ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Seconda Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

SVILUPPO DI APPLICAZIONI MOBILE PLATFORM-INDEPENDENT: IL LINGUAGGIO E PIATTAFORMA MOBL

Elaborata nel corso di: Sistemi Operativi

Tesi di Laurea di:
ENRICO BENINI

Relatore:
Prof. ALESSANDRO RICCI

ANNO ACCADEMICO 2011–2012
SESSIONE II

PAROLE CHIAVE

Mobl

Platform-Independent

Applicazioni Mobile

Analisi Prestazioni

Desidero innanzitutto ringraziare il mio relatore Dott. Alessandro Ricci per la guida, i consigli e il tempo dedicato alla mia tesi. Inoltre intendo ringraziare tutti i miei compagni di corso che mi hanno aiutato durante il percorso di studi ed anche i miei amici che mi hanno sostenuto moralmente, in particolare Lorenzo Corneo che ha condiviso con me la maggior parte di progetti ed esami. Infine volevo dedicare questa tesi alla mia famiglia, in particolare mia madre Antonella, per il sostegno ed i sacrifici che hanno fatto per me.

Grazie di cuore

Enrico

Indice

Capitolo 1

Introduzione

L'obbiettivo di questa tesi consiste in un analisi dello sviluppo di applicazioni mobile, rivolgendo particolare attenzione riguardo a quelle soluzioni che consentono di astrarre dall'ambiente su cui effettivamente queste verranno eseguite, per poi prendere in esame la piattaforma e il linguaggio Mobl.

Sarà quindi necessario effettuare una panoramica sullo stato attuale del mercato, introducendo i diversi sistemi presenti e le metodologie proposte per la costruzione del software. Da questo studio sarà possibile ricavare i pregi e i difetti di una programmazione nativa.

Volendo, in seguito, ricercare un processo di produzione software che favorisca un'indipendenza dai sistemi precedentemente descritti, verranno illustrati alcuni tra i più importanti Mobile Frameworks e tra questi si prenderà in esame Mobl, che si contraddistingue grazie a caratteristiche peculiari.

In questo breve capitolo introduttivo verrà esposto un quadro generale sugli argomenti che poi verranno discussi maggiormente in dettaglio nel resto del documento.

1.1 Storia e Scenario Mobile

Benché la nascita dei dispositivi mobili per la telefonia risale agli anni settanta, la vera e propria diffusione dei cellulari è avvenuta intorno alla metà degli anni novanta. Questo perché fino ad allora i prezzi di questi apparecchi li rendevano, di fatto, oggetti di lusso. Da allora questo mercato ha

subito una rapida diffusione che ha portato a dotare più della metà della popolazione mondiale di un proprio telefono portatile.

La vera rivoluzione in questo campo però si ottiene con l'introduzione, nel 2008, dell'Iphone, che sancisce la nascita degli smartphone. Questi implementano nuove caratteristiche rispetto ai telefoni precedenti che ne sanciscono il successo e aumentandone l'utilizzo. Alcune tra le più importanti sono: l'introduzione di una fotocamera che consenta di scattare foto e girare video, il touchscreen che consente di utilizzare un diverso tipo di interfacce, cambiando radicalmente l'iterazione con l'utente, la dotazione di un maggior numero di sensori, ed infine, la presenza di processori più prestanti e memorie più capienti. Nonostante tutte queste novità, il punto forte di questi dispositivi rimane la possibilità di installare nuove applicazioni, anche di terze parti, tramite appositi store online, che consentono di ampliare notevolmente le funzionalità dei device. Infatti esistono svariate categorie di applicazioni disponibili, dal classico client e-mail ai giochi, che espandono di fatto le capacità di uno smartphone, ormai sempre meno inferiori a quelle di un PC tradizionale.

Per rendere possibile tutto ciò è necessario dotare questi apparecchi di veri e propri sistemi operativi appositi per la gestione delle risorse. Di conseguenza, nel corso degli anni, sono nate diverse piattaforme con questo scopo, ognuna delle quali, nonostante alcune similitudini, adotta politiche e strategie differenti tra loro. Si è giunti in fretta ad un alto grado di eterogeneità, sia di dispositivi, che di sistemi.

Uno scenario di questo tipo ha portato, e porta tutt'ora, molte aziende ad interessarsi nello sviluppo di applicazioni mobile, ed ognuna di queste ha dovuto investire capitali umani e finanziari per acquisire le conoscenze necessarie e rendere così disponibile il proprio software su questi ecosistemi. Quindi uno dei principali problemi delle imprese consiste: nella scelta di quale piattaforma privilegiare, o se conviene sceglierne più di una e nella scelta degli smartphone su cui il software può essere eseguito.

Di fronte a questa problematica è sempre più sentita la necessità di trovare una soluzione unica, che consenta a sviluppatori indipendenti ed aziende di produrre le proprie applicazioni in maniera completamente svincolata dall'ambiente su cui queste verranno eseguite.

1.2 Soluzioni Platform-independent

Per cercare un modo di superare questo ostacolo, si sono esaminati i punti in comune tra i vari sistemi operativi mobile, per trovare un punto di contatto che consentisse di creare applicazioni platform-independent nel modo più semplice possibile. L'unica cosa che i vari OS mobile hanno in comune è che tutti possono accedere al web browser tramite codice nativo. Utilizzando linguaggi comuni e relativamente semplici per ogni sviluppatore, HTML5, CSS3 e Javascript, è possibile quindi creare applicazioni che si presentino in uguale maniera in tutte le diverse piattaforme su cui verrà distribuita l'applicazione.

Conseguentemente a queste osservazioni sono nati svariati frameworks e librerie che sfruttano proprio le tecnologie web per raggiungere l'obbiettivo. Tali frameworks contengono parti di codice, che permettono l'utilizzo di vari sensori presenti sul device, ad esempio: la fotocamera, l'accelerometro, il GPS e incapsulano l'accesso a questi sensori con librerie Javascript. Sarà quindi sufficiente costruire la propria applicazione web, e richiamare queste librerie apposite per l'utilizzo delle funzionalità del dispositivo. In questo modo si riesce ad astrarre dal sistema su cui l'applicazione così costruita verrà eseguita, perchè saranno i frameworks a gestire le chiamate invocate dal software al sistema stesso.

1.3 Caso di Studio: Mobl

Per esaminare uno di questi framework si è scelto Mobl, in particolare per alcune delle sue caratteristiche che lo differenziano dagli altri middleware. Infatti Mobl non richiede di costruire un'applicazione tramite le classiche tecnologie web, bensì introduce un suo linguaggio di programmazione che ha come scopo quello di unire le potenzialità di tutti i linguaggi sopracitati. Infatti, dal procedimento descritto nella precedente sezione, è necessario utilizzare HTML e CSS per costruire l'interfaccia utente, il Javascript per la parte logica ed eventualmente l'SQL per la gestione di un database. Nel linguaggio Mobl invece vengono definite svariate sintassi per ognuno di questi compiti ed infine, tramite un apposito compilatore, verrà generata l'applicazione web specifica. Inoltre si hanno anche altre particolarità che verranno discusse nel relativo capitolo.

Si ottengono così svariati vantaggi e si risolvono molte problematiche che verranno descritte nel seguito del documento. Nonostante tutto però anche con questa strategia si incorrono in difficoltà ancora irrisolte.

Capitolo 2

Applicazioni Mobile Native

In questo capitolo si vogliono introdurre le varie piattaforme mobile, analizzando i principi e le metodologie utilizzate per progettare costruire applicazioni mobile native. In una prima parte si esploreranno in maniera sintetica i singoli sistemi, prendendo visione delle parti principali che li compongono, mentre in seguito si approfondirà la tematica riguardante lo sviluppo delle applicazioni su ognuno di essi. Sarà così possibile inquadrare le somiglianze e le differenze tra di loro, e trarre significative conclusioni.

2.1 Piattaforme Mobile

Nel capitolo introduttivo abbiamo già discusso le cause della nascita dei diversi ambienti, ora verrà effettuata una panoramica su come questi sono composti.

I sistemi più diffusi al momento sono i seguenti:

- Android
- iOS
- Symbian
- BlackBerry OS
- Bada
- Windows Phone

E' interessante notare che tutte le piattaforme citate sono composte da strati(layers) ognuno dei quali svolge un compito ben specifico e fornisce le proprie funzionalità a quello superiore. Questa somiglianza richiama molto il modello ISO/OSI utilizzato nelle reti di telecomunicazioni per definire l'architettura logica di rete.

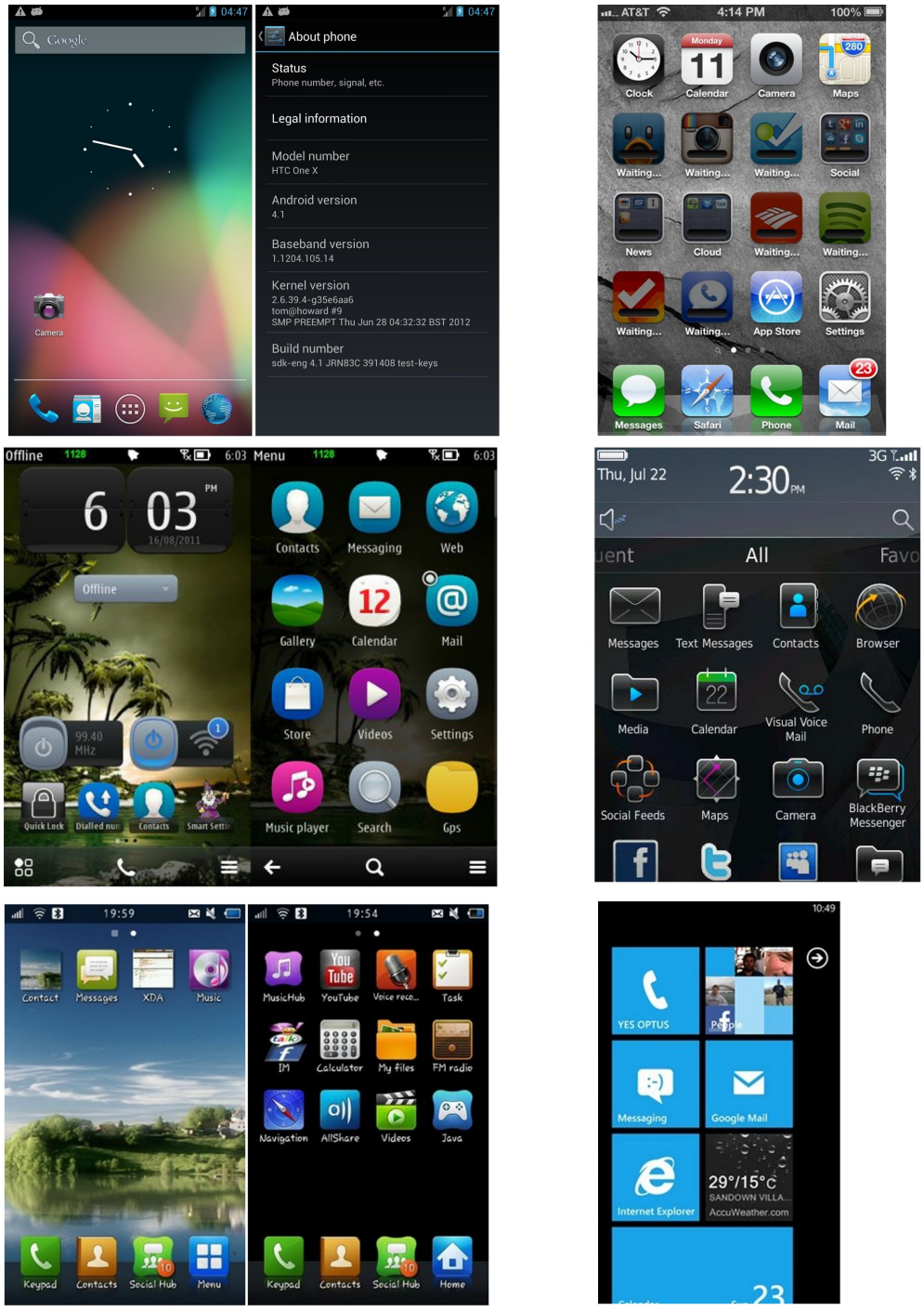


Figura 2.1: Sistemi Mobile

2.1.1 Android

Android è una piattaforma mobile basata su linux e distribuita per smartphones e tablets sviluppata da Open Handset Alliance: un consorzio formato da varie compagnie guidate da Google. Al momento è il sistema più diffuso e si basa su una variante del linguaggio Java, viene strutturato in steak software che comprendono: vari middleware per le comunicazioni, un sistema operativo e le applicazioni di base:

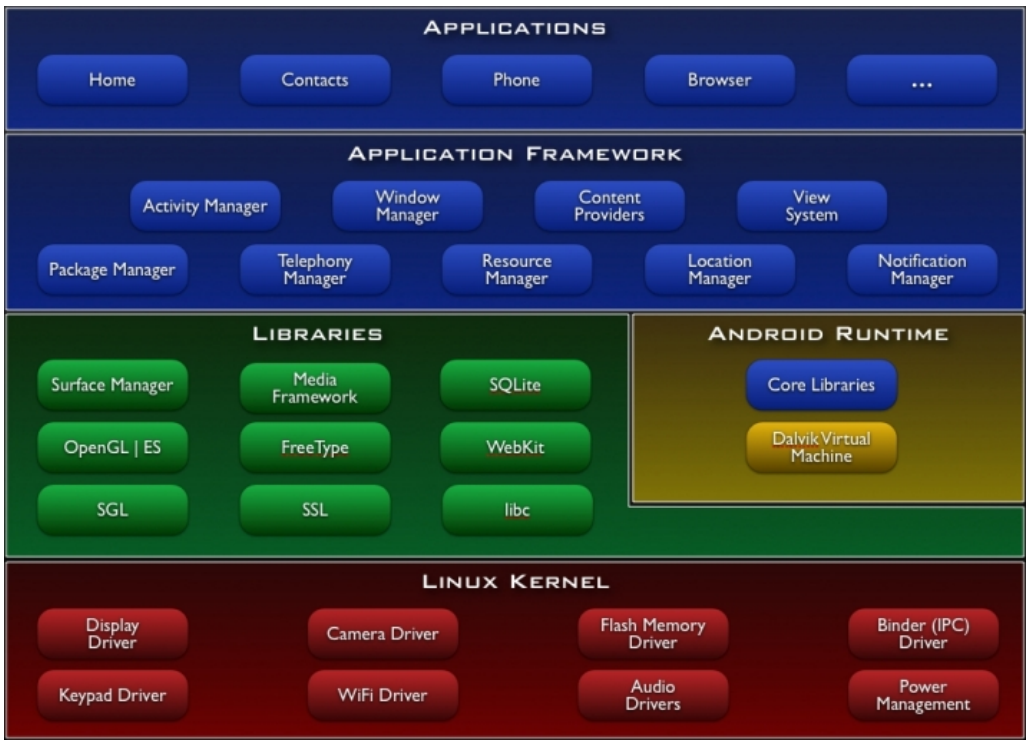


Figura 2.2: Architettura della Piattaforma Android

1. Application framework

   le applicazioni vengono eseguite tramite la Dalvik virtual machine: ottimizzata per utilizzare la poca memoria presente nei dispositivi mobile, possono essere presenti più istanze della macchina virtuale e nasconde al sistema operativo sottostante la gestione della memoria e dei thread.

2. Services frameworks

   In questo livello sono presenti tutti i servizi e le librerie su cui si basano le applicazioni sovrastanti, in particolare viene utilizzato un database SQLite per la memorizzazione dei dati, in questo strato del sistema è presente il framework webkit per la navigazione web e la Dalvik virtual machine su cui vengono eseguite le applicazioni.

3. Linux Kernel

   Infine in questo layer sono presenti tutti i driver, la parte riguardante la gestione della rete e la gestione delle comunicazioni interprocessi.

Android risulta essere in parte open source, solo di alcune versioni il codice sorgente non è ancora disponibile, per sviluppare le applicazioni è presente un SDK (software development kit) che, una volta creata la propria applicazione consente di, consente di creare un file autoinstallante in formato apk per favorirne la diffusione.

2.1.2 iOS

IOS è la piattaforma sviluppata da Apple Inc per i propri dispositivi Iphone, Ipad, e Ipod Touch. Il software è una derivazione di UNIX e utilizza un microkernel basato su Darwin OS, quindi si tratta di software quasi completamente proprietario. IOS ha quattro livelli di astrazione:

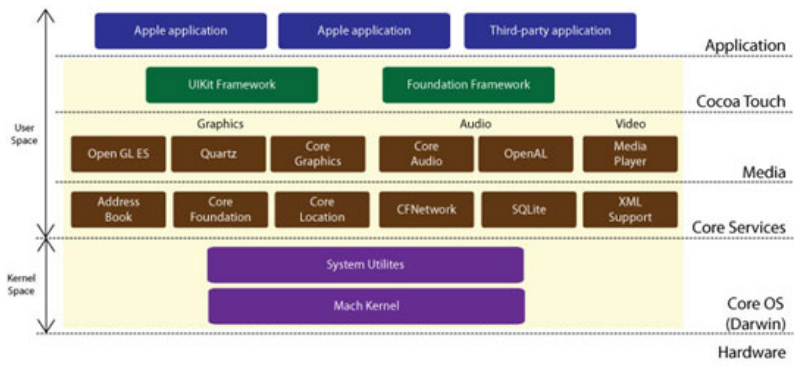


Figura 2.3: Architettura della Piattaforma iOS

1. il Core OS layer

   Si tratta dello strato più basso della struttura a steak tipica di questo sistema, è posto a diretto contatto con l'hardware presente sul dispositivo e si occupa di gestire tutti quei servizi essenziali riguardanti: accesso alla memoria, gestione dei thread, accesso agli accessori esterni e gestione della rete a basso livello.

2. il Core Services layer

   Si occupa di fornire molti servizi basati principalmente sulla gestione delle applicazioni. Alcuni esempi delle funzionalità messe a disposizione da questo layer sono: la gestione dell'objective-C e tutto quello che riguarda la sua esecuzione, la manipolazione di stringhe, XML, URL, la geo-localizzazione, l'accesso alla rubrica e la gestione dei dati di ogni applicazione.

3. il Media layer

   In questo layer invece troviamo tutto quello che riguarda la gestione della multimedialità sul device, cioè la gestione dell'audio/video, comprese le animazioni e tutte le capacità grafiche.

4. il Cocoa Touch layer

   Infine, l'ultimo parte della struttura del sistema operativo, serve per consentire agli sviluppatori di poter creare le proprie applicazioni: infatti utilizza della API standard(Mac Os X Cocoa API) utili per poter accedere a tutti i servizi svolti dai layer inferiori, è scritto anch'esso, come tutte le applicazioni in objective-C.

Tutti le sezioni presentate precedentemente sono poi a loro volta suddivise in altri framework dedicati a specifici compiti all'interno del layer stesso. Uno degli aspetti di questo sistema, che può essere considerato sia un pregio che un difetto, riguarda la produzione della piattaforma e dei dispositivi a cui è dedicata. Infatti entrambi sono prodotti dalla stessa azienda e quindi si può contare su un ben determinato tipo di hardware, potendo così rendere il software maggiormente specifico. Quanto detto però causa al contempo una riduzione della possibilità di scelta da parte del consumatore, che si vede obbligato a scegliere esclusivamente un determinato prodotto.

2.1.3 Symbian

La piattaforma Symbian è la piattaforma mobile successore del precedente Symbian OS e sviluppata dalla Accetture. Anche questo sistema, come il precedente, si struttura in layers. In ordine dal layer superiore a quello inferiore si ha:

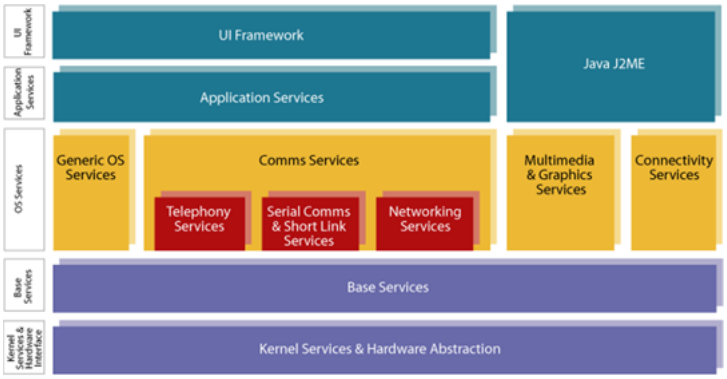


Figura 2.4: Architettura della Piattaforma Symbian

1. UI Framework Layer

   Si trova al di sopra di tutte le altre parti della struttura e si occupa di gestire le interfacce utente, fornisce quindi tutte le librerie, classi, strumenti e frameworks necessari per costruire le interfacce e i suoi componenti. Per quanto riguarda i framework a disposizione, questi sono utilizzati, sia per le interfacce, sia per quanto riguarda le applicazioni.

2. Application Services Layer

   Come dal nome, in questa sezione del sistema sono presenti i servizi utili per costruire le applicazioni indipendentemente dall'interfaccia. I servizi più importanti che si possono trovare in questa porzione del sistema riguardano: quelli generici utilizzati da tutte le applicazioni (Es. Text handling), quelli utili per integrare applicazioni tipiche del sistema(ad esempio il calendario la sincronizzazione con lo store delle applicazioni) ed infine i servizi incentrati sulle tecnologie(Es. Http Transport protocol).

3. OS Services Layer

   Vengono estese le funzionalità delle sezioni sottostanti, ed in particolare si completa la parte vera e propria che costituisce l'SO. Infatti gli stati superiori mettono a disposizione servizi tipicamente orientati alle applicazioni, mentre in questo si vanno a definire quelle funzionalità tipiche di un qualsiasi SO. Alcuni esempi rilevanti sono: il framework multimediale, servizi di networking (TCP/IP) e task scheduler.

4. Base Services Layer

   In questo livello vengono definiti i servizi, nel loro aspetto di più basso livello, in particolare il file server e le librerie utente.

5. Kernel Services & Hardware Interface Layer

   Nel layer di più basso livello risiede il kernel vero e proprio e tutte le interfacce utili per astrarre dall'hardware. In particolare sono presenti tutti i driver utili per poter utilizzare l'hardware del dispositivo. Inoltre viene gestita la memoria e vengono supportati i servizi elencati negli strati superiori.

2.1.4 BlackBerry OS

BlackBerry OS è distribuito e prodotto da RIM appositamente per i propri smartphone BlackBerry. Questa piattaforma è stata una delle prime presenti sul mercato mobile, sopratutto per quanto riguarda il supporto nativo della posta elettronica. E' in grado di supportare il multitasking e consente attraverso le API del sistema di sviluppare applicazioni apposite da parte di entità esterne. Una particolarità che ha da sempre caratterizzato i dispositivi Blackberry, e quindi il Blackberry SO, è rappresentata dalla presenza dei tasti fisici e della trackball. Purtroppo essendo un prodotto proprietario non sono molte le informazioni a disposizione per quanto riguarda la struttura interna del sistema operativo.

2.1.5 Bada

Bada è un sistema operativo distribuito e prodotto da Samsung Electronics.

1. Kernel

   Una delle particolarità di questa piattaforma consiste nel suo kernel configurabile, esso consente di poter scegliere se utilizzare un kernel real-time oppure un kernel linux. Questa caratteristica fa si che il sistema sia molto flessibile e quindi facilmente adattabile a vari dispositivi oltre a smartphones e tablets, ma anche, ad esempio le smart-tv.

2. Device Layer

   Contiene le funzioni base di tutto il sistema: grafica, sicurezza, multimedia, comunicazione e telefonia.

3. Service Layer

   Contiene tutte quelle funzioni che vengono rese disponibili dalle applicazioni. Queste funzionalità poi vengono richiamate dalle API presenti nello strato superiore. Un esempio di queste funzionalità può essere il servizio SMS e della gestione dei contatti.

4. Framework

   Si tratta dello strato superiore del sistema che contiene vari framework, scritti in linguaggio C++, utili per mettere a disposizione delle applicazioni e dell'utente tutti i servizi precedentemente menzionati. Inoltre, in questo strato del sistema, è presente tutta la gestione del "ciclo di vita" delle applicazioni e la gestione degli eventi.

2.1.6 Windows Phone

Windows Phone è un sistema operativo mobile proprietario sviluppato e distribuito da Microsoft. Supporta vari dispositivi e richiede delle specifiche tecniche minime per essere messo in esecuzione. Anche questo sistema è strutturato a strati come i precedenti:

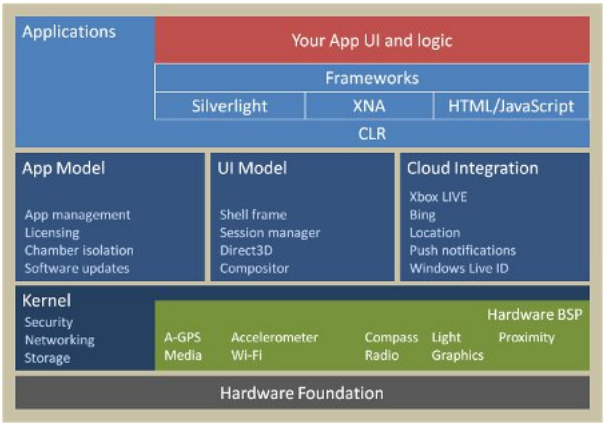


Figura 2.5: Architettura della Piattaforma Windows Phone

1. Kernel

   La parte a stretto contatto con l'hardware del dispositivo, si incarica di gestire la sicurezza del sistema, la sua interazione con la rete e la gestione dello storage dei file, inoltre vi sono presenti tutti i driver utili per poter rendere portabile il sistema su diverse tipologie di dispositivi.

2. Middleware Layer

   Si suddivide a sua volta in 3 parti differenti:

   - Modello delle applicazioni: si occupa di gestire ogni software in esecuzione sul sistema, gestendone gli aggiornamenti e assicurandone l'isolamento dal sistema, in modo che eventuali guasti di queste non incidano sulla stabilità del sistema stesso.
   - Modello delle interfacce: racchiude tutti gli strumenti per la costruzione della grafica del sistema, come le animazioni e il rendering 3d.
   - Integrazione cloud: ha come compito quello di gestire tutte le applicazioni che richiedono uno scambio di dati costante con il web, ad esempio:le nofiche, Xbox live e il Windows Live ID.

