafo si ha l'intenzione di mettere in evidenza l'interazione di un dispositivo con un sensore, a tale scopo ho scelto come esempio di studio il sensore GPS. Il sensore in esame utilizza una combinazione di approcci che possono includere: posizionamento della WIFI, triangolazione delle torri GSM e il segnale GPS per recuperare latitudine e longitudine che forniscono quindi le coordinate relative alla posizione dell'utente. È di sostanziale importanza sottolineare che l'interazione con i sensori è una delle caratteristiche cardine dei dispositivi mobile che si differenziano appunto dai computer portatili e PC per la presenza di sensori di bordo. Al fine di poter interagire con i sensori è necessario far ricorso all'utilizzo delle API sia per quanto riguarda iOS che per quanto riguarda PhoneGap, quanto appena detto è necessario perché le API forniscono la possibilità di ottenere informazioni dai sensori utilizzando diversi servizi. Quindi a tale dimostrazione ho sviluppato una seconda applicazione che in risposta alla pressione di un pulsante, interagendo con il GPS al fine di recuperare e fornire i valori interessati, visualizza sullo schermo del dispositivo in uso i valori di latitudine e longitudine relativi alla posizione dell'utente che utilizza il dispositivo. In seguito si cercherà di scendere più in dettaglio e definire quella che è l'interazione con i sensori e se c'é la presenza o meno di piccole differenze tra interazione con i sensori in iOS e con PhoneGap.

4.3.1 L' interazione con iOS

Nel caso in esame per poter interagire con il sensore GPS e ottenere di conseguenza i valori di latitudine e longitudine si ha il bisogno di utilizzare le classi del Core Location Framework che fornisce un insieme di servizi che permettono di ottenere e monitorare la posizione corrente del dispositivo. Per utilizzare la funzionalità di tale framework è necessario aggiungerlo al progetto. Successivamente per poter accedere alle classi e headers del framework bisogna includere la dipendenza nel file .h e aggiungere il protocollo

CLLocationManagerDelegate e dichiarare la property di tipo CLLocation-Manage, si utilizza l'istanza di questa classe per stabilire quando la posizione e gli eventi devono essere consegnati e per avviare e arrestare tali eventi.

```
#import <UIKit/UIKit.h>
#import <CoreLocation/CoreLocation.h>


@interface ViewController : UIViewController <CLLocationManagerDelegate>{

    IBOutlet UILabel *latitudine;
    IBOutlet UILabel *longitudine;
}



@property(weak,nonatomic)IBOutlet UILabel *lat;

@property(weak,nonatomic)IBOutlet UILabel *lon;

@property(strong, nonatomic)CLLocationManager *locationManager;


@end
```

Non rimane altro che implementare il metodo del protocollo CLLocationManagerDelegate nel file .m per recuperare i valori di latitudine e longitudine correnti del dispositivo.

```
- (void)locationManager:(CLLocationManager *)manager didUpdateToLocation:
            (CLLocation *)newLocation fromLocation:(CLLocation *)oldLocation
{

  lat.text = [NSString stringWithFormat:@"%f",newLocation.coordinate.latitude];

  lon.text = [NSString stringWithFormat:@"%f",newLocation.coordinate.longitude];

}

@end
```

4.3.2 L' interazione con PhoneGap

Ora centriamo l'attenzione sullo sviluppo dell'app dal punto di vista di PhoneGap. Come abbiamo già precedentemente detto PhoneGap mette a disposizione un insieme di API che ci permette di interagire con il dispositivo senza dover scrivere del codice nativo ma esclusivamente facendo uso di HTML5 e JavaScript. In particolare l'API utilizzata per interagire con

il sensore e la Geolocation che mette a disposozione una serie di funzioni per poter recuperare informazioni dal sensore GPS quali, nel nostro caso, latitudine e longitudine. In particolare oltre alle API Geolocation è bene specificare che c'é la necessità di usare un'ulteriore API "events". Tale necessità nasce dal bisogno di sapere se PhoneGap è a pieno carico o meno. Per fare ciò si utilizza la callback onDeviceReady.

```
function onDeviceReady() {

    alert("Ready");

    lc = document.getElementById("Info");

}
```

