Comparison Of Apple's Ios 5 And Android For Mobile Applications Development: A Developer's Perspective

Estela Pochintesta
University of Mississippi

Comparison of Apple's iOS 5 and Android for Mobile Applications Development: A Developer's Perspective.

A Thesis

Presented for the

Master of Science

Degree

The University of Mississippi

Estela Pochintesta

July 2012

ABSTRACT

The aim of this research is to study the Apple iOS and Android mobile frameworks from a developers' perspective. The first chapter introduces a historical overview of operating systems for mobile devices as well the motivation and objectives of the research. The second chapter introduces basic principles of the iOS and Android mobile frameworks in order to evaluate the platforms. The third chapter evaluates and compares the main features of both frameworks. The fourth chapter discusses the implementation of several applications on the iOS and Android mobile frameworks, which are used to demonstrate the advantages and disadvantages of each platform. The final chapter contains a summary, which gives an outlook of the future of the platforms.

DEDICATION

This work is dedicated to my family and the love of my life Bichito, without whose tireless encouragement I would have given up long ago.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor Dr. Dawn Wilkins, who originally suggested and guided my research. For her encouragement, enthusiasm, and knowledge. Besides my advisor, I would like to thank the rest of my thesis committee: Dr. Yixin Chen and Dr. H Conrad Cunningham, for their encouragement, guidance and suggestions. In addition, I would like to thank the Writing Center services for the help in editing of the final document

LIST OF ABBREVIATIONS

ADT	Android Development Tools
IDE	Integrated Development Environment
SDK	Software Development Kit
IDC	Information Development Consultants
MVC	Model View Controller
NFC	Near Field Communications
DDMS	Dalvik Debugging Monitor Service

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

This thesis is the outcome of research performed while investigating the mobile technology industry, with a specific focus on the leading mobile platforms at this time. The primary focus is from the application developer's perspective.

Since 1992, a revolutionary type of device was brought into the industry under the name of smartphone, first developed by IBM. Simon, the first smartphone was a device with primarily phone functions and a fixed set of applications, which included a calendar, address book, calculator, and several applications. The term smartphone refers to a mobile phone that offers advanced computational power and networking capabilities. Formally it is defined as: "a mobile electronic device which runs an advanced operating system that is open to installing new applications, is always connected to the Internet, and which provides very diverse functionality to the consumer" (Cromar, 2011).

With the evolution of smartphones over the past decade, along with the evolution of communication and networking technologies, the mobility factor of smartphones has brought about a new style of computing, defined as mobile computing. Mobile computing is a broad term used to describe technologies that allow people to have networking capabilities using portable devices.

The mobile technology is generally defined as a three-part technology consisting of: mobile hardware, data, and software (Reen, 2000). Mobile computing opened an industry for hardware manufacturers, platform providers, and content providers. The hardware manufacturers are the companies that create and design mobile devices, mainly tablets and cellphones. Platform providers are companies that provide the mobile operating system and the development tools that allow third party developers to create applications to be used on the devices. Finally, third party developers can be companies or freelance programmers who decide to publish applications in the market for users to access.

Several companies act as platform providers. The most popular providers today are: Symbian, Android, iOS, RIM and Mango for Windows Phone. The mentioned platform providers develop the mobile operating systems and provide the required tools that allow third party developers to create applications to be run on their operating systems. Each platform offers its own set of developer tools and features. However, all of them face several similar challenges of mobile computing (Chlamtac 1998), (Holzer 2009):

- Display device characterized by a small size screen
- Input device to incorporate a speech based interface and touch screen keyborads
- Power source management
- Non- continuity connection and latency problems

The mobile software component of the equation is the most relevant for this research, given the incredible growth of the development of third party applications for mobile platforms. Jahns *et al* (Jahns, 2010) predict that the smartphone application market will reach $15.65 Billion in 2013. When a developer plans to develop mobile applications, she needs to choose a platform from the available platforms. In general, the developer will face a tradeoff between the features and tools that each mobile platform offers and the effort and price needed in order to develop the application. This explosion in the smartphone application market made third party application development a new, very promising business. According to Sharma (Sharma 2010), the mobile application industry is estimated to reach $17.5 million by the end of 2012. Moreover, mobile application downloads are expected to grow at a rate of 92% per year, since from 2009 until 2012 it has reached almost 50 billion.

Developers have a broad set of different platforms and techniques that they can choose from in order to develop mobile applications. Each platform provides it own solution that presents some tradeoffs, translated into advantages and disadvantages, which are related to the special features and the approach that each platform takes.

The motivation of this thesis is mainly driven by the popularity of mobile application development around the software industry and the lack of a comprehensive documentation that will give developers the insight of the main strengths and weakness of each platform. One of this main goals of the thesis is to study and evaluate the main mobile platforms from a developer's perspective in order to help developers choose a suitable platform for a specific combination of requirements. Therefore, this thesis aims to: (1) present the most popular alternatives in the industry to develop applications for smartphones, (2) describe each platform from a technical perspective with focus on development environment, language, techniques, distribution and special features, (3) compare and contrast each platform theoretically and in practice.

The outcome will be a set of suggestions that aim to help the developer to choose one platform over the other, given a certain fixed set of parameters. This mobile platform comparison is limited to the iPhone and Android mobile platforms; the mentioned platforms cover a large portion of the smartphone device share. Figure 1 shows the applications downloaded per day, which gives an insight of the preferable mobile platforms, on which to develop. Clearly, iPhone and Android are the most popular now.

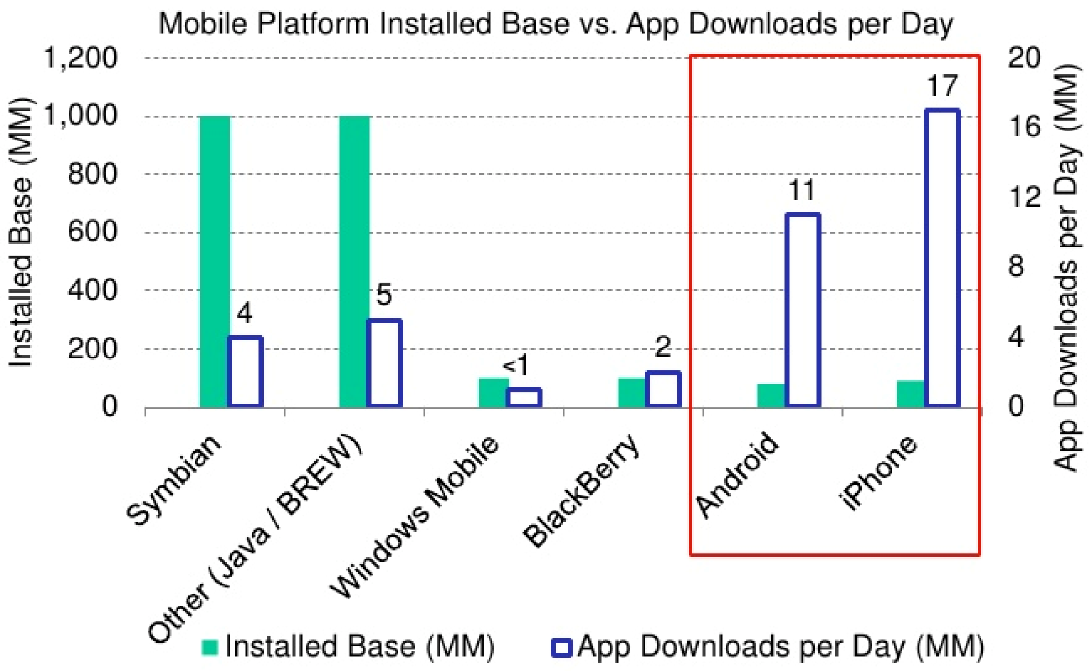


Figure 1. Application downloads

This research is based on a three-step methodology where in the first step an exhaustive literature review of the frameworks was undertaken in order to write a survey describing the theory behind the frameworks. Next, a fact-based comparison will be outlined in order to discover the advantages and disadvantages of each mobile platform. The outcome of the comparison will lead to the development of application features that will support conclusions regarding the advantages and disadvantages of each mobile platform from the perspective of a developer.

The main contribution of this research is to expose, in a theoretical and practical manner, a comprehensive comparison between the Android and iOS mobile platforms. The goal is to help developers to choose which platform is best for development. The research will be limited to the study and comparison of iOS and Android operating systems for mobile devices. On one hand, the motivation behind investigating the iOS operating system is because of the popularity of Apple devices in the market and the quantity of applications that the Apple Store contains for these devices. On the other hand, the motivation behind studying the Android operating system is the fact that it is a popular open platform for mobile devices. Specifically, this first chapter introduces the reader to the details of the research context, objective, motivation, limitations, and methodology.

The remainder of this thesis is organized as follows: Chapter 2 provides a technical description of the Android mobile platform. Chapter 3 provides a technical description of the Apple mobile platform. Chapter 4 outlines a comparison of the two frameworks from a developer’s perspective in a comprehensive, theoretical, and technical manner. Chapter 5 presents the outcome of the research, including conclusion and suggestions in order to identity the most suitable environment or conditions to identify under which platform is superior to the other in various application domains.

CHAPTER II

THE ANDROID MOBILE PLATFORM

Chapter 2, introduces and overview of the Android mobile platform, which was created by Android Inc. It is an open source platform bought and managed by Google since 2005 and part of the Open Handset Alliance. Therefore, each manufacturing company can choose which set of built-in applications and set of hardware features want to include in the Android compatible devices (phone, tablet). Android is built-up on a modified version of Linux kernel and consist of a software stack for mobile devices and a software platform foundation for development of third party applications. It is divided in four layers, and consists of an operating system, middleware, and applications. Android offers a Software Development Kit (SDK) and Android Developer Tools (ADT) to third party developers for application development. The ADT offers tools and instruments for prefer development IDE (Interactive Development Environment) Eclipse.

The most recent version of the SDK is 4.0. Previous versions, including 2.X did not include tablets as permitted devices until the inclusion of the 3.X versions. Chapter 2 introduces the mobile platform by describing: android development essentials, hardware variety, SDK, platform fragmentation, development tools, application development architecture, features and distribution and monetization of applications.

II.1 ANDROID DEVELOPMENT ESSENCIALS

Android is open source and therefore there is no fixed hardware and software configurations. In addition, the framework enables reuse and replacement of the application components. The platform offers a rich development environment including a SDK and ADT, which contain tools and instruments to debug, test and measure the performance of applications. It includes a plug-in for Eclipse, the suggested IDE for the development. In addition, the SDK supports the following features: storage, connectivity, messaging, web browser, media support, hardware support, multi-touch, multi-tasking, flash, and tethering.

The framework is based on object-oriented techniques, where each application consists of a set of components modeled by Java classes and that might have an interface associated with them which is a XML file. The basic organization of an Android application is that components call each other, to achieve the application's functionality. There are four types of Android application components (activities, services, content providers, and broadcast receivers), which are activated by asynchronous messages called intents. The openness of the platform promotes reusability of its components. Each component within the application can publish its functionality, and any other application can make use of those capabilities. Each Android application runs in its own sandbox and Dalvik Virtual Machine in order to be isolated from the rest of the applications, providing a secure environment for the operating system. However, applications are allowed to communicate and share resources by explicitly declared permissions.

II.II ARCHITECTURE

The software stack architecture is built-up on four layers, as shown in Figure 2 within the layers; the Linux kernel is the layer closest to the hardware while the application layer is the one closest to the end user.

The Linux Kernel layer contains the drivers for the hardware components that the Android device offers, memory management, process management, security model, and networking. Afterwards, the following layer includes two groups the Libraries and the runtime. The libraries group is written in Java and contains all the native APIs. The runtime consist in two components. The core libraries, enables developers to write applications in Java language programming, given that the Kernel does not process the Java language. The Dalvik Virtual Machine, which enables Android applications to run in their own process with their own instance of the Virtual Machine and translates the Java code in order to be processed by the kernel. Finally, the last layer, Application framework, contains the built-in applications as well as the developer's applications and a toolkit for the development and testing of applications and (Lee 2011) and (Android 2012) .

The toolkit is defined in terms of (Android 2012):

- Activity manager, in charge of life cycle of applications
- Package manager, maintains information of the available applications on the device
- Windows manager
- Telephony Manager, contains the APIs that allow third party developers to write phone and SMS applications.

- Content provider, allows applications to expose their data to other applications
- Resource Manager, Manages the layout and resources that the application uses as strings, images, and audio.
- View System, contains the components that are need it to create the user interface
- Location Manager, APIs to manage location-aware and GPS services.
- Notification Manager, alerts the user about events

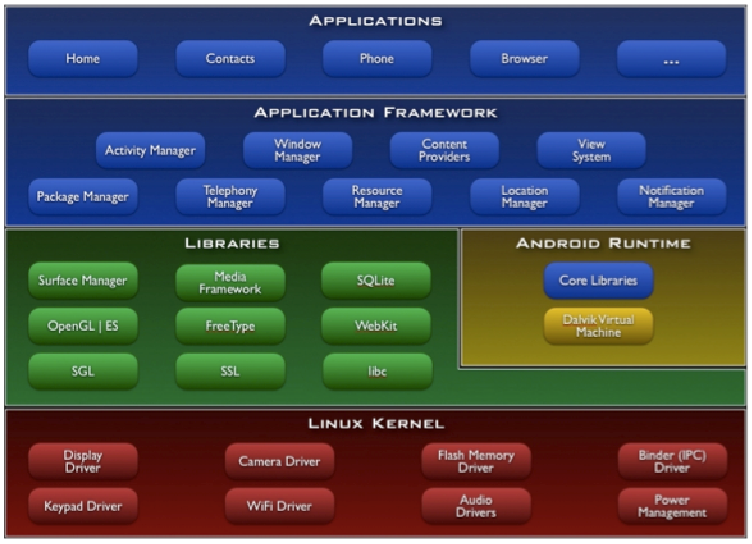


Figure 2. Application Architecture (Android 2012)

II.III HARDWARE VARIATY

Android is an open platform available for device manufacturers. This implies that each device provider can choose the set of features that it will include within the device. Moreover, this means that the developers will need to take into account and make the best effort to generate device-independent code so the developer's application will work regardless of underlying hardware. Specifically, functionalities regarding input methods, screens and sensors are the most variable hardware pieces among the different devices. James and Allen outlined (James 2010), (Allen 2012) that the main differences are in terms of input methods, sensors and screen size and resolution.

Input Methods: Different devices provide a variety of user input methods. Newer devices will provide touchscreens, while others will have an alternative method to access the screen, such as the Trackball or Track pad. Touch screen it is important to mention since there are different technologies that need to be considered in order to avoid drawings or touch combinations that could only be available to certain technologies (James 2010), (Allen 2012).

Sensors: Sensors are becoming a key component for the developer. They give environmental information, which is very useful considering the development of location-based applications that will incorporate gravity and temperature measurements. The majority of Android compatible devices will have three types of sensors: accelerometer gyroscope and magnetizer (James 2010), (Allen 2012).

Screen: There is a wide availability of screens and, the variability involves both screen size and pixel density. This is a key aspect when developing an application that attempts to be phone and tablet compatible and to consider the variability of resolutions (James 2010), (Allen

2012).

The Android API provides a structured way for the developer to detail the hardware that is required for the application to run as is expected using *Android ID* or *target system ID* to indicate the target devices for the specific application. In addition, the developer can specify the hardware that the application **requires** to run properly. The following snippet of code is an example of the tags and hardware requirements that the developer can outline in the manifest.

```
android:reqTouchScreen="finger"
                    android:reqKeyboardType="qwerty" />
<uses-configuration android:reqFiveWayNav="true"
android:reqTouchScreen="finger"
                    android:reqKeyboardType="twelvekey" />
```

The latest SDK that was releases is 4.0 Ice Cream Sandwich, which was released on October 19 of 2012. It will be evaluated in this research. Figure 2.2 shows the home screen of an Android device running Ice Cream Sandwich (Android 2012).

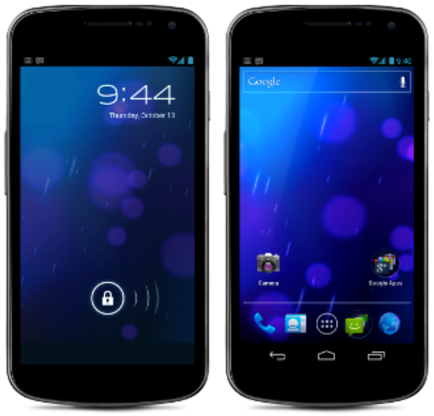


Figure 3. Ice cream sandwich home screen. (Android 2012)

II.IV SDK

The new SDK (Ice Cream Sandwich) brings improvements in the Multimedia area by allowing animation of videos and photos. In addition, the camera includes face detection capabilities. Moreover, the addition of new content providers and permissions allows developers to extend their applications capabilities including by being able to access: calendar, voicemail, and spell checker and contacts profile. For a detailed list of new features read Table 1.

Feature name	**Comments**
Social APIs in Contacts Provider	
Content Providers	Voicemail and Calendar
Multimedia	Incorporates video and photo animation
Camera	Incorporates face detection
Android Beam	
Wi-Fi Direct	
Bluetooth Health Devices	
Accessibility	
Spell Checker Services	To be incorporated by third party applications
Text-To-speech Engines	To be incorporated by third party applications
Network Usage	
Render Script	
Enterprise	
Device Sensors	
Action Bar	
User Interface and Views	

Input framework	
Properties	
Hardware Acceleration	
JNI Changes	
WebKit	
Permissions	To spell checker, voicemail, and profile contacts.
Devices Features	

Table 1. Ice Cream Sandwich SDK new features. (Android 2012)

II.V PLATFORM FRAGMENTATION

Given the variety of hardware devices and the fast evolution of the mobile platform developers might not be able to take advantage of all the new features of the latest SDK release. The mobile fragmentation is an important topic for Android developers since, it is not desirable to write code that will not be able to run on all hardware's. This subject is key in the Android world given the variety of devices and the fact that only “a 0.6 percent of Android phones are running Ice Cream Sandwich, about 55 percent are running its predecessor, Gingerbread, and 30.4 percent are on Frodo” (Enterprise Mobile 2012).. As a rule, developers should develop for the lowest version needed by application to have the widest audience possible.

II.VI DEVELOPMENT TOOLS

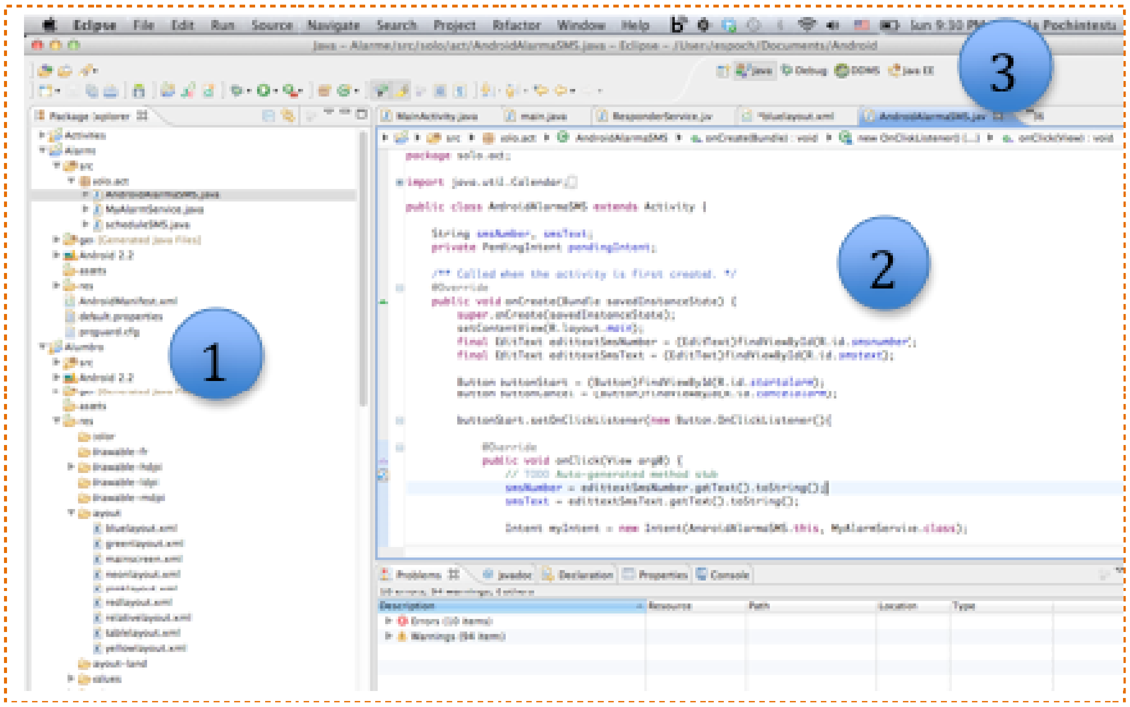
Android does not require a specific underlying hardware a Mac, Windows PC, or Linux can be used. The programming tools are free and can be acquired from the web. The software required is the SDK and ADT, which contain the libraries and frameworks that make up Android mobile platform and allow developers to write and test applications for the platform. The programming language environment is Java, however other languages such as Ruby and C/C++ can de used to develop native applications. The IDE commonly used is Eclipse, which offers a plug in that allows the creation, debugging and testing on Android compatible emulator devices of Android applications. The software that needs to be download and integrate in order to start creating Android applications are (Lee 2011) :

- Eclipse, Helio version or superior
- Java Virtual Machine (JVM), in order to run Eclipse
- Android Software Development Kit contains the Android libraries, emulator, libraries, documentation, and examples.
- Android Development Tools (ADT) plug-in for Eclipse that supports the creation and debugging of Android application.

II.VI.I Preferred IDE

The following figure is divided in four sections to show the main features that the most used IDE.

1. Project structure
2. Code editor
3. ADT, Android Tools (Debugger, Junit test, DDMS, Log cat)


1
2
3
Eclipse File Edit Run Source Navigate Search Project Refactor Window Help
Java Debug DDMS Java EE
Package Explorer
MainActivity.java main.java ResponderService.jav *bluelayout.xml AndroidAlarmaSMS.jav
Problems Javadoc Declaration Properties Console
Description Resource Path Location Type
Errors (10 items)
Warnings (84 items)

```
package solo.act;

import java.util.Calendar;

public class AndroidAlarmaSMS extends Activity {

    String smsNumber, smsText;
    private PendingIntent pendingIntent;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        final EditText edittextSmsNumber = (EditText)findViewById(R.id.smsnumber);
        final EditText edittextSmsText = (EditText)findViewById(R.id.smstext);

        Button buttonStart = (Button)findViewById(R.id.startalarm);
        Button buttonCancel = (Button)findViewById(R.id.cancelalarm);

        buttonStart.setOnClickListener(new Button.OnClickListener(){

            @Override
            public void onClick(View arg0) {
                // TODO Auto-generated method stub
                smsNumber = edittextSmsNumber.getText().toString();
                smsText = edittextSmsText.getText().toString();

                Intent myIntent = new Intent(AndroidAlarmaSMS.this, MyAlarmService.class);
```

Figure 4. IDE main screen Eclipse with ADT plug-in.

II.VI.II PROJECT STRUCTURE

Using the file new menu in Eclipse and selecting Android Application create each application. There is only one type or template available to create Android Mobile applications. At this point, there is no necessity of specifying which device this application is going to be built on. A series of different type of files will be auto generated and structured in a hierarchy of folders described below and show in Figure 5 (Lee 2011) and (Android 2012):

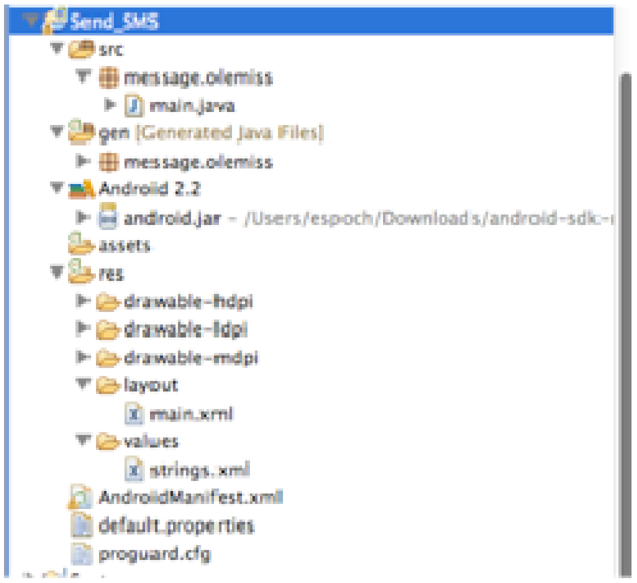


Figure 5. Android project structure

- **Src:** Allocates the Java source files for each file. The code for the application is written in java files and stored in this folder. Eclipse auto-generates the main Activity file AppName.java
- **Gen:** contains R.java, a compiler generated file that contains a reference to all the resources in the project.
- **Android versionNumber (Android 2.2)**: The folder contains one .jar file, which has all

the libraries needed in order to build Android applications.

- **Assets**: Holds the assets used by the application, such as HTML, text files, and databases.
- **Res:** Holds all the resources of the application. It has subfolders: drawable –resolution, layout, values. The file main.xml is auto generated and represents the layout that will be associated to the main activity.
- **Android Manifest.xml**: Some of the settings, components, and permissions needed by an application are defined here. It is an xml file, auto generated, that enables editing using the editor in Eclipse. All the application components need to be defined in this file, along with their Intent Filters.

