CAPÍTULO 1

Hardware y Software

1.1. Introducción

Introducción

1.2. Software de procesamiento de imágenes

Software de procesamiento de imágenes

1.2.1. Lenguaje C

Ventajas del Lenguaje C para procesamiento de imágenes

1.2.2. Librerías y recursos

1.2.2.1. OpenCV

1.2.2.2. IPOL

1.2.2.3. ITK e ImageMagik

1.3. Elección de plataforma

La elección de la plataforma para desarrollar fue una de las primeras desiciones que se tuvo que hacer más allá del proceso natural inicial de investigación del proyecto. Esto es debido a que existen muchos factores que se ven afectados en función de la plataforma que se eligiera que van desde aprendizaje de lenguaje, entorno de desarrollo hasta la adquisición de plataformas y/o máquinas para desarrollar. Así entonces las dos grandes posibilidades que se tenían al comienzo y que determinaban estos factores era la elección de trabajar en platformas que tuvieran los sistemas operativos Android o iOS (en ningún momento se consideró desarrollar sobre plataformas con Symbian dado que está en proceso de desaparición). Uno de los aspectos desfavorables que se veía sobre Android era la multiplicidad de plataformas existentes, de distintas características de procesamiento, cámara y sensores entre otras cosas con respecto a las plataformas que utilizan iOS. Por otra parte luego de un proceso de investigación se vio que el conjunto de herramientas existentes y el estado de maduración del desarrollo de aplicaciones para iOS hacían de iOS, un camino más seguro para comenzar en un área que era desconocida. Es sabido que Android es un sistema operativo que viene en pleno

crecimiento y que a nivel masivo sea una buena alternativa desarrollar en él, pero para los fines de investigación y de desarrollo de sotware que tenía el presente proyecto se vio que era mejor desarrollar sobre iOS y plataformas Apple.

Al trabajar con Apple entonces se cuenta con la ventaja de contar con pocas variantes en cuanto al Hardware utilizado. Básicamente existen tres familias de dispositivos en los que se puede desarrollar: iPhone, iPad y iPod Touch. Para cada variante de plataforma existen distintos modelos que hacen que algunas características importantes como la capacidad de procesamiento, la resolución de cámara o el tamaño de la pantalla entre otras puedan verse afectadas. A continuación se presenta brevemente cómo fue el surgimiento de cada uno de los dispositivos al mercado y se describen resumidamente las principales características.

1.3.1. iPhone, iPad, iPod Touch

1.3.1.1. Comparación de plataformas.

Sin dudas el iPhone fue uno de los saltos más grandes en el mundo tecnológico en los últimos años. Logró llenar el hueco que los PDAs de la década de los 90 no habían sabido completar y comenzó a desplazar al invento que revolucionó el mercado de los contenidos de música, el iPod. Gracias a su pantalla táctil capacitiva de alta sensibilidad logró reunir todas las funcionalidades agregando solamente un gran botón y algunos extra para controlar volumen o desbloquar el dispositivo.
La primera generación del iPhone fue lanzada por Apple en Junio de 2007 en Estados Unidos, luego de una gran inversión de la operadora AT&T que exigía exclusividad de venta dentro de dicho país durante los siguientes cuatro años. La misma soportaba tecnología GSM cuatribanda y se lanzó en dos variantes de 4GB y 8GB de ROM. El segundo modelo lanzó como novedad el soporte de tecnología 3G cuatribanda y GPS asistido. Luego le siguieron el iPhone 3GS, 4, 4S y el 5, siendo este último, la sexta y última generación disponible al momento de la redacción de este trabajo.

Por su parte tanto el iPad como el iPod Touch también representaron un gran salto en el mundo de las plataformas y *Tablets*, agrandando las posibilidades de desarrollo y procesamiento. Como se dijo, de cada una de estas tres familias de dispositivos existen distintas versiones y modelos. Por eso, a continuación se muestra una tabla comparativa de determinadas características que son de interés a los efectos del presente proyecto.

1.3.1.2. Algunas características a detallar.

Hay algunos comentarios respecto de la Tabla 1.1 que es bueno destacar. Primeramente, es importante decir que se eligieron esos cuatro dispositivos pues pareció de interés conocer al menos una plataforma de cada familia y dentro de las mismas se eligieron las que fueron utilizadas para desarrollar.
Uno de los puntos a evaluar es el **SoC**, que refiere a *System on Chip* por sus siglas en inglés. *System on Chip* es un concepto de los sistemas embebidos que refiere a la integración de todo lo necesario para poder correr un sistema operativo, en un solo circuito integrado. En contraposición a un microcontrolador que es capaz de realizar procesamiento más básico y menos potente, con poca interacción de usuario y menor flexibilidad, un *SoC* refiere a la idea de tener todo lo necesesario para desarrollar sobre la plataforma y poder hacer procesamiento sin tantas limitaciones. Básicamente cumplen funciones similares pero el *SoC* forma parte de una evolución de los microcontroladores, siendo de una complejidad mayor e integrado en un tamaño muy reducido buscando poco consumo

