Plataforma para guías turísticas virtuales basadas en teléfonos móviles

Juan José Molinero Horno

Febrero de 2012

Agradecimientos

En vista de que este ya es el segundo intento y como en el primero ya agradecí su participación a los *sospechosos habituales*, supongo que en esta ocasión el mayor agradecimiento tiene que ser para Ana Cris, quien es posiblemente la mayor culpable de que este PFC se haya terminado (incluso más que el propio autor). Así pues gracias a tí por haber sido, además de directora del proyecto, amiga durante estos meses y haberme dado el *coñazo* suficiente para que al final, lo haya terminado despues de ocho años.

... y ya que estamos, gracias también a los componentes de la empresa Disline, por haberme permitido participar en un proyecto con ellos. Espero que tengáis mucho éxito con la aplicación y con todos los proyectos que emprendáis en el futuro.

Resumen

El negocio de la gestión de contenidos turísticos se ha vuelto muy competitivo en estos últimos tiempos. Desde la aparición de los dispositivos móviles de última generación o *smartphones*, los usuarios demandan la obtención de este tipo de servicios en cualquier lugar usando su conexión a Internet. La empresa Disline, dedicada a este tipo de productos, propuso la realización de un sistema de publicación de contenidos de interés turístico que se adecuara a las necesidades impuestas desde el mercado.

Durante el presente proyecto se ha desarrollado un sistema que se compone de dos aplicaciones. Una parte llamada *gestor web*, desde la cual la empresa de creación de contenidos puede introducir información referente a los puntos de interés de un determinado entorno turístico. Una vez se ha introducido toda la información deseada, se generá una base de datos con los contenidos que puede ser usada en la segunda parte del sistema, la *aplicación móvil*. Esta aplicación muestra, usando la información que obtiene de la base de datos generada anteriormente, los puntos de interés deseados en forma de guía turística de forma que resulte útil para el usuario. En este proyecto se ha realizado una versión de la aplicación móvil para el sistema iOS.

Además del desarrollo de las aplicaciones mencionadas anteriormente, se ha realizado un estudio sobre las técnicas de reconocimiento de objetos mediante visión por computador que son más adecuadas en el ámbito del proyecto. Aunque las técnicas que se basan en el reconocimiento del objeto sin utilizar ningún marcador adicional muestran grandes posibilidades, para la realización de un producto comercial las técnicas de reconocimiento de objetos mediante marcadores (códigos QR) son las que se encuentran más maduras actualmente y por ello se ha decidido implementar estas dentro de la aplicación.

Finalmente, durante el desarrollo del proyecto, otras características interesantes para el sistema han sido identificadas en colaboración con la empresa para desarrollar en otros proyectos en paralelo o futuros que añadan dichas funcionalidades dentro del sistema.

Índice general

Índice de figuras

Capítulo 1

Introducción y definición del problema

El uso de dispositivos móviles se está haciendo cada vez más común para todo tipo de gente y todo tipo de tareas. Desde la aparición de los llamados *smartphones* que integran una serie de hardware no disponible en dispositivos anteriores como WIFI, GPS, cámaras de última generación, sistemas de pago NFC [1], ... se han aumentado las posibilidades ofrecidas por estos dispositivos de manera exponencial. Sin embargo, lo que hace verdaderamente diferente a este tipo de dispositivos de los que se podía disponer anteriormente es la posibilidad de instalar aplicaciones en ellos desarrolladas por terceras partes.

Otra característica que ha ayudado a la expansión del mercado de dispositivos móviles es la posibilidad por parte de los usuarios de disponer de conexión a Internet de banda ancha en casi cualquier sitio en el que se encuentren. Esto hace que algunos de estos aparatos tengan un poder casi adictivo que hace que los usuarios pasen una gran cantidad de tiempo usandolos.

Hasta hace relativamente poco tiempo (aparición del App Store de Apple en Julio de 2008) el desarrollo de una aplicación para un dispositivo móvil era un trabajo tedioso tanto para el desarrollador (falta de herramientas de calidad para el desarrollo, poca integración de las API's existentes con la plataforma hardware, excesiva segmentación de modelos de los fabricantes, ...) como para el usuario (falta de un lugar donde buscar aplicaciones que se adaptaran a una necesidad, dificultad a la hora de instalar una aplicación, ...). Desde la aparición del App Store (y subsiguientes repositorios de aplicaciones por parte de otras empresas de sistemas operativos móviles) todos los problemas anteriores se han ido solucionando o al menos mitigando de forma que hasta los usuarios menos avanzados son capaces de encontrar

[1] http://es.wikipedia.org/wiki/Near_Field_Communication

Figura 1.1: Diferentes modelos de dispositivos *smartphones*

aplicaciones que se adapten a sus necesidades (aunque han aparecido nuevos problemas como la aparición de un número excesivo de aplicaciones para cada tarea que hacen difícil ser capaz de encontrar la mejor solución).

Entre las diferentes plataformas móviles sobre las que desarrollar aplicaciones hay tres que están teniendo un éxito superior al resto: Android, Blackberry e iOS. Todas ellas han sido capaces de concentrar la mayor parte de la atención por parte de desarrolladores de aplicaciones y usuarios de dispositivos móviles de forma que entre ambas. En el caso de Blackberry sin embargo el número de aplicaciones desarrolladas por terceras partes y descargadas por los usuarios es mucho menor ya que parece ser que los usuarios de estos dispositivos tienden a usar mayoritariamente las aplicaciones que viene por defecto en el dispositivo. Con estos datos, podemos concluir que las mejores plataformas tanto para el desarrollo de aplicaciones como para la distribución de las mismas al mayor número posible de clientes, son Android e iOS.

1.1. Motivación

En este proyecto, se decidió usar la plataforma de desarrollo iOS (aunque hay planes para portarlo tambien a Android, y si hubiera demanda también a otras plataformas) debido a que los datos que se tienen en este momento indican que el número de aplicaciones que compran y descargan los usuarios de esta plataforma es mayor que el cualquier otra platforma. Además, el

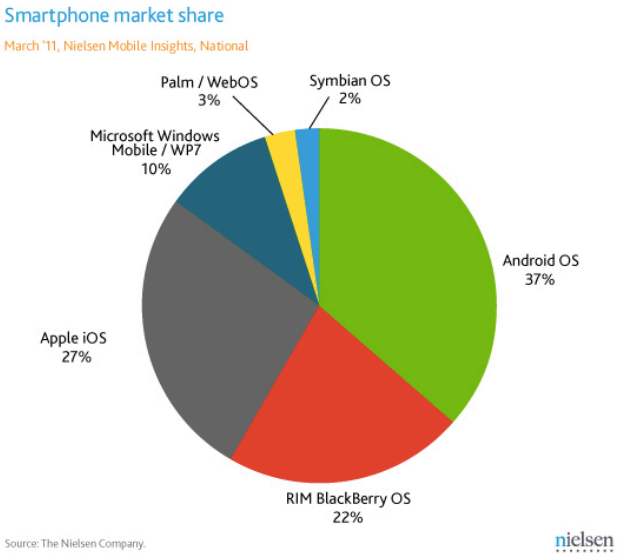

Figura 1.2: Cuota de mercado de los diferentes sistemas operativos para teléfonos móviles

tiempo de desarrollo por aplicación es menor debido a que esta plataforma proporciona herramientas de desarrollo más maduras.

Lo que se pretende durante el desarrollo del mismo es la creación de una herramienta que permita la creación semi-automática de guías multimedia adaptadas sobre todo al turismo rural. Más concretamente, se ha desarrollado una aplicacion móvil para la plataforma iOS base, que puede ser parametrizada mediante los datos obtenidos de una herramienta web de forma que cambie el contenido y el diseño de la misma y la intervención del equipo de desarrollo sea la mínima posible en futuras aplicaciones.

Debido a la gran cantidad de plataformas móviles disponibles y a la diversidad de entornos sobre los que se desarrolla para ellas, en ocasiones es posible que el coste de realizar una serie de aplicaciones para las mismas pueda acabar siendo demasiado elevado, tanto en tiempo como en dinero. Por ello, en la realización de este proyecto se ha intentado dotar a la empresa de más flexibilidad y *autonomía* a la hora de generar nuevas guías. Para ello se ha decidido generar un sistema en el que el coste de crear nuevas guias turisticas sea practimante marginal (al menos la parte de desarrollo de software) y únicamente haya que gastar tiempo y dinero en la parte de creación de contenidos que es en realidad el negocio básico de la empresa para la que estamos realizando el trabajo.

1.2. Trabajos previos

Debido a la gran cantidad de aplicaciones disponibles para la plataforma en la que estamos desarrollando, es imposible que no haya disponibles varias aplicaciones que se dediquen a lo mismo que se esta desarrollando. Sin embargo, la aplicación que vamos a desarrollar tiene la ventaja sobre todas ellas de que nuevas versiones de la aplicación para diferentes entornos (otros pueblos, museos, ...) se crearán de forma semiautomatica por parte de personal no cualificado técnicamente de la empresa.

Por otra parte, el resto de las aplicaciones de turismo y guias de viaje (Lonely Planet, Tripwolf, ..) son bastante similares y ofrecen una serie de posibilidades bastante parecidas:

- Introducción.
- Listado de monumentos y puntos de interés.
- Descripción de los monumentos y puntos de interés con posibilidad de acceder a material multimedia e interactivo.
- Mapa con la posición de los diferentes elementos.

Nuestra aplicación ademas de todo esto, será capaz de soportar también el uso de marcadores de posición que puedan ser reconocidos de forma automática por la cámara del dispositivo de forma que se pueda localizar puntos de interés mediante una fotografía del usuario al codigo correspondiente. Esto puede ser de utilidad en zonas con poca precision del GPS o en interiores, donde se pierde la cobertura casi totalmente.

1.3. Objetivos y entorno de trabajo

Objetivos

Recientemente ha habido una gran explosión de nuevas aplicaciones, funcionalidades y servicios asociadas al desarrollo de aplicaciones sobre moviles de ultima generación o *smartphones*. La tendencia de los usuarios a llevarlos siempre encima y las posibilidades que ofrecen los elementos hardware de estos dispositivos (cámara, acelerómetros, GPS, conexión WIFI, ..) ofrecen grandes posibilidades a los desarrolladores.

El objetivo principal de este proyecto es la realización de un sistema informativo que permita la creación y mantenimiento de guias turisticas adaptadas a dispositivos moviles. El sistema constara de diferentes formas de

localización e identificación de elementos de interés de forma que se adapten a los diferentes tipos de entorno donde se pueden realizar visitas turisticas. Por ejemplo, se distinguira entre exteriores donde el usuario se puede guiar usando la localización por GPS e interiores, donde al no poder usar el GPS, usaremos algun tipo de sistema de reconocimiento de códigos de barras o similares.

El sistema constará de dos partes, una apliacion servidor que será donde se cree las guías por parte de la empresa de creación de contenidos, y una aplicación cliente desarrollada en un teléfono (para la primera versión hemos elegido el iPhone e iOS como sistema para el prototipo), que podrá descargarse las guias del servidor y asistirá al usuario en sus visitas a lugares de interés.

Los objetivos o tareas a realizar en este proyecto en más detalle son:

- Obtención de requisitos por parte de la empresa Disline [2](empresa dedicada a los servicios de promoción de la cultura y el turismo). El prototipo será utilizado en la empresa interesada.
- Familiarización con la arquitectura de la plataforma iPhone, el entorno de desarrollo XCode y el lenguaje Objective-C y las librerías compatibles para visión por computador (reconocimiento de códigos).
- Familiarización con el entorno de desarrollo web a utilzar para la parte del servidor (Ruby on Rails).
- Diseño y desarrollo de la aplicación web servidor que permita la creación y mantenimiento de guías.
- Diseño y desarrollo de la aplicación móvil cliente que permita la localización en un mapa (usando Google Maps) de puntos de interés y el reconocimiento de algún tipo de código de barras para identificar objetos.