II.VII APPLICATION DEVELOPMENT ARCHITECTURE

Each application consist of a set of components as mentioned before modeled by Java classes and XML file. The openness of the platform promotes reusability of tis components. Each Android application runs in a separate process and is composed of one, a group, or a combination of the components mentioned before. By default, all the components run on the same execution thread and the application is initiated by running the main component. Each application can be modeled as a block structure of its different components at Table 2 shows at (Android, 2012):

Component	**Functionality**	**Java Base Class**	**Examples**
Activity	UI component, corresponding to one screen.	Activity	Edit a note, play a game
Service	Background process	Service	Play music, update weather icon
Broadcast receiver	Response to notification or can initiate a broadcast.	BroadcastReceiver	Trigger alarm upon event
Content provider	Enable application to share data	ContentProvider	Open a phone contact

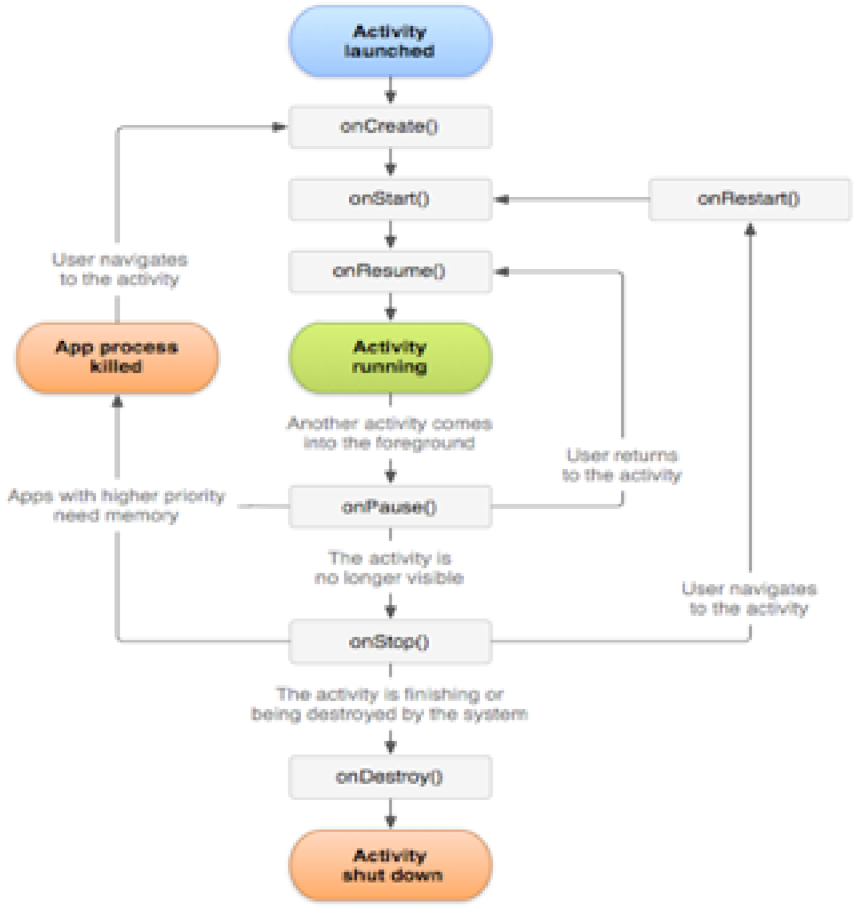
Table 2. Android application components

The communication between these components is performing by the Intent object aforementioned, which is the glue between the components. The following sub-section will detailed describe each component.

II.VII.I APPLICATION COMPONENTS

Activities are application components that represent the functionalities the user can perform while interacting with a screen. In general, an activity will have associated visual interface that occupies a single screen, but it can also draw widgets, dialogs or faceless. Activities use view objects in order to display content and manage the interaction with the user and events; these views create a view hierarchy of single, compound view, and layouts. The pair (activity, view) is similar to the view controller concept on the MVC (Model View Controller) design pattern. An Android application can be a combination of its own activities and other activities exposed by other applications. This allows for reusable tasks such as sending emails, text messages, or browsing the list of contacts on the phone to be easily integrated. The entry point of any Android application is the component defined as *main* in the manifest file. Each component besides the content providers has a lifecycle associated that defines the different states of the component between its launch and termination. The references for the entire section of II.VII.I are (Lee 2011), (Android 2012) and (Steele 2011) .

The activity application life cycle is described by seven states and several transitions. Each change of state corresponds to handling of a system interruption or event, as can be seen Figure 6.



Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Figure 6. Activity - Application component life cycle. (Android 2012)

The most relevant states from a developer's perspective are (Lee 2012):

- onCreate(). Initializing the activity is the first state on the life cycle. All activities override this method in order to link the XML file that lays out the activity's interface using the method setContentView() and creates a reference to the widgets that are on the view and need to be referenced programmatically.
- onPause(). When an interruption comes in like a phone call, the OS automatically sends the application to this state.
- onResume(). It is invoke when an application was in the background and it will be send to the foreground. It is important for developers to override in order to retrieve the state of the application so the user will be able to see the same information that was seen before.
- onDestroy(). When a configuration change happened the application automatically destroys, developers might override this state for the system to be able to re-lunch the application after the changes on the device configuration as was before.

The Android platform supports multitasking, which allows it to run several applications at once, as well the possibility of running background processes. Background processing is achieved by the use of the service application components. A background process is implemented by extending the Service base class. It is characterized by having its own life cycle and not having a user interface associated with it. Figure 7 shows the different states, where the developer can override any or all of the associated (onCreate(), onDestroy()) methods.

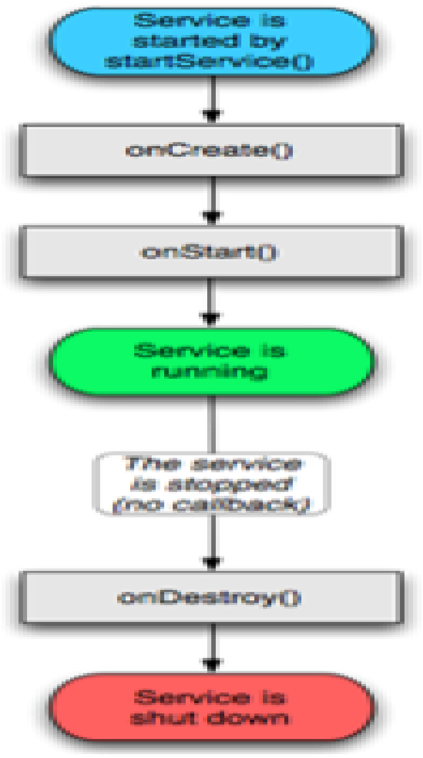

Service is started by startService()
onCreate()
onStart()
Service is running
The service is stopped (no callback)
onDestroy()
Service is shut down

Figure 7. Service application component life cycle.
(Android 2012)

Services can be started by being invoked by another Android component. There are two types of services, the main difference between them is the way that are instantiated:

- Unbound, which is a service that once started, can be running even though the component that started the service has finished or was killed.
- Bound services run until the component that the service is bound to stops running. The life cycle of the services consist of three callback methods, which are outlined in Figure 7.

Broadcast receivers are components used to listen and response to broadcast announcements. They are implemented as a subclass of *BroadcastReceiver* programmatically and each broadcast is delivered as an Intent object. It has a simple life cycle that consists of only one-callback method valid, for call duration.

“ *void onReceive(Context curContext, Intent broadcastMsg).*”

A broadcast can be sent as ordered broadcasts that allow sending one and catching by a broadcast receiver, and gives intent receivers the ability to exchange result of previous intent receivers or to abort the broadcast. The order of execution is controlled by the **android: priority** attribute defined in the *AndroidManifest.xml* file. Highest priority is 1000. Broadcast receiver objects are broadly used within the operating system. It uses these components in order to announce that the battery is low or an SMS was received, registering all of these messages in the notification bar.

Content providers act as a bridge between applications to enable them to share and exchange data. The content providers cannot be used directly; instead, a wrapper, which is defined as a content resolver object, needs to be instantiated. Each content provider has an associated URI used to differentiate with the following structure: *<standard_prefix>://<authority>/<data_path>/<id>*.

It is relevant to mention that there are content providers that are supported by the Android API and there are some others that were discontinued and undocumented on the API. For example, the content provider URL to view all the contacts that are saved on the:

*allContacts = Uri.parse(“content://contacts/peopl*e.

However, for the SMS and MMS the content provider is *content://sms/inbox*, but is currently not documented.

All the application components need to be registered on the manifest file, which was mentioned earlier as a component of the project structure. If a component is not declared in, the manifest is not visible to the system and cannot be invoked or run. However, either broadcast receivers can be declared in the manifest, or they can be created dynamically in code. It is relevant to mention that is not necessary to define the main application since it presents the default.

```
<activity android:name=".main"android:label="@string/app_name">
  <intent-filter>
   <action android:name="android.intent.action.MAIN"/>
   <category android:name="android.intent.category.LAUNCHER" />
 </intent-filter>
</activity>
```

The rest of the activities that are not the main, services, content providers broadcast receivers are defined with the following tags:

- Activities : <activity> <intent-filter> </intent-filter> <meta-data /> </activity>
- Service: <service> <intent-filter> </intent-filter> <meta-data /> </service>
- Content provider: <provider> > <grant-uri-permission /> <meta-data /> </provider>
- Broadcast receivers: <receiver> <intent-filter> </intent-filter> <meta-data /> </receiver>

II.VII.II COMMUNICATION COMPONENTS

In order for an application to work, all the components need to communicate and work together in order to achieve the general functionality. The intent object facilitates the different types of interaction allowed in the operating system. However, the intent can also be used to share information between activities or to specify that an action needs to be performed. From a developer's perspective, it is broadly used to specify an action that a developer wants to perform or wants to give the user the ability to do. In general, the intent is used in combination with the following methods:

- startActivity(intent), here the intent could represent an specific activity or a general action
- broadcastIntent(intent), here the intent is going to be broadcaster
- startService(intent) or bindService(intent), to start a service

Intents can be categorized as (Android 2012) on of two types.

- Explicit Intents specifically state which activity to invoke.
- Implicit Intents are broadcasts that define a type of action that needs to be completed but does not specify which activity should respond to the intent.

In order to declare implicit components, developers can also specify what components of the application are capable of in order to advertise the actions that the application component can perform. In order to do this, the developer needs to define an intent filter on the component definition on the manifest file. An example of registering an intent filter on the manifest is described below:

```
<intent-filter . . . >
    <action android:name="com.example.project.SHOW_CURRENT"
/>
    <action android:name="com.example.project.SHOW_RECENT" />
    <action android:name="com.example.project.SHOW_PENDING"
/>
    . . .
</intent-filter>
```

Implicit intents can be used from one activity to invoke another application component (activity) that is on the same application or another one. For example, calling the built-in application to send email instead of re-writing a component to perform the same action every time is demonstrated as a snippet below.

This intent structure allows for easy extensibility of the framework. For example, if a user has multiple browser applications installed, when clicking on a link, a menu will pop up and allow the user to choose which browser app to handle the intent.

```
Uri uri =
Uri.parse("sms:"+numberIntent.getText().toString());
Intent it = new Intent(Intent.ACTION_SENDTO, uri);
startActivity(it);
```

II.VII.III INTERFACE LAYOUT AND USER INTERACTION

Mobile applications are characterized by being event-driven by user interactions. The developer needs to be creative and design applications that will take advantage of the reduced space of the screen and the software features that the mobile platform provides in order to develop applications that can provide a rich and ergonomic for the user. For that reason, the following section will provide an overview of which elements conformed the screen, how the user interface can be enhanced, and the objects that will help to connect the components on the screen to the logic in the code. The reference material for this sub-section is: (Lee 2011), (Android 2012) and (Steele 2011)

Each activity usually has an associated screen or user interface. This user interface is presented to the user in a window, which is made up of a group of two types of views and linked to the activity programmatically by the method *setContentView* provided by the *Activity* class. In other words, each screen is represented as an associated hierarchy of views.

This hierarchy can be viewed as a tree with root and each internal node being a group of views while the leaf nodes are single views. The hierarchy of views is compound by objects derived from the *View* and *ViewGroup* classes. Objects sub-classing the View class represent single widgets or controls. Examples of views are: editText, Button, Checkbox, and Spinner for example. Objects sub-classing ViewGroup class, represent the different types of layouts. *ViewGroup* instances are responsible for the position, viewing, and color of its child. This characteristic is recursively propagated until the leaf nodes are reached.

The *ViewGroup* class defines the different layouts that a screen can use to present its views.

- Linear Layout arranges its children in a single column or a single row.

- Table Layout arranges its children in rows and columns like a table.

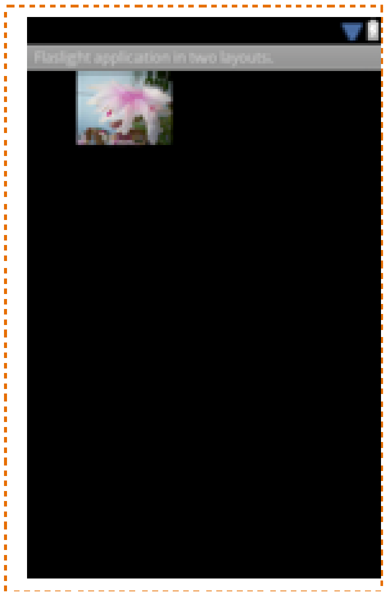
- Relative Layout positions the children relative to each other.

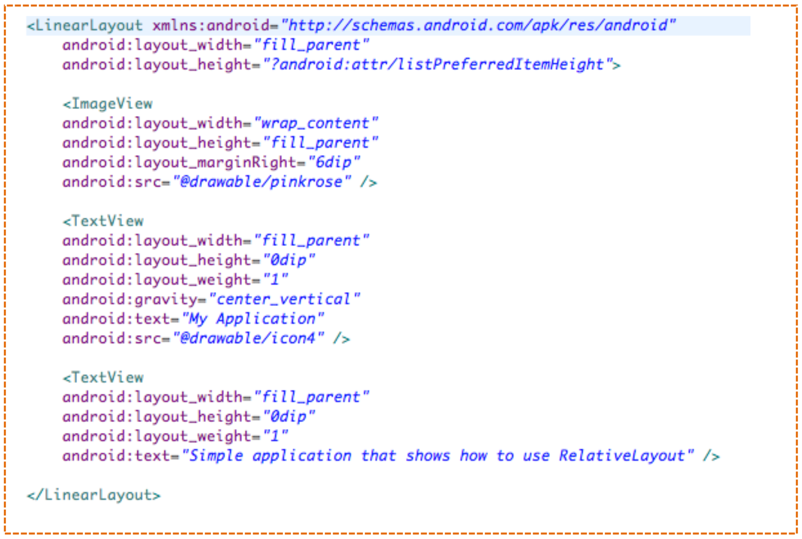
- Absolute requires specific screen coordinates for a component.

- Frame Layout defines a space on the screen to display a single item.

Compound layouts can be created. The Table Layout can be achieved using a horizontal linear layout and a vertical one. In addition, all the positioning of the views and widgets can be done with drag and drop controls and also by using the required XML tags that allow developers to set the orientation, background color, positioning, size, text types and more characteristics. Eclipse has a special drag and drop interface, or the screen can be designed directly in the XML.

In addition, the View Hierarchy for each object can be created programmatically on each activity, by extending the Java class that the framework provides. In addition, using the editor in an XML file can create the hierarchy. The name of an XML element in this file is in one-to-one correspondence with a Java class that it represents. The following snippet of code is an example of the definition of a Linear Layout view using the editor in Eclipse and Figure 8 represents the graphical view of the XML.





```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="?android:attr/listPreferredItemHeight">

    <ImageView
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    android:layout_marginRight="6dip"
    android:src="@drawable/pinkrose" />

    <TextView
    android:layout_width="fill_parent"
    android:layout_height="0dip"
    android:layout_weight="1"
    android:gravity="center_vertical"
    android:text="My Application"
    android:src="@drawable/icon4" />

    <TextView
    android:layout_width="fill_parent"
    android:layout_height="0dip"
    android:layout_weight="1"
    android:text="Simple application that shows how to use RelativeLayout" />

</LinearLayout>
```

Figure 8. Screen layout

In addition to the basic hierarchy, the Android framework provides several UI arrangements and event handlers. The most relevant ones are (Lee 2011):

- Gallery and Image View
- Image Switcher
- Picker Views
- List Views
- Web View
- Analog Clock and Digital Clock View

Moreover, often in game development the necessity of displaying content dynamically, which binds data to the views, depending on user’s input or interactions with the application; can be implemented on Android using the Adapter View object. In addition, with the layout of the screen event handlers need to be defined to capture the user events.

The event handlers are the objects that notify the application and the Android operating system about external objects or persons interacting with the application, in order to inform that action performed needs attention. The event handlers for widgets are defined using Java classes and registering it with the associated view. The view class provides a set of interface methods, named onAction, for example, *onClickListener*, *onTouchListener*, or *OnKeyListener*, that handle the different interactions that the user can perform with the views.

II.VII.IV EXECUTION WORKFLOW OF AN ANDROID APPLICATION

As was mention before the workflow of an application is defined by the interaction and calls of the application components and outlined by the life cycle. The entry point of any application is the Java class that models the component defined as the main one on the manifest. When the main application is found and its screen is loaded, the application will execute the onCreate() method in the main application. When it finishes processing, the application will be kept in the foreground until the user decides to no longer interact with it or an interruption (phone call, text message) comes in. Graphically, the execution of an Android application can be divided in three groups, which divided in Figure 9, by three concentric circles. The division can help the developer to divide the processing and functionality of the application in order to decide at which stages, it needs to be performed. Specifically, the most inner circle stages when the application is active, while the further represents when the application is alive, on pause, or the middle circle represents the active stage when the application is visible and interacting with the user on active or pause states and (Lee 2011) (Android 2012).

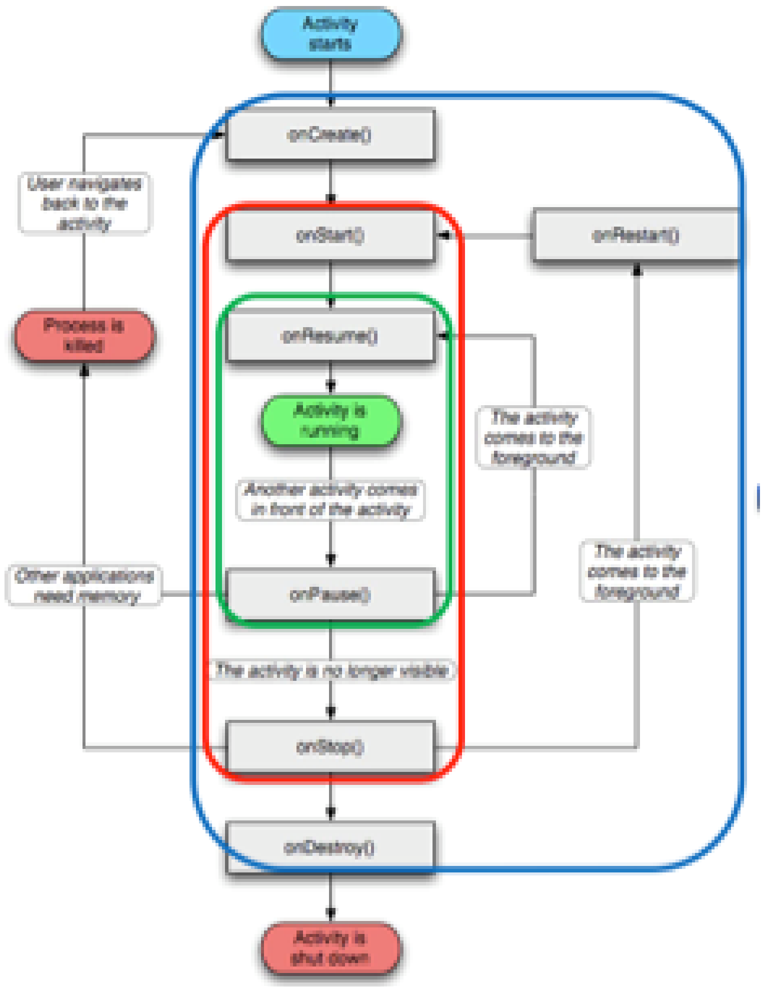

Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
The activity comes to the foreground
Other applications need memory
onPause()
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

Figure 9. Workflow of main activity. (Android,2012)

As was mentioned before the operating system supports multitasking capabilities where more than one application can be executed at a time. Moreover, it supports background applications, where a task is a sequenced collection of related activities that the user interacts with while performing a task. Normally, the activity that starts the task is called the root activity, and the navigation between activities generates a activity stack, which arranges the activities in a stack in the order in which each activity is launched. More specifically, “The way Android manages tasks and the back stack, as described above—by placing all activities started in succession in the same task and in a last in, first out fashion manner." (Android, 2012). However, it is also possible to interrupt this behavior and customized it according to the developers preference. Figure 10 describes a sequence of applications and how the applications are placed on a popped off activity stack.

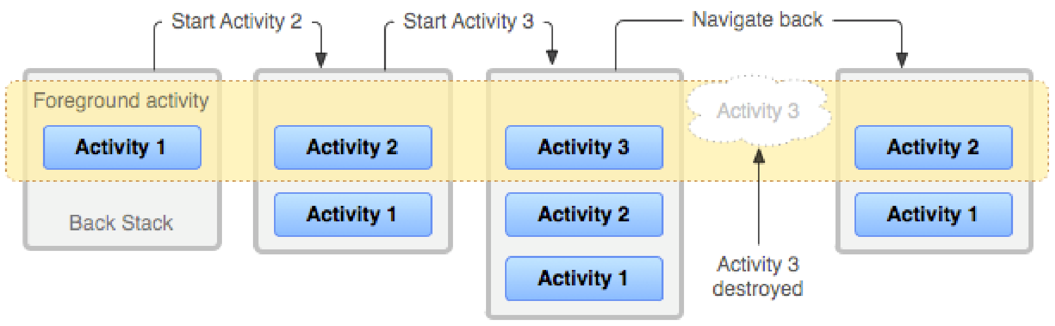


Figure 10. Activity stack. (Android, 2012)

II.VII.V APPLICATION PACKAGING

After going over the workflow, execution is interesting to understand how the component model is collected on an unique file that later will be publishable on the Android online store called Google Play. The component model is collected into the Android application package file, which has an APK extension. Each application is compiled and packaged in a single file that contains the application code, resources, and manifest file. It is usually compressed using a ZIP format, which is generally located under the directory of /system/app folder for built in applications and /data/app for third party applications installed (FileExtension.org, 2012) and (Android 2012).

The primary components of every applications are:

- **Assets** inform about application, HTML files.
- **META-INF** application metadata. There are several files in the META-INF folder namely, CERT.RSA, CERT.DSA, CERT.SF and MANIFEST.MF.
- **res** folder contains resource files that the application used, such as graphics, sounds, settings, and strings.
- **AndroidManifast.xml** file contains the application properties, it is an XML file.
- **classes.dex** is Dalvik virtual machine executable file. This file contains compiled Java source codes.
- **resource.arsc** is a binary resource file after compilation.

II.VIII FEATURES

The following section summarizes the most important features that the Android mobile platform offers in terms of software and hardware and the description of some relevant special feature that the platform offers to developers.

II.VIII.I MAIN FEATURES

The main features of Android in terms of software are related with the powerful multitasking model with background processing that offers to developers, thread programming management that allows the creation of asynchronous tasks in order to do not over load the main thread, where the UI is running. In addition, it provides debugger and testing tools including an emulator that can mimic most software and hardware features of real devices. On the other hand, in terms of hardware Android does not require devices to support a minimum set of peripherals, which present a clear disadvantage to the developer since he or she cannot use the API to optimize the application for a specific hardware. Table 3 summaries the main features that the new version of the SDK will include.

	Description	**Comments**
Debugger	Log cat, DMS perspective	
Networking support	WIFI, Bluetooth	
Memory management	Garbage Collector	
Testing	JUnit test	
Last version	Ice Cream Sandwich 4.0	Jelly Bean 4.1 coming fall 2012
System calls	Performed by using Android tasks	

Inter-process communication	Content providers	
Multitasking	Yes, background processing	
Location based services	Yes	
Location services	Full integration with Google Maps API	
Developing on devices	Emulator and real devices	
Web Services	Rest	
Persistence	SD card, SQLite	
Networking	WIFI, Bluetooth, 3G.	
Sensors	Broad variety	
Threads	Multithread programming	Main and UI run on the same thread
Application distribution	Online store, Google play.	

Table 3. Android framework main features. (Android 2012)

II.VIII.II SPECIAL FEATURES

Android has several built in applications, each of which has associated a database along with content provider that allows its manipulation. Below is the list of built-in applications that most of the Android devices will have. It is important to mention that not all devices might have the same set of application

- Alarm Clock
- Browser
- Calculator
- Camera
- Contacts
- Custom Locale (developer app)
- Developer Tools (developer app)
- Email
- Gallery
- IMEs for Japanese, Chinese, and Latin text input
- Messaging
- Music
- Phone
- Settings
- Spare Parts (developer app)

An interesting feature is the openness of the Android platform, which allows third party applications to have root access. By default, each application has the lowest level of privilege however it can ask for higher levels of privilege by adding the correct tags and information into the manifest. In addition, and thanks to the openness of the platform, Android allows developers to define their own applications as Android components that another developer can use to build their own applications. In order to do that the developer will need to create a library .jar and in order to other developers to use it they will only need to include it as any other library on Eclipse.