Dopo aver verificato che PhoneGap è a pieno regime viene caricata la pagina che contiene un button "Search" che, a sua volta, premuto chiamerà una callback denominata "search" che restituirà, grazie al comando "navigator.geolocation.getCurrentPosition(onLocationSuccess, onLocationError)", la posizione attuale del dispositivo come oggetto posizione. Se l'operazione è andata a buon fine verrà successivamente chiamata la callback onSuccess che provvederrà a visualizzare sullo schermo i valori di latitudine e longitudine, altrimenti in caso di errore verrà chiamata la callback onLocationError che genererà un allert che indicherà il tipo di errore.

```
function onLocationSuccess(loc) {

    lc.innerHTML = ' <b>Latitude</b>: ' + loc.coords.latitude +
            '<br /><b>Longitude</b>: ' + loc.coords.longitude;
}

function onLocationError(e) {

    alert("Geolocation error: #" + e.code + "\n" + e.message);
}
```

4.3.3 Un secondo confronto

Avendo quindi fino a questo momento concentrato l'attenzione sulla descrizione dell'interazione di un dispositivo con un sensore possiamo concludere che, sia per quanto riguarda iOS che per PhoneGap, al fine di riuscire ad interagire con un sensore, c' è la necessità di usare le API relative alla piattaforma che consideriamo, le quali ci danno modo di ricavare le informazioni

desiderate. Soffermandoci però in particolare modo al caso di PhoneGap c' è da fare un'ulteriore precisazione in merito: le API proprie di phoneGap non hanno la facoltà di accedere direttamente ai sensori del dispositivo (cosa che al contrario fanno le API relative a iOS) ma accedono in un primo momento alle API native del dispositivo e solo in un secondo momento invece accedono ai sensori del dispositivo in uso.

4.4 Confronto finale

In seguito alle varie considerazioni fatte in precedenza possiamo ritenere di avere tutti gli elementi necessari per un confronto finale fra un'app nativa e una web app . Tale confronto verrà strutturato in vari sottoparagrafi utilizzando di volta in volta parole chiavi per intitolarli al fine di rendere il punto focale del discorso di più facile comprensione.

4.4.1 Linguaggio di programmazione

Durante la stesura di tale lavoro di tesi abbiamo più di una volta ribadito che sviluppare un'applicazione nativa per ogni piattaforma significa conoscere il linguaggio nativo della piattaforma specificata. Quanto appena detto però provoca un elevato dispendio di risorse sia umane che economiche a differenza invece di quanto accade per sviluppare una web app . Infatti per effettuare lo sviluppo di una web app , quindi eseguibile su più piattaforme, c' è la necessità di conoscere sempre e solo la coppia di linguaggi di programmazione costituita da HTML5 e JavaScript, consentendo così allo sviluppatore una vera e propria libertà di sviluppo cosa che nn si verifica essere possibile per quanto riguarda le applicazioni native che necessitano di un differente e singolare processo di sviluppo (vedi iOS).

4.4.2 Interfaccia Utente

Le applicazioni native hanno un proprio look&feel che viene ereditato da tutte le applicazioni sviluppate per un determinato SO. Tale espressione in lingua inglese "look&feel" viene spesso utilizzata nel mondo informatico per descrivere le caratteristiche di un'interfaccia grafica percepite dall'utente che ne fa uso. Tale percezione da parte dell'utente è intesa sia in termini appunto di apparenza visiva, quindi in termini di look, sia in termini di modalità di interazione, appunto il feel. Pertanto da tale punto di vista è necessario sottolineare che ogni sistema operativo dotato di una propria

interfaccia grafica ha quindi quello che definiamo un proprio look&feel che lo distingue in maniera netta ed evidente da qualsiasi altro SO. In genere tale look&feel viene ereditato dalle applicazioni sviluppate per quel particolare SO, questo particolare aspetto caratteristico favorisce l'usabilità del software poiché l'utente che impara a usare l'interfaccia di una determinata applicazione sarà successivamente in grado di riutilizzare la conoscenza acquisita anche nell'uso di altre applicazioni dotate dello stesso look&feel tipico. In tal modo per quanto riguarda quindi le applicazioni native l'utente avrà sempre una user experience coerente. Al contrario invece le web app sono caratterizzate da un'interfaccia grafica comune per tutte le diverse piattaforme quindi non presentano un caratteristico look&feel che le contraddistigue in relazione alla piattaforma in uso. A tal proposito infatti ho notato questa differenza anche nelle piccole applicazioni sviluppate da me. Inoltre c' è da aggiungere che sono presenti delle librerie quali GPL che rendono le interfacce utente simili a quelle delle piattaforme native. Però l'adattamento dell'interfaccia grafica di un'app per una determinata piattaforma provocherebbe un ulteriore lavoro da parte dello sviluppatore che comunque potrebbe farlo usando per esempio la query media, come abbiamo già in precedenza visto.

