Título: Diseño e integración del componente social del Racó Mobile para iOS

Autor: Adrián Cañadas González

Fecha: 22/06/2012

Director: Hugo Hernández Pibernat

Departamento del director: Llenguatges i sistemes informátics - LSI

Co-Director: Jaume Moral Ros

Departamento del co-director: Laboratori de Calcul FIB - LCFIB

Ponente: Anna Río Doval

Departamento de la ponente: Matemática aplicada II - MAII

Índice general

Índice de figuras **5**

1. Introducción **6**
1.1. Descripción del proyecto . . . 7
1.1.1. Descripción y objetivos . . . 7
1.1.2. Motivaciones . . . 8
1.2. Entorno de desarrollo . . . 9
1.2.1. iOS . . . 9
1.2.2. Cocoa touch, Xcode y iOS Simulador . . . 10
1.3. Patrones de diseño . . . 11
1.3.1. Patrón Modelo-Vista-Controlador . . . 12
1.3.2. Patrón Two-stage creation . . . 13
1.3.3. Patrón Singleton . . . 13
1.3.4. Patrón delegate . . . 13
1.3.5. Patrón Controlador . . . 16
1.3.6. Patrón Fachada . . . 16
1.3.7. Patrón Outlets and Actions . . . 17
1.4. Organización de la memoria . . . 19

2. Análisis **20**
2.1. Requisitos funcionales . . . 20
2.1.1. Autenticación OAUTH . . . 20
2.1.2. Mensajería . . . 21
2.1.3. Notificaciones . . . 21
2.1.4. Persistencia de datos . . . 22
2.1.5. Interfaz . . . 22
2.2. Requisitos no funcionales . . . 22
2.3. Decisiones de diseño . . . 24

3. Especificación y diseño **25**
3.1. Arquitectura del sistema . . . 25
3.2. Mapa navegacional . . . 27
3.3. Actores . . . 32
3.4. Definición de los casos de uso . . . 32
3.4.1. Caso de uso: Obtención de token OAUTH . . . 33
3.4.2. Caso de uso: Consulta del listado de chats . . . 36
3.4.3. Caso de uso: Consulta mensajes . . . 36
3.4.4. Caso de uso: Consulta del listado de contactos . . . 36
3.4.5. Caso de uso: Añadir contactos . . . 37
3.4.6. Caso de uso: Eliminar contacto . . . 37
3.4.7. Caso de uso: Gestionar una petición de amistad . . . 38
3.4.8. Caso de uso: Crear un chat convencional . . . 38

3.4.9. Caso de uso: Crear un chat multiusuario . . . 39
3.4.10. Caso de uso: Eliminar chat convencional . . . 41
3.4.11. Caso de uso: Eliminar chat grupo . . . 41
3.4.12. Caso de uso: Enviar un mensaje . . . 42
3.5. Modelo conceptual . . . 44
3.5.1. Modificaciones del modelo anterior . . . 44
3.5.2. Chat y mensajes . . . 47
3.5.3. Contactos . . . 47
3.6. Vistas . . . 48
3.7. Controladores . . . 48
3.8. Planificación . . . 52

4. Detalles de implementación **54**
4.1. Notificaciones . . . 54
4.2. Elementos de navegación personalizados . . . 58
4.2.1. La rueda táctil . . . 58
4.2.2. El gestor de navegación . . . 61
4.3. Chat . . . 63
4.3.1. Comunicación iOS - Openfire . . . 63
4.3.2. Pantalla . . . 64

5. Conclusiones y ampliaciones **68**
5.1. Objetivos . . . 68
5.2. Valoración general del proyecto . . . 69
5.3. Valoración económica . . . 69
5.4. Ampliaciones . . . 70

6. Bibliografía **72**

Índice de figuras

1.1. Flujo en el patrón MVC 12
1.2. Diagrama de un sistema que no utiliza el patrón Delegate [11] 14
1.3. Diagrama de un sistema que utiliza el patrón Delegate [11] 15
1.4. Diagrama de un sistema que utiliza el patrón Delegate múltiples veces [11] 16
1.5. Diagrama de un sistema que utiliza el patrón Multicast Delegate 17
1.6. Ejemplo Pizzería: Diagrama de un sistema que utiliza el patrón Fachada [11] . . . 18

3.1. Comparativa arquitectura del sistema versión 1 (arriba) y versión 2 (abajo) 26
3.2. Mapa navegacional de la primera versión del Racó Mobile [10] 28
3.3. Mapa navegacional de la Rueda Táctil 29
3.4. Mapa navegacional de las pantallas de Chat y Contactos 30
3.5. Mapa navegacional de la obtención del Token OAUTH 31
3.6. Esquema original de los actores en Racó Mobile 1.0 [10] 32
3.7. Diagrama de secuencia de la obtención de un Token OAUTH 35
3.8. Diagrama de secuencia del caso de uso Añadir Contactos 37
3.9. Diagrama de secuencia del caso de uso Crear un chat convencional 39
3.10. Diagrama de secuencia del caso de uso Crear un chat multiusuario 40
3.11. Diagrama de secuencia del caso de uso Eliminar un chat multiusuario 43
3.12. Modelo conceptual del Racó Mobile 1.0 [10] 45
3.13. Modelo conceptual del Racó Mobile 2.0 46
3.14. Esquema de las vistas del Racó Mobile 2.0 49
3.15. Esquema de los controladores del Racó Mobile 2.0 51
3.16. Planificación de tareas: Diagrama de Gantt del proyecto 53

