POLITECNICO DI MILANO

Polo Regionale di Como

Corso di Laurea Magistrale in Ingegneria Informatica

Sviluppo di applicazioni mobile cross-platform con Apache Cordova: Il caso di Urbanopoly

Tesi di laurea magistrale

Attilio Fiumanò matr. 770633

Relatore: **Prof. Emanuele Della Valle**

Correlatore: **Ing. Irene Celino**

Anno Accademico 2011/2012

Indice

Indice delle figure

1 - Introduzione

Al giorno d'oggi, nell'epoca in cui smartphone e tablet sono prepotentemente entrati nell'uso comune di una porzione sempre crescente di utilizzatori finali, diventa fondamentale dotare tali periferiche di software che ne permettano il pieno godimento e la piena funzionalità. A questo proposito gli sviluppatori di tali applicativi sono costantemente al lavoro per realizzare applicativi che crescano di pari passo alla crescita dei componenti hardware soggetti a continue evoluzioni e mutamenti.

A complicare il lavoro dello sviluppatore si aggiunge la vastissima eterogeneità di device disponibili sul mercato che si traduce nello sviluppo moltiplicato per ogni tipo di standard di uno stesso applicativo. E' facile comprendere quanto possa risultare lungo, e quindi costoso, lo sviluppo di un'applicazione che abbia lo scopo di essere eseguito sul dispositivo di ogni tipo di utente: ci può essere il caso di un dispositivo con 3 o 4 anni di vita con determinate caratteristiche, un dispositivo dell'anno precedente con altre caratteristiche e diverso sistema operativo o un dispositivo di ultimissima generazione con caratteristiche software e hardware non paragonabili ai casi descritti precedentemente. E' facile rendersi conto che considerare tutte le possibili casistiche (peraltro in continua evoluzione) è praticamente impossibile.

Per questo motivo usualmente gli sviluppatori decidono di dedicarsi ad uno specifico target di sviluppo (es. iOS quindi iPhone,iPod, iPad) dividendosi il compito con altri che parallelamente si dedicano ad un'altra fascia di device (es. Android ,Blackberry o altro). Appare evidente che questo approccio non sia altro che un ripiego vista l'apparente impossibilità di poter trovare una soluzione migliore che possa soddisfare le esigenze di tutti o dei più.

1.1 I due mondi

Attualmente il mercato di smartphone e tablet sembra sostanzialmente dividersi in due grandi mondi a se stanti senza (quasi) nessun punto d'incontro o possibilità di avvicinamento. Uno è il mondo Apple quindi dei dispositivi con sistema operativo iOS, l'altro è quello di Google col rispettivo sistema operativo Android. L'approccio, nei due casi, è completamente opposto.

Uno, il primo, impone un sistema molto rigido e soggetto a una politica di chiusura, un approccio conservativo che si oppone alla direzione verso la quale il mondo è proiettato, dove la conoscenza è sempre più condivisa. Non a caso lo sviluppo di applicazioni iOS deve nativamente passare dal linguaggio Objective C, assolutamente in disuso prima del boom di Apple, possono avvenire solo tramite l'uso di X-Code (quindi tramite computer targati Apple vedi MacBook Pro, MacBook Air, iMac,..) a fronte dell'acquisizione di una licenza di sviluppatore Apple fruibile al seguito del pagamento di una quota annuale.

Di tutt'altra concezione è l'approccio di Google: lo sviluppo di un'applicazione Android si fonda sul linguaggio Java, linguaggio di grande diffusione in ogni ambito, di conseguenza gestibile tramite Eclipse (disponibile per tutti i sistemi operativi, inclusi quelli Apple) con un apposita SDK e permette il debugging sulla periferica e la pubblicazione sull'Android Market in maniera completamente gratuita.

Di facile intuizione è quindi la difficoltà in cui si possa trovare uno sviluppatore nel momento in cui ci sia la necessità di dover uniformare un'applicazione dovendo garantire una compatibilità totale: si pensi al tempo che deve quindi dedicare all'apprendimento di un nuovo linguaggio di programmazione, come l'Objective C, completamente diverso, dal punto di vista dell'approccio concettuale e della sintassi, dai tradizionali linguaggi di programmazione object-oriented di moderna concezione.

Il risultato è per l'appunto lo sviluppo di due (o più) applicazioni distinte col conseguente raddoppio (o più) dei costi di sviluppo.

1.2 Sviluppo cross-platform

Il problema sembra essere ben noto e compreso da chi si occupa di dotare gli sviluppatori di strumenti informatici in modo da poter ottimizzare il loro lavoro, evitare ridondanze o repliche di stesse parti, essenziali e non, del codice. D'altronde il tema della ridondanza di codice e la conseguente moltiplicazione delle risorse necessarie ad accontentare gli utilizzatori di diverse piattaforme ha origini ben più remote rispetto all'attuale stato della tecnologia.

Nasce così l'esigenza di mettere in condizione lo sviluppatore di creare un codice che qualsiasi device sia in grado di comprendere e di utilizzare, indipendentemente dalla piattaforma, dal sistema operativo e dall'hardware di cui è dotato.

Il principio è quello del **riutilizzo del codice** secondo cui un algoritmo, una parte di programma, una soluzione informatica debba essere scritta una sola volta e poi richiamata più volte all'occorrenza. I vantaggi sono molteplici e si concentrano in un unico concetto che è facile intuire: un codice unico da la possibilità di "colpire" tutti i device quindi di minimizzare i tempi di sviluppo di un applicativo, di dare la possibilità di aggiornare, modificare, ottimizzare un applicazione (estensione del codice) in maniera **uniforme** e **univoca**, e garantire la **manutenibilità del codice** che essendo unico rimane sotto il pieno controllo dello sviluppatore. Viene così abbandonato il problema di repliche di codice, di struttura o componenti, uno dei problemi informatici di più difficile gestione.

L'esigenza diventa quindi quella di avere uno strato intermedio, un **framework**, che si occupi di offrire allo sviluppatore la possibilità di sviluppare in un linguaggio di programmazione **universale** e la possibilità di accedere alle risorse hardware dei vari dispositivi utilizzando un sistema univoco. Il compito di questo layer intermedio è dunque quello di "tradurre" le chiamate provenienti da un livello più elevato (quindi generale) dell'applicazione nei corrispettivi linguaggi nativi (quindi di livello più basso) e di conseguenza restituire i relativi risultati al chiamante garantendo allo sviluppatore uniformità di fruizione di tali servizi per tutti i casi possibili di device, risorse hardware e sistemi operativi.

Ovviamente si tratta di un obiettivo ambizioso che si pone come punto d'incontro di due (e più) mondi completamente diversi ognuno con caratteristiche, funzionalità, concetti e principi propri e molto lontani tra loro. Sarebbe però un errore, data appunto questa grande eterogeneità, pensare che il framework possa pensare a tutto: non tutte le problematiche e i possibili casi

possono essere gestiti dal framework, è quindi necessario, in certi casi, che sia lo sviluppatore a dover prendere delle scelte e gestire in modi differenti specifici problemi che non possono avere una soluzione universale (per esempio a causa di funzionalità hardware diverse o che possono essere presenti o non presenti).

1.3 Stato dell'arte

Data l'importanza dell'obiettivo ricercato, non pochi team e organizzazioni si sono avviati in direzione di soddisfare le esigenze degli sviluppatori tramite la creazione di framework e tool dedicati. Di questi però sono due i grandi progetti che, ad oggi, sono da ritenersi sufficientemente pronti e adeguati per l'utilizzo in contesti reali a livello professionale: Apache Cordova (inizialmente denominato PhoneGap) e Appcelerator Titanium.

Si tratta di due grandi progetti che perseguono lo stesso obiettivo utilizzando strade diverse.

1.3.1 Apache Cordova

Apache Cordova permette di utilizzare gli standard HTML 5 e CSS 3 per scrivere e disegnare la propria applicazione mobile fornendo anche la completa SDK Javascript per accedere alle risorse native del dispositivo. Si consiglia nei casi dove l'obiettivo è la fruizione sulla maggior parte di tablet e dispositivi mobile in circolazione (iOS, Android, Blackberry, Windows Phone, Palm WebOS, Bada e Symbian). Da non sottovalutare il fatto di poter utilizzare tutte le librerie Javascript esterne e di grande diffusione e utilità come jQuery, jQuery Mobile, ecc. E' sicuramente la migliore alternativa per il **riutilizzo del codice** di contro può causare problemi o difetti in termini di performance e fluidità.

1.3.2 Appcelerator Titanium

Pur basandosi su codice Javascript, Appcelerator Titanium, risulta molto diverso dal precedente framework in quanto l'applicazione viene effettivamente compilata a partire dal codice Javascript che viene convertito in codice nativo. Questa scelta pur garantendo migliore performance può implicare il dover riscrivere parte dell'applicazione per target specifici. Tra i contro c'è da sottolineare che solo le API Javascript base sono supportate senza possibilità di godere dell'utilizzo di estensioni come le già citate jQuery, jQuery Mobile e molte altre. Infine Titanium ha come target esclusivamente iOS e Android.

Oggetto di questa ricerca è l'approfondimento e la valutazione delle potenzialità di Apache Cordova.

2 - Apache Cordova

Apache Cordova (altre volte chiamato col nome del suo predecessore “PhoneGap”), come già introdotto, rappresenta un set di API che permette a uno sviluppatore di applicazioni mobile di accedere alle funzione native del device stesso quali la fotocamera, l’accelerometro, le coordinate GPS, il File System e altri, tramite l’uso di chiamate JavaScript. L’uso combinato con framework UI come jQuery Mobile fa si che l’applicazione venga generata semplicemente con l’uso HTML, CSS e Javascript seguendo il tradizionale approccio delle applicazioni web. Cordova si occuperà di tramutare la web application in una mobile application [1].

Utilizzando le API di Cordova l’applicazione potrà essere realizzata senza la necessità di utilizzare il codice native (che sia Java, Objective-C, o altri). Cordova si basa invece sull’utilizzo di tecnologie web, solitamente situate sul device stesso e non su un server remoto. E’ proprio lo standard web di HTML CSS e JavaScript che rappresentano la garanzia che l’applicativo sia a tutti gli effetti multipiattaforma e che il porting su dispositivi diversi non comporti, se non in minima parte, la necessità di modifiche al codice. Le applicazioni generate con l’uso di Apache Cordova saranno, come le applicazioni native, compilate, usando le relative SDK (Android SDK, X-Code, o altri), in modo che siano installabili dai tradizionali App Store proprietari di ogni piattaforma.

Cordova, come anticipato, fornisce quindi un set di librerie Javascript che possono essere invocate. Ogni chiamata verrà poi tramutata dal framework nella relativa chiamata in codice nativo in base al device di utilizzo. Il risultato di questa operazione è un informazione che risulterà essere uniforme indipendentemente dalla piattaforma di utilizzo, e che metterà lo sviluppatore in condizione di disinteressarsi di quale sia l’effettivo dispositivo di utilizzo finale.

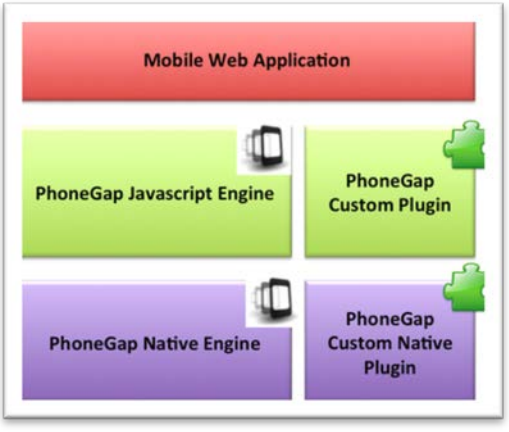

Figura 1 - *Architettura logica di applicazioni mobile basate su Apache Cordova.[2]*

Cordova è disponibile per le seguenti piattaforme: iOS, Android, Blackberry, Windows Phone, Palm WebOS, Bada, e Symbian.

Apache Cordova si è classificato nel mese di Ottobre 2012 come progetto “top level” nella Apache Software Foundation (ASF). Il progetto è rimarrà gratuto e open source sotto licenza Apache v.2.0.

Nonostante il framework si fondi sul principio di unicità del codice è comunque inevitabile che la fase ultima di deploy dell’applicazione debba essere effettuata su una piattaforma di sviluppo nativa: una volta completato lo sviluppo basato su HTML, JS e CSS la fase finale prevede la creazione, su X-Code per iOS, su Eclipse per Android , di un progetto che, sfruttando le librerie native di Cordova (una per ogni tipo di piattaforma) permetterà la fruizione e l’interoperabilità con l’applicativo scritto in linguaggio universale.

Per la creazione di tali progetti necessari al porting dell’applicazione non è richiesta alcuna conoscenza di linguaggio nativo in quanto i passaggi sono illustrati nel dettaglio sul sito di Apache Cordova.

2.1 Funzionalità offerte

Le funzionalità di Cordova prevedono praticamente tutte le interazioni che sono possibili con tutti i device di ultima generazione e non.

Ecco una breve rassegna di tutte le classi di funzionalità che è possibile richiamare:

- **Accelerometer:** Cattura il movimento del dispositivo misurando la sua accelerazione lungo le direzioni x, y e z.

- **Camera:** Permette l'accesso alla fotocamera, da la possibilità di salvare l'immagine come file oppure di ottenerla con codifica base64 (codifica utilizzata per la trasmissione tramite chiamate http). Dà, per la maggior parte delle piattaforme, anche la possibilità di accedere alle immagini già presenti nella libreria fotografica del dispositivo.

- **Capture:** Più generico del precedente oggetto Camera, permette l'acquisizione di file multimediali di tipo audio, immagine o video. E' possibile specificare diversi formati di acquisizione.

- **Connection:** Fornisce informazioni sulla connettività attuale: se non è presente, se è di tipo ETHERNET, WIFI, 2G, 3G o 4G.

- **Contacts:** Consente di accedere alle informazioni relative ai contatti, di crearne nuovi o modificarli.

- **Device:** Informazioni hardware e software del device in uso.

- **Events:** Permette di gestire diversi tipi di eventi: device pronto (framework cordova completamente caricato), pausa (quando l'applicazione va in background), online (rilevata la connettività), offline (rilevata mancanza di connettività), pressione tasto back (se presente), rilevato livello critico di batteria, basso livello batteria, rilevata pressione di bottoni hardware vari.

- **File:** Permette l'accesso al file system, è possibile leggere e scrivere dai file e cartelle, eseguirne l'upload online e ottenere i metadata.

- **Geolocation:** Permette l'accesso al sensore GPS quindi di rilevare la posizione corrente del device che corrisponderà alle coordinate fornite dal sensore (se presente) o calcolate in base alle specifiche W3C delle API di geolocalizzazione nel caso non fosse presente il sensore. Le coordinate vengono restituite sotto forma di latitudine e longitudine.