Tabla 1.1: Comparativa de algunas plataformas Apple

	iPhone 4	**iPhone 4s**	**iPod Touch 4G**	**iPad 2**
ROM	8, 16 o 32 GB	16, 32 o 64 GB	8, 32 o 64 GB	16, 32 o 64 GB
RAM	512 MB	512 MB	256 MB	512 MB
SoC	Apple A4	Apple A5	Apple A4	Apple A5
CPU	1 GHz, ARM Cortex-A8	1 GHz, dual-core ARM Cortex-A9	800 MHz, ARM Cortex-A8	1 GHz dual-core ARM Cortex-A9
GPU	PowerVR SGX535 GPU	PowerVR SGX543MP2 (2-core) GPU	PowerVR SGX535 GPU	PowerVR SGX543MP2 (2-core) GPU
CÁMARA	Foto: 5.0 MP Video: 720p HD (30 fps)	Foto: 8.0 MP Video: 1080p HD (30 fps)	Foto: 0.7 MP Video: 720p HD (30 fps)	Foto: 0.7 MP Video: 720p HD (30 fps)
PANTALLA	Diagonal: 3.5” Pixels: 960x640 Densidad de Pixels: 326 ppi Multitáctil	Diagonal: 3.5” Pixels: 960x640 Densidad de Pixels: 326 ppi Multitáctil	Diagonal: 3.5” Pixels: 960x640 Densidad de Pixels: 326 ppi Multitáctil	Diagonal: 9.7” Pixels: 1024x768 Densidad de Pixels: 123 ppi Multitáctil
SENSORES	Girsóscopo de 3 ejes Acelerómetro Sensor de luz ambiente Sensor de proximidad	Girsóscopo de 3 ejes Acelerómetro Sensor de luz ambiente Sensor de proximidad	Girsóscopo de 3 ejes Acelerómetro Sensor de luz ambiente	Girsóscopo de 3 ejes Acelerómetro Sensor de luz

y eficiencia de costos. Así entonces un *SoC* puede estar conformado por un microcontrolador y hardware adicional como procesadores de señal y bloques de memoria.
En la Figura 1.1 se ilustran los dos tipos de *SoC* de los dispositivos de la Tabla 1.1: Apple A4 y Apple A5. Algunos dispositivos que no figuran en la tabla como el *iPhone 5* o el *iPad 4* usan *SoCs* más recientes como el Apple A6 o Apple A6x respectivamente. La familia de *SoCs* Apple Ax, es la que la mencionada firma utiliza en todas sus plataformas, inclusive en el *Apple TV* y es manufacturada por Samsung. Estos *SoCs* se caracterizan por utilizar CPUs de arquitectura ARM, en su mayoría ARMv7 y GPUs de PowerVR de la línea SGX.

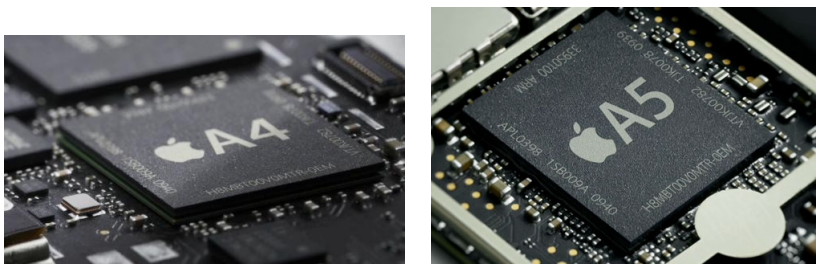

Figura 1.1: *System on Chip*: SoC.

La arquitectura ARM incorpora algunas características de la arquitectura RISC (*Reduced Instruction Set Computer*) como el hecho que las operaciones sean llevadas a cabo sobre un conjunto de registros a tales efectos y no sobre la memoria directamente. Otra característica que tiene de RISC es que tiene un modo simple de direccionamiento donde las direcciones son también guardadas sobre registros destinados a tales efectos. La mayoría de los procesadores están hechos con un ancho de palabra de 32-bit salvo el reciente ARMv8 que incorpora la posibilidad de utilizar ambos anchos de palabra: 32-bit o 64-bit. Otra característica a destacar sobre estos procesadores es que es posible programar sobre ellos utilizando lenguaje C/C++. Por más información sobre esta arquitectura referirse a la web de la empresa `http://www.arm.com`.

Por su parte las GPU utilizadas en los *SoCs* de la serie Apple Ax, son GPUs de PowerVR, una división de la firma Imagination Technologies (`http://www.imgtec.com/`) que desarrolla hardware y software para *rendering* 2D y 3D, procesamiento de imágenes y codificación. La función que tiene la GPU es asignar a cada pixel de la pantalla su color para cada cuadro. En particular, estas GPU implementan un concepto innovador de *renderizado* que mejora notoriamente la performance de los gráficos: *Tile-Bassed Deferred Rendering* (TBDR). Este concepto aprovecha la independencia de áreas alejadas de la pantalla y de la correlación de píxeles cercanos y divide la pantalla en *tiles* o baldozas. A cada *tile* se le asocia un procesamiento paralelo y con esto se mejora notablemente la performance respecto al método tradicional: Immediate mode renderers (IMRs), que procesa la pantalla completa. Las imágenes *renderizadas* están hechas por triángulos (polígonos), por lo que uno de los indicadores fundamentales para evaluar la performance de una GPU es la cantidad de triángulos (polígonos) que es capaz de procesar por segundo. En (referencia benchmark: http://www.anandtech.com/show/4216/apple-ipad-2-gpu-performance-explored-powervr-sgx543mp2-benchmarked) se puede ver un análsis interesante que compara entre otros, a los dos tipos de GPU que se presentaron en la Tabla 1.1: SGX535 y SGX543MP2. En dicho análisis se muestra por ejemplo que en el mejor de los casos la GPU SGX535 fue capaz de procesar 8.69 millones de triángulos por segundo frente a los 29 millones procesados por la GPU SGX543MP2.