4.4.3 Performance

Come ho personalmente riscontrato durante lo sviluppo dei miei piccoli esempi di app , le performance delle web app sono leggermente più basse rispetto a quelle delle applicazioni native. Tale inferiorità si riscontra sia in termini di reattività grafica che di interrogazione dei sensori. Tale differenza è stata riscontrata sebbene le applicazioni native siano state da me testate tramite l'uso del simulatore mentre le web app siano state testate tramite l'utilizzo di un dispositivo Android. Quindi si presuppone che seppure l'app nativa fosse stata anch' essa testata direttamente su un adeguato dispositivo avrebbe sicuramente confermato la differenza appena esposta. Tale osservazione gode anche della conferma presente in letteratura.

4.4.4 Funzionalità e Sensori

Lo sviluppo delle applicazioni native prevede sia l'accesso a tutte le funzionalità native che a tutti i sensori del dispositivo a differenza invece di quel che riguarda le web app che presentato un accesso limitato sia alle funzio-

nalità che, per alcuni framework, anche per i sensori, PhoneGap però nn è tra questi.

4.4.5 Supporto allo sviluppo di applicazioni per device differenti

I sitemi operativi mobile sono sempre in continua evoluzione, questo crea un grosso problema per i framework che presentano una particolare difficoltà a stare a passo con l'evoluzione dei SO mobile. Tale difficoltà deriva dal fatto che l' introduzione di una nuova API è un processo particolarmente lento in quanto dietro di esso c' è un grosso studio ma soprattutto una lunga fase di testing su tutte le piattaforme. Pertanto questo provoca un rallentamento evidente dell'ascesa dei framework.

Capitolo 5

Conclusioni

Prima di intraprendere questo lavoro di tesi ovviamente mi sono prefissato un obiettivo cardine con l'intento di raggiungerlo.Tale obiettivo era quello di capire, attraverso anche lo sviluppo di piccole app, quale approccio tra quello nativo e quello cross-platform fosse il più vantaggioso da utilizzare al fine di sviluppare un'applicazione. Si è cercato di raggiungere tale obiettivo scendendo in dettaglio in quella che è l'organizzazione di iOS in tutte le sue caratteristiche e peculiaritÃ e al contempo dirigendo l'attenzione verso i nuovi approcci e tecnologie web connesse al mondo del mobile, nel mio caso specifico PhoneGap. Interessante quindi era scoprire come mai l'approccio cross-platform, facendo particolare riferimento a PhoneGap, non abbia avuto una dilagante affermazione nonostante richieda un minore quantitativo di risorse sia in termini di impiego di personale addetto che di denaro. Il percorso seguito durante la realizzazione del mio lavoro di tesi può essere suddiviso in vari step. La prima parte del lavoro svolto è stata prevalentemente caratterizzata dalla fase relativa allo studio e apprendimento in dettaglio di quelle che sono le specifiche proprietà caratterizzanti il sistema operativo iOS e il successivo studio del funzionamento del framework PhoneGap e di tutte le sue API. Tale parte iniziale è stata di fondamentale importanza per riuscire ad impostare quelle che sarebbero state le fasi successive. In seguito ho proseguito inoltrandomi in quello che ho definito il confronto tra queste due differenti piattaforme grazie anche all'ausilio di piccole applicazioni sviluppate da me col fine di individuare pro e contro dell'una o dell'altra piattaforma messe a confronto. Le prestazioni rappresentano un indice di valutazione per definire se si tratta di una buona applicazione o meno. A tal proposito, se si parla di prestazioni, è doveroso sottolineare che esse non sono di certo un punto forte delle tecnologie web