II.IX DISTRIBUTION AND MONETIZATION OF APPLICATIONS

The main distribution channel for Android applications is the online store provided by Google, called Google Play, which replaces the Android Market. In order to submit an application to Google Play, the developer will need to register using a Google account as a developer one by paying a one-time fee of $25 dollars. Applications can be distributed under the free or pay model. If the developer chooses to distribute the application with a price, it also needs to register as a Google Checkout Merchant. In addition, Google will retain a transaction fee of 30% of the total price and some countries will require that the prices reflect the inclusion of taxes. The developers are required to link to an AdSense account in order to receive payments from the applications purchases. Moreover, if the developer does not like the conditions or the application is decline for submission in Google Play, it can be publish on Amazon Market or the developer’s personal web page.

CHAPTER III
IOS MOBILE PLATFORM

The following subsection is intended to be a quick overview of the iOS mobile platform. Apple named the mobile platform iOS after the 4.0 version. The iPhone was released in 2007. In these chapter the version 5.0 is particularly discuss in this research, although version iOS 6 will be released in fall 2012. The iOS operating system is a reduced version the iMac operating system. The different version have been improving the platform, specially iOS 4 allow developers to write applications that perform background processing. The operating system philosophy is propietary and does not the developer any customization or direct communication with the operating system. Each application run on its own Sandbox and applications have restricted access towards sharing information with others applications. iOS provides a complete SDK and preferred IDE (Xcode) to help developers built applications for the mobile platform.

The following sections describe the platform in terms of: iOS development essentials, hardware variety, SDK, platform fragmentation, development tools, application development architecture, features and distribution and monetization of applications.

III.I iOS DEVELOPMENT ESSENCIALS

The mobile platform offers developers a complete SDK, grouped by framework functionality. The preferred IDE and to which the platform provides documentation is Xcode. Xcode provide a complete environment that allows developers to have code, interface view at the same time. In addition, code can de directly link to the user interface components by drag and drop controls.

The iOS system framework is built using object oriented design and is modeled around the following design patterns: Model View Controller, Delegate, Target-action, Block Object, and Sandbox. The iOS operating system assumes that each third party application is built using the MVC pattern. The rest of the patterns are mainly used to manage the integration with the framework, interaction with the user and to define and allocate the third application space within the iOS operating system. . The developer's code need to be integrated into the framework by modifying the hot spots of each framework component, it order to use it and access the system functionalities. The iOS supports the development of native and web applications based on HTML 5 and CCS. Each application represents a process that has only one entry point the main file.

The main patterns are described bellow: (Apple 2012) and (Smyth 2011):

- **Model View Controller (MVC).** Defines the overall architecture of an iOS application. An example is the UIKit framework, which is the core of any user interface application, and follows the principles of MVC.

- **Delegation.** Defines the inter-object communication and exchange of data. For example, the MapKit framework defines delegates to perform location-based functionalities.

- **Target-action.** Defines the architecture of the event handling system. As an example, the UIKit framework uses this pattern in order to handle the user actions for buttons on the user interface.
- **Block –objects.** Implements the callbacks and asynchronous calls. As an example, it is used by the framework in order to implement self-contained functions that are triggered by an event such as the scroll of a view when the user wants to edit a field or the view of animations and transitions.
- **Sandbox.** Defines the application space the application settings, data, and files are saved on the sandbox. This provides a secure environment for the operating system and the rest of the applications since they do not allow third party applications to access to anything other part than its own sandbox. Moreover, the sandbox does not allow applications to root in the system.

III.II ARCHITECTURE

The available frameworks are distributed within the iOS layered architecture. The mentioned architecture is built in four layers shown in Figure 11.

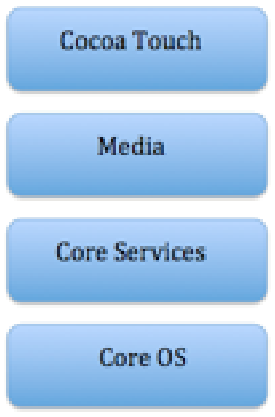


Figure 11. iOS software layered architecture. (Apple, 2012)

The layer closest to the hardware is called Core OS Layer, followed by Core Services Layer, which technically provides wrappers based on Objective-C to the services provided by the lower layer. The third layer is denominated Media Layer and provides the majority of the frameworks related with audio and multimedia services that iOS 5 offers. Finally, the fourth layer is called Cocoa Touch Layer, which is a minimized and modified version of Cocoa Layer for Mac Computers. It allocates the frameworks that help the developer to define the UI elements of the application, and manage and control the life cycle of applications and the frameworks that manage the user events and interruptions (Apple 2012).

III.III SDK

One of the main characteristics of the iOS platform is the mature and extensive list of frameworks available since it is built as a reduction of the available frameworks for Apple desktop and laptop computers. In iOS, the frameworks provide infrastructure to access the system components that the application needs. It provides the generic structure that each third party application will customize by adding specific application behavior. In order to give a quick overview of the main frameworks that can be found on the iOS mobile platform, Table 4 provides a description of each framework divided by layer. It is structured to emphasize how Apple designed the architecture in order to provide access and keep developers away from the low level programming concerns of the underlying hardware of the operating system itself (Apple 2012).

Layer	**Frameworks**
Cocoa Touch Layer	AddressBookUI, EventKitUI, GameKit, MapKit, MessageUI, Twitter, UIKit
Media Layer	AssetsLibrary, Audio Toolbox, AudioUnit, AVFoundation, CoreAudio, CoreGraphics, CoreImage, CoreMDI, CoreText, CoreVideo, GLKit, ImageIO MediaPlayer,OpenAL, OpenGLES, QuartzCore
Core Services Layer	Accounts, AddressBook, CFNetwork, CoreData, CoreFoundation, CoreLocation, CoreMedia, CoreMotion, CoreTelephony, EventKit, Foundation,MobileCoreServices NewsstandKit, QuickLook, StoreKit, SystemConfiguration and UIAutomation
Core OS	Accelerate, CoreBluetooth, ExternalAccessory , Security and System

Table 4. SDK iOS5 Frameworks

II.IV DEVELOPMENT TOOLS

The hardware needed to develop third party applications is an Intel Mac computer, either desktop or laptop version. The iOS operating system supports the development for iPhone, iPod, and iPad devices. In addition, the underlying software requires the hardware to be able to run Mac OS Leopard or later versions of Mac operating system. The development language used in the framework is based on Objective-C. Because Objective-C is a superset of C, the developer can easily integrate code based on this language into the application project. It is relevant to mention that information in section II.IV was extract from (Apple 2012), (Lee 2012) , (Smyth 2011) and (BeginningIos5Development 2012).

The iOS SDK includes the following elements: Integrated Development Environment (IDE), XCode, iOS Developer Library, Interface Builder, Emulator and Instruments (Apple 2012).

- **XCode**: Integrated Development Environment, is the centralized place that allows you to an application. It includes an interface builder, which provides a visually drag and drop interface builder. Xcode provides multiple iOS application templates: Page-Based, Empty Application, Single View Application, Utility Application, Tabbed Application, and OpenGL Game.
- **iOS Developer Library:** Provides frameworks such as: foundation framework, UIKit, Map Kit, Core Motion
- **iOS Emulator:** It emulates the iPhone abilities and is useful for testing. It is important to take into account that it cannot emulate the software feature. For that reason, the following functionalities can be tested: phone, camera, GPS, accelerometer, Bluetooth, and memory usage.

- **Instruments**: Contains tools that allow performance analysis and debugging tools, This allows measuring non-functional requirements related with memory usage, performance, and graphics performance.

III.IV.I PREFERED IDE

The preferred IDE and recommended by Apple is Xcode as shows Figure 12. It is possible to downloaded for free although applications will only be tested on the emulator and not real devices.

1. Project structure
2. Code view
3. UI view

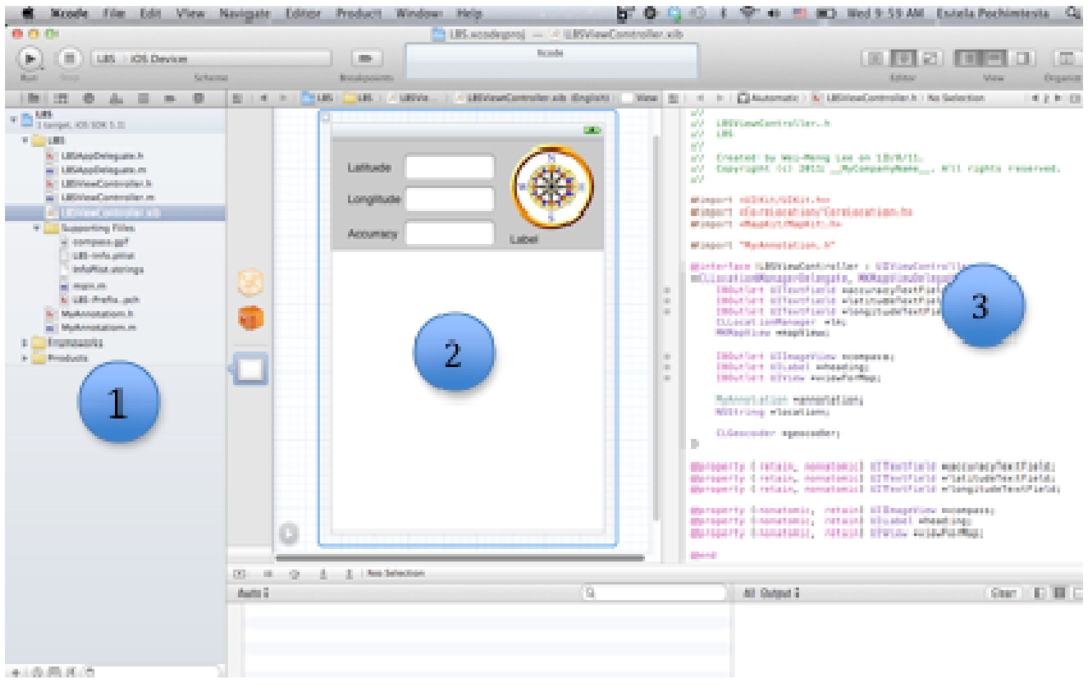


Figure 12. Xcode IDE

III.IV.II PROJECT STRUCTURE

It is important to notice that when a new project is created using the IDE tool (Xcode), the classes mentioned before are organized in a unique project layout that contains the four folders described below. It is important to notice that the IDE will auto generate some classes at the creation of the project, which corresponds with the following iOS application architecture . (Smyth 2011)

(1) **Name:** ApplicationName. Allocates the binary files of the compile version of the developer's source code and the files that define the user interface. Since iOS applications are developed using objective-C, each source code file has a header (.h) and implementation file (.m). The user interface file is named nib file whose extension is a .xib, a binary file that allocates all the user interfaces, objects, and connections with the view and controllers, its extension changed because now it is based on xml tags. **Auto generated files:** Generates the view controller files under the generic name of AppNameViewController.h and AppNameViewController.m extensions.

(2) Resources. Contains source code and resources that are not written in objective-C, but are necessary for the application to function. As an example, this folder can contain pictures or audio files. Files automatically generated:

a. AppName.plist: This is a property list xml file that contains properties such us language, built in applications called, whether background processing etc.

b. InfoPlist.strings. This is a text file that contains strings that might be referenced during the application if the developer's application implements internationalization, for example.

c. Main.m. contains the starting point of the developer's application, the main method.

d. App_Name_Prefix.pch. Contains a list of header files, from the external frameworks, that will be used within the application.

(3) Frameworks. Contains references to Apple and third party frameworks used by the application. It is important to take into account that Xcode will not load the framework by default this needs to be manually added by the developer.

(4) **Products.** Holds the application itself under a file called AppName.app.

The project organization is provided by Xcode in order to simplify developer's use of it. Xcode also creates an application bundle that is a directory on the file system. It groups the executable files and the other resources needed for the application as audio files or pictures.

III.IV.III APPLICATION PACKAGE

iOS package the application on its bundle, which contains the following structure (Apple 2012):

- Application executable
- Information property list file (info.plist)
- Application icons
- Launch images
- User interface files
- Ad hoc distribution icon
- Settings bundle
- Localize resource files
- Non-localize resource files

III.V APPLICATION ARCHITECTURE

In order to develop an application, first it is necessary to understand the underlying architecture, its components that come together to create an executable application. It is also important to understand the IDE structure, the project which represents the application, and the life cycle of a general application, since the third party application will be running on the operating system, which handles the third party application as well interruptions coming from a phone call or text message. As mentioned previously, the structure of an application is defined by the MVC design pattern. Given that each application skeleton will consist of a Model, View, and Controller, each application can have one or a group of MVC model.

III.V.I APPLICATION SKELETON

More specifically, the model component of the skeleton allocates the objects that model the data structure and business logic that defines the specific engine of the developer's application. In addition, the view component allocates the objects that present the application content on the iPhone screen and interact with the user. Moreover, the controller component allocates the objects that manage the views and provides the logic or protocol that handles the communication between the view and the model. The communication between the view and controllers is well structured in terms of actions, outlets, and delegate objects. The actions communicate the view to the controller through a controller method; the outlet allows the controller to communicate with the view in order to modify some of its properties. Finally, the delegate objects act on behalf of from (and at the request of) another framework object that handles interruptions of touch events so the controller can identify which element the user interfaces in the view and act accordingly, (Smyth 2011) and (Apple 2012).

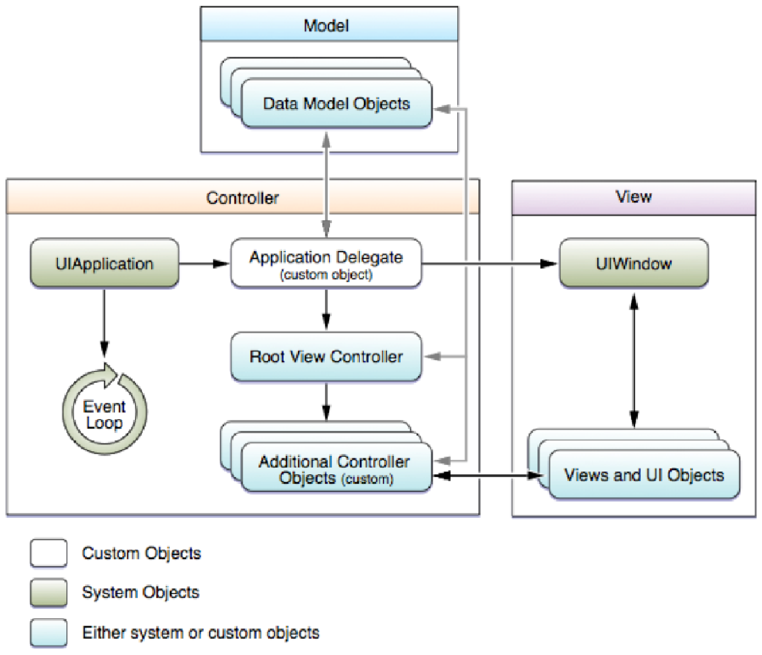


Figure 13. IOS application Skeleton. (Apple 2012)

Figure 3.3 illustrates the model, view and controller sections and the distribution and communication of core objects, which provides the basic structure for MVC component (Apple 2011)

- **Model**: Data model Objects
- **Controller**: UIApplication Object, App Delegate Object, and View Controller Objects
- **View:** Views and UI objects and UIWindow

The references on the picture categorize the different objects. The top group help the developer with the definition and construction of the model, view, and controller. For example, for the model it provides data classes in the foundation model that allows the creation of a custom data model if the model does not present a well-defined structure. Moreover, if the data model is based on files, the UIDocument framework provides services to define a data model based on documents. As is well known, the iOS applications are characterized by having a rich user interface. Keeping in mind the MVC model, it will be modeled as object views. iOS provides two: the UIKit framework and the OpenGL ES (The Open Graphics Library), used in cases of content change of the user interface or a programming environment that requires high frames and sophisticated drawings. Finally, for the controller the iOS offers drag and drop functionality from the Interface builder into the controller class in order to define outlets and actions without the necessity of writing code.

III.V.II INTERFACE LAYOUT AND USER INTERACTION

The phone is characterized by a very small screen, which allows creative arrangement of views to display content. The relationship between the view components is a very important part of the application design because iOS applications are event driven and most of the time the user initiates them by touch events. The UI is an important component of the application since it represents the screen that the user sees and interacts. The UI is compounded by a window object, a group of views organized in a hierarchy, and view controllers. The references used in the entire section are: (Apple 2012), (Smyth 2011) and (Lee 2012).

Each iOS application has one single instance of a window in charge of displaying content, which is an instance of the Window class. It also serves as a place to allocate one or more views. A view is an instance of the class *UIView* and is the primary mechanism for interacting with the user. Each view has a controller associated that is instance of the class *UIViewController*. The view controller is responsible for swapping the different views in the unique window and event-handling methods. As a summary, each application will have one single window with a group of views and controllers that the user is not be able to manipulate directly to open or close them. This feature needs to be implemented programmatically by the developer. The views are the main objects that the user sees and its main responsibilities are: Layout, sub view management, display of content, drawing, animation and event handler.

All the views that compound an application are structured in a hierarchy that needs to be understood in order to handle events correctly. Controllers are in charge of managing and propagating the events to the operating system, UIKit and Core Motion. For example, each specific type of view, such as the clock and table bar, is independent, but they will sit on top of each other, so that when an upcoming event needs to be passed to the right view. However, from a developer's perspective the views are not on top of each other, they form a hierarchy as a three layered structure. Figure 14 illustrates the views from the user's perspective. (Apple 2012)

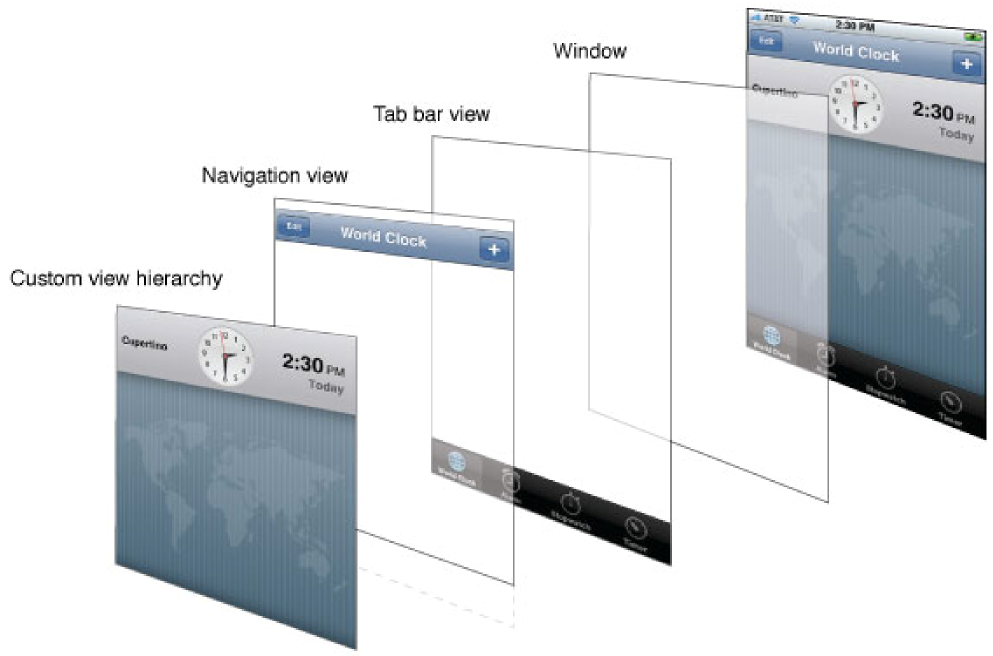


Figure 14. Stack of view's application. (Apple, 2012)

The developer can choose to create these hierarchical views programmatically or using the interface builder. The interface builder is recommended because the developer can assemble the views, see the relationship, and visualize how they will appear when the application is executing. The implementation of dynamic views is supported in iOS 5, which allows the generation of the application dynamically by receiving input from the user, the user interface will adapt in response to specific inputs. For more information, please see the Apple documentation (Apple 2012). Figure 15 shows a few of the navigation controllers.

iOS framework provides the following pre-built layouts (Apple 2012):

- Container View. The views that have complex behavior that allows them to perform higher functions rather than just displaying controls and contents. An example is *UIScrolView* class, which adds scrolling capabilities into the application without the necessity of the developer to code anything.
- Display View. Display content with a nice layout. An example is *UILabel* or *UIImageView* class.
- Text and Web View. Allows text or web content to be displayed in a structured way. For example, the class *UIWebView* allows the display of HTML data. In addition, UITextView allow the display and editing of text in a scrollable area.
- Alert Views and action sheets are views that show the user options or notifications that are the result of their interaction. For example, a *UIActionSheet* displays a menu option in the bottom of the screen.
- Navigation Views. Allows tab bars and navigation views to be implemented
- Table View. Provides a way to display a dynamic list of objects.
- UIPicker View and Date *PickerView*, allows the user to make a selection from a list of options. This view is used in the built-in calendar application

Each of these views has an associated controller, which is responsible for

- Creating the views
- Responding to orientation changes
- Unloading unused views during low memory conditions
- Navigating between the views and the related controllers that will help handle the

events and show or hide content-related views

UIKit framework provides the classes that manage the view controllers. Each view control is an object type of the UIViewController class. The main subclasses that manage the views described above are:

- UITabBarController
- UINavigationController
- UITableViewController
- UIImagePickerController
- UIContentController

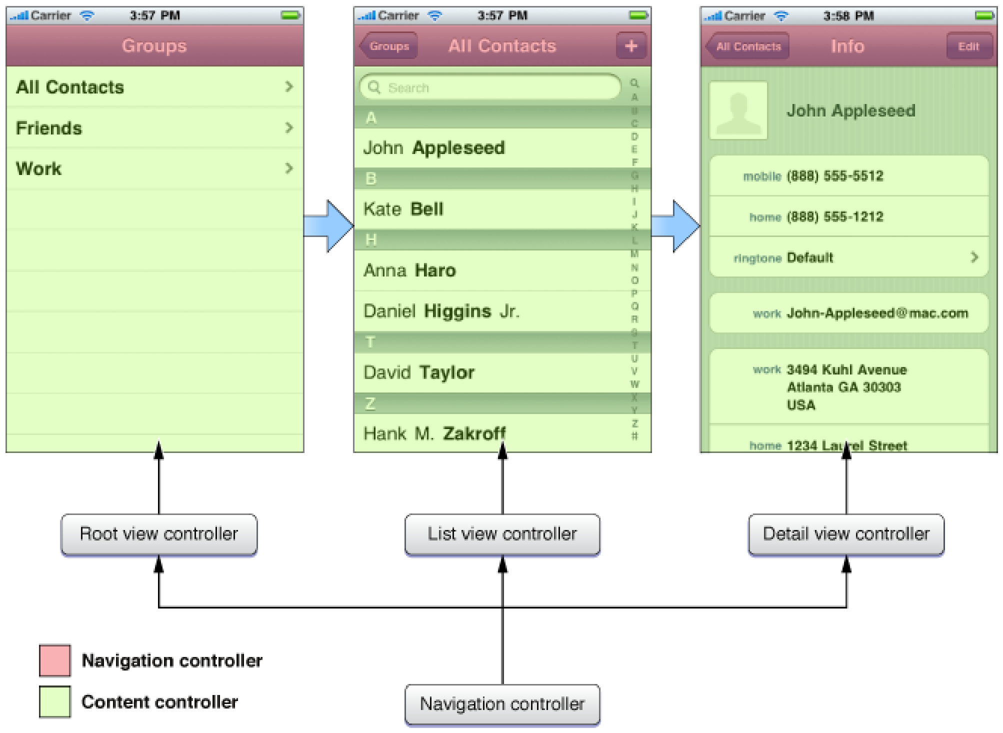


Figure 15. Navigation control. (Apple, 2012)

III.V.III LIFE CYCLE

The lifecycle, is a sequence of events and transitions that occurs between the launch and termination of the application. It is divided chronologically into six stages: (1) Launch, (2) App Initialized, (3) Load main lib, (4) Wait for event, (5) Handle event and (6) Exit Application. However, each application must behave differently in the Foreground and Background. The operating system notifies the application when a transition happens. This involves changing several aspects of the application behavior, and for that reason, the life cycle is divided in two section, depending on whether the application is launched from the foreground or background. (Lee 2012), (Smyth 2011) and (Apple 2011). Table 5 summarizes the main application states.

State name	**Description**
Not-running	The application has not yet started running or was just terminated
Inactive	The application is running in the foreground but not receiving events. It can be inactive for a period of time when the system prompts the user to respond to a system event such as: a phone call or SMS message, or the user locks the screen.
Active	Application is in the foreground and receiving events
Background	This will be the initial state an application has been launched from in order to execute background code.

Suspended	The application is in the background, not executing code. The system could automatically move an application to this state during low-memory conditions.

Table 5. Main application lifecycle states. (Smith 2011)

Figures 16 and 17 provide a visual representation IO of the life cycle application when it is launched from the foreground and the background respectively. These events/states are chronologically ordered from top to bottom, and each has two sides, which represent the transitions that happen within the framework source code and the developer source code. The label “Your code” means “Developers code.” It is important to point out that iOS 5 does not allow all applications to be run as background applications. Only applications that manage Audio, VOIP, or Location can be implemented as background processing.