1.4. Software de Apple Inc.

1.4.1. Sistemas Operativos

Para poder desarrollar aplicaciones sobre dispositivos móviles de la firma Apple Inc. es necesario contar con computadoras que corran el sistema operativo **Mac OS X**. Esto puede ser llevado a cabo, ya sea adquiriendo plataformas de desarrollo de la mencionada firma o creando máquinas virtuales que corran dicho sistema operativo. Para la segunda opción (la más económica pero con ciertas dificultades de performance), es necesario que la computadora cuente con virtualización de hardware. Se comenzó trabajando de esta manera hasta el momento de adquirir plataformas de desarrollo que contaran con Mac OS X en forma nativa.
Mac OS X refiere a la versión número 10 (en números romanos) de una serie de sistemas operativos que comenzaron a desarrollarse en la década de los 80 (Mac OS 1 data del a?o 1984). En los últimos 28 a?os se han ido sucediendo nuevas versiones que han ido mejorando características en la estructura de datos con la incorporación de la jerarquía de archivos en Mac OS 3 por ejemplo, en la búsqueda de archivos, con la simultaneidad de tareas, multiplicidad de usuarios o incluso con el énfasis en la interfaz de usuario por mencionar algunas características importantes en la evolución de esta familia de sistemas operativos. Dentro de Mac OS X existen distintas versiones, siendo la más actual la Versión 10.8: Mountain Lion lanzada durante 2012.
Por su parte todas las plataformas móviles de Apple Inc corren otro dispositivo de código cerrado: **iOS**. Originalmente llamado así por ser el sistema operativo utilizado por la plataforma iPhone, este sistema operativo está también en las plataformas iPad, iPod Touch y Apple TV en todas sus versiones. La versión más reciente de este SO es el iOS 6.1.
Una de las grandes innovaciones de estas plataformas es el hecho de poder desarrollar aplicaciones y correrlas en el propio dispositivo (por supuesto también sucede lo mismo en el mundo Android). Para poder lograr esto, es necesario como se ha dicho, contar con una máquina que corra Mac OS X y contar con el SDK apropiado llamado **Xcode**. Este entorno de desarrollo y su lenguaje se explican en la sección 1.4.2.

1.4.2. Objective-C

El lenguaje que fue elegido por Apple Inc para desarrollar sobre plataformas móviles es Objective-C. Este lenguaje fue desarrollado en la década de 1980 como un superconjunto de C orientado a objetos. Es decir que es una extensión del standard ANSI C que incorpora un modelo orientado a objetos basado en **Smalltalk**. Una de las diferencias sustanciales del modelo orientado a objetos de Objective-C respecto a otros lenguajes como Java o C++, es el hecho de la invocación de los métodos (procedimientos) de las instancias de clases. En objective-C esta invocación se da enviando mensajes, algo que se hereda de Smalltalk. Así entonces para invocar un método se procede con el siguiente código por ejemplo:

```
[receiver message];
```

Donde *receiver* es un objeto que recibe un mensaje (acción) *message* a realizar. Esta acción puede tener parámetros asociados, como por ejemplo el siguiente código real:

```
[myRectangle setWidth:20.0];
```

Esta diferencia conceptual de utilizar mensajes se representa en el hecho de que en tiempo de compilación estos mensajes no son más que una etiqueta y no están asociados al bloque de código como es el caso de Java o C++. Entonces es factible que suceda el hecho de que ese mensaje o método no esté implementado por esa clase y recién en tiempo de ejecución es que saltará el error al sustituirse esa etiqueta por un código inexistente, pues un objeto recibe un mensaje para realizar un método que no está dentro de su repertorio. Para esto es que en la documentación de Apple Inc se recomienda utilizar ciertos trucos para garantizar que el objeto que reciba el mensaje sea capaz de responder correctamente, como por ejemplo consultando primero si es capaz de realizar dicha acción y luego en caso de poder realizar dicha acción.
Otro detalle a destacar es que este lenguaje, al igual que Java también soporta la herencia múltiple. Esto es, dado un conjunto de métodos que son comunes a un conjunto de clases pero que no llegan a tener un lazo tan fuerte como para estar jerárquicamente relacionadas con una superclase común, se puede generar una clase abstracta cuyos métodos sean implementados por más de una clase sin necesidad de generar ese vínculo fuerte que es la herencia. Así como en Java existen las interfaces, que hacen esto posible, en Objective-C existen los protocolos. Existen protocolos formales e informales y con métodos obligatorios de implementar y otros opcionales. Una clase que implemente un protocolo dado tiene que tener dentro de su encabezado declarado el nombre del protocolo. Esto es:

```
@interface ClassName : ItsSuperclass < protocol list >
```

Hay otras particularidades del lenguaje pero que no van más allá de la sintaxis como los métodos de clase y los métodos de instancia, como los métodos *get* y *set* para acceder y setear atributos (propiedades) de los objetos, como la notación de *import* en lugar de *include* para quienes están acostumbrados a C y así varias detalles más. Sin embargo más allá de estas y otras diferencias y particularidades resulta un lenguaje relativamente ágil y dentro de todo sencillo de aprender para quien tiene ya un conocimiento de otros lenguajes orientados a objetos.