- **Globalization:** Permette di avere informazioni circa la lingue e le impostazioni internazionali correnti (lingua attuale, separatore migliaia, decimali, formato per la data ecc).

- **Notification:** Permette di eseguire azioni di notifica come Visualizzazione di messaggi (alert), vibrazione, suoni, utilizzando audio e grafica native del device utilizzato.

- **Storage:** Permette l'utilizzo dello storage del dispositivo basata sulla specifica W3C dei Database SQL Web.

2.2 Tanti pregi, pochi difetti

A dispetto di tutti i vantaggi apportati dal framework, derivanti dalla possibilità di fruire di un linguaggio di programmazione universale basato sugli standard Web, bisogna però evidenziarne anche i (pochi) difetti.

Essenzialmente i difetti, peraltro inevitabili, sono di fatto due:

- L'impossibilità di utilizzare un debugger (degno di questo nome)
- Vincoli legati alle risorse di sistema utilizzabili (processore, ram)

Per quanto concerne il primo punto, si può dedurre che a livello concettuale sia impensabile poter utilizzare le funzioni di debugging native dell'ambiente di sviluppo (che si tratti di X-Code o di Eclipse): l'applicativo agli occhi del compilatore non è altro che un browser, seppur integrato all'utilizzo delle risorse hardware del dispositivo, che non può riconoscere come linguaggio proprio il codice Javascript integrato nell'applicazione, esso infatti viene visto solo come testo puro (come anche le altre porzioni di codice HTML, CSS e via dicendo). Nonostante questo gli sviluppatori di Cordova hanno cercato, tramite la piattaforma Weinre [3], di creare un debugger fruibile online. L'idea è quella di includere nel proprio codice html un riferimento a un codice javascript situato online e che abbia un riferimento del progetto stesso. Il codice html da incorporare è il seguente:

```
<script src="http://debug.phonegap.com/target/target-script-min.js#nome_progetto"></script>
```

Successivamente visitando la pagina http://debug.phonegap.com/client/#nome_progetto, e contemporaneamente eseguendo l'applicazione, sarà possibile in tempo reale vedere via browser una schermata di debugger simile agli "Strumenti per sviluppatori" di Google Chrome. Sebbene l'idea sia ottima questo tipo di debugger prevede solo la parte relativa alla visione del codice html in tempo (quasi) reale e del codice CSS applicato a ciascun elemento. Rimane quindi irrisolta la problematica relativa al debug del codice Javascript e la gestione delle chiamate http a server remoto.

Il secondo problema segnalato riguarda invece i limiti inerenti alle risorse hardware utilizzabili. Trattandosi di un'applicazione sviluppata all'interno di un'applicazione vera e propria (ovvero quella simil browser) non è possibile usufruire appieno delle capacità hardware del device

utilizzato. Tale limitazione sovviene soprattutto in fase di allocazione della memoria RAM che, in molti casi, avviserà tramite dei warning (visibili in fase di test su X-Code o Eclipse) in caso di superamento della soglia di memoria allocabile.

Anche dal punto di vista del comparto grafico non si può certo pensare di ottenere gli stessi risultati paragonando un applicazione nativa all'applicazione web che Cordova ci permette di creare. Sempre a causa degli stessi limiti si potrà notare come alcune operazioni come lo scroll, o altre azioni tipiche di un dispositive mobile, potranno risultare meno fluide rispetto alle corrispettive versioni native del prodotto.

2.3 Deploy

2.3.1 Deploy su iOS

Il deploy su iOS prevede qualche step necessario alla compilazione dell'applicazione. Per questo step è comunque necessario essere dotati di un dispositivo Apple (quale MacBook o simili),del software XCode, e di una licenza sviluppatore per poter avere la possibilità di poter eseguire il deploy sul proprio dispositivo.

Volendo riassumere gli step (ampiamente spiegati sul sito del produttore) bisogna procedere come segue [4]:

1. Installare la versione più aggiornata di X-Code dal Mac App Store.
2. Scaricare la versione più recente di Apache Cordova contenente, tra le tante, le libreria dedicata a iOS.
3. Estrarre, dal file scaricato, il progetto Cordova proposto di default in una cartella a piacere.
4. Creare un progetto specificando il nome dello stesso, il nome del package (es. it.polimi.Urbanopoly) e la directory appena creata.
5. Creare la propria applicazione in HTML, CSS e Javascript (il passo precedente crea all'interno del progetto delle directory dove c'è un applicativo basico che basterà quindi sostituire con il proprio).
6. Compilare e eseguire l'applicazione sulla periferica o sull'emulatore.

La documentazione del framework spiega, inoltre, i passi da seguire per dare i permessi all'applicativo di poter accedere alle risorse hardware (es. GPS, Fotocamera, o altri).

2.3.2 Deploy su Android

Parallelamente al deploy su iOS anche per quanto riguarda la parte Android c'è la necessità di creare un progetto in ambiente nativo. Il principio di creazione del progetto è lo stesso e si può riassumere nei seguenti passi [5]:

1. Download e installazione di Eclipse, Java SDK, Android SDK.
2. Scaricare la versione più recente di Apache Cordova contenente, tra le tante, le libreria dedicata a Android.
3. Estrarre, dal file scaricato, il progetto Cordova proposto di default in una cartella a piacere.

4. Creare un progetto tramite il wizard proposto da Eclipse selezionando la modalità Android Project From Existing Code selezionando la directory del progetto esistente.
5. Creare la propria applicazione nella cartella dove è presente l'applicativo di test.
6. Eseguire il deploy sul device o sull'emulatore

Anche in questo caso la documentazione fornisce spiegazione circa i vari step specifici per ogni risorsa hardware di cui si ha necessità.

2.4 Esempi pratici [6]

2.4.1 Geolocalizzazione – ottenere le coordinate correnti

Attraverso l'oggetto *navigator.geolocation* si ha accesso a tutti i metodi e informazioni sulla geolocalizzazione del device. Per ottenere le coordinate è sufficiente richiamare il suo metodo getCurrentPosition con i parametri relativi alle funzioni di callback in caso di esito positivo dell'operazione o esito negativo. Queste funzioni verranno quindi richiamate al termine dell'acquisizioni delle informazioni geografiche e, in caso di esito positivo, verrà trasmesso un oggetto position con tutte le informazioni ricavate (vedi Fig.2).

```
// Wait for Cordova to load
//
document.addEventListener("deviceready", onDeviceReady, false);

// Cordova is ready
//
function onDeviceReady() {
    navigator.geolocation.getCurrentPosition(onSuccess, onError);
}
// onSuccess Geolocation
//
function onSuccess(position) {
  var element = document.getElementById('geolocation');
  element.innerHTML = 'Latitude: '          + position.coords.latitude              + '<br />' +
                      'Longitude: '         + position.coords.longitude             + '<br />' +
                      'Altitude: '          + position.coords.altitude              + '<br />' +
                      'Accuracy: '          + position.coords.accuracy              + '<br />' +
                      'Altitude Accuracy: ' + position.coords.altitudeAccuracy      + '<br />' +
                      'Heading: '           + position.coords.heading               + '<br />' +
                      'Speed: '             + position.coords.speed                 + '<br />' +
                      'Timestamp: '         + position.timestamp          + '<br />';
}
// onError Callback receives a PositionError object
//
function onError(error) {
    alert('code: '    + error.code    + '\n' +
          'message: ' + error.message + '\n');
}
</script>
</head>
```

Figura 2 - *Esempio pratico: ottenere informazioni di geolocalizzazione tramite Javascript.*

2.4.2 Camera – scattare una foto

L'oggetto *navigator.camera* dà pieno accesso all'integrazione con la fotocamera o in alternativa con la galleria di immagini presenti nel dispositivo. Nell'esempio (fig. 3) all'interno della funzione getPhoto viene richiamata la funzione getPicture. A questa funzione vengono passate le funzioni di callback in caso di successo o di errore e un oggetto contenente informazioni circa la qualità della foto desiderata, il formato dell'ouput (si può scegliere tra ottenere la stringa del percorso del file salvato o l'intera immagine sotto forma di stringa con codifica base64) e informazioni sulla sorgente (la fotocamera stessa o la galleria di immagini). L'esempio mostra come al termine dell'acquisizione, in caso di successo, l'immagine venga mostrata all'interno di un elemento html ("largeImage").

```
//
document.addEventListener("deviceready",onDeviceReady,false);
function onDeviceReady() {
    pictureSource=navigator.camera.PictureSourceType;
    destinationType=navigator.camera.DestinationType;
}
// Called when a photo is successfully retrieved
//
  // Called when a photo is successfully retrieved
//
function onPhotoURISuccess(imageURI) {
  // Uncomment to view the image file URI
  // console.log(imageURI);

  // Get image handle
  //
  var largeImage = document.getElementById('largeImage');

  // Unhide image elements
  //
  largeImage.style.display = 'block';

  // Show the captured photo
  // The inline CSS rules are used to resize the image
  //
  largeImage.src = imageURI;
}

// A button will call this function
//
function getPhoto(source) {
  // Retrieve image file location from specified source
  navigator.camera.getPicture(onPhotoURISuccess, onFail, { quality: 50,
    destinationType: destinationType.FILE_URI,
    sourceType: source });
}
```

Figura 3 - *Esempio pratico: scattare una foto e ottenere il suo percorso tramite Javascript.*

2.4.3 Globalization - Ottenere le impostazioni di lingua correnti

L'oggetto *navigator.globalization* viene usato per accedere alle impostazioni internazionali del dispositivo. Si possono ottenere informazioni circa la lingua corrente, il modello di conversione di data in stringa predefinita, il primo giorno della settimana secondo il sistema adottato, il carattere utilizzato per separare le migliaia e i decimali e altre informazioni di questo tipo. Nell'esempio seguente (fig. 4) richiamando il metodo getLocaleName si ottiene informazioni sulla lingua corrente, anche qui i parametri sono le due funzioni di callback richiamate in caso di successo o in caso di errore. A differenza dei casi precedenti le funzioni vengono implementate *inline*.

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Cordova</title>
    <script type="text/javascript" charset="utf-8" src="cordova-2.2.0.js"></script>
    <script type="text/javascript" charset="utf-8">

    function checkLocale() {
      navigator.globalization.getLocaleName(
        function (locale) {alert('locale: ' + locale.value + '\n');},
        function () {alert('Error getting locale\n');}
      );
    }
    </script>
  </head>
  <body>
    <button onclick="checkLocale()">Click for locale</button>
  </body>
</html>
```

Figura 4 - *Esempio pratico: ottenere impostazioni internazionali del device tramite Javascript.*

2.4.4 File – Upload di una foto presa dalla libreria su un server

Apache Cordova da la possibilità di caricare un file su un server web opportunamente configurato per riceverlo. Questo avviene attraverso l'oggetto FileTransfer. Nell'esempio seguente (fig. 5) viene selezionata un'immagine dalla galleria (metodo navigator.camera.getPicture) e successivamente essa viene inviata a un server web all'interno della funzione uploadPhoto. Si noti che tale funzione sia appunto indicata all'interno del metodo getPicture per essere richiamata in caso di successo dell'acquisizione dell'immagine. Il metodo ft.upload permette quindi l'invio dell'immagine con la possibilità di unire parametri arbitrari al trasferimento del file.

```
function onDeviceReady() {
    // Retrieve image file location from specified source
    navigator.camera.getPicture(uploadPhoto,
                                function(message) { alert('get picture failed'); },
                                { quality: 50,
                                destinationType: navigator.camera.DestinationType.FILE_URI,
                                sourceType: navigator.camera.PictureSourceType.PHOTOLIBRARY
                                );

}
function uploadPhoto(imageURI) {
    var options = new FileUploadOptions();
    options.fileKey="file";
    options.fileName=imageURI.substr(imageURI.lastIndexOf('/')+1);
    options.mimeType="image/jpeg";

    var params = {};
    params.value1 = "test";
    params.value2 = "param";

    options.params = params;

    var ft = new FileTransfer();
    ft.upload(imageURI, encodeURI("http://some.server.com/upload.php"), win, fail, options);
}

function win(r) {
    console.log("Code = " + r.responseCode);
    console.log("Response = " + r.response);
    console.log("Sent = " + r.bytesSent);
}

function fail(error) {
    alert("An error has occurred: Code = " + error.code);
    console.log("upload error source " + error.source);
    console.log("upload error target " + error.target);
}
```

Figura 5 - *Esempio pratico: eseguire l'upload di un file su un server web.*

3 - Tool e tecnologie utilizzate

3.1 HTML e CSS

L'HTML [7] è un linguaggio di pubblico dominio la cui sintassi è stabilita dal World Wide Web Consortium (W3C), e che è basato su un altro linguaggio avente scopi più generici, l'SGML.

È stato sviluppato verso la fine degli anni ottanta da Tim Berners-Lee al CERN di Ginevra assieme al noto protocollo HTTP che supporta invece il trasferimento di documenti in tale formato. Verso il 1994 ha avuto una forte diffusione in seguito ai primi utilizzi commerciali del web.

Nel corso degli anni, seguendo lo sviluppo di Internet, l'HTML ha subito molte revisioni, ampliamenti e miglioramenti, che sono stati indicati secondo la classica numerazione usata per descrivere le versioni dei software. Attualmente l'ultima versione disponibile è la versione 4.01, resa pubblica il 24 dicembre 1999. Dopo un periodo di sospensione, in cui il W3C si è focalizzato soprattutto sulle definizioni di XHTML (applicazione a HTML di regole e sintassi in stile XML) e dei fogli di stile (CSS), nel 2007 è ricominciata l'attività di specifica con la definizione, ancora in corso, di HTML5, attualmente allo stato di bozza all'ultima votazione.

Un'ulteriore ed importante caratteristica di HTML è che esso è stato concepito per definire il contenuto logico e non l'aspetto finale del documento. I dispositivi che possono accedere ad un documento HTML sono molteplici e non sempre dotati di potenti capacità grafiche. Proprio per questo gli sviluppatori di HTML hanno optato per un linguaggio che descrivesse dal punto di vista logico, piuttosto che grafico, il contenuto dei documenti. Questo significa che non esiste alcuna garanzia che uno stesso documento venga visualizzato in egual modo su due dispositivi. Se da una parte questo ha imposto in passato dei forti limiti agli sviluppatori di pagine Web, ha dall'altro garantito la massima diffusione di Internet ed evitato che essa diventasse un medium di élite.

Attualmente i documenti HTML sono in grado di incorporare molte tecnologie, che offrono la possibilità di aggiungere al documento ipertestuale controlli più sofisticati sulla resa grafica, interazioni dinamiche con l'utente, animazioni interattive e contenuti multimediali. Si tratta di linguaggi come CSS, JavaScript e jQuery, XML, JSON, o di altre applicazioni multimediali di animazione vettoriale o di streaming audio o video.

HTML soffre di limiti propri di un sistema di contrassegno ideato per scopi molto lontani da quelli attualmente richiesti dal Web Design [8]. L'HTML è stato progettato per gestire singole pagine. Se ciò ha generato la definizione di un linguaggio semplice e diretto, non è più aderente alle necessità di siti spesso composti da migliaia di pagine. A questo si affianca il problema incontrato dai grafici professionisti riguardo la difficoltà di posizionamento all'interno del documento di immagini e testo; questi limiti risultano fastidiosi e spesso immobilizzanti, per i professionisti.

Posizionare un'immagine, creare una banda laterale, giustificare del testo in HTML diventa un problema risolvibile esclusivamente con strumenti nati per tutt'altro scopo.

A queste problematiche HTML 4.0 cerca di far fronte adottando i fogli di stile, i quali possono essere inseriti all'interno di un documento iperstestuale in tre modi:

1. attraverso un file esterno;
2. attraverso l'intenstazione generale di ogni singolo documento;
3. attraverso la specifica degli attributi nel singolo TAG.

Sfruttando i fogli di stile è possibile modificare la struttura di diverse pagine agendo su un unico file esterno che ne determina l'aspetto. La marcatura HTML dovrà, quindi, tornare alla sua originaria funzione di definizione della struttura logica del documento, lasciando ai fogli di stile la gestione del layout di pagina. Alcuni linguaggi specializzati come il CSS (Cascading Style Sheets) permettono di riunire direttive di stile provenienti da fonti diverse, facendo prevalere le une alle altre in base alle scelte del webmaster.

Ecco un esempio di come è possibile a una semplice pagina HTML un codice CSS che ne determini lo stile e di conseguenza il layout.

3.1.1 Esempio Codice HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 4.01//EN">
<html>
<head>
  <title>My first styled page</title>
</head>

<body>

<!-- Site navigation menu -->
<ul class="navbar">
  <li><a href="index.html">Home page</a>
  <li><a href="musings.html">Musings</a>
  <li><a href="town.html">My town</a>
  <li><a href="links.html">Links</a>
</ul>
<!-- Main content -->
<h1>My first styled page</h1>

<p>Welcome to my styled page!</p>

<p>It lacks images, but at least it has style. And it has links, even if they don't go anywhere</p>

<p>There should be more here, but I don't know what yet.</p>


</body>
</html>
```