Launch application in the foreground

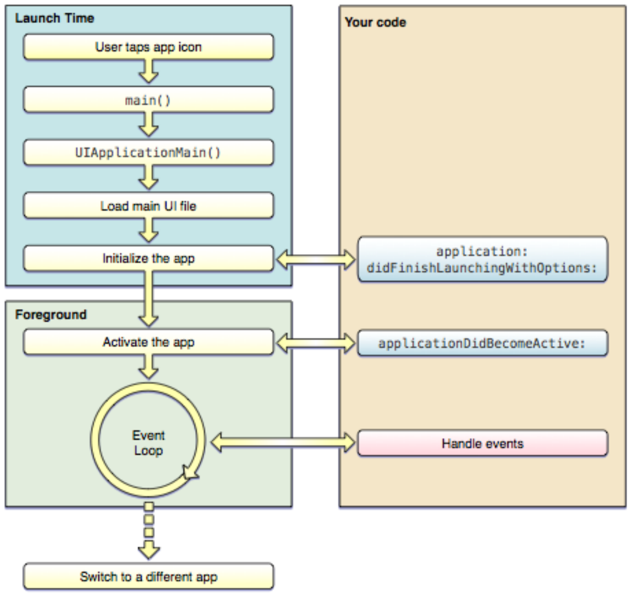


Figure 16. Figure3 6. Application lifecycle launch in the foreground. (Apple 2012)

Launch application in the background

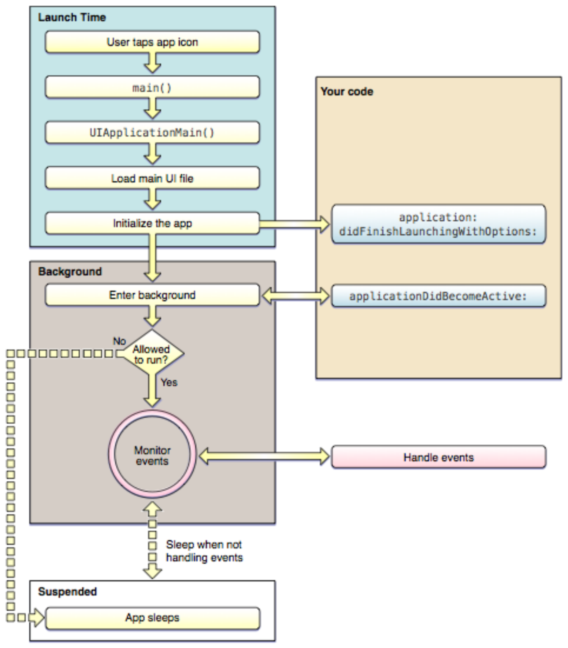


Figure 17. Application lifecycle launch in the background. (Apple 2012)

III.V.IV EXECUTION WORKFLOW OF AN iOS APPLICATION

The entry point for any application is the file Main.m that is auto generated, it will create an instance of UIApplication object, which belongs to the UIKit Framework. However, in the application project structure there is not a subclass of UIApplication class. Each has one instance of it, generally implemented as a delegate, and for that reasons the project has an AppDelegate class. After invoking *UIApplicationMain()*, reading in the .xib files to display the UI and linking up all the objects, the view controller is loaded. This will start the event loop, where it will remain until an event, or interruption is processed by the operating system. These events are classified in three categories. (Apple 2012) and (Smith 2011):

- Remote control, which are devices that are externally connected to the phone and are multimedia related, for example headphones.

- Multi-touch events, which are generated when the user interacts with the view.
- Accelerometer events or, an application that receives motion when the user moves the device.

Looking at the life cycle, the developer will need to think in which stage to perform the basic functionality of the application. These main stages or moments according to the life cycle. are: **Launch code**, at this moment all the functionality that needs to be accomplish right when the application starts needs to be implemented at the launch stage of the life cycle. **Run** is where anything that can change while the application is running at the loop stage of the life cycle goes. Finally, **Shut down,** contains the code that will liberate resources, and save data that needs to persist at the termination stage at the life cycle.

III.VI FEATURES

iOS provides an extensive set of software and hardware feature will be summarized in the following subsections. In addition, a description of the special feature that is unique to the iOS mobile platform.

III.VI.I MAIN FEATURES

iOS is characterized by providing strong application architecture defines by the MVC pattern and where Xcode allows developer to easily create user interfaces and complex navigation controls between the screens of the application. In terms of hardware, developers can take advantage of the uniformity and develop applications using the APIs that are optimized for the underlying hardware. Table 6 shows the main features that the SDK offers.

	Description	**Comments**
Location based services	Consume functionality of Google Maps web service and MapKit API to display and annotation of maps.	
Animation	Audio, photo and Video animation	
Persistence	SQLite	
Multimedia	Open GL ES	Provides a specific interface API that allows the creation of 3D animations and complex graphics techniques.
Web services	SOAP	API's that enable the third party application to consume web services using SOAP.
Networking		
Memory management	ARC (Automatic Reference Counting)	
Multitasking	Limited background processing	
Threading programming	Yes	Threads share memory

Table 6. iOS main features

III.VI.II SPECIAL FEATURES

This section will introduce some special features that the iOS offers to the programmers. The most relevant references at this point are (Smyth 2011), (Lee 2012), and (Apple 2012) (BeginningIos5Development 2012).

- **iCloud:** Allows sharing documents among a user's device. The API provides easy access using a generated URL that access to a share directory. The API provides services to access, query, and modify the documents store in the iCloud.

- **VoIP:** Allows integrating VoIP capabilities into iOS applications.
- **Bonjour Programming:** API that auto-discover devices connected over the same network. It is useful add to the third party application the functionality of sharing resources and files trough the network without adding to much complex of the current source code of the application.
- **Storyboards:** Allow developers to see in one place all the screens and their transitions that conform.
- **Integration of built in applications:** One of the main characteristics of IPhone is that includes a big group of built in applications. The developer has the right to invoke and use them in the third party application by using API's within the iOS SDK. However, the developer from the third party application has not root access so it is not allow to access to the built in applications data base or its code in order to modify it or built on top of them. A built in application can be invoke by using the URL associated to it, for example, the SMS built in application define the following URL.

```
NSString *stringURL = @"sms:+12345678901"; NSURL
*url = [NSURL URLWithString:stringURL];
[[UIApplication shared Application] openURL:url];
```

A side note regarding root access of third party applications is the fact that If the IPhone goes trough the Jailbreak process which is privilege escalation, the third application will have direct access to the lower layer API's tat allow you access to: iMessage CoreTelephony API for example. This will allow the developer to send a text message without the necessity of using the built-in application or intercept text messages before it is displaying on the screen as a new incoming message.

III.VII DISTRIBUTION AND MONETIZATION OF APPLICATIONS

iOS applications can be distributed on the Apple online store called Apple store. Developers need to be register on the Apple’s Developer Program. This program offers two types of memberships such as: individual or company with a yearly fee associated. As a developer of applications for the App Store, you are bound by the terms of the Program License Agreement. Applications can be distributed as free or charge of the download or use of it. Submitting updates to your application on the App Store.

CHAPTER IV
MOBILE PLATFORM COMPARISON

As was mentioned in earlier sections, this research focuses on Android and iOS mobile platforms. However, developers have several other mobile platforms to choose from beyond just Android and iOS. Although the mobile application industry is growing incredibly fast, there is still a lack of software engineering processes or best practices for mobile application development. However, it is generally accepted by developers that user experience, non-functional requirements, processes, tools, architecture, and portability are the main aspects that the developer needs to effectively address (Wasserman 2010).

In order to know which platform will be best for the developer's specific needs, it is useful to have documentation available that will expose the strengths and weaknesses of each platform around the main aspects that were mentioned before. Over the years, research has been performed to compare the most popular mobile platforms, including (Oliver 2009)(Asthana, 2012) and (Hall, 2009). However, previous research generally addresses the advantages and disadvantages of each platform and lacks a comprehensive analysis.

The outcome of this research will be a set of suggestions to address which platform is the better to use under certain conditions or functionality that the developer wants to achieve. On one hand, one platform might be more suitable for developing games given the broad and sophisticated APIs, which allow developers to create more complex animations and graphics.

On the other hand, this same platform might not be suitable to create a business application that requires sophisticated multitasking and screen transitions. The following chapter is divided into two sections: a comprehensive analysis of mobile platforms and practical analysis of mobile platforms. Section IV.I presents a fact based side-by-side comparison of both iOS and Android with a comprehensive set of comparison criteria used to evaluate each mobile platform, This comparison will aid in identifying weaknesses of each platform from a developers perspective. Section IV.II presents a practical comparison based on the author’s experiences through the implementation of basic functionalities. This approach will allow the reader to take a closer look at how to interact and use each framework and make use of APIs, which will provide a better understanding of the differences of each platfor

IV.I COMPREHENSIVE ANALYSIS

The following section presents a fact based, side-by-side comparison of the Android and iOS mobile platforms. Several criteria were identified in order to compare the two platforms. Those criteria are group into six different categories: **Development system, Developer Utilities, Testing and Devices, Hardware features, Application enhancement, and non-technical features**. Within each category, a group of comparison criteria is evaluated in order to present a more fine-grained comparison. The chosen criteria were based on the criteria used in earlier research (Leader, 2011). Table 4.1 gives a summarized review of the criteria discussed within each section.

The first comparison category, called **Development Comparison,** compares the basic hardware, software, and technologies a developer needs to be familiar with in order to being developing for each platform. Moreover, the steps involved in becoming a registered developer and creating the first application is explained. The second category is **Developer Utilities**, which aims to introduce the developer to the tools and instruments that each platform provides in order to manage the non-functional requirements of an application. This includes the performance and features available that will allow a third party application to communicate with its device and other applications, including the operating system. **Testing and Devices** is the third category, which describes the testing capabilities that each platform provides. Following testing features, the **Hardware Feature** category introduces the developer the functional hardware that each device incorporates and the API's and framework that support the hardware. The next category is **Application enhancement,** which summarizes the main utilities that the developer has available to incorporate the latest technology trends such as iCloud and VoIP. Finally, the last category is the **Non-technical Features** section, which aims to introduce the three main concerns that are not technical, such as application security, data privacy, and cross-platform development. In Cross-platform development developers write applications that can run in more than one mobile platform, is a native application that can run in more than one mobile platform. Usually the developer will use a third party framework that will allow this type of development.

IV.I.I DEVELOPMENT COMPARISON

The first set of comparison criteria is related to the main aspects that a developer will take into consideration when deciding on a mobile platform for development. The environment required for each platform includes software and hardware requirements, programming language, and user interface development. Some platforms may require extra effort by the developer to learn a new language or purchase software and hardware start up material.

Table 7 summarizes the differences between using the Android and iOS mobile platforms, looking at the infrastructure and environment of each mobile platform. Specifically, iOS requires the developer to purchase specific Apple hardware and software in order to start developing iOS applications, while Android does not require more than the average hardware, and the software can be acquired on the Internet free.

Another interesting point is that Android applications are developed in Java, a language that aims to be written once and run anywhere, as opposed to Objective-C, which is a language specifically for developing applications on the different Apple OS platforms. However, since Objective-C is known as a superset of C, it is possible for part of the source code can be written in C++ and incorporated into the project. Android is open source software licensed with Apache 2.0, which allows any manufactures to adopt it as the mobile operating system. This is not the case with iOS since it is not open and only runs on Apple devices. Finally, the application submission into the store is a key point for developers. Both platforms provide an online store that allows developers to distribute their applications under both the free and pay models.

The Android application store is called Google Play and in order for developers to distribute their applications, they need to register and pay a one-time twenty-five dollar fee. The Apple Store requires the developer to register in their developer program, which has several memberships with annual fees. It is relevant to mention that application submission and review process is very different for each platform, Apple has a strict set of guidelines that applications need to follow and they are reviewed before publishing on the market. As a final remark it should be noted that iOS requires a unique programming environment while Android is less restricted but strongly recommends using Eclipse as the IDE. In addition, the possibility of having an online store to distribute the applications is an advantage to the developer. The developer will have a tradeoff between having a big expense acquiring the tools required to develop with iOS or using Android that usually does not.

	Android	**iOS**
Supporting OS for development	Hardware requirements will be fulfilled with almost every computer available now days.	Hardware Macintosh computer either a desktop or laptop is required.
Development operating system	Windows XP, Vista or Windows 7. Mac OS X 10.5.8 or later (x86 only) and Linux (tested on Ubuntu Linux, Lucid Lynx).	iOS application can run is on Mac OS X.
Latest version	Ice Cream Sandwich 4.0 and Jelly Bean 4.1 coming this fall.	iOS5 and iOS 6 coming this fall.
IDE	Eclipse IDE Helios + JDT (Java Developer Tool) and the ADT (Android Developer Tool) + JDK 6	Xcode and integrated

	+ SDK.	development environment.
Open source	Yes	No
Programming Language	Java	Objective-C
Development device	HTC, Samsung, LG, Sony, and others.	iPhone, iPod and iPad
Application distribution	Google Play	Apple Store
Acquiring the software	Eclipse IDE+ JDK + SDK. Reference site: http://developer.apple.com/iphone	Xcode. Reference site http://developer.android.com
Programming model	Multicomponent	MVC design pattern
Registration developer	One time fee of $25	Individual program $ 99, Free program limitations towards publishing application and testing on real device and University limitations towards publishing on the store.
Creating the first project	Creation of application components: activities, services, broadcast receivers, and the UI using the XML.	Select template and target device. Design UI, define the model and the controllers.

Table 7. Development Comparison

IV.I.II DEVELOPMENT UTILITIES

One of the main challenges in mobile applications is to design an ergonomic and attractive user interface, which is difficult since the developer needs to fit all the user interface controls into a small screen and the user expects a nice look and feel. Moreover, the application will need to present an efficient response time. The performance of mobile applications is already limited by external factors such as: hardware and networking latency. It is important that the developer recognizes and incorporates the utilities that each framework exposes efficiently into the source code in order to reach higher performance rates. This category presents the following comparison criteria: thread and memory management, interruptions, networking, and notifications,

Table 8 summarizes the differences between using the Android and iOS mobile platform with respect the availability of tools that allow improving the performance of an application, for example, the garbage collector or power management tool. By observing the table criteria on memory management, iOS has a clear disadvantage compare to Android because of the lack of a garbage collector feature. The ARC (Automatic Reference Counting) on iOS tool can only take care of the assignment and release of memory. Both iOS and Android applications run in their own sandbox, which does not allow the application to exchange data or communicate with other applications directly. Conversely, the Android platform allows access to other application resources using content providers.

An interesting feature, more efficiently implemented by iOS, is the notification system, which has a few more choices when it comes to how the developer wants to show and receive notifications and even implement push notifications. As a summary, given the open nature of the

Android mobile platform, it offers developers who make third party applications the possibility of root access. Alternatively, an Android developer can use another third party application as a component and have access to build-in and third applications data. Regarding their programming the provide different approaches. For example by default, Android uses the main thread to modify the UI if components need heavy computation there is the possibility to define separate trends for these application components. Moreover, asynchronous tasks can done implementing multiple threads. Although in iOS there is a main thread and each functional component that can be performed on separated threads, the memory is shared by all of them. Having a large number of threads can slow the performance and consume more memory.

Another important aspect is the multitasking where both platforms present that capability but Android provides a special component with its own life cycle to be optimized to model background processing (a service). iOS introduces multitasking capabilities on the later version, but it is restrict to VOIP, audio or location services. Finally, regarding power system management neither of the platforms has come up with a solution to the battery consumption on mobile devices. However, iOS presents a superior approach that manages the software that is not being used in order to control the power consumption. As a concluding remark the developer will face a trade-off between having a more powerful memory management system or having more options and liberty regarding thread programming, background processing and inter-process communication.

	Android	**iOS**
Thread Management	Main and UI runs on the same thread. Wrapped on a runnable	Main

	object.	
File management system		
Memory management	Garbage collector.	Automatic Reference Counting (ARC)
Interruptions		
Application notification	Notification center	Push-up and notification center.
Multitasking	Yes, threads class to perform asynchronous processing	Limited background processing
Desktop synchronization	USB cable	ITunes
Inter-process communication	Content providers, intents.	URL protocol handlers and custom file type handlers by file extension
Power system management	toggle screen & keyboard back-light, brightness sleep and standby CPU mode, power setting change event and battery state change event	iOS manages the hardware that is not being use in order to control the power consumption.

Table 8. Development Utilities

IV.I.III TESTING AND DEVICES

The Android market, now Google Play, and Apple strongly enforce the testing of applications before making them available in the online stores. Apple specifically requires submission of any third party application to a review process before publishing it in the store. Given that, testing it is a very important task that the developer will need to perform before releasing the application. It is common knowledge that mobile testing is a challenging task. Testing is even more difficult because not all mobile platforms provide virtual devices that emulate the hardware and software device capabilities. Without a good emulator, a developer is required to purchase a device in order to test applications. In addition, the complex application life cycle that mobile applications have and need to handle incoming calls and messages at the same time, testing is even more difficult. Moreover, several mobile operating systems support a broad group of devices, so it is also important to be familiar with platform fragmentation and compatibility in order to develop applications relatively independent of hardware API version.

Table 9 summarizes the differences between the Android and iOS mobile platforms using the following comparison criteria: debugger, development on device, development on virtual devices, application notification, compatibility, test cases, and extensibility. Android faces the problem of fragmentation due to the disparate hardware and compatibility with the latest SDK version. This issue it is only present in Android because iOS offers a fixed set of devices developed by the same company that develops the OS. However, iOS only provides developers with a emulator virtual device, which only emulates part of the operating system capabilities. However, iOS provides, the APIs necessary to run structured unit tests on the applications feature that Android does not offer. In addition, iOS provides Xcode settings that allow applications to be made universally so they can be run on both the iPhone and iPad. This feature is achieved by the definition of two .xbi files, the auto-resizing widgets property, use of the latest SDK but oldest OS and defining weak links to the frameworks within the application. The developer will face a tradeoff between the ability of more powerful virtual devices (Android).

	Android	**iOS**
Debugger	util.Log and DDMS perspective.	Xcode
Test case	JUnit tests	Yes
Developing on devices	Yes	Yes.
Developing on virtual devices	Emulator that mimic the software and the hardware's phone.	Apple provides an emulator, which only is able to mimic the software environment.
Fragmentation	Yes, backward compatibility	Less, good updating service
Universal applications	No	Yes
Unit test	No	Yes

Table 9. Testing and Devices

IV.I.IV HARDWARE FEATURES

One of the advantages of third party applications is that they enhance the use of the hardware peripherals that the phone includes. Popular applications use sensors to determine the temperature, fine location, the allow tracking capabilities, and so on. Although each device presents its own set of peripherals, there is a basic set of hardware features, which are the comparison criteria under this category: input options, graphics, memory expansion, GPS, sensors, and camera. From the developers' perspective this is important if the application that is going to be developed requires a certain set of hardware on the device in order to function properly.

Table 10 summarizes the differences between the hardware that Android and iOS mobile platforms offer to developers. iOS does not include the possibility of a memory expansion slot, which is required by some applications. Android includes a mechanism the developer can used to describe the hardware needed for the application to run smoothly. Both platforms present several input options. In particular Android with the text-to-speech and speech recognition API allows the developer to add a new input method to access the phone. Also, both mobile platforms provide APIs that allow the user of the GPS to implement navigation and location aware services. In particular, Android includes the version of Google Maps API while iOS signed a service agreement to consume a set of features that the Google Web Services offers and has its own API to allow annotation and the possibility of embedded maps. An interesting feature is the possibility that iOS offers to avoid the constant use of GPS in order to get location coordinate. Instead, it will use alternative methods according to the precision of the coordinate update request in order to save power consumption. Android devices provide an extensive list of sensors, although it is not guaranteed that all the devices will have a particular sensor.

Both platforms define graphics in terms of windows, layers, and views. The main difference is that in iOS, several processes (applications) share the screen, which is represented as an one window, and in Android, each activity has an associated screen. As a summary, both platforms present the basic hardware with API's that allow the developer to interact with them using a high level language that provides an abstraction of low-level programming details. The developer will face the tradeoff between not being able to develop software which will be optimized for the underlying software or have the option of memory expansions lot and more sensors that might be available.

	Android	**iOS**
Input Options	Depends on the hardware usually Touchscreen or quart keyboard. Speech interfaces	Touchscreen with keyboard on the screen.
Graphics	Windows, hierarchy of views and layers	Windows, hierarchy of views and layers
Memory expansion slot	SD cards	No
Network capabilities	WIFI, 3G+ Bluetooth, and tethering.	WIFI, 3G+ radio capabilities and Bluetooth.
Navigation	Gps with Google Maps integration	Yes with a reduce version of Google Maps and its own map view API.

Camera	Varies	2 pixels with a front facing camera.
Sensors	Varies up to 13.	Accelerometer, gyroscope, magnetometer, and shaking detector.

Table 10. Hardware Features

IV.I.V APPLICATION ENHACEMENT

The mobile platform industry is becoming very competitive, where each platform aims to provide a broad range of extraordinary software features that will allow developers to create interesting applications with new technologies or software components that will improve the user experience. In addition, developing cross-platform applications is an achievable task by cross-compilation or by using web development techniques (Hartmann 2011) . The comparison criteria considered within the software features category are: Cloud, framework customization, localization/internationalization, VoIP, advertisement features, UI design, multimedia, web development, and cross-platform development.

Table 11 summarizes the differences between the software features that Android and iOS mobile platforms offer to developers. The summary suggests that on one hand iOS of supports iCloud and VoIP features, for example. On the other hand because Android is an open platform, it supports framework customizations that will allow developers to write applications that replace the functionality of any major system component, including the home screen. In addition, both platforms support web applications and cross-platform development by using HTML5, Java Script and CSS technologies or C++, C# and Ruby for cross-development platform by using external frameworks, such as PhoneGap or Titanium. iOS leads by incorporating Apple TV on the equation and the performance of video animation. Regarding animation, iOS provides a set of callbacks in order to listen to some animation event while Android listens to them through the definition of listeners as inner classes of the main java class. In summary, the developer will face the tradeoff of having the possibility to develop third party code applications or having more possibility towards the multimedia and cloud programming techniques.

	Android	**iOS**
Cloud	Google sync	iCloud
Framework customizations	Yes	No
Localization/Internationalization	Yes	Yes
VoIP	No	Yes
Inclusion of advertising on the application	AdMob	AdMob and iAid
UI design	XML	Xcode or programmatically.
Multimedia	Recording of video and images, audio, effects	Recording of video and images, audio, effects
Animation		Asynchronous, runs on a different thread from the UI
Social network integration	Yes, 3 party solutions	Twitter
Cross platform development	Yes + HTML 5, Java Script, CSS	Yes + HTML 5, Java Script, CSS

Table 11. Application Enhancement

IV.I.VI NON-TECHNICAL FEATURES

There are some aspects less technical but as important as the rest. For example, with more online services incorporating mobile device support, security and privacy must be fully supported. In addition, the privacy of personal data that is stored on the phone can be shared without the user's permission if developers do not take precautions. Recent research suggests the appearance of malicious applications in the online stores for mobile software (Zhou 2012), (Davi 2011).

Table 12, summarizes the differences between the implementation of non-technical requirements in the Android and iOS mobile platforms. Neither of the platforms offers a fine-grained privacy solution in order to protect the user's private information. Regarding security, both platforms provide encryption solutions to exchange personal information over the network provided by the operating system or third party developers.

	Android	**iOS**
Security	None	Encryption of hardware and software.
m-commerce	Yes	No
Privacy	Permissions on the manifest.	By default built in applications, access certain user data as contacts, calendars, and photo library.
Portability	Yes	No, tight to the OS.

Table 12. Non-technical Requirements

As a closing remarks of the section IV a matching concepts is summarized on Table 13. On each platform it is represented, which will be useful for developers that are already familiar with one platform to familiarize with the commonalities and difference of iOS and Android platforms

	Android	**iOS**
Application	It is a multicomponent unit	Monolithic
Communication between activities	Content providers and intents	URL
Project properties	XML File	XML File
Navigation between activities	Intents	Navigation controls
Running application environment	Sandbox and permission system	Sandbox
Event handler	finviewObject and listeners for widgets	Outlets and actions

Table 13. Androiod & iOS Matching page

IV.II PRACTICAL ANALYSIS OF MOBILE PLATFORM

The results of the side-by-side comparison in the previous subsection, where the two development platforms were compared based on a variety of criteria gives a purely theoretical comparison. In addition, to the mentioned comparison a practical hands-on is conducted thought the development of applications on both platforms in order to assist in the analysis of the platforms. This section describes the implementation and development process provided by each platform through a series of specifically chosen applications.

The analysis is focus on functionalities related with the familiarization of the platform, internationalization, UI design, and navigation features. Each application class or feature to develop will be describe in general in terms of application objective, type of application and components. Followed by the implementation on each platform. Finally, at the end of each implementation application a summary on how well each mobile platform implement the feature is describes.

Application name: Hello World

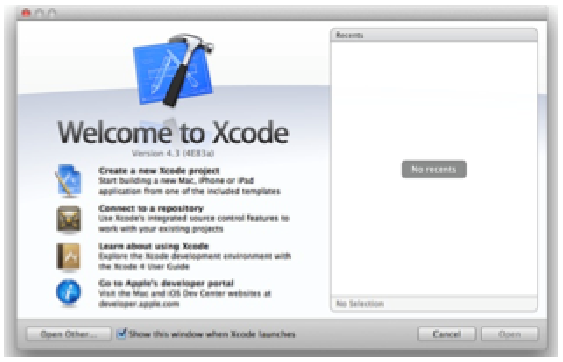
Application objective: Introduce the platform tools that facilitate the developer's task towards the creation and management the project.

Type of application: Application that response to basic user input.

Components: Basic UI with text field, label and button.

iOS platform

After installing Xcode, the creation of the project involves the lunch of the Xcode application, the first time will prompt into the welcome screen, select Create a new Xcode project.