3. Applications

Lo strato superiore del sistema che contiene le applicazioni del sistema e tutti i framework a cui essi si appoggiano per la loro corretta esecuzione.

2.2 Sviluppo di Applicazioni Native

Ora prenderemo in esame lo sviluppo di applicazioni su queste piattaforme, facendo attenzione alle differenze e alle difficoltà che i programmatori devono affrontare per progettare e creare il proprio software. In particolare verranno visualizzate le procedure e i pattern maggiormente consigliati.

Vista la grande eterogeneità dei dispositivi e delle piattaforme nello scenario mobile, anche la metodologia per la creazione di applicazioni in questo ambito risulterà essere molto variegato. In particolare l'approccio utilizzato per progettarle e costruirle saranno fortemente dipendenti dal sistema su cui queste dovranno essere messe in esecuzione, quindi l'ambiente di sviluppo dovrà essere considerato fin dalle prime fasi di analisi, per poi non ritrovarsi con problemi legati alla tecnologia utilizzata, che non si sposa con la struttura pensata per la propria applicazione. Quindi, come già visto nella sezione precedente, sarà necessario differenziare il metodo di progettazione in base al tipo di piattaforma.

Nonostante siano presenti molte differenze tra queste, esistono vari aspetti comuni che riguardano la programmazione mobile sopratutto sulle linee guida generali che ogni progettista software deve considerare. Alcuni esempi di questi importanti principi sono:

1. Struttura del progetto in base agli obbiettivi

   Si tratta della prima considerazione che un progettista deve sostenere. Bisogna sapere, con chiarezza e fin dal principio, il tipo di applicazione che si andrà a costruire e quindi capire se si tratterà di un software che necessita di un uso abbondante di risorse o meno. Tutto questo servirà per stabilire fin da subito se si tratterà di un software leggero e agile oppure più completo.

2. Azioni ed utilità dell'applicazione

   Una delle prime preoccupazioni di uno sviluppatore consiste nell'inquadrare fin da subito tutte le azioni che il proprio software dovrà

compiere, senza necessariamente scendere nei dettagli di come verranno implementate e perchè questo risulta effettivamente utile. Un suggerimento può essere stillare una lista delle operazioni da compiere e degli obbiettivi da raggiungere

3. Dispositivi supportati

   E' importante sapere fin da subito quali dispositivi si ha intenzione di supportare. Se per esempio si deciderà di creare un applicazione in grado di essere eseguita su vari dispositivi, bisognerà procedere nella maniera più generale possibile, tenendo conto anche delle differenze hardware tra questi, mentre se ci si concentra solo su una certa fascia di apparecchi, allora è possibile strutturare il software in maniera più peculiare.

4. Stato della Rete

   Bisogna considerare il comportamento dell'applicazione anche quando non è presente la rete oppure se questa funziona ad intermittenza.

5. Separazione delle parti

   Cercare di mantenere sempre ed il più possibile separate la parte logica del software dalle restanti, in modo da facilitarne le modifiche.

6. Progettazione interfacce

   Quando si ideano le interfacce e si implementano le ottimizzazioni dell'applicazione bisogna considerare l'ambiente in cui il software verrà eseguito e la capacità del dispositivo stesso, tenendo sempre presente la possibilità di poter ideare una successiva versione desktop e quindi cercando di mantenere inalterate le viste.

Prendendo sempre come esempio i sistemi precedentemente introdotti, procediamo con l'analisi dello sviluppo software per alcuni di essi.

2.2.1 Applicazione Android

Le applicazioni Android si basano sulla collaudata tecnologia Java e quindi ne ereditano tutte le proprietà e caratteristiche che hanno reso molto popolare questo linguaggio. Java è orientato agli oggetti e possiede molte librerie utili per poter utilizzare servizi in modo rapido e testato, si avvale

di un'ampia documentazione consultabile online, un pieno supporto nello sviluppo grazie a diversi IDE e una comunità molto attiva sul web. Un aspetto particolare di questo linguaggio open source riguarda la sua portabilità, infatti si avvale di una virtual machine per essere interpretato e eseguito su un qualsiasi dispositivo.

Proprio grazie a queste caratteristiche, il linguaggio Java si presta benissimo all'esecuzione su dispositivi mobile, Infatti, gli sviluppatori di Android hanno potuto costruire il sistema avvalendosi di quello che è già presente in questa tecnologia. Non a caso nella struttura Android è presente la Dalvik virtual machine che è simile a quella Java.

2.2.1.1 Esecuzione di Applicazioni Android

Prima di addentrarci su come è strutturata una applicazione Android è utile esplicitare rapidamente come queste vengono eseguite e come il sistema le organizza.

Gli strumenti racchiusi nel SDK Android (software development kit) consentono di, una volta creata la nostra applicazione, di compilarla e generare un Android package (.apk) che può essere letto da tutte le versioni del sistema Android, e quindi installato.

Vediamo quali sono le caratteristiche, dal punto di vista delle relazioni con il sistema operativo, di un'applicazione Android:

- Ogni esecuzione dell'applicazione genera un singolo processo Linux il quale viene terminato da Android non appena l'applicazione viene chiusa.

- Ogni processo possiede una propria Virtual Machine (necessaria per l'esecuzione di un file java) il quale ha il compito di gestire il processo. Ciò implica che ogni processo viene eseguito da Android in maniera autonoma, isolata dal resto delle altre operazioni e dalle routine di sistema.

- Ad ogni applicazione viene associata un User ID univoco. Questa caratteristica permette ai file delle applicazioni di essere visibili a tutte le altre applicazioni che hanno lo stesso User ID e all'applicazione stessa.

E' possibile identificare diversi componenti in una applicazione Android: le actvities, i services, i broadcast receivers, i content providers, le views e gli intent. Che tratteremo nel prossimo capitolo

2.2.1.2 Componenti di un applicazione Android

Sono i blocchi fondamentali dell'applicativo e definiscono punti di accesso al programma: non tutti riguardano direttamente l'iterazione con l'utente, ma rimangono comunque fondamentali per specificare il comportamento e l'iterazione con il resto del sistema. Ci sono quattro tipi di Componenti:

- Activity

  Un'attività rappresenta una singola schermata, ce ne possono essere varie e tutte insieme coesistono per creare la struttura del software. In base a come è costruita l'applicazione possono essere accessibili direttamente da altre applicazioni, oppure come semplici conseguenze di determinate azioni all'interno di altre attività del programma. Dato che in un'applicazione possono coesistere più Activity, che sono delegate ad uno specifico compito, risulta evidente che ogni Activity è indipendente dalle altre. E' necessario però identificare una sorta di gerarchia tra le Activities, e nello specifico identificare una di queste come primaria, in modo che all'esecuzione sia distinguibile quale deve effettivamente essere lanciata per prima.

  Le activities possono incorrere in vari stati durante il loro ciclo di vita, e per ogniuno di essi è possibile specificarne il comportamento. Le fasi di questo percorso sono ben visibili nella seguente immagine:

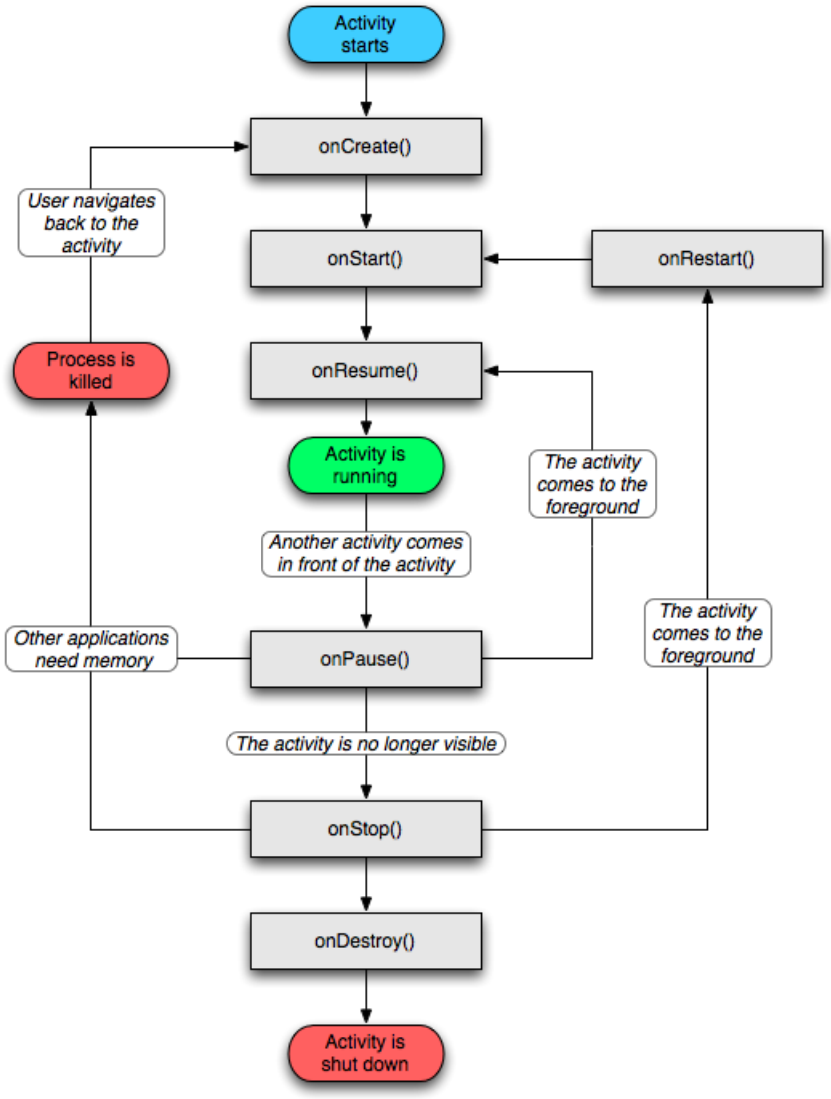


Figura 2.6: Ciclo di Vita di una Generica Activity

Prima di tutto identifichiamo i vari stati nei quali un'activity si può trovare una volta istanziata:

- Attivo: in questo stato l'activity viene mostrata in primo piano sullo schermo e l'utente può interagire con l'interfaccia grafica mostrata.
- In pausa: l'activity viene comunque mostrata a schermo, ma l'utente non può interagire con essa perchè un'altra activity è stata sovrapposta alla primaria. Un classico esempio è quando viene ricevuta una chiamata dove l'activity relativa alla chiamata viene sovrapposta a quella dell'applicazione che l'utente stava utilizzando.
- Stoppata: si ricade in questo stato quando l'activity viene completamente sovrapposta da un'altra e dunque non è più visibile all'utente. Di fatto l'activity iniziale è nascosta all'utente e nel caso in cui la memoria stia per esaurirsi Android chiuderà l'activity nascosta.

Elencati gli stati in cui si può trovare un'activity è opportuno presentare al lettore le transizioni, sotto forma di metodi, che permettano il passaggio da uno stato ad un altro:

- void onCreate()
- void onStart()
- void onRestart()
- void onResume()
- void onPause()
- void onStop()
- void onDestroy()

- Servizi

  Un servizio consiste in un processo che viene eseguito in background, quindi non visibile da parte dell'utente, e che molto spesso rimane in esecuzione per lungo tempo, ad esempio un processo che controlla

l'arrivo di nuove mail. Non posseggono un'interfaccia e possono interagire o essere eseguite da un altro componente, come un attività. Una particolarità molto importante di un servizio è la capacità di non interrompere la normale iterazione tra l'utente e il sistema.

- Content Providers

  Si occupa di gestire i dati immagazzinati nel device, su file system o in un database SQLlite. I dati gestiti da un Content providers possono essere condivisi, quindi accessibili e modificabili da diverse applicazioni. Infatti possono essere utili proprio per passare dei dati tra diverse applicazioni, o ad esempio solo per memorizzare dati privati utili ad una applicazione. Un esempio di questo componente può essere la memorizzazione dei contatti. Con questi componenti vengono gestiti anche dati che risiedono nel web.

- Broadcast Receivers

  I Broadcast sono semplicemente dei messaggi, che possono essere inviati dal sistema o da processi in esecuzione, che consentono di notificare un particolare avvenimento. Non posseggono una interfaccia grafica, ma possono modificare gli elementi presenti nella status bar del sistema. Inoltre sono in grado di lanciare dei servizi specifici quando un particolare evento si verifica.

- Views

  il cuore di questi controlli è rappresentato principalmente da due classi, *android.view.View* e *android.view.ViewGroup*.

  Una View è sostanzialmente un'area rettangolare nello schermo responsabile del disegno degli elementi grafici e della cattura e gestione degli eventi generati dall'utente. Un oggetto ViewGroup è anch'esso una View, ma che contiene altre View.

  Nella piattaforma Android si utilizza il concetto di "layout" per gestire l'impaginazione dei controlli all'interno di un contenitore di oggetti View. Utilizzare i layout facilita la progettazione delle interfacce perché è più semplice orientare e posizionare i controlli che le compongono.

  Possiamo considerare una View come un widget: bottoni, etichette, caselle di testo, tabelle, sono tutti esempi di controlli e widget che

possiamo utilizzare per comporre le interfacce grafiche delle nostre applicazioni.

- Intent

  Un Intent è un meccanismo che descrive un'azione specifica, come ad esempio "chiamare a casa" o "inviare un sms": in Android praticamente ogni cosa passa attraverso un Intent, e lo sviluppatore li può utilizzare per riusare o sostituire diversi componenti software.

  Ad esempio è disponibile un Intent "inviare una email": se nella nostra applicazione abbiamo la necessità di gestire l'invio di email possiamo invocare l'Intent che il sistema ci mette a disposizione, oppure possiamo scriverne uno nuovo e utilizzare quest'ultimo nelle nostre Activity sostituendo l'Intent di default con quello implementato da noi.

  Possiamo immaginare un Intent come un'azione che possiamo far invocare ad Android: un Intent può essere molto specifico ed essere richiamabile da una specifica Activity oppure può essere generico e disponibile per qualsiasi Activity che rispetta determinati criteri.

  Schematizzando, un Intent può essere utilizzato per:

  - Trasmettere l'informazione per cui un particolare evento (o azione) si è verificato
  - Esplicitare un'intenzione che una Activity o un Service possono sfruttare per eseguire un determinato compito o azione, solitamente con o su un particolare insieme di dati
  - Lanciare una particolare Activity o Service
  - Supportare l'interazione tra qualsiasi applicazione installata sul dispositivo Android, senza doversi preoccupare di che tipo di applicazione sia o di quale componente software gli Intent facciano parte.

  L'Intent è uno dei concetti e dei meccanismi fondamentali di Android: la sua implementazione trasforma ogni dispositivo Android in una collezione di componenti indipendenti ma facenti parte di un singolo sistema interconnesso.

Prima di concludere questa parte, è necessario citare ancora 2 elementi importanti della struttura del sistema android: il file Manifest, e la gestione delle risorse. Questi 2 aspetti vengono gestiti in maniera separata proprio perchè non sono componenti rilevanti per l'utilizzatore del sistema, ma sono fondamentali per la costruzione di ogni applicazione.

Il Manifest file è quel componente che serve al sistema per conoscere la struttura e le informazioni di contesto della nostra applicazione, prima di lanciare infatti il software il sistema legge questo file .xml per capire: i permessi che un applicazione richiede, ad esempio la necessità di accedere ad internet o quella di poter leggere i contatti memorizzati sul telefono, dichiarare un livello minimo di API dell'applicazione o rendere noto al sistema le caratteristiche hardware e software utili per l'esecuzione dell'applicativo. Il Manifest file ha anche altre funzioni molto importanti oltre a quelle sopra citate, che però non vengono discusse nel dettaglio in questa sede.

Per quando riguarda la gestione delle risorse specifiche per un certo programma, queste vengono organizzate in cartelle separate ed inseguito associate ad un identificativo univoco. Tutto ciò è indispensabile per poter richiamare le risorse e utilizzarle all'interno di interfacce grafiche o per altri scopi. Un aspetto notevole de sistema riguarda come effettivamente vengano separate la parte di codice dal resto delle utility software: infatti la parte logica e i suoi sorgenti risiedono in una ben specifica porzione del progetto, mentre librerie e risorse in altre due posizioni separate.

Infine ci sono delle particolarità di questo sistema che sono interessanti ed è giusto analizzare. Ad esempio è importante specificare che un componente in esecuzione è in grado di richiamare un altro componente in seguito ad un determinato evento, questo verrà eseguito su un processo separato, ma per l'utente sembrerà esattamente che questo componente faccia effettivamente parte dell'applicazione iniziale. Un esempio classico si ha quando un utente decide di condividere un particolare link: gli vengono proposte le varie applicazioni che consentono di condividere link, una volta scelta su quale di queste effettuare la condivisione, il software viene aperto e, finita l'operazione, si potrà tornare al programma di partenza, esattamente come se il nuovo processo, fosse integrato in quello che lo ha richiamato. Ovviamente per fare questo bisogna avere i permessi per farlo, quindi è necessario che il componente richiamato consenta questo tipo di operazione.

Un altro aspetto interessante è l'assenza di un vero e proprio entry point per un applicazione (no main() function), infatti con la gestione a processi

separati è il sistema che, una volta richiesto l'esecuzione di un determinato software, si occupa di creare il processo apposito, se non è già in esecuzione, e di stanziare tutte le classi che servono.

2.2.1.3 Costruzione Tipica di un Applicazione Android

Proprio grazie alla struttura sopra descritta e alle molteplici somiglianze con il noto linguaggio Java, Android consente di utilizzare a pieno, tutte le tipiche politiche esistenti nei più diffusi linguaggi ad oggetti. A causa di questa grande varietà di scelta, messa a disposizione del programmatore, non viene effettivamente specificata un progetto di base e consigliato per le applicazioni di questo sistema, ma spetta al progettista esperto effettuare le dovute scelte, considerando le possibilità che l'ambiente mobile offre rispetto ad un tipico ambiente desktop.

Tuttavia vengono specificati dei design pattern riguardanti le varie possibilità di input, da parte dell'utilizzatore, nel sistema Android, in modo da redendere il progettista a conoscenza delle varie possiblità presenti e sapere come queste devono essere gestite ed implementate.

In particolare, derivando da un linguaggio ad oggetti come il java, la struttura del sistema e l'organizzazione delle sue applicazioni cosentono di utilizzare il popolare pattern architetturale MVC che verrà introdotto nel quarto capitolo.

2.2.2 Applicazione iOS

Per quanto riguarda le applicazioni iOS invece la Apple inc. fornisce una serie di linee guida utili per sviluppare la propria applicazione sopra questo sistema. In seguito verranno riassunte, tralasciando le specifiche del sistema e concentrandosi in particolar modo sui passaggi fondamentali per la costruzione del software. Prima di esporre questi passaggi però è giusto visionare l'ambiente in cui verranno eseguite queste applicazioni per comprenderne meglio il loro funzionamento.

2.2.2.1 Introduzione all'ambiente iOS

Di seguito verranno elencati e introdotti alcuni fondamentali aspetti dell'ambiente di sviluppo iOS, in particolare verranno visualizzate le caratte-

ristiche del sistema, e gli accorgimenti che uno sviluppatore deve adottare, che maggiormente condizionano l'esecuzione delle applicazioni:

- Virtual Memory System

  Ogni programma ha un proprio indirizzo di memoria virtuale dedicatogli, ma a differenza di un sistema desktop, la quantità di memoria che il processo ha a disposizione è strettamente correlato alla quantità fisica di memoria disponibile nel sistema, questo perchè non è effettivamente possibile utilizzare tecniche come il paging e lo swapping su disco fisso. In caso di poca memoria rimanente il sistema invia una notifica ai processi in esecuzione chiedendo di liberare memoria in modo da risolvere il problema. Esistono diversi modi per verificare se si è in uno stato di low-memory del dispositivo: uno di questi consiste nell'eseguire un override di un metodo, spiegato in seguito tramite il pattern delegation, oppure tramite una registrazione per ricevere la notifica (*IUApplicationDidReveiveMemoryWarningNotification*).

- Automatic Sleep Timer

  Il sistema dispone di un timer apposito per conservare la batteria del sistema che, dopo un certo periodo di tempo senza input touch, provvede a spegnere il display. E' importante per le applicazioni considerare questo componente, soprattutto se si tratta di software che non richiedono input touch, ma necessitano comunque di mantenere il display acceso. Un possibile esempio di questa situazione comprende tutti i giochi che utilizzano solamente l'accelleromentro, e che quindi devono disabilitare questo timer.

  Per disattivare l'oscuramento del monitor bisogna impostare la proprietà *idleTimerDisablel* nell'oggetto condiviso *UIApplication* al valore *YES*.

- Multitasking Support

  A partire da IOS 4 la applicazioni possono essere ridotte in background, generando così multitasking. Quindi è giusto considerare nel proprio progetto questo particolare, in particolare se l'applicazione non deve effettuare, se ridotta, altre operazioni, quindi deve entrare in sospensione. Inoltre visto che la memoria è limitata, il sistema elimina le applicazioni non usate di recente, il tutto può accadere in

qualsiasi momento e senza preavviso. Diventa quindi fondamentale salvare lo stato dell'applicazione e i dati dell'utente quando questa viene messa in secondo piano. Quindi durante il rilancio l'applicazione dovrà utilizzare i dati salvati per ripristinare l'applicazione allo stato precedente, questo fa sembrare all'utente che l'applicazione non si sia mai chiusa.

E' obbligatorio rispondere adeguatamente ai cambi di stato che si verificano durante l'esecuzione in multitasking, gli stati in cui un'applicazione si può trovare sono:

- Not Running: l'applicazione non è stata lanciata o era in esecuzione ma è stata chiusa dal sistema
- Inactive: l'applicazione è in esecuzione in prima piano ma non riceve eventi. Solitamente l'applicazione rimane in questo stato solo per pochi istanti come transizione ad un diverso stato, l'unico caso in cui l'applicazione rimane a lungo in questo stato è quando il sistema chiede all'utente di rispondere ad un qualche evento come una chiamata o un SMS, oppure quando l'utente blocca lo schermo
- Active: l'applicazione è in esecuzione in primo piano e riceve eventi.
- Background: l'applicazione è in secondo piano e esegue codice. La maggior parte delle applicazioni entrano in questo stato per breve tempo o stanno per essere sospese. Applicazioni possono richiedere tempo di esecuzione extra. Se questo stato non è disponibile, le applicazioni passano automaticamente allo stato di *not running*.
- Suspended: l'applicazione è in background ma non esegue codice. In condizioni di poca memoria il sistema può, senza alcun preavviso, chiudere le applicazioni in questo stato. Se questo stato non è disponibile le applicazioni passano automaticamente allo stato di *not running*.

E' obbligatorio seguire le linee guida per il comportamento delle applicazioni quando vengono messe in secondo piano:

  - Non fare chiamate a Open-GL ES, cioè eseguire comandi di disegno di qualsiasi tipo, pena la terminazione immediata dell'applicazione
  - Chiudere tutti i servizi aperti prima della sospensione. Se questo non viene fatto dallo sviluppatore viene eseguito dal sistema quando si ha la sopensione.
  - Salvare lo stato prima di passare in secondo piano, come precedentemente accennato
  - Rilasciare la memoria non necessaria
  - Evitare di aggiornare le finestre, anche se questa operazione potrebbe essere fatta al ritorno in foreground.
  - Registrarsi per notifiche di collegamento o scollegamento di accessori esterni
  - Rimuovere informazioni sensibili dalla vista. Quando un'applicazione passa in secondo piano il sistema fa un'istantanea della finestra principale, tale immagine viene utilizzata brevemente quando l'applicazione torna in primo piano
  - Fare un lavoro minimo in background (applicazioni che eseguono troppe istruzioni in background possono essere chiuse automaticamente dal sistema e senza preavviso).

Un compito fondamentale di iOS è quello di garantire la sicurezza del dispositivo e l'esecuzione delle applicazioni su di esso. A tal fine vengono implementate diverse funzionalità per correggere l'integrità dei dati dell'utente e per garantire che le applicazioni non interferiscano tra di loro. Le strategie utilizzate dal sistema sono le seguenti:

- Sandbox

  La piattaforma, per incrementare la sicurezza del sistema, installa ogni applicazione in una particolare posizione del file system, nella quale si sviluppa tutta la struttura dell'applicazione e le risorse a cui questa può accedere. Sandbox è un insieme di controlli che limitano l'accesso di un'applicazione a file, preferenze, risorse di rete, hardware e così via. Ogni applicazione ha accesso al contenuto della propria sandbox, ma non è possibile accedere a quelle di altre applicazioni. Si

ha inoltre un particolare controllo anche alle risorse del sistema a cui, dalla sandbox, i programmi possono accedere.

- Keychain Data

  Keychain Data è un contenitore criptato di password, può essere utile per conservare un piccolo ammontare di dati sensibili del software, non per fungere da vero e proprio gestore di dati. I dati del keychain sono memorizzati all'esterno della sandbox. Quando viene eseguito il backup dei dati da iTunes anche i dati presenti nel portachiavi vengono salvati.

- Protezione dei File

  Da iOS 4 è stato reso possibile alle applicazioni di utilizzare una funzionalità di protezione, per crittografare i file e renderli inaccessibili quando il dispositivo è bloccato. Quando il dispositivo è bloccato, neanche l'applicazione proprietaria può accedere a tali dati, l'utente deve effettuare lo sblocco del dispositivo tramite il codice di accesso.

2.2.2.2 Procedura Operativa Consigliata

Prima di cominciare ad implementare conviene definire con chiarezza i requisiti e le caratteristiche del programma, in questo modo saranno più chiari gli obbiettivi e le attese da parte del progetto. In seguito conviene esplorare le caratteristiche del sistema iOS, capendo così se sono presenti funzioni in grado di facilitare gli obbiettivi. E' consigliabile inoltre porre particolare attenzione all'interfaccia grafica che si intende utilizzare considerando le strategie che il sistema consiglia per renderla operativa.

Ogni applicazione iOS è costruita utilizzando il framework UIKit e hanno essenzialmente la stessa architettura di base. UIKit fornisce tutti gli oggetti chiave necessari per eseguire l'applicazione, per coordinare e gestire l'input e per visualizzare il contenuto sullo schermo.