Figura 6 - *Esempio di codice HTML con un elenco, un titolo e alcuni paragrafi.*

3.1.2 Esempio Regole CSS

```
body {
    padding-left: 11em;
    font-family: Georgia, "Times New Roman", Times, serif;
    color: purple;
    background-color: #d8da3d
}
ul.navbar {
    list-style-type: none;
    padding: 0;
    margin: 0;
    position: absolute;
    top: 2em;
    left: 1em;
    width: 9em
}
ul.navbar li {
    background: white;
    margin: 0.5em 0;
    padding: 0.3em;
    border-right: 1em solid black
}
ul.navbar a {
    text-decoration: none
}
```

Figura 7 - *Assegnazione delle regole CSS alla pagina HTML.*

3.2 Tecnologie client-side

3.2.1 Javascript

JavaScript è un linguaggio di scripting orientato agli oggetti comunemente usato nei siti web [9]. Fu originariamente sviluppato da Brendan Eich della Netscape Communications con il nome di Mocha e successivamente di LiveScript, ma in seguito è stato rinominato "JavaScript" ed è stato formalizzato con una sintassi più vicina a quella del linguaggio Java di Oracle. JavaScript è stato standardizzato per la prima volta tra il 1997 e il 1999 dalla ECMA con il nome ECMAScript. L'ultimo standard, di marzo 2011, è ECMA-262 Edition 5.1. È anche uno standard ISO.

A differenza di altri linguaggi, quali il C o il Java, che permettono la scrittura di programmi completamente stand-alone, JavaScript viene utilizzato soprattutto in quanto linguaggio di scripting, integrato, quindi, all'interno di un altro programma. L'idea di base è che il programma ospite (quello che ospita ed esegue lo script) fornisca allo script un'API ben definita, API che consente l'accesso ad operazioni specifiche, la cui implementazione è a carico del programma ospite stesso. Lo script, quando eseguito, utilizza riferimenti a questa API per richiedere (al programma ospite) l'esecuzione di operazioni specifiche, non previste dai costrutti del linguaggio JavaScript in sé. In effetti, questo è esattamente lo stesso meccanismo che viene adottato anche in un linguaggio quale il C o il Java, nel quale il programma si affida a delle librerie, non previste dal linguaggio in sé, che permettono di effettuare operazioni quali l'I/O o l'esecuzione di chiamate a funzioni di sistema.

L'esempio tipico (e, forse, il più noto) di programma ospite per uno script JavaScript è quello del browser. Un browser tipicamente incorpora un interprete JavaScript; quando viene visitata una pagina web che contiene il codice di uno script JavaScript, quest'ultimo viene portato in memoria ed eseguito dall'interprete contenuto nel browser.

Le interfacce che consentono a JavaScript di rapportarsi con un browser sono chiamate DOM (Document Object Model). Molti siti web usano la tecnologia JavaScript lato client per creare potenti applicazioni web dinamiche.

Un uso principale del Javascript in ambito Web è la scrittura di piccole funzioni integrate nelle pagine HTML che interagiscono con il DOM del browser per compiere determinate azioni non

possibili con il solo HTML statico: controllare i valori nei campi di input, nascondere o visualizzare determinati elementi, ecc.

Sfortunatamente, gli standard DOM imposti dal W3C non sempre vengono rispettati dai vari browser: browser diversi (anche a seconda del loro motore di rendering) espongono diversi oggetti o metodi allo script (Internet Explorer è solito aderire agli standard con piccole modifiche, e tratta ad esempio l'oggetto event come globale; Opera non supporta le funzioni alert() e confirm()), ed è quindi spesso necessario implementare controlli aggiuntivi ad una funzione JavaScript, per garantirne la compatibilità con ciascun browser.

3.2.2 AJAX

AJAX, acronimo di Asynchronous JavaScript and XML, è una tecnica di sviluppo per la realizzazione di applicazioni web interattive (Rich Internet Application) [10]. Lo sviluppo di applicazioni HTML con AJAX si basa su uno scambio di dati in background fra web browser e server, che consente l'aggiornamento dinamico di una pagina web senza esplicito ricaricamento da parte dell'utente. AJAX è asincrono nel senso che i dati extra sono richiesti al server e caricati in background senza interferire con il comportamento della pagina esistente. Normalmente le funzioni richiamate sono scritte con il linguaggio JavaScript. Tuttavia, e a dispetto del nome, l'uso di JavaScript e di XML non è obbligatorio, come non è necessario che le richieste di caricamento debbano essere necessariamente asincrone.

AJAX è una tecnica multi-piattaforma utilizzabile su molti sistemi operativi, architetture informatiche e browser web, ed esistono numerose implementazioni open source di librerie e framework.

La tecnica Ajax utilizza una combinazione di:

- HTML (o XHTML) e CSS per il markup e lo stile;
- DOM (Document Object Model) manipolato attraverso un linguaggio ECMAScript come JavaScript o JScript per mostrare le informazioni ed interagirvi;
- l'oggetto XMLHttpRequest per l'interscambio asincrono dei dati tra il browser dell'utente e il web server. In alcuni framework Ajax e in certe situazioni, può essere usato un oggetto Iframe invece di XMLHttpRequest per scambiare i dati con il server e, in altre implementazioni, tag <script> aggiunti dinamicamente (JSON).

In genere viene usato XML come formato di scambio dei dati, anche se di fatto qualunque formato può essere utilizzato, incluso testo semplice, HTML preformattato, JSON e perfino EBML. Questi file sono solitamente generati dinamicamente da script lato server.

3.2.3 jQuery

jQuery è una libreria di funzioni (framework) javascript, cross-browser per le applicazioni web, che si propone come obiettivo quello di semplificare la programmazione lato client delle pagine HTML [11].

Pubblicato, per la prima volta il 22 agosto 2005 da John Resig, ha raggiunto la versione 1 (stabile) il 26 agosto dell'anno successivo.

Tramite l'uso della libreria jQuery è possibile, con poche righe di codice, effettuare svariate operazioni, come ad esempio ottenere l'altezza di un elemento, o farlo scomparire con effetto dissolvenza.

Anche la gestione degli eventi è completamente standardizzata, automatica; stessa cosa per quanto riguarda l'utilizzo di AJAX, in quanto sono presenti alcune funzioni molto utili e veloci che si occupano di istanziare i giusti oggetti ed effettuare la connessione e l'invio dei dati.

Il **core** di jQuery fornisce:

1. I costruttori per l'utilizzo della libreria stessa
    - Per ottenere elementi tramite un selettore (vedere sotto)
    - Per ottenere un elemento referenziandolo come parametro
    - Per creare ex novo un elemento partendo da codice HTML grezzo
2. I metodi e le proprietà per accedere agli elementi contenuti in un oggetto jQuery
    - Per conoscere il numero di elementi (funzione size() oppure proprietà length)
    - Per iterare ogni elemento (funzione each())
    - Per conoscere il selettore utilizzato o l'elemento DOM referenziato (proprietà selector o context)
    - Per ottenere e manipolare il/gli elementi nativi (funzioni get() e index())
3. I metodi per creare e utilizzare liste e code (di oggetti o anche funzioni)
4. I metodi per estendere il framework mediante plugin (funzione extend() e fn.extend())

5. I metodi per eseguire della animazioni tramite le funzioni show(),hide() e animate()

3.2.4 jQuery Mobile

Solitamente quando pensiamo ad applicazioni per dispositivi mobile, pensiamo ad applicazioni native, cioè scaricabili ed installabili su uno specifico dispositivo [12].

In effetti lo sviluppo di applicazioni native resta ancora oggi il miglior modo per sfruttare pienamente le potenzialità di smartphone e tablet ed offre un'esperienza utente integrata e coerente fra tutte le applicazioni.

Il problema di questo approccio è che per supportare più di un sistema operativo è necessario sviluppare diverse applicazioni, moltiplicando così i tempi di realizzazione ed il codice da mantenere.

La migliore risposta a questo problema arriva oggi con i framework per applicazioni web mobile, che hanno come primo obiettivo quello di fornire un'esperienza utente il più vicino possibile a quella nativa pur garantendo il supporto ad un largo numero di dispositivi e sistemi operativi diversi. Uno di questi è jQuery Mobile.

Basato su jQuery, jQuery Mobile è un progetto molto giovane, ma offre una buona stabilità ed un numero di funzionalità adeguate per sviluppare applicazioni web mobile complete.

Come accennato, uno dei vantaggi dello sviluppo web mobile è quello di poter realizzare un'unica applicazione fruibile con un largo numero di dispositivi. Pur mirando a questo scenario, gli sviluppatori di jQuery Mobile hanno ben presente che offrire la stessa esperienza utente e le stesse funzionalità al mondo dei dispositivi mobile è un'impresa pressoché impossibile, sia per le peculiarità hardware di ogni device (dimensioni dello schermo, potenza del processore), sia per la difficoltà nell'aggiornare il software, almeno per gli utenti con uno skill tecnico basso.

Proprio per questo hanno scelto un approccio basato su progressive enhancement nel quale i device vengono raggruppati in gruppi con diversi livelli di supporto. In questo modo i device più avanzati fruiranno di un'esperienza ed un'interfaccia più ricca di quelli più obsoleti, tuttavia ambedue potranno comunque fruire dei contenuti e delle funzionalità offerte dall'applicazione.

Sul sito ufficiale di jQuery Mobile è disponibile una griglia, aggiornata all'ultima versione della libreria, nella quale verificare il tipo di supporto offerto.

Questa griglia sarà anche utile nel caso doveste decidere quali device saranno supportati dalla vostra aplicazione. Ricordate che, sebbene esistano dei simulatori, il debug di applicazioni web mobile richiede sempre un controllo fisico sul device e, se amplierete troppo il supporto, potreste ritrovarvi in sessioni di verifica abbastanza snervanti, ne sanno qualcosa gli sviluppatori di jQuery Mobile.

3.3 Tecnologie Server-side

3.3.1 Apache Tomcat

Apache Tomcat (o semplicemente Tomcat) è un contenitore servlet open source sviluppato dalla Apache Software Foundation. Implementa le specifiche JavaServer Pages (JSP) e Servlet di Sun Microsystems, fornendo quindi una piattaforma per l'esecuzione di applicazioni Web sviluppate nel linguaggio Java. La sua distribuzione standard include anche le funzionalità di web server tradizionale, che corrispondono al prodotto Apache [13].

In passato, Tomcat era gestito nel contesto del Jakarta Project, ed era pertanto identificato con il nome di Jakarta Tomcat; attualmente è oggetto di un progetto indipendente.

Tomcat è rilasciato sotto la Licenza Apache, ed è scritto interamente in Java; può quindi essere eseguito su qualsiasi architettura su cui sia installata una JVM.

Componenti di Tomcat:

- Catalina

  Catalina è il contenitore di servlet Java di Tomcat. Catalina implementa le specifiche di Sun Microsystems per le servlets Java e le "JavaServer Pages (JSP, Pagine JavaServer). In Tomcat un elemento del Reame rappresenta un database di usernames, passwords e ruoli (analoghi dei gruppi di UNIX) assegnati a quegli utenti. Differenti implementazioni del Reame permettono a Catalina di essere integrato in ambienti dove tali informazioni di autenticazione sono già state create e supportate, e poi gli permettono di utilizzare tali informazioni per implementare una cosiddetta "Container Managed Security" come descritto nelle Specifiche delle Servlet.[3]

- Coyote

  Coyote è il componente "connettore HTTP" di Tomcat. Supporta il protocollo HTTP 1.1 per il web server o per il contenitore di applicazioni. Coyote ascolta le connessioni in entrata su una specifica porta TCP sul server e inoltra la richiesta al Tomcat Engine per processare la richiesta e mandare indietro una risposta al client richiedente.

- Jasper

  Jasper è il motore JSP di Tomcat. Tomcat 5.x utilizza in realtà Jasper 2, che è un'implementazione delle specifiche 2.0 delle Pagine JavaServer (JSP) di Sun Microsystems. Jasper parsa i file JSP per compilarli in codice Java come servlets (che verranno poi gestite da Catalina). Al momento di essere lanciato, Jasper trova i cambiamenti avvenuti ai file JSP e, se necessario, li ricompila.

3.3.2 PostgreSQL

PostgreSQL è un completo database relazionale ad oggetti rilasciato con licenza libera. PostgreSQL è una reale alternativa sia rispetto ad altri prodotti liberi come MySQL, Firebird SQL e MaxDB che a quelli a codice chiuso come Oracle, Informix o DB2 ed offre caratteristiche uniche nel suo genere che lo pongono per alcuni aspetti all'avanguardia nel settore dei database. Un rapido esame di PostgreSQL potrebbe suggerire che sia simile agli altri database. PostgreSQL usa il linguaggio SQL per eseguire delle query sui dati. Questi sono conservati come una serie di tabelle con chiavi esterne che servono a collegare i dati correlati. La programmabilità di PostgreSQL è il suo principale punto di forza ed il principale vantaggio verso i suoi concorrenti: PostgreSQL rende più semplice costruire applicazioni per il mondo reale, utilizzando i dati prelevati dal database [14].