También se ha investigado la posibilidad de añadir a la aplicación mejoras para hacerla más general e innovadora:

- Reconocimiento de objetos sin el uso de códigos de barras o similares, usando técnicas de visión por computador.
- Localización de puntos de interés en mapas cargados en el dispositivo móvil sin necesidad de conexión a Internet, solamente mediante el acceso a la información del GPS.

[2]http://www.disline.es

Entorno

Como ya se ha comentado previamente, este PFC se ha realizado en colaboración con la empresa Disline. Esto ha hecho que haya habido un periodo de adaptación a los sistemas de trabajo de ambas partes. Más adelante se explicará un poco la metodología de trabajo empleada para la realización de los diferentes elementos que componen el proyecto así como una pequeña explicación de porque se ha elegido la misma.

Para la parte del desarrollo del cliente móvil se ha usado como entorno de programación el Xcode de Apple (así como algunas herramientas adicionales como por ejemplo el Menial Base para accesos a la base de datos Sqlite utilizada o el sistema de control de versiones Mercurial) debido a que es básicamente la única forma de realizar aplicaciones para el sistema operativo iOS de una forma más o menos sencilla. El lenguaje de programación exigido por este entorno es Objective-C, por lo que a parte de algunas pocas líneas en SQL para la creación y mantenimiento de la base de datos ha sido el único lenguaje empleado en la parte del cliente.

En cuanto al desarrollo del servidor, despues de estudiar diferentes frameworks y lenguajes de programación, se ha decidido utilizar el lenguaje de programación Ruby y el entorno Ruby on Rails [3]. Se ha elegido debido a la gran popularidad que esta empezando a tener en el prototipado y desarrollo rápido de aplicaciones web. Este entorno, unido al editor de texto Emacs [4] y el plugin Rinari [5] facilita en gran medida el poder realizar cambios relativamente grandes en el sistema de forma comoda lo que dado la forma de trabajar que se utilizo con la empresa resultó de gran ayuda.

1.4. Resumen del contenido de la memoria

En el resto de la memoria se detalla, en el capítulo 2, las técnicas y librerías de visión por computador que se han estudiado para la parte del reconocimiento de códigos de barras o similares en la parte del cliente móvil, así mismo se explicará el diseño e implementación del cliente realizado en iOS. El capítulo 3 resume la arquitectura y diseño del gestor de contenidos de las guias turisticas así como las decisiones de diseño que se han tomado para adaptarlo a las necesidades del cliente. El capítulo 4 concluye la memoria con la descripción de posibles ideas a implementar en un futuro (algunas de las cuales se comenzaron a desarrollar en paralelo a este proyecto).

[3] http://www.rubyonrails.org
[4] http://www.gnu.org/software/emacs/
[5] http://rinari.rubyforge.org/

Capítulo 2

Diseño y desarrollo de la aplicación de turismo en el móvil

Después de las reuniones iniciales con la empresa se decidió que lo mejor para que el desarrollo del proyecto fuera lo más ágil posible era empezar con la aplicación móvil. De esta forma, se estimarían mejor las necesidades que el gestor web debería satisfacer.

El trabajo necesario para el desarrollo de dicha aplicación se dividió en tres partes:

- En primer lugar, un estudio de las técnicas de visión por computador para reconocimiento de objetos, de forma que se pudiera decidir cual es la que mejor se adaptaba a las necesidades del cliente.
- Seguidamente, se llevaron a cabo una serie de reuniones conjuntas con la empresa en las cuales se decidieron las caracteristicas principales del sistema, y se aclararon las posibles dudas que hubiera de cara a que el desarrollo fuera lo más sencillo posible.
- En tercer lugar se procedió al diseño, implementación y pruebas de la aplicación.

En los siguientes apartados, se hace una pequeña descripción de todo el proceso. En el apéndice A se incluye una descripción temporal de las distintas partes del proyecto.

2.1. Sistemas y librerías de reconocimiento de códigos de barras y similares

Una de las tareas estipuladas al principio del proyecto, era la de analizar y estudiar que técnicas de visión por computador podrían ser las más adecuadas para integrar directamente en el prototipo inicial del cliente de forma que se pudieran reconocer objetos descritos en la guía con la cámara del dispositivo móvil. Esta sección presenta primero un pequeño análisis de trabajos relacionados (tanto comerciales como de investigación) y después describe en más detalle la opción elegida para incorporar en el prototipo (reconocimiento mediante el uso de códigos QR).

Reconocimiento visual con la cámara de un móvil

Debido a ciertas características de estos dispositivos, a la hora de desarrollar técnicas de visión por computador para los mismos se presentan una seríe de desafíos que tienen que ser superados cuidadosamente para que el resultado pueda ser eficiente. Hay algunos obstáculos tradicionales que con el avance de la tecnología se van superando, como el tamaño, peso y potencia de los disposivos, mientras que otros siguen todavía vigentes, como por ejemplo la duración de la batería. Otro de los problemas a tener en cuenta, que no solo afecta al entorno de las aplicaciones móviles, es el hecho de que el campo de la visión por computador involucra elementos de percepción e inteligencia artificial que todavía no estan completamente resueltos a la hora de hacer aplicaciones que se ejecuten en máquinas de escritorio, con lo que mucho menos se puede esperar que funcionen perfectamente en los, por el momento, menos potentes dispositivos móviles. El reconocimiento de objetos es un campo de investigación muy activo en el área de visión por computador [1] [2], con grandes avances en los últimos años. Por ello, pese a los problemas todavía sin resolver, ya hay algunas aplicaciones que estan empezando a tener cierto éxito en el mercado. Por ejemplo podemos nombrar:

- **Google Googles**[1]: Una aplicación de la empresa Google en la cual se pueden usar imágenes tomadas con la camara del dispositivo móvil para buscar la web.
- **Kooaba**[2]: Una empresa que se dedica a hacer aplicaciones de visión por computador para teléfonos móviles. Como ejemplo podemos citar

[1] http://www.google.com/mobile/goggles
[2] http://www.kooaba.com/

Shortcut, una aplicación que permite escanear noticias de publicaciones y que te conecta con contenido relacionado con las mismas en Internet.

Análisis de las diferentes opciones

En la comunidad internacional de investigadores de visión por computador y procesamiento de imágenes, encontramos múltiples librerías de código abierto que facilitan la investigación en los mismos. Incluso también han empezado a aparecer algunas librerías que facilitan todo el proceso de llevar la visión por computador a los dispositivos móviles, como por ejemplo la portabilidad de la librería OpenCV [3] a la plataforma iOS [4]. Haciendo uso de estas librerías, se realizaron unas pruebas iniciales para las posibilidades de incluir técnicas bien establecidas de visión por computador en el prototipo a diseñar. En particular, se decidió probar el reconocimiento de objetos basado en descriptores de imágen muy conocidos y utilizados en estas tareas, SURF [referencia extracción SURF). Estos descriptores de imágen capturan los puntos más distintivos en una imagen, y han sido presentados en aplicaciones similares a nuestros objetivos [3] [4]. Al final se decidió buscar una alternativa más madura en productos comerciales que el uso de esta técnica, para el reconocimiento de puntos de interés, debido a dos problemas:

- Se consiguió implementar y ejecutar los elementos básicos del proceso (detección y procesado de los puntos SURF descritos) en el teléfono móvil, pero el tiempo de ejecución de los métodos más estandar es muy elevado para la aplicación deseada.

- Además, para el uso de este tipo de técnicas, hay que entrenar al sistema para obtener los parametros adecuados para cada una de los objetos a reconocer, esto implica que los empleados de la empresa deberían utilizar un tiempo mayor en la creación de contenidos del que se emplea ahora (obtención de fotos desde diferentes ángulos, introducción de fotos en el sistema, ...).

Por lo tanto se estimó no implementar este tipo de reconocimiento de objetos en este proyecto, debido a que requiere una tarea de investigación más larga que se pospuso como una tarea independiente de este proyecto, para desarrollar en trabajos futuros. Este tipo de técnicas sería muy interesante por ejemplo en lugares en los que no sea posible manipular el entorno para poner códigos que identifiquen a las piezas o elementos de interés que tengan

[3] http://opencv.willowgarage.com/

[4] http://www.eosgarden.com/en/opensource/opencv-ios/overview/

algun tipo de movilidad y por lo tanto sea más difícil etiquetarlos. Por lo tanto, es posible que en un futuro se puedan implementar dentro del sistema, ya sea sustituyendo a los códigos QR o como complemento de los mismos.

Después de analizar varias técnicas más maduras en el ámbito del reconocimiento visual de objetos, se decidió que lo más robusto y flexible de cara a las necesidades de la empresa es el reconocimiento de códigos de barras y similares. En concreto se decidió utilizar los códigos QR que permiten la codificación de más información que otro tipo de técnicas.

Implementación y detalle de la técnica elegida

Los códigos QR (del inglés Quick Response Code, código de respuesta rápida) es un tipo de código de dos dimensiones en el cual se puede codificar hasta 4200 carácteres. El código consiste en un patrón de cuadrados negros sobre un fondo blanco. Fue inventado por Toyota en 1994 para realizar un seguimiento de sus vehiculos y se ha convertido en uno de los códigos más utilizados debido a la velocidad con la que se puede decodificar.

Figura 2.1: Ejemplo de código QR.

Entre sus características principales se encuentran:

- Alta de velocidad de descodificación.
- Gran capacidad de almacenaje.

- Posibilidad de encriptación.
- Corrección de errores.

En la actualidad este tipo de códigos se están introduciendo en un gran tipo de aplicaciones diferentes como puede ser publicidad, realidad aumentada, inventarios, ... Para más información sobre este tipo de códigos se puede consultar el estándar ISO que lo describe [5].

Para este proyecto se ha utilizado la librería ZBarSDK [5] que es un *software* de código libre para la decodificación de varios tipos de códigos de dos dimensiones.

2.2. Requisitos del cliente y decisiones de diseño adoptadas

Como se ha mencionado anteriormente, durante las reuniones mantenidas con la empresa Disline, se especificaron los requisitos básicos que debería cumplir la aplicación. Para ello, gente de la empresa aporto su conocimiento del modelo de negocio y del tipo de clientes con los que habitualmente tratan. Estos requisitos están basados tanto en la interacción que los clientes van a tener con la aplicación como en la relación que esta debe tener con el gestor web. Así mismo, también se decidió durante estas reuniones la plataforma de desarrollo del primer prototipo.

El primer punto que se acordó fué que la interfaz de usuario debía ser lo más sencilla posible, ya que por experiencia propia las funcionalidades más complejas de las aplicaciones tienden a ser utilizadas por un número muy pequeño de usuarios con lo que el tiempo que se tarda en desarrollarlas no suele resultar rentable para la empresa. Por ello, se redujo la funcionalidad al minimo posible de forma que en un futuro, y si los usuarios lo demandan, se pudiera ir añadiendo nuevas funcionalidades. Al final, la lista de funcionalidades que va a tener la aplicación puede ser resumida en:

- Listados e información detallada (texto, imágenes y videos informativos) de puntos de interés clasificados en diferentes categorías para una búsqueda más facil de los mismos.
- Mapa con todos los puntos de interés en el que el usuario vea también su propia posición para poder comprobar cuales son los puntos más cercanos a si mismo. Este mapa se ha de poder filtrar de forma que en

[5] http://zbar.sourceforge.net/iphone/sdkdoc/

lugares donde haya muchos puntos de interés concentrados, el usuario pueda encontrar aquellos en los que este interesado (e.g. restaurantes si es la hora de la comida). La base de este mapa será el entorno y estilo de "google maps", ya que mucha gente está habituado a ellos y resultan cómodos y fáciles de usar.