Per aiutare i progettisti software vengono proposti vari design pattern, molti dei quali presenti anche su Mac OS X, qui di seguito ne verranno elencati i principali:

- Model-View-Controller

  Classico pattern architetturale che verrà definito nel capitolo 4

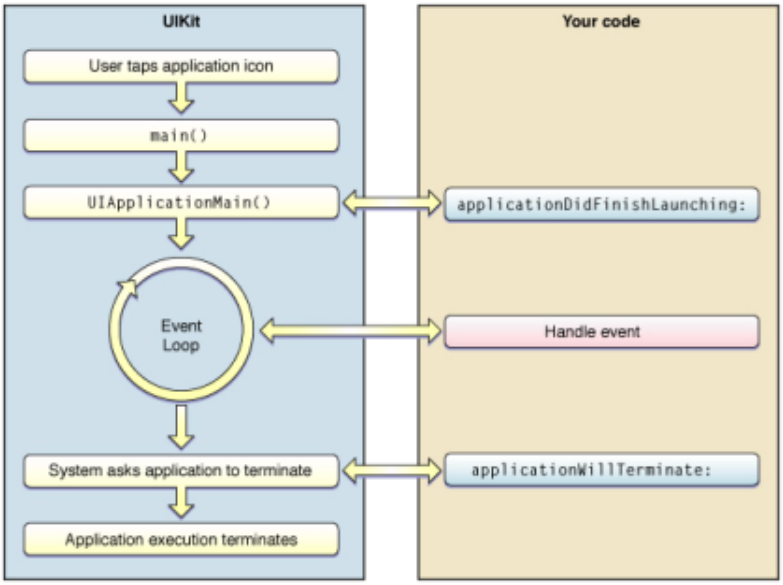


Figura 2.7: Ciclo di Vita di un Applicazione iOS

- Delegation

  Porta un oggetto a inviare notifiche ad un'altro oggetto delegandolo ad operare in un determinato modo. In particolare per applicazioni Iphone dovremo fare i conti con i delegate in 2 distinte occasini:

  – Ricevere messaggi dal sistema operativo
  – Ricevere messaggi dagli elementi della nostra applicazione.

  Analizzando il primo caso e il ciclo di vita della nostra applicazione siamo in grado di individuare svariati messaggi scambiati dal sistema con la nostra applicazione e questi vengono raggruppati all'interno della classe *¡myApp¿AppDelegate*. Di seguito possiamo vedere un elenco di questi messaggi e il loro significato. Bisogna sottolineare che l'utilizzo di questo pattern è prettamente facoltativo ma altamente consigliato, soprattutto per la gestione dei primi 3 messaggi.

  – applicationDidFInishLaunching: Il sistema operativo notifica alla nostra applicazione che la fase di caricamento è terminata.

Possiamo usare questo metodo per inizializzare la nostra applicazione con i valori di un'esecuzione precedente o con valori di default.

- applicationWillTerminate: Il sistema operativo notifica alla nostra applicazione che l'utente o il sistema stesso ha richiesto la chiusura della nostra applicazione. Possiamo usare questo metodo per salvare lo status della nostra applicazione.
- applicationDidReceiveMemoryWarning: Il sistema ha richiesto della memoria, è compito nostro liberarla. Nonostante ci siano altri metodi per liberare memoria usare questo metodo potrebbe essere la scelta giusta se decidiamo di implementare il protocollo UIApplicationDelegate.
- applicationSignificantTimeChange: Il sistema notifica un cambio dell'ora significativo, per esempio l'arrivo della mezzanotte (cambio del giorno) o cambio dell'ora legale (cambio dell'ora).
- applicationWillResignActive: Il sistema notifica alla nostra applicazione che sta per diventare inattiva.
- applicationDidBecomeActive: Il sistema notifica alla nostra applicazione che è tornata attiva
- application:HandleOpenUrl: Il sistema notifica alla nostra applicazione che è arrivata una richiesta da un'url.
- application:willChangeStatusBarOrientation:duration: Il sistema notifica che l'orientamento dell'interfaccia della status bar sta per cambiare
- application:didChangeStatusBarOrientation: Il sistema notifica che l'orientamento dell'interfaccia della status bar è cambiato
- application:willChangeStatusBarFrame: Il sistema notifica che l'orientamento del frame della status bar sta per cambiare
- application:didChangeStatusBarFrame: Il sistema notifica che l'orientamento del frame della status è cambiato

- Target-action

  L'interfaccia utente di una tipica applicazione consiste in un determinato numero di oggetti grafici, appartenenti al campo di presentazione

dell'informazione; la controparte, oggetti di controllo, hanno la funzione di interpretare l'intenzione dell'utente che interagisce con la GUI, ed istruire qualche altro oggetto per soddisfare la richiesta.

Quando avviene un'interazione con l'utente, i componenti hardware del dispositivo generano eventi, accettati dalla parte di controllo che li incapsulano in apposite strutture per l'ambiente Cocoa, traducendo il tutto in un'istruzione che risulta specifica per il contesto della singola applicazione. Purtroppo, il solo concetto di Evento non porta con sé un quantitativo di informazioni sufficienti circa l'intenzione dell'utente; infatti essi riportano solamente l'istante e il tipo di comportamento avvenuto.

Risulta necessario un meccanismo ideato al di sopra di tutto ciò che provveda alla traduzione del concetto di evento ed istruzione, chiamato Target–Action. Lo stesso meccanismo viene utilizzato in ambiente Cocoa in forme differenti a seconda della piattaforma utilizzata (Mac OS X o iOS); ad ogni modo risulta utile per la comunicazione fra uno o più elementi di controllo ed un qualunque altro oggetto, in modo tale che i primi mantengano le informazioni necessarie per inviare un messaggio al secondo, al verificarsi di un generico evento. Tali informazioni sono rappresentate da due elementi principali:

- l'azione di selezione, che identifica il metodo da invocare;
- il target, l'oggetto ricevente.

L'evento che innesca l'azione può‘ essere di qualunque tipo, anche se il meccanismo target-action è molto spesso usato in relazione ad oggetti di controllo come bottoni o sliders. Per quanto riguarda l'oggetto di controllo, il framework UIKit ha dichiarato ed implementato diverse classi di controllo, tutte ereditanti da UIControl, che definisce molti meccanismi target-action specifici per iOS.

- Block Object Sono un modo pratico per incapsulare codire. Tali oggetti sono disponibili a partire da iOS 4, spesso sono utilizzati per callback in attività asincrone.

- Menaged Memory Model: Objective-C utilizza un sistema reference-counted per determinare quando liberare memoria dagli oggetti. Quan-

do un oggetto viene creato il contatore viene impostato a uno. Quando il questo raggiunge lo zero, viene richiamato il runtime di pulizia dell'oggetto che lo rilascia.

- Programmazione concorrente e threads: Tutte le versioni di iOS supportano la creazione di operazioni che utilizzano Threads secondari, in particolare a partire da iOS 4 le applicazioni possono utilizzare il Grand General Dispatch, che non verrà qui discusso.

Un'ulteriore aspetto da considerare è la gestione della memoria, infatti il sistema non gestisce autonomamente i processi, ma delega allo sviluppatore la gestione di questi. Quindi è importante fare attenzione a consumare meno memoria possibile, volendo si possono utilizzare strumenti esterni che sono in grado di gestire la memoria in maniera efficiente.

Vengono consigliati anche determinati approcci per la costruzione del modello dei dati e dell'interfaccia grafica. In particolare la gestione di questi due importanti aspetti dell'applicazione dipendono fortemente dalla complessità dell'applicazione stessa.

Prendendo in esame la modellazione dei dati è possibile importarla già all'interno del progetto se questa è scritta in un linguaggio basato sul C e quindi compatibile con quello del sistema (objective-C), questo consente di riutilizzare il codice già esistente, ma si ripercuote in termini di prestazioni, infatti per essere eseguito sarà necessario un wrapper apposito che consenta l'interpretazione del linguaggio. Se invece non si ha già un modello dei dati implementato è possibile costruirlo, ma con attenzione sempre alla sua complessità, utilizzando degli appositi oggetti che consentono di ideare un modello semplice, con tipi di dato basilari, e quindi senza complesse strutture, ideale per piccole applicazioni. Oppure ci si può avvalere di un modello dei dati basato su un piccolo database e quindi con tutte le caratteristiche tipiche di questi ultimi.

Per quanto riguarda invece le interfacce si hanno due approcci consigliati, il primo consiste nel costruire l'interfaccia come un insieme di blocchi di base e predefiniti all'interno del sistema, questo facilita molto il lavoro di uno sviluppatore, consentendo di creare UI molto complesse con semplici passaggi. Se invece è necessaria una grafica più elaborata e con frequenti periodi di refresh, ad esempio software ludico, conviene utilizzare la tecnologia OpenGL ES, specifica per alte prestazioni.

2.2.3 Applicazione Windows Phone

Per la creazione di software per Windows Phone, Microsoft mette a disposizione gratuitamente una serie di strumenti racchiusi in un apposito SDK. Gli strumenti necessari sono:

1. Visual Studio Express for Windows Phone
2. Microsoft Expression Blend for Windows Phone
3. Un emulatore per testare le applicazioni sul PC
4. Application Deployment: un componente che consente di eseguire il deploy delle applicazioni sull'emulatore o sul device

la creazione della user interface avviene utilizzando XAML, il linguaggio dichiarativo presentato da Microsoft a fine 2003 e già familiare a chi sviluppa applicazioni Windows con Windows Presentation Foundation o applicazioni Web con Silverlight. Infatti, per gli sviluppatori .NET, Windows Phone SDK rappresenta la naturale continuazione di un processo di continua evoluzione degli strumenti di sviluppo.

Per gli sviluppatori di questa piattaforma risulta fondamentale apprendere le specifiche dettate dalle linee guida prima di sviluppare le applicazioni. Microsoft ha reso disponibile sin da maggio 2010 la User Experience Design Guidelines for Windows Phone, ovvero le linee guida per la progettazione di applicazioni sulla nuova interfaccia utente. È importante infatti seguire queste linee guida per aderire alle specifiche funzionali e uniformarsi al tipo di "esperienza" che l'utente farà sul telefono.

Prima di esaminare effettivamente come si progetta e sviluppa il software su Windows Phone è obbligatorio introdurre la nuova interfaccia windows 8, che risulta un'importante cambiamento rispetto a quelle attualmente presenti.

2.2.3.1 Interfaccia Windows 8

Sia per il sistema operativo desktop che mobile, Microsoft ha progettato una nuova interfaccia che prende spunto dai segnali con cui le persone sono abituate ad interagire ogni giorno. Infatti questa interfaccia prevede l'utilizzo di grossi blocchi, del tutto simili alla segnaletica che si può vedere all'interno di una metropoli. Infatti, malgrado la diversità linguistica,

l'iconografia all'interno dei vari segnali rimane comprensibile da chiunque, con questo nuovo stile si vuole promuovere un nuovo modo di interagire in cui il linguaggio di comunicazione e il codice visivo è immediato, semplice, facilmente comprensibile e veloce. Questa novità non vuole solo essere un semplice cambiamento di interfaccia, ma una vera e propria innovazione a livello di linguaggio di design.

2.2.3.2 Principi dell'Interfaccia Windows 8

Un'applicazione in stile Windows 8, prevede la progettazione di UI, che siano pulite, leggere, aperte e veloci, e cioè UI in cui il superfluo è ridotto al minimo: tutto ciò che non è strettamente necessario, viene eliminato o messo in secondo piano (nel caso di applicazioni Windows Phone ad esempio rilegare le azioni secondarie nella app bar a scomparsa). In particolare con questa nuova concezione dello spazio, che vede un largo utilizzo degli spazi vuoti, si vogliono mettere in risalto solo le operazioni fondamentali e più comuni, direzionando quindi l'attenzione dell'utente su di esse.

Per capire meglio quando appena descritto è utile prendere visione di un esempio. In questo esempio sarà possibile notare come solo le operazioni importanti sono poste nella schermata principale, quelle secondarie invece sono disponibili, ma appunto in secondo piano.

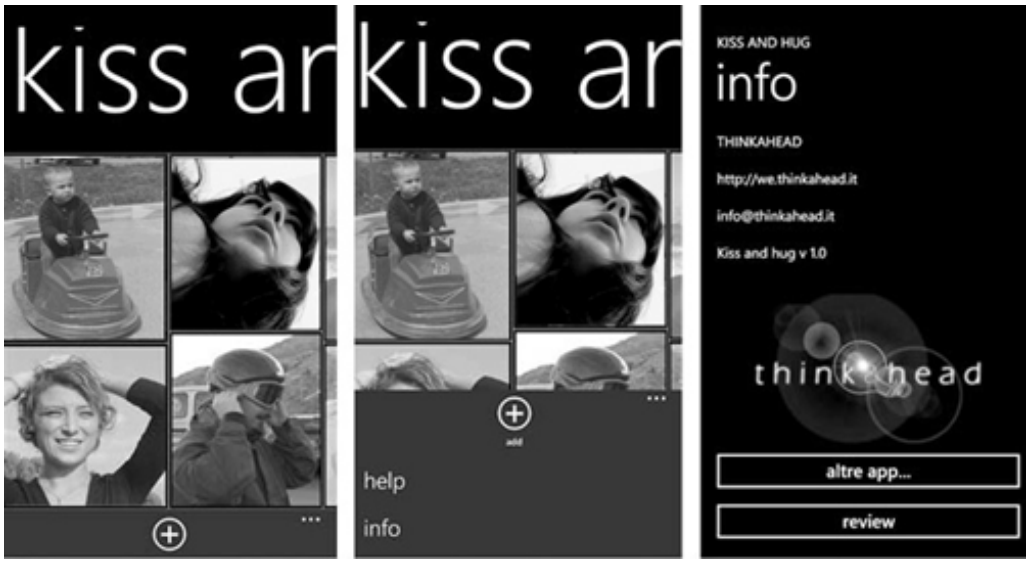


Figura 2.8: Esempio dell'Interfaccia Windows 8

Piuttosto che utilizzare molteplici icone, come invece sono ricche le altre interfaccie, in Windows 8 vengono preferiti diverse tipologie di tipografia, differenziando molto il peso del testo. In questo modo è possibile definire l'importanza dell'elemento, sempre per indirizzare l'attenzione dell'utilizzatore. Inoltre si utilizzano caratteri ad alta densità di pixel per renderli maggiormente leggibili. Di seguito possiamo vedere un altro esempio di come il tutto si presenza.

Figura 2.9: Esempio dell'Utilizzo della Tipografia nell'Interfaccia Windows 8

Per quanto riguarda le animazioni dell'interfaccia, queste sono state minuziosamente studiate per evidenziare un cambiamento nell'applicazione stessa, infatti quando si ha un'animazione questa vuole identificare una qualche novità nel contenuto dell'applicazione. Una precisazione importante riguardo alle animazioni consiste nel loro utilizzo parsimonioso in modo che l'utente non venga distratto da esse e queste non rendano l'ambiente più pesante.

Il Contenuto e non il Contenitore, forse questo è il principio più importante suggerito da METRO. Il contenuto è l'applicazione stessa non le sue decorazioni. Evitando tutto il superfluo le nostre applicazioni non diventeranno solo più fruibili, ma anche più leggere ed efficaci, inoltre l'utente si sentirà al centro dell'attenzione soprattutto quando è lui a creare il contenuto.

I capisaldi di Windows 8, sono gli ultimi suggerimenti di questo nuovo Linguaggio di Design, tre importanti concetti su cui basare la User Experience:

- Personale

  E' l'utente che sceglie cosa per lui è importante e lo mette in primo piano, su windows phone nello start.

- Rilevante

  Ad esempio in una applicazione che sfrutta la posizione GPS, come potrebbe essere un'applicazione sulle previsioni del tempo, le prime informazioni date sono quelle riguardo la nostra posizione attuale.

- Connesso

  Le applicazioni ci permettono di rimanere in contatto, di avere notizie, di ricevere informazioni sulle persone per noi importanti.

2.2.3.3 Template e Introduzione a Silverlight e XAML

In base ai principi descritti ad inizio sezione, vengono messi a disposizione degli sviluppatori diversi template che verranno visionati ora per strutturare la propria applicazione. Questi template si suddividono in due principali categorie: XNA e Silverlight. Di seguito si può visionare una tabella contentente ogni template disponibile:

Template	Descrizione
Windows Phone Application	è il template per la creazione di una applicazione nel più tradizionale dei termini: avremo a disposizione il designer di Visual Studio e l'editor del codice XAML per comporre le varie pagine e relativo code behind
Windows Phone Databound Application	consente di creare applicazioni basate su una navigazione master/detail, seguendo alcuni principi del pattern Model-View-ViewModel
Windows Phone Class Library	consente la creazione di una DLL (assembly .NET)per centralizzare le classi comuni da riutilizzare nei vari progetti o per creare i vari layer dell'applicazione stessa
Windows Phone Panorama Application	è un progetto derivato da Windows Phone Application che propone, nella pagina principale l'utilizzo del controllo Panorama
Windows Phone Pivot Application	è un progetto derivato anch'esso da Windows Phone Application che propone, nella pagina pagina principale, l'utilizzo del controllo Pivot ovvero il sostituto in chiave moderna del Tab Control
Windows Phone Silverlight and XNA Application	permette di creare applicazioni Silverlight in grado di renderizzare oggetti grafici tramite il framework XNA
Windows Phone Audio Playback Agent	è un progetto per creare assembly, che verranno eseguiti dal runtime in background con funzionalità di riproduzione di file audio
Windows Phone Audio Streaming Agent	simile al precedente template ma consente la riproduzione di stream audio
Windows Phone Scheduled Task Agent	permette di creare class library in grado di eseguire attività in background, siano esse schedulate o intensive

Abbiamo parlato precedentemente di Silverlight e XAML, ma non abbiamo specificato come questi effettivamente hanno a che fare con la costruzione di un'applicazione mobile.

Silverlight fu originariamente progettato per fornire agli sviluppatori un metodo per creare applicazioni Internet avanzate (RIA), utilizzando gli strumenti ed i linguaggi standard .NET, in questo modo coloro che sviluppano applicazioni Windows utilizzando .NET possono trasferire le proprie competenze sullo sviluppo web senza dover imparare JavaScript ed i diversi modi in cui i vari browser interpretano questo tipo di codice. Si è pensato quindi di adottare questa tecnologia come piattaforma su cui effettivamente sviluppare anche le applicazioni mobile su Windows Phone.

Lo sviluppo di applicazioni Windows Phone con Silverlight si basa sul linguaggio XAML. Per chi conosce HTML e XML non è complicato comprendere il linguaggio XAML. Quando si lavora con HTML infatti tipicamente su una pagina in cui è necessaria una certa interattività si dichiarano controlli come ¡img¿ per la visualizzazione di immagini. Quando si utilizza XAML l'interfaccia utente si crea in modo del tutto simile. Per esempio, se si vuole visualizzare in una pagina un pulsante basta scrivere il seguente codice XAML:

```
<Button Content="Cliccare" />
```

La sintassi XAML è molto simile a quella HTML. Un importante beneficio derivante dall'utilizzo di XAML per la creazione delle interfacce utente è che tramite questo linguaggio è possibile separare completamente il codice dell'applicazione dalla progettazione dell'interfaccia grafica. Infatti è possibile utilizzare Expression Blend per realizzare le interfacce delle applicazioni WP7 e poi importare le stesse all'interno di Visual Studio. In pratica si tratta della stessa funzione che il linguaggio XML esegue su Android.

2.2.3.4 Struttura di un'Applicazione

Ogni applicazione Windows Phone contiene un file *App.xaml* che rappresenta l'intera applicazione. A questo file non corrisponde un'interfaccia grafica, esso contiene soltanto codice XAML che non è dunque riservato alle pagine a cui corrisponde un'interfaccia utente. Questo file ingloba le risorse che sono globalmente disponibili all'interno dell'applicazione. Nel codice è possibile vedere che è presente una voce *PhoneApplicationService* contenente

diversi gestori corrispondenti agli eventi di lancio, attivazione, disattivazione e chiusura dell'applicazione. I corrispondenti gestori di questi eventi (event handler) sono posizionati all'interno del file *App.xaml.cs*, se desideriamo eseguire qualche operazione durante uno o più di questi eventi basta inserire il corrispondente codice all'interno del gestore corrispondente.

L'altro file fondamentale è *MainPage.xaml*. L'interfaccia utente che visualizziamo nel simulatore viene rappresentata utilizzando il codice XAML presente in questo file. All'inizio della pagina notiamo la dichiarazione dell'elemento *PhoneApplicationPage*. Esso rappresenta l'elemento padre di ogni oggetto all'interno dell'applicazione ed ogni schermata addizionale viene considerata figlia di quest'oggetto.

Il file *WMAppManifest.xml* invece contiene alcune impostazioni specifiche per l'applicazione, come l'indicazione della pagina che deve essere caricata all'avvio.

Infine un breve accenno a due immagini predefinite presenti all'interno del nostro progetto. L'immagine SplashScreenImage.jpg rappresenta un orologio ed è l'immagine che viene mostrata durante il caricamento dell'applicazione. Ovviamente è possibile sostituirla con un'immagine a proprio piacimento, assicurandosi però di utilizzare lo stesso nome.

L'altra immagine è ApplicationIcon.png e corrisponde all'icona con cui viene rappresentata un'applicazione nella lista delle applicazioni installate sul dispositivo. Anche in questo caso è possibile sostituirla, utilizzando sempre lo stesso nome.

2.3 Conclusioni

Si può notare quindi che allo stato attuale sono presenti diverse tipologie di sistemi mobile. Anche se alcuni di questi presentano molte caratteristiche in comune, nella maggior parte dei casi risulta difficile effettuare il porting di un applicazione da una piattaforma all'altra in maniera agile. Inoltre la presenza di numerosi ambienti porta ad una grande eterogeneità, che ha come vantaggio la possibilità di scelta, da parte dei progettisti software, riguardo quale di questi più congeniale alle proprie capacità e che meglio si adatta al compito che il proprio software deve svolgere, ma ha come svantaggio la grande deframmentazione del mercato mobile stesso, portando

ad una divisione sostanziale del bacino di utenza dell'intero mercato, in base alla piattaforma.

Quindi se si volesse sviluppare il proprio applicativo con lo scopo principale di renderlo reperibile al maggior numero di utenti, ciò sarebbe un grande problema da risolvere, senza contare che per sviluppare su diverse piattaforme sono necessarie disparate conoscenze, uno stanziamento cospicuo di risorse e bisogna considerare le differenze riguardanti l'esperienza utente sui diversi sistemi.

Per superare questo problema, come già spiegato nel capitolo introduttivo, si è pensato di costruire svariati frameworks che consentono di rendere il software portabile quasi ovunque e, soprattutto, con una sola stesura del codice stesso.

Quindi nel prossimo capitolo verranno introdotte queste tecnologie, mostrandone gli aspetti principali e le differenze e le tecnologie che adottano.

Capitolo 3

Mobile Frameworks Per Applicazioni Web

Di mobile frameworks che si pongono come obbiettivo lo sviluppo di applicazioni mobile platform-independent ne esistono svariati. In questo capitolo ne verranno introdotti i principali e le conseguenti strategie adottate per lo sviluppo tramite questi strumenti.

Come già introdotto, è possibile individuare fin da subito la tecnologia principale che questi, e molti altri, adottano per rendere l'applicativo più portabile ed indipendente possibile: la tecnologia web.

A fine capitolo inoltre verrà effettuata una panoramica su come effettivamente le applicazioni web vengono sviluppate, possiamo individuare fin da subito le problematiche che queste tecnologie creano nella produzione di applicazioni.

3.1 Problemi Derivanti dall'Utilizzo di Tecnologie Web

Volendo ampliare l'introduzione del primo capitolo si possono citare i motivi principali che portano alla scelta delle tecnologie web: innanzitutto sono strumenti ampiamente utilizzati e a conoscenza di quasi tutti gli sviluppatori e progettisti informatici ed inoltre, ultimamente, le applicazioni web hanno acquisito svariate funzionalità aggiuntive che derivano dai recenti aggiornamenti delle tecnologie. Infatti con il passaggio a HTML5 e CSS3 le

applicazioni web sono evolute, fino a raggiungere molti punti di incontro tra queste e le applicazioni mobile. Ad esempio ora sono in grado di eseguire il *data e application caching* che gli consente di poter essere eseguite anche offline, mentre le applicazioni mobile richiedono sempre più un iterazione con il web.

Nonostante questi vantaggi, insorgono altre problematiche da affrontare, causate dalla natura ed inadeguatezza delle tecnologie web, come per esempio:

1. la progettazione stessa dell'applicazione risulta difficoltosa

   Non si ha una ben definita architettura, questo perchè si utilizzano diverse tecnologie che non sono state ideate per la progettazione di software locale. Questa problematica può portare a una difficoltà nel *separation of concerns*, nonchè a del codice ridondante.

2. Scarso supporto IDE

   Proprio a causa dell'eterogeneità dei linguaggi difficilmente si ha un supporto IDE completo come lo si avrebbe per un linguaggio specifico. Con questo si rinuncia ad esempio all'individuazione degli errori in modo statico, alla risoluzione dei collegamenti e al refactoring.

3. Scarsa astrazione

   Le tecnologie web non supportano pienamente i meccanismi di astrazione tipici di un linguaggio di programmazione. Questo causa l'assenza di interfaccie e il difficile riutilizzo del codice, che quindi spesso deve essere inutilmente duplicato.

4. Continuation-Passing Style

   Questo stile di programmazione nasce dal Javascript, infatti JS non consente di avviare più thread durante l'esecuizione (*Single-Threaded*) quindi per ovviare a questa mancanza molte API sono sincrone: quando queste vengono chiamate, si genera un thread gestito dal browser che, al completamento, torna al thread iniziale invocando un apposito metodo specificato al momento della chiamata (*Callback Method o Continuation Method*) che come argomento riceve il valore di ritorno della chiamata precedente.

Questo stile porta vantaggi e svantaggi: consente di mantenere elevate le performance evitando di bloccare il browser anche a fronte di chiamate che richiedono tempo per risolversi, ma porta come svantaggio la difficoltà da parte dello sviluppatore di adattarsi a questo nuovo stile di programmazione inusuale.