Welcome to Xcode
Version 4.3 (4E83a)

Create a new Xcode project
Start building a new Mac, iPhone or iPad application from one of the included templates

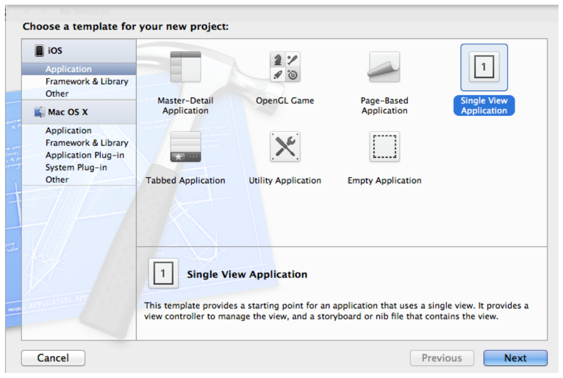
Connect to a repository
Use Xcode's integrated source control features to work with your existing projects

Learn about using Xcode
Explore the Xcode development environment with the Xcode 4 User Guide

Go to Apple's developer portal
Visit the Mac and iOS Dev Center websites at developer.apple.com

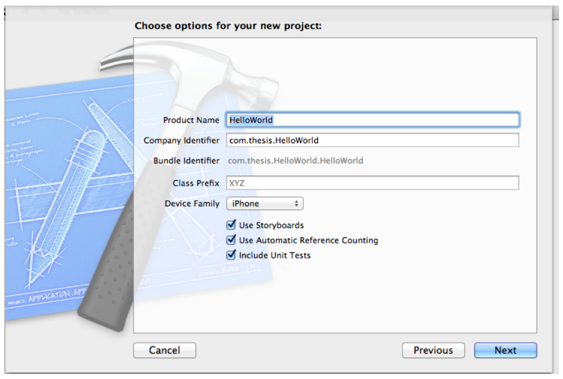
☑ Show this window when Xcode launches

1. Xcode opens a new window to choose an application template. Xcode includes several built-in app templates, for example, the Tabbed template creates an app that is similar to iTunes and the Master-Detail template creates an app that is similar to Mail. In this example the "Single View" template was selected. In addition, a project detail dialog window opens to fill out: **Product Name**: HelloWorld **Company Identifier**: Your company identifier, if you have one. **Class Prefix**: HelloWorld. **Device family:** iPhone.



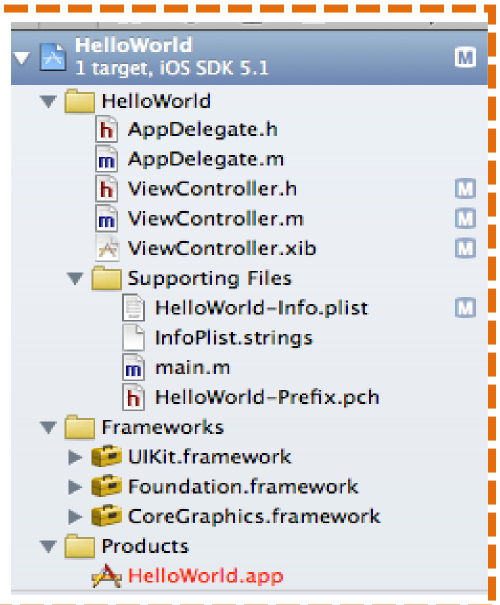
Single View Application

This template provides a starting point for an application that uses a single view. It provides a view controller to manage the view, and a storyboard or nib file that contains the view.



Product Name: HelloWorld
Company Identifier: com.thesis.HelloWorld
Bundle Identifier: com.thesis.HelloWorld.HelloWorld
Class Prefix: XYZ
Device Family: iPhone
[✓] Use Storyboards
[✓] Use Automatic Reference Counting
[✓] Include Unit Tests

Afterwards, Xcode opens your new project in a window (called the *workspace window*), which displays several tools of Xcode and the auto generated project structure.

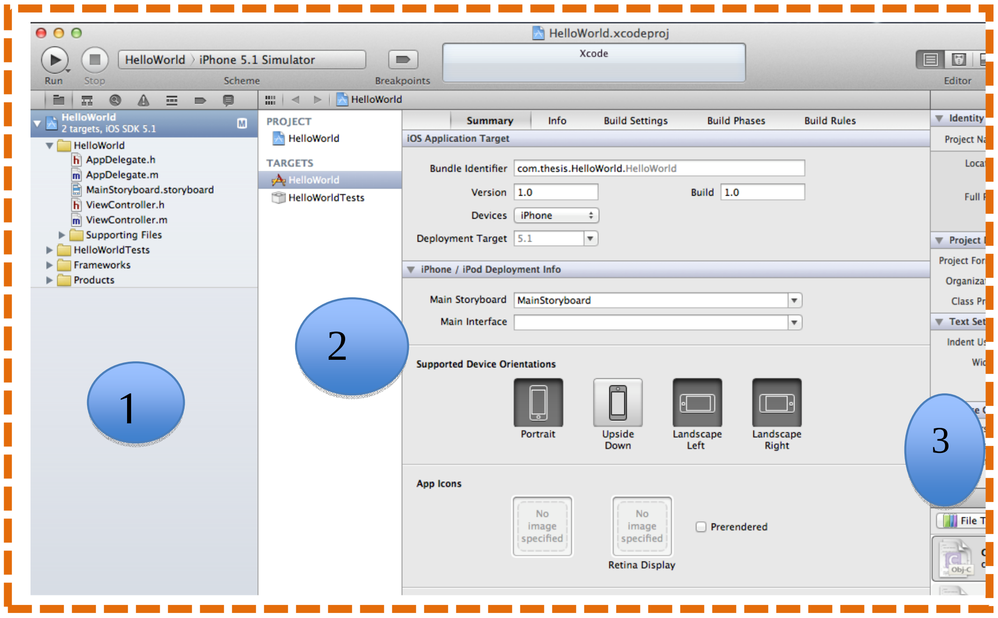


```
HelloWorld                          M
1 target, iOS SDK 5.1
  HelloWorld
    AppDelegate.h
    AppDelegate.m
    ViewController.h                M
    ViewController.m                M
    ViewController.xib              M
    Supporting Files
      HelloWorld-Info.plist         M
      InfoPlist.strings
      main.m
      HelloWorld-Prefix.pch
  Frameworks
    UIKit.framework
    Foundation.framework
    CoreGraphics.framework
  Products
    HelloWorld.app
```

In the picture where the project structure is showed different files, it is relevant to mention that the source code files have a m and h version because of the underline language objective-C, which has a header (.h) file and an implementation (.m). **The delegate** pair of flies contains the logic of the life cycle application and it is the developer should add the custom code into the overall life cycle application to achieve the desire functionality. **The view controller** pair of files manages the UI components. **The nib (.xib)** file provides an interface to the application.

The workspace window is virtually divided in three main areas:

1. Navigation
2. Editor
3. Utility



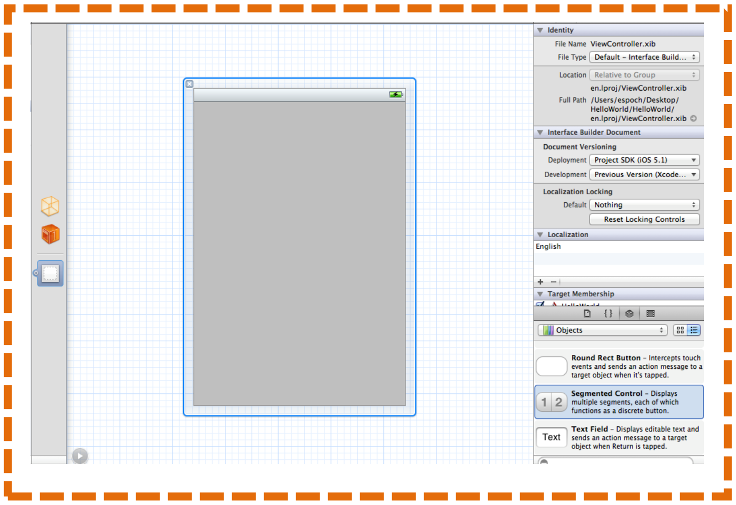
Bundle Identifier: com.thesis.HelloWorld.HelloWorld
Version: 1.0
Build: 1.0
Devices: iPhone
Deployment Target: 5.1
Main Storyboard: MainStoryboard
Main Interface:

Because an Xcode template was used, much of the basic app environment is automatically created, including the application file within the Products folder. **The delegate files** is where the developer will integrate the custom code into the overall life cycle of the application. **View Controllers files** manages the UI components. **Nib file** provides the interface to the application.

Each iOS application is develop around several patterns, specifically the MVC and Application Delegate define the main skeleton of the application. The Hello World application does not require a complex model definition for that reason it is embedded into the controllers.

UI Configuration

Using the interface building embedded on Xcode, within a white empty window which represent the screen that will be presented to the user and interact with it. The first step will be adding two widgets to the window: label and a button. Selecting the objects does it and draping into the screen, notice that the re-sizing, color text alignment is done by the tools bard.



Identity
File Name ViewController.xib
File Type Default – Interface Build...
Location Relative to Group
en.lproj/ViewController.xib
Full Path /Users/espoch/Desktop/HelloWorld/HelloWorld/en.lproj/ViewController.xib

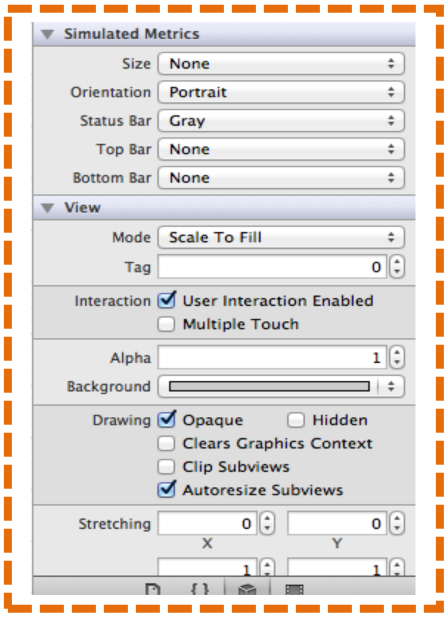
Interface Builder Document
Document Versioning
Deployment Project SDK (iOS 5.1)
Development Previous Version (Xcode...
Localization Locking
Default Nothing
Reset Locking Controls

Localization
English

Target Membership

Objects

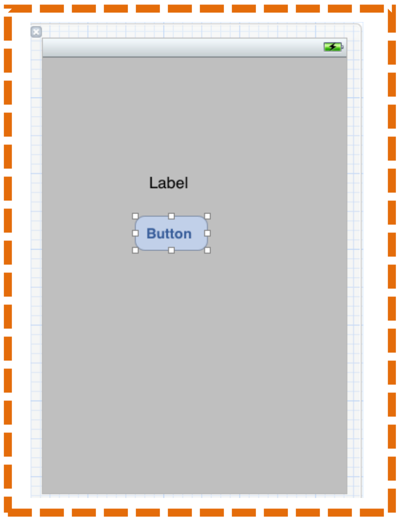
Round Rect Button – Intercepts touch events and sends an action message to a target object when it's tapped.
Segmented Control – Displays multiple segments, each of which functions as a discrete button.
Text Field – Displays editable text and sends an action message to a target object when Return is tapped.

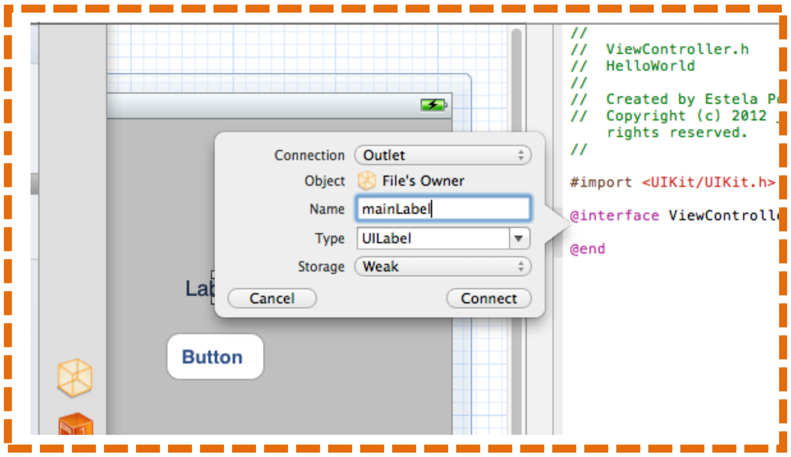


Simulated Metrics
Size None
Orientation Portrait
Status Bar Gray
Top Bar None
Bottom Bar None

View
Mode Scale To Fill
Tag 0
Interaction ✓ User Interaction Enabled
☐ Multiple Touch
Alpha 1
Background
Drawing ✓ Opaque ☐ Hidden
☐ Clears Graphics Context
☐ Clip Subviews
✓ Autoresize Subviews
Stretching 0 0
X Y
1 1

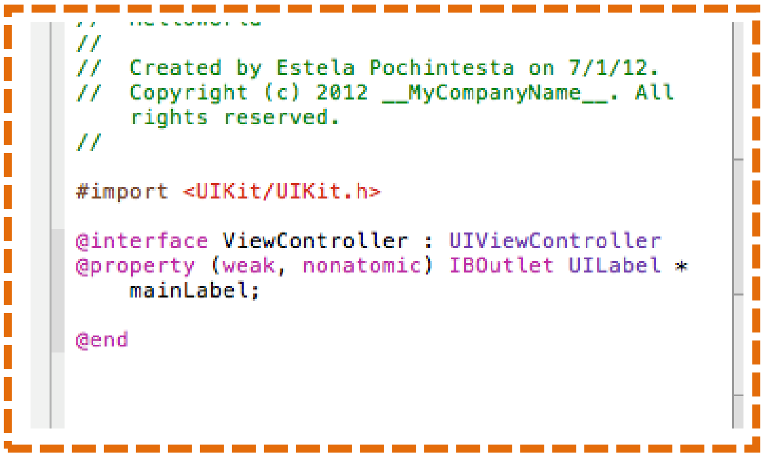
Configuration the code after placing and layout the widgets on the screen it is necessary to integrate them within the code and define controls. iOS use outlets and actions to achieve this. Outlets are define over widgets that need to be referenced within the code, in this case the label need an outlet so when the user press the button the application can set the hello world text in order to be display. This is achieving while pressing the control button over the label a connection will be establish from the design to the code view. It sets a property that it can be use to retrieve or set the text on the label.





Connection Outlet
Object File's Owner
Name mainLabel
Type UILabel
Storage Weak

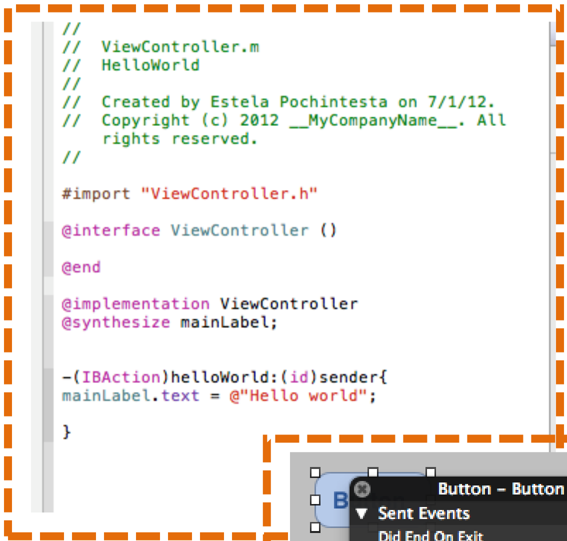
```
//
//  ViewController.h
//  HelloWorld
//
//  Created by Estela P
//  Copyright (c) 2012
    rights reserved.
//

#import <UIKit/UIKit.h>

@interface ViewControll

@end
```



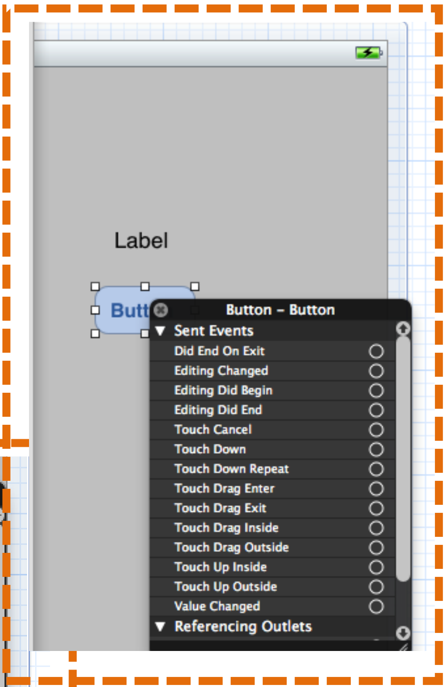
```
//
//  Created by Estela Pochintesta on 7/1/12.
//  Copyright (c) 2012 __MyCompanyName__. All
    rights reserved.
//

#import <UIKit/UIKit.h>

@interface ViewController : UIViewController
@property (weak, nonatomic) IBOutlet UILabel *
    mainLabel;

@end
```

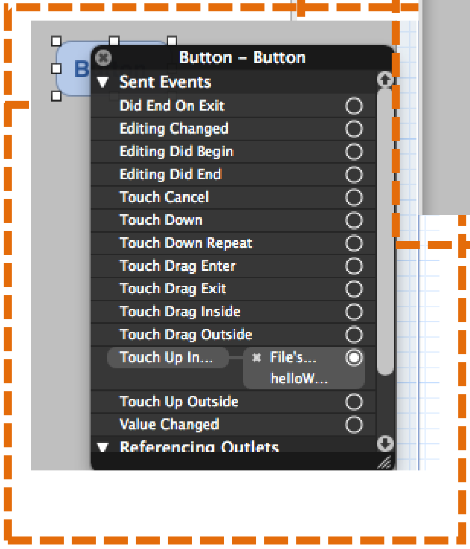
The next step is to create a method that will set the text and it is triggered by the user pressing the button. In order to create this method and then associated to the button an action object is define

The definition of the method is done on the viewController.h file and the implementation on the viewController.m file. Afterwards, a connection needs to be establishing between the code and the button. This is done by do a “right click” on the button and pressing control make the connection with the code view.

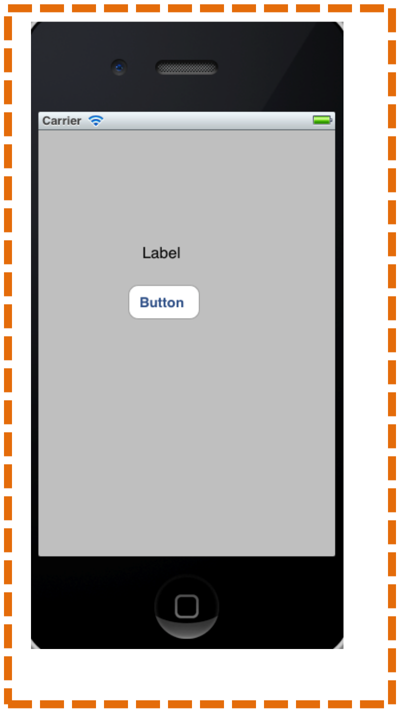


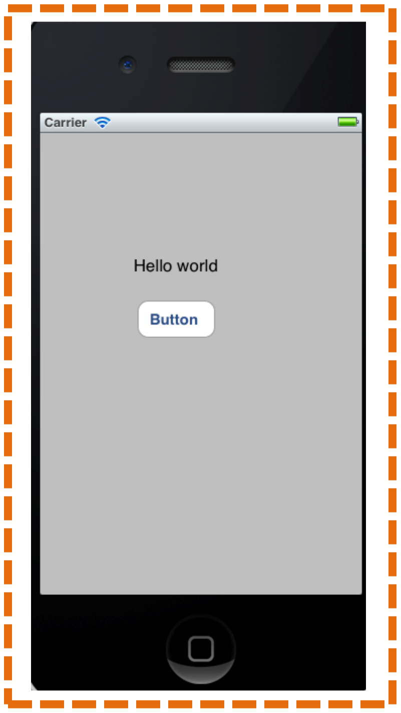
```
//
//  ViewController.m
//  HelloWorld
//
//  Created by Estela Pochintesta on 7/1/12.
//  Copyright (c) 2012 __MyCompanyName__. All
    rights reserved.
//

#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController
@synthesize mainLabel;


-(IBAction)helloWorld:(id)sender{
mainLabel.text = @"Hello world";

}
```





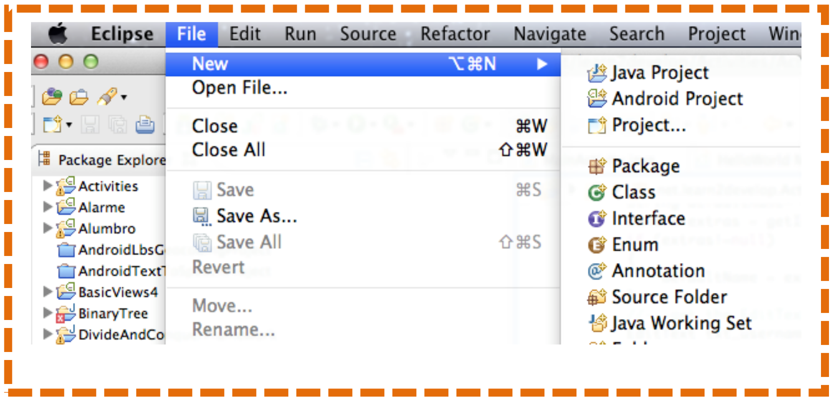
Now everything is ready for execution. It is going to be tested on the emulator.

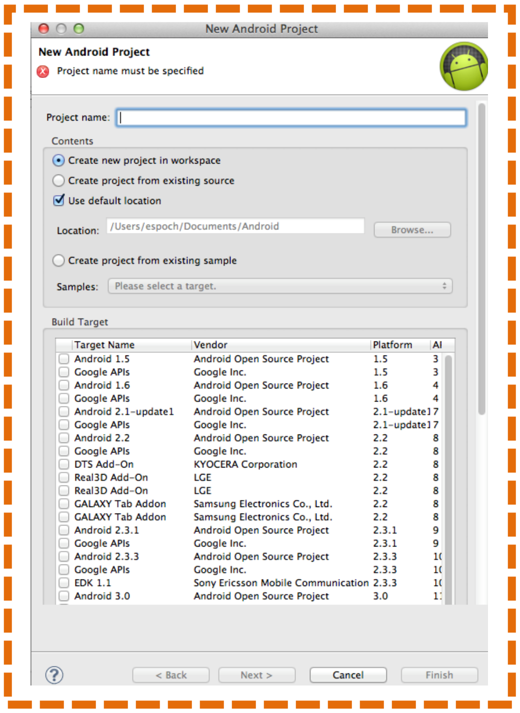




Android

The Eclipse IDE with the Android Plug-in is used to create the first Android project.





New Android Project

New Android Project

Project name must be specified

Project name:

Contents

(●) Create new project in workspace

( ) Create project from existing source

[✓] Use default location

Location: /Users/espoch/Documents/Android Browse...

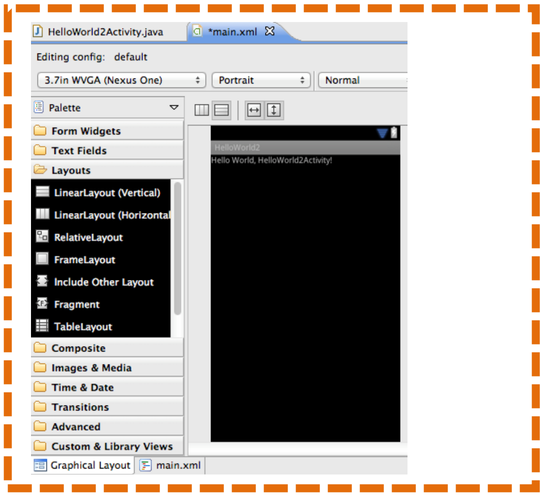
( ) Create project from existing sample

Samples: Please select a target.

Build Target

Target Name	Vendor	Platform	AI
Android 1.5	Android Open Source Project	1.5	3
Google APIs	Google Inc.	1.5	3
Android 1.6	Android Open Source Project	1.6	4
Google APIs	Google Inc.	1.6	4
Android 2.1-update1	Android Open Source Project	2.1-update1	7
Google APIs	Google Inc.	2.1-update1	7
Android 2.2	Android Open Source Project	2.2	8
Google APIs	Google Inc.	2.2	8
DTS Add-On	KYOCERA Corporation	2.2	8
Real3D Add-On	LGE	2.2	8
Real3D Add-On	LGE	2.2	8
GALAXY Tab Addon	Samsung Electronics Co., Ltd.	2.2	8
GALAXY Tab Addon	Samsung Electronics Co., Ltd.	2.2	8
Android 2.3.1	Android Open Source Project	2.3.1	9
Google APIs	Google Inc.	2.3.1	9
Android 2.3.3	Android Open Source Project	2.3.3	1(
Google APIs	Google Inc.	2.3.3	1(
EDK 1.1	Sony Ericsson Mobile Communication	2.3.3	1(
Android 3.0	Android Open Source Project	3.0	1:

< Back Next > Cancel Finish



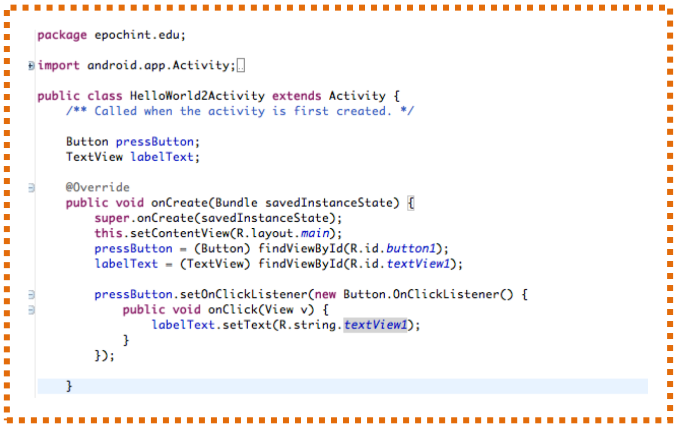
HelloWorld2Activity.java *main.xml

Editing config: default

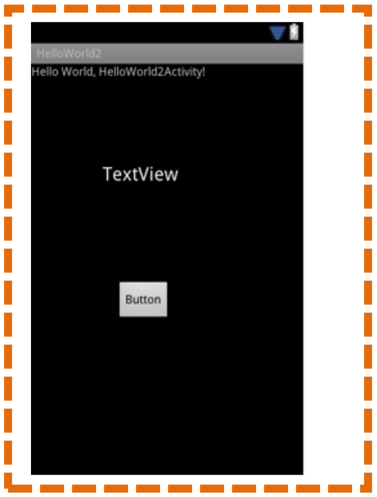
3.7in WVGA (Nexus One) Portrait Normal

Hello World, HelloWorld2Activity!