1.4.3. Xcode: Herramientas y Librerías

Como se dijo anteriormente el entorno de desarrollo de aplicaciones típico es Xcode, el cual es gratuito y permite compilar código C, C++, Objective-C, Objective-C++, Java y AppleScript. Xcode integra en una sola interfaz todo lo que involucra código, dise?o de interfaz de usuario (**Interface Builder**) y *debugging*. También viene con un conjunto herramientas útiles para evaluar la performance de la aplicación en distintos aspectos que se llama **Instruments**. Por otra parte viene con

un conjunto importante de *Frameworks* entre los cuales se encuentran **Cocoa** y **Cocoa Touch** que proveen de herramientas útiles para desarrollar más fácilmente aplicaciones para Mac OS X e iOS respectivamente.
Las aplicaciones que corren sobre los distintos dispositivos como iPhone, iPod Touch, iPad o AppleTV están desarrolladas en Objective-C pero sobre la base de estas librerías o *Frameworks* de iOS que se pueden separar en cuatro grandes capas según el nivel de abstracción: Cocoa Touch, Media, Core Services y Core OS. Así entonces, dentro de cada capa existen distintos *Frameworks* según la

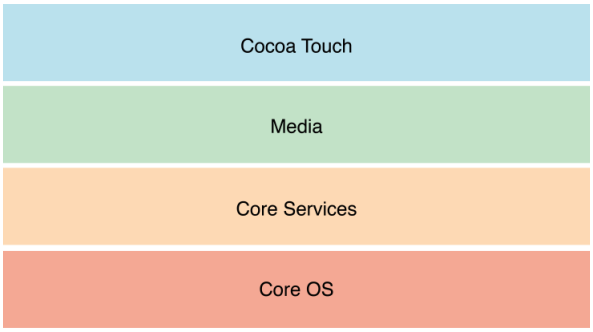

Figura 1.2: Capas de iOS

funcionalidad. A continuación se explica un poco más en detalle el rol de cada capa, los distintos *Frameworks* que tiene cada una y para qué sirven.

1.4.3.1. Cocoa Touch Layer

Cocoa Touch es la capa de más alto nivel de iOS y es la encargada de proveer al desarrollador de ciertos *Frameworks* que permitan lograr distintas tecnologías como la posibilidad de multitarea, el ingreso de órdenes a la aplicación a través de la pantalla táctil, notificaciones y alertas, preservación del estado de la aplicación al salir de la misma, reconocimiento de gestos en la pantalla y otro tipo de funcionalidades de alto nivel. Permiten al desarrollador, sin tener que involucrarse demasiado a bajo nivel, el acceso a determinados servicios que ya están resueltos en forma bastante modular.
Cocoa Touch está basado en la arquitectura **Modelo-Vista-Controlador**, en el que se separa en tres áreas distintas el modelo de la información, la interfaz de usuario y el conjunto de reglas que negocian la presentación de la información en base a la interacción con el usuario. Así pues, el usuario y una aplicación se podrían considerar dos sistemas que interaccionan. Por su parte el usuario tiene como entrada la vista de la aplicación y como salida tiene su respuesta a esta entrada, generando efectos sobre el control de la aplicación. Por otro lado la aplicación tiene como entrada las órdenes dadas por el usuario que tienen efectos sobre el modelo de la información y este sobre la vista, quien resulta ser la salida de la aplicación. Esta interacción se puede ilustrar con la figura 1.3. Como se dijo, dentro de Cocoa Touch, existen distintos *Framewokrs* enfocados en permitirle al desarrollador resolver en alto nivel distintos aspectos. Los mismos son los siguientes:

(1) Address Book UI Framework

(2) Event Kit UI Framework

(3) Game Kit Framework

(4) iAd Framework

(5) Map Kit Framework

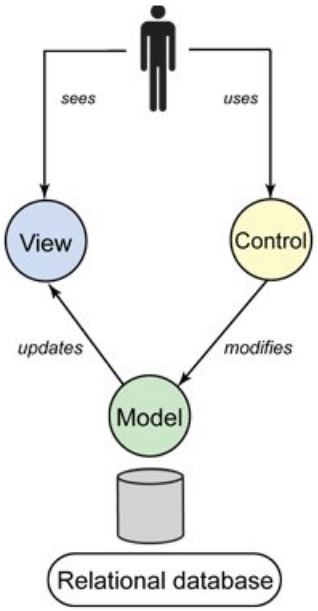

Figura 1.3: Interacción entre las tres partes del MVC

(6) Message UI Framework

(7) Twitter Framework

(8) UIKit Framework

Quizá sea bueno mencionar que varias de estas API no fueron utilizadas en el presente proyecto dada su función específica y que no fueron necesarias. Sin embargo hay una en particular que tiene bastante importancia y que permite la mayoría de las funcionalidades básicas que toda aplicación tiene. Se trata del **UIKit**, encargado de gestionar la aplicación, su interfaz de usuario y gráficos, encargado soportar eventos frente al toque de la pantalla, de manejar sensores como el acelerómetro y giroscopio, y de tener acceso a la cámara y galería de fotos entre lo más importante a destacar. El soporte de la multitarea y de **Storyboards** también está a cargo de este *Framework*.
Hay funcionalidades que han ido cambiando con las distintas versiones de iOS. Una de ellas y quizá una que ha generado bastantes diferencias respecto a versiones anteriores a iOS 5, es esta última, el Storyboard, una herramienta muy útil de programación gráfica, que permite generar instancias de clases y vínculos entre las mismas en forma visual a la vez de ser una contraparte de interfaz de usuario. Con una biblioteca de objetos disponibles, listos para ser usados, mediante el uso de Storyboard se hace accesible con algunas horas de dedicación implementar aplicaciones sencillas. Esta herramienta vino para sustituir los archivos *.nib* que permitían dise?ar la interfaz pero no tantas funcionalidades programáticas como el Storyboard. En particular éste último permite agregar la funcionalidad de *segues*, encargados de vincular un *ViewController* con otro. Este tipo de diferencias vinieron con la idea de evitar la necesidad de implementar ciertos bloques de código en forma repetitiva. Un Storyboard luce como en la figura 1.4.

Si bien se podría extender bastante más la explicación sobre los detalles de Cocoa Touch, a los efectos del presente proyecto, no es de tanta relevancia excederse en este punto.

1.4.3.2. Media Layer

Media Layer es la capa encargada de gestionar correctamente elementos multimedia y es posible ditinguir tres grandes grupos que engloban distintos *Frameworks*: Gráficos, Audio y Video.

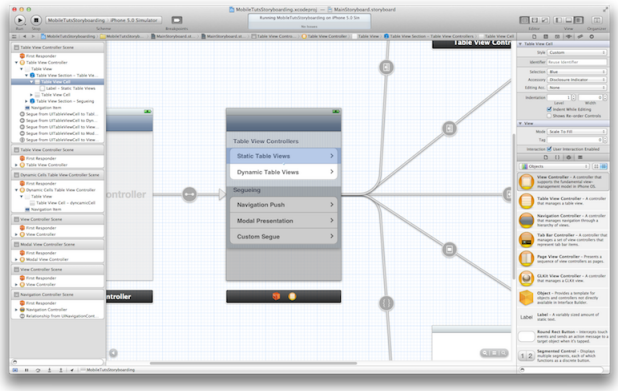

Figura 1.4: Ejemplo de Storyboard.

Dentro de las tecnologías más destacadas está todo lo vinculado a **gráficos** 2D y 3D, dentro de lo que se puede incluir algunos *Frameworks* bastante utilizados en el presente proyecto, tales como: **Core Graphics**, muy utilizado para dibujos 2D, **Quartz Core**, quien contiene las herramientas necesarias para interactuar con otro *Framework* para animación de vistas, de una capa de más bajo nivel como *Core Animation*, que es comentado más adelante en la sección 1.4.3.3. También es parte de lo vinculado a gráficos, el *Framework* **Core Image**, conteniendo lo vinculado a procesamiento de imágenes a través filtros que utilizan directamente la lu unidad de procesamiento de gráficos (GPU) y otros dos *Frameworks* bastante importantes en lo que respecta a *rendering* como **Open GL ES** y **GLKit** (utilizado por el motor de juegos *Isgl3d* entre otros).
Por otra parte hay otra gran familia de *Frameworks* dentro de Media Layer que apunta a resolver todo lo vinculado al manejo de audio, ya sea de grabación como procesamiento y reproducción de alta calidad. Existen algunos SDK como **iSpeech** o **Dragon Mobile** que resuelven de manera similar al proyecto Siri, el procesamiento de la voz humana reconociendo palabras e interpretando, que utilizan algunos de los *Frameworks* de procesamiento de audio de esta familia.
En cuanto a lo vinculado al manejo de video, como parte de esta capa, se tienen dos *Framework* importantes con distintos niveles de abstracción: **MediaPlayer** y **AVFoundation**. También existen otros *Frameworks* fuera de esta capa que son capaces de manejar video como la clase UIImagePickerController (muy utilizada en el proyecto) del mencionado UIKit. En la figura 1.5 se esquematiza el nivel de abstracción de los *Frameworks* de las distintas capas que son capaces de manejar multimedia. Con MediaPlayer es posible reproducir audio y video muy fácilmente en determinada área