I database SQL conservano dati semplici in "flat table" (tabelle piatte n.d.t.), richiedendo che sia l'utente a prelevare e raggruppare le informazioni correlate utilizzando le query. Questo contrasta con il modo in cui sia le applicazioni che gli utenti utilizzano i dati: come ad esempio in un linguaggio di alto livello con tipi di dato complessi dove tutti i dati correlati operano come elementi completi, normalmente definiti oggetti o record (in base al linguaggio).

La conversione delle informazioni dal mondo SQL a quello della programmazione orientata agli oggetti, presenta difficoltà dovute principalmente al fatto che i due mondi utilizzano differenti modelli di organizzazione dei dati. L'industria chiama questo problema "impedance mismatch" (discrepanza di impedenza): mappare i dati da un modello all'altro può assorbire fino al 40% del tempo di sviluppo di un progetto. Un certo numero di soluzioni di mappatura, normalmente dette "object-relational mapping", possono risolvere il problema, ma tendono ad essere costose

e ad avere i loro problemi, causando scarse prestazioni o forzando tutti gli accessi ai dati ad aver luogo attraverso il solo linguaggio che supporta la mappatura stessa.

PostgreSQL può risolvere molti di questi problemi direttamente nel database. PostgreSQL permette agli utenti di definire nuovi tipi basati sui normali tipi di dato SQL, permettendo al database stesso di comprendere dati complessi. Per esempio, si può definire un indirizzo come un insieme di diverse stringhe di testo per rappresentare il numero civico, la città, ecc. Da qui in poi si possono creare facilmente tabelle che contengono tutti i campi necessari a memorizzare un indirizzo con una sola linea di codice.

PostgreSQL, inoltre, permette l'ereditarietà dei tipi, uno dei principali concetti della programmazione orientata agli oggetti. Ad esempio, si può definire un tipo codice_postale, quindi creare un tipo cap (codice di avviamento postale) o un tipo us_zip_code basato su di esso. Gli indirizzi nel database potrebbero quindi accettare entrambi i tipi, e regole specifiche potrebbero validare i dati in entrambi i casi. Nelle prime versioni di PostgreSQL, implementare nuovi tipi richiedeva scrivere estensioni in C e compilarle nel server di database. Dalla versione 7.4 è diventato molto più semplice creare ed usare tipi personalizzati attraverso il comando "CREATE DOMAIN".

La programmazione del database stesso può ottenere grandi vantaggi dall'uso delle funzioni. La maggior parte dei sistemi SQL permette agli utenti di scrivere una procedura, un blocco di codice, che le altre istruzioni SQL possono richiamare. Comunque l’SQL stesso rimane inadatto come linguaggio di programmazione, pertanto gli utenti possono sperimentare grandi difficoltà nel costruire logiche complesse. Inoltre l’SQL stesso non supporta molti dei principali operatori di base dei linguaggi di programmazione, come le strutture di controllo di ciclo e condizionale. Pertanto ogni venditore ha scritto le sue estensioni al linguaggio SQL per aggiungere queste caratteristiche, e pertanto queste estensioni non per forza operano su diverse piattaforme di database.

3.4 Ambienti di sviluppo

3.4.1 X-Code

X-Code è un ambiente di sviluppo integrato (Integrated development environment, IDE) sviluppato da Apple Inc. per agevolare lo sviluppo di software per Mac OS X e iOS. È fornito gratuitamente in bundle con il sistema operativo a partire da Mac OS X 10.3 Panther, sebbene sia in grado di generare programmi per qualsiasi versione di Mac OS X. Estende e rimpiazza il precedente tool di sviluppo della Apple, Project Builder, che era stato ereditato dalla NeXT. Ufficialmente Xcode non funziona su Mac OS X 10.2 Jaguar [15].

Xcode lavora in congiunzione con Interface Builder (proveniente da NeXT), un tool grafico per realizzare interfacce grafiche.

Xcode include GCC, che è in grado di compilare codice C, C++, Objective C/C++ e Java. Supporta ovviamente i framework Cocoa e Carbon, oltre ad altri.

Una delle caratteristiche tecnologicamente più avanzate di Xcode è che supporta la distribuzione in rete del lavoro di compilazione. Usando Bonjour e Xgrid è in grado di compilare un progetto su più computer riducendo i tempi. Supporta la compilazione incrementale, è in grado di compilare il codice mentre viene scritto, in modo da ridurre il tempo di compilazione.

3.4.2 Eclipse

Eclipse è un ambiente di sviluppo integrato multi-linguaggio e multipiattaforma. Ideato da un consorzio di grandi società quali Ericsson, HP, IBM, Intel, MontaVista Software, QNX, SAP e Serena Software, chiamato Eclipse Foundation sullo stile dell'open source [16].

Eclipse può essere utilizzato per la produzione di software di vario genere, si passa infatti da un completo IDE per il linguaggio Java (JDT, "Java Development Tools") a un ambiente di sviluppo per il linguaggio C++ (CDT, "C/C++ Development Tools") e a plug-in che permettono di gestire XML, Javascript, PHP e persino di progettare graficamente una GUI per un'applicazione JAVA (Eclipse VE, "Visual Editor"), rendendo di fatto Eclipse un ambiente RAD.

Il programma è scritto in linguaggio Java, ma anziché basare la sua GUI su Swing, il toolkit grafico di Sun Microsystems, si appoggia a SWT, librerie di nuova concezione che conferiscono ad Eclipse un'elevata reattività.

La piattaforma di sviluppo è incentrata sull'uso di plug-in, delle componenti software ideate per uno specifico scopo, per esempio la generazione di diagrammi UML, ed in effetti tutta la piattaforma è un insieme di plug-in, versione base compresa, e chiunque può sviluppare e modificare i vari plug-in. Nella versione base è possibile programmare in Java, usufruendo di comode funzioni di aiuto quali: completamento automatico ("Code completion"), suggerimento dei tipi di parametri dei metodi, possibilità di accesso diretto a CVS e riscrittura automatica del codice (funzionalità questa detta di Refactoring) in caso di cambiamenti nelle classi.

Essendo scritto in Java, Eclipse è disponibile per le piattaforme Linux, HP-UX, AIX, Mac OS X e Windows.

3.5 Debugger

Firebug

Firebug è un'estensione di Mozilla Firefox che permette il debug, la modifica e il monitoraggio di tutti gli aspetti di una pagina web, come i fogli di stile, il codice HTML, la struttura DOM e il codice JavaScript. Firebug fornisce anche altri strumenti per lo sviluppo web come una console JavaScript e una funzione chiamata "Net" che permette di monitorare il tempo di caricamento in millisecondi di immagini e script. Oltre che per il debug, Firebug è uno strumento indispensabile per testare la sicurezza e la performance dei siti web [17].

Firebug è uno strumento di sviluppo web che facilita il debugging, la modifica e la verifica di CSS, HTML, DOM, XHR (Xml http Request) e Javascript.

Il pannello di controllo Javascript tiene traccia del log degli errori, chiamate a funzioni e permette allo sviluppatore di eseguire codice Javascript arbitrariamente.

Dal pannello di controllo NET mostra ogni chiamata http effettuata dal browser come CSS esterni, Javascript, o immagini. Per ogni risorsa richiesta è possibile verificare anche i relativi request header e response header.

4 - Urbanopoly

4.1 Introduzione a Urbanopoly

Urbanopoly è un gioco per piattaforma Android che raccoglie dati sulla città di Milano e la Regione Lombardia. Il gioco è stato sviluppato da DEI e Cefriel nell'ambito del progetto europeo Planet Data [18] [19].

Disponibile su App Store Google Play, è gratuito ed è in grado di raccogliere informazioni di qualità sugli ambienti urbani. Questa applicazione per cellulare è attualmente disponibile per i giocatori che si spostano a Milano e nella Regione Lombardia, ma altre città saranno presto aggiunte. Durante il gioco, gli utenti sono invitati a fornire preziose informazioni circa l'ambiente urbano in cui si stanno muovendo. Lo scopo del gioco è acquisire locali commerciali e a guadagnare denaro. I luoghi sono sedi reali che il giocatore ha selezionato da OpenStreetMap, come ad esempio negozi, ristoranti e monumenti.

L'applicazione mostra la classifica dei migliori giocatori, che possono condividere le loro sedi di proprietà sulla propria bacheca di Facebook, promuovendo così una dimensione sociale del gioco. In effetti, Urbanopoly incentiva la concorrenza tra i giocatori e stimola anche i loro amici di Facebook a partecipare al concorso.

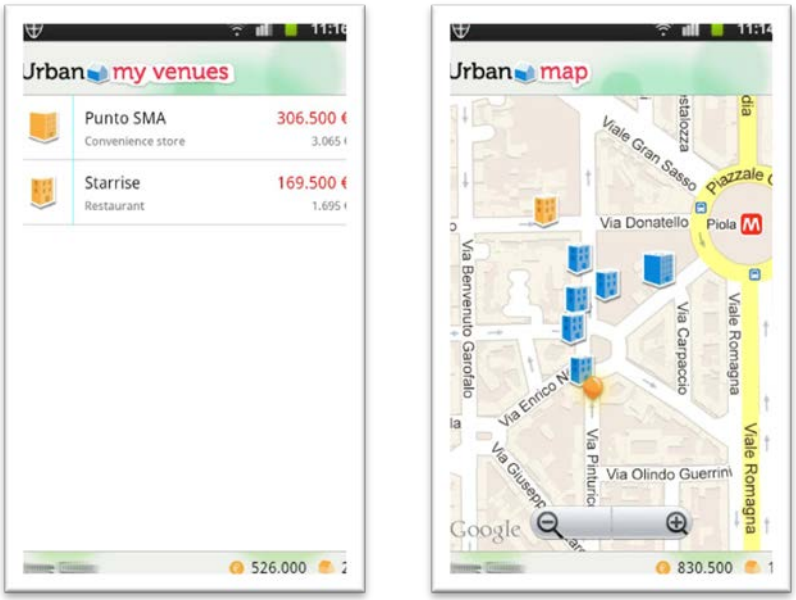

Figura 8 - *Screenshot di Urbanopoly: I miei luoghi e Mappa.*

Per ampliare le proprie proprietà è possibile **visitare** i luoghi disponibili tramite la mappa di gioco che automaticamente si posizionerà sulla propria posizione corrente sfruttando il GPS dando successivamente la possibilità di visualizzare e selezionare un luogo vicino. Sono possibili diversi casi: il luogo può essere libero ed è dunque possibile acquisirlo a patto di avere sufficiente denaro. Altrimenti il luogo può appartenere a un altro giocatore, in tal caso bisognerà far ruotare una ruota che deciderà la prossima azione da intraprendere.

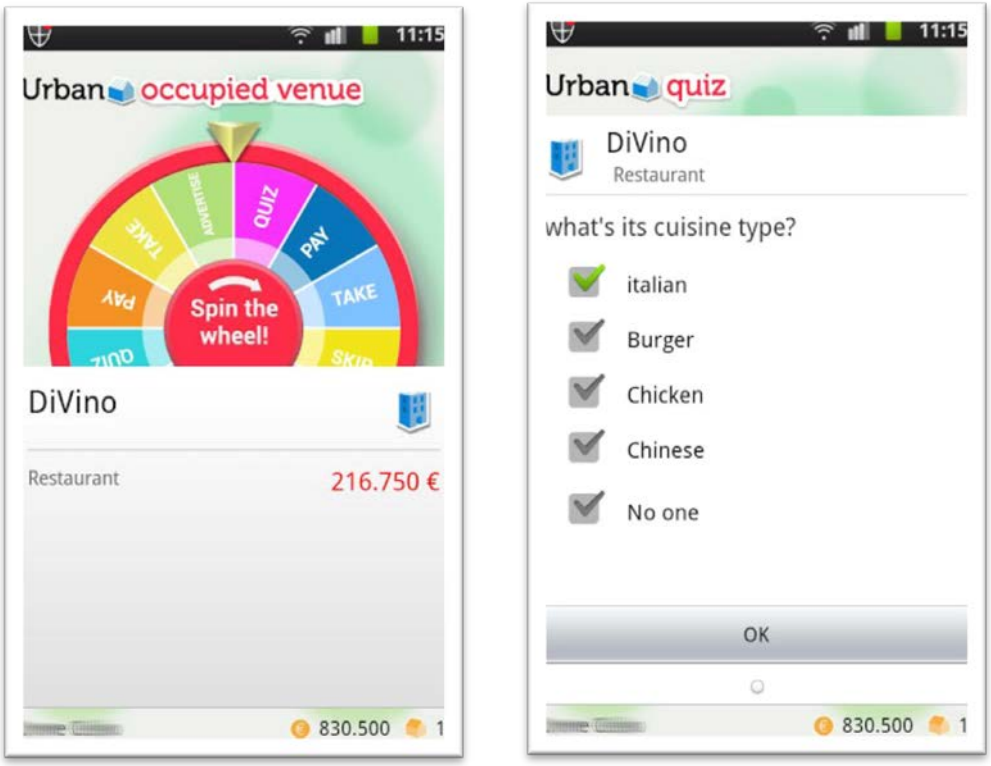

Figura 9 - *Screenshot di Urbanopoly: Ruota e Quiz.*

Tra le azioni possibili c'è la possibiità di guadagnare denaro rispondendo ai sondaggi o ai quiz proposti dal gioco. Se si è più fortunati si può avere la possibilità di soffiare il luogo al proprietario o come ultima possibilità, nel caso più sfortunato, bisognerà pagargli un dazio.

4.2 Gameplay e regole di gioco

Il menu principale dell'applicazione presenta 6 voci: I miei luoghi, Mappa, Notifiche, Classifica, Tutorial e Crediti.

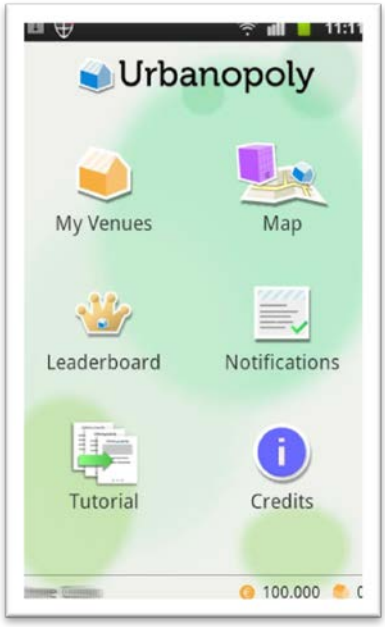

Figura 10 – *Screenshot di Urbanopoly: menù principale.*

Come già accennato, il meccanismo di gioco prevede la possibilità di acquisire nuove proprietà a partire da luoghi "liberi" selezionati sulla mappa o tramite un azione di tipo "**TAKE**" in caso di visita su un luogo già occupato. In quest'ultimo caso il prezzo da pagare per acquisire il luogo è pari al 150% del suo valore standard.