Qui di seguito ecco un esempio di questo stile di programmazione:

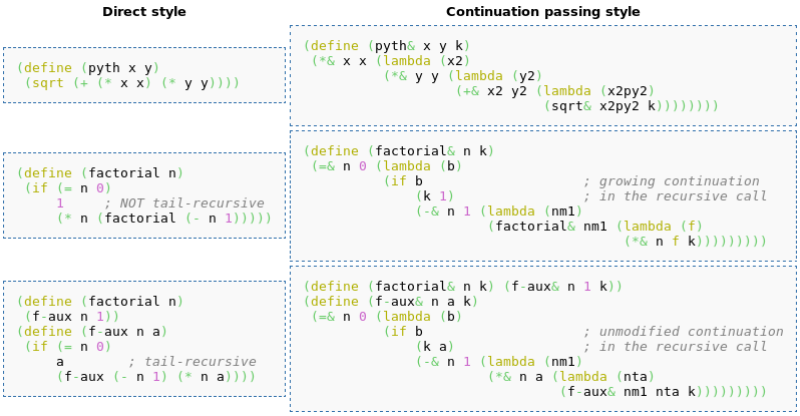


Figura 3.1: Confronto tra Stile Diretto e Continuation-Passing Style

5. Utilizzo delle risorse

   Visto la moltitudine di novità introdotte dai dispositivi mobile e soprattutto l'utilizzo di sensori particolari come ad esempio l'accelerometro o il sistema GPS, risulta particolarmente complicato disporre di tutte le API necessarie per interagire con queste risorse.

6. Performance

   Nonostante i dispositivi mobile stiano acquisendo sempre maggior potenza di calcolo e i browser diventando sempre più performanti, le prestazioni di una mobile web application saranno sempre inferiori rispetto ad un'applicazione nativa. Nonostante questo consentono comunque di ottenere delle buone prestazioni, dipendenti soprattutto dalla natura stessa del software, ad esempio è raccomandabile optare per un'applicazione nativa se questa deve effettuare pesanti calcoli, mentre conviene scegliere una mobile web application se sono previste grandi scambi di dati attraverso la rete.

Per far fronte a questi quesiti, ogni framework, decide di adottare una propria strategia e soluzione. Quindi vedremo ora, in generale, quattro tra i più importanti della categoria: PhoneGap, JQueryMobile, Secha Touch, titanium e Jo. In seguito ci si concentrerà su un ulteriore middleware chiamato Mobl, che si differenzia dagli altri per strategie e particolarità interessanti, pur mantenendo sempre come base per le proprie applicazioni i linguaggi sopra citati.

3.2 PhoneGap

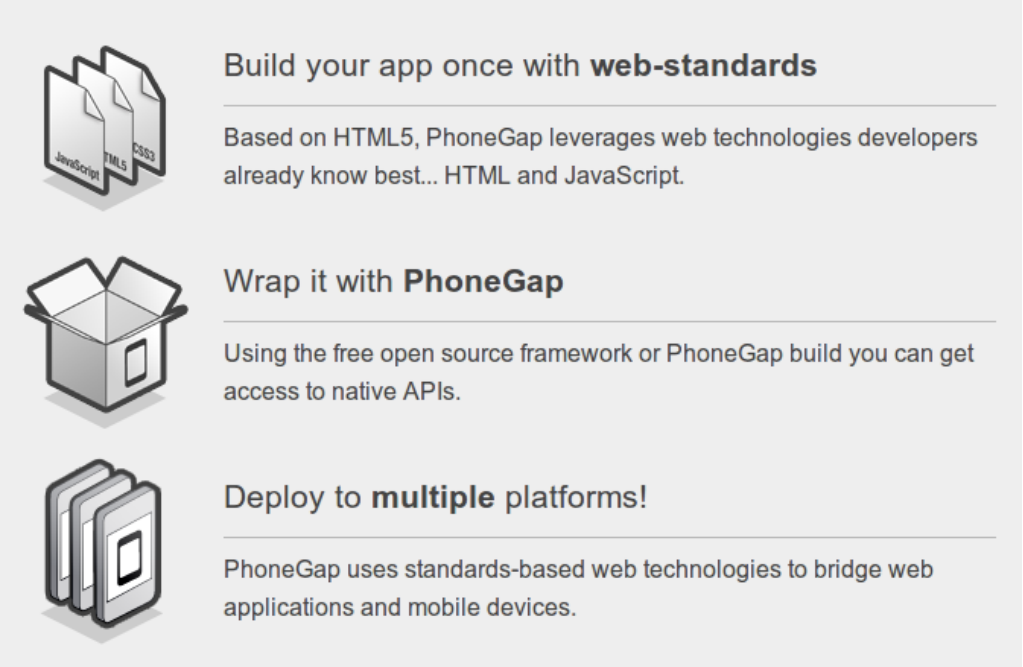


Figura 3.2: Riassunto Phonegap

Si tratta di un framework open source che svolge la funzionalità di wrapper per applicazioni web, scritte appunto in HTML5, CSS3 e Javascript, consentendo di eseguirle all'interno delle piattaforme mobile come applicazioni native. Tutto questo è possibile tramite apposite API del framework

che sono collegate alle librerie native della piattaforma. Una problematica di questo sistema consiste nel diverso supporto tra i sistemi operativi mobile: può capitare infatti che alcune funzionalità siano presenti per certi sistemi, mentre assenti in altri causa l'arretrato sviluppo delle API.

L'approccio di input in PhoneGap è detto event-driven: cioè vengono definite delle azioni eseguite dall'utente che generano un qualche effetto all'interno dell'applicazione. Innanzitutto è necessario definire quali eventi gestire e, tramite il pattern event-listener, associare ad ognuno un ascoltatore apposito, sarà questo che, all'esecuzione dell'evento eseguirà il codice associato e scritto nell'applicazione web, generalmente in javascript. I vantaggi principali di questa strategia riguardano la possibilità di: creare eventi personalizzati, associare ad un evento ascoltatori multipli e collegare o scollegare facilmente ascoltatori agli eventi.

Un'aspetto interessante e notevole di questo framework consiste nella possibilità di utilizzare le funzionalità native di una piattaforma, questo consente di accedere ad esempio ai sensori del device e quindi adattare l'applicativo di conseguenza. Anche in questo caso però bisogna porre molta attenzione alla presenza dei sensori utilizzati in tutti i device supportati e all'interno delle piattaforme, per non incorrere in problematiche future.

3.3 JQuery Mobile

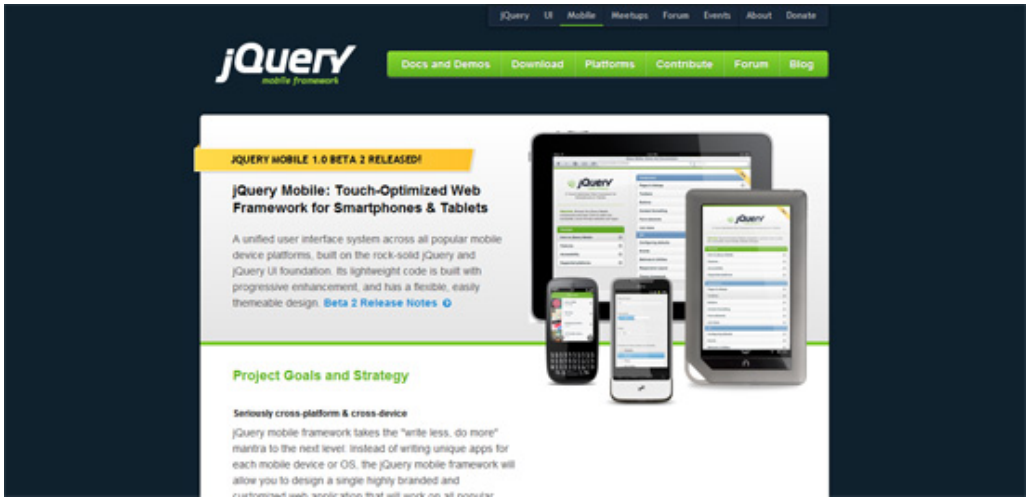


Figura 3.3: JQuery Mobile

Questo tipo di framework è diverso da PhoneGap, infatti non si preoccupa di effettuare un wrapping di web application verso le piattaforme mobile, ma si concentra sullo sviluppo delle interfacce utente delle applicazioni web finalizzate all'utilizzo mobile. Si basa su un core molto leggero, che quindi non incide pesantemente sulle performance del software, che consente di manipolare il DOM, gestire eventi e la comunicazione con i server tramite Ajax, componente utile anche per la costruzione vera e propria dell'interfaccia, in particolare delle animazioni di questa.

JQuery Mobile è uno dei framework più supportati sia per quanto riguarda le piattaforme che i browser stessi. Infatti uno degli obbiettivi principali del sistema è quello di rendersi il più compatibile possibile con tutte le tipologie di dispositivi presenti. Per far ciò infatti, nonostante la tecnologia maggiormente utilizzata sia HTML5 e CSS3, tecnologie attualmente all'avanguardia, JQueryMobile consente di eseguire molte funzionalità di base anche in quei browser che non posseggono il supporto a queste tecnologie.

Inoltre si ha una notevole facilità d'uso di questo sistema, infatti è possibile ideare semplici pagine senza conoscere l'utilizzo del Javascript.

Infine, due caratteristiche importanti che vanno sottolineate sono: l'elevata personalizzazione dell'interfaccia grazie alla possibilità di utilizzare la grande varietà di temi messi a disposizione dall'attiva comunità e la possibilità di utilizzare PhoneGap per realizzare un'applicazione nativa da quella web costituita con JQuery Mobile.

3.4 Secha Touch 2

Figura 3.4: Secha Touch 2

Anche per questo mobile application framework si utilizza la tecnologia web HTML5/CSS3, e si pone come alternativa valida ai framework proposti precedentemente. Infatti Secha Touch 2 presenta una vasta compatibilità con le principali piattaforme sul mercato: Android, iOS, Blackberry, Kindle fire e molti altri e propone inoltre delle caratteristiche che lo rendono un ottimo prodotto. Innanzi tutto bisogna specificare gli obbiettivi primari di questo middleware, cioè: la maggior velocità delle applicazioni in fase di avvio e di iterazione con l'utente. Per far ciò consente di utilizzare l'accelerazione hardware quando disponibile dal dispositivo.

Alcune features principali di questo sistema sono: le numerose API che mette a disposizione per poter usufruire di tutte le specifiche di un parti-

colare device, il pattern architetturale MVC per la strutturazione dell'applicativo, l'ampia scelta di componenti ed effetti per la costruzione di un layout gradevole e dettagliato ed infine la possibilità tramite l'apposito SDK di portare le proprie applicazioni web in formato nativo sulle piattaforme mobile.

Questo framework è utilizzato da molte aziende importanti che decidono di approcciarsi al mercato mobile e dispone di una ricca documentazione, correlata di esempi, che rendono lo studio e l'apprendimento del sistema molto intuitivo e rapido.

3.5 Titanium

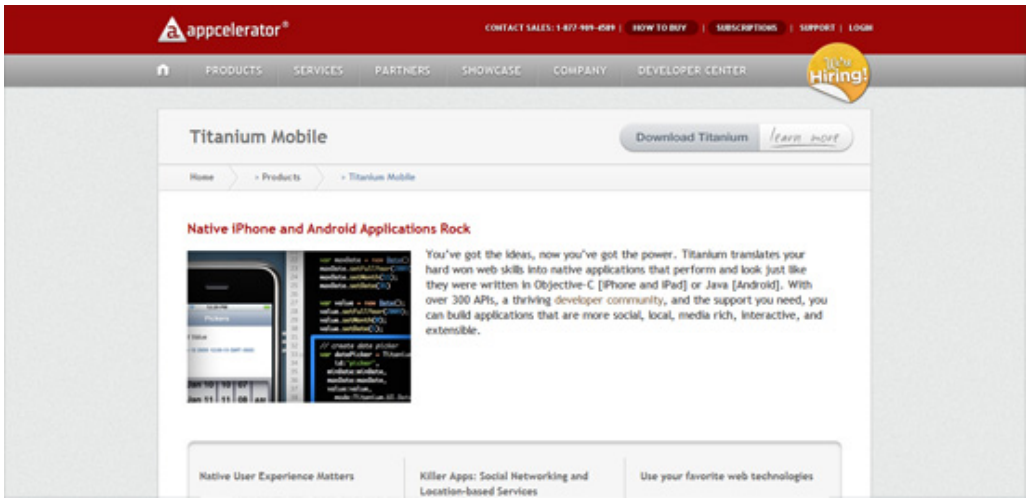


Figura 3.5: Titanium Mobile

Titanium non è propriamente un framework ma un interfaccia, basata su javascript, che si pone l'obbiettivo di tradurre tutto il codice di una applicazione web in linguaggio nativo per le varie piattaforme supportate, in modo da ottenere il così detto look and feel per la piattaforma desiderata, senza dover ovviamente riscrivere il sorgente da zero. Per far ciò dispone di circa 5000 API e una comunità molto attiva ed in continua crescita.

3.6 Jo

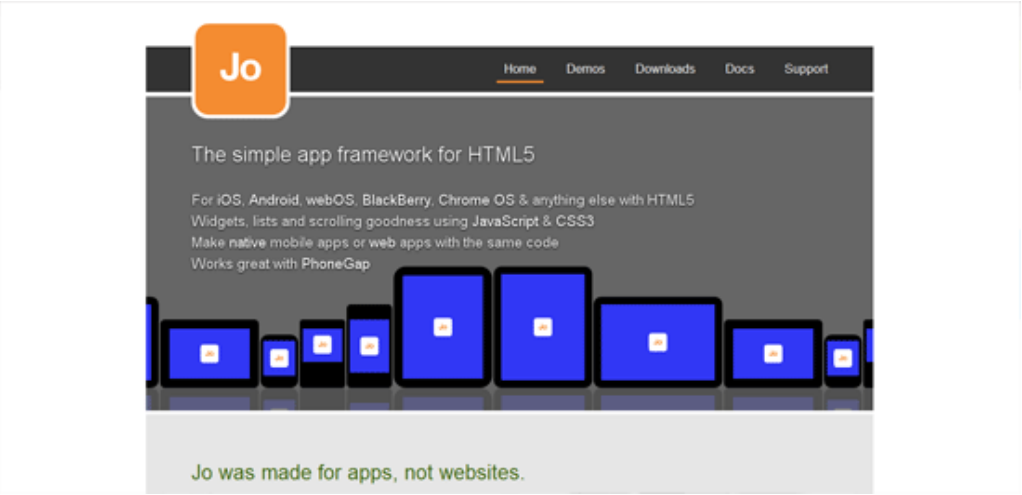


Figura 3.6: Jo

Anche Jo è un framework scritto in javascript che si occupa di portare applicazioni web in campo mobile, ma a differenza dei precedenti questo mostra alcune caratteristiche particolari degne di nota, ad esempio è molto leggero e si integra con le librerie del più noto Phonegap. In particolare può essere la scelta ideale per quegli sviluppatori che vogliono trasformare una pagina web esistente in applicazione web e trasportarla su dispositivi mobili, soprattutto perchè non richiede di conoscere a fondo il framework per essere utilizzato.

3.7 Applicazioni Web Mobile

Prima di visualizzare gli aspetti caratteristici di un'applicazione web è necessario effettuare alcune precise distinzioni sulla natura stessa di queste applicazioni. Possiamo definire un'applicazione web come una via di mezzo tra un software locale ed un sito web. In sostanza si tratta di pagine web evolute in grado di offrire funzionalità complesse simili a quelle fornite dalle comuni applicazioni installate sul dispositivo. Ciò è possibile grazie ai linguaggi di scripting, sia lato client che lato server, che consentono di creare

degli applicativi veri e propri fruibili attraverso un browser, simili, anche dal punto di vista della interfaccia utente, alle comuni applicazioni mobile. Ciò è possibile grazie a tecnologie come AJAX(Asynchronous JavaScript and XML) che consentono di riprodurre in ambiente di Rete comportamenti tipici delle interfacce software, ad esempio gestendo aggiornamenti dei contenuti senza effettuare alcun refresh di pagina. Bisogna però effettivamente specificare che con un semplice linguaggio di scripting è difficile effettuare tutte le operazioni, oltremodo se complesse, che invece sono possibili nativamente.

Questa tipologia di applicazioni portano sia vantaggi che svantaggi. I principali vantaggi sono l'indipendenza dalla piattaforma e dall'utilizzo dei vari marketplace che a volte impongono regole molto rigide per ammettere le varie applicazioni. Inoltre questi software non incidono minimamente sulle capacità di storage dei dispositivi e le loro performance non dipendono dal tipo di dispositivo, bensì dalla bontà della connessione di rete e dalla capacità di calcolo del server su cui vengono eseguite. Quest'ultimo aspetto può diventare a sua volta uno svantaggio però, perchè non sempre è disponibile una rete per scambiare dati con il server, oppure la qualità di quest'ultima può essere scarsa e quindi minare le performance dell'applicazione; anche se ultimamente, con le nuove tecnologie, sono stati ideati dei meccanismi che consentono di eseguire applicazioni web anche senza la presenza di una rete, ma questo richiede necessariamente un'allocazione di spazio sul dispositivo. Seguono poi un'altra serie di svantaggi che si notano rispetto ad una applicazione nativa che consistono nel così detto look and feel del software stesso e nell'esperienza utente riguardo certi comportamenti che non possono essere emulati in modo perfetto a livello web.

Si può infine sottolineare una forte parentela tra siti web ottimizzati per dispositivi mobile e applicazioni mobile. Infatti la sostanziale differenza risiede nella complessità delle operazioni che possono essere svolte con un'applicazione web rispetto che con un sito web ottimizzato, e questo richiede un importante bagaglio di conoscenze tecniche specifiche.

In questa sezione si introdurranno le linee guida da seguire quando si progetta un'applicazione web e gli elementi caratteristici che le tecnologie web mettono a disposizione per implementare questi software.

3.7.1 Suggerimenti per la Progettazione

Quando si progetta un'applicazione web bisogna porre particolare attenzione a vari aspetti caratteristici di questa e di conseguenza effettuare le scelte giuste per ottenere il miglior risultato possibile.

Ogni applicazione web differisce dalle altre per i contenuti che questa mette a disposizione: un'applicazione che mostra contenuti multimediali è certamente differente da un'altra con contenuti testuali. Di conseguenza occorre effettuare una distinzione a questo livello ed in particolare si individuano due macro-categorie: contenuti statici e dinamici. Questa differenziazione porta ad una importante considerazione sulla natura del nostro software, infatti nel primo caso lo si potrà utilizzare anche offline, mentre nel secondo caso è necessario una costante presenza della rete per monitorare i cambiamenti dei contenuti stessi.

Una volta stabilita la macro-categoria bisognerà individuare con che tipo di questi contenuti avremo a che fare: audio, video o immagini. Anche in questo caso risulta cruciale esaminare questo aspetto, perchè sarebbe un grave errore considerare questi elementi allo stesso modo sia in ambito mobile che desktop. Infatti in ambito mobile è necessario tenere conto di queste puntualizzazioni su ogni sottotipo di contenuto:

- Il testo dovrà avere una certa spaziatura orizzontale in modo da evitare lo spiacevole effetto visivo derivante dall'assenza di spazi tra i margini orizzontali e la finestra del browser.

- Le immagini dovranno adattarsi al dispositivo e al suo cambio di orientamento (landscape o portrait).

- Per i video vale quanto detto per le immagini. In più occorre valutare attentamente l'impatto di questi elementi sulla performance.

3.7.2 Struttura di un'Applicazione Web

Fino ad ora abbiamo menzionato le tecnologie che compongono queste applicazioni, ma risulta fondamentale specificare il compito di ognuna di queste per fare in modo che l'applicazione funzioni.

Il modo migliore per la gestione della struttura del software consiste nel suddividere i principali compiti in questo modo:

- Struttura: HTML5
- Presentazione: CSS
- Comportamento: Javascript

Ciascun modulo è interdipendente con gli altri. Un errore in un modulo si ripercuoterà sugli altri e il tutto non funzionerà come previsto. Questa può essere considerata una forte limitazione di tutta la struttura, inoltre è impensabile utilizzare un'architettura differente da questa, farlo sarebbe deleterio.

Occorre innanzitutto verificare che la struttura sia flessibile: cioè con la possibilità di poter in seguito estendere le funzionalità e le parti del software. Ora, senza scendere nei dettagli implementativi, elenchiamo alcuni importanti suggerimenti da seguire per ottenere un risultato migliore.

Per quanto riguarda la struttura, quindi la parte HTML:

- Limitate il numero di attributi id ai blocchi fondamentali
- Usate le classi CSS in modo cumulativo sfruttando la cascata
- Aggiungete dati agli elementi tramite gli attributi custom data di HTML5.

Per quanto riguarda il CSS invece bisogna distinguere, anche in questo caso se il layout può essere statico o deve presentare degli elementi dinamici, quindi con utilizzo di animazioni e transizioni. In particolare nel caso dell'utilizzo di queste ultime, è conveniente legare il codice con le classi che poi dovranno essere modificate da Javascript.

Il codice JavaScript deve essere OOP. Evitate sempre l'approccio procedurale, ma pensate al codice in modo astratto, considerando ad esempio un elemento come un oggetto, con proprietà e metodi. Questo vi permetterà di riusare spesso i vostri oggetti in altre applicazioni web, cambiandone solo il comportamento e non la struttura.

Nonostante sia possibile definire una struttura per queste applicazioni delegando ogni singolo linguaggio al suo compito, bisogna ricordare quanto specificato ad inizio capitolo sulle problematiche della gestione di un software, soprattutto se complesso, con queste tecnologie.

3.7.3 HTML5

Si tratta dell'ultima versione della tecnologia per eccellenza in ambito web ed in particolare vengono introdotte numerose novità che aiutano i vari web designer e sviluppatori web a progettare ed implementare più efficientemente e velocemente i propri prodotti sulla rete. Le nuove caratteristiche che questa tecnologia propone sono tantissime, ma nel nostro caso prenderemo in esame solamente alcune di queste che servono in particolare per lo sviluppo di applicazioni mobile web.

3.7.3.1 HTML5 Cache Manifest e Local Storage

Il *Cache Manifest* in HTML5 è una funzionalità che permette di accedere ad un'applicazione web anche senza l'accesso alla rete, questo però richiede di scaricare un certo quantitativo di dati per rendere ciò possibile.

Spieghiamo ora brevemente come funziona questo meccanismo. Come abbiamo già detto, un'applicazione web è una pagina web evoluta, e quindi sarà identificata da un URL e contenente varie risorse gestite tramite HTML, CSS e Javascript. L'URL identificativo di una di queste può essere copiato all'interno di un file manifest, che tiene traccia delle risorse presenti e opportunamente modificato. La prima volta che uno di questi software viene messo in esecuzione, il file manifest viene letto, e il browser web si preoccupa di scaricare le risorse ivi elencate e conservarle localmente. Successivamente, in caso di mancanza di rete, il browser potrà utilizzare le risorse precedentemente salvate ed effettuare il rendering di queste. Bisogna specificare che la cache utilizzata non è quella solita del browser, ma una appositamente creata per l'occasione.

Anche da implementare risulta molto semplice. Infatti possiamo osservare di seguito un esempio di un file manifest:

```
CACHE MANIFEST
index.html
/images/logo.png
/css/styles.css
/js/jquery-1.4.min.js
/js/offline.js
```

L'unico accorgimento risiede nel specificare nella prima riga del file il comando *CACHE MANIFEST*. Una volta costruito il nostro file, basterà

salvarlo, ad esempio come offline.manifest, e specificarne la presenza sulla nostra pagina con un apposito tag HTML:

```
<html manifest="offline.manifest">
```

Uno degli aspetti meno pratici dell'application cache è che il semplice aggiornamento di un file incluso nel manifest non comporta l'aggiornamento della versione in cache. Se insomma noi cambiamo una pagina (o immagine o script o css) già inclusa nel file manifest questa non verrà mai modificata di conseguenza perché il browser, per verificare che ci sono stati aggiornamenti, controllerà i contenuti del file manifest e dato che questi non sono cambiati, i file sono sempre gli stessi, è solo il contenuto dei file ad essere stato cambiato, li riterrà non modificati e dunque non li riscaricherà in cache.

Per ovviare a questo inconveniente ogni volta che cambiamo un file della cache è necessario cambiare anche il file manifest, il metodo migliore è di farlo aggiungendo un commento di versione, quindi qualcosa tipo:

```
CACHE MANIFEST
# versione 1.0.2 del 19 gennaio 2011
index.html
/images/logo.png
/css/styles.css
/js/jquery-1.4.min.js
/js/offline.js
```

I vantaggi della application cache sono molteplici, il principale rispetto a quella tradizionale è il controllo: con lo strumento offerto da HTML5 possiamo dire noi con precisione quali risorse tenere in memoria così da offrire agli utenti per un'esperienza di navigazione offline piacevole e completa, mentre precedentemente questa scelta era delegata al browser. Oltre a questo è possibile salvare anche script così da consentire un utilizzo totale delle funzionalità HTML5 e non solo, anche offline. Infine, è possibile salvare anche file che non si sono visitati, ma che potrebbero essere utili per una navigazione completa del sito, mentre nella normale cache del browser una pagina per essere memorizzata deve essere stata visitata almeno una volta.

Per quanto riguarda l'utilizzo di questa funzionalità su dispositivi mobile, bisogna fare attenzione a non abusarne. Costringere un browser a scaricare svariati megabyte di contenuti ha un senso solo se si è assolutamente certi che gli utenti dispongono di una connessione veloce (Wi-Fi).

Nel caso in cui gli utenti navighino con una connessione diversa, il browser non riuscirà a scaricare tutti i file in tempi ragionevoli.

Il local storage, analogamente all'application cache, conserva informazioni sul dispositivo, ma in questo caso non si tratta di elementi strutturali della web application, ma di dati inseriti dall'utente. Si possono così salvare le preferenze impostate da questo, oppure lo stato corrente dell'applicazione.

Supponiamo per esempio che un utente stia compilando un form, e che ad un certo punto, per un qualsiasi motivo, il browser si chiuda; una volta riaperto, grazie al local storage l'utente ritroverebbe le informazioni inserite fino a quel momento, e non dovrebbe perciò ricompilare il form dall'inizio.