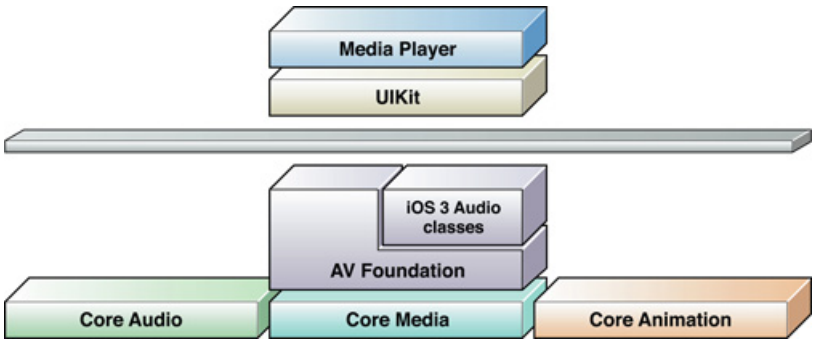

Figura 1.5: Frameworks de las distintas capas para manejo de video

de la pantalla ya sea desde un URL o de un archivo, es posible mostrar o no los elementos de control del video así como también controlar, volumen y tama?o de la pantalla. Por su parte, con AVFoundation es posible capturar con la cámara, reproducir, editar y procesar audio y video. Es posible

implementar ciertos protocolos que hace de esto algo relativamente sencillo.

1.4.3.3. Core Services

Core Services es la capa de más bajo nivel de iOS y contiene los elementos fundamentales sobre los que se construyen las capas superiores. Es posible que al comenzar a programar para iOS no se tenga mucha interacción con esta capa pero sin embargo existen algunos conceptos importantes de esta capa que sí vale la pena mencionar dado que en el presente proyecto se tuvieron que entender y discutir. Una de ellas es la *Automatic Reference Counting* o **ARC**. Esta funcionalidad compete a la reserva y liberación de memoria por parte de los objetos. La idea básica es lograr que el uso de memoria sea el mínimo posible, logrando que los objetos existan en la medida que son necesarios y que su memoria sea liberada ni bien sea posible. Típicamente, al crear una instancia de un objeto se

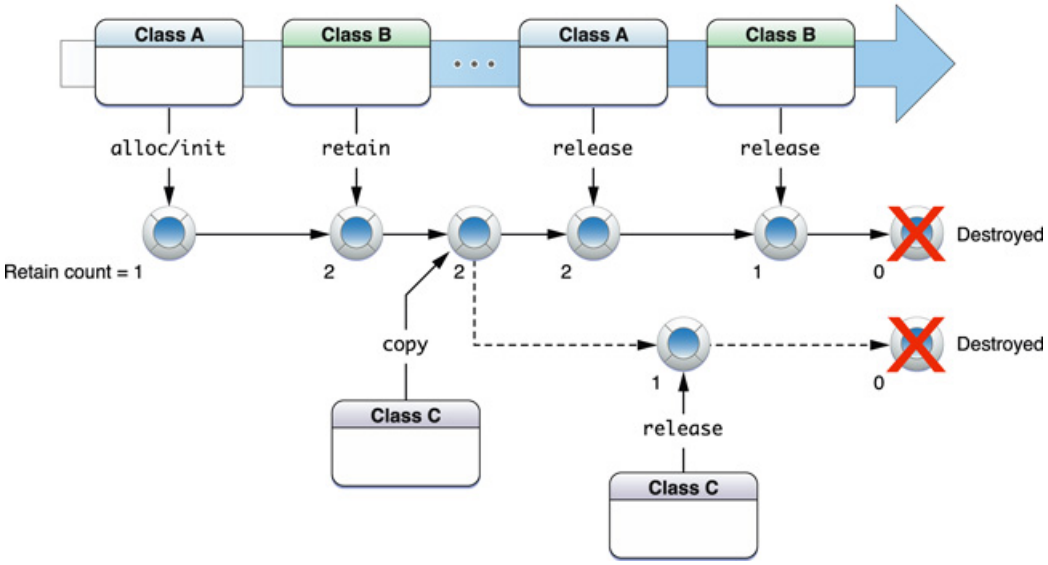

Figura 1.6: Ciclo de vida de objetos, Manual-Retain-Release.

incrementa un contador y al liberar se decrementa y entonces se tiene cierto control sobre la reserva y liberación de memoria en base al contador. Sin embargo, la liberación de memoria reservada por objetos queda bajo la responsabilidad del desarrollador y en casos de un código complejo puede llegar a ser habitual olvidarse de la liberación de memoria. Lo anterior refiere a una gestión manual de la reserva y liberación conocido como *manual retain-release*. Para no tener que enfrentar este tema y poder instanciar clases sin tener presente la posterior liberación de memoria (pues quizá se sepa cuándo no será más necesario un objeto o no), se utiliza ARC. Esta funcionalidad evalúa el ciclo de vida de los objetos y agrega código en tiempo de compilación en caso de considerarlo necesario. Es bueno aclarar que esto refiere a memoria reservada pura y exclusivamente por objetos, es decir mediante *alloc*. En caso de tratarse de memoria reservada para variables de lenguaje C (*malloc*), es necesario proceder de igual manera que en dicho lenguaje, liberando la memoria mediante un *free*. Además del ARC, Core Services permite el manejo de archivos XML y manejo de base de datos SQL así como también la protección de datos cuando el dispositivo está bloqueado entre otros servicios importantes. Tiene varios *Frameworks* como **Core Media** que logran un nivel aún más bajo que AVFoundation para el manejo de multimedia, **Quick Look** para las vistas previas de archivos, **Social** que viene a suplantar el *Framework* para la utilización de Twitter que existe en iOS 5 y extiende la gestión para otras redes sociales, **Core Motion** para el manejo de sensores como el acelerómetro y el giroscopio, **Core Telephony** para el manejo de la información de red del dispositivo

como elemento de la red de telefonía, **CFNetwork** para el manejo de protocolos de red como http, https, ftp y resolución de servidores DNS, entre otros *Frameworks* importantes dentro de la capa.