In fase di acquisizione di un luogo il gioco chiederà in ogni caso di compilare i dati relativi al NOME del luogo e alla categoria di appartenenza (es. Trasporti -> Fermata di mezzi pubblici). Successivamente il luogo entrerà a far parte dei propri luoghi e il proprio credito scalato del valore opportuno.

Nel tentativo di acquistare un luogo di valore superiore ai crediti disponibili l'applicazione bloccherà l'acquisto proponendo la possibilità di **vendere** o **ipotecare** un proprio luogo per poter acquisire il luogo.

La vendita di un luogo, molto semplicemente, permette di riacquisire fondi pari all'intero valore del luogo. L'effetto sarà quindi la perdita di proprietà del luogo stesso e un incremento del denaro a disposizione per altri acquisti.

L'ipoteca di un luogo, invece, permette di recuperare immediatamente metà del valore della proprietà (es. per avere subito a disposizione dei fondi per un nuovo acquisto) con l'obiettivo di poterlo **riscattare** entro 24 ore. In caso contrario verrà persa anche l'altra metà del luogo considerato.

Le azioni di maggiore interesse, e legati agli obiettivi degli sviluppatori, sono certamente quelle di **QUIZ** e **ADVERTISE** (sondaggio). Si tratta di possibili risultati della "Ruota della fortuna" che si può quindi ottenere in caso di visita a un luogo in possesso di un altro giocatore.

Il giocatore è incentivato a rispondere alle domande per poter ottenere denaro, allo stesso tempo il server che coordina il gioco si occupa della raccolta delle informazioni che i giocatori forniscono.

I quiz prevedono delle domande a risposta chiusa dove la risposta può essere singola o multipla. Il numero delle domande è variabile con un minimo di una domanda.

Il sondaggio, oltre alle opzioni proposte nei quiz, integra la possibilità di rispondere in maniera aperta alle domande, quindi inserendo del testo. Oltre alle domande che possono variare per numero e per contenuto, in ogni caso il sondaggio chiede il nome del luogo e la sua categoria (come visto in precedenza in fase di acquisizione) e, infine, richiede di fare una fotografia del luogo stesso. Una volta compilato un sondaggio è possibile visualizzare un **poster** col dettaglio delle risposte fornite. Nei sondaggi è prevista la possibilità di saltare la risposta alle domande (non si può invece saltare l'immissione del nome del luogo e della categoria).

Nel caso più sfortunato la ruota può produrre un risultato di tipo **PAY** che quindi costringerà il giocatore a dover pagare una tassa al legittimo proprietario.

Ultimo risultato possibile della ruota è l'azione **SKIP** che di fatto da la possibilità di tornare alla mappa senza fare nulla.

L'ultima azione possibile è chiamata **JUDGE** e consiste nell'attribuzione di un voto da 0,5 a 5 (previsti i mezzi voti) a un poster creato.

Le altre voci menù che si possono considerare di "contorno" sono le notifiche, la classifica, il tutorial e i crediti, si tratta di componenti essenzialmente visuali.

Le **notifiche**, gestite dal server centrale, informano il giocatore di eventuali guadagni generati dai propri luoghi, perdita di luoghi (dovuto a ipoteche non riscattate o all'acquisizione da parte di altri giocatori) o altro tipo di informazioni.

La **classifica** segue un criterio di ordinamento ottenuto dalla somma dei valori dei proprio luoghi e del denaro disponibile. La visualizzazione mostra i primi 3 giocatori in senso assoluto e un intorno della posizione del giocatore corrente.

Il **tutorial** consiste in una spiegazione visuale degli delle varie fasi di gioco, utile per chi si trova a giocare per la prima volta.

La sezione crediti fornisce dettagli su chi si è occupato della realizzazione dell'applicazione.

4.3 Architettura e flusso di comunicazione

L'architettura dell'applicazione prevede l'esistenza di un server centrale a cui i client (i dispositivi mobile) si connettono. Il server gira su piattaforma Apache Tomcat ed è connessa a un database PostgreSQL. Esso agisce come fosse un web service, fornisce infatti una serie di chiamate che i client invocheranno, coi relativi parametri, per poter ottenere tutte le informazioni necessarie alla corretta esecuzione dell'applicazione. Gli oggetti scambiati sono in formato JSON, formato dati che si presta particolarmente alla gestione tramite Java (parlando della versione per Android) e, come nel nostro caso, alla gestione tramite JavaScript (nella nostra applicazione basata su Apache Cordova). JSON è per l'appunto l'acronimo di JavaScript Object Notation.

Nella fattispecie i metodi offerti dal server sono i seguenti:

- **GetHighcores**: Servizio richiamato per ottenere la classifica generale del gioco. Si ottengono sempre 10 risultati tra cui: i primi 3 giocatori assoluti, il giocatore attuale e 6 giocatori nell'intorno della posizione del giocatore.

  INPUT: uid, e informazioni sulle soglie per specificare un intorno
  OUTPUT: vettore di oggetti 'score'.

- **GetNotifications**: restituisce le notifiche non ancora inviate al giocatore. Non appena inviate il server setta queste notifiche come viste e non verranno più inviate al giocatore. Esistono diversi tipi di notifiche e per ognuno di queste esistono dei messaggi preimpostati da visualizzare all' utente.

  INPUT: uid giocatore.
  OUTPUT: un array di notifiche non ancora inviate e lo stato del giocatore aggiornato. Il server applica le notifiche. Unica funzione è notificare all'utente circa i cambiamenti avvenuti.

- **GetPlayArea**: è la funzionalità principale del gioco. Restituisce l'area e i luoghi con cui il giocatore potrà interagire. Il numero di luoghi restituite e la distanza massima vengono calcolate dal server in base all'area di gioco. Anche in questo caso bisogna aggiornare lo stato del giocatore e i luoghi anche se già scaricate In precedenza. Con questo servizio vengono generate e restituite le visite QUIZ e ADVERTISE. Le alte visite tranne quelle JUDGE devono

essere generate dal client. L'utente potrà eseguire solo una visita per ogni "giorno gioco" in un determinato luogo.

INPUT: uid, latitudine e longitudine
OUTPUT: array di venue, alcuni array di visite possibili e il giorno gioco e il giocatore aggiornato

- **GetPlayerVenues** serve ad ottenere le proprietà del giocatore aggiornate.
  INPUT**:** uid
  OUTPUT: array venue dell'utente, array di visite JUDGE se disponibili, e l'utente aggiornato

- **GetVersion** è la prima richiesta e restituisce la versione corrente del server. Se la versione del client è diversa bisogna aggiornare uno dei 2.

- **PlayerSubscribe** serve a registrar l'utente, è la seconda chiamata da eseguire ogniqualvolta che viene avviata l'applicazione sul device in quanto serve ad aggiornare o creare l'utente.
  INPUT: uid facebook e nome utente
  OUTPUT: se l'utente esiste già restituisce lo stato del giocatore aggiornato se no lo crea e restituisce lo stato dell'utente nuovo.

- **SetVisits** serve a inviare le azioni dell'utente al server.
  INPUT: un array di visite. Se la visita contiene il campo contenuto offensivo invia una mail al controllore
  OUTPUT: stato dell'operazione. Se fallisce bisogna reinviare le visite

- **UploadPhoto** riceve e salva le foto.

4.4 Dati e oggetti (Model)

4.4.1 Oggetto Stato

L'oggetto Stato (stato.js) è l'oggetto che gestisce le informazioni essenziali di un giocatore quali l'id, il suo nome, il numero dei suoi luoghi e il suo denaro a disposizione. E' definito come segue:

```
function Stato (id,nome,money,numVenues) {
    this.id=id;
    this.nome=nome;
    this.money=money;
    this.numVenues=numVenues;
}
```

Figura 11 - *Codice sorgente: Oggetto Stato.*

4.4.2 Oggetto Venue

L'oggetto Venue (venue.js) gestisce tutti i dati e i metodi relativi al luogo. Esso mantiene infatti le informazioni ricevute dal server come il suo nome, la sua categoria (tramite id), l'id dell'eventuale proprietario e, tra le altre cose, la probabilità col quale generare l'estrazione di una singola visita nella Ruota in caso di luogo già occupato. L'algoritmo usato per la generazione di un azione basata sulle probabilità fornite è il seguente:

```
this.randomAction = function() {
    if (this.data.wheel==null) return null;
    var numeroEstratto=Math.random()*100;
    var confronto=this.data.wheel.takePercentage;
    if (numeroEstratto<=confronto) return "TAKE";
    confronto+=this.data.wheel.skipPercentage;
    if (numeroEstratto<=confronto) return "SKIP";
    confronto+=this.data.wheel.payPercentage;
    if (numeroEstratto<=confronto) return "PAY";
    confronto+=this.data.wheel.advertisePercentage;
    if (numeroEstratto<=confronto) return "ADVERTISE";
    return "QUIZ";
}
```

Figura 12 - *Codice sorgente: Oggetto Venue.*

4.4.3 Oggetto Categoria

Le categorie sono salvate lato client a livello JavaScript (category.js). In esso è salvata ogni categoria con relativo id, descrizione e parente (ossia la categoria della quale la categoria

selezionata fa parte) con le relative traduzioni. Se la categoria non ha il padre (quindi è per forza una categoria generale) non avrà un parametro "parent" definito. Ecco un esempio:

```
{
    "parent_en" : "Food and drink",
    "parent_it" : "Cibo e Bevande",
    "name_en" :   "Bar",
    "bool" :      "TRUE",
    "id" :     "582",
    "name_it" :   "Bar",
    "type" :      "Item"
},
{
    "name_en" : "Food and drink",
    "id" :   "5",
    "name_it" : "Cibo e Bevande",
    "type" :    "Item"
},
```

Figura 13 - *Codice sorgente: Oggetto Categoria.*

Il controller di gioco caricherà le categorie in fase di avvio di applicazione. Per la precisione controllerà la lingua del dispositivo corrente e, una volta ottenuta, caricherà in memoria le categorie nella lingua scelta.

```
globalU.playMap.categories=new Array();
globalU.playMap.categoryMapping=new Array();
for (var i=0; i<globalU.categories.length;i++) {
    if (scrivi.lingua=="IT")
        globalU.playMap.categories[parseInt(globalU.categories[i].id)]={
            "id":parseInt(globalU.categories[i].id),
            "name": globalU.categories[i].name_it,
            "parent": globalU.categories[i].parent_it,
            "bool": globalU.categories[i].bool
        }
    else
        globalU.playMap.categories[parseInt(globalU.categories[i].id)]={
            "id":parseInt(globalU.categories[i].id),
            "name": globalU.categories[i].name_en,
            "parent": globalU.categories[i].parent_en,
            "bool": globalU.categories[i].bool
        }
    globalU.playMap.categoryMapping[i]=globalU.playMap.categories[parseInt(globalU.categories[i].id)];
}
```

Figura 14 - *Codice sorgente: Assegnamento delle categorie in base alla lingua.*

4.4.4 Oggetto Visit

Gli oggetti di tipo Visit (visit.js per gli oggetti semplici, question.js per gli oggetti complessi) sono gli oggetti scambiati col server e generati dal client per comunicare al server l'esito di un azione. Gli oggetti Visit di tipo semplice sono BUY, TAKE, SELL, MORTGAGE e PAY, quelli di tipo complesso sono QUIZ, ADVERTISE e JUDGE.

```
function visitSell (venue,player,date,value) {
    if (typeof(value)=="undefined") value=venue.getSellValue();
    this.data={
        "type":"SELL",
        "player":player,
        "venue":venue.data.id,
        "date": date,
        "deltaMoney":parseInt(value),
        "sent":0,
        "offensive":false
    }
}
```

Figura 15 - *Oggetto Visit di tipo semplice: Vendita.*

La generazione di un oggetto di tipo ADVERTISE, QUIZ o JUDGE è ben più complessa, la sua forma finale sarà però la seguente:


```
{  "type": "ADVERTISE",
    "player": "706659389",
    "skips": [true, false, false],
    "venue": 922,
    "date": {"year": 2012,"month": 10,"dayOfMonth": 6,"hourOfDay": 18,"minuute": 54,"second": 53},
    "deltaMoney": 12000, "sent": 0,
    "venueName": "Nome della venue ",
    "offensive": false, "finishAt": -1,
    "venueCategory": 603,
    "featureTypes": [{
        "advertiseQuestion": "what is the name of the street?",
        "featureRange": {
            "values": null,
            "openDataType": "null",
            "type": "open",
            "id": 133
        },
        "name": "street",
        "multivalue": false,
        "id": 326
    }, {
        "advertiseQuestion": "what is the house number?",
        "featureRange": {
            "values": null,
            "openDataType": "null",
            "type": "open",
            "id": 133
        },
        "name": "housenumber",
        "multivalue": false,
        "id": 324
    }],
    "featureValues": ["", "21"]
}
```


Figura 16 - *Oggetto Visit di tipo complesso: Advertise.*

4.5 Logica di gioco (Controller)

4.5.1 Controllore generale

Il controllore generale (controller.js) ha molteplici funzioni: istanzia tutti gli oggetti e controllori indispensabili al funzionamento e altre operazioni di inizializzazione (inizializzazione oggetti dedicati a interagire con Facebook, col Server remoto, il controllore di gestione del codice e degli elementi html, il controllore delle logiche di gioco, il caricamento delle categorie).

Il lavoro di questo componente è importantissimo, funge infatti da connettore di tutti gli altri componenti dell'applicazione: gestisce quindi i flussi applicativi di gioco, parte della logica, e viene richiamato ogni volta che vengono ricevuti dati dall'oggetto Server, o che l'oggetto View (che verrà successivamente approfondito) gestisce il cambio di una pagina in una nuova. E' il centro nevralgico dell'applicativo il suo scopo è prevalentemente connettivo, la logica infatti è per gran parte demandata a altri oggetti.