- Escaner de códigos QR, de forma que si el usuario encuentra alguno de estos códigos mientras esta realizando turismo en alguna localidad pueda obtener más información sobre el elemento de la guía asociado a dicho punto. Esta característica deberá estar presente únicamente en algunas guías, con lo que deberá ser posible esconderla de forma paramétrica.

Otro de los puntos importantes que se acordó es que debería ser muy fácil para la empresa el crear una nueva guía para otro de sus clientes de forma que no se necesitará mucho tiempo ni dinero para la creación de un nuevo producto. Para ello, se estimó que lo mejor sería que la aplicación fuera parametrizable basandose en la información contenida en una base de datos situada en el mismo teléfono y con la cual se podría configurar la aplicación completamente, no solo el contenido turístico en si, sino también el color básico de las ventanas, los iconos, ... Además el hecho de tener toda la información de apariencia y contenido dentro de una base de datos (en oposición a codificada directamente) hace que sea más fácil la portabilidad a otras plataformas con el mismo contenido y apariencia similar.

Plataforma y metodología de desarrollo

Se decidió que la mejor plataforma para empezar con el prototipo sería Apple iOS, ya que a parte de tener una cuota de mercado considerable, las herramientas de desarrollo de las que dispone son de mayor calidad a la del resto de plataformas. En el Apéndice C se describirá en mayor profundidad la plataforma de desarrollo.

En cuanto a la metodología de desarrollo, y debido principalmente a que la empresa con la que se colabora no tenía muy claro todas las características que esperaba de la aplicación, se estimó que lo mejor sería usar algo parecido a las metodologías ágiles de desarrollo. Se decidió hacer periodos de trabajo cortos (en torno a las dos semanas en este caso) tras los cuales se realizarían breves demostraciones de lo que estuviera desarrollado en ese momento. Debido a esto, se tuvieron que acomodar cada una de las funcionalidades a desarrollar como un conjunto cerrado. En el apéndice B se explica un poco más en que consisten las metodologías de diseño ágiles.

2.3. Arquitectura, diseño y pruebas de la aplicación cliente

Modelo-vista-controlador (MVC)

Debido al uso de la plataforma iOS para el desarrollo de la aplicación cliente, la arquitectura básica para la aplicación se basa en el patrón de diseño MVC (modelo-vista-controlador) [6]. En esta arquitectura la aplicación se divide en tres partes que interactuan entre sí para obtener los *inputs* del usuario y mostrarle los datos que este haya pedido.

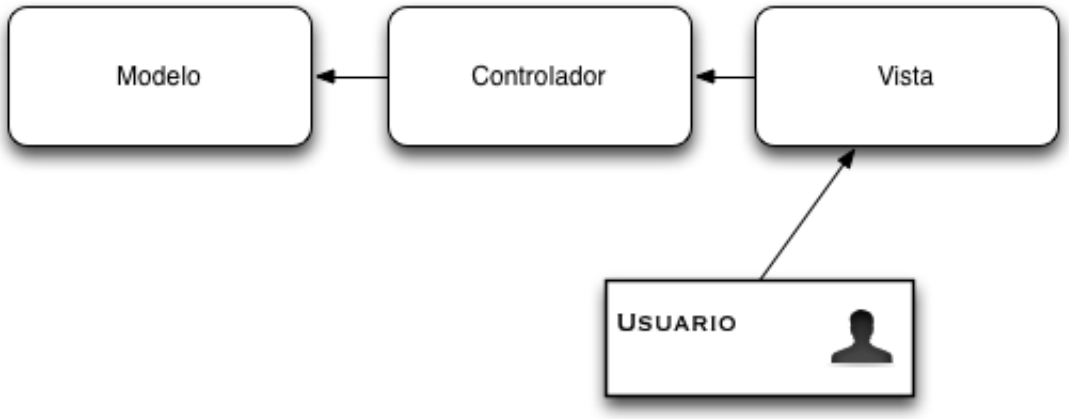

Figura 2.2: Arquitectura de la aplicación cliente - Patrón MVC

El usuario actúa directamente contra los diferentes elementos que se han codificado como *vistas*, en este caso las vistas son las pantallas que se han diseñado usando la aplicación Interface Builder de Xcode. La vista a su vez envía los datos al *controlador*, que son las clases que se encargan de organizar el trabajo dentro de la aplicación y de decidir que parte del *modelo* (clases dedicadas al almacenamiento y tratamiento de los datos) ha de ser utilizado para obtener los datos que el usuario ha solicitado. Una vez el *controlador* ha obtenido los datos necesarios, elige la siguiente *vista* que se mostrará al usuario y le pasa los datos para que a su vez esta sea la que los organice en una nueva pantalla que será la que el usuario pueda visualizar.

[6] https://developer.apple.com/library/mac/#documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

2.3.1. Arquitectura y módulos de la aplicación

A parte de esta organización de los diferentes elementos de la aplicación, estos se pueden clasificar también de forma que los elementos que trabajen juntos para realizar una determinada función, esten agrupados.

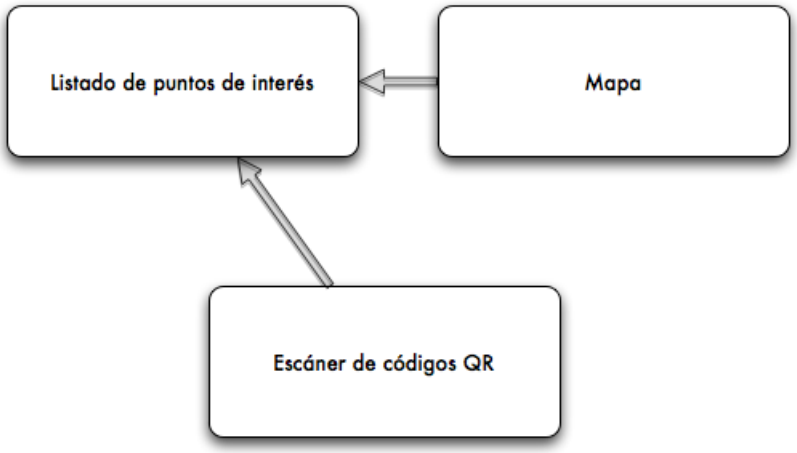

Figura 2.3: Diagrama de módulos de la aplicación móvil

Desde el punto de vista de la funcionalidad, se puede dividir la aplicación en tres módulos principales. Estos se corresponden con las tres principales acciones que se pueden realizar en el aplicación:

- Listado de puntos de interés: este es el módulo dedicado a la obtención de una lista de los puntos de interés solicitados por el usuario (según estan englobados en una categoría determinada) y mostrarselos al usuario, a su vez si el usuario selecciona un punto en particular le mostrará información sobre el mismo.

- Mapa de puntos de interés: este es el módulo dedicado a mostrar en un mapa de los proporcionados por *Google Maps*[7] el listado completo de puntos de interés. Además ofrece la posibilidad de filtrar todos estos puntos de forma que el usuario solo vea aquellos que corresponden a unas determinadas categorías. También permite mostrar información de un punto determinado si es seleccionado por el usuario en la interfaz táctil, además de mostrar la posición del usuario en el mapa para que este pueda estimar cuales con los puntos más cercanos a la misma.

[7] http://maps.google.com

- Escáner de códigos QR: Este módulo se encarga de obtener una imagen de la cámara del dispositivo móvil y analizarla en busca de códigos QR que pudieran tener información relevante de la aplicación. Si se encuentra un código de este tipo, muestra cierta información por pantalla y permite ademas obtener una información más detallada del punto de interés (la misma a la que accederíamos pinchando en el elemento correspondiente desde el listado de elementos de la aplicación) incluyendo contenido multimedia y una descripción del mismo.

En cada uno de estos módulos, su puede apreciar a su vez la división entre los módelos, controladores y vistas que se ha explicado en parrafos anteriores. Debido al solapamiento entre algunas de estas funcionalidades (e.g. visualización de los datos detallados de los puntos de interés), alguno de los elementos está compartido entre varios módulos.

2.3.2. Interfaz de usuario

Como ya se ha comentado anteriormente, una de las principales características en las que debía estar basado el sistema, es la sencillez de la interfaz de usuario. Para ello, (basándonos en las en la guía de interacción persona-ordenador publicadas por Apple [8], se agrupó cada uno de los módulos que se han descrito con anterioridad en una pestaña de la aplicación visibles de manera constante que manejan la navegación principal por la aplicación (Fig. 2.5).

La figura 2.4 incluye el diagrama de navegación de toda la aplicación, en ella se pueden ver las pantallas más importantes de la misma, así como la forma de llegar a ellas.

[8] https://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG

Figura 2.4: Diagrama de flujo de la interfaz de usuario

A continuación describimos en mas detalle los elementos principales de los componentes de la aplicación, *Guía*, *Mapa*, *Escáner*:

Figura 2.5: Pantalla principal de la aplicación cliente y sus diferentes pestañas.

Pestaña Guía

Cuando abres la aplicación, se encuentra seleccionada por defecto la pestaña de *Guía* (Fig. 2.5), en esta pantalla encontramos un menú con las principales categorías de la guía turística que una vez seleccionadas nos llevan a una lista de los puntos de interés para dicha categoría. Debido a que no caben todas las categorías existentes en la pantalla, se ha creado un botón de *Otros* que te lleva a una lista con las categorías menos importantes.

Una vez seleccionado un punto de interés (Fig. 2.6), es posible acceder a la información multimedia asociada a este usando el boton *Ver video* asociado a este. Desde esta pantalla, es posible además abrir la aplicación de mapas del dispositivo para que esta nos de una ruta desde el punto en el que nos encontramos al punto de interés (Fig. 2.7).

Pestaña Mapa

La segunda pestaña en la que contiene el mapa en el que se localizan los puntos de interés como se puede ver en la figura 2.8. Si se pulsa en cada uno de los puntos de interés, se da la posibilidad de acceder a información más detallada sobre el mismo (la misma a la que se accedía en la pestaña

Figura 2.6: Pantallas de detalle de punto de interés y visualización multimedia

Figura 2.7: Ruta desde la localización actual al punto de interés elegido.

anterior). Además es posible el filtrar los puntos de interés por categorías para ver solo aquellos que nos interesen.

Pestaña Escáner

En la última pestaña (esta es opcional y no aparece en todas las guías) se encuentra el escaner de códigos QR. Una vez pulsado el botón de *escanear*, podremos ver en pantalla la imagen de la camara, y si aparece dentro de su rango algún código QR, se extraerá la información contenida en el mismo sin necesidad de pulsar ningún botón. Una vez que hayamos obtenido la información de un código correspondiente al sistema, usando el botón de *Más información* se obtiene la misma información sobre el punto de interés que en anteriores ocasiones.

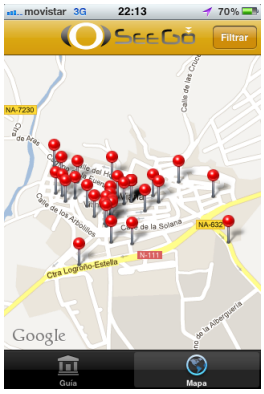

Figura 2.8: Pestaña de mapa.