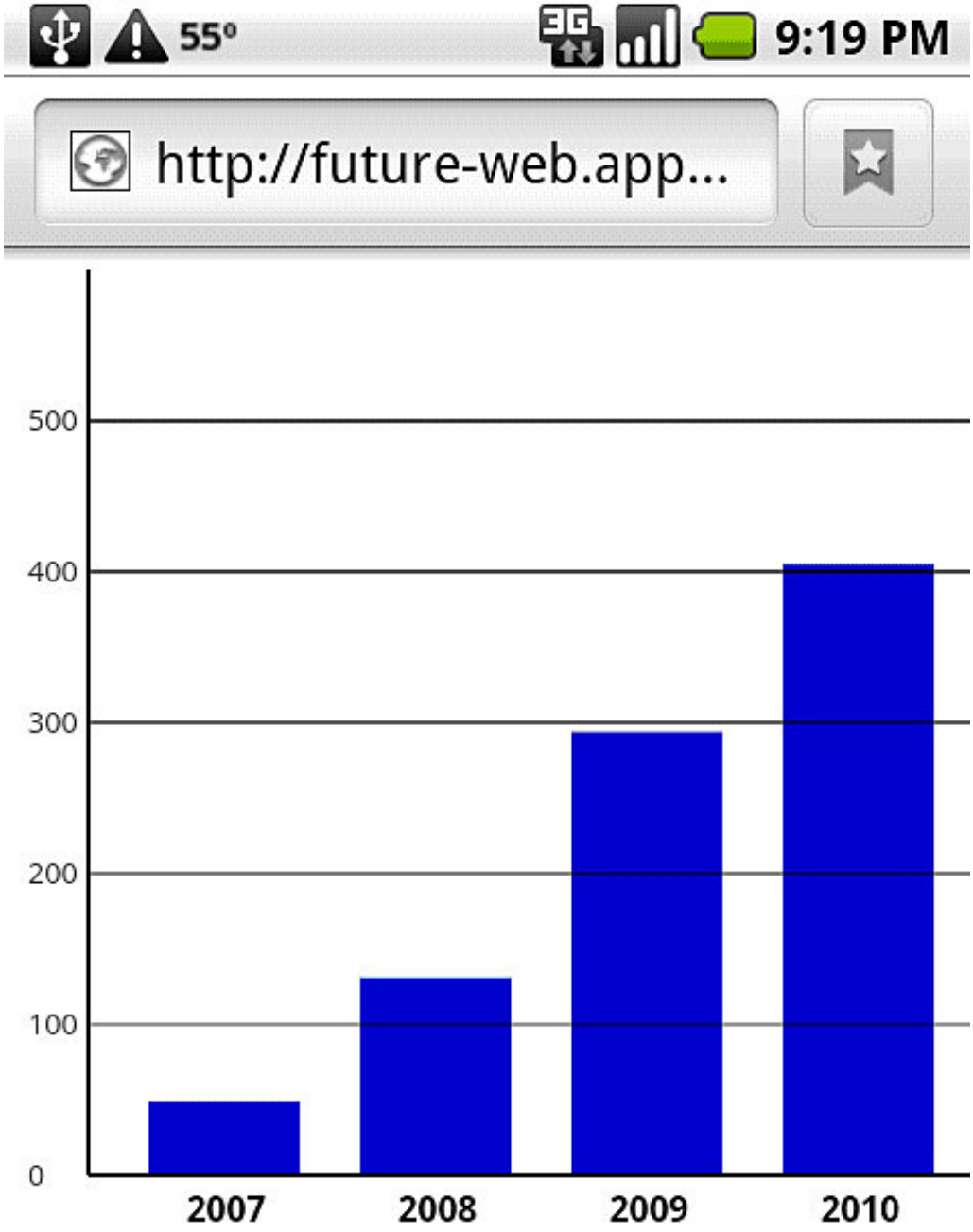
3.7.3.2 Elementi Grafici Canvas

Gli elementi canvas sono strumenti specifici per creare grafiche, anche abbastanza complesse, all'interno del browser. Il supporto a questi elementi e alle relative API è sempre stato difficile perchè erano necessari plugin vari e alcuni browser non supportavano affatto questa tecnologia, in particolare i primi browser mobile e le versioni di Internet Explorer precedenti alla 9. Per questo fino ad ora questa tecnologia è sempre stata sfruttata poco rispetto alle sue grandi potenzialità grafiche. Fortunatamente ora tutti i moderni browser, in particolare quelli basati su webkit, supportano questa tecnologia e quindi è finalmente supportata e utilizzabile.

Le API per l'utilizzo di elementi canvas sono API di basso livello, quindi consentono di disegnare: linee, curve, cerchi, poligoni e riempirli con colori e gradienti. Le sue potenzialità non si fermano qui, infatti è possibile ottenere anche grafiche molto complesse e il web è ricco di esempi che possono mostrarne le potenzialità.

Riportiamo di seguito un esempio semplice di ciò che è possibile ottenere con questi elementi.

Figura 3.7: Semplice Esempio dell'Utilizzo degli Elementi Canvas



Si può notare come il risultato sopra ottenuto è visualizzabile all'interno di un browser mobile ed in seguito riportiamo il codice HTML riferito alla pagina che conterrà il richiamo all'elemento canvas e la funzione che grafica effettivamente il disegno sopra illustrato.

Listing 3.1: Pagina html contenente il richiamo alla funzione canvas

```
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=
       UTF-8">
    <meta name="viewport" content="width=device-width; initial-
       scale=1.0;
        maximum-scale=1.0; user-scalable=0;"/>
    <meta name="apple-touch-fullscreen" content="YES" />
    <title>HTML 5 Reports</title>
    <script type="text/javascript">
        function init(){
            var data = [{year : "2007",sales : 49},
                {year : "2008",sales : 131},
                {year : "2009",sales : 294},
                {year : "2010",sales : 405}];
            var report = {x : "year",
                    y : "sales",
                    values : data};
            graph(report, 350, 300);
        }
    </script>
</head>
<body onload="init()">
    <canvas id="graph"></canvas>
</body>
</html>
```

Riportiamo di seguito anche il contenuto della funzione graph:

Listing 3.2: Contenuto della funzione graph responsabile del disegno

```
function graph(report, maxWidth, maxHeight){
    var data = report.values;
    var canvas = document.getElementById("graph");
    var axisBuffer = 20;
    canvas.height = maxHeight + 100;
    canvas.width = maxWidth;
    var ctx = canvas.getContext("2d");
```

```
    var width = 50;
    var buffer = 20;
    var i = 0;
    var x = buffer + axisBuffer;
    ctx.font = "bold 12px sans-serif";
    ctx.textAlign = "start";
    for (i=0;i<data.length;i++){
        ctx.fillStyle = "rgba(0, 0, 200, 0.9)";
        ctx.fillRect(x, maxHeight - (data[i][report.y] / 2),
                 width, (data[i][report.y] / 2));
        ctx.fillStyle = "rgba(0, 0, 0, 0.9)";
        ctx.fillText(data[i][report.x], x + (width / 4),
           maxHeight + 15);
        x += width + buffer;
    }

    // draw the horizontal axis
    ctx.moveTo(axisBuffer, maxHeight);
    ctx.lineTo(axisBuffer+maxWidth, maxHeight);
    ctx.strokeStyle = "black";
    ctx.stroke();

    // draw the vertical axis
    ctx.moveTo(axisBuffer,0);
    ctx.lineTo(axisBuffer,maxHeight);
    ctx.stroke();

    // draw gridlines
    var lineSpacing = 50;
    var numLines = maxHeight/lineSpacing;
    var y = lineSpacing;
    ctx.font = "10px sans-serif";
    ctx.textBaseline = "middle";
    for (i=0;i<numLines;i++){
        ctx.strokeStyle = "rgba(0,0,0,0.25)";
        ctx.moveTo(axisBuffer, y);
        ctx.lineTo(axisBuffer + maxWidth,y);
        ctx.stroke();
        ctx.fillStyle = "rgba(0,0,0, 0.75)";
        ctx.fillText(""+(2*(maxHeight -y)), 0, y);
        y += lineSpacing;
    }
}
```

Ad un oggetto canvas è possibile quindi associargli uno stile tramite un CSS e una funzione Javascript che consente di effettuare il disegno voluto.

3.7.4 Javascript

Javascript è il linguaggio di scripting più utilizzato sulla rete per effettuare semplici controlli sulle pagine web e per specificarne il comportamento. Da quando è nato, Javascript si è sempre evoluto integrando nuove caratteristiche per rimanere aggiornato e poter rendere il web sempre più dinamico. Quindi recentemente anche in questo linguaggio sono state introdotte delle nuove funzionalità per consentire di costruire le mobile web application, di seguito introduciamo le più importanti e notevoli.

3.7.4.1 Geolocalization e Tracking

Tramite Javascript e apposite API è possibile attivare la geo-localizzazione. Possiamo vedere qui di seguito come è possibile ottenere la posizione corrente di un'utente:

Listing 3.3: Ottenere la posizione di un utente

```
navigator.geolocation.getCurrentPosition(successCallback,
    errorCallback, options);
```

Il codice visto sopra racchiude l'essenza della geo-localizzazione, infatti nella maggioranza dei casi viene richiestra la posizione corrente. Comunque sono presenti molti altri metodi all'interno dell'oggetto *geolocation* che a sua volta fa parte dell'oggetto *navigation* che non verranno descritti qui.

Va precisato che la chiamata per la geo-localizzazione è una chiamata asincrona, perchè non si sanno effettivamente i tempi di riposta dei satelliti, e può anche restituire esito negativo. Infatti si possono notare due diversi argomenti che corrispondono alle callback da chiamare nel caso di successo o fallimento della chiamata.

In caso di successo viene restituito un oggetto di tipo *position* che a sua volta è composto da due proprietà: un timestamp e un oggetto *Coordinates*. Un oggetto di tipo Coordinates ha varie proprietà:

- latitude
- longitude

- altitude
- accuracy
- altitudeAccuracy
- heading
- speed

Molte di queste proprietà dipendono esclusivamente dal tipo di dispositivo, ma latitudine, longitudine e accuratezza sono sempre presenti, in qualunque device che supporta la geo-localizzazione.

All'interno della callback di errore invece è presente un oggetto del tipo *PositionError* che è composto da un codice ed un messaggio. Il messaggio dipende dal dispositivo ed è utile per effettuare eventuali debug, mentre il codice è di tre tipi:

1. PERMISSION DENIED
2. POSITION UNAVAILABLE
3. TIMEOUT

L'API per il tracking è molto simile al precedente e richiede gli stessi parametri, viene prende il nome di *watchPosition*. Una differenza principale tra le due e che con il tracking si ha in ritorno un ID, questo viene utilizzato con la chiamata *clearWatch* e serve per segnalare la fine del tracking. Quando viene chiamato watchPosition il browser invia, ad intervalli di tempo, la posizione attuale fino a che non viene invocata la clearWatch. La funzionalità di tracking deve essere usata con parsimonia perchè causa un rapido esaurimento della batteria.

3.7.4.2 Multithreading Programming: Web Workers

Javascript non supporta il multi-threading perchè al momento della sua creazione questo non era stato concepito per svolgere importanti operazioni computazionali bensì per eseguire semplici compiti all'interno delle pagine web. Con l'evoluzione del web però è necessario far evolvere anche questo

linguaggio affinché possa effettivamente supportare la programmazione concorrente. Per ottenere questi obbiettivi si utilizzano i *web workers*, questi consentono l'esecuzione di codice Javascript in modo asincrono senza intaccare le performance della pagina web in esecuzione. Effettivamente possono quindi essere comparati a dei thread e non sono altro che file Javascript. La pagina in questione può richiedere uno o più workers e dialogare con essi attraverso semplici metodi, a loro volta un Webworker può eseguire altri webworkers al suo interno ed ogniuno di essi può effettuare operazioni di I/O, calcoli complessi eccetera.

Le API si consentono di creare questi thread partendo da due diverse classi chiamate *Worker* e *ShareWorker*: nel primo caso la sua esecuzione sarà limitata alla specifica sessione di navigazione all'interno della finestra del browser che l'ha invocato, nel secondo invece ogni sessione di navigazione che condivide la stessa origine (lo stesso dominio) potrà connettersi e scambiare messaggi con il medesimo worker. In questo modo lo SharedWorker assume il ruolo di coordinatore, ottimo per, ad esempio, propagare su tutte le finestre del browser puntate su di un particolare dominio un messaggio ricevuto dal server.

Per quanto riguarda l'implementazione, questa avviene nel modo seguente:

```
//pagina principale
bot = new Worker('bot.js');
bot.onmessage = function(event){
  bot.postMessage(prompt(event.data));
}
```

Lo script invocato verrà eseguito in asincrono e potrà inviare messaggi verso la pagina principale attraverso la funzione *postMessage*:

```
// bot.js
postMessage('Perchè la terra è tonda?');
onmessage = function(event){
  if(event.data != null){
    postMessage('Perchè:' + event.data + ' ?');
  }
}
```

Oltre all'utilizzo di postMessage, per consentire la comunicazione tra la pagina principale e il Worker è necessario registrare una funzione all'handler *onmessage*, contenuta all'interno dell'oggetto bot. Si può vedere che, una volta settata la registrazione da parte di entrambi si è stabilito un canale di comunicazione in entrambe le direzioni.

Le API dello SharedWorker differiscono in modo sensibile rispetto a queste, anche se ovviamente mantengono immutato il funzionamento di base; queste non verranno affrontate in questa sede.

3.7.5 CSS3

Il CSS consente, dalla nascita del web, di modellare l'aspetto delle varie pagine e degli elementi che compongono queste pagine per fornire un design sempre più moderno e rendere le pagine web migliori. Nei tempi recenti e con l'introduzione delle mobile web application anche in questa tecnologia sono state effettuate delle migliorie e delle novità proprio per migliorare l'aspetto di queste applicazioni e qualche comportamento tipico puramente estetico.

Per esempio visioneremo tre particolari caratteristiche del CSS3: le transizioni, le animazioni e le trasformazioni. A portare concetti come transizioni, trasformazioni ed animazioni nell'ambito dei CSS è stata Apple, in particolare il team di sviluppo che lavora su Webkit, il motore di rendering del browser Safari (e che poi lo è diventato pure di Google Chrome).

L'esigenza da cui tutto è nato è semplice: trovare un'alternativa a Javascript e soprattutto a Flash nel contesto del mobile, su dispositivi che non hanno la potenza di calcolo e l'autonomia di un computer tradizionale e per i quali il consumo di risorse deve necessariamente ridursi al minimo.

Quanto queste specifiche (a un certo punto accolte tra i moduli ufficiali del W3C) abbiano il sapore di alternativa a Flash, è possibile verificarlo soprattutto con le animazioni CSS3. Si tratta infatti di qualcosa di più complesso e avanzato rispetto alle transizioni perché entra in gioco l'idea di animazione sviluppata su una sorta di timeline a partire da una serie di keyframe.

3.7.5.1 Transizioni

Le transizioni servono per definire quegli effetti di cambiamento sugli oggetti realizzati con l'ausilio di Javascript. Con questa nuova categoria di effetti è possibile dare maggior dinamismo a pagine e applicazioni web andando a realizzare un senso di spostamento e navigazione attraverso le pagine o le applicazioni, caratteristiche tipiche anche di tutte le applicazioni native.

Fino al CSS2 questi effetti non si avevano e quindi quando si voleva ottenere un cambiamento dell'interfaccia al passaggio del mouse per esempio, questo veniva reso immediatamente tramite una renderizzazione istantanea dell'oggetto secondo le specifiche immesse. Possiamo vederne un esempio con un semplice blocco div:

```
div {
  width: 200px;
  height: 200px;
  background-color: 98d925#;
}

div:hover {
  background-color: #ff5c00;
}
```

Con le transizioni invece è possibile effettuare questo passaggio dal vecchio valore al nuovo con un certo ritardo di tempo che renderà effettivo il senso di passaggio. Dall'esempio precedente si ha:

```
div {
  width: 200px;
  height: 200px;
  background-color: 98d925#;

  transition-property: background-color;
  transition-duration: 2s;
}
div:hover {
  background-color: #ff5c00;
}
```

Come si può intuire, il passaggio del colore avverrà in maniera graduale di durata due secondi.

Ci sono diverse tipi di proprietà per impostare le transizioni, elencheremo qui di seguito questi comandi con una breve descrizione:

- transition-property

  Definisce le proprietà a cui verrà assegnata la transizione. Senza però la definizione delle altre proprietà non effettua alcuna transizione.

- transition-duration

  Definisce la durata della transizione, ovvero il tempo che la transizione impiegherà per passare dallo stato iniziale a quello finale e viceversa. Il valore è espresso in secondi e di default è settata a 0.

- transition-delay

  Permette di ritardare l'esecuzione della transizione del numero di secondi passati come parametro.

- transition-timing-function

  Descrive come i valori intermedi usati durante la transizione vengono calcolati. Anziché utilizzare una ripartizione lineare del tempo è possibile utilizzare delle funzioni che cambiano velocità durante la transizione.

3.7.5.2 Trasformazioni

Le trasformazioni mettono a disposizione dei metodi per la manipolazione 2D su elementi contenuti nella pagina. Per far ciò è necessario definire un sistema di coordinate che ha origine nell'angolo in alto a sinistra dell'elemento stesso. Le trasformazioni possono anche essere combinate tra loro. È possibile, ad esempio, ruotare e scalare contemporaneamente un'immagine oppure inclinare e capovolgere un oggetto, etc. Le proprietà messe a disposizione sono due:

- transform

  con cui si specificano le trasformazioni da effettuare. Il valore di default è *none*.

- transform-origin

  con cui si specifica il punto di origine da cui avviene la trasformazione. La proprietà richiede due valori, il primo per l'asse delle X mentre il secondo per l'asse Y. I valori possibili possono essere espressi sia in percentuali che tramite le keywords:left, right, center, bottom, top.

Definiamo una semplice lista delle funzioni possibili per la proprietà transform:

- Matrix

  La funzione consente di effettuare una trasformazione in forma di matrice a sei valori nella forma:

  ```
  matrix( a, b, c, d, tx, ty );
  ```

  Dove a, b, c, d costruiscono la matrice e gli ultimi due indicano il valore di traslazione

- Translate

  La funzione consente di traslare l'oggetto in base al valore passato alla funzione. La funzione prende in ingresso uno o due valori che indicano rispettivamente l'asse X e l'asse Y. Se il secondo parametro non viene passato, la traslazione sull'asse Y assume valore zero.

- TranslateX – TranslateY

  Le due funzioni sono del tutto simili a quella precedente con la sola differenza che prendono in ingresso un unico parametro riferito all'asse specifico.

- Scale

  La funzione scale consente di ridimensionare l'oggetto in base ai valori passati come parametri; se viene passato un solo parametro, esso viene assegnato ad entrambi gli assi.

  Se il valore è maggiore di uno, la dimensione dell'oggetto verrà ingrandita mentre, se il valore è compreso tra zero e uno, la dimensione verrà ridotta.

- Scale X – Scale Y

  Come per la funzione translate, anche queste funzioni consentono di ridimensionare l'oggetto solo su uno degli assi.

- Rotate

  Probabilmente la funzione al momento più usata, rotate consente di ruotare l'oggetto in base al numero di gradi passati come parametro.

- Skew

  La funzione distorce l'oggetto per un valore pari al numero di gradi passati come parametro. Se vengono passati due parametri l'inclinazione verrà effettuata su entrambi gli assi, altrimenti solo sull'asse X.

- SkewX – SkewY

  Come per le funzioni precedenti, anche in questo caso entrambe si riferiscono all'inclinazione di uno solo degli assi.

Inoltre è possibile definire più trasformazioni per un singolo oggetto e bisogna porre particolare attenzione all'implementazione differente tra i vari browser. Infatti ognuno di questi prevedere un proprio comando specifico.

3.7.5.3 Animazioni

Per l'implementazione delle animazioni il tutto diventa più complicato. Concentrandoci solamente sull'implementazione lato CSS è necessario specificare due blocchi fondamentali:

```
@-webkit-keyframes 'pulse' {
0% {
  background-color: red;
  opacity: 1.0;
  -webkit-transform: scale(1.0) rotate(0deg);
}

33% {
  background-color: blue;
```

```
  opacity: 0.75;
  -webkit-transform: scale(1.1) rotate(-5deg);
}

67% {
  background-color: green;
  opacity: 0.5;
  -webkit-transform: scale(1.1) rotate(5deg);
}

100% {
  background-color: red;
  opacity: 1.0;
  -webkit-transform: scale(1.0) rotate(0deg);
}
}
```

Ogni animazione CSS deve contenere un blocco di questo tipo. Si tratta di una cosiddetta @-rule, uno di quei costrutti CSS introdotti dal simbolo @. Nelle animazioni CSS3 per introdurre il blocco delle dichiarazioni usiamo @-keyframes. Nell'esempio abbiamo @-webkit-keyframes per via del prefisso proprietario, in altri casi bisognerà specificare il blocco proprietario specifico. Subito dopo @-keyframes va inserito il nome dell'animazione. Nell'esempio è stato usato 'pulse', ma può essere un nome a piacimento. Tutto quello che segue nella dichiarazione va racchiuso tra parentesi graffe. All'interno di queste Troviamo quattro dichiarazioni introdotte dalle percentuali. Ciascuna dichiarazione contiene proprietà CSS che definiscono l'aspetto e/o la posizione dell'oggetto. Ciascuna dichiarazione corrisponde a un keyframe dell'animazione.

In qualunque animazione vanno definiti almeno due stati di un oggetto, quello iniziale e quello finale. Nelle animazioni CSS3 lo stato iniziale va dichiarato con il valore 0% oppure con la parola chiave from. Lo stato finale con il valore 100% oppure con la parola chiave to. Si poteva scrivere la regola vista qui sopra in questo modo:

```
@-webkit-keyframes 'pulse' {
from {
  background-color: red;
```

```
  opacity: 1.0;
  -webkit-transform: scale(1.0) rotate(0deg);
}

33% {
  background-color: blue;
  opacity: 0.75;
  -webkit-transform: scale(1.1) rotate(-5deg);
}

67% {
  background-color: green;
  opacity: 0.5;
  -webkit-transform: scale(1.1) rotate(5deg);
}

to {
  background-color: red;
  opacity: 1.0;
  -webkit-transform: scale(1.0) rotate(0deg);
}
}
```

Chiariamo il significato delle percentuali. Si supponga di creare un'animazione di 10 secondi. Impostando il primo fotogramma chiave al 33%, esso sarà attivo al 33% di 10 secondi, ovvero a 3,3 secondi. In questo modo abbiamo solo impostato i fondamenti dell'animazione senza associarla ad alcun elemento.

L'associazione avviene nel CSS partendo dal selettore legato all'elemento da animare. Noi abbiamo un div con classe .pulsedbox. Per cui, nel CSS, andremo a scrivere:

```
.pulsedbox {
-webkit-animation-name: pulse;
-webkit-animation-duration: 4s;
-webkit-animation-iteration-count: infinite;
-webkit-animation-timing-function: ease-in-out;
}
```

Elenchiamo anche in questo caso un insieme di proprietà utili nelle animazioni:

- animation-name

  Con la proprietà animation-name si definisce il nome dell'animazione da associare ad un elemento. Il nome deve corrispondere a quello impostato nella regola @-keyframes. Nel nostro caso, infatti, abbiamo usato 'pulse'.

- animation-duration

  Questa proprietà serve a impostare la durata dell'animazione. Il valore va espresso in secondi.

- animation-iteration-count

  La proprietà animation-iteration-count è usata per impostare il numero di volte che un'animazione sarà ripetuta. Il valore può essere un numero intero pari o superiore a uno oppure la parola infinite, con cui si crea una sorta di loop infinito. Il valore di default è uno.

- animation-timing-function

  Questa proprietà è del tutto analoga alla proprietà transition-timing-function vista nelle transizioni CSS3: descrive come i valori intermedi usati durante l'animazione vengono calcolati.

- animation-direction

  Possiamo far sì che l'animazione ripetuta venga eseguita in ordine inverso. Basta assegnare alla proprietà il valore reverse.

- animation-delay

  Per impostare invece un ritardo nell'esecuzione dell'animazione, possiamo usare un valore espresso in secondi.

Capitolo 4

Mobl

4.1 Introduzione

Nel seguito del capitolo si analizzeranno le specifiche del framework e si approfondiranno le strategie relative a come il middleware Mobl affronta le problematiche discusse nella prima sezione del precedente capitolo, per rendere l'implementazione e la progettazione del software più agile e veloce. In seguito si effettueranno delle sperimentazioni su questo sistema per testarne le particolarità ed ottenere un riscontro pratico delle sue caratteristiche.

Mobl si presenta come un nuovo linguaggio dichiarativo di alto livello, gratuito e open source, su cui costruire Mobile Web Applications. Questo però, è a sua volta basato su una struttura sottostante ben definita, che di conseguenza classifica tutto il sistema come un Mobile Frameworks. La vera e propria peculiarità di questa piattaforma risiede proprio nel suo obbiettivo: cioè far convergere all'interno di un unico linguaggio aspetti di design, styling, data modelling, query e logica applicativa.

4.2 Da Applicazioni Web Mobile ad Applicazioni Mobile

Abbiamo già accennato come le applicazioni web mobile affrontano l'indipendenza dal web: tramite il Data caching e l'application caching. E' proprio grazie a questa tecnologia che l'applicazione può ottenere tre benefici fondamentali:

1. Utilizzo Offline

   E' senza dubbio la più importante caratteristica che rende le applicazioni web similari con i software locali e quindi le pone come la più valida alternativa platform-independent.

2. Prestazioni

   Nonostante non si possano mai raggiungere le prestazioni tipiche di un software nativo, con l'utilizzo della cache si riescono ad ottenere ottimi risultati.

3. Minore traffico di Rete

   Non è necessario uno scambio intensivo di dati tra i dispositivi e il web, ma basterà sincronizzare solamente i cambiamenti avvenuti.

Con l'introduzione di queste funzionalità integrate nel linguaggio, che sono state discusse nel capitolo riguardante le applicazioni web, è possibile ottenere delle vere e proprie applicazioni puramente mobile e indipendenti dal web. Infatti tutta la parte logica dell'applicazione viene salvata all'interno di un database SQL le cui dimensioni dipendono dal dispositivo, ma in genere si tratta di una quantità di dati molto piccola. La gestione di questo database deve essere fornita dagli sviluppatori tramite il linguaggio SQL a basso livello, e questo richiede conoscenze specifiche. Approfondiremo questo aspetto quando si parlerà del modello dei dati di mobl e quindi come affrontare questo problema.

Per attivare esplicitamente la funzionalità offline e di conseguenza il data e application caching nelle applicazioni Mobl è necessario specificare nell'apposito file di configurazione *config.mobl* il seguente settaggio:

```
offline true
```

E' altamente consigliato utilizzare inserire questo settaggio solo a progetto ultimato e non durante lo sviluppo. Purtroppo non è utilizzabile per applicazioni Mobl che effettuano chiamate AJAX.