Nell'esempio è presente la porzione di codice che viene richiamata quando il server riceve dei dati e il controllore gestisce le attività da eseguire.

```
unction callbackServer (request,esito,dati) {
   console.log(request+" "+esito);
   if (request == "checkversion") {
       checkVersion();
   }
   if (request == "serverLogin") {
       if (esito) {
               globalU.myStatus=dati;
               globalU.view.updatePersonalInfo(dati,null);
               globalU.server.getPlayerVenues(globalU.facebook.data.uid);
               globalU.server.getNotifications(globalU.facebook.data.uid);
               globalU.view.loadMainMenu();
               globalU.view.openPage("logged");
               globalU.view.setCurrentPage(globalU.view.mainmenu);
       }
   }
   if (request == "playarea") {
       if (esito) {
           globalU.myStatus=dati[2];
           globalU.view.updatePersonalInfo(dati[2],null);
           PlayMap.callbackPlayArea(dati[0],dati[1]);
       }
   }
   if (request=="playervenues") {
       if (esito) {
           globalU.myStatus=dati[2];
           globalU.view.updatePersonalInfo(dati[2],null);
           PlayMap.callbackPlayerVenues(dati[0],dati[1]);
       }
   }
```

Figura 17 - *Il server una volta eseguita l'operazione richiama il controllore generale che gestisce il flusso del processo.*

4.5.2 Controller PlayMap

Anche il controllore PlayMap (playmap.js), come il precedente, è di fondamentale importanza.

Esso gestisce tutte le interazioni che vengono fatte con la mappa, mantiene le informazioni sulle Venue e sulle categorie presenti e la loro associazione.

Di conseguenza si occupa della generazione dei marker di ogni Venue sulla mappa e della generazione dei relativi oggetti Visit semplici e delle relative conseguenze (es. rimuovere una venue venduta, accreditare i soldi al profilo del giocatore, ecc.).

In quanto controllore della mappa interagisce anche direttamente con le API di Apache Cordova nell'acquisizione delle coordinate GPS mostrando successivamente i luoghi circostanti.

```
this.getCoordinate = function () {
    printOut("Rilevo la posizione....");
    this.gpsError=0;
    navigator.geolocation.getCurrentPosition(PlayMap.callbackPosition,
    PlayMap.errorCallbackPosition,{ timeout: globalU.gps_timeout,
    enableHighAccuracy: true
    });
}
PlayMap.callbackPosition = function (position) {
    printOut("Posizione ottenuta");
    var obj=globalU.playMap;
    obj.position=position;
    obj.refreshPosition();
}

PlayMap.errorCallbackPosition = function (PositionError) {
    var obj=globalU.playMap;
    obj.gpsError++;
    if (obj.gpsError<globalU.tentativi_gps) {
        navigator.geolocation.getCurrentPosition(PlayMap.callbackPosition,
        PlayMap.errorCallbackPosition,{ timeout: globalU.gps_timeout,
        enableHighAccuracy: true });
        printOut("Rilevo la posizione, nuovo tentativo....");
    }
    else console.log("Impossibile identificare la posizione corrente");
}
```

Figura 18 - *Interazione con il sensore GPS e aggiornamento della posizione corrente.*

4.5.3 Interazione col Server

L'oggetto Server (server.js) si occupa di interagire appunto col server remoto, ne richiama i servizi, e riceve le informazioni in formato JSON che poi vengono restituiti al controllore generale.

Nel caso in cui più richieste al server siano richieste in contemporanea, esse verranno gestite tramite un meccanismo di coda in modo che non vengano eseguite in sovrapposizione. L'oggetto gestisce inoltre tentativi di riconnessione in caso di raggiungimento di un opportuna soglia di timeout. La soglia e il numero di tentativi di connessione sono configurabili dal file di configurazione config.js.

```
this.getPlayArea=function (uid,lat, lon, distance,cb) {

    if (typeof(cb) == "undefined") cb=callbackServer;
    var url;
    if (distance) url=this.getUrl()+ "/GetPlayArea?uid="+uid+"&lat="+lat+"&lon="+lon+"&maxDist="+distance;
    else url=this.getUrl()+ "/GetPlayArea?uid="+uid+"&lat="+lat+"&lon="+lon+"&maxDist="+globalU.maxDist;

    this.requestList.push(new Request("playarea","get",url,cb));
    this.nextRequest();
}
this.sendPicture=function (uid,venueId,imgUrl) {
    var options = new FileUploadOptions();
    options.fileKey="photo";
    options.fileName=imgUrl.substr(imgUrl.lastIndexOf('/')+1);
    options.mimeType="application/octet-stream";
    options.params= {
        uid: uid,
        venueId:venueId
    }
    var ft = new FileTransfer();
    ft.upload(imgUrl, encodeURI(this.getUrl()+ "/UploadPhoto"),
    Server.callbackPictureSuccess,Server.callbackPictureFail, options);
}
```

Figura 19 - *Metodi di acquisizione dei luoghi adiacenti alla posizione sulla mappa e di invio delle foto scattate al server web.*

4.5.4 Integrazione con Facebook

L'oggetto Facebook (facebook.js) sfrutta la Facebook SDK rilasciata per favorire l'integrazione delle mobile application con Facebook. Per poterla utilizzare è necessario registrare l'applicazione sul sito http://developers.facebook.com/ [20] per potere ottenere un App ID valido.

Una volta effettuata questa operazione e una volta predisposta l'applicazione per poter supportare il plugin di facebook (operazione da eseguire a livello di codice nativo, nella fattispecie con Eclipse o X-Code) sarà semplice ottenere le informazioni dall'account collegato e gestire i permessi di pubblicazioni di contenuto sulla propria bacheca (nel caso si voglia condividere l'acquisizione di un luogo per esempio).

La procedura di integrazione del progetto Cordova con Facebook è spiegata passo passo al link https://github.com/davejohnson/phonegap-plugin-facebook-connect/ [21].

4.6 Visualizzazione (View)

L'oggetto View (View.js) gestisce la visualizzazione la modifica e la rimozione degli elementi html fornendo appositi metodi di creazione e impaginazione degli elementi al controllore. Sono stati alcuni metodi statici che permettono la creazione di bottoni dello stesso "tipo" come ad esempio il pulsanti Azione o i pulsanti di tipo Switch To . Questi tipi di pulsanti hanno come azione la chiamata a un metodo del controllore generico che gestisce le azioni successive al tipo di pulsante considerato (tale tipo è specificato a livello di proprietà javascript dell'elemento html). Uno dei metodi chiave di questo oggetto è il metodo Switch To, esso permette di passare da una pagina ad un'altra tenendo traccia dei percorsi seguiti in modo da abilitare l'uso del pulsante Indietro secondo specifiche logiche di gioco.

Esistono pagine di tipo prettamente **statico** dove solo qualche informazione è di carattere dinamico: un esempio è la visualizzazione di una Venue libera dove la parte dinamica è semplicemente composta da parametri come il nome, la categoria, il valore e così via. In questo caso l'oggetto View richiamerà una semplice funzione Javascript che permetterà di assegnare un nuovo contenuto a elementi html con un certo id. Nell'esempio si mostra il prototipo della pagina statica che mostra la scheda della venue.

```
<div id="view-venue" class="content-page">
    <div class="subtitle"><h2 id="view-venue-name"></h2><img src="" alt="" id=
view-venue-icon" class="venue-icon" /></div>
    <div id="view-venue-info">
        <div id="view-venue-caregory-title"></div><div id=
view-venue-caregory-subtitle"></div>
        <table>
            <tr class="main-row"><td id="view-venue-value" class="left-align":
/td><td id="view-venue-salary" class="right-align"></td></tr>
            <tr class="sec-row"><td id="view-venue-value-text" class=
left-align"></td><td id="view-venue-salary-text" class="right-align"></td></tr>
        </table>
    </div>
    <div id="view-venue-judge"></div>
    <div id="view-venue-action"></div>
</div>
```

Figura 20 - *Codice HTML della visualizzazione di un Luogo.*

Altre pagine sono invece prettamente di tipo **dinamico:** gli esempi concreti di questo tipo di pagine sono la pagina "I miei luoghi", "Classifica" e "Notifiche" dove non esiste alcun tipo di contenuto statico ma la pagina viene caricata in maniera completamente dinamica in base al

numero degli elementi da mostrare. Essendo l'intero contenuto caricato dinamicamente e di conseguenza il suo prototipo html è essenzialmente vuoto:

```
<div id="view-luoghi" class="content-page">

</div>
```

Figura 21 - *Codice HTML di "I miei luoghi".*

4.7 Esempi di soluzioni adottate

4.7.1 La ruota

L'oggetto Ruota (Wheel.js) è sicuramente una delle componenti più ardue da implementare con l'utilizzo di Javascript e Html.

Diversi sono gli ostacoli alla realizzazione di questo componente:

- Le forme gestite dagli oggetti HTML hanno sempre forme rettangolari
- La gestione degli eventi "touch" non coincide con la gestione dei tradizionali eventi "click" (in concomitanza dell'assenza di un debugger il problema si amplifica)
- Gli stessi eventi sopracitati agiscono su oggetti HTML quindi di forma rettangolare ma l'azione deve avere effetto solo se avviene all'interno del perimetro circolare e non del box che lo circonda

Il primo punto, come suggerito, viene risolto tramite la creazione di un box esterno di contorno che verrà fatto ruotare intorno al proprio centro (dopo il calcolo delle coordinate di tale centro). La rotazione di un elemento html è possibile grazie alle nuove specifiche CSS 3. Qui di seguito la funzione che si occupa di ruotare un elemento con id "id" a un certo angolo di "rad" gradi.

```
Wheel.ruotaAngolo=function(rad,id) {
        $(id).css("transform","rotate(" + rad + "deg)");
        $(id).css("-moz-transform","rotate(" + rad + "deg)");
        $(id).css("-webkit-transform","rotate(" + rad + "deg)");
        $(id).css("-o-transform","rotate(" + rad + "deg)");
}
```

Figura 22 - *Metodo di rotazione di un elemento html di un certo angolo.*

La soluzione al secondo punto è ottenibile solo a seguito di un lungo lavoro di test e debugging (l'unico modo è richiamando la funzione console.log(variabile) per verificare il comportamento del proprio codice). L'unico aiuto viene offerto da jQuery Mobile che permette di uniformare il controllo degli eventi touch accessibili a prescindere dal device su cui viene eseguito. Le seguenti istruzioni funzionano sia nel caso in cui la piattaforma di utilizzo sia un browser su un normale PC (in tal caso viene colto il click-down, il mouse move e il click-up) che un dispositivo mobile (touch-down, touch-move e touch-up).

```
/* LIBRERIA JQUERY MOBILE */
$(Wheel.idWheel).bind("vmouseup",Wheel.mouseEnd);
$(Wheel.idWheel).bind("vmousedown",Wheel.mouseDown);
$(Wheel.idWheel).bind("vmousemove",Wheel.mouseMove);
```

Figura 23 - *Codice Javascript per catturare gli eventi touch tramite jQuery Mobile.*

L'ultimo punto riguarda la situazione mostrata nella seguente figura, dove la X rossa indica un punto che normalmente verrebbe "intercettato" ma in maniera indesiderata:

Figura 24 - *Immagine del box contenente la ruota. Il punto evidenziato verrebbe considerato valido se non gestito correttamente.*

Impedire che l'evento venga scatenato è impossibile, in quanto appunto HTML e Javascript gestiscono gli elementi di forma rettangolare. L'unica considerazione che può essere fatta è che

si tratta di un cerchio, quindi a distanza dal centro costante, e che quindi l'unico modo per rilevare un touch esterno è ottenere le coordinate del click del mouse, calcolare le coordinate del centro, calcolare la distanza e verificare che non sia superiore a quella del raggio della sfera (che si assume nota a priori). E' stata inoltre posta la condizione che tale distanza non fosse inferiore al raggio del cerchio interno dove non vogliamo che l'evento touch intervenga. Una volta effettuate tali verifiche sarà immediato decidere in quale caso proseguire con l'azione o se ignorare l'evento appena scatenato.

```
Wheel.distanzaAttuale = function () {
    var distanzaA=Wheel.distanza(Wheel.center_left,Wheel.center_top,window.mouseXPos,window.mouseYPos);

    return distanzaA;
}
Wheel.distanza =function (x0,y0,x1,y1) {
    return Math.sqrt(Math.pow((x1-x0),2)+Math.pow((y1-y0),2));
}
Wheel.mouseMove=function(e) {

        Wheel.updateCoordinate(e);
        if (!Wheel.rotationEnabled) return false;
        var a=Wheel.distanzaAttuale();
        if (a>Wheel.distanzaMax || a<Wheel.distanzaMin) {
    //      console.log("Attuale: "+Wheel.distanzaAttuale());
            Wheel.mouseEnd();
            return false;
        }
```

Figura 25 - *Codice Javascript che valida la posizione del touch rispetto alla ruota.*

4.7.2 Pagine a Step

Le pagine a Step sono previste nel gioco durante le sessioni di BUY, TAKE, QUIZ e ADVERTISE.

Sono contraddistinte a livello visuale da icone circolari nel footer della pagina che indicano il numero degli step totali da completare e quelli attualmente completati. In alcuni casi le domande proposte sono a numero fisso, altre sono variabili, esiste anche il caso in cui ci sia una parte fissa e una parte variabile, è il caso dell'ADVERTISE.

Esempi di step a pagine fisse sono il BUY e il TAKE che sono per natura analoghe a livello di risposte da fornire. In entrambi i casi come già spiegato gli step sono 2, nel primo viene richiesto il nome del luogo, nel secondo se ne specifica la categoria.

Il quiz invece ha un numero di step completamente variabili e dipendenti dall'oggetto di tipo Visit che il client riceve dal server e dalle domande in esso presente.

Infine l'ADVERTISE prevede i primi due step fissi e analoghi a quelli di BUY e TAKE, successivamente e' presente un numero variabile di quesiti (generati dal server, solitamente sono 3) e infine lo scatto della foto come ultimo step.

L'oggetto Question (question.js) ha come scopo la gestione di queste diverse pagine a step in un unico oggetto, esso si occuperà di mostrare sempre i corretti elementi HTML in base al tipo di azione da eseguire e al tipo di domanda generata. Il contenuto viene quindi ottenuto in maniera dinamica in quanto deve prevedere tutti i tipi di casi: domande a risposta chiusa a scelta singola, a scelta multipla, a risposta aperta, la scelta della categoria e lo scatto della foto. Una volta posta la domanda nel formato corretto l'oggetto, in base al tipo di domanda generata (tra quelle appena descritte), si occupa ricevere la risposta fornita dall'opportuno elemento html dove le informazioni vengono inserite. Acquisite tutte le risposte e processati tutti gli step verrà creato il corrispettivo oggetto Visit. Le risposte vengono inserite negli opportuni schemi JSON di ciascun tipo di Visita rispettando le caratteristiche di ciascun oggetto e i campi previsti alla raccolta di tali risposte.

Di seguito le funzioni utilizzate per accedere tramite le API Cordova alla fotocamera. La chiamata genererà un file e fornirà ai successivi metodi il suo percorso nel File System:

```
Question.Scatta = function () {
    navigator.camera.getPicture(Question.Scattata, Question.NonScattata,
        quality: 50,
        allowEdit: false,
        targetWidth: 200,
targetHeight: 200,
        destinationType: navigator.camera.DestinationType.FILE_URI)
//  destinationType: navigator.camera.DestinationType.DATA_URL}
    );
}
Question.Scattata=function (fileuri) {
    var obj=globalU.view.question;

    obj.risposta(true,fileuri);

}
Question.NonScattata=function (message) {
    var obj=globalU.view.question;
    obj.risposta(false,null);
}
```

Figura 26 - *Acquisizione di un immagine scattata dalla fotocamera.*

5 – Valutazioni post implementative

Successivamente alla fase di implementazione è doveroso effettuare un'approfondita analisi delle caratteristiche emerse. Si può dire che il Framework abbia soddisfatto appieno i requisiti, le aspettative e le funzionalità disponibili dichiarate.