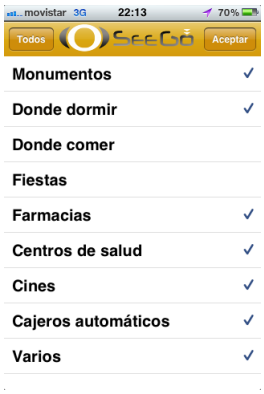

Figura 2.9: Pantalla de filtro por categoría

2.3.3. Plan de pruebas

En las reuniones en las que se definieron los requisitos que debería tener la aplicación, así como el calendario de hitos de la misma, se definió además un plan de pruebas que obligara a ambas partes a revisar la aplicación de forma periódica con el fin de tener el menor número de errores posible.

Para ello se definieron tres pasos fundamentales a seguir antes de cada demostración:

- La parte del *negocio* de la aplicación (la correspondiente al modelo en la arquitectura MVC) se testeo usando la técnica conocida como *unit testing*. Para ellos se diseñaron una serie de test que pueden ser ejecutados de forma automática cada vez que se realiza un cambio en esta parte. Se utilizó la herramienta Google Toolbox for Mac [9] para la

[9] http://code.google.com/p/google-toolbox-for-mac/wiki/iPhoneUnitTesting

realización de estas pruebas.

- Antes de entregar una versión para la realización de una demonstración el desarrollador se hacía responsable de la realización de una serie de pruebas que cubrieran los casos estandares que un usuario podría realizar con la aplicación.
- Un testeo más en profundidad del uso *habitual* de la misma, debía ser realizado por la empresa una vez terminada la demostración de modo que si se encontraban errores en la aplicación, estos pudieran ser reparados antes de la siguiente reunión.

Capítulo 3

Diseño y desarrollo del gestor web

Una vez finalizada la aplicación móvil, se consiguió una idea más exacta de lo que hacía falta para la parte del gestor web. La aportación de la empresa durante estas reuniones fue especialmente importante, ya que es la parte del sistema con la que ellos van a trabajar día a día y era fundamental que se encontraran cómodos con ella.

En los siguientes apartados se describe de forma detallada todo el proceso llevado a cabo para el desarrollo del gestor web. Esto incluye tanto la toma de decisiónes, como el diseño de la aplicación y la implementación de la misma. Durante todo el proceso además, se fueron intercalando periodos de pruebas de la aplicación.

3.1. Requisitos del cliente y decisiones de diseño adoptadas

Como ya sucedió con el diseño de la aplicación móvil, una de las principales características que se plantearon fue la necesidad de que la interfaz de usuario fuera lo más sencilla y autoexplicativa posible. Esto debería servir para que la curva de aprendizaje de los empleados / usuarios fuera menor y fueran realmente productivos desde un principio.

Así mismo, se incidió en que el sistema móvil debería ser lo más configurable posible desde esta plataforma, haciendo posible el hecho de no tener que cambiar nada de forma *manual* en el código de la aplicación móvil, a la hora de crear una nueva guía.

También se decidió que a pesar de que las guías debían ser multilenguaje, no era práctico el hecho de tener una aplicación con los cuatro idiomas que

se deseaban (español, alemán, francés e inglés) dentro de su base de datos interna, debido principalmente a que el tamaño de los elementos multimedia de una guía suele ser bastante elevado, con lo que si lo multiplicamos por cuatro (uno para cada idioma), esto hace que la guía alcance un tamaño prohibitivo (hay que recordar que la mayoría de las aplicaciones de este tipo son descargadas directamente en el dispositivo desde una tienda virtual, con lo que el tamaño de las mismas puede ser un factor a considerar por los usuarios a la hora de descargarse una aplicación o no). Esto incidía en el diseño que era necesario para el gestor web.

Desarrollo web. Se decidió por Ruby on Rails [1] como plataforma de desarrollo web debido a su flexibilidad, facilidad de programación y popularidad (las plataformas más populares, generan una comunidad mayor a su alrededor que hace que sea más fácil encontrar respuestas a los posibles problemas que se créen durante el desarrollo. Además existe una posibilidad mayor de que ya existan librerías desarrolladas que hagan que algunas tareas puedan ser abstraidas).

Base de datos. En cuanto a la base de datos donde almacenar la configuración e información específica de una guía, y debido a que el número de personas que van a usar la aplicación dentro de la empresa es reducido, se decidió usar un sistema de menor potencia del que es habitual en este tipo de sistemas (normalmente se usan sistemas como MySQL [2] o PostgreSQL [3], de gran potencia pero más difíciles de mantener) como es SQLite [4]. Durante las pruebas efectuadas, se estimó que SQLite puede soportar de sobra la carga a la que va a ser sometido desde la empresa.

3.2. Arquitectura, diseño y pruebas del gestor web

A lo largo de esta sección se va a explicar el diseño adoptado para el gestor web y la interfaz de usuario que se ha implementado para el mismo.

3.2.1. Interconexión del cliente con el servidor

A la hora de conectar los datos introducidos en el servidor con la aplicación cliente se pensó en dos posibilidades:

[1] http://www.rubyonrails.org
[2] http://www.mysql.com/
[3] http://www.postgresql.org/
[4] http://www.sqlite.org/

- El servidor puede servir los datos a través de Internet mediante servicios web. La aplicación obtendrá estos datos cada vez que se inicie.
- Creación de una base de datos en el servidor con todos los datos de una guía. El fichero que contenga la base de datos debe ser introducido antes de compilar la aplicación cliente para cargar todos los datos necesarios y generar la aplicación de móvil correspondiente.

Se descartó la primera opción debido a que, a pesar de la ventaja que supone el hecho de tener siempre los datos actualizados cada vez que se inicia la aplicación, la posibilidad de que la conexión a Internet sea insuficiente para poder descargarlos a una velocidad aceptable es bastante frecuente y por lo tanto hay entornos en los que es posible que la aplicación quede totalmente inutilizada.

3.2.2. Base de datos

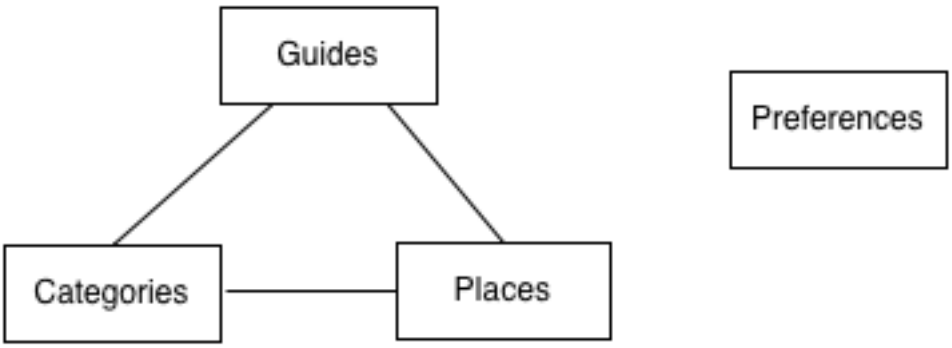

Figura 3.1: Tablas principales de la base de datos

Como se puede ver en la figura 3.1 el esquema de la base de datos utilizado es muy sencillo. En primer lugar tenemos una tabla *Guides* en la que encuentran todas y cada una de las guías generadas con el gestor web. En esta tabla encontramos campos como el nombre y descripción de la guía, además del lenguaje de la misma.

Existe también una tabla *Categories* en la que se almacenan las diferentes categorías de puntos de interés que existe en está guía en particular. Esta tabla tiene campos para el nombre y una imagen para la categoría y esta relacionada con la tabla de guías.

Los puntos de interés de una guía se almacenan dentro de la tabla *Places*. Esta tabla esta relacionada con la categoría y la guía a la que pertenece

y además tiene campos para almacenar los principales valores de un punto de interés como son nombre, descripcion, imagen, elementos multimedia, geoposicion, ...

Finalmente, existe una tabla llamada *Preferences* que almacena pares clave-valor con las preferencias que hacen que cambie la apariencia de la aplicación (color de las ventanas, existencia de escáner de códigos QR, ...)

3.2.3. Arquiterctura de la aplicación

Las aplicaciones web hechas en Ruby on Rails (y en general la mayoría de las aplicaciones web) se basan en el mismo patrón que hemos explicado antes para la aplicación móvil, el cual se conoce como modelo-vista-controlador (MVC). En este caso, tiene la ventaja de que es muy fácil el cambio de la interfaz de usuario sin tener que tocar la parte del código que describe la lógica de negocio.

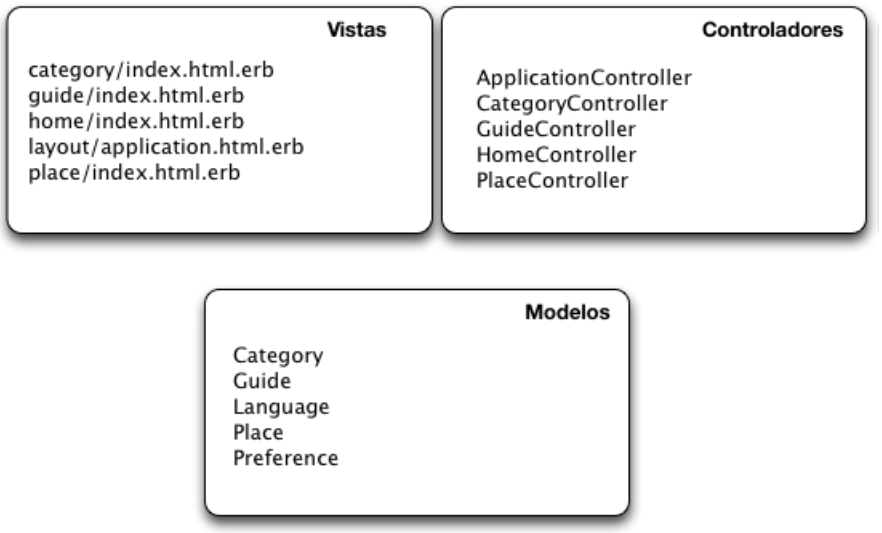

Figura 3.2: Diagrama de módulos del gestor web.

Dentro del módulo de vistas nos encontramos con todas los elementos que se encuentran en la carpeta *views* de la aplicación. Estos consisten en archivos de plantilla eRuby que son páginas HTML a las que se puede añadir cierto componente dinámico (en el apéndice D se explica un poco mejor su funcionamiento). Hay aproximadamente una plantilla (archivo RHTML) por cada una de las pantallas del gestor web.

En el módulo de controladores hay una clase por cada grupo de pantallas que se encuentran relacionadas, como se puede ver en la figura 3.2 estas

son una para cada uno de los elementos principales, ademas de uno para la pantalla inicial (HomeController) y otro del cual extienden el resto que sirve para implementar las funciones comunes a todos (ApplicationController):

- ApplicationController
- CategoryController
- GuideController
- HomeController
- PlaceController

En cuanto al módulo de modelos, al ser estas clases que se identifican de forma principal con las tablas de la base de datos, esta compuesto de una clase por cada tabla de la base de datos en la que hay aparte de cada uno de los datos de dicho elemento, aquellas funciones que manipulan estos datos:

- Category
- Guide
- Language
- Place
- Preference

3.2.4. Interfaz de usuario

Durante esta sección, se va a describir la interfaz de usuario que se ha desarrollado para el gestor web. En las figuras 3.3 y 3.4 se puede ver el diagrama de flujo entre las pantallas principales y como se mueve el usuario entre unas y otras. A continuación se describe en más detalle los distintos componentes/pantallas de la interfaz resumidos en dichas figuras:

- Cuando el usuario entra en la aplicación, le aparece una lista de las guías que se han creado hasta el momento en la aplicación y botón para crear una nueva guía (pantalla número 1 de Fig. 3.3). Una vez seleccionada la guía con la que quiere trabajar, le aparece una pantalla con varias pestañas entre las que podrá elegir para ver y editar los diferentes elementos de la guía.