1.4.3.4. Core OS

Con esta capa de iOS en general es difícil que el desarrollador tenga que involucrarse directamente dado que es la de más bajo nivel. Salvo que se esté frente a una aplicación que requiera aspectos de seguridad o comunicación con HW externo, esta capa solamente existe para ser la base sobre la cual se desarrollan los *Frameworks* de las capas de más alto nivel. Los distintos *Frameworks* que tiene están enfocados en resolver temas de procesamiento basados en el hardware de iOS, en comunicarse con dispositivos externos basados en iOS y de garantizar la seguridad de los datos de una aplicación.

1.4.3.5. Simulador

Uno de los detalles más importantes del entorno de desarrollo es la capacidad de simular lo que se programa antes de probarlo en un dispositivo. Esto es útil por cuestiones de seguridad e incluso permite programar sin la necesidad de contar con una plataforma. Esto existe para *Xcode* y es necesario decir que funciona muy bien, generando una representación bastante fiel de lo que sucede en el dispositivo real. La única crítica que se le podría hacer es el hecho de no contar con cámara y para el caso de aplicaciones de realidad aumentada esto es algo bastante importante. Sin embargo, sin ser eso, el simulador cuenta con conexión a internet, pantalla multitáctil, con información de GPS ingresada por el programador, acceso a la galería de fotos, capacidad de procesamiento y todas las funcionalidades que un dispositivo real tiene.

1.4.3.6. Instruments

Dentro de las herramientas que vienen con el entorno de desarrollo viene *Instruments*, un conjunto de herramientas que permiten analizar la performance de una aplicación para iOS o para Mac OS X desde distintos puntos de vista. Resulta muy útil pues muchas veces sucede que una aplicación compila y se ejecuta correctamente y sin embargo puede que el desarrollador no esté conforme en cuanto a los tiempos de procesamiento o el uso de memoria consumido.
Para poder hacer uso del *Instruments*, es necesario correr la aplicación en modo *Profile*. Eso despliega una ventana como la de la Figura 1.7. Allí es posible elegir dentro de cada una de las posibilidades que ofrece *Instruments*, si se quiere analizar tiempos, memoria, recursos de CPU, *multithreading* entre otros tipos de datos de interés que es posible recoger de la aplicación. También es posible elegir la plataforma, ya sea iOS, simulador iOS o Mac OS X.

Luego de elegir el tipo de datos a ser recolectados según lo que se busque analizar, se despliega una ventana como se ve en la Figura 1.8. Allí es necesario registrar durante varios segundos los datos mientras se corre la aplicación y luego de terminado el registro, *Instruments* dedica un tiempo a analizar los datos recolectados. En el detalle inferior, se desglozan los procesos que corre la aplicación en forma de árbol. Cuando se desea medir tiempos por ejemplo, esto resulta muy útil porque entre otras cosas se puede analizar qué porcentaje del tiempo de la aplicación es consumido por un método en particular. Esto es posible simplemente buscando dentro del árbol mencionado, al método y leyendo el valor asignado de tiempo. Para el caso de análisis de memoria también es posible identificar fácilmente en qué parte del código se está dando algún problema de reserva de memoria no liberada. En el presente proyecto se hizo uso principalmente del **Time Profiler** que permite analizar tiempos y del **Memory Leak** que permite hacer un análisis de la reserva de memoria

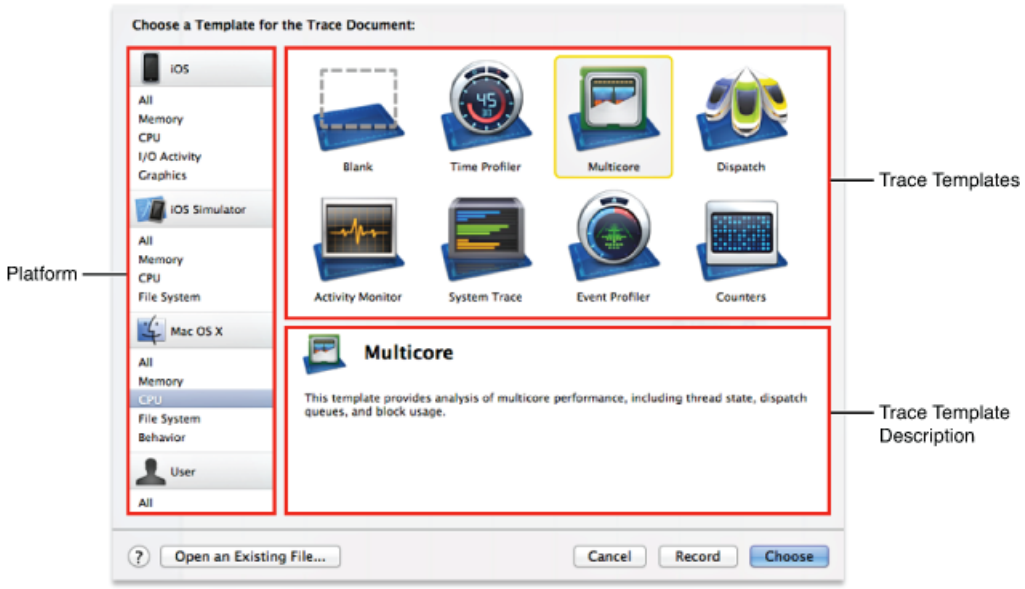

Figura 1.7: Distintas opciones de *Instruments*.