Un secondo metodo per rendere l'applicazione Mobl indipendente dal web consiste nell'integrazione con il framework phonegap che, partendo dall'applicazione web generata da Mobl, ne genera una nativa per le piattaforme specificate. Anche questo è possibile tramite l'inserimento del settaggio nel file *config.mobl* nel seguente modo:

```
native {
    username "email@example.com"
    password "password"
    platforms android, blackberry, symbian, webos, ios
}
```

dove *username* e *password* indicano le credenziali del proprio account Phonegap mentre in *platforms* sono elencate le piattaforme su cui dovrà essere disponibile l'applicazione.

4.3 Architettura delle Applicazioni Mobl

Il passo successivo, una volta risolto il problema dell'esecuzione offline, consiste nell'individuare quale sia effettivamente l'architettura migliore per costruire software, anche di una certa complessità, in modo che il processo di produzione risulti il più veloce e efficente possibile. Il modo migliore per raggiungere tale scopo è partire da un'architettura già ampiamente diffusa e popolare per le sue caratteristiche, e cercare di adattarla alle tecnologie che compongono l'applicazione.

Il più diffuso ed importante pattern architetturale è il MVC. Questo pattern è stato adottato dai principali framework per la programmazione object-oriented, come ad esempio: Java, Objective-C, .Net, PHP, Python e Ruby. La struttura che caratterizza questo pattern comprende 3 elementi principali:

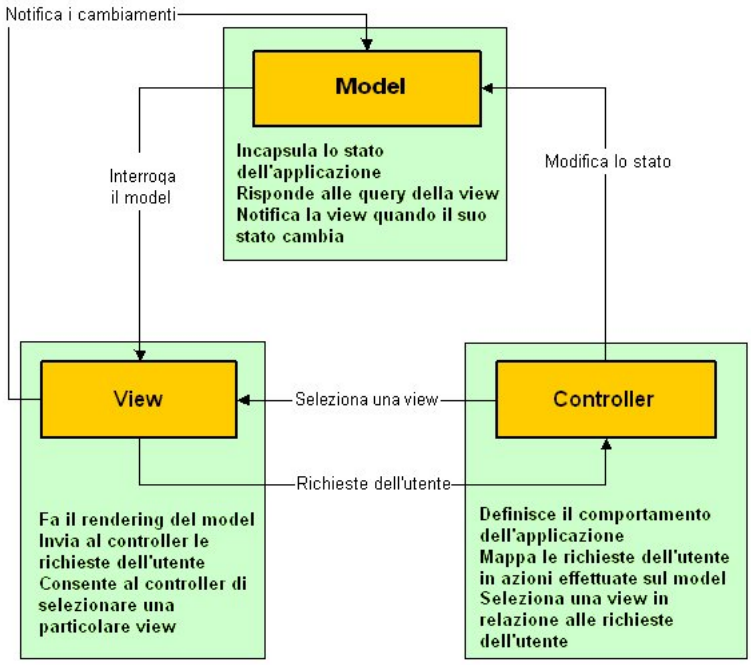


Figura 4.1: Architettura MVC

- Model

  Fornisce i metodi per accedere ai dati utili per l'applicazione

- View

  visualizza i dati contenuti nel model e si occupa dell'interazione con utenti e agenti

- Controller

  riceve i comandi dell'utente (in genere attraverso il view) e li attua modificando lo stato degli altri due componenti

Questo schema, fra l'altro, implica anche la tradizionale separazione fra la logica applicativa, a carico del controller e del model, e l'interfaccia utente a carico del view.

Nel nostro caso però occorre modificare quest'architettura per essere utilizzata con i linguaggi web che sono stati concepiti per un architettura

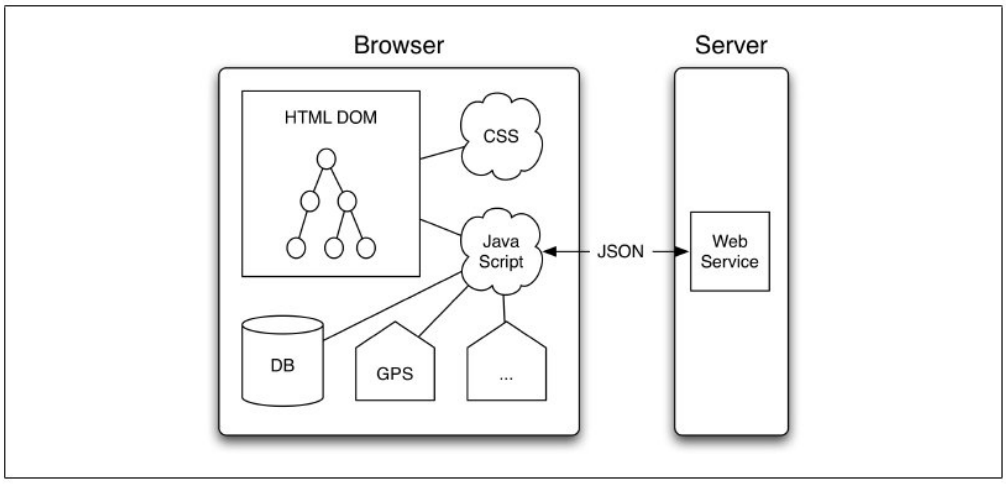


Figura 4.2: Architettura Tipica di un'Applicazione Web Standard

tradizionale RestFull (*Request Orinented*): quando arriva una richiesta da un client, questa viene gestita dal server che la elabora e ne restituisce il risultato secondo gli standard web. In particolare si è notato che con l'utilizzo del MVC, il controller risulta di difficile implementazione perchè non è possibile ottenere l'iterazione Model/View senza dover effettivamente modificare anche il controller(si noti la figura 4.3). Questo inconveniente causa l'inserimento di linee di codice aggiuntive ed inutili, infatti il controller viene richiesto in più ambiti rispetto al suo naturale ruolo, ad esempio nella lettura degli input.

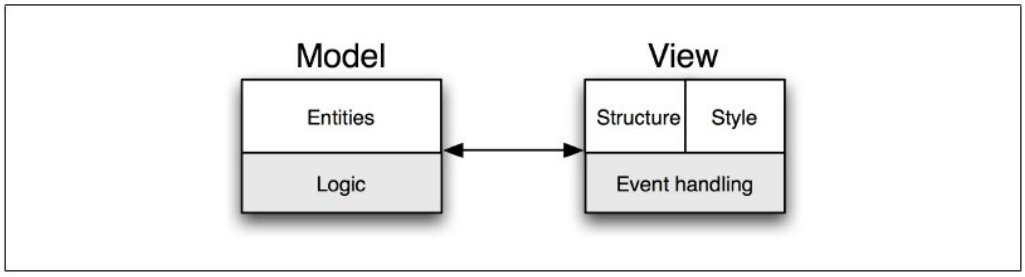


Figura 4.3: Architettura Model-View

4.3.1 Model-View

In Mobl si è deciso quindi si utilizzare l'architettura Model-View, spezzando il controller in 2 parti e integrandolo nei restanti due elementi. Con questa strategia è possibile evitare la ridondanza del codice e semplificare quindi lo sviluppo, ma sopratutto cambieranno tutte le iterazioni tra il sistema e l'utente. Infatti, mentre precedentemente era il controller ad incaricarsi di modificare la view con i dati del model sotto richiesta dell'utente, ora tutto parte dalla view, infatti al suo interno vengono gestiti gli input da parte dell'utente e successivamente chiamare un metodo del model oppure un altra view. Le view sono parametrizzate con uno o più oggetti del modello da rappresentare, e possono a loro volta richiedere, sempre al model, che gli vengano forniti i dati necessari.

Nella parte del model invece si ha la gestione del database, nonchè la sincronizzazione delle modifiche nel caso fosse disponibile una connessione con il server. Infine sarà compresa sempre nell'entità model la parte di data buolding che si occupa di stabilire la connessione diretta tra model e view in maniera automatica.

4.4 Data Model

Si sono già discussi i problemi relativi alla gestione del database locale tramite l'HTML5 Data Persistent, che utilizza un SQL di basso livello, e il problema di sicurezza che ne deriva (4.2). In questa sezione si vuole

analizzare come Mobl cerca di risolvere queste problematiche e come il tutto viene inserito all'interno del linguaggio.

Mobl definisce delle astrazioni di dominio per definire le strutture dei dati in modalità dichiarativa, in questo modo si evita di dover utilizzare l'SQL e risulta più agevole per i programmatori gestire il database, inoltre la gestione dei dati in modo persistente viene effettuato da Mobl a runtime ed in maniera trasparente.

Esistono due tipi di dati nelle applicazioni Mobl: dati persistenti e dati transienti. I primi, salvati in degli oggetti chiamati *entity*, ad ogni modifica che subiscono vengono salvati all'interno del database tramite il sistema Mobl, mentre i secondi riguardano i dati volatili che vengono persi alla chiusura del programma oppure quando ne si perde il riferimento all'interno del software. In questa sezione ci occuperemo principalmente della prima tipologia, mentre la seconda verrà discussa in un secondo momento. La dichiarazione del modello dei dati è definito da:

- Entity

  Indicano gli oggetti effettivamente presenti nel database e sono composte da: nome, proprietà e delle funzioni associate utili per definire il comportamento logico relazionato all'entità

- Proprietà (Nome, Tipo, Annotazioni[opzionali])

  Si tratta delle informazioni dell'entità e ne indicano la composizione. La particolarità consiste nella possibilità di specificare una proprietà con un tipo collezione di entità(*Collection*)

- Collection

  Rappresenta un gruppo di istanze di entità che possono essere filtrate, ordinate e manipolate. Vengono utilizzate per rappresentare relazioni (uno a molti, molti a molti) o per eseguire query.

- (searchable)

  E' un annotazione opzionale la quale indica che la proprietà deva essere inclusa all'interno delle ricerche full-text.

Di seguito si potrà osservare la struttura sintattica del modello dei dati di Mobl e un esempio della dichiarazione delle entità.

Listing 4.1: Sintassi Data Model Mobl

```
Def ::= "entity" ID "{" EBD* "}"

EBD ::= ID ":" Type ("("  {Anno ","}* ")")?
        | "static"? "function" ID "(" {FArg ","}* ")" ":" Type
          "{" Stat* "}"

Type ::= ID
        | "Collection" "<" Type ">"
        | "[" Type "]"
        | "(" {Type ","}* ")"

Anno ::= "inverse:" ID
        | "searchable"

FArg ::= ID ":" Type
        | ID ":" Type "=" Exp
```

Listing 4.2: Esempio della Dichiarazione di un Entità

```
entity Task     {
        name    :       String (searchable)
        done    :       Bool
        due     :       DateTime
        category:       Category (inverse: tasks)
        tags    :       Collection<Tag> (inverse: tasks)

        function postpone(days : Num) {
                this.due = DateTime.create(
                this.due.getFullYear(),
                this.due.getMonth(),
                this.due.getDate() + days);
        }
        static function import(user : String,pw : String){
                var tasksJSON = httpRequest ("/export?user="
                 + user + "%pw=" + pw);
                foreach(t in taskJSON) {
                        add(task.formSelectJSON(t);
                }
        }
}
```

4.4.1 Tipi di Base

In questa sottosezione sverranno brevemente introdotti i tipi, relativi alle proprietà delle entità, di base presenti all'interno delle librerie di Mobl.

- Void

  Questo tipo rappresenta il nulla e viene principalmente utilizzato per le funzioni o le schermate che non restituiscono alcun risultato al termine della loro esecuzioni. Se in una funzione viene omesso il tipo di ritorno, questo viene sottinteso come void.

- Object

  E' il tipo più generale possibile da cui discendono tutti gli altri tipi di dati, possiede infatti solamente il metodo *ToString()* per la rappresentazione di qualsiasi tipo di parametro questo formato.

- String

  Rappresentano una successione di caratteri che, una volta definita, non può essere modificata.

- Num

  Rappresenta i numeri, sia di tipo integer che di tipo float, ed è in grado di adattarsi a seconda che si utilizzino l'uno o l'altro tipo. Le operazioni consentite sono: addizione, sottrazione, divisione, moltiplicazione e modulo.

  Per quanto riguarda operazioni più complicate conviene invece far riferimento ad un altro tipo, Math, che racchiude operazioni più complesse

- Bool

  Tipo booleano che può assumere solo i valori true e false. Può essere manipolato tramite le operazioni di and e or.

- DateTime

  Serve per rappresentare il tempo e la data, è possibile costruire questo tipo da una stringa oppure tramite metodi come *now()* e *getTime()*

- Collection

  Come già introdotto, rappresentano una collezione di oggetti, tipicamente entità, e non preservano un particolare ordine tra i suoi componenti, che va specificato tramite un apposito filtro. Queste sono definite per ogni entità e specificate dall'utente stesso, ma ne esistono anche di generiche come *Entity.All()* o per ogni proprietà uno a molti e molti a molti. Spesso sono di tipo virtuale, cioè rappresentano una certa tipologià di oggetti senza effettivamente contenerli, ma è all'atto della chiamata di una collezione (tramite il metodo *get()*) che il risultato viene calcolato e restituito tramite un array fisso.

- Array

  Gli array sono una vera e propria lista di oggetti, di solito di composti da dati semplici. Sono utilizzati specialmente perchè sono di facile manipolazione, ma risulta difficile effettuare operazione di filtraggio e ordinamento.

- Tuple

  Risultamo molto utili quando bisogna utilizzare una moltudine di dati che non hanno una propria categoria tra i tipi di base.

Un aspetto molto importante che vale la pena specificare è la possibilità in mobl di specificare una serie di propri tipi di dati, che poi possono essere utilizzati.

La sintassi generale per definire un proprio dato è la seguente:

```
type <TypeName> {
<properties>
}
```

4.4.2 Query

La gestione delle query avviene in modo automatico all'interno del linguaggio Mobl ed in particolare vengono utilizzate le Collection. Come già precedentemente discusso, le collection possono essere filtrate, ordinate e manipolate con appositi metodi, e all'occorrenza restituire il risultato specificato. Un esempio di come questo avvenga è il seguente:

```
Task.all().filter("done", "=", true)
  .order("due", false)
  .limit(10)
```

Nonostante sia possibile utilizzare questo metodo per eseguire una query, si è deciso di implementare una sintassi apposita per la strutturazione delle query. Il motivo che ha portato a questa scelta consiste nella possibilità di avere, in maniera del tutto statica, un controllo sugli errori e il supporto per il completamento automatico del codice da parte dell'IDE di supporto. Di conseguenza l'esempio che si è analizzato si tramuta in:

```
Task.all() where done == true
order due desc
limit 10
```

Infine, un aspetto molto importante, è la possibilità di riutilizzare le query già costruite potendo salvare le collection virtuali all'interno di variabili, che possono anche essere passate a funzioni. Ad esempio, è possibile utilizzare un strutturare un metodo all'interno di una entity che restituisca una collection ed infine questa può essere richiamata all'interno della User Interface, infatti, come già specificato, i risultati delle collection vengono calcolati solamente quando queste vengono utilizzate. Infine vediamo uno schema riassuntivo della sintassi utilizzata per definire le query in Mobl.

Listing 4.3: Sintassi per la Definizione di Query in Mobl

```
Exp     ::= Exp Filter+

Filter  ::= "where" SetExp
        | "order" "by" OrderExp
        | "limit" Exp
        | "offset" Exp

OrderExp ::= ID | ID "asc" | ID "desc"

SetExp  ::= ID "==" Exp | ID "!=" Exp
        | ID "<" Exp | ID "<=" Exp
        | ID ">" Exp | ID ">=" Exp
        | ID "in" Exp | ID "not" "in" Exp
        | SetExp "&&" SetExp
```

4.5 User Interface

Per la costruzione di un interfaccia utente nelle comuni web application risulta necessario utilizzare le tecnologie HTML5 e CSS. Insorge però un problema fondamentale nella costruzione di interfacce utilizzando questi linguaggi: HTML e CSS, non definendo gli elementi all'interno delle loro interfacce come oggetti, non consentono di riutilizzare un elemento già precedentemente specificato ed eventualmente parametrizzarlo, inoltre è necessaria una ridefinizione nella struttura delle interfacce per l'adattamento con la grandezza degli schermi ed in generale per la diversa natura dei dispositivi web. Infatti, per come sono costruite le odierne applicazioni mobile, le interfacce sono organizzate ad albero o a steak e quindi con una precisa organizzazione e sequenza. Purtroppo le tecnologie web non consentono di mantenere questa astrazione.

Queste avversità relative alle UI vengono affrontate anche da altri framework già introdotti ed ogniuno di questi introduce una sua soluzione, ad esempio: JQueryMobile e JQTouch rendono possibile, estendendo le funzionalità del CSS il riutilizzo degli elementi, ma non si occupano della ridefinizione della struttura, Altri come Secha Touch invece astraggono dall'HTML e consentono di costruire interfacce tramite Java o Javascript API, ma facendo così corrispondere un linguaggio dichiarativo con uno imperativo, il che può portare ad incomprensioni.

In Mobl invece si è deciso di affrontare il problema definendo una interfaccia dichiarativa ed utilizzando due elementi principali: *Screen & Controls*. Tramite l'utilizzo di questi due elementi si introduce una gerarchia nella costruzione delle interfacce e si rende possibile l'utilizzo di un elemento grafico già specificato in più parti del programma.

Gli Screen sono dei contenitori per i Control e rappresentano appunto una ben determinata schermata, possono essere chiamati come effetto di un particolare evento ed in particolare si differenziano dai Control perchè prevedono un valore di ritorno. Queste caratteristiche si sposano perfettamente con la struttura delle applicazioni mobile che conosciamo oggi e consentono di avere una visione chiara e d'insieme dell'applicazione. La struttura di Screens e Controls è la stessa, entrambi sono composti da: un nome, una serie di argomenti e un corpo.

Esattamente come dei comuni metodi, all'interno del corpo viene definito il comportamento dell'elemento, per far ciò si possono utilizzare: Variabi-

li locali, HTML Tags, Chiamate ad altri controlli, cicli e condizioni per il rendering della UI. Particolare attenzione va posta sull'utilizzo delle variabili, infatti queste vengono settate e dichiarate per tenere traccia di una particolare azione compiuta dell'utente e di conseguenza attivare delle parti condizionate a seconda della situazione, per decidere che cosa visualizzare in base agli avvenimenti trascorsi. Una caratteristica interessante è l'inclusione, all'interno del corpo, degli HTML Tags che consentono di modificare direttamente il DOM all'occorrenza. Una nota di fondamentale importanza riguardante gli HTML Tags consiste nella possibilità di parametrizzare tramite: numeri, espressioni, variabili e chiamate di sistema, oltre alle comuni stringhe le loro proprietà. Questo consente di rendere molto più flessibile l'utilizzo dell HTML e risolve il problema introdotto ad inizio sezione. Infine, ma non per importanza, la gestione degli argomenti, che vengono passati da funzione a funzione per argomento, consentendo così sia a Screens che Controls di poterli modificare a piacimento.

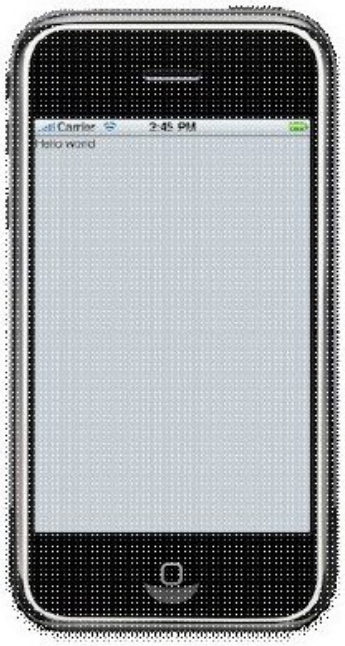
Di seguito sarà possibile prendere visione della sintassi utilizzata per la costruzione dell'interfaccia dichiarativa di mobl e di semplici esempi riguardanti Screens e Controls.

Listing 4.4: Sintassi dell'Interfaccia Dichiarativa di Mobl

```
Def ::= Anno* "control" ID "(" {FArg ","}* ")" "{" SE* "}"
        | Anno* "screen" ID "{" {FArg ","}* "}" ":" Type "{" SE*
            "}"

SE ::= "<" HTMLID HtmlArg* ">" SE* "</" HTMLID ">"
          | Exp "(" {NamedExp ","}* ")" "{" SE* "}"
          | "var" ID "=" Exp
          | "list" "(" ID "in" Exp "}" "{" SE* "}"
          | "when" "(" Exp ")" "{" SE* "}"

HtmlArg ::= ID "=" Exp
        | "body" "=" Exp

NamedExp ::= Exp
        | ID "=" Exp

Anno ::= "@when" Exp
```

4.5.1 Screens

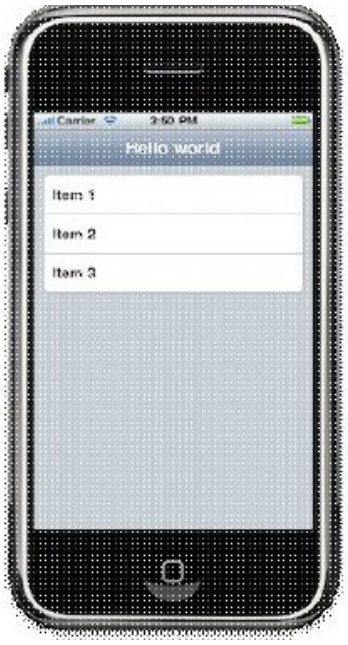
Ogni applicazione Mobl necessita di uno Screen principale che sarà la prima schermata che l'applicazione mosterà una volta iniziata la sua esecuzione. Questo primo elemento dovrà chiamarsi obbligatoriamente *root*. Ecco un semplice esempio (*Hello World*) per capire il funzionamento base di questa piattaforma:




```
application hello

screen root() {
        "Hello world!"
}
```

Come si può osservare dall'esempio appena illustrato, si riconoscono subito alcune caratteristiche descritte nel capitolo precedente che riguardano gli Screen, infatti si nota il nome dello screen ed il suo corpo. In questo caso è stato omesso un valore di ritorno, questo significa che esso coincide con il valore *void*.

Ora mostriamo un esempio di una complessità maggiore aggiungendo un Header e tre oggetti all'interno della schermata. Per far ciò è necessario innanzitutto importare una libreria di Mobl che acconsente di utlizzare gli Header e modificare propriamente il corpo dello Screen di root per apportare le modifiche volute. Di seguito è possibile verificarne il risultato.




```
import mobl::ui::generic

screen root(){
        header("Hello World")
        group{
                item ( "item 1" )
                item ( "item 2" )
                item ( "item 3" )
        }
}
```

4.5.2 Controls

Listing 4.5: Esempio della Dichiarazione di un Entità

```
import mobl::ui::generic

control headerGroup(title : String) {
        header (title)
        group {
                elements()
        }
}

control itemNumber (n : Num) {
        item { "item" label(n) }
}

screen root(){
        headerGroup ("Hello World") {
                itemNumber(1)
                itemNumber(2)
                itemNumber(3)
        }
}
```

Per mostrare un esempio molto semplice sull'utilizzo dei Controls e sul loro funzionamento si può modificare l'esempio precedente. Definiamo un Control molto semplice che, in base al numero passato come argomento, inserisce un oggetto, ed inseriamolo all'interno della nostra applicazione, così impiegato si può notare che il risultato è il medesimo dell'esempio precedente.

L'importanza di questa modifica consiste nel sottolineare come sia possibile impiegare i Controls all'interno di altri elementi, in questo caso dell'elemento groups e come richiamino molto il concetto di metodo nei linguaggi di programmazione ad oggetti tradizionali.

4.5.3 Variabili

Screen e controls sono in grado di definire delle variabili che poi possono essere utilizzate all'interno dell'interfaccia grafica. In particolare ci sono due tipi di variabili:

- Variabili Regolari: Queste variabili possono essere editate e lette.

- Variabili Derivate: Possono solo essere lette, derivano da altre variabili.

Le variabili regolari si comportano come le variabili classiche della programmazione ad oggetti, possono quindi essere passate come argomento e vengono dichiarate specificando: `tipo NomeVariabile = Valore;`

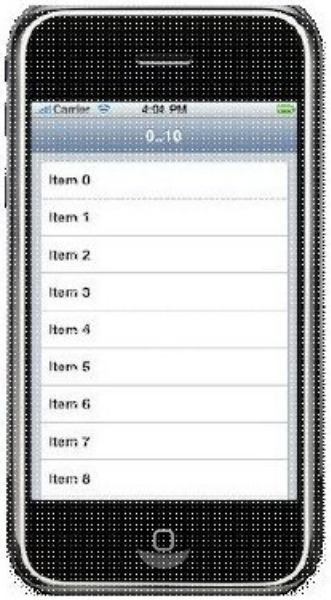
Le variabili derivate invece si differenziano perchè si autoaggiornano in base al cambiamento di una particolare espressione, infatti la loro sintassi è la seguente `tipo variabile <- expressione`.

Per chiarire il funzionamento di queste variabili si può osservare un ulteriore esempio che prevede l'inserimento di un valore da parte dell'utente, che a sua volta serà conservato all'interno della variabile `n`. Da questa variabile, se ne costituirà una derivata chiamata `nSquared` che ha il compito di calcolare il quadrato del valore di `n`.

In questo esempio si può osservare come venga gestito automaticamente dalla piattaforma l'aggiornamento della UI, una volta avvenuto un particolare evento. Questa caratteristica principale della piattaforma Mobl verrà discussa in seguito.


```
import mobl::ui::generic

screen root(){
        header("Number")
        var n = 0
        var nSquared <- n*n

        group{
                item { inputNum(n,
                    label="N:" ) }
                item { label(nSquared)
                     }
        }
}
```

Infine si allegano due semplici esempi riguardanti l'utilizzo dei comandi: `list` e `when`.




```
import mobl::ui::generic

screen root(){
        header("0..10")
        group{
                list (n in range
                   (0,10)) {
                 item { "item"
                    label(n) }
                 }
        }
}
```

```
import mobl::ui::generic

screen root(){
        header("Check!")
        var b = false
        group{
                item { checkbox(
                   b, label="
                   Check me!")
                   }
        }
        when (b) {
                "The checkbox is
                    checked!"
        }
}
```

4.5.4 Higher-Order Control

Gli Higher-Order Constrols sono controlli che si differenziano da quelli già introdotti perchè si trovano ad un livello di astrazione maggiore e che quindi possono utilizzare gli altri controlli come argomenti. Sostanzialmente si tratta di controlli di alto livello. Tutto questo è possibile grazie alle librerie che Mobl mette a disposizione e che facilitano il riutilizzo dei controlli stessi. Vediamo due esempi significativi di questa nuova tipologia di strumenti che spesso si utilizzano in ambito mobile: tabSet e masterDetail.