A seguire vengono analizzate nel dettaglio aspetti relativi all'implementazione delle funzionalità di gioco e quanto essa sia stata agevolata o sfavorita dall'uso di Apache Cordova.

5.1 Analisi di implementazione delle funzionalità di gioco

5.1.1 Introduzione di un sistema Multilingua

Descrizione: Urbanopoly è, nella versione originale Android, fruibile in lingua italiana e in lingua inglese (in funzione delle impostazioni di lingua presenti sul device di utilizzo).

Difficoltà: Facile

Motivazione: il linguaggio Javascript, in quanto linguaggio Object Oriented, permette l'accesso a istanze di oggetti diverse tramite stessi comandi e sintassi. Per questo motivo è bastato creare due oggetti per le due lingue, ottenere la lingua corrente tramite una API di Apache Cordova e istanziare l'oggetto desiderato come "lingua corrente" .

5.1.2 Connessione a Server

Descrizione: L'oggetto di comunicazione col server si occupa di gestire tutte le chiamate ai servizi. Tipicamente riceve le richieste dal controllore, contatta il server e al termine restituisce la risposta del server al chiamante.

Difficoltà: Difficile

Motivazione: Gestire le risposte del server su JavaScript è più complicato quando il sistema deve ricevere risposte a richieste multiple. In fase di ricezione viene perso il riferimento all'oggetto corrente ed è quindi difficile stabilire a quale richiesta associare la risposta ricevuta dal server. La soluzione è stata l'implementazione di un sistema di code che gestisca in maniera sequenziale le connessioni in modo da non aver il problema di associazione di richieste e risposte. Il contenuto delle risposte è invece facilmente gestibile in quanto si tratta di oggetti JSON che JavaScript gestisce nativamente.

5.1.3 Integrazione Mappa e Marker

Descrizione: Il cuore del gioco è certamente la Mappa e la visualizzazione dei vari marker per ogni venue presente.

Difficoltà: Facile

Motivazione: L'integrazione a livello JavaScript con Google Maps è ormai un attività molto comune per qualsiasi applicazione web. Google fornisce tutte le API (oggi alla versione 3) per poter caricare la mappa, impostare il centro su determinate coordinate, ad un certo grado di Zoom e fornendo metodi per la creazione di marker personalizzati (quindi con icone personalizzate).

5.1.4 Rilevamento posizione GPS

Descrizione: Le mappa precedentemente descritta deve agire sulla posizione corrente ed è quindi necessario calcolare la posizione.

Difficoltà: Molto Facile

Motivazione: E' nell'accesso all'hardware di sistema che Cordova mostra tutte le sue più grandi qualità. Ottenere tutte e le coordinate tramite il sensore GPS (o in sua assenza tramite informazioni della connessione) è di una facilità sconvolgente. E' necessario semplicemente richiamare l'API e fornire un metodo di callback che il framework richiamerà una volta ottenuta l'informazione. Esempi concreti e documentazioni sono interamente disponibili online sul sito di Apache Cordova.

5.1.5 Integrazione con Facebook

Descrizione: In fase di login l'utente si connette al server sfruttando i dati del proprio profilo facebook. A questo proposito è necessario legare la propria applicazione al social network tramite l'uso di plugin e della Facebook SDK.

Difficoltà: Molto difficile

Motivazione: L'integrazione è risultata piuttosto complicata (e in parte irrisolta). Questa operazione va gestita a livello di piattaforma di deploy (Android, iOS, ecc) in quanto per ognuna di esse, l'integrazione avviene in maniera diversa, o almeno in parte. Facebook mette a disposizione una libreria Javascript di connessione che dovrebbe garantire una fruizione omogenea del servizio in maniera indipendente dalla piattaforma. Malauguratamente utilizzando Apache Cordova il meccanismo non è così semplice, anzi è necessario, per ciascuna piattaforma, eseguire alcune procedure specifiche di configurazione e integrazione. Passo necessario è la creazione di una Facebook App sul sito dedicato agli sviluppatori Facebook, ottenere un App Id e configurare il proprio ambiente di sviluppo per consentire le chiamate ai servizi offerti dal Social Network. Ulteriore difficoltà è data dal differente approccio a Facebook e ai Social Network in generale di ogni singola versione si sistema operativo installata (iOS 5 ha una gestione completamente diversa da iOS6 e la procedura da effettuare cambia sensibilmente).

5.1.6 Visualizzazione classifica e notifiche

Descrizione: Classifica e notifiche si possono considerare insieme come esempio di interazione con oggetti grafici semplici. Il contenuto delle relative pagine è puramente dinamico e viene fornito dal server. Ogni risultato fornito corrisponde a un blocco HTML da mostrare, di conseguenza se esso è vuoto la pagina non avrà alcun blocco e sarà di conseguenza completamente vuota.

Difficoltà: Facile

Motivazione: Una volta ottenute le informazioni tramite l'oggetto server basterà scorrere l'array di oggetti e, per ognuno di essi, creare il codice HTML di visualizzazione

richiamando i campi dall'oggetto ricevuto. In questa fase si prevedono le classi da associare agli elementi HTML da gestire successivamente tramite CSS.

5.1.7 Gestione del Back-Button

Descrizione: Il pulsante indietro è fondamentale in ogni fase di gioco perché permette di tornare al menu principale. La semantica del pulsante è particolare e si modifica in base a ogni contesto di gioco. Talvolta il pulsante è disabilitato, a volte semplicemente porta alla pagina precedente e in altre occasioni ha l'effetto di un back semantico che non coincide con la visualizzazione della pagina precedentemente mostrata.

Difficoltà: Difficile

Motivazione: Si tratta sicuramente di una gestione di difficoltà alta ma non dovuto a caratteristiche dell'ambiente di sviluppo ma legate alla logica di gioco. La successiva corretta implementazione ha comunque richiesto diversi test prima di ottenere un algoritmo stabile e efficace.

5.1.8 Elementi HTML

Descrizione: Tutta la parte di visualizzazione e di grafica è gestita a livello di elementi HTML sui quali vengono applicate opportune regole di stile CSS. Il passaggio da oggetti logici a elementi html prevede una gestione del processo di visualizzazione.

Difficoltà: Media

Motivazione: A livello di codice JavaScript per gestire il passaggio sopradescritto è stato previsto un oggetto dedicato con dei metodi richiamati dal controllore generale. Tale oggetto (l'oggetto View sopradescritto) si occuperà di creare tutti i vari tipi di elementi: pulsanti azione, pulsanti switch-to, elementi html per popolare la pagina classifica, notifiche, i miei luoghi e tutti gli altri elementi che vengono mostrati in fase di gioco.

5.1.9 Grafica

Descrizione: La componente di grafica consiste nell'applicare lo stesso layout e approccio visuale al nuovo progetto Cordova in modo che sia conforme all'applicazione originale.

Difficoltà: Difficile

Motivazione: Nonostante l'uso di HTML e CSS sia di larga diffusione e abbiano una notevole capacità applicativa, risulta molto complesso la loro applicazione a un progetto Apache Cordova. La difficoltà consiste nel non poter lavorare direttamente alle regole vedendo un applicazione immediata a causa dell'assenza di uno strumento che permetta tale tipo di debugging (es. Firebug). La difficoltà e i tempi subiscono quindi un importante aumento, le modifiche vengono fatte alla cieca per poi poter vedere l'effettiva applicazione nel giro di qualche minuto (tempo di ricompilare e eseguire l'applicazione sull'emulatore).

5.1.10 Ruota

Descrizione: La ruota è un componente essenziale del gioco in quanto interviene ogniqualvolta si visita un luogo già occupato. Essa deve entrare in funzione a pieno ritmo in corrispondenza di un "drag" sufficientemente veloce lungo la sua traiettoria circolare per poi successivamente fermarsi in maniera pilotata su un determinato risultato calcolato a priori.

Difficoltà: Molto difficile

Motivazione: Grazie a CSS 3 è possibile effettuare le rotazioni degli oggetti HTML, le problematiche nascono nel coordinare la rotazione in concomitanza con gli eventi touch, regolare le variazioni di velocità e infine dover pilotare l'esito in corrispondenza col risultato calcolato precedentemente. Durante la fase di rotazione, quando si scatena l'evento touch-up (quando il dito si stacca dal display) viene calcolata la velocità del movimento e, se oltre una certa soglia, comincerà a girare in maniera sempre più veloce. Dopo aver raggiunto una velocità massima la ruota comincerà a decelerare fino al fermarsi sul risultato

desiderato. L'assenza di debugger in questa fase è di importanza elevata e fonte di maggiori problemi e rallentamento.

5.1.11 Oggetti visita semplice

Descrizione: Gli oggetti di visita semplice sono gli oggetti di tipo PAY, TAKE, SELL, MORTGAGE, REDEEM e BUY e servono a comunicare al server le azioni eseguite dal giocatori.

Difficoltà: Facile

Motivazione: L'invio delle visite semplici consiste nella creazione di oggetti JSON su un modello predefinito contenente il nome del luogo, la sua categoria, il prezzo, la data e altre informazioni di questo tipo. Una volta preso un oggetto di modello facilmente vengono realizzate le varianti dei diversi tipi di visite.

5.1.12 Oggetti visita complessi

Descrizione: Gli oggetti ADVERTISE, QUIZ e JUDGE contengono molte più informazioni in quanto prevedono lo scambio di domande, risposte e informazioni relative al tipo di risposte da proporre (testo libero, scelte a risposta singola, multipla o foto).

Difficoltà: Difficile

Motivazione: Creare gli oggetti opportuni è sicuramente un compito complicato, bisogna prevedere tutti i formati di domanda/risposta, acquisire le risposte e adattarle in maniera giusta all'oggetto JSON corretto. Un apposito oggetto Question si occupa di aggregare tutte le risposte fornite nelle diverse modalità, in tutti i casi previsti per ogni step di raccolta dati.

5.1.13 Acquisizione e caricamento foto su Server

Descrizione: Al termine di un ADVETISE il gioco invia al server la foto scattata durante l'ultimo step.

Difficoltà: Facile

Motivazione: Apache Cordova tra le sue API offre la possibilità di scattare le foto e, in questo caso, di archiviarle fornendo l'URI. Tramite un'altra chiamata sempre offerta dal framework sarà possibile procedere all'upload del file sul server con l'annessione dei relativi parametri (venue di riferimento e id giocatore).

5.1.14 Porting su Android

Descrizione: Una volta concluso il progetto è necessario eseguire un operazione di porting sulla piattaforma specifica, in questo caso la piattaforma Android.

Difficoltà: Molto facile

Motivazione: La guida disponibile sul sito web del framework è molto chiara a proposito, in pochi passi sarà possibile creare un progetto Eclipse includendo le librerie di Cordova e creando nel file manifest i permessi necessari per accedere alle funzionalità hardware del dispositivo Android.

5.1.15 Porting su iOS

Descrizione: Come nel caso appena descritto la stessa operazione va fatta su piattaforma iOS.

Difficoltà: Media

Motivazione: Anche in questo caso è presente una guida sul sito che guida alla creazione di un progetto X-Code con annessa libreria di Apache Cordova. Seguendo più o meno gli step del caso Android il progetto viene creato ma talvolta il compilatore di X-Code può dare problemi e ulteriori azioni correttive (reperibili online) diventano necessarie alla corretta compilazione.

5.2 Pregi e difetti emersi

Al termine del percorso di sviluppo è giusto fare alcune considerazioni di carattere generale che sono emerse durante questa fase. L'utilizzo del framework in un progetto completo dà la possibilità di valutare tutti i pregi e difetti che sarebbero altrimenti difficilmente riassumibili a priori valutando le caratteristiche dello stesso.

Questi i vantaggi chiave:

- Con certezza c'è da confermare che la promessa di sviluppare attraverso un **linguaggio di programmazione universale,** viene mantenuto appieno e messo in atto con molta facilità. Si tratta del principio portante dell'intero framework, ciò che spinge lo sviluppatore a scegliere questo tipo di soluzione. Il framework risponde perfettamente in relazione a quanto dichiarato è quindi funzionale all'obiettivo perseguito.
- L'unicità e ciò che mette in rilievo l'importanza del framework è la possibilità di estendere l'universalità di linguaggio all'universalità di accesso all'**hardware nativo**. Questa caratteristica lo rende unico ed è in questo che Apache Cordova si differenza da una normale web application.
- L'utilizzo di standard web di ultima generazione e di grande diffusione come **HTML 5**, **CSS 3** e **Javascript** si rivela una scelta ottimale e vincente per unificare in un unico linguaggio tutte le diverse piattaforme supportate da Apache Cordova. Si tratta di standard in continua crescita e giunti a un livello di potenza elevatissimo, frutto di anni di esperienze web raccolte dai webmaster e utenti di tutto il mondo, dalle esigenze che si sono presentate nel tempo e accresciute dalla corrispettiva evoluzione dei browser moderni.
- Diretta conseguenza del punto precedente è la possibilità di sfruttare tutte le potenzialità di JavaScript e quindi di usufruire di tutte librerie disponibile online per tutti i gusti. In tal senso le più rilevanti in ambito mobile sono certamente **jQuery** e **jQuery Mobile**. Tali librerie, già viste nel dettaglio, garantiscono un tutt'altro spessore rispetto al tradizione approccio Javascript. Di queste l'utilizzo di jQuery Mobile si sposa alla perfezione con le caratteristiche del framework: mentre il framework garantisce una semplice interazione con l'hardware, jQuery Mobile offre la possibilità di gestire la User Interface captando eventi di tipo touch, tap, slice e un gran numero di gestures entrate ormai negli automatismi degli utilizzatori finali. Il framework fornisce inoltre librerie grafiche per riprodurre i pulsanti, effetti e animazioni tipiche dell'ambiente mobile.

- La scelta di Javascript si manifesta essere vincente anche dato l'utilizzo frequentissimo di **JSON** come standard di comunicazione con i Web Service. In molte applicazioni e webservice viene preferito all'uso di XML anche grazie all'uso massivo di AJAX (Asynchronous JavaScript and XML) nei siti web. JSON, come già spiegato, viene gestito nativamente da Javascript, il risultato è quindi un integrazione coi Web Service molto agevolata.

Come spesso accade bisogna però fare i conti con il lato rovescio della medaglia.