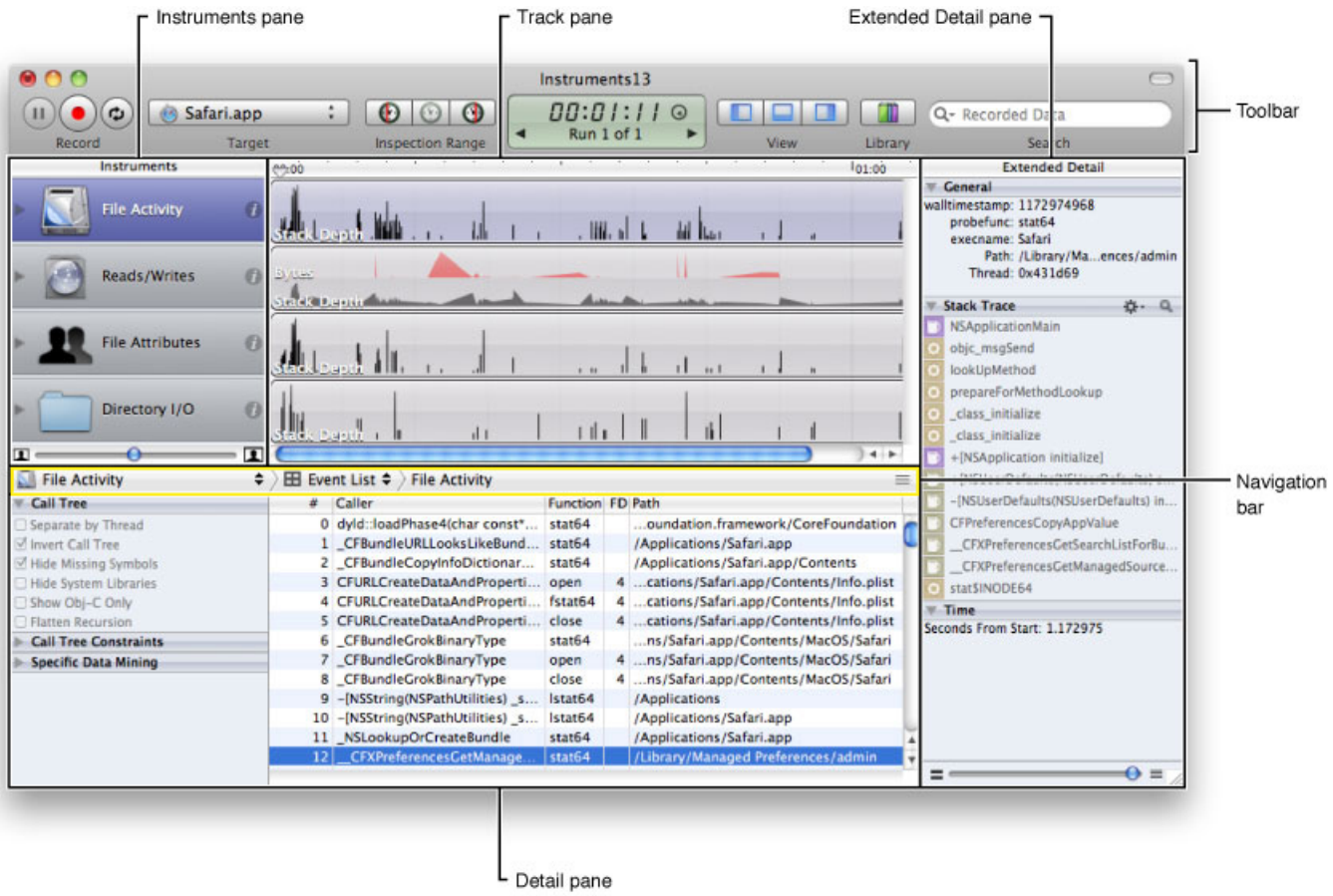

Figura 1.8: Trazado y análisis de datos recogidos.

no liberada. Con los mismos fue posible optimizar tiempos en determinados métodos del procesamiento, así como también eliminar problemas de memoria no liberada que desencadenaban en la interrupción abrupta de la aplicación luego de llegado un cierto nivel de reserva. Esta interrupción

abrupta es una forma de proteger la memoria del dispositivo y evitar que se vea afectada cierta memoria útil a otros efectos. El resto de las herramientas de *Instruments* fueron probadas pero no utilizadas en detalle para resolver problemas particulares.

1.5. Herramientas

Herramientas

1.5.1. GIT

1.5.2. GoogleCode

1.5.3. Github

[**?**].