4.5.4.1 tabSet

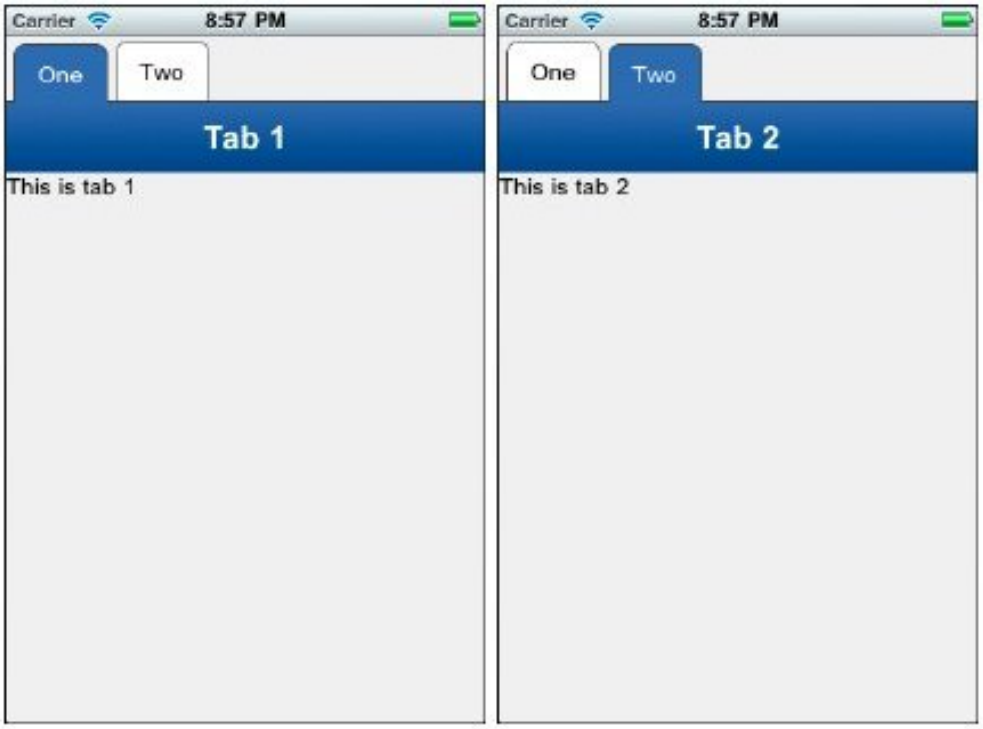
La schermata di root richiama il tabSet con una lista di tab da rappresentare e quali di queste è effettivamente attiva di default. Per rappresentare un tab si utilizzano due elementi: una stringa che rappresenta il nome del tab e un controllo che rappresenta il contenuto del tab stesso. Andando ad analizzare il l'implementazione del tabSet si può notare un nuovo tipo di controllo specifico per lo stile (*block*) che consente di decidere se una particolare tab è: attiva, inattiva, visibile o non visibile.

Listing 4.6: Implementazione di un tabSet

```
control tabSet (tabs : [(String,Control)],activeTab : String) {
        list ((tabName, tabControl) in tabs) {
                block (onclick={ activeTab = tabName; }, style =
                     activeTab==tabName ? activeTabButton :
                    inactiveTabButton){
                         label(tabName)
                }
        }
        list ((tabName, tabControl) in tabs) {
                block (activeTab==tabName ? visibleTab :
                    invisibleTab) {
                         tabControl()
                }
        }
}
```

Listing 4.7: Esempio dell'utilizzo di un tabSet

```
control tab1() {
        header ("Tab 1")
        label ("This is tab 1")
}

control tab2() {
        header ("Tab 2")
        label ("This is tab 2")
}

screen root(){
        tabSet([("One",tab1), ("Two",tab2)],defaultTab="One")
}
```



4.5.4.2 masterDetail

Questo controllo implementa un pattern piuttosto comune nei dispositivi mobili e consiste nel visualizzare una lista di oggetti e, una volta che ne viene scelto uno, ne vengono visualizzati i dettagli. Una particolarità di questo controllo consiste nel diverso comportamento in base all'orientamento del dispositivo, infatti, nel caso di un display ristretto si utilizzeranno due schermate per ogni fase con la possibilità di spostarsi tra le schermate come descritto in seguito, mentre nel caso di display più ampi viene visualizzata la lista degli oggetti sulla sinistra e nella restante parte dello schermo, i dettagli della selezione dell'apposito elemento. Per rappresentare rispettivamente un elemento all'interno del masterDetail o per specificare il contenuto di un elemento specifico si introducono due nuovi controlli chiamati: taskItem e taskDetail. In particolare all'interno di questi due sotto-controlli saranno presenti le strutture che si avranno nelle loro corrispondenti schermate. Infine, nella schermata in cui viene richiamato il masterDetail, questo possiederà i seguenti argomenti: una lista di task, e i controlli sopra citati.

Listing 4.8: Implementazione di un masterDetail

```
control masterDetail(items : Collection<?>, masterItem : Control
    <?>,detail : Control<?>){
        group {
                list(it in items) {
                        item(onclick={detailScreen(it,detail);})
                             {
                                 masterItem(it)
                        }
                }
        }
}

screen detailScreen(it : ?, detail : Control<?>) {
        header("Detail") {
                backButton()
        }
        detail(it)
}
```

```
control taskItem(t : Task) {
        checkBox(t.done, label=t.name)
}

control taskDetail(t : Task) {
        textField(t.name)
        datePicker(t.due)
}

screen root(){
        header("Tasks")
        masterDetail(Task.all() order by due desc, taskItem,
            taskDetail)
}
```

Figura 4.4: Esempio dell'utilizzo di un masterDetail

4.6 Data Building e Reactive Programming

Come già introdotto nella architettura di Mobl () in questa sezione si prenderà in esame la parte architetturale che si occupa di collegare tra di loro le parti relative alle View con quelle del Model.

Lo scopo che ci si prefigge in Mobl consiste nella costruzione di una *Reactive User Interface*, cioè ottenere un mutuo aggiornamento tra Model e View ogni qualvolta una delle due parti subisce una modifica; senza che lo sviluppatore debba quindi occuparsi di propagare tale modifica manualmente anche sul resto dell'applicazione.

Ecco esempio di come vengano propagate le modifiche attraverso la piattaforma:

```
var name = "John"
<input type="text" value = name/>
<span body="Hello, " + name/>
```

In questo caso si avrà una interfaccia che predispone l'inserimento da parte dell'utente di un input. Una volta effettuato l'inserimento e quindi la modifica del valore nella variabile, questa si propagherà all'interno del sistema per poi venire automaticamente visualizzata nel controllo successivo. Si possono inoltre utilizzare anche delle condizioni o cicli che controllano lo stato delle variabili modificate in modo da adattare di conseguenza lo stato della UI. Di seguito possiamo osservarne un esempio tramite la condizione *when* che mostra come il messaggio posto sotto condizione venga visualizzato solo se questa è soddisfatta.

```
var name = "John"
<input type="text" value=name/>
when (name.length < 3) {
        <span body="Name should be ad least three characters"/>
}
```

Tutto questo è possibile tramite l'utilizzo di un famoso pattern architetturale: il *Pattern Observer*. Infatti ogni dato in Mobl è osservabile, mentre i componenti della UI si registrano come osservatori di ogni loro cambiamento.

4.7 Navigazione attraverso la UI

In questa sezione si osserverà come avviene la navigazione attraverso l'interfaccia utente in Mobl. Le modalità di navigazione nell'ambiente web risultano molto differenti rispetto a quelle riguardanti il mobile. Nel caso mobile infatti il percorso risulta molto più vincolato. Solitamente si ha un'organizzazione tipicamente ad albero e la navigazione è possibile muovendosi avanti ed indietro le varie strade consentite. Un esempio di quanto descritto si ha nei dispositivi che utilizzano IOS, infatti in questo caso si utilizza uno steak; si procede di pagina in pagina e quando si torna indietro viene ricaricata la pagina precedente(*push/pop*).

Questa logica è famigliare alla maggioranza degli sviluppatori software e assomiglia molto alla logica che risiede dietro le chiamate di sistema sincrone. Per far si che una normale web application adotti questo tipo di politica è necessario impostare manualmente il comportamento dell'applicazione stessa. Fortunatamente in Mobl invece il tutto è già predisposto automaticamente nella piattaforma stessa, infatti tramite l'utilizzo dell'astrazione degli screen(4.5.1) e la loro possibilità di restituire un valore di ritorno alla loro terminazione, quindi adottano appunto un comportamento tipico delle chiamate sincrone, la navigazione attraverso il software risulterà esattamente identico a quanto capita per le applicazioni native.

```
screen root(){
        var phrase = ""
        header("Tasks"){
                button("Add", onclick=( addTask(); ))
        }
        searchBox(phrase)
        group {
                list (t in Task.search(phrase) limit 20){
                        item {
                                checkBox (t.done, label=t.name)
                        }
                }
        }
}
screen addTask(){
        var t = Task()
        header("Add") {
                button("Done", onclick={ add(t); screen return
                    ;})
```

```
        }
        textField(t.name)
        datePicker(t.due)
}
```

Figura 4.5: Esempio della Navigazione Attraverso gli Schermi

4.8 Styling

Per quanto riguarda lo stile delle applicazioni e quindi il layout generale applicativo è logico pensare di utilizzare gli strumenti del web quali HTML e CSS, che già appunto vengono impiegati per la gestione dei siti. Nonostante questa soluzione sia generalmente corretta, il CSS presenta una particolare problematica che viene affrontata nella piattaforma Mobl e corretta.

Il vincolo che il CSS presenta consiste nella mancanza di parametrizzazione nei suoi selettori. Un selettore, come è noto, definisce varie peculiarità di stile che in seguito vengono adottate da uno o più elementi HTML, ma la mancanza di parametrizzazione consiste nell'impossibilità di definire il valore delle proprietà specificate all'interno del selettore CSS tramite un argomento definito sempre a livello di tags HTML. Mobl consente, oltre a tutte le specifiche CSS classiche, anche di adottare costanti, espressioni e altri mix di style, questo è possibile tramite l'utilizzo di un'estensione del CSS chiamata SASS.

In Mobl, a differenza dell'HTML, sono i controlli a possedere un argomento di tipo style e che quindi adottano un particolare stile definito. Con

l'aggiunta quindi della parametrizzazione è possibile adottare delle costanti e, con la loro semplice modifica, cambiare tutte lo stile di tutte quelle parti dipendenti da essa.

Listing 4.9: Sintassi per l'Utilizzo dello stile

```
Def     ::= "style" ID "{" StyleProp* "}"
         | "style" "mixin" ID
           "(" {StyleFarg ","}* ")"
           "{" StyleProp* "}"
         | "style" "$" ID "=" StyleVal

StylePro        ::= ID "=" StyleVal* ";"
        | ID "(" {StyleVal"," }* ")" ";"

StyleVal        ::= CSSSTYLEVALUE
        | "$" ID
        | "$" ID "." "r"
        | "$" ID "." "g"
        | "$" ID "." "b"
        | StyleVal "+" StyleVal
        | StyleVal "*" StyleVal
        | StyleVal "-" StyleVal
```

Poniamo particolare attenzione per quanto riguarda la parametrizzazione osservando un esempio notevole di come l'utilizzo delle costanti e la loro modifica all'interno dell'applicativo possano effettivamente facilitare la costruzione dello stile e soprattutto rendere rapida una particolare modifica.

Listing 4.10: Dichiarazione degli stili

```
style $baseColor = rgb(72,100,180)
style $textButtonColor = rgb($baseColor.r-50,$baseColor.g-50,
   $baseColor.b-50)
style buttonStyle{
        color: $textButtonColor;
        ...
}
```

Figura 4.6: Esempio di come la modifica di una costante influisce sugli stili

4.9 Limitazioni del Linguaggio

Nonostante questo linguaggio e piattaforma presenti molte caratteristiche interessanti su ogni punto della programmazione, purtroppo ancora presenta molte lacune che sono comuni anche con molti altri frameworks che, come Mobl, si approcciano alla problematica principale della programmazione mobile platform-independent. Elenchiamo brevemente queste lacune:

- Data/Syncronization

  Deve ancora essere fatta manualmente dagli sviluppatori tra web server e applicativo, in seguito si vuole rendere questa sincronizzazione trasparente e integrata nel linguaggio.

- Portable e Look and Feel

  Mobl è portabile su tutti i dispositivi che supportano HTML5, ma la UI non supporta il Look and Feel del particolare enviroment incui viene inserita l'applicazione. E' possibile risolvere questo problema con le @when annotations, ma purtroppo non molte piattaforme non sono supportate.

- Performance

  Per la loro natura web, le applicazioni sviluppate in questo modo, saranno sempre meno performanti delle loro controparti native. Grazie al caching e al crescente aumento delle prestazioni dei dispositivi mobile, questa differenza si assottiglia sempre più. Si ha la possibilità infine di impostare il compilatore Mobl in modo che aumenti al massimo le performance eliminando definizioni inutilizzate, spazi vuoti e rinominando le variabili affinchè i dati da scaricare sul web siano meno.

- Distribuzione

  Le applicazioni Mobl non sono normali applicazioni, quindi non sono individuabili tramite un URL specifico, di conseguenza non indicizzabili tramite i comuni motori di ricerca.

- Completo Utilizzo del Device

  Purtroppo essendo applicazioni derivanti dal web difficilmente saranno presenti tutte le API utili per accedere agli strumenti che i vari smartphone mettono a disposizione. In particolare grazie alle API Javascript fortunatamente la maggior parte di questi strumenti sono supportati, ma alcuni invece rimangono ancora inaccessibili, ovviamente tutto dipende dal dispositivo. Esiste comunque un modo per ovviare a questa mancanza che consiste nel progettare un'applicazione ibrida native/web, questo è possibile tramite l'integrazione con altri framework, come ad esempio phonegap, che dispongono di un numero di API maggiore e che consentono di distribuire la propria applicazione web attraverso gli store ufficiali, come se fosse un'applicazione nativa.

- User-Experience

  Ovviamente anche l'User-Experience risulterà molto diversa e limitante rispetto alle applicazioni native, ad esempio non sarà possibile ottenere in una web application lo stesso comportamento in seguito ad uno scrolling che si ottiene invece nei software nativi, oppure risulta particolarmente complicato effettuare un posizionamento fisso degli elementi all'interno dell'interfaccia utente, come ad esempio un header fisso.

Capitolo 5

Analisi delle Prestazioni

In questo capitolo vengono analizzate le performance di un'applicazione mobile realizzata tramite il linguaggio Mobl, integrata successivamente con il framework Phonegap, Confrontandola con un secondo applicativo con le stesse funzionalità e realizzato nativamente su piattaforma Android. In particolare sono stati utilizzati degli appositi test per mettere sotto sforzo le due piattaforme e poterne così trarre conclusioni significative sulle prestazioni. Durante l'esecuzione di questi algoritmi viene calcolato il tempo trascorso per portarli a compimento ed in seguito, equiparando i risultati ottenuti dai due software, si è in grado di confrontarli.

Nelle sezioni successive verranno descritti i algoritmi e sopratutto i motivi che hanno portato alla scelta di questi e successivamente verranno mostrati i risultati ottenuti in ciascuno dei due casi.

5.1 Test Utilizzati

Prima di introdurre i singoli benchmark è utile specificare che cosa effettivamente si è intenzionati a testare della piattaforma Mobl. In particolare bisogna chiedersi quali siano effettivamente le caratteristiche principali che una generica applicazione mobile deve possedere e sulle quali deve garantire particolare reattività. Si sono individuate così due di queste: capacità computazionale e reattività dell'interfaccia. E' necessario infine puntualizzare come queste due proprietà siano strettamente correlate tra di loro, infatti solo con l'aggiornamento dell'interfaccia l'utente percepisce il cambiamento, che a sua volta è naturale conseguenza della fine della computazione.

Alla luce di queste considerazioni si sono implementati questi quattro algoritmi:

1. Calcolo delle Cifre Decimali del Pigreco

   Per questo test si è implementato l'algoritmo di Gauss-Legendre ed è stato scelto per verificare puramente le capacità computazionali delle due piattaforme.

   Per Costruire questo semplice test è stato necessario solamente un parametro numerico in ingresso che indica il numero di iterazioni dell'algoritmo. Purtroppo in questa verifica, a causa della limitata disponibilità di numeri decimali da parte delle due piattaforme, già dalla terza iterazione non si verificano cambiamenti nel risultato.

2. Approssimazione della Funzione Seno

   In questo caso si è implementata la serie di taylor specifica per l'approssimazione della funzione seno, anche in questo caso si tratta di una verifica delle sole capacità computazionali.

   Sono necessari due parametri di ingresso e un pulsante per azionare la computazione. Il primo di questi parametri indica il numero di iterazioni che si vogliono compiere per approssimare al meglio il valore, in particolare indica il numero di addendi utilizzati nella serie di taylor. Il secondo parametro invece indica l'argomento della funzione seno espresso in radianti. E' bene che il primo valore sia sufficientemente evitare risultati errati, in particolare rispetto al secondo valore, ad esempio se si lascia una sola iterazione il risultato sarà il secondo valore, per come è intesa la serie di taylor che qui di seguito mostriamo:

$$\sin x = \sum_{k=1}^{N} \frac{(-1)^k}{(2k+1)!} x^{2k+1}$$

3. Reattività dell'Interfaccia

   Questo è un test fatto appositamente per esaminare le capacità di Mobl di fornire l'astrazione dell'interfaccia reattiva, con riferimento all'apposita sezione del capitolo precedente, e riguarda sia le proprietà di calcolo che la reattività dell'interfaccia. Infatti, oltre ad essere rapidi nel portare a termine i calcoli, i due software dovranno mostrare

una particolare velocità anche nell'aggiornare la loro interfaccia con tali risultati.

Per la sua realizzazione è necessario un parametro numerico in ingresso che costituirà la base del processo vero e proprio, infatti, all'inserimento di questo, l'interfaccia si aggiornerà immediatamente fornendo vari risultati relativi a: moltiplicazioni, divisioni ed esponenziali tra il numero in input e i primi numeri interi. Quindi l'interfaccia verrà popolata di numeri che riguarderanno il risultato di ogni singola operazione. Vediamo di seguito le operazioni che verranno mostrate (si indica con 'x' il parametro in ingresso):

- Moltiplicazioni:

  $$x*2, x*3, x*4, x*5, x*6, x*7, x*8, x*9, x*10$$

- Divisioni:

  $$x/2, x/3, x/4, x/5, x/6, x/7, x/8, x/9, x/10$$

- Esponenziali:

  $\exp(x), \exp(x{+}1), \exp(x{+}2), \exp(x{+}3), \exp(x{+}4), \exp(x{+}5), \exp(x{+}6), \exp(x{+}7)$

4. Caricamento d'Immagini

   Contrariamente ai precedenti, questo benchmark è stato pensato per testare esclusivamente la capacità di entrambi i sistemi di caricare oggetti grafici.

   Si tratta di una semplice sequenza di immagini che prevede l'inserimento di un parametro numerico e la presenza di un pulsante per decretarne la partenza. Alla pressione del pulsante vengono mostrate, a distanza di un secondo l'una dall'altra, un numero di immagini pari al numero inserito in ingresso.

Come già introdotto ad inizio capitolo per ognuno di questi test, alla partenza, viene azionato un timer che si fermerà solamente alla fine del processo, in questo modo sarà possibile confrontare le tempistiche tra i due applicativi. Per quanto riguarda l'ultimo benchmark che prevede una pausa di un secondo tra un'immagine e la successiva, queste pause vengono sottratte al valore finale del timer, in questo modo anche per l'ultimo caso, si ottiene il valore effettivo del tempo impiegato per caricare le immagini.

I risultati ottenuti verranno mostrati nelle sezioni successive.

5.2 Confronto tra gli Applicativi

Ora che sono stati introdotti gli algoritmi utilizzati per esaminare i software, passiamo alla sezione più importante di questo capitolo che mette a confronto le prestazioni. Prima di inoltrarci nell'analisi vera e propria di ogni singolo test, introduciamo la schermata iniziale dei due programmi, in particolare si vuole mostrare come nella costruzione dei due si sia cercato di mantenere la stessa interfaccia e, dove possibile, le stesse metodologie di calcolo, per far si che le differenze riscontrate siano attribuibili pienamente alla piattaforma su cui gli applicativi vengono eseguiti e non alla loro costruzione.

La schermata principale di entrambi i software mostra immediatamente una lista dei test e la possibilità quindi di sceglierne uno di questi. Per accedere ad un particolare algoritmo basterà toccare la sezione relativa al medesimo e i programmi reagiranno mostrando il layout specifico. Vengono mostrati di seguito le due schermate iniziali.

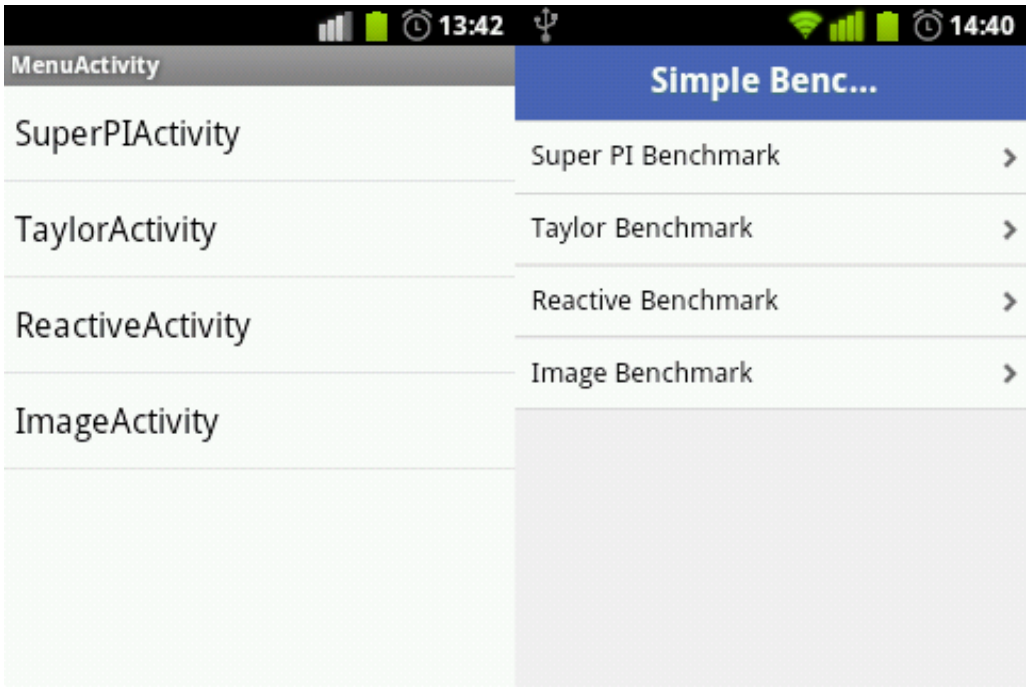


Figura 5.1: Schermata Iniziale dei Due Applicativi: Android e Mobl

5.2.1 Tempistiche d'Avvio

Purtroppo non si è in grado di definire in modo preciso la differenza di tempo nell'avvio delle due diverse applicazioni, ma si percepisce immediatamente che l'applicazione nativa Android risulta molto più veloce all'avvio rispetto a quella ideata tramite Mobl.

Un motivo per cui si ottiene questa differenza potrebbe essere dovuto al fatto che l'applicazione nativa si preoccupa di caricare solamente le risorse necessarie per visualizzare la schermata iniziale, mentre l'applicazione Mobl invece organizza tutte le risorse fin da subito. Ciò che ci spinge a fare questa considerazione è il comportamento che in seguito adotta il software nativo rispetto a quello Mobl, infatti quando si richiede l'ultimo test, questo impiega un tempo maggiore per caricare questo algoritmo rispetto alla controparte.

A parte queste due differenze a livello di tempistiche, l'accesso agli altri test risulta pressoché immediato all'utente per entrambi gli applicativi.

5.2.2 Calcolo delle Cifre Decimali del Pigreco

Prima di mostrare una tabella contenente i risultati messi a confronto tra i due programmi visualizziamo l'interfaccia utente di ognuno di essi.