L'esperienza diretta evidenzia quanto, anche in un applicazione che non presenta un livello di logica e funzionalità che prevedano elevata complessità algoritmica, l'assenza di un debugger vero e proprio rappresenti un ostacolo con cui lo sviluppatore non è abituato a convivere. Tutte le piattaforme di sviluppo, infatti, sono corredate di opportuno debugger e offrono quindi la possibilità di scorrere il proprio codice step by step e, valutando quindi i valori delle variabili, degli oggetti e comportamenti anomali, "fixare" gli eventuali errori. Il debugger su cui si appoggia è qualcosa ma ancora insufficiente.

Altro difetto rispetto alle applicazioni native è l'impossibilità di accedere al 100% delle capacità hardware per quanto riguarda processore e RAM. L'applicazione quanto a utilizzo di risorse non è di grandi pretese, ma ugualmente si ha spesso l'impressione della presenza di ritardi di risposta agli input degli utenti. Anche lo scroll degli elementi sembra non essere fluido come le applicazioni native. In generale però tali limiti sono sostanzialmente oscurati in relazione alle esigenze di giocabilità non particolarmente restrittive e quindi più che soddisfatte.

Sebbene il proposito del framework sia quello di mettere in condizione lo sviluppatore di lavorare solo basandosi sulla conoscenza di Javascript, la corretta implementazione di alcune operazioni implica comunque delle modifiche o integrazioni da effettuare in linguaggio nativo. L'esempio concreto è quello del plugin facebook l'integrazione del quale comporta non pochi interventi da effettuare su piattaforma nativa (specialmente nel caso di iOS 6).

5.3 Apache Cordova vs Sviluppo nativo

Per delle valutazioni opportune gli sviluppatori di entrambe le versioni sono stati chiamati a un confronto. Seguendo il metodo **Delphi** a essi è stato chiesto di rispondere ad alcune domande comuni in modo da poter poi incrociare le informazioni e poter trarre delle conclusioni riconducendo insieme le eterogenee risposte ottenute.

Il metodo Delphi è un metodo d'indagine iterativo, particolarmente utilizzato nel campo del business, che si svolge attraverso più fasi di espressione e valutazione delle opinioni di un gruppo di esperti o attori sociali ed ha l'obiettivo di far convergere l'opinione più completa e condivisa in un'unica "espressione".

Nella fattispecie, dopo aver posto le opportune domande allo sviluppatore della versione Apache Cordova circa l'esito dello sviluppo, problemi vantaggi e svantaggi del framework (già analizzati nel precedente capitolo), analoghe domande sono state poste a chi invece ha sviluppato il gioco nella versione nativa Android.

5.3.1 Tratti comuni

Per molti tratti vi sono aspetti che si manifestano in entrambe le versioni:

- In entrambi i casi è facile gestire lo scatto di una foto, da una parte perché il framework fornisce un API stabile, dall'altra perché Android fornisce un opportuna gestione delle Activity che garantisce il buon esito dell'operazione.
- La costruzione di pagine prettamente statiche è risultato semplice, su Cordova per via dell'approccio HTML + CSS, su Android tramite la gestione delle interfacce grafiche con un file XML (in linea di principio non c'è differenza tra i due approcci).
- Javascript come Java si prestano alla gestione di oggetti scambiati in formato JSON. Nel primo caso il parsing non è necessario, in quanto il tipo di dato è già gestito nativamente, nel secondo viene gestito tramite un API **GSON** che permette la conversione da oggetti JSON a oggetti Java.
- L'interfaccia multilingua è di facile implementazione purchè considerata fin dall'inizio. Javascript non fornisce alcun tipo di predisposizione a tal fine ma può essere opportunamente risolta in fase di programmazione. Android invece prevede una gestione dedicata di tutte le stringhe a valore costante che agevola il processo.

- La ruota è risultata difficile da entrambi i lati. Fatto salvo per la gestione dell'animazione (di equa difficoltà concettuale indipendentemente dalla piattaforma) la difficoltà consiste nella gestione degli eventi touch dell'utente, nel distinguere uno scroll lento da uno veloce e poter essere in grado di validare un "tocco" solo quando all'interno della superficie della ruota.
- La gestione del flusso di gioco all'interno dei sondaggi, in cui sono presenti pagine a step, è risultata complessa. La gestione nei due sistemi viene fatta in maniera diversa. In Apache Cordova (quindi in Javascript) la pagina HTML è la stessa che di step in step viene aggiornata col contenuto corrente della domanda. In Java, invece, tale flusso viene gestito tramite la gestione di Activities che vanno a sovrapporsi per poi essere chiuse tutte insieme al termine del sondaggio.
- Integrazione con Facebook risulta essere difficile nei due ambienti di sviluppo. Il motivo è perlopiù legato alla mancanza di stabilità e consistenza dei plugin, librerie e SDK presenti.

5.3.2 Differenze chiave

Due sono invece le differenze chiave tra le piattaforme emerse dalle risposte fornite dagli sviluppatori:

- Come già ampiamente previsto il **debugging**, o meglio la sua assenza, rappresenta su tutti uno degli ostacoli più rilevanti di Apache Cordova. Al termine dello sviluppo delle due versioni da un lato, quello nativo, esso si pone come vantaggio chiave e utilissimo strumento di soluzione ai problemi, dall'altro, come già espresso a più riprese, si pone come la più grande fonte di rallentamento in fase di sviluppo e di testing.
- A dispetto di quanto si potesse pensare l'integrazione con la **mappa** rappresenta un altro punto di discordanza. Sorprende il fatto che tale integrazione sia più semplice su Cordova che su Android. La stabilità e l'uso che si è fatto negli anni delle mappe di Google nei siti web e, più in generale, nelle pagine HTML si rispecchia in una integrazione col progetto Cordova immediata. La stessa integrazione su piattaforma Android avviene invece in maniera più complicata e macchinosa.

5.3.3 Migrazione al cross-platform?

“Apache Cordova è a tutti gli effetti pronto per supportare un possibile passaggio abbandonando così lo sviluppo nativo?”

“Cosa può incentivare o disincentivare uno sviluppatore a percorrere una strada piuttosto che l’altra?”

“Quali i più grandi timori in un passaggio di questo tipo?”

Tali quesiti sono fonte di molto dissenso. Uno sviluppatore Android di esperienza può legittimamente manifestare le proprie preoccupazioni su un framework del genere rispetto alla stabilità e capacità fornite da una piattaforma nativa.

L’aspettativa è che l’utilizzo di un tool del genere porti a un risparmio notevole di tempo di fronte a una già acquisita conoscenza di HTML, CSS, Javascript in accoppiamento all’uso di librerie come jQuery Mobile.

La manutenzione di un codice unico porterebbe non pochi vantaggi di fronte all’esigenza di porre modifiche, ottimizzazioni e miglioramenti. Di contro la necessità di avere una versione del software per ogni piattaforma ne rallenterebbe i tempi.

La possibilità di interagire con le funzionalità hardware del dispositivo (come GPS, fotocamera, back button, etc.) in maniera trasparente e stabile su tutte le piattaforme è vantaggio universalmente riconosciuto e ben gradito a chi è abituato a un approccio nativo.

La possibilità di incorrere in applicazioni lente, o non fluide, e la difficoltà di ottimizzazione su piattaforme diverse (es. esigenza di usare un pulsante back in-app nel caso di iPhone in quanto non previsto) frenano non poco gli sviluppatori in vista di una possibile migrazione. L’user experience potrebbe essere a rischio qualora il framework proponesse risultati diversi sulle differenti piattaforme, cosa sicuramente non gradita e non tollerabile.

Infine, come presumibile, lo scoglio più arduo da superare rimane il dover sopperire alla mancanza di un debugger, il miglior alleato di ogni sviluppatore. La ricerca dei bug, l’ottimizzazione del codice e i test sono fasi essenziali durante lo sviluppo di un applicativo. E’

necessario quindi che tali operazioni possano essere svolte in tempi proporzionati e con strumenti adeguati. Il rimedio offerto dal framework risulta troppo parziale e approssimativo specialmente per chi è abituato all'uso dei debugger offerti da Eclipse, Firebug o X-Code.

5.3.4 Resoconto finale

Di seguito viene mostrata la tabella comparativa dei tempi (e di conseguenza i costi) di sviluppo nelle due versioni del gioco.

Attività	Android (ore)	Cordova
Gestione interazioni con il server	24	30
Game Manager & Visit Manager	30	30
Advertise	24	20
Quiz	16	10
Mappa+GPS	20	2
Ruota	12	30
Buy/take/skip/sell/mortage	16	8
Classifica	4	3
Tutorial	6	5
Poster	4	2
Judge	8	4
Test & Debugging	40	30
Grafica	46	40
Interazioni con Facebook	10	10
Notifiche	6	2
Crediti	1	1
Porting su iOS	-	8
Porting su Android	-	2
Totale	267	237

Tabella 1 - ***Scheda comparativa: Sviluppo nativo vs Apache Cordova***

6 - Conclusioni

Dopo tutte le considerazioni fatte è necessario valutare quanto sia effettivamente proponibile, in ambito industriale, un applicazione sviluppata su Apache Cordova paragonandola alle corrispettive versioni native.

Dall'esperienza maturata durante le fasi di sviluppo è possibile riscontrare quanto l'utilizzo di Apache Cordova risulti essere la soluzione più comoda in termini di tempi e costi. Dal confronto del capitolo precedente emerge che l'esito, dal punto di vista dei tempi di sviluppo, è una sostanziale equivalenza in ambedue le versioni. La durata di sviluppo totale risulta essere dello stesso ordine di grandezza e di conseguenza da considerarsi sullo stesso piano. Risulta chiaro come questa apparente parità si trasformi in una caratteristica vincente del framework rispetto alla versione nativa, considerando l'applicabilità della soluzione a piattaforme eterogenee. La proporzione è forte: 7 (numero di piattaforme supportate) a 1. Partendo da una stima di tempi/costi di sviluppo equa nelle due versioni, si evince la convenienza e l'utilità offerta. E' opportuno però prendere in considerazione tutti gli aspetti coinvolti: è impensabile generalizzare la totalità delle applicazioni nel tentativo di dedurre conclusioni univoche. Ogni applicazione, in base alla propria funzione, i propri scopi e caratteristiche può essere sottoposta a diverse esigenze e requisiti in termini di performance, consistenza, e impatto hardware. L'applicazione sviluppata, Urbanopoly, rispecchia un profilo di utilizzo di risorse medio-basso e risponde quindi perfettamente al profilo di applicazione a cui Apache Cordova meglio si adatta. E' su questo tipo di applicazioni che il software dà il meglio di se e si pone come soluzione valida per un deploy multipiattaforma.

Al termine dello sviluppo del progetto con Apache Cordova, nonostante tutti i pregi e le funzionalità di cui gode, ritengo che di fronte a progetti di sviluppo di consistenza ben superiori che richiedano elevate capacità di calcolo, che allochino un'ingente quantità di dati in memoria o che possano necessitare di elaborazioni grafiche complesse (2D/3D), difficilmente il framework potrà soddisfare le esigenze necessarie. I tempi di sviluppo potrebbero crescere in maniera più consistente rispetto alle versioni native e, come già spiegato, si andrebbe incontro a problemi di performance e di capacità applicativa. La scelta nativa diverrebbe quindi obbligatoria.

La possibilità offerta da Apache Cordova di sviluppare applicazioni con l'uso di semplice HTML, CSS e Javascript lo porta ad essere accessibile anche da sviluppatori meno esperti o con minor familiarità con la programmazione nativa e di poter arrivare a tutti i dispositivi in breve tempo in concomitanza con un importante risparmio. In questa ottica è da tenere in considerazione in quanto in molti casi si può avere la necessità di avere delle piccole applicazioni basiche (o anche più strutturate) in poco tempo o con limiti iniziali di budget.

Di fronte a questo genere di richieste Apache Cordova è in grado di soddisfare i requisiti offrendo la possibilità di proporre soluzioni che, pur non garantendo il massimo dal punto di vista di qualità e performance, possono essere adeguate alle esigenze sottoposte e consentano di ottenere applicazioni multipiattaforma in tempi brevi.

Bibliografia

[1] **Apache Cordova** - *About Apache Cordova,* http://incubator.apache.org/cordova/

[2] **PhoneGap Plugin System,** *Fylhan* - http://30minparjour.la-bnbox.fr/2012/phonegap-plugin-system

[3] **PhoneGap debugging** - http://debug.phonegap.com/

[4] **Getting Started with iOS -** http://docs.phonegap.com/en/2.2.0/guide_getting-started_ios_index.md.html#Getting%20Started%20with%20iOS

[5] **Getting Started with Android** - http://docs.phonegap.com/en/2.2.0/guide_getting-started_android_index.md.html#Getting%20Started%20with%20Android

[6] **PhoneGap API Reference** - http://docs.phonegap.com/en/2.2.0/

[7] **HTML - Descrizione Generale**,*Wikipedia* - http://it.wikipedia.org/wiki/HTML

[8] **Introduzione ai fogli di stile, Cascading Style Sheet** - http://users.dimi.uniud.it/~giorgio.brajnik/dida/docs/rapporto_CSS/Intro.html

[9] **JavaScript,** *Wikipedia* **-** http://it.wikipedia.org/wiki/JavaScript

[10] **AJAX,** *Wikipedia* - http://it.wikipedia.org/wiki/AJAX

[11] **jQuery,** *Wikipedia* - http://it.wikipedia.org/wiki/JQuery

[12] **Introduzione a jQuery Mobile,** *Html.it* - http://www.html.it/pag/19526/introduzione-a-jquery-mobile/

[13] **Apache Tomcat,** *Wikipedia* - http://it.wikipedia.org/wiki/Apache_Tomcat

[14] **PostgreSQL,** *Wikipedia* - http://it.wikipedia.org/wiki/PostgreSQL

[15] **X-Code,** *Wikipedia* - http://it.wikipedia.org/wiki/Xcode

[16] **Eclipse**, *Wikipedia* - http://it.wikipedia.org/wiki/Eclipse_(informatica)

[17] **Guida a Firebug**, *Attilio Fiumanò* - http://www.blographik.it/2009/05/27/guida-firebug/

[18] **Urbanopoly - a Social andLocation-based Game with a Purpose to Crowdsource your Urban Data**, *Irene Celino, Dario Cerizza, Simone Contessa,Marta Corubolo, Daniele Dell'Aglio, Emanuele Della Valle e Stefano Fumeo.*

[19] **Urbanopoly: Collection and Quality Assessment of Geo-spatial Linked Data via a Human Computation Game,** *Irene Celino, Dario Cerizza, Simone Contessa, Marta Corubolo, Daniele Dell'Aglio, Emanuele Della Valle, Stefano Fumeo, e Federico Piccinini.*

[20] **Sviluppatori di Facebook**, *Facebook Developers* - https://developers.facebook.com/

[21] **Apache Cordova Facebook Connect Plugin**, *Github* - https://github.com/davejohnson/phonegap-plugin-facebook-connect/