- La primera de estas pestañas es la de los datos generales de la guía (pantalla número 2) donde se puede editar el nombre y la descripción de la guía.

- En la segunda pestaña, la que tiene por título *Categorías* (pantalla número 3), se pueden crear y editar las diferentes categorías de la aplicación. Para cada una de ellas se puede dar tanto un nombre como una imagen que será la que aparezca en la aplicación móvil. Para editar una categoría, se usa la pantalla número 6.

- La pantalla número 4 corresponde con la lista de lugares pertenecientes a la guía. Junto con la pantalla número 7 se usan para crear y editar nuevos puntos de interés de la guía. Para un lugar se pueden editar una serie de datos como nombre, descripción, geolocalización, imagen que aparecerá en pantalla, ... También existe el botón de *Código QR* que genera la imagen de un código con los datos del lugar que podrá ser utilizado después con el escáner de la aplicación móvil.

- La última pantalla que aparece es la pantalla de las preferencias de la aplicación (pantalla número 5). En esta pantalla se pueden modificar ciertas configuraciones de la aplicación móvil como el icono de la aplicación, el fondo de pantalla, ...

3.2.5. Plan de pruebas

Al tener ya experiencia en el trabajo con la empresa en la parte de la aplicación móvil y debido a que el sistema de pruebas adoptado había funcionado bastante bien, se decidió el usar un acercamiento parecido para las pruebas de esta parte del sistema:

- Pruebas unitarias para la parte de negocio (*modelo*) de la aplicación. Es este caso no hubo que utilizar ninguna librería externa, ya que Ruby on Rails cuenta con soporte integrado para la realización de este tipo de tests [5].

- Pruebas estándares por parte del desarrollador antes de cada reunión.

- Pruebas más exhaustivas por parte de la empresa después de cada reunión con el objetivo de tener solucionados los errores antes del próximo ciclo.

[5] http://guides.rubyonrails.org/testing.html

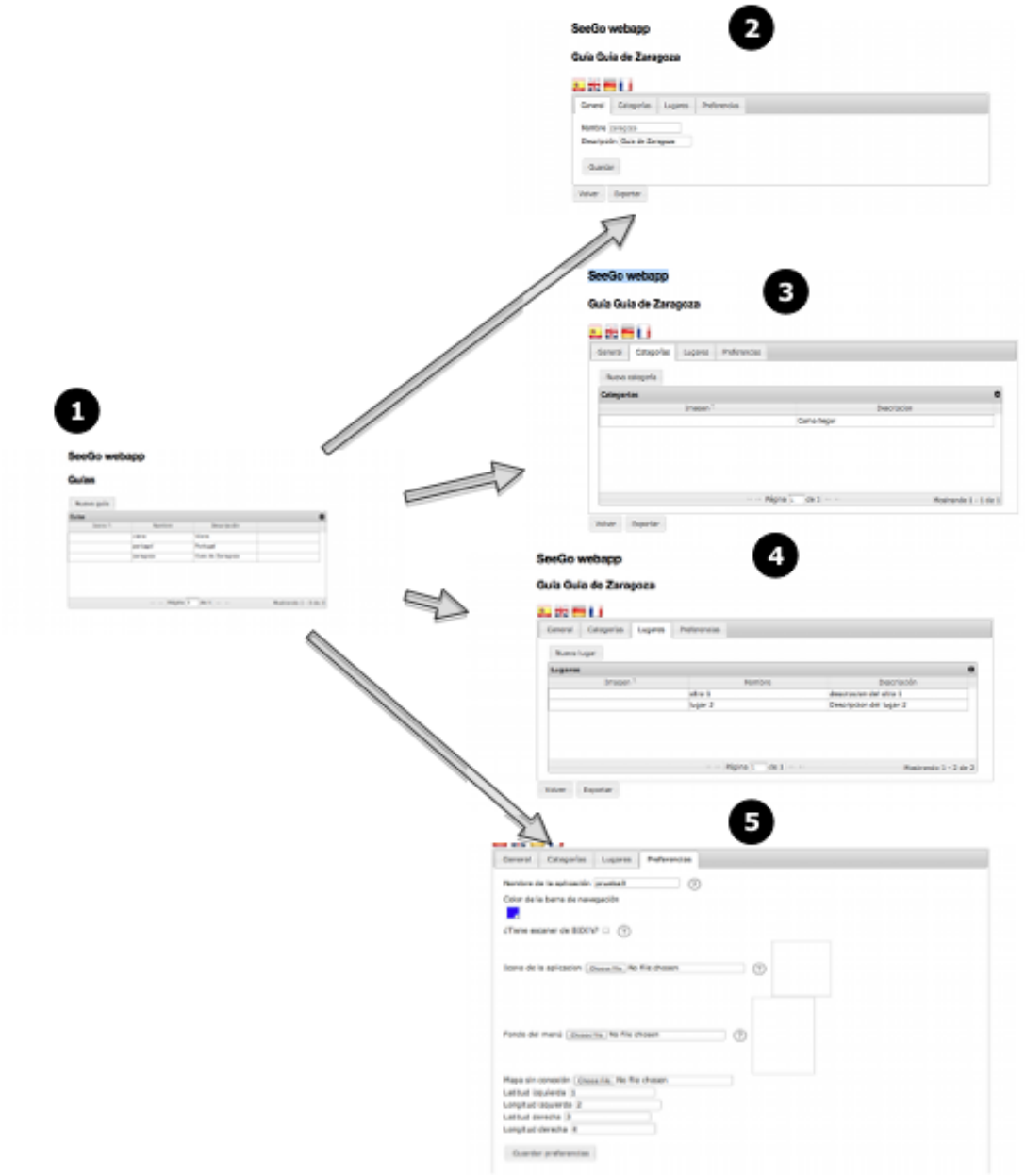

Figura 3.3: Interfaz de usuario web (I).

Figura 3.4: Interfaz de usuario web (II).

Capítulo 4

Conclusiones y trabajo futuro

Como se ha comentado anteriormente este proyecto se ha desarrollado junto a la empresa Disline. Sus principales objetivos al comienzo del proyecto eran la realización de una plataforma software que les permitiera entrar en un nuevo mercado dentro de su negocio orientado al turismo y realización de guías turísticas. Dicha plataforma debe ser de fácil uso por parte de sus empleados y flexible (de forma que no hubiera que ser totalmente dependiente del desarrollador inicial a la hora de crear nuevas guías turísticas). Por otro lado, la solución aportada debía ser lo suficientemente potente para que pudiera acomodar diferentes tipos de guías que pudieran interesar a diferentes clientes (guías orientadas a monumentos en exteriores, guías para interiores, ...).

En una primera fase de estudio sobre cómo son las aplicaciones similares disponibles se decidió que técnicas se iban a incluir en este trabajo (reconocimiento con QR, autolocalización con GPS, ...) y cuales definen posibles líneas de trabajo futuro (detalladas al final de esta sección).

Para cumplir el primer requisito, la parte que tiene que ser manipulada por los empleados de la empresa se basó en una plataforma web muy sencilla, en la que los empleados prácticamente ni necesiten hacer uso de un manual de usuario. Al estar la mayoría de las personas ya habituadas a trabajar con aplicaciones web en su día a día (Gmail, Facebook, ...) la transición a introducir los datos de los puntos de interés de esta forma resulta bastante fácil.

En cuanto a la flexibilidad de la plataforma, se ha creado un sistema que permite que únicamente haya que sustituir un archivo perteneciente a la base de datos (generado automático desde la platarforma web) y adjuntar los ficheros multimedia correspondientes (vídeos y fotografías de los elementos de la guía) para tener una nueva guía. Es verdad que hay que compilar la nueva aplicación antes de subirla al mercado de aplicaciones correspondiente,

pero este es un trabajo fácilmente realizable por cualquier profesional del desarrollo de aplicaciones informáticas y que apenas conlleva tiempo por lo que hace que la empresa no sea dependiente de nadie externo a la misma.

Debido a que el mercado principal de la empresa son las guías turísticas para pequeños municipios y aquellas destinadas museos que se encuentran es dichas localidades, se ha diseñado un sistema de identificación de puntos de interés que pueda ser útil tanto en interiores como en exteriores. Para la localización de puntos de interés en exteriores, se ha usado un mapa en el cual el usuario se puede mover e ir buscando lo que le interesa. Para cuando el usuario se encuentra en interiores sin embargo, la creación y lectura de códigos QR, hace que el usuario pueda saber ante lo que se encuentra con facilidad y conseguir la información deseada sobre ello.

Debido a todo lo expuesto anteriormente, se puede concluir que se han conseguido implementar los requisitos, expuestos al principio del proyecto por la empresa, de forma satisfactoria y que la aplicación será de gran ayuda para un mayor desarrollo del negocio en el que se encuentran inmersos. A lo largo del proyecto también se han estudiado las posibles mejoras que podrían ser interesantes en trabajos o estudios futuros. A continuación se describen las tres lineas principales de trabajo futuro analizadas:

Reconocimiento de objetos mediante técnicas de visión por computador sin marcadores artificiales

Se dejó como trabajo futuro o complementario, el desarrollo prototipos para reconocimiento de objetos (cuadros, monumentos,...) sin necesidad de utilizar marcadores. Estas técnicas tienen la ventaja de no necesitar una instalación añadida a lo ya existente con el correspondiente gasto y mantenimiento que ello supone. Ademas, permitirían que mediante actualizaciones de software se puedan añadir nuevos objetos al catálogo de la aplicación, mientras que si usamos algun tipo de marcador artificial (como los códigos QR por los que se ha optado finalmente en este trabajo) para cada uno de los objetos que se quieran añadir, hay que realizar algún tipo de instalación en el mundo real con lo que no es tan inmediato. Sin embargo este tipo de desarrollo implicaba un trabajo de investigación y desarrollo de nuevas técnicas, equivalente a un proyecto completo. Por ello, para este prototipo comercial se opto por la opción de reconocimiento visual mas madura y robusta disponible (los códigos QR), dejando la opción descrita como trabajos futuros y de investigación.

Localización de puntos de interés sin necesidad de conexión a Internet

Otra posible mejora de la solución presentada, consiste en el desarrollo de un sistema de localización que no requiera de la conexión a Internet, para permitir guiado y localización de puntos de interés en entornos donde es difícil obtener una conexión a Internet de buena calidad. La localización de puntos de interés usando los mapas de Google Maps que es totalmente adecuado dentro de nucleos urbanos, se torna inusable en entornos remotos o de montaña: a pesar de poder obtener la localización GPS del usuario, la licencia de uso de los mapas de Google impide la descarga de estos en el dispositivo móvil para su posterior uso sin conexión.

La solución más evidente a este problema es añadir a la aplicación móvil la posibilidad de usar otro tipo de mapas que puedan se accesibles de forma *offline*. Para ello y después de pensar las diferentes posibilidades, se pensó que la solución más general y que mayor facilidad de uso aportaría a la empresa es la posibilidad de cargar en el dispositivo una imagen con el mapa y además añadir como información a la guía las coordenadas de localización de dos puntos de la misma (un par de esquinas de la imagen es posiblemente lo más comodo). De este modo, se pueden interpolar el resto de las localizaciones que se quieran poner en dicho mapa y el usuario no tendra ningún problema cuando transite por lugares en los que la conexión no sea del todo buena.