The application will contain one screen to which the user will interact with for that reason the application will be modeled with only one activity, which will have an UI, associated to it. By default, the creation of the project will create the class main.java, this class is characterize by extending from the Activity class defined on the Android API. In this case, it is necessary to override the onCreate() method in order to link the activity whit the UI that need to be display when the user will click on the application. First, the creation of the screen will be presented followed by the code need it to link the UI to the main application.

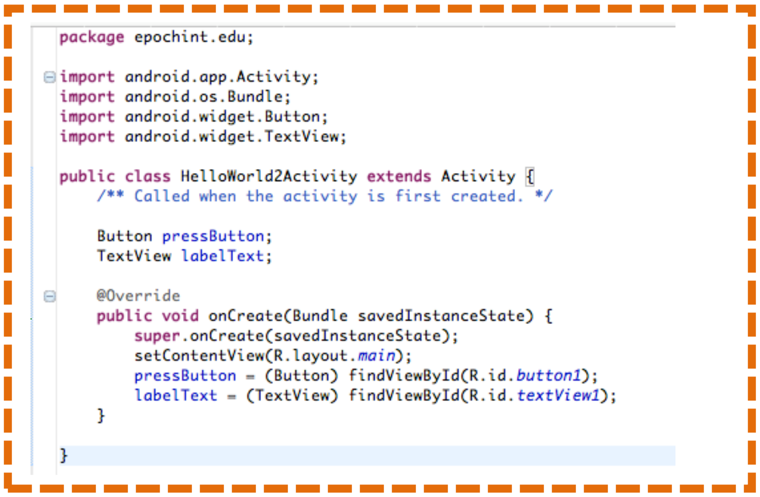


```
package epochint.edu;

import android.app.Activity;

public class HelloWorld2Activity extends Activity {
    /** Called when the activity is first created. */

    Button pressButton;
    TextView labelText;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.setContentView(R.layout.main);
        pressButton = (Button) findViewById(R.id.button1);
        labelText = (TextView) findViewById(R.id.textView1);

        pressButton.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                labelText.setText(R.string.textView1);
            }
        });

    }
```

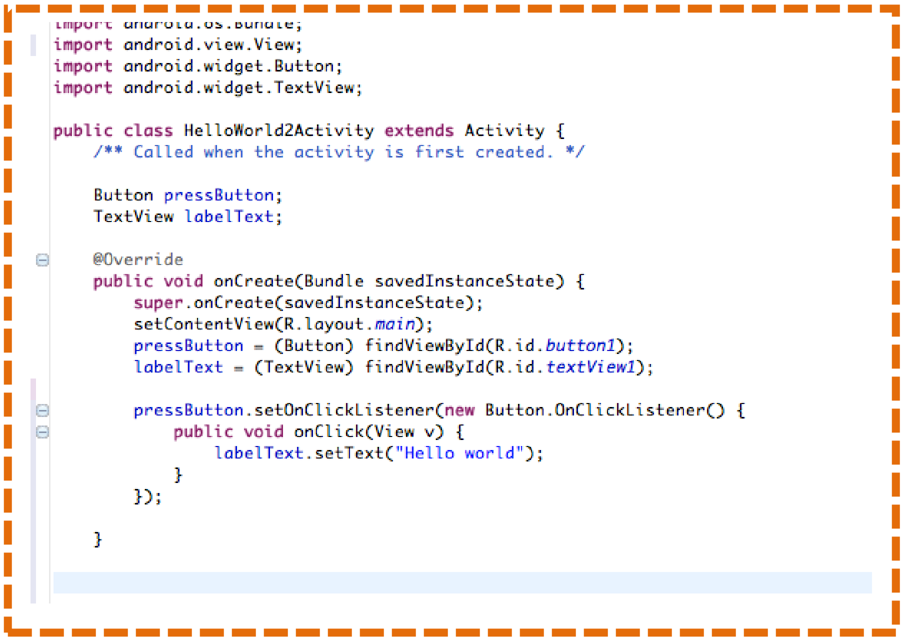


The next step is to link the UI elements with the code, two things need to be done. First, register the button and label on the source code (java class) so it can be set. Secondly, an event needs to be associated with the button so when the user pressed it an action is triggered in order to receive a response. The link between the UI elements and the code will be done using the id that Android generates to them and the method findViewById and the set of the activity interface by the method setContentView(R.layout.*main*);.

The next step is to define the control over the button in order to handle the event generated by the user pressing it, this is done on the code part by the creation of an inner listener class and the override of the onClick() method.

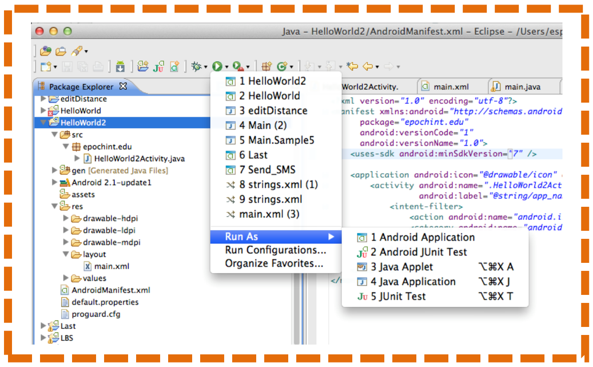


```
package epochint.edu;

import android.app.Activity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.TextView;

public class HelloWorld2Activity extends Activity {
    /** Called when the activity is first created. */

    Button pressButton;
    TextView labelText;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        pressButton = (Button) findViewById(R.id.button1);
        labelText = (TextView) findViewById(R.id.textView1);
    }

}
```

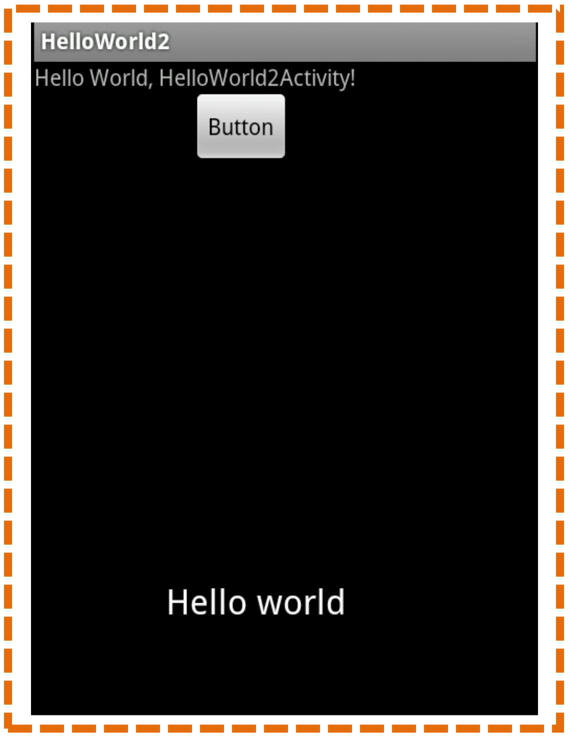


```
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class HelloWorld2Activity extends Activity {
    /** Called when the activity is first created. */

    Button pressButton;
    TextView labelText;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        pressButton = (Button) findViewById(R.id.button1);
        labelText = (TextView) findViewById(R.id.textView1);

        pressButton.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                labelText.setText("Hello world");
            }
        });

    }
```

Execution, it is perform on the emulator by selecting → run as → android application.



```
xml version="1.0" encoding="utf-8"?>
anifest xmlns:android="http://schemas.android
    package="epochint.edu"
    android:versionCode="1"
    android:versionName="1.0">
  <uses-sdk android:minSdkVersion="7" />

  <application android:icon="@drawable/icon"
      <activity android:name=".HelloWorld2Act
                android:label="@string/app_na
          <intent-filter>
              <action android:name="android.i
```



Application name: Internationalization

Application objective: Introduce how an application can be modify to implement location-awareness services.

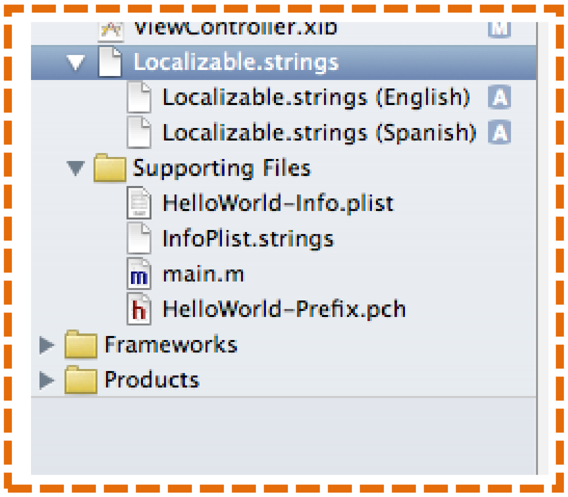
Type of application: Application that adapt to location configuration settings.

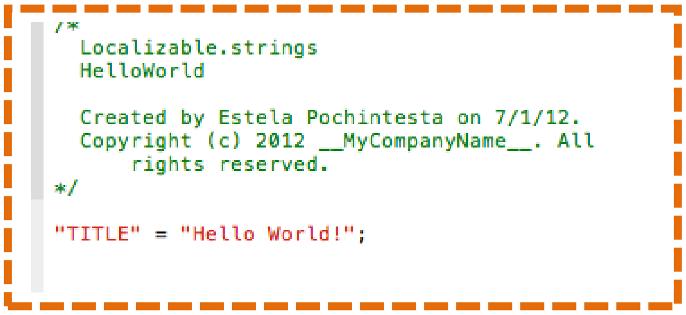
Components: One screen, strings resource locale for each language and internationalization of widgets.

iOS

The hello world application has strings which are hardcoded as the text on the button and the one that is set to the label, in order to localize them it is necessary to define and store them using string files for each language, iOS utilized the strings file format as key = content. In addition, the name of the application need to change according with the new locate, however this is done by the definition of another info.plist.strings for each language. **Three steps: localize the strings, localize the widgets, and localize the applications name.**

Localize strings: It is necessary to create a string file called Localization.strings and then on the right used the tool box to add the Spanish locale under the Localization tab. Afterwards two Localizable.strings files will be created under the name of: Lozalizable.strings(English) and Localizable.strings(Spanish).

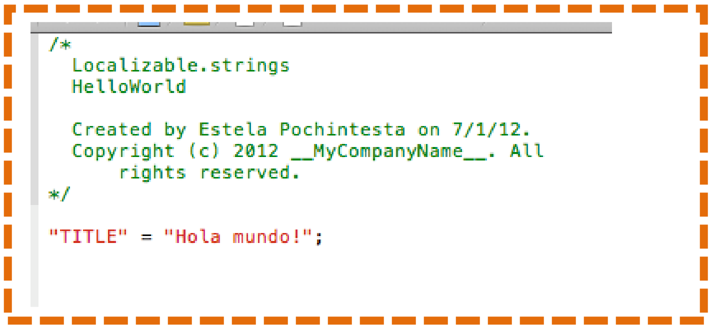




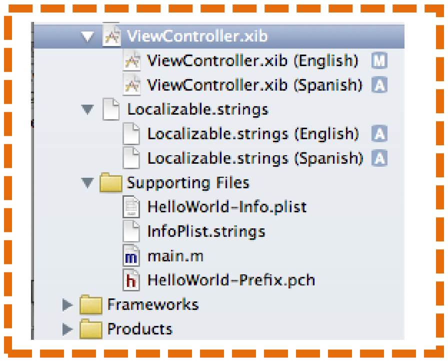
```
/*
  Localizable.strings
  HelloWorld

  Created by Estela Pochintesta on 7/1/12.
  Copyright (c) 2012 __MyCompanyName__. All
      rights reserved.
*/

"TITLE" = "Hello World!";
```

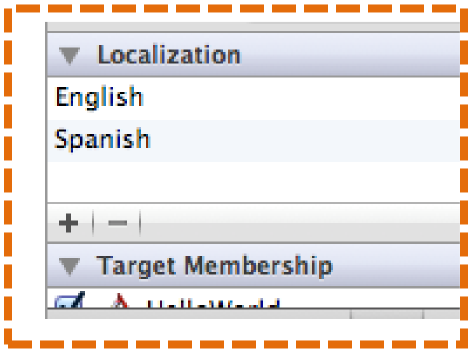
Localization of widgets

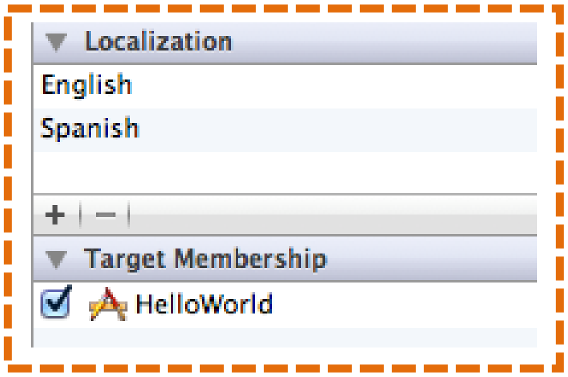
The localization of widgets is performing by doing a similar procedure with the files associated with the interface, which is the .nib file. The idea is to select the .nib file and add a new locale, which will auto generate two xib. Files one for each locale under the name of ViewController.xib(English) y ViewController.xib(Spanish).

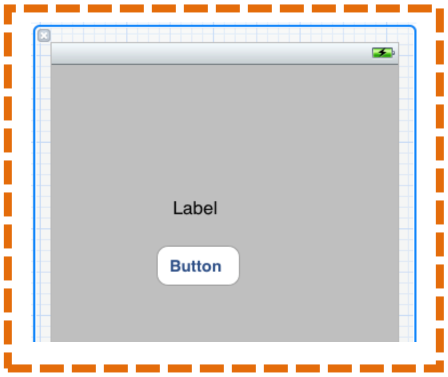


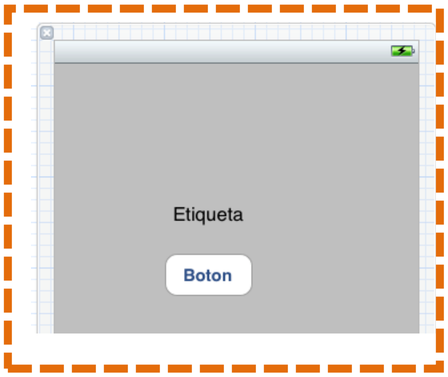
```
/*
  Localizable.strings
  HelloWorld

  Created by Estela Pochintesta on 7/1/12.
  Copyright (c) 2012 __MyCompanyName__. All
      rights reserved.
*/

"TITLE" = "Hola mundo!";
```


ViewController.xib
ViewController.xib (English) M
ViewController.xib (Spanish) A
Localizable.strings
Localizable.strings (English) A
Localizable.strings (Spanish) A
Supporting Files
HelloWorld-Info.plist
InfoPlist.strings
main.m
HelloWorld-Prefix.pch
Frameworks
Products



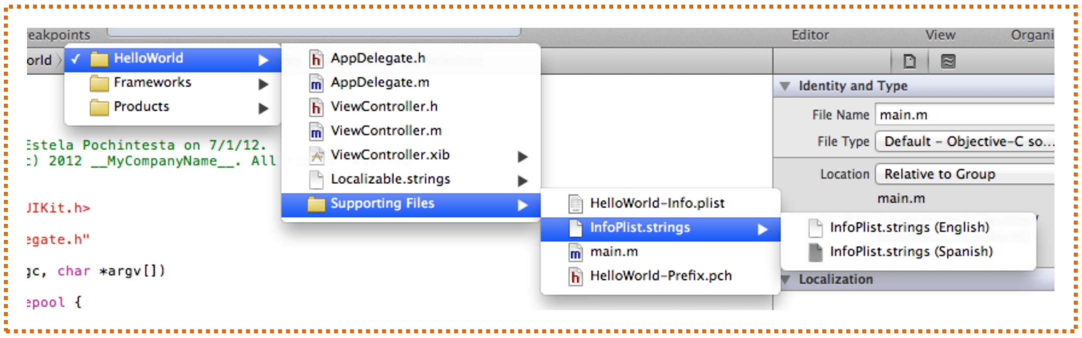




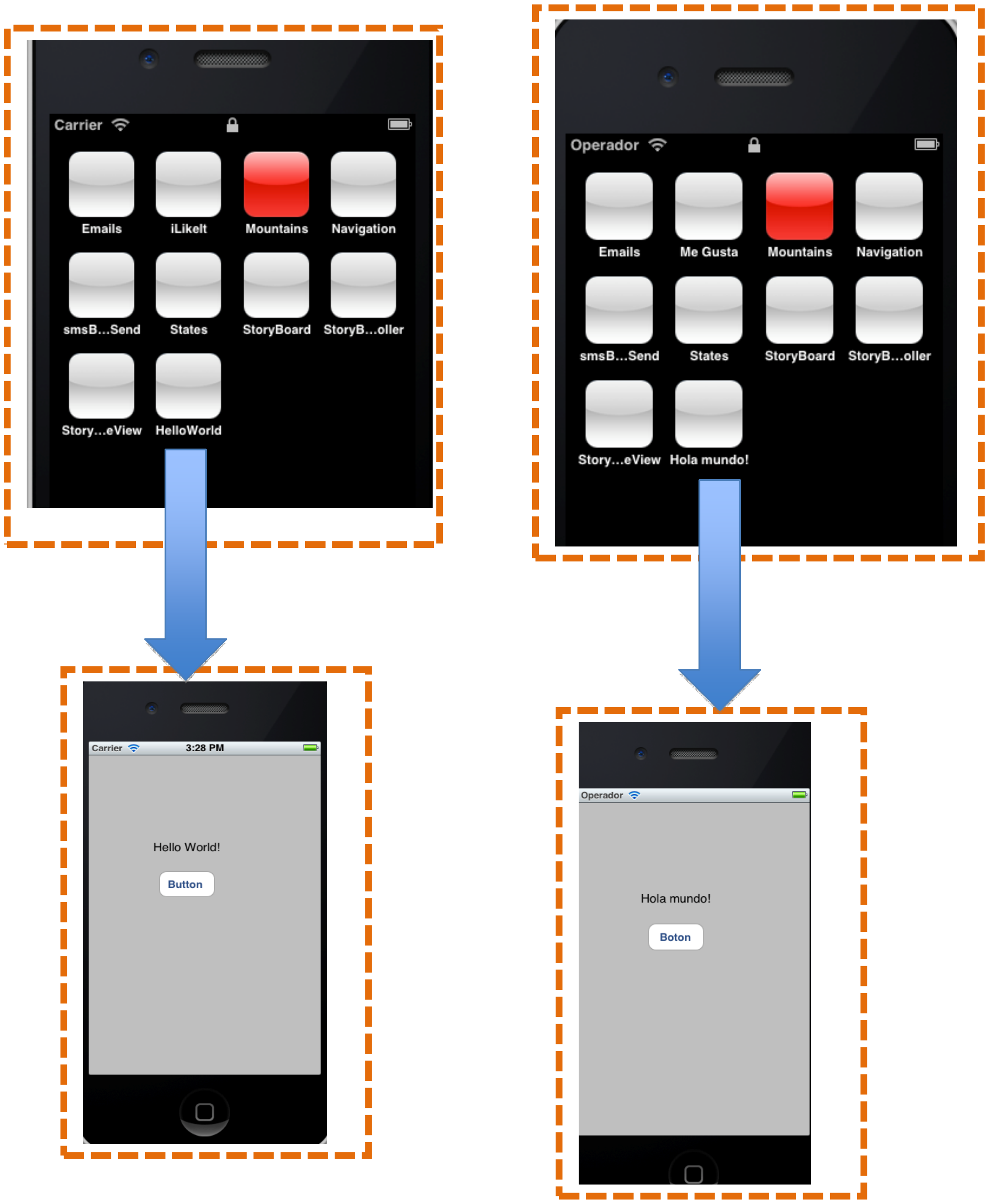


Localization of the bundle

The localization of the bundle is achieved by creating two different version of the file that contains the properties of the application *info.plist.*

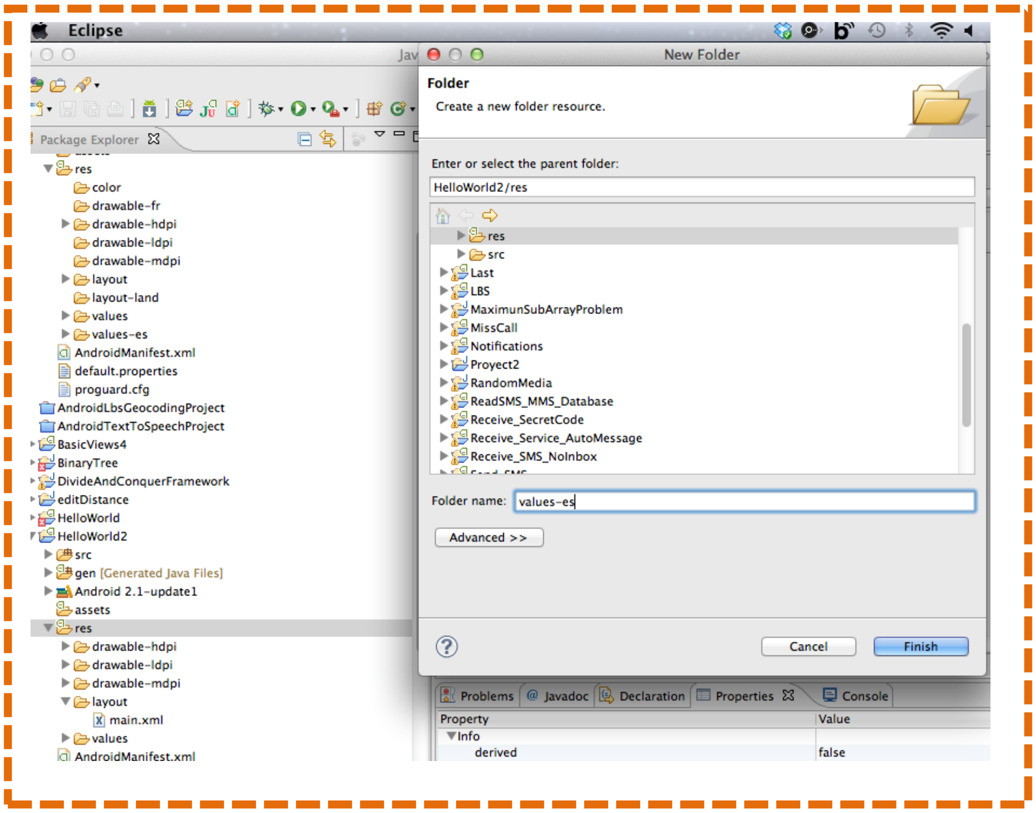


Execution


Carrier
Emails
iLikeIt
Mountains
Navigation
smsB...Send
States
StoryBoard
StoryB...oller
Story...eView
HelloWorld
Operador
Emails
Me Gusta
Mountains
Navigation
smsB...Send
States
StoryBoard
StoryB...oller
Story...eView
Hola mundo!
Carrier
3:28 PM
Hello World!
Button
Operador
Hola mundo!
Boton

Android

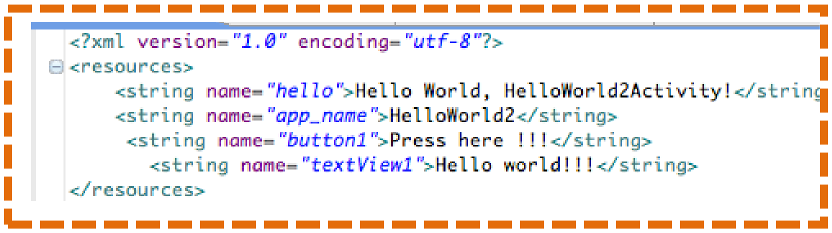
The internationalization is made externalizing all the hard code strings that the application contain to the resource string file on the project structure, each language supported will have a string file saved on a folder under resources named values- follow by the code language. For example, Spanish is value-es and English values. In this case there is no need to separate implement the localization of application name and widgets text everything is done in one place by externalizing on the string file by a key = value structure all the text.


Folder

Create a new folder resource.