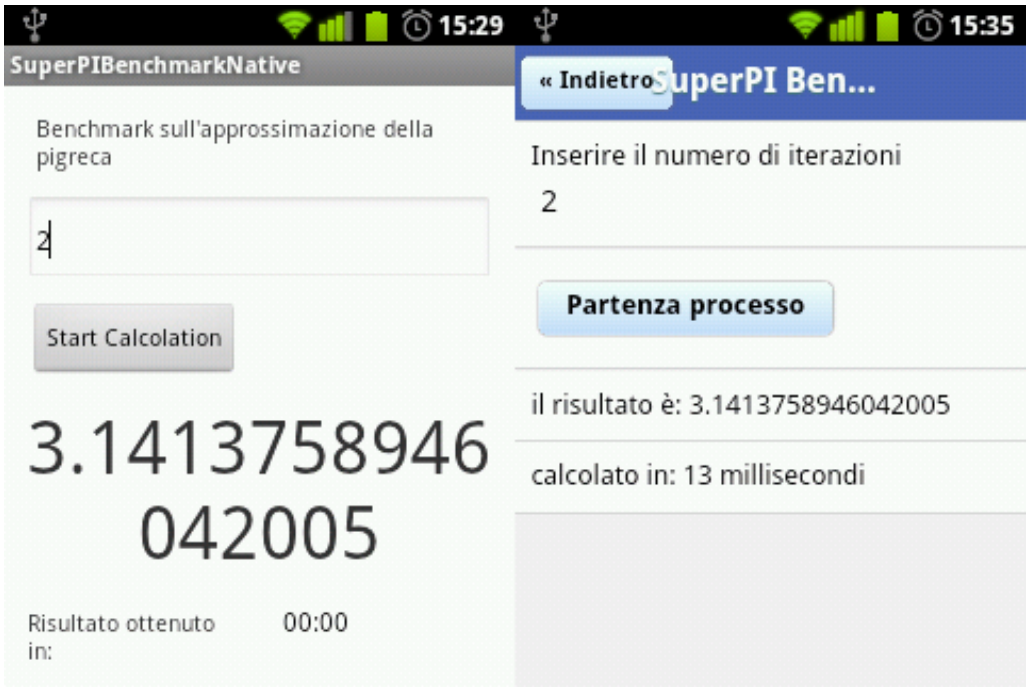


Figura 5.2: Approssimazione delle Cifre Decimali del Pigreco: Android e Mobl

Vediamo ora una semplice tabella dei risultati ottenuti espressi in millisecondi. Nonostante l'invariabilità del risultato ad ogni iterazione si riscontra un cambiamento anche nel tempo impiegato per l'elaborazione.

n. Iterazioni	Applicazione Nativa	Applicazione Mobl
15	3.4	10.4
62	9.5	34.1
125	18.2	67.3
250	25	110.1
500	36	213.4
1000	65.9	420

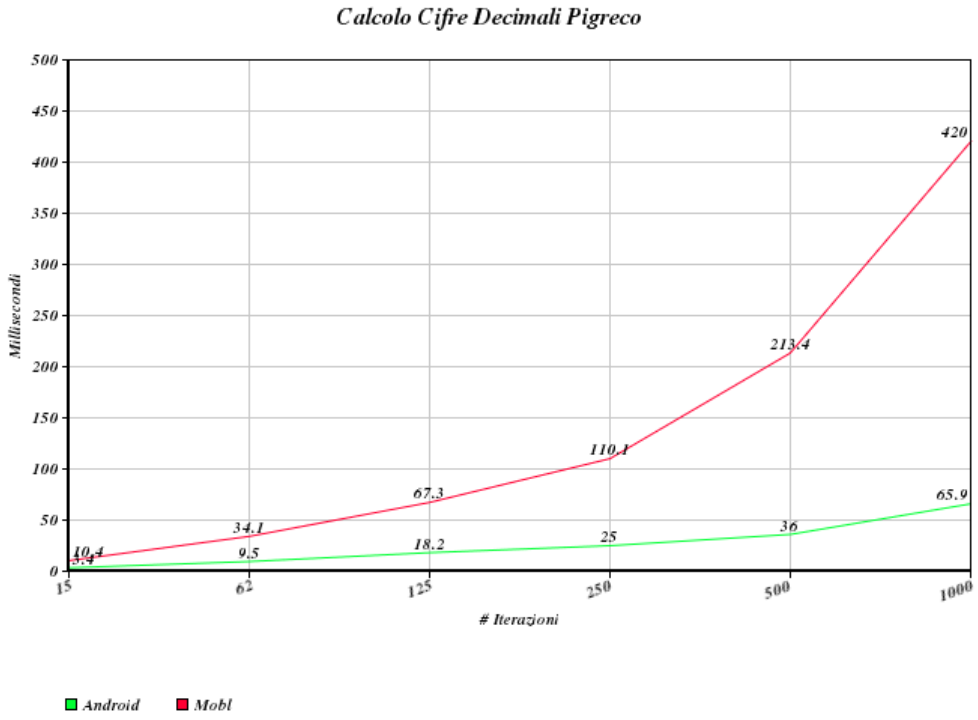


Figura 5.3: Grafico Riassuntivo dei Risultati: Test Pigreco

Come si può osservare, la differenza tra l'applicazione nativa e quella Mobl è notevole, in particolare la seconda tende sempre a raddoppiare il tempo di computazione mentre la prima riesce ad ottimizzare le tempistiche soprattutto ad alte computazioni. Di conseguenza i due applicativi risultano pressoché simili con algoritmi semplici piuttosto che in casi più complessi. I valori riportati nella tabella sopra sono delle semplici medie su dieci tentativi ad iterazione.

5.2.3 Approssimazione della Funzione Seno

Per questo tipo di Benchmark si è deciso di mantenere fissato il parametro del seno ad un valore pari a 5, che incide in maniera minore a livello computazionale, mentre si raddoppiano ogni volta il numero di iterazioni partendo da 10 fino a 160. In questo modo saremo in grado di comprendere meglio

le differenze tra i due programmi rispetto al precedente test che mostra un distacco temporale tra i due, apparentemente indipendente dal numero di iterazioni.

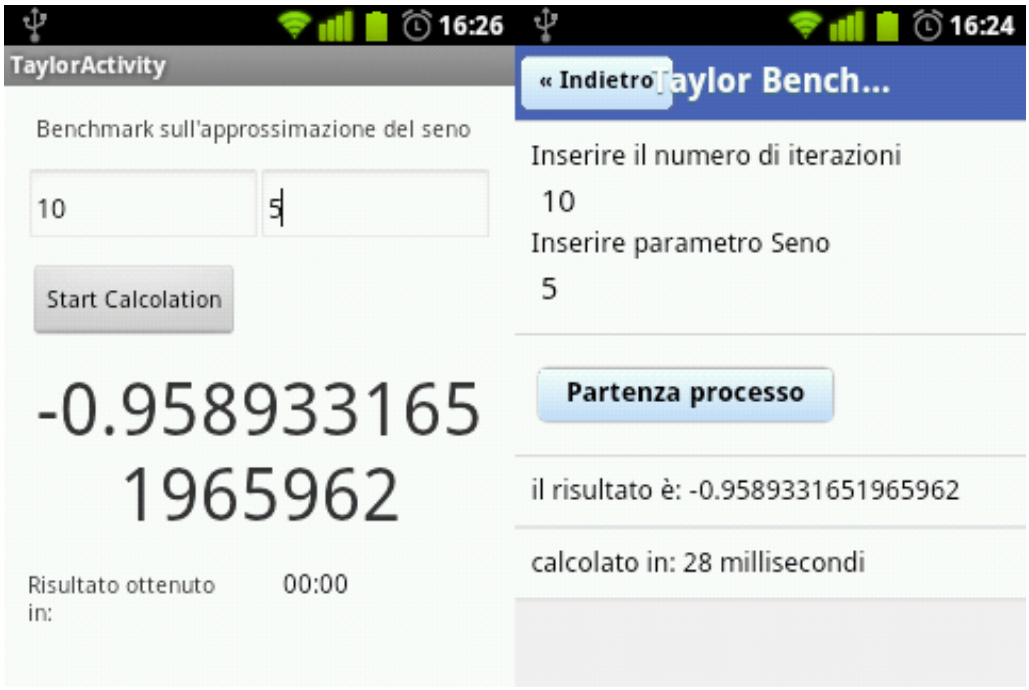


Figura 5.4: Approssimazione della Funzione Seno: Android e Mobl

n. Iterazioni	Applicazione Nativa	Applicazione Mobl
10	4.6	30.9
20	9.4	58.9
40	14.6	124.8
80	29.7	243.3
160	59.1	506.9

Con questo test si notano in maniera ancora più profonda le differenze di performance computazionali e come all'aumentare delle iterazioni le prestazioni delle applicazioni Mobl si degrada rispetto a quelle native che rimangono sempre al di sotto dei cento millisecondi.

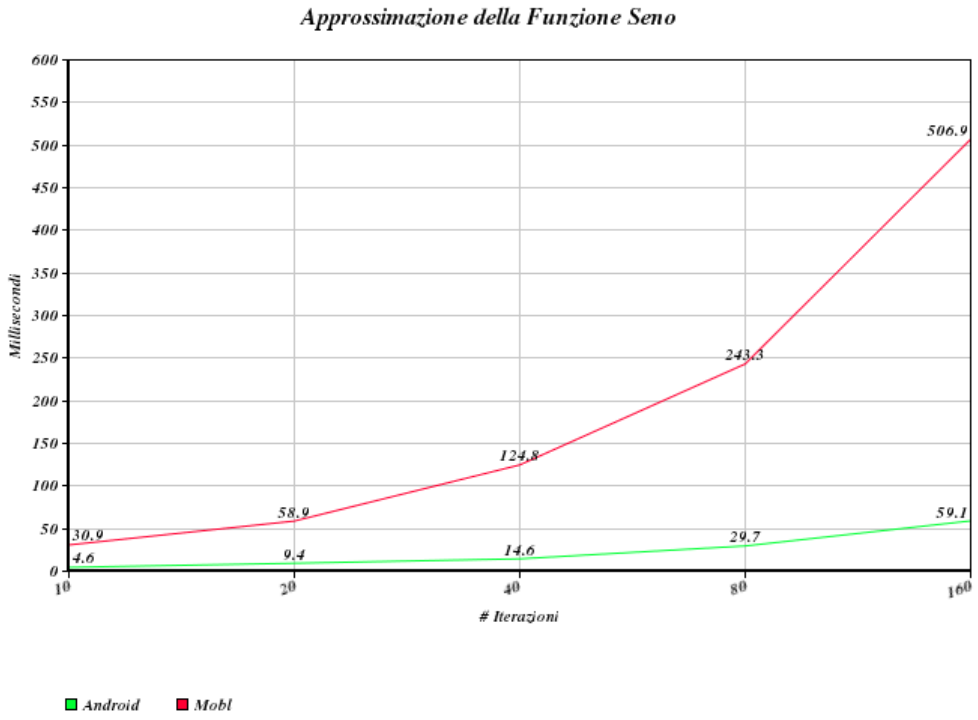


Figura 5.5: Grafico Riassuntivo dei Risultati: Test Seno

Oltre ad affermare quanto già visto, lo scopo di questo test è stato quello di verificare l'effettivo calcolo da parte dell'applicazione nativa. Infatti grazie alla possibilità di modificare il valore del parametro del seno, andando così a modificare di conseguenza anche il risultato, si è stati in grado di verificare che il software effettivamente calcolava il nuovo risultato invece di riproporre sempre il valore calcolato in precedenza anche all'aumentare delle iterazioni, sfatando così il dubbio poteva insorgere con il precedente test.

5.2.4 Reattività dell'Interfaccia

Con questo benchmark si vuole verificare le prestazioni che l'astrazione introdotta da mobl, *Reactive User Interface*, che consente di riflettere i cam-

biamenti di un dato direttamente sull'interfaccia grafica e sui valori a lui associati.

In questo caso il valore del parametro è meno rilevante, ma per effettuare un test completo si sceglieranno numeri casuali a diverse cifre significative tra loro, in modo del tutto simile ai precedenti.

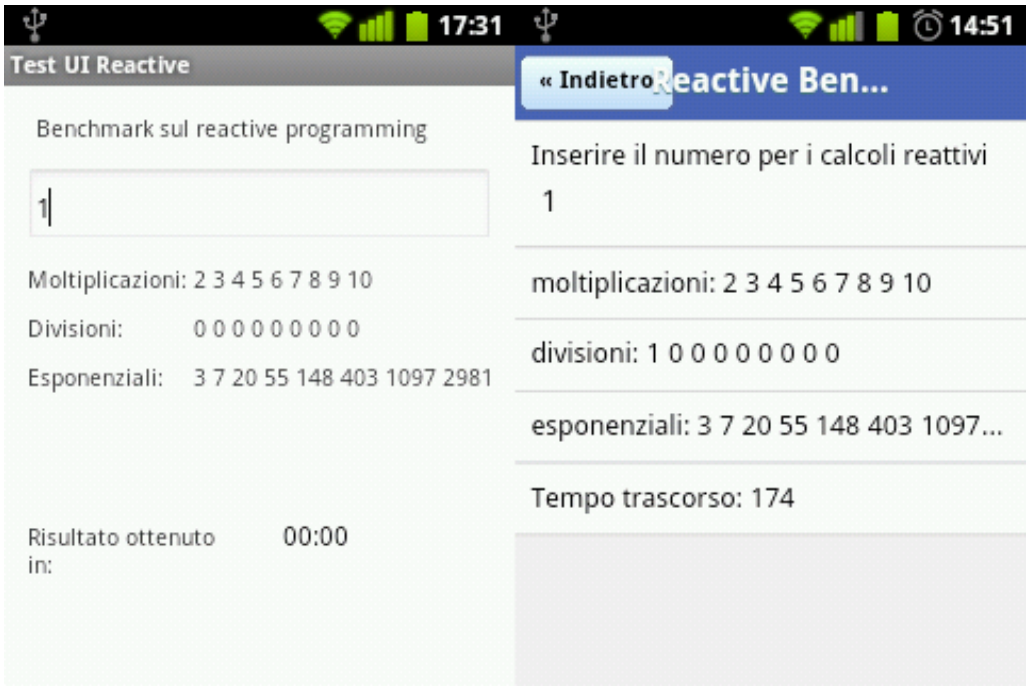


Figura 5.6: Interfaccia Reattiva: Android e Mobl

Valore Parametro	Applicazione Nativa	Applicazione Mobl
5	14.3	130
45	19	131.4
156	18	122
598	20.8	120.8
772	20.2	119.3

Si nota come i valori per entrambi i software siano stazionari tra i vari valori di ingresso, ma comunque si ha un lieve divario che vede sempre l'applicazione nativa godere di prestazioni più elevate.

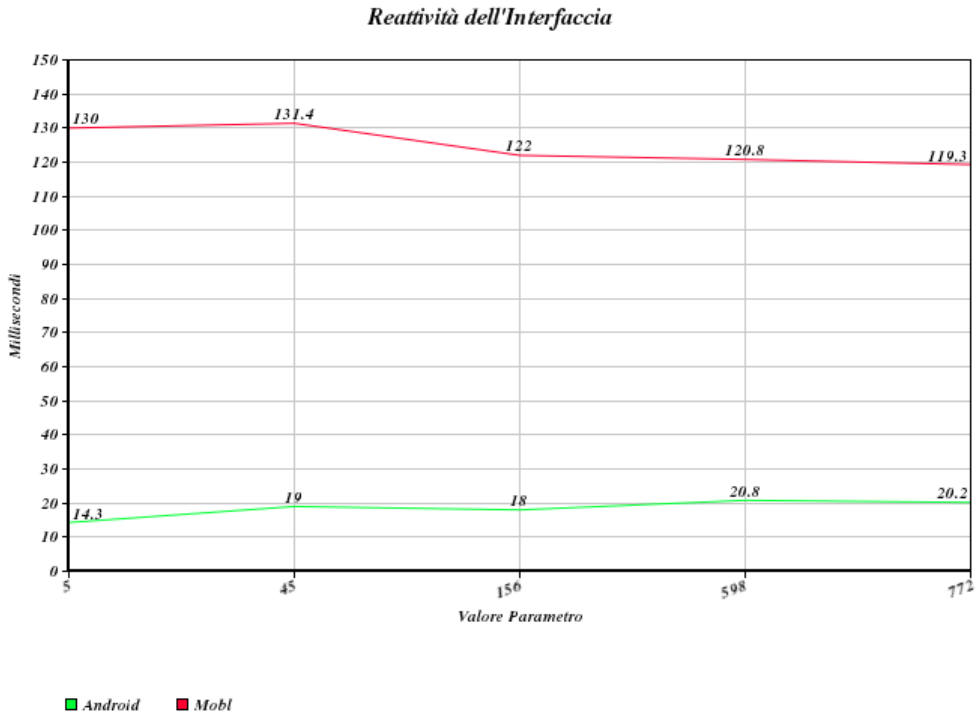


Figura 5.7: Grafico Riassuntivo dei Risultati: Test Interfaccia Reattiva

Si può notare una particolarità di questo test, che riguarda solamente l'esperienza utente: durante il raccoglimento dei dati l'utilizzo dell'applicazione Mobl è risultata particolarmente scomoda perchè per ottenere un aggiornamento dell'interfaccia è necessario dirigere il focus dell'applicazione al di fuori del controllo adibito all'input.

5.2.5 Caricamento d'Immagini

Questo benchmark è particolarmente importante perchè l'unico che riguarda solo ed esclusivamente l'interfaccia e la capacità dei due applicativi di aggiornarla e gestirla. Di seguito mostriamo come al solito le immagini dei software e la tabella riassuntiva dei risultati.

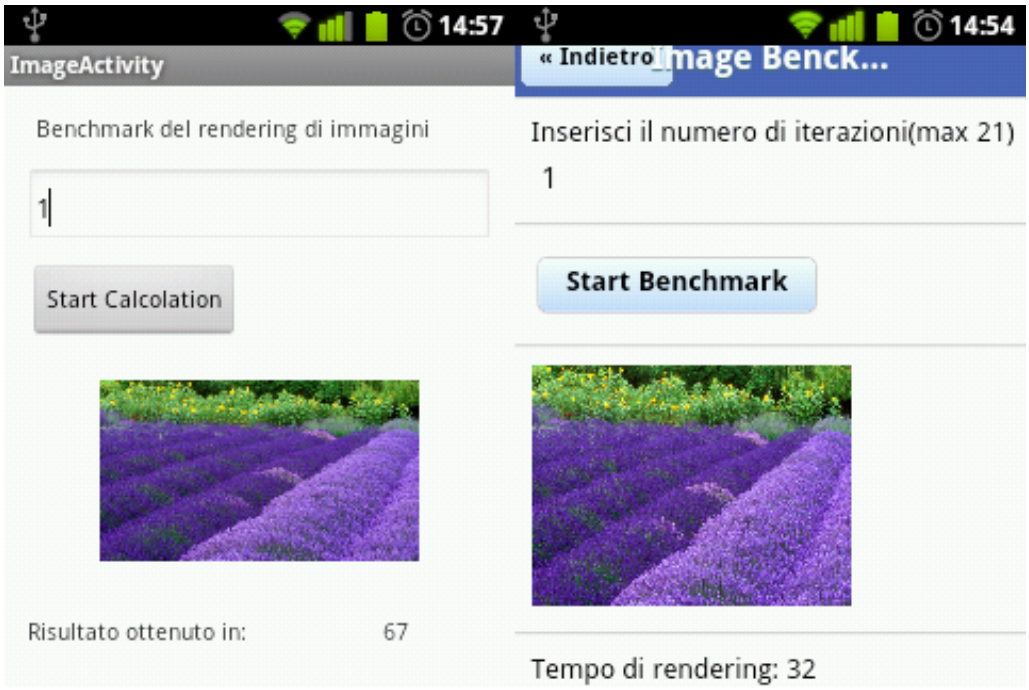


Figura 5.8: Caricamento d'Immagini: Android e Mobl

n. Immagini	Applicazione Nativa	Applicazione Mobl
2	64.8	36
6	74.4	96.4
12	87.6	177
19	101.5	291.8

Anche in questo caso si nota un particolare divario nel caricamento delle immagini, in particolare se si decide di mostrarne una quantità considerevole. Va comunque considerato che con un numero limitato di transizioni i due software si equivalgono e difficilmente si notano le differenze, tanto che per sole due immagini l'applicazione Mobl risulta a volte più veloce di quella nativa.

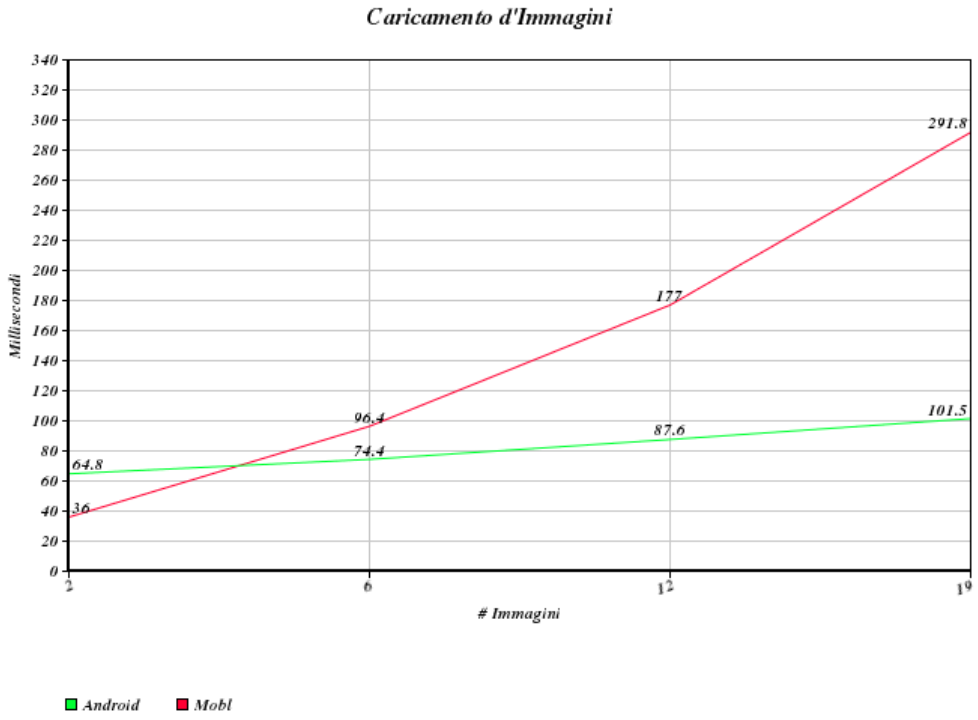


Figura 5.9: Grafico Riassuntivo dei Risultati: Test Caricamento Immagini

5.3 Conclusioni

Dai risultati di questo capitolo si può concludere che l'applicazione nativa supera quella costruita tramite Mobl, in maniera più netta se vengono richiesti compiti particolarmente complessi. Va considerato inoltre che i test visti in questo capitolo sono particolarmente semplici, quindi le prestazioni in situazioni più complesse potrebbero essere differenti.

I risultati ottenuti erano comunque alquanto prevedibili visto che le applicazioni Mobl, come quelle web, risiedono ad un livello maggiore rispetto a quelle native, infatti queste ultime risiedono direttamente sulla piattaforma mobile, mentre gli altri software necessitano dell'intervento del browser per poter accedere alle primitive di sistema. Questo introduce un passaggio aggiuntivo che causa il deterioramento delle prestazioni. Allo stato attua-

le, purtroppo, questo è il costo che bisogna considerare per poter ottenere l'indipendenza dalle piattaforme Mobile.

Capitolo 6

Conclusioni

In questo capitolo conclusivo verranno espresse delle considerazioni personali derivanti dall'analisi e dallo studio degli argomenti trattati precedentemente, confrontando le soluzioni platform-independent rispetto alle soluzioni native. In particolare si vuole capire se effettivamente le prime siano abbastanza mature per essere impiegate oppure non siano ancora in grado di offrire un supporto adeguato a progettisti e programmatori. Infine ci si concentrerà sulle novità che Mobl introduce rispetto agli altri frameworks e quindi gli aspetti positivi conseguenti, senza tralasciare però le mancanze che ancora questo sistema presenta.

6.1 Confronto tra Soluzioni Mobile

Abbiamo già visto all'interno di questo documento i punti di forza e le mancanze di una programmazione nativa rispetto all'utilizzo di una soluzione platform-independent.

Per quanto riguarda la programmazione nativa, il suo più grande difetto consiste appunto nella dipendenza stretta dalla piattaforma e dalle sue caratteristiche, mentre il suo punto di forza risiede: nel controllo totale sulle risorse del dispositivo(sensori, eventi, input, eccetera) e su tecnologie progettate per la costruzione di applicazioni mobile. I frameworks che abbiamo descritto invece sacrificano, totalmente o in parte, alcune caratteristiche della programmazione nativa per consentire la portabilità del proprio software.

A mio avviso, allo stato attuale, la scelta dell'opzione migliore sulla quale puntare, dipende strettamente dal tipo di applicativo che si vuole costruire. In particolare se si necessita di un utilizzo intensivo delle risorse del dispositivo e di una notevole capacità computazionale, allora è conveniente sposare una programmazione platform-specific, anche a fronte di spese aggiuntive. Se invece il proprio software non presenta grandi richieste prestazionali, ma invece si considera più importante renderlo disponibile al maggior numero di utenti possibile allora sicuramente converrà utilizzare una soluzione platform-independent. Quindi per ora l'utilizzo di applicazioni web-based su dispositivi mobile è confinata a software considerati agili e leggeri piuttosto che più complessi e articolati.

La conclusione appena formulata deriva, oltre che da un controllo delle risorse di un dispositivo, anche dalle prestazioni che un applicazione nativa può fornire rispetto ad una web-based. Infatti la prima viene eseguita direttamente sulla piattaforma mentre la seconda viene eseguita all'interno di un browser che ne aumenta la latenza. In particolare le prestazioni si degradano se il contenuto del software deve essere reperito in remoto piuttosto che conservato in locale; in questo caso entrano in gioco anche altri fattori come la disponibilità e la qualità della rete. Per questi motivi le applicazioni basate sulle tecnologie web risulteranno sempre meno performanti di quelle native, anche a fronte dell'evoluzione dei browser mobile. Fino a che questa grande differenza rimarrà ampia, difficilmente sarà possibile adottare questo tipo di applicazioni anche per casi complessi.

Per concludere si può sottolineare un ulteriore ostacolo della programmazione platform-independent: l'esperienza utente. Infatti risulta molto difficile, se non impossibile, riprodurre fedelmente l'interfaccia e il comportamento di un'applicazione nativa tramite le attuali tecnologie web. Questo difetto per l'utente medio può risultare fondamentale, tanto da spingerlo a preferire applicazioni native piuttosto che web-based, quindi anche questo aspetto deve essere considerato per i progettisti di applicazioni mobile. Per risolvere questa situazione alcuni framework consentono di implementare dei plugin per estendere le funzionalità del sistema.

6.2 Mobl

Anche Mobl, come gli altri frameworks, è basato su tecnologie web, infatti il prodotto finale di questo sistema è proprio un'applicazione web specifica per il mobile, di conseguenza anch'esso soffre delle problematiche sopra descritte. Mobl però si differenzia per vari aspetti principali, che sono stati anche il motivo per cui lo si è scelto come caso di studio e che abbiamo già visto nei precedenti capitoli.

I concetti di base di Mobl nascono da considerazioni, secondo me, giuste e corrette: infatti mi trovo personalmente in accordo nell'utilizzo di un nuovo linguaggio ad alto livello, che sia progettato con lo scopo di costruire applicazioni mobile, piuttosto che utilizzare le tecnologie web che sono ovviamente nate per scopi differenti, mantenendo come obbiettivo il platform-independent. Così facendo è stato possibile, come abbiamo visto, l'utilizzo di un'architettura più consona ad un software e molte altre caratteristiche: come il supporto di un IDE, il controllo degli errori staticamente e la possibilità di gestire tutti gli aspetti dell'applicativo, dall'interfaccia alla logica e alla gestione dei database, tramite un unico linguaggio.

Su queste basi, il progetto di Mobl, presenta grandi potenzialità, ma allo stato attuale si hanno anche molte problematiche come ad esempio: una documentazione ancora incompleta, pochi esempi ed ancora poche librerie per l'utilizzo di molte risorse dei vari dispositivi.

Mi auguro che questo progetto possa progredire e colmare queste lacune in modo da divenire una valida alternativa ai più famosi framework.