Portabilidad de la aplicación a otras plataformas

Una de las principales características que se deseaba que tuviera el sistema cuando se iniciaron las reuniones que se realizo la toma de requisitos, era que el sistema fuera accesible por el mayor número de personas posibles. Esto pasa principalmente porque el cliente móvil se pueda utilizar en el mayor número de plataformas móviles en el que sea posible. En la practica, esto se reduce a las cuatro pincipales que copan aproximadamente hasta un 95 % de la cuota de mercado como son Apple iOS, Google Android, RIM Blackberry OS y Microsoft Windows Mobile. Debido a que la portabilidad de la aplicación excede las posibilidades de un PFC, se decidió inicialmente la creación del cliente para una de las más populares (Apple iOS) dejándose para un futuro la realización del cliente en el resto de plataformas.

Durante las fases finales del proyecto y en paralelo a la creación del gestor web, el autor de este proyecto ha estado dando apoyo necesario a otro desarrollador que se encargará de la portabilidad a Android para que sea totalmente compatible con todo lo desarrollado a lo largo de estos meses. En el momento de redactar la memoria de este PFC el desarrollo de la portabilidad

estaba practicamente terminado con lo que seguramente ambas plataformas llegarán al mercado al mismo tiempo.

En cuanto a las dos plataformas restantes (Blackberry y Windows Mobile), en el futuro se seguirá con detenimiento la evolución de ambas plataformas (así como otras que pudieran aparecer con el tiempo y alcanzar una buena posición en el mercado), para comprobar si es factible y rentable para la empresa el expandir su oferta con nuevos desarrollos software.

Bibliografía

[1] Yue Liu, Ju Yang, Mingjun Liu, *Recognition of QR Code with mobile phones* Control and Decision Conference, pags. 203 - 206, 2008

[2] Yu-Hsuan Chang, Chung-Hua Chu, Ming-Syan Chen, *A General Scheme for Extracting QR Code from a Non-uniform Background in Camera Phones and Applications*, Ninth IEEE International Symposium on Multimedia, pags. 123 - 130, 2007

[3] Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool, *SURF: Speeded Up Robust Features*, Computer Vision and Image Understanding (CVIU), Vol. 110, No. 3, pp. 346–359, 2008

[4] Herbert Bay, Beat Fasel, Luc Van Gool, *Interactive Museum Guide: Fast and Robust Recognition of Museum Objects*, 2006

[5] International Organization for Standardization.; International Electrotechnical Commission, *Information technology – automatic identification and data capture techniques – bar code symbology – QR code*, 2000

[6] Bryan O'Sullivan, *Mercurial: The Definitive Guide*, O'Reilly Media, 2009

[7] Roy Thomas Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, Univeristy of California (Irvine), 2000

[8] Robert C. Martin, *Agile Software Development, Principles, Patterns, and Practices*, 2002

[9] Alistair Cockburn, *Agile Software Development: The Cooperative Game*, 2nd Edition, 2006

[10] Sam Ruby, Dave Thomas, David Heinemeier Hansson, *Agile Web Development with Rails (Pragmatic Programmers)*, 4th Edition, 2011

[11] Obie Fernandez, *The Rails 3 Way*, 2nd Edition, 2010

Apéndices

Apéndice A

Diagrama temporal de desarrollo

Como se ha ido explicando a lo largo de este documento el trabajo en este proyecto se ha dividió en tres partes (cuyo desarrollo temporal se puede ver en la figura A.1):

- Primero se realizaron una serie de estudios sobre las técnologías a utilizar, tanto de desarrollo como de reconocimiento de objetos mediante técnicas de visión por computador.
- Después de los estudios, se realizó la aplicación móvil, empezando por la toma de requisitos y el diseño de la aplicación y finalizando con la implementación que se realizo junto con las pruebas del sistema.
- En tercer lugar se desarrolló el gestor web. Este desarrollo tuvo aproximadamente los mismos pasos que el anterior.

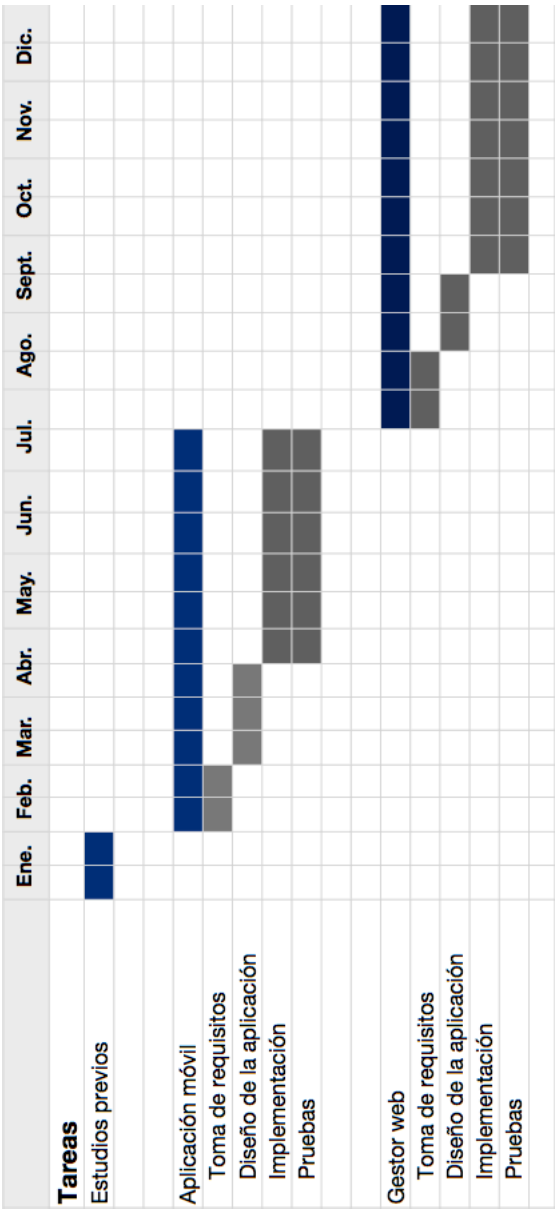

Figura A.1: Diagrama temporal del desarrollo

Apéndice B

Metodologías de desarrollo ágiles

Las metodolgías de desarrollo ágiles está compuestas por una serie de conceptos que nacieron como contraprestación a la forma de desarrollo tradicional empleada, conocida como desarrollo en cascada. En el desarrollo en cascada, todo el diseño se hace al principio del proyecto de forma que se planifica hasta el último detalle del mismo con la intención de hacer toda la implementación en una sola iteración. Por contra, en las metodologías ágiles, se entiende que los requisitos del producto a desarrollar van a ir cambiando a lo largo del tiempo y que por lo tanto no es posible conocer con antelación lo que el cliente espera del producto o no será posible que el cliente nos transmita dicho conocimiento de forma eficaz sin ir viendo antes partes del producto ya terminado.

Es por todo lo anterior que las metodologías ágiles proponen una interacción mayor entre clientes (entendiendo estos como los usuarios finales del producto) y desarrolladores. Para ello, se hacen ciclos de desarrollo más cortos y demostraciones al cliente al final de cada ciclo de forma que se puedan ir haciendo las modificaciones que se estimen adecuadas para que el producto cumpla con la funcionalidad deseada por el cliente de una forma más eficaz.

Otra de las diferencias básicas se encuentra en la forma de probar los productos. Mientras que en la forma de desarrollo clásica, se reservaba un espacio practicamente al final del proyecto en el cual se hacia un testeo completo de la aplicación, en estas nuevas metodologías se va testeando conforme se va desarrollando (incluso hay veces que se crean tests automatizados antes incluso de la implementación de una determinada funcionalidad, ejerciendo estos como especificación de dicha funcionalidad. Esto es conocido como *Test Driven Development (TDD)*.

Aunque hay veces que se acusa a las metodologías ágiles de ser totalmente

anárquicas, en realidad estas usan una serie de metodos formales para, por ejemplo, medir la velocidad de desarrollo del equipo del proyecto. De esta forma se puede estimar el coste en tiempo de las futuras funcionalidades a desarrollar de una forma mas certera no teniendo que hacer conjeturas al principio del proyecto sobre como va a funcionar el equipo, ya que la comunicación entre los diferentes miembros del equipo (una de las cosas que enfatizan este tipo de metodologías) es muy importante de cara a no perder tiempo de desarrollo debido a malentendidos entre dos o más miembros.

Hay una serie de métodos ya establecidos como por ejemplo:

- Extreme Programming (XP).
- Scrum.
- Kanban.
- Agile Unified Process (AUP).

aunque en la práctica, la mayoría de equipos de desarrollo acaban implementando una mezcla de todos ellos (hecho conocido como *tailoring*) de forma que obtengan las características con las que se encuentren más comodos de cada uno. Alguno de estos métodos (e.g. Extreme Programming) incluso incluyen el *tailoring* como parte del método en si mismo, con lo que fomentan la práctica del mismo.

Para más información sobre este tema, existen multitud de libros y artículos entre los que se incluyen:

- Agile Software Development, Principles, Patterns, and Practices [8].
- Agile Software Development: The Cooperative Game [9].

Apéndice C

Plataforma de desarrollo iOS

iOS es el sistema operativo que se encuentra dentro de los dispositivos iPhone, iPod Touch e iPad. Esta plataforma esta fuertemente influenciada por el sistema de desarrollo de OSX al ser ambos pertenecientes a la misma compañía.

Para desarrollar aplicaciónes para iOS, se usa entorno de desarrollo (IDE) Xcode, el cual provée de todas las herramientas necesarias para el diseño de la interfaz de la aplicación y para escribir código para la misma. En este apéndice se va a explicar las herramientas de desarrollo básicas de las que se compone el entorno. Para más información sobre el desarrollo de aplicaciones el mejor lugar es la página del desarrollador de Apple [1].

Figura C.1: Entorno de desarrollo Xcode (Inteface Builder, editor de código, App Store, dispositivo).

[1] https://developer.apple.com/iphone

Entorno de desarrollo

Para el desarrollo de una aplicación iOS, se empieza creando un proyecto en la aplicación Xcode. Un proyecto gestiona toda la información asociada con la aplicación, incluyendo los archivos de código fuente, los diseños de la interfaz y todas aquellas preferencias y propiedades necesarias para construir la aplicación. El trabajo en el proyecto se hace a traves de la llamada *workspace window*, la cual provee acceso rápido para todos los elementos que componen la aplicación (Fig. C.2).