poiché, a mio avviso in base alle sperimentazioni effettuate personalmente, sono sicuramente, seppur non di molto, distanti dalle prestazioni che possono essere offerte da un'applicazione nativa. Il look&feel è un altro aspetto caratteristico dello sviluppo di un' app, infatti sviluppare un'app che rispecchi le caratteristiche grafiche della piattaforma specifica è di notevole importanza poiché l'utente ritiene molto più semplice e intuitivo capirne il funzionamento. Inoltre la mancanza di uno Store per le web app, a mio avviso, rappresenta uno dei motivi per i quali tale tecnologia non abbia avuto abbastanza successo. Basti pensare che l'introduzione dell'AppStore da parte di Apple ha radicalmente cambiato il mondo del mobile. Sapendo che comunque quindi le tecnologie web offrono la possibilità di sviluppare applicazioni cross-platform ,allora sorge spontanea una domanda: "Perché scegliere di sviluppare un'app nativa o una web app?" Una risposta secca a tale quesito non è possibile fornirla ma bisogna tenere in considerazione diverse variabili tra le quali ne citiamo alcune:

- Budget
- Piattaforma scelta
- Tipo di app
- Quali funzionalità deve implementare

Se si dovesse scegliere di implementare un'app cross-platform la risposta sarebbe ovvia: web app. Se si volesse creare un gioco, dalle prestazioni e grafica importanti, la scelta al contrario ricadrebbe sullo sviluppo di un'app nativa anche perché per tutto ciò che comporta lo sviluppo di un tale tipo di app, in termini di risorse impiegate ma al contempo di quelle guadagnate, andando ad esempio a fissarle un costo nello Store, andrebbe ad incrementare il guadagno dell'azienda madre. Se si volesse sviluppare un'app per iOs e Android io personalmente opterei per lo sviluppo di applicazione native poiché attualmente Android e iOS hanno i guadagni più elevati rispetto alle altre piattaforme e in più si avrebbe la possibilità di offrire all'utente un prodotto sempre aggiornato con le nuove tecnologie introdotte dai rispettivi produttori senza dimenticare di offrire agli utenti una user experience sempre coerente con le pregresse conoscenze. Per quel che riguarda la mia opinione a riguardo direi che purtroppo le tecnologie web saranno sempre un passo indietro rispetto alla piattaforme native in quanto le novità introdotte dalle varie piattaforme avranno senza dubbio bisogno di più tempo prima di poter essere aggiunte ai vari framework. Però al contempo ritengo che

creare uno Store solo ed esclusivamente per le web apppotrebbe contribuire in modo più che positivo all'affermazione di tali tecnologie anche in termini remunerativi.

Ringraziamenti

Vorrei ringraziare il Professore Alessandro Ricci, relatore, per essere stato sempre presente durante la stesura di tutta la tesi e ringrazio inoltre l'Ingegnere Santi. Ringrazio tutta la mia famiglia in particolare mio padre Giuseppe, mia madre Maria Pia e i miei fratelli Enzo e Mattia per avermi dato la possibilità di raggiungere questo stupedo traguardo e per avermi dato tutto l'appoggio possibile . Ringrazio la mia fidanzata Rossella che mi è stata accanto durante tutto il cammino. Infine rigrazio la mia piccola Lilly che mi ha fatto sempre sorridere anche nei momenti no.

Bibliografia

[1] Adobe. Phonegap documentation, 2013. http://docs.phonegap.com/en/2.9.0/index.html.

[2] A. Charland and B. LeRoux. Mobile Application Development: Web vs. Native. 2011.

[3] D. W. David Mark, Jack Nutting. *Beginning iOS6 Development*. Apress, 2010.

[4] P. GmbH. The-m-project, 2012. http://www.the-m-project.org/.

[5] J. C. . A. HILLEGASS. ios programming - the big nerd ranch guide. Pearson Technology Group, 2011.

[6] A. Inc. ios developer library, 2012. http://developer.apple.com/library/ios/navigation/.

[7] A. Inc. Titanium mobile development environment, 2013. http://www.appcelerator.com/platform/titanium-platform/.

[8] S. Inc. Sencha touch, 2013. http://www.sencha.com/products/touch.

[9] F. R. James A. Landay, Anthony D. Joseph. Smarter Phones. 2009.

[10] T. jQuery Foundation. Jquery mobile, 2013. http://jquerymobile.com/.

[11] S. G. Kochan. *Programming in Objective-C*. Pearson Education, Inc., 2012.

[12] Y. P. Rohit Ghatol. *Beginning PhoneGap: Mobile Web Framework for JavaScript and HTML5*. Apress, 2012.

[13] A. Wright. Get Smart. 2009.

Elenco delle figure