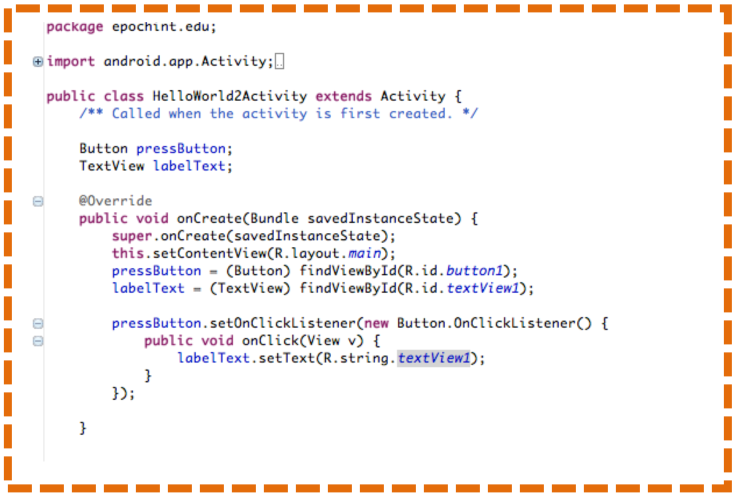
Enter or select the parent folder: HelloWorld2/res

Folder name: values-es

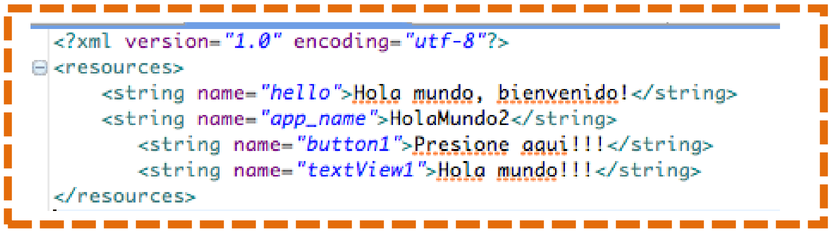


```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, HelloWorld2Activity!</string
    <string name="app_name">HelloWorld2</string>
     <string name="button1">Press here !!!</string>
       <string name="textView1">Hello world!!!</string>
</resources>
```

The last step is to externalize the string that was linked to the label so it can be set and adapt according with the locate, in the step before it was added as a textView1 string name, now it is going to change in the main class according to the following code:



```
package epochint.edu;

import android.app.Activity;

public class HelloWorld2Activity extends Activity {
    /** Called when the activity is first created. */

    Button pressButton;
    TextView labelText;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.setContentView(R.layout.main);
        pressButton = (Button) findViewById(R.id.button1);
        labelText = (TextView) findViewById(R.id.textView1);

        pressButton.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                labelText.setText(R.string.textView1);
            }
        });

    }
```



```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hola mundo, bienvenido!</string>
    <string name="app_name">HolaMundo2</string>
       <string name="button1">Presione aqui!!!</string>
       <string name="textView1">Hola mundo!!!</string>
</resources>
```

Application name: Navigation between activities

Application objective: Introduce the creation and management of an application with more than one activity.

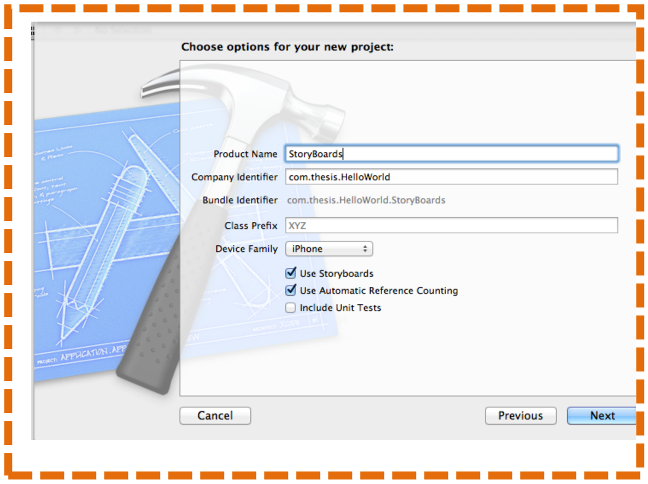
Type of application: Application that transition between screens based on user input.

Components: Two screens, navigation elements.

iOS

In particular, iOS4.1 introduces a new element called storyboards, which describes the transitions between the various screens. In order for the application to include the storyboard it need to be selected on the first screen of the creation project sequence.

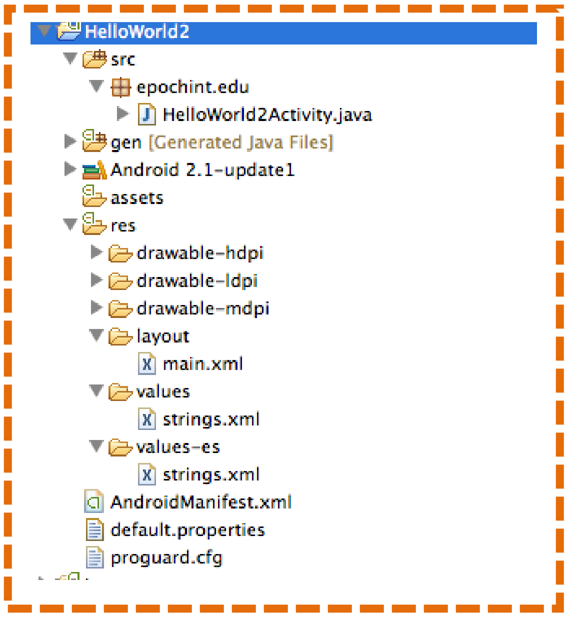
It created the following project structure



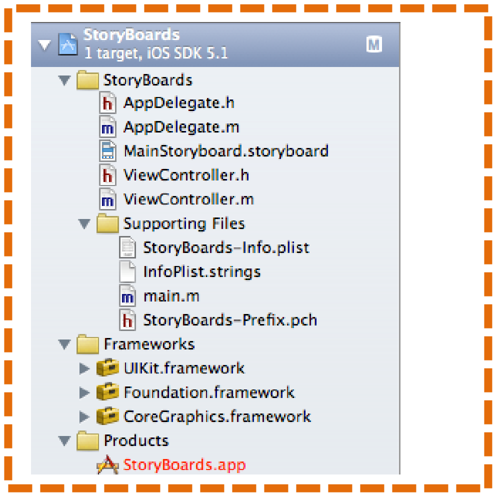
Choose options for your new project:

Product Name: StoryBoards
Company Identifier: com.thesis.HelloWorld
Bundle Identifier: com.thesis.HelloWorld.StoryBoards
Class Prefix: XYZ
Device Family: iPhone

☑ Use Storyboards
☑ Use Automatic Reference Counting
☐ Include Unit Tests

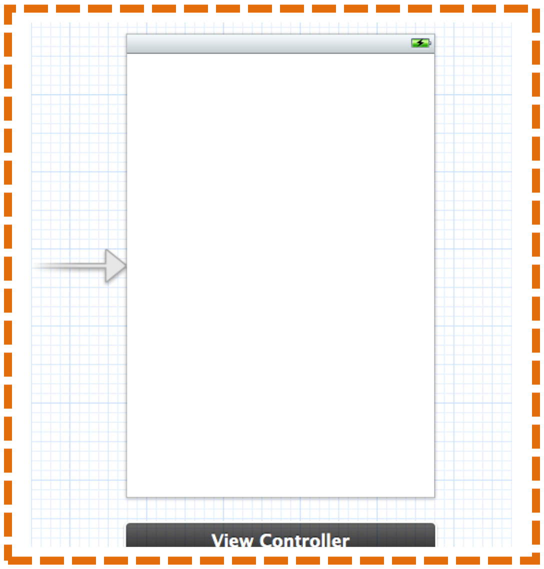


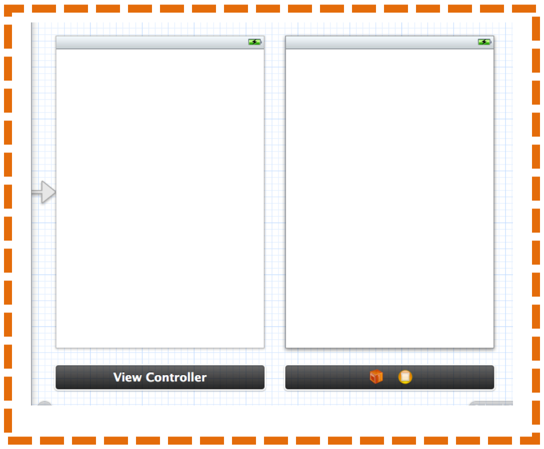
```
HelloWorld2
  src
    epochint.edu
      HelloWorld2Activity.java
  gen [Generated Java Files]
  Android 2.1-update1
  assets
  res
    drawable-hdpi
    drawable-ldpi
    drawable-mdpi
    layout
      main.xml
    values
      strings.xml
    values-es
      strings.xml
  AndroidManifest.xml
  default.properties
  proguard.cfg
```


StoryBoards
1 target, iOS SDK 5.1
StoryBoards
AppDelegate.h
AppDelegate.m
MainStoryboard.storyboard
ViewController.h
ViewController.m
Supporting Files
StoryBoards-Info.plist
InfoPlist.strings
main.m
StoryBoards-Prefix.pch
Frameworks
UIKit.framework
Foundation.framework
CoreGraphics.framework
Products
StoryBoards.app

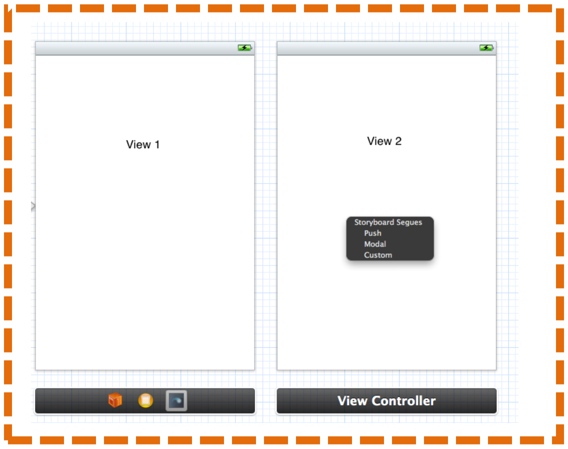
It is relevant to mention that there is no xib file, although a .storyboard file was created, this is because the storyboard will allocate not only one view controller for the application but all of them.

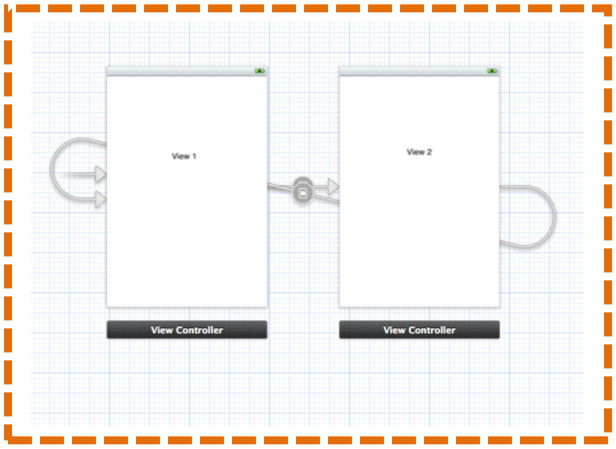
Once the project is created, the .storyboard file allocates a point by an arrow, which described the first screen with which the user will interact, since this example aim to show the transition between the screens, the next step is to add another screen/controller and associated their transition with a "Swipe Gesture Motion."





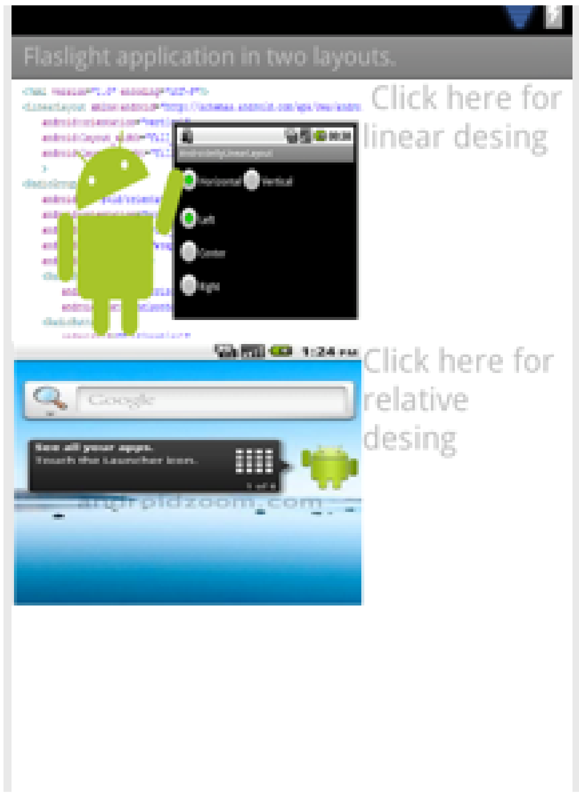
The association of the gesture with the second view it is perform by pressing the “control” key and daring the arrow to the second view.

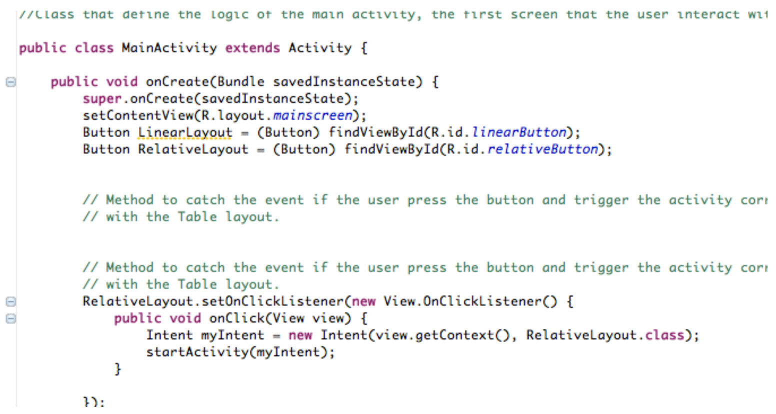




Android

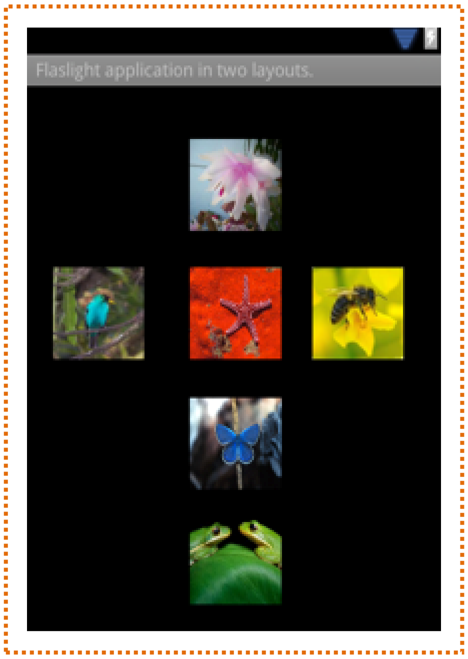
The navigation between two different screens on the same application is perform by using the intent object.


Flaslight application in two layouts.
Click here for linear desing
Click here for relative desing



```
//Class that define the logic of the main activity, the first screen that the user interact wi

public class MainActivity extends Activity {

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.mainscreen);
        Button LinearLayout = (Button) findViewById(R.id.linearButton);
        Button RelativeLayout = (Button) findViewById(R.id.relativeButton);


        // Method to catch the event if the user press the button and trigger the activity cor
        // with the Table layout.


        // Method to catch the event if the user press the button and trigger the activity cor
        // with the Table layout.
        RelativeLayout.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                Intent myIntent = new Intent(view.getContext(), RelativeLayout.class);
                startActivity(myIntent);
            }

        });
```

The navigation between the two screens (activities) is archived programmatically, by creating the intent object and the method *startActivity()*



As a closing remark of chapter IV, it is relevant to point out the commonalities and main differences between each platform. This will be useful for developers who are already familiar with one platform to familiarize them with the commonalities and difference of both mobile platforms. The main difference is related with the philosophy under which each mobile platform was built. Android is an open source platform and iOS is tightly closed. While Android promotes peer production, community development, and improvement of the platform and framework customization, Apple is not interested in opening the platform.

From a developer's perspective, Android's openness provides more flexibility and the ability to use the operating system as a white box being able to create applications that modify the core functionality of the device and reuse components by others or the operating system it. In addition, Android does not require a specific set of development tools, operating system, or language. Moreover, given the openness if a third party application is rejected by Google Play it can still be publish using Amazon Market or even the developer web site. However, the openness also has disadvantages for the developer such as variety of hardware, which makes developers unable to optimize the application for the underlying hardware and fragmentation, that limit developers on using the latest SDK features.

The iOS philosophy on the other hand, does not allow any framework customization or communication with the operating system unless Apple decides. Developers are limited to use Apple IDE Xcode on an Apple hardware and software. In addition, iOS devices are able to run the iOS operating system and third party applications can only be distributed using Apple's online store. Moreover, if an application it is rejected from the Apple store it cannot be published anywhere else. The closed philosophy, comes along with the advantages for developers to optimize the software for the iPhone device and built universal applications that can easily adapt and run on iPad as well. Despite the openness of the platform and like iOS, Android isolates applications by assigning its own Sandbox. However, the openness of Android platform allows application to share data by defining permissions.

The commonalities of both platforms are related with development environment, programming paradigm and application architecture. The development environments are very similar both provide a well structure IDE and developer tools and instruments that allow developers to debug, emulate and measure performance metric of third party application. In addition, both platforms adhere to the concept of object-oriented programming and patterns, which are extensively use in the development of the applications using both platforms. Finally, both platforms support the creation of event driven applications. Although, iOS uses a pure MVC model it can also be achieved by Android but is not impose as the architecture of the applications. Finally, on a fine-grained look the diagram bellow summarized the commonalties and differences between both platforms in terms of features available to the developer. Figure 18 summarizes all this concepts.

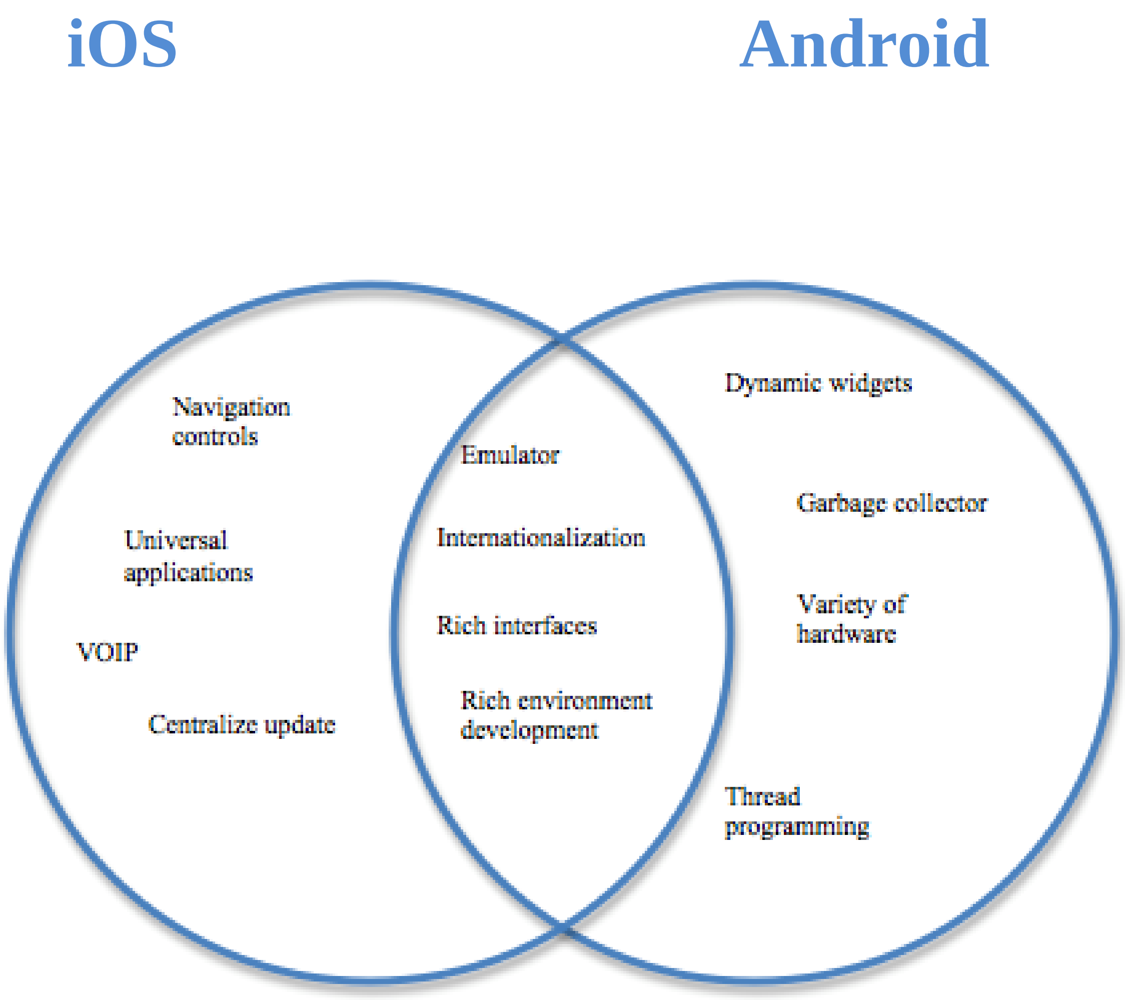


Figure 18. Androiod vs. iOS

CHAPTER V
CONCLUSIONS AND FUTURE WORK

After introducing the architecture of both platforms and presenting a theoretical and practical comparison it is relevant to conclude the research by detailing suitability of each platform for certain classes of applications and for the implementation of specific features. This is followed by a comparative square that aims to put the platforms in perspective. Afterwards, a set of recommendations, based on the experience of developing applications for both platforms, is given. Finally, a conclusion, future work, and closing remarks, are provided.

V.I MOST SUITABLE PLATFORM

Android and iOS mobile platforms allow the development of an extensive list of applications, including: e-readers, budgeting management applications, calculators, voice recorders etc. As a general rule, almost any kind of application can be equally implemented using the Android or iOS mobile platform, because both provide a structured and solid IDE and SDK, which includes APIs to implement complex functionality, debugging, coding and testing. However, both platforms do not equally offer tools, software and hardware feature. What makes one platform superior for the development of certain classes of applications. The difference between the platforms is influenced by several factors:

- Android is an open source platform, which utilizes Java as the development language.
- In addition, each application is represented as a combination of components that cooperate in order to create the application's functionality.
- iOS is a propietary platform that uses Objective C as the development language. In addition, each application represents a monolithic process modeled under the concepts of object oriented design patterns including MVC and application delegate.

In order to evaluate which platform is better to develop certain classes of application or features it is necessary to organize applications into classes. Although the literature reviewed does not outline a universal taxonomy, and Android and iOS online stores use their own categorization, this research used the taxonomy outlined by Nickerson, *et al* (Nickerson 2009). It is relevant to mention that, for this research, nine categories were included because they are related to the main challenges that mobile platforms face (power management, computational power, security, small screen). These categories are: mobile communications, mobile messaging, purchasing, location-based consent, product location and tracking for individuals, mobile actions and financial services, mobile games, mobile advertisement, mobile habitat/environmental monitoring and mobile social networking. The outline used to described each class consists of: category name and description, application examples (that each store offers), mobile platform that is believed to be more suitable for development and the rationale of the choice.

Category name: Mobile advertisement

Category description: Although a purely mobile application is not as common to create, the aim of developers to distribute free applications brought the advertisement business into mobile applications as embedded advertisements on third party applications, generating a source of revenue for developers.

Application examples: Angry Bird, Bubble.

Preferred mobile platform: iOS

Rationale: Android and iOS provide APIs that allow developers to define spaces where advertisements could be displayed within their applications. This feature provides a source of revenue by the monetization and promotion. Monetization refers to connecting the application with advertisers and showing relevant ads in the application, obtaining revenue when the users click on the ads (AdmobMonetizeAnd) . Promotion refers to the design and distribution of online campaigns through the mobile network (AdmobMonetizeAnd). Android uses AdMob and Google services while iOS has its own solution named iAd and supports AdMob solutions. Moreover, both platforms implement filters that allow the developer to discourage the advertisement engine from selecting ads based on URLs and text filters.

The main advantages, in the case of Android is the support of house ads, which allows developers to promote their applications for free on their own application space and incorporate custom search ads that Google offers. However, iOS offers a better monetization model by giving developers 70 percent of the iAd network revenue. Google, on the other hand, does not provide a fixed rate, but instead uses a variable rate that increases with the popularity of the application and decreases if a large amount of filters is defined. Finally, the platform chosen is iOS, since it supports Google AdMob and it has its own advertisement functions in SDK incorporated into Xcode, which does not require extra configuration on the IDE or APIs in order to enable development with iAds as Google does. In addition, iOS allows instruments and tools to generate content that will be advertised on iAd, which lets users, click on the applications banner and download an application instantaneously without going through the Apple store.

Category: Mobile Games

Category description: This category groups the creation of applications that represent games and can be played individually or in groups

Application examples: Angry birds or Pacman.

Preferred mobile Platform: iOS

Rationale: Android and iOS support Open GL (Open Graphics Library), which is a software interface to graphics hardware that allows the implementation of complex drawing in two and three dimensions and other complex drawing techniques. However, iOS is the most suitable platform because Android device's hardware varies in terms of CPU and screen resolution, which needs to be fixed, for hardware optimization. Moreover, iOS offers the game center feature to third party developers, which allows players to keep track of game scores and competing with friends in multiplayer games, by using the Gamekit framework, which uses an internal mechanism that also detects devices in proximity, running the same application.