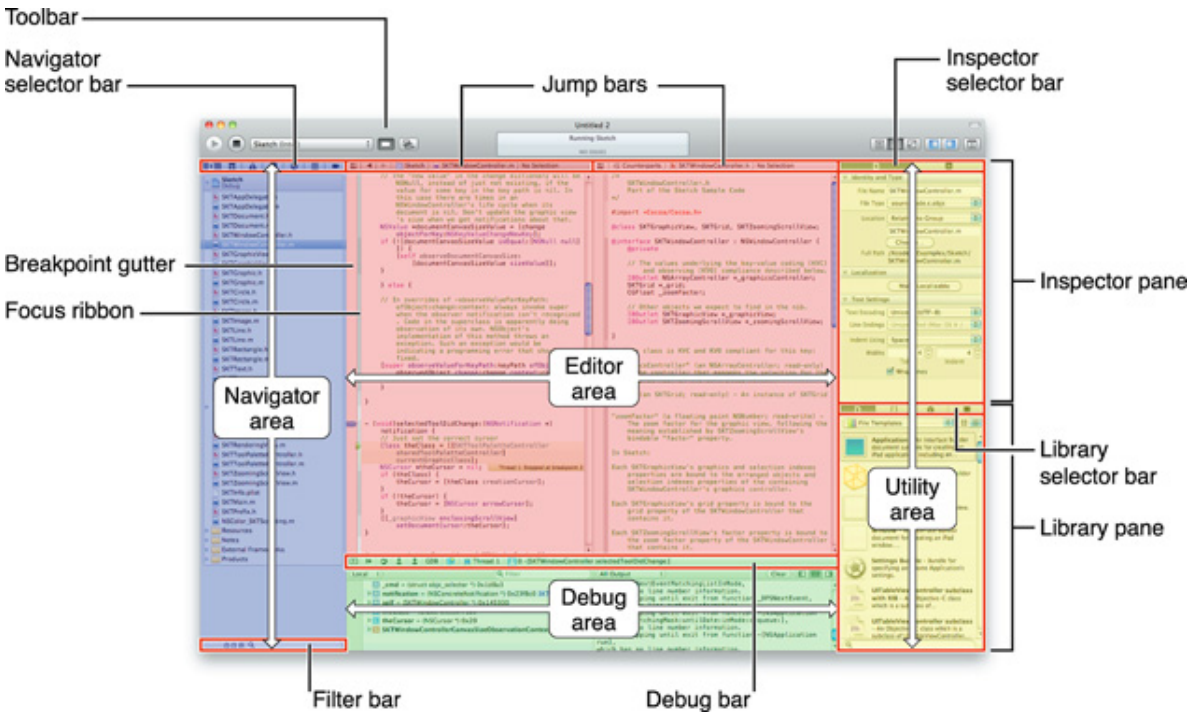

Figura C.2: Esquema de la workspace window.

Esta ventana de trabajo esta dividia en cuatro áreas principales: el área de navegación, el área de edición, el área de debug y el area de utilidades.

- *El área de navegación* es donde se gestionan los archivos del proyecto, y otra información como símbolos, breakpoints, hilos de la aplicación, pilas de ejecución, errores y logs de las actividades efectuadas por el usuario.

- *El área de edición* es donde el usuario edita los archivos del proyecto, diseña la interfaz, configura las propiedades del proyecto y ve todo tipo de información acerca del proyecto.

- *El área de debug* se utiliza cuando el usuario esta probando la aplicación (bien sea en un simulador o directamente en un dispositivo). Se utiliza para ver el contenido de las variables de la aplicación y la salida por consola de la misma. Es posible introducir ordenes del debugger en esta área.

- *El área de utilidad* se utiliza para configurar las propiedades de un objeto o archivo de la aplicación. También es posible ver es este area los recursos asignados al proyecto.

Xcode además provée ayuda contextual de forma que se puede acceder a ella desde el elemento para el cual se necesita ayuda.

Edición de archivos de código fuente

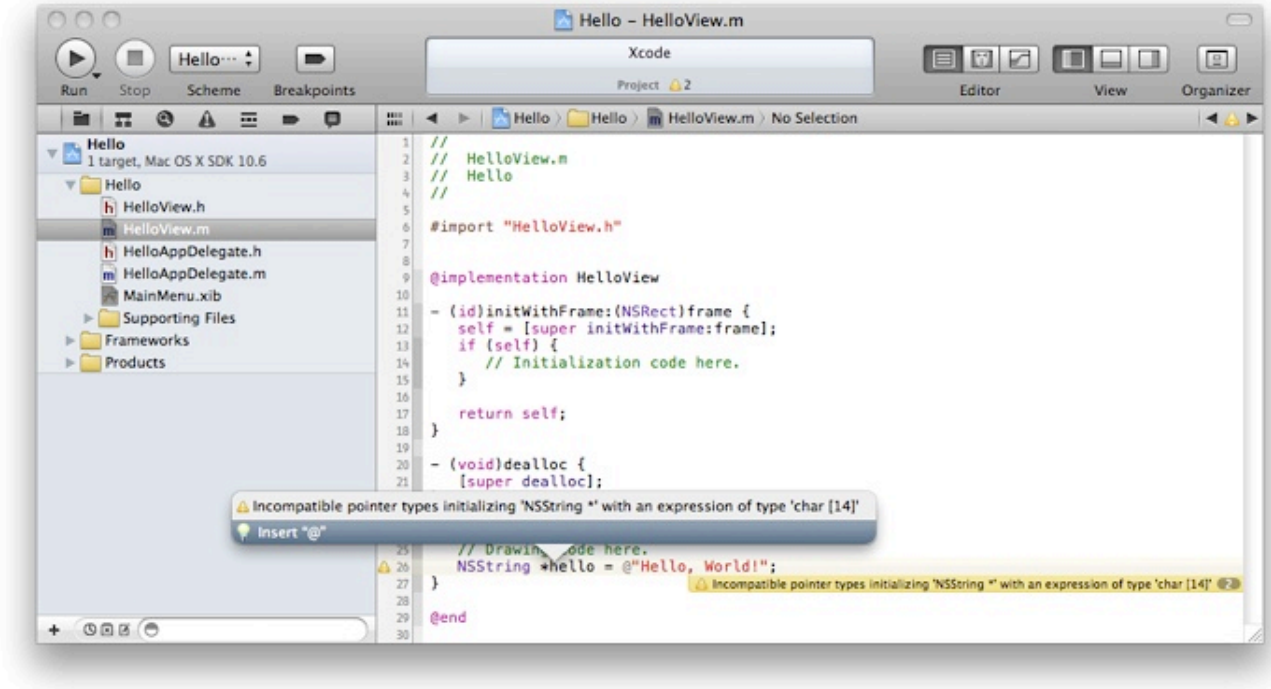

Figura C.3: Xcode mostrando un error y sus posibles soluciones.

Para facilitar al usuario el trabajo con el código fuente, este implementa características como las sugerencias de código, indentación automática dependiendo del contexto, plegado de código (ocultación de partes del código de forma temporal). Además provée información de todos los símbolos del código directamente en el mismo.

Además Xcode va analizando el código conforme se escribe de forma que es capaz de detectar errores y realizar sugerencias sobre como solucionar los mismos.

Diseño de la interfaz de usuario

La herramienta *Interface Builder* se encuentra disponible dentro del entorno para la creación y edición de la interfaz de usuario de la aplicación usando objetos predeterminados. Estos objetos incluyen ventanas, controles (campos de texto, botones, ...) y vistas (agrupaciones reutilizables de otros elementos) que se usan para representar la información de la aplicación.

Con este editor se posicionan los objetos, configuran sus propiedades y se establecen relaciones entre los mismos y con los archivos de código fuente de forma que el flujo de la aplicación sea el correcto.

El editor guarda los diseños en unos documentos llamados *archivos nib*, que contienen toda la información que el sistema operativo necesita para reconstruir la aplicación en tiempo de ejecución. El hecho de que estos archivos se puedan editar de forma visual hace que el usuario pueda ver en todo momento como va quedando la interfaz sin necesidad de recurrir a probar la aplicación.

Probar la aplicación

Xcode proporciona dos formas de probar la aplicación y eliminar errores, el simulador iOS y directamente dentro de un dispositivo. Usando el simulador, el desarrollador se puede hacer una idea de como funciona la aplicación y solucionar errores de forma rápida. Una vez que se este satisfecho del funcionamiento básico de la aplicación se puede probar la aplicación en un dispositivo conectado a Xcode, de esta forma se pueden detectar problemas relacionados con la memoria consumida por la aplicación y otros más sutiles como la forma en la que se maneja la aplicación en la pantalla táctil.

Mejorar el rendimiento de la aplicación

Una vez que el desarrollador ha visto que no hay problemas que impidan el funcionamiento normal de la aplicación el siguiente paso en el desarrollo de una aplicación para iOS es el uso de la aplicación *Instruments* para asegurarse que la aplicación no tiene elementos que hagan que se ejecute de forma más lenta de lo necesario. Manejando la aplicación en un dispositivo conectado a *Instruments* hace que podamos ver gráficas que nos resumen el consumo de recursos de nuestra aplicación (uso de memoria, actividad de disco, actividad de red, ...).

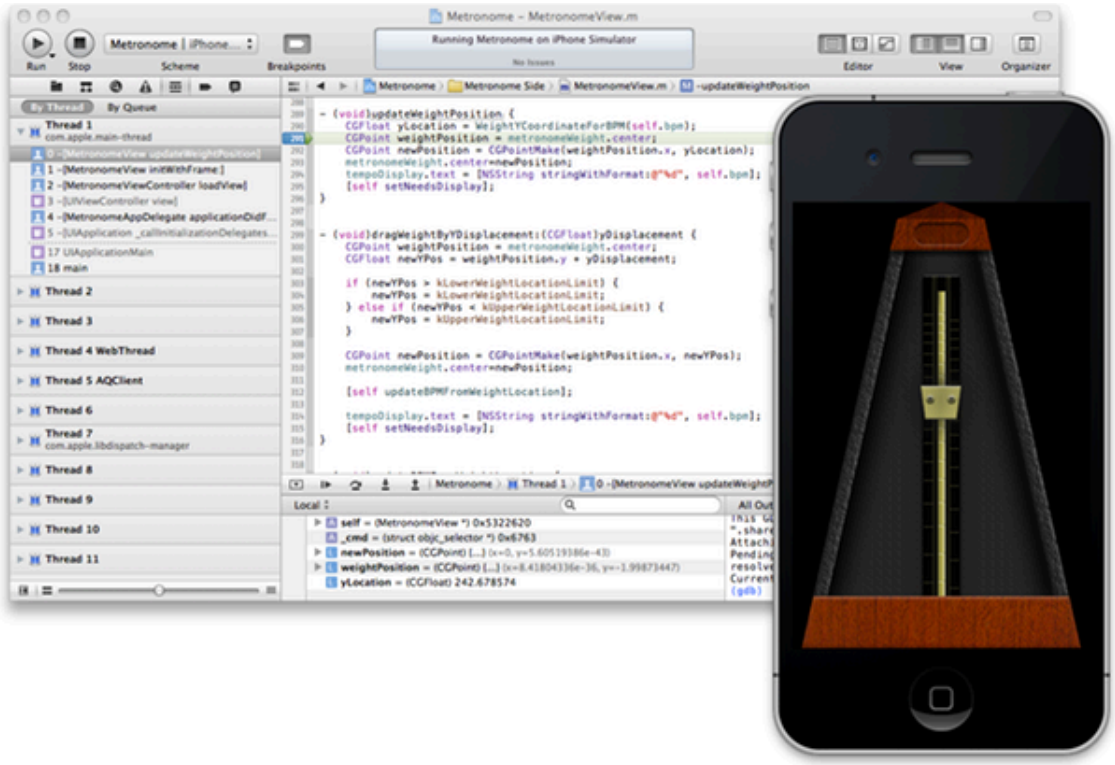

Figura C.4: Aplicación corriendo en el simulador de iOS.

Distribución de la aplicación

Xcode sirve además para empaquetar la aplicación de forma que sea fácil publicarla tanto para los posibles probadores externos que tengamos en el equipo de desarrollo como en el *App Store* (tienda virtual donde se distribuyen y venden aplicaciónes para dispositivos iOS). Entre otras cosas Xcode hace una serie de pruebas para determinar que no falta ningún elemento necesario para la publicación de la aplicación (iconos, propiedades, ...).

Apéndice D

Plataforma de desarrollo Ruby on Rails

Rails es un *framework* de desarrollo de aplicaciones web escrito en el lenguaje de programación Ruby. Esta diseñado para facilitar el desarrollo de aplicaciones haciendo suposisiones sobre lo que el desarrollador puede estar pensando en cada momento. Con esto se consigue que el desarrollador escriba menos código dedicado a encajar la aplicación con el entorno de desarrollo y más sobre la funcionalidad de la aplicación en si misma.

Este *framework* entra dentro de lo que se conoce como *software* de opinión. Esto quiere decir que Rails asume que hay una manera que es la mejor para hacer cierto tipo de cosas y por ello intenta obligar al desarrollador a hacerlo de esta forma. Para ello hace que intentar hacerlo de otra forma suponga un trabajo extra que haga que los desarrolladores no esten dispuestos a asumirlo. La filosofía Rails incluye varios principios:

- *Don't Repeat Yourself (DRY)* implica que escribir el mismo código una y otra vez es un mal hábito.
- *Convenciones sobre configuraciones* significa que el *framework* hace suposiciones sobre como se tienen que hacer las cosas en vez de dejar en manos del desarrollador el hecho de usar múltiples archivos de configuración.
- *REST* es un patrón de diseño para aplicaciones web que usa los recursos y los verbos HTTP (*GET, POST*) para organizar la aplicación.

Arquitectura MVC

Como ya se ha comentado en algun cápitulo de esta memoria, Rails esta basado en una arquitectura llamada modelo-vista-controlador (MVC) cuyos

beneficios principales son:

- Aislamiento de la lógica de negocio y la interfaz de usuario.
- Facilidad del mantenimiento del código mediante el mencionado anteriormente *DRY*.
- Dejar claro donde tiene que ir cada tipo de código haciendo que el desarrollador tenga que tomar menos decisiones.

Modelos: Un modelo representa la información (datos) de la aplicación y las reglas con las que es posible manipular estos datos. En el caso de Rails, se usan principalmente para almacenar información en una base de datos y para manipular dicha información. La mayoría de la lógica de negocio de la aplicación debería estar dentro de estos elementos.

Vistas: Las vistas representan la interfaz de usuario de la aplicación. Normalmente son páginas HTML con código Ruby incrustado para obtener los datos que hemos recogido en el resto de la aplicación. La principal función de estos elementos es la de proveer la información requerida por el usuario (u otros tipos de agentes) y mostrarsela de forma que les resulte conveniente.

Controladores: Los controladores son el *pegamento* entre las vistas y los modelos. Obtienen las peticiones del navegador web, interrogan a los modelos para obtener los datos necesarios y devuelven los datos a las vistas de nuevo para que la información sea mostrada al usuario.

Componentes de Rails

Rails en si mismo, tiene una arquitectura modular en la cual varios componentes que realizan tareas individuales (y que podrían ser usados de forma aislada en otros sistemas informáticos) se juntan para componer el *framework*. Estos componentes son:

- *Action Pack (Action Controller, Action Dispatch, Action View):* Librería que contiene la parte *VC* de MVC. Convierte las plantillas de las vistas en código HTML, redirecciona las peticiones HTTP al controlador adecuado, extrae los parametros de las peticiones y gestiona las sesiones de usuario.
- *Action Mailer:* Librería para la construcción de servicios de correo electrónico. Se puede utilizar para la recepción y manejo de correo entrante, y también para el envío de correos basados en plantillas.

- *Active Model:* Interfaz entre los servicios que provee Action Pack y los datos que proporciona Active Record (libreria de objetos usada por defecto en Rails). El uso de esta capa extra ayuda a usar otras librerías de objetos si el desarrollador lo encuentra conveniente.

- *Active Record:* Librería de objetos que provee independencia de la base de datos utilizada, funcionalidad CRUD (crear, leer, actualizar, borrar), multiples formas de búsqueda de objetos y relaciones entre los diferentes objetos de la aplicación.

- *Active Resource:* Librería para gestionar la conexión entre los objetos de negocio y los servicios web REST, que facilita la creación de funcionalidad CRUD basandose en verbos HTTP.

- *Active Support:* Conjunto de clases y librerías de utilidad, que son usadas en diferentes puntos del *framework*.

- *Railties:* Código que consigue juntar todos los componentes anteriores de forma que el usuario los vea como un todo.

REST

La abreviatura REST quiere decir *Representational State Transfer* y son los fundamentos básicos de lo que se conoce como arquitecturas RESTful. Estas fueron descritas por primera vez en la tesis doctoral de Roy Fielding [7] y que llevados al contexto de una aplicación Rails se pueden resumir en dos principios:

- Usar los identificadores de recursos HTTP (URLs) para identificar los recursos (datos) de la aplicación.

- Transferir cambios de estado de dichos recursos entre los componentes del sistema.

Por ejemplo, la siguiente petición HTTP:

```
DELETE /photos/17
```

sería entendida por la aplicación como que hay que borrar el objeto *photo* que tenga como identificador el numero 17. Para más información sobre este tipo de arquitecturas, se puede leer el tutorial *A Brief Introduction to REST* [1] de Stefan Tilkov.

[1] http://www.infoq.com/articles/rest-introduction

Generación de código

Uno de los principios básicos sobre los que se contruye Rails y a lo que debe una parte importante de su éxito es la generación de código y *scaffolding*. Mediante una serie de ordenes de linea de comandos sencillas, es posible contruir funcionalidad CRUD sin ningún tipo de trabajo adicional. Si esta funcionalidad básica no se ajusta totalmente a las necesidades del desarrollador, es muy sencillo el personalizar el código generado resultando aun así en un decrecimiento del tiempo de desarrollo.

Esta funcionalidad hace que sea muy fácil el hacer pruebas para comprobar el funcionamiento de la aplicación sin tener que esperar a que este todo desarrollado.

Más información

Para más información sobre el *framework* de desarrollo Rails, se puede visitar la documentación oficial [2] o leer alguno de los dos libros clásicos sobre el tema:

- Agile Web Development with Rails [10].
- The Rails 3 Way [11].

[2] http://guides.rubyonrails.org/

Apéndice E

Control de versiones con Mercurial

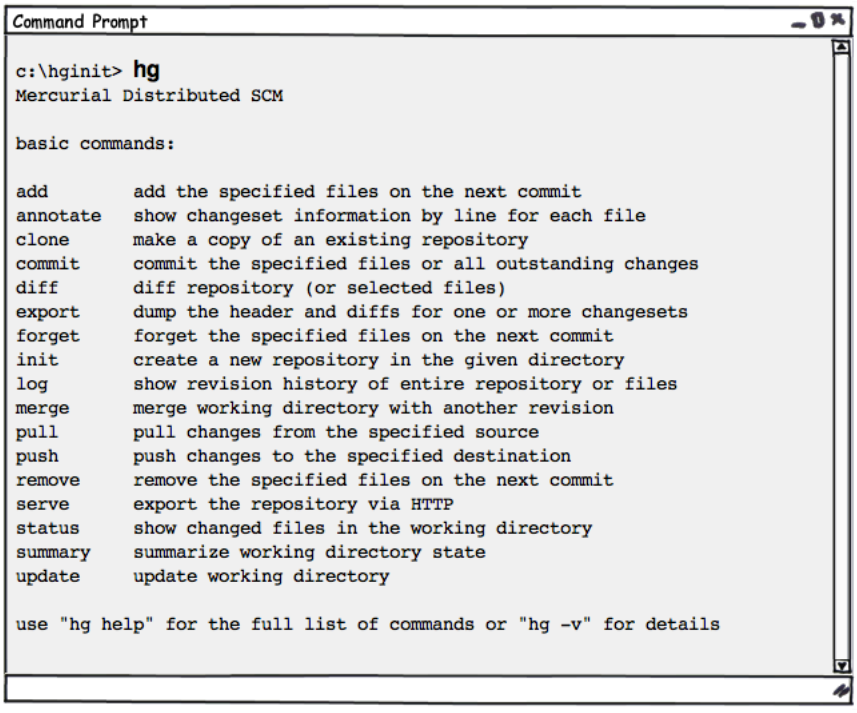

Figura E.1: Salida de la orden hg sin ningún argumento.

Mercurial es un sistema de control de versiones usado por los desarrolladores para gestionar el código fuente de una aplicación. Sus dos principales propositos son:

- Conservar cada uno de los cambios realizados en versiones viejas de cada archivo.
- Unir diferentes versiones de un mismo código de forma que varios desarrolladores puedan trabajar en paralelo en el código para despues mezclar sus cambios.

La forma de trabajo principal con Mercurial es a través de la linea de comandos, la cual funciona en sistemas Windows, Unix y Mac. El comando para Mercurial se llama *hg*.

Para obtener todas las ventajas de un sistema de control de versiones, se necesita un repositorio. Un repositorio almacena todas las versiones antiguas de cada uno de los archivos del código fuente. En realidad, para no utilizar demasiado espacio, no almacena directamente estas versiones, sino únicamente los cambios producidas sobre ellas.

En otros sistemas de control de versiones, se necesitaba de la instalación del sistema en un servidor central, pero Mercurial es lo que se conoce como sistema de control de versiones distribuido. Esto significa que en su versión más básica solo es necesario instalarlo en tu propia máquina.

Para la creación de un repositorio, lo único que hay que hacer es ir al directorio donde se almacena el código de la aplicación y usar la orden *hg init*.

```
/home/juanjo> cd Proyecto
/home/juanjo/Proyecto> hg init
```

Esta orden creará un nuevo directorio oculto llamado *.hg* en donde se almacena toda la información del repositorio. El contenido de este directorio no debe ser nunca manipulado de forma directa, únicamente a traves del comando *hg*.

Para añadir nuevos archivos al repositorio y que Mercurial sepa sobre que archivos tiene que guardar información, se usa la ordden *hg add*.

```
/home/juanjo/Proyecto> hg add
adding Prueba.rb
adding ...
```

Una vez que se han añadido los archivos a un repositorio hay que hacer un *commit* de los cambios. La primera vez que lo hagamos, lo que hará sera añadir el contenido de todos estos archivos (para Mercurial, al principio los archivos carecían de contenido). Lo normal es usar el argumento *-m* para añadir un mensaje a este grupo de cambios y sepamos de que estamos hablando en el futuro.

```
/home/juanjo/Proyecto> hg ci -m ''Commit inicial''
```

Una vez que ya tengamos algunos cambios, se puede usar la orden *hg log* para ver el historial de cambios del repositorio en el que nos encontramos.

```
/home/juanjo/Proyecto> hg log
changeset:   0:b9fa4ebea246
tag:         tip
user:        Juanjo Molinero <jjmolinero@gmail.com>
date:        Mon Feb 13 14:15:20 2012 +0100
summary:     Commit inicial
```

Si realizamos un cambio en alguno de los archivos, podemos ver que cambios hemos hecho usando la orden *hg st*.

```
/home/juanjo/Proyecto> hg st
M Prueba.rb
```

Si a pesar de haber hecho algunos cambios, no nos interesan (bien porque nos hemos equivocado, porque eran una prueba, o porque otra persona ha hecho unos cambios más adecuados) podemos volver a la versión anterior usando la orden *revert*.

```
/home/juanjo/Proyecto> hg revert --all
reverting Prueba.rb
```

Si en lugar de eso, no estamos seguros de los cambios que hemos hecho, podemos hacer una revisión de los mismos usando la orden *hg diff*.

```
/home/juanjo/Proyecto> vi Prueba.rb
/home/juanjo/Proyecto> hg st
M Prueba.rb
/home/juanjo/Proyecto> hg diff
diff -r b9fa4ebea246 Prueba.rb
--- a/Prueba.rb Mon Feb 13 14:15:20 2012 +0100
+++ b/Prueba.rb Mon Feb 13 14:26:39 2012 +0100
@@ -0,0 +1,1 @@
+otros cambios más
```

Para más información sobre Mercurial, se puede seguir el tutorial publicado por Joel Spolsky [1] o el libro de referencia del sistema [6].

[1] http://hginit.com/