Category: Location and Map navigation

Category description: This class includes applications with GPS capabilities, map navigation, tracking, and location-aware services.

Application examples: Tom Tom mobile solution.

Preferred mobile platform: Android for Map Navigation or continued tracking and iOS for location-aware based services.

Rationale: Android fully supports the Google Maps API, while iOS has a service agreement with Google, which is a front end client to the Google Maps web service that allow iOS developers to utilize certain features of Google Maps API. In addition, iOS provides the Mapkit framework, which allows developers to embed maps and manipulate of them. However, iOS does not support street view or navigation with speech directions. In this class both platforms are suitable for Location and Map navigation because Android provides a better platform for implementing applications that involve map navigation and directions, while iOS is a better platform for location-aware sevices. In iOS, Map Kit can be used with the Core Location framework in order to display locations and by using a combination of tower cellular, Wi-Fi, and GPS for calculating the user’s location according to the precision that the developer needs in the application.

Table 14 detailed each of them with the precision . This feature avoids measuring position periodically (like in Android) .

Source	**Accuracy**
Cellular tower	Within city or region
WI-FI	Within city block
GPS	Within a maximum distance of 5 meters.

Table 14. Capacity of network related peripherals

Category: M-commerce

Description: The M-commerce class of applications refers to applications that allow users to perform business transactions. This includes all e-commerce mobile version applications.

Application examples: EBay, Amazon, and Mint.

Preferred mobile platform: Android

Rationale: The strong implementation of two features determined the success of this class of applications: security and the transaction component commonly known as Near-field communications, which enables mobile payment. iOS offers hardware and software encryption by the implementation of the NSFileProtectionComplete framework, over the stored files, which are encrypted on the file system. This feature takes care of protection the information that need to be saved in the phone in order to perform m-commerce transactions. However, iPhone devices are not NFC compatible, which is why Android is the most suitable platform in this category, at this time since Android supports encription features as well.

Category: Multimedia applications

Category description: Multimedia includes a combination of text, audio, images, animation, video, or interactivity content forms**.**

Application examples: Brushes, Shazam, Movies.

Preferred mobile platform: iOS

Rationale: Android and iOS both provide APIs that allow developers to record and manipulate audio, photos, and video and create animations. Moreover, iOS provides integration with iTunes and support for AirPlay, which allows streaming video via WIFI to a TV.

Category: Environmental applications

Category description: This category includes applications that use the peripheral sensors that the mobile devices include in order to collect information from the environment to be processed in the application.

Application examples: Gain Span

Preferred mobile platform: Android and iOS

Rationale: Sensors in mobile phones allow applications to measure and analyze several environmental features, such as temperature, noise, and motion. Android provides an API that supports the following sensors: accelerometer, temperature, gravity, gyroscope, light, linear acceleration, magnetic field, orientation, pressure, proximity, relative humidity, rotation vector, and temperature. However, not all of them might be included on all Android devices and if they are, they are not disabled automatically when the application is not in use, which can affect the power consumption. On iOS, it is possible to define update intervals for each sensor. For that reason both platforms are suitable, to develop environmental applications since Android provides more variety and on iOS it is possible to turn them on and off automatically.

Category: Core functionality

Category description: This category includes the ability to create application that are an alternate set-up for the core applications that the device provides, such as home screen, wallpapers, and lock screen.

Application examples: Slide your phone (only supported if your phone is in jailbreak mode) for iPhone.

Preferred mobile platform: Android

Rationale: For certain purposes, the creation of an application that will change the core functionality of the phone is useful. On iOS 5, it is now possible to run the camera and audio reproduce from the lock screen. However, this API is not available to let developers include more features. Android, on the other hand, allows the creation of a third party application that will replace the default lock screen. Android is the most suitable platform since iOS does not provide this capacity or feature unless the iPhone is in jailbreak mode. In addition, the Apple store does not allow the inclusions of such applications into the store, citing the statement. “Apps that create alternate desktop/home screen environments or simulate multi-app widget experiences will be rejected” (Apple, 2011).

Category: Enterprise applications

Category description: This category represents the applications that companies develop to offer a mobile version of their current desktop application. These applications aim to connect to enterprise resources, increase productivity, or promote collaboration within the company.

Application examples: Delta, American Airlines.

Preferred mobile platform: iOS

Rationale: Generally, enterprise applications are characterized by presenting a complex data model that needs to be acquired using mobile application computing. iOS is superior given the architecture of each application, which allows separating the model implementation of the view and the controllers. Moreover, iOS offers a Core Data Framework (CDF), which provides the architecture and API to design and maintain the complex data models. In Android, there is no specific API to model complex data management because the data model of an application is represented as java objects. In addition, iOS’s CDF presents a migration tool to manage schema changes, which is very useful to an application that will adapt to business changes.

V.II FEATURES

In general, both platforms include the same basic features, although it might happen that one platform does not offer a feature or provides a superior implementation from the developer's perspective. Table 1 shows the summarized version of the main relevant features, distinguishing which features are better in one of the platforms or both. The first feature on Table 1 refers to the implementation of notifications, which is a resource that developers use to inform the user of changes or updates. Both platforms provide an API that allows the creation and placing of notifications on the home screen presented to the user. In addition to the notification center where notifications are grouped to present to the user, iOS also supports push notifications.

	Android	**iOS**	**Better support by**
Notifications	Does not support push up notifications.	□	iOS
Browsing	Chrome	Safari	Both
Internationalization	□	□	Both
Rich interfaces	□	□	iOS better controllers
Text-to-speech and speech recognition	□	□	Android
Social media integration	□	Twitter	Android
Dynamic content	Widgets, home screen	Push up notifications	Both
Usage of built in applications	Content providers, APIs to access functionality	Initiating calls with URLs	

Table 15. Summary of features implemented by Android and iOS

Internationalization, refers to the possibility of mobile platforms that allow the developer to internationalize the application. Localization, or internationalization, is the process of adapting the application's content to a specific region or language by adding a locale-specific component for each language. As was showed in the Internationalization example in chapter 4, there are two language changes on an application:

- Change of the language of the application.
- Change of the language of the application in order to adapt to system language changes.

Both mobile platforms support this feature by the implementation of different resource files that externalize the strings used in the application. More specifically, the different languages in Android have a code associated with it, and the string file should be presented for each language supported. For example, Italian translations would be located in *'res\values-it\strings.xml* and Spanish translation is *'res\values-en\strings.xml '*. In iOS, the feature is implemented by using the same concept, based on the translation of the text/strings and attaching them to the resource folder, so when the application loads the correct string resources. On iOS the different text/string, files are saved under the following directory structure: 'resources/*Localizable.strings(English) resources/Localizable.strings(French)'***.** In both mobile platforms, a change in the system language required the destruction and re-start of the application in order to adapt to new configurations and reloading of the alternative resources. On Android, the application will execute be sent to the *onDestroy()* method and on iOS the *applicationWillTerminate* method will execute when the application received a UIApplicationWillTerminateNotification. In both iOS and Android, the developer is responsible for saving and restoring the state of the application. Finally, iOS allows the internationalization of the nib file in order to localize the text on widgets, while in Android the widget text is retrieved from the strings file, as usual.

The text-to-speech and the speech recognition engine allow devices to incorporate a speech command based on interaction. Unfortunately, iOS presents the capabilities of both features, but they are reserved for use by the operating system only. However, Android provides the APIs that allow developers to implement text-to-speech and speech recognition capabilities. Moreover, in Android speech processing is to internationalization. The text-to-speech engine is supported in several languages, so it can be adapted to changes in the system locale.

The creation and layout of interfaces, must be designed ergonomically and attractively, and associated with complex controls. The UI components and layouts that both platforms provide are similar, including tables, sliders, pickers, text fields, checkboxes, web view, switches, and lists. The main difference, is related to how easy it is to design, layout and update the view hierarchy. In this aspect, iOS is the better of the two.

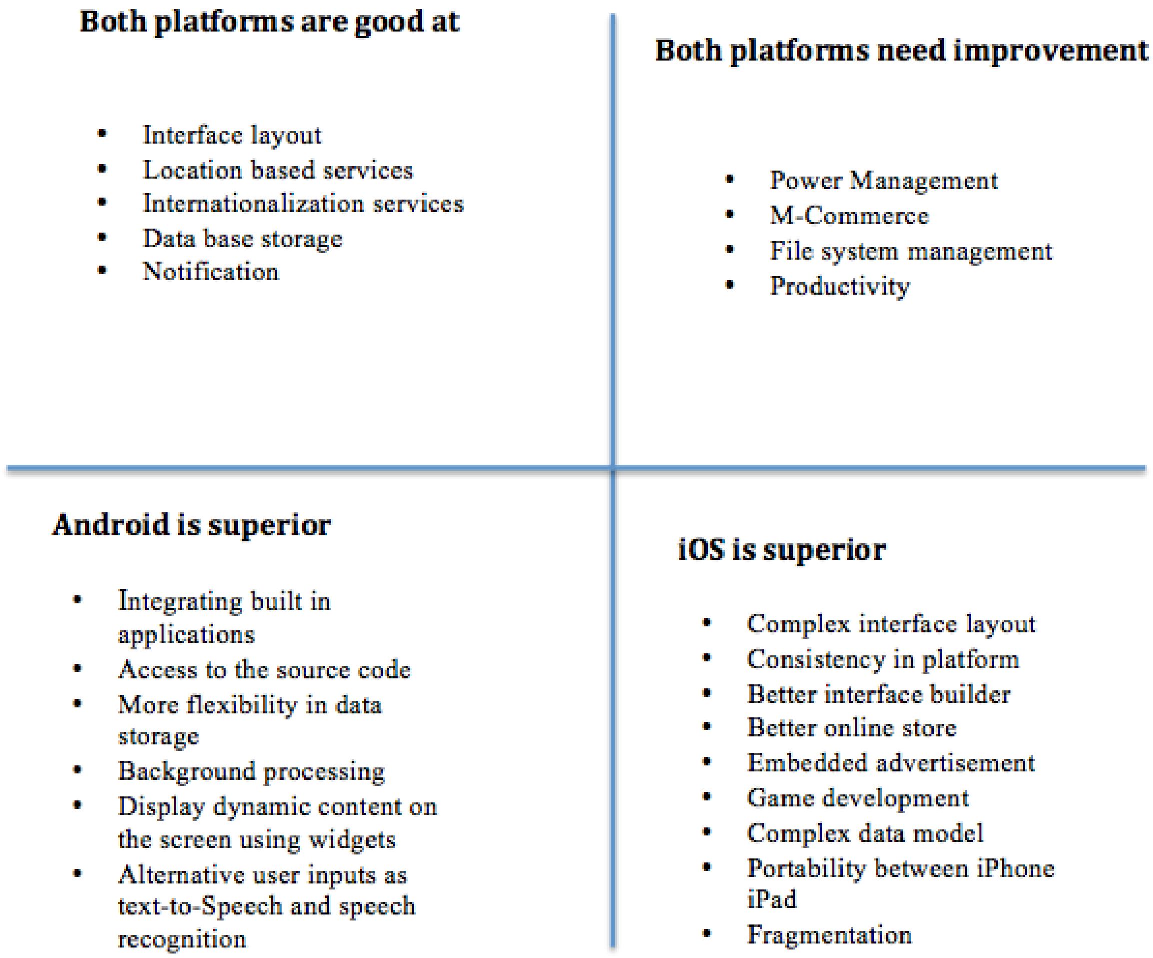
One final feature to consider is the possibility to add dynamic content to the screen through the customization of interactive widgets, which are simple controls that can be placed on the home screen to inform the user about important updates within the applications. It is common to have a weather or stock application as a widget. This is a handy feature for applications that need to have updated information constantly and instant access to the application without the need to open the application screen. Unfortunately, iOS does not allow the creation of third party widgets. However, Android has an API dedicated to the creation of widgets. It is relevant to know that iOS provides a dock that contains all the open application but does not allow users to place them in the home screen or showing information.

V.III COMPARATIVE SQUARE

The comparison square in Figure 19 provides a summary version of features where both platforms need improvement, both platforms provide a reasonable solution or one platform is better than the other Android tends to outperform iOS the development of applications that alter the environment of the home screen, wallpaper, or allow users to customize the appearance of the device. In addition, and continuing with the customization aspect of Android, it is also possible to use the built-in applications as components in third party applications and modify or extend their capabilities Moreover, it is possible expand the memory by using an SD card. Finally, Android is the superior platform regarding the implementation of multitasking and background processing by using specialized components/objects (services), that do not require a user interface and can be started and used from others components. They also can run longer than the activity that initiated them or associate a timer to the service.

On the other hand, iOS is superior concerning features related with fragmentation since iOS runs on an unique hardware set up, and the latest iPhone supports latest version of iOS, version 5. In addition, Xcode incorporates features that facilitate the creation of applications that are portable to the iPad device. Continuing with the Xcode advantages, it provides a powerful interface builder and instruments that allow developers to layout views, incorporate controllers and measure the performance of the applications. Finally, iOS includes the game center where users can keep up with their score and share it with friends, which, makes iOS superior to Android, since both support the same Open GL API.

To conclude, it is relevant to mention that both platforms include APIs and libraries to implement location-based services, data base storage, internationalization features and interface layout, in general. Moreover, each platform might vary on some extra functionality, but neither of them presents a clear advantage over the other one. Particularly, both platforms need improvement in power management, privacy, productivity, and security.



Both platforms are good at

- Interface layout
- Location based services
- Internationalization services
- Data base storage
- Notification

Both platforms need improvement

- Power Management
- M-Commerce
- File system management
- Productivity

Android is superior

- Integrating built in applications
- Access to the source code
- More flexibility in data storage
- Background processing
- Display dynamic content on the screen using widgets
- Alternative user inputs as text-to-Speech and speech recognition

iOS is superior

- Complex interface layout
- Consistency in platform
- Better interface builder
- Better online store
- Embedded advertisement
- Game development
- Complex data model
- Portability between iPhone iPad
- Fragmentation

Figure 19. Comparative Square

V.IV SUGGESTIONS

The suggestions are based on the research performed, which presents several factors that need to be considered before choosing the right mobile application platform. First, considering of which class of application is needed and whether the application will be possible submitted to the store. Second, given the fragmentation concerns with Android, developer must identify features that must be included to select the earliest SDK that supports the features. The third consideration is related to testing since the developer might not be able to acquire real devices and certain application features might not be emulator compatible. The fourth factor is distribution and monetization. If the application aims to be a source of revenue, adding advertisement or having a free version with a minimum set of features should be considered.

V.V FUTURE WORK

The future of mobile platforms it is very dynamic and in constant change. There is a lot of competition between the mobile platforms. Each new releases pushes the other to do better. The changes can be challenging for developers, but in the end, users are the beneficiaries of greater functionality. OS providers are iterating to add greater functionality so their mobile platform can be the leading one.

However, it is not possible to identify the leading mobile platform at any moment. In this research and given the constant growth of the mobile platform, the future work is described in terms of near and long term future.

The near feature is driven by the announcements of Google, Microsoft, and Apple reading new SDK versions. Android with Jelly Bean 4.1, Apple with iOS 6, and finally Microsoft with Windows Phone 8. The main improvements of each platform concentrate on providing richer interfaces, map and navigation system, m-commerce, and browsing. Specifically, within each area the mobile platforms announced:

- **Interface:** Android continues to improve the widgets to provide a better experience though it interaction with the user, iOS6 is strengthening Siri, which is not available to developers. Finally, Windows 8 adopted a Live Tiles system. It is basically a grid of square where each of them expose content to the user and allow third party developers to customize and update this content.
- **Maps:** Android is based on Google maps, supporting the pedestrian, and motorist with satellite navigation, where developers can include calendar suggestions about when to leave according to traffic. iOS6 has deprecated the use of Google maps services to introduce a new property system, without many updates regarding which Map APIs developers will have available in the future. Finally, Windows 8 uses the Nokia mapping services instead of Bing Maps, which worries developers with respect to support and optimization features of its APIs on non-Nokia hardware.
- **M-commerce:** Android is planning to bring back the Google wallet service only to the US territory. iOS6 introduces Passbook, which is not yet an NFC technology, although it provides storage and dynamic update of airline tickets, sports, and storage on cards, all in one centralized place.
- **Browsing:** An interesting number of web site provide a mobile version in order to allow

better mobile navigation and efficient and functional browsing technologies for developers to interact with the browsing APIs. In addition, web developers can enhance or optimize their mobile applications to mobile browsers. Android with Jelly Bean will include a version of Google Chrome, bringing opportunities of synchronization with Google and Google + accounts for developers. iOS6 will include offline reading and integration with the key cloud features for developers, as well the possibility to advertise their applications on web banners that will run on Safari. Windows 8 will include the latest version of Internet explorer. Finally, the idea of each platform to integrate their own browsing technologies might extend the developers capabilities allowing cross product development.

In addition, developers look for answers in the short run regarding fragmentation, mobile devices, and cross-platforms solutions, which are several ongoing issues . Regarding fragmentation Android is still trying to move users to at least Ice Cream Sandwich in order to reduce the fragmentation rate. Windows dropped the phone version of the SDK to build Windows 8 by creating a completely different API, which presents a disadvantage to developed applications for the phone. Related with mobile devices available, all the platforms offer phone and tablet versions Android faces adaptation between phone and tablet applications, iOS provides support to easily adapt the applications using Xcode and Windows will introduce this fall a tablet version named Surface. Finally, frameworks that will facilitate cross-platform development are introduced to the market such as Phone Gap and AppMobi. Cross-platform and web development present an advantage to developers a single language and technologies they are likely already familiar with will increase productivity. In addition, cross-platform and web development provide built in security solutions to potential security problems. Hybrid approaches between native and web applications appear to be promising for future mobile development efforts.

Android must continue to work on device compatibility and fragmentation issues. In order to compete, iOS must consider opening the speech-processing framework. With Microsoft introduction of a new renovated mobile operating system, and tablet this fall, all platforms will have a pressure to improve, especially in Microsoft strength of productivity solutions. Google wants to penetrate the productivity market with the recently acquisition of Quick Office, while iOS has its own productivity version of iWork, not currently supported on mobile phones.

Finally, no mobile platform seems to be superior among all areas. For that reason, during the next couple of years, each platform will try to mimic the strengths of the others and add new the features. Android has its own solution towards social networks and Microsoft towards productivity while Apple does provide unification among Apple devices that allow its synchronization across iTunes. The latest version suggests that there is still more improvement needed in the following areas: interface design and design improvements, alternative input interfaces, map and navigation services, and productivity capabilities. These improvements could produce a decrease on PC. Purchases according to a study performed by IDC predicts from the end of 2012 *et al ()* with a 2 to 1 rate.

The main challenges on mobile devices are clear. For example, power management solutions by energy profiling, utilization of multiple radios and effective transmissions techniques are needed. In addition, computing power solution that incorporate cloud services to reduce the necessity of high-end hardware and shift the computation and processing to the Cloud will likely be more relevant. Finally, the solution to smaller screens is the introduction of tablet devices and a unified platform ecosystem that will allow users to adapt and choose the most efficient device to display its content. As a summary, mobile platform providers aim to drive consumers into a platform ecosystem and not just a device. Motivated by Apple's success, companies and consumers all their devices (pc's, laptops, tablets and phones) running on the same system and displayed on whatever screen is the best. Providing a strong content platform delivered from the cloud. Supported by one of the top ten predictions of IDC (Information Development Consultants) *et al* (IDC, 2011) is that Big data, mobile and cloud will represent the combination necessary, given the volume of digital content growth to 2.7ZB(1ZB zettabyte = 1 billion terabytes = 10^{21}) in 2012, up 48% from 2011.

V.VI CLOSING REMARKS

As closing remarks, from the fact-based comparison, where technical features were considered, comes the result that Android is the leading platform with factors related to the openness and the application structure. In addition, the inclusion of an emulator and a plug-in to Eclipse ensures short curve learning. On the other hand, the disadvantages of the Android mobile platform are related to the platform fragmentation, several device manufacturers, and difficulty in creating rich user interfaces. The practical comparison section showed how easily iOS allows linking complex view objects to the screen compared with Android. Moreover, different manufactures use different user interface components, which does not ensure the best experience for the user interaction. Finally, the mobile applications future is promising with the introduction of new types of applications in the productivity area and m-business capabilities. With a third platform, Microsoft, having potential to gain significant market share, the race is on to develop even better software for mobile devices.

BIBLIOGRAPHY

AdMob -- monetize and promote your mobile apps with ads – Google Ads. Retrieved July 11, 2012, from http://www.google.com/ads/admob/

Android Developers. Retrieved July 11, 2012, from http://developer.android.com/index.html

2012, *Beginning iOS 5 development : exploring the IOS SDK*, Apress, US.

Asthana, A. & Asthana, R.G.S., 2012, IOS 5, Android 4.0 and Windows 8--A Review, *IEEE Code of Ethics*.

Chlamtac, I. & Redi, J., 1998, Mobile Computing: Challenges and Potential, *Encyclopedia of Computer Science*, 4.

Charland, A. & Leroux, B., 2011, Mobile application development, *Communications of the ACM*, 54(5), p. 49.

Computing Now Archive | November 2011: Challenges and Opportunities in Mobile Web and App Development. Retrieved July 11, 2012, from http://www.computer.org/portal/web/computingnow/archive/november2011

Davi, L., Dmitrienko, A., Sadeghi, A.R. & Winandy, M., 2011, Privilege escalation attacks on Android, *Information Security*, pp. 346-60.

Egele, M., Kruegel, C., Kirda, E. & Vigna, G., 2011, Proceedings of the Network and Distributed System Security Symposium, *PiOS: Detecting privacy leaks in iOS applications.*.

Less Than 1 Percent of Android Devices Run Latest OS - www.enterprisemobiletoday.com. Retrieved July 12, 2012, from http://www.enterprisemobiletoday.com/features/hardware/less-than-1-percent-of-android-devices-run-latest-os.html

File-Extensions.org - File extension library. Retrieved July 11, 2012, from http://www.filextensions.org/

Goadrich, M.H. & Rogers, M.P., 2011, Proceedings of the 42nd ACM technical symposium on Computer science education, *Smart smartphone development: iOS versus Android.* pp. 607-12.

Grønli, T.M., Hansen, J. & Ghinea, G., 2010, Proceedings of the 3rd International Conference on PErvasive Technologies Related to Assistive Environments, *Android vs Windows Mobile vs Java ME: a comparative study of mobile development environments.* p.45.

Hall, S.P. & Anderson, E., 2009, Operating systems for mobile computing, *Journal of*

Computing Sciences in Colleges, 25(2), pp. 64-71.

Hartmann, G., Stead, G. & DeGani, A., 2011, Cross-platform mobile development, Holzer, A. & Ondrus, J., 2009, Mobile Wireless Middleware, Operating Systems, and Applications-Workshops, *Trends in mobile application development.* pp. 55-64.

Holzer, A. & Ondrus, J., 2011, Mobile application market: A developer's perspective, *Telematics and Informatics*, 28(1), pp. 22-31.

iOS Dev Center - Apple Developer. Retrieved July 11, 2012, from https://developer.apple.com/devcenter/ios/index.action

Leader, M.W.T., Comprehensive Analysis of SmartPhone OS Capabilities and Performance, .Jahns Ralf-Gordon. Smartphone Application Market To Reach US$15.65 Billion In 2013. Retrieved June 11, 2012, http://www.research2guidance.com/smartphone-application-market-to-reach-us15-65-billion-in-2013/

Lee, W.-M., 2011, *Beginning Android application development*, Wiley Pub, Indianapolis, IN.

Lee, W.-M., 2012, *Beginning ios 5 application development*, Wiley Pub., Inc, Indianapolis, IN.

Mary Meeker Matt Murphy, 2011. TOP MOBILE INTERNET TRENDS
Cromar, 2011. Smartphones in the U.S.: Market Analysis

Nickerson, R., Muntermann, J., Varshney, U. & Isaac, H., 2009, Taxonomy development in information systems: developing a taxonomy of mobile applications, .

Oliver, E., 2009, A survey of platforms for mobile networks research, *ACM SIGMOBILE Mobile Computing and Communications Review*, 12(4), pp. 56-6

Oliver, E., 2009, A survey of platforms for mobile networks research, *SIGMOBILE Mob. Comput. Commun. Rev.*, 12(4), pp. 56-63

Pocatilu, P., 2011, Android Applications Security, *Informatica Economică*, 15(3), pp. 163-71.

Puder, A., 2010, Proceedings of the 8th International Conference on the Principles and Practice of Programming in Java, *Cross-compiling Android applications to the iPhone.* pp. 69-77.

Rani, C.R., Kumar, A.P., Adarsh, D., Mohan, K.K. & Kiran, K.V., 1963, LOCATION BASED SERVICES IN ANDROID.

Smyth, N., 2011, IPhone iOS 5 development essentials, s.n.], S.l. Steele, J. & To, N., 2011

The Android developer's cookbook : building applications with the Android SDK, Addison-Wesley, Upper Saddle River, NJ.

Wasserman, T., 2010, Software engineering issues for mobile application development, FoSER 2010.

VITA

Born in 1986, Montevideo Uruguay. Estela Pochintesta attend High School and College in Montevideo. She graduate from her Science bachelors at Catholic University in December of 2009. Afterwards, start her Master in Science at the University of Mississippi in Spring 2011. Estela it is a member of the Upsilon Pi Epsilon chapter at the University of Mississippi. Her hobbies are cooking, traveling, and reading.