4.1. Tipos de notificaciones Push según la configuración 55
4.2. Flujo de datos entre el servidor de la aplicación, el servidor de Apple y nuestra aplicación [3] 56
4.3. Mensaje emergente de permisos para notificaciones 56
4.4. Validación de los certificados dispositivo-servidor [3] 57
4.5. Obtención y envío del token identificador del dispositivo [3] 58
4.6. Flujo completo de envío y recepción de una notificación Push [3] 59
4.7. Comparativa de la barra de la versión 1.0 (izq.) y 2.0 (der.) 59
4.8. Punto de punto de origen (azul) y rotación (rojo) 60
4.9. Esquema de la utilización de un array adicional 63
4.10. Captura de la aplicación de mensajería iMessage para iOS 65
4.11. Esquema de las vistas que componen una celda básica 66
4.12. Esquema de las vistas que componen nuestra celda personalizada 66
4.13. Texto cubriendo por completo la imagen del globo 67
4.14. Texto tocando los márgenes del globo 67
4.15. Globo de texto con márgenes y el timestamp 67

5.1. Resumen del coste en recursos humanos del proyecto 70

Capítulo 1

Introducción

El crecimiento desmesurado de los *Smartphones*[1] es uno de los ejes centrales de la sociedad actual y más todavía en España ya que estudios recientes nos sitúan a la cabeza de Europa en la venta de este tipo de terminales, pues un 55,2 % de los usuarios de telefonía móvil opta por esta nueva tecnología.

Como era de esperar este crecimiento no solamente provoca que las ventas de terminales se disparen sino que todas aquellas actividades que mantienen una relación directa o indirecta con los Smartphones, como por ejemplo el desarrollo de aplicaciones o la publicidad móvil, hayan empezado a tener un gran peso en la sociedad actual creando nuevas modas, nuevas formas de comunicación y una gran cantidad de puestos de trabajo. Se podría decir que la aparición y proliferación de los Smartphones ha marcado un antes y un después en la historia de la comunicación.

Existe una gran cantidad de fabricantes de estos dispositivos, cada uno con sus modelos y sus marcas específicas, pero siendo realistas y observando detenidamente el mercado, veremos que se puede dividir en dos grandes bloques: iOS y Android.

Los terminales que utilizan Android se caracterizan por la amplitud de terminales que engloba, pues dicho sistema creado y mantenido por la empresa Google se puede instalar en todo tipo de terminales ya que su distribución es libre y por lo tanto, es decisión del fabricante si desea o no crear un producto compatible con el sistema Android. Este hecho y su indudable calidad como sistema es lo que hace que a día de hoy sea el sistema operativo móvil más extendido y utilizado del mundo.

Por otra parte encontramos los terminales con iOS, creado y mantenido por la empresa Apple, que a diferencia de Android únicamente puede instalarse en productos fabricados por Apple, ya que en este caso la distribución del sistema no es libre sino propietaria. Esto podría parecer un problema pero un diseño cuidado hasta el mínimo detalle, la optimización extrema y la longevidad de las actualizaciones han hecho de iOS el segundo sistema operativo móvil más utilizado del planeta.

Como se podría suponer existe una *batalla* entre los creadores de estos dos sistemas operativos para móviles hecho que a simple vista debería parecer beneficioso pues la competencia constructiva implica más trabajo por parte de desarrolladores para no quedarse atrás frente a la empresa rival y por lo tanto el usuario final del producto debería salir beneficiado por la evolución de los productos, por desgracia esto no siempre es así ya que en algunas ocasiones se ha llegado a los tribunales por disputas referentes a patentes y propiedades intelectuales y ello ha puesto en peligro la estabilidad del mercado en numerosas ocasiones provocando por ejemplo que en determinados países no se pudiese adquirir uno u otro tipo de terminal concreto, hecho que claramente perjudica directamente al usuario final y a la imagen de las marcas (y por extensión de los sistemas) implicadas.

[1] Teléfonos inteligentes

Pese a este tipo de problemas ocasionales que pueden surgir nunca suelen llegarse a extremos drásticos y los temas suelen zanjarse con acuerdos entre las distintas empresas que puedan verse implicadas y muchas veces lo que suele ocurrir es que tras estas disputas las ventas de alguno de los implicados se disparen.

Por lo tanto podemos ver como el mercado de Smartphones y otros dispositivos móviles similares goza de buena salud y eso implica la creación de nuevo sectores que ofrecen servicios relacionados con nuestros terminales. Los primeros fueron las compañías de telefonía que comenzaron a ofrecer distintas variedades de tarifas de datos que permitían exprimir al máximo las capacidades los Smartphones hecho que más tarde provocó la explosión del mercado de aplicaciones para móviles que actualmente son uno de los pilares básicos de una gran cantidad de empresas, muchas de las cuales han surgido gracias a la explotación de estos terminales e incluso algunas han crecido desmesuradamente tras conseguir sacar a la venta una aplicación novedosa o rompedora como fue el caso de *Rovio*, la empresa responsable del videojuego *Angry Birds*.

Por lo tanto no podemos negar la increíble revolución que ha supuesto la llegada de esta tecnología que incluso a día de hoy nos intenta sorprender con novedades cada vez más elaboradas como por ejemplo el reconocimiento de voz, el guiado por caminos y carreteras o el reconocimiento facial de las cámaras integradas en estos equipos portátiles. Es mucho el capital que se invierte en desarrollar estas tecnologías que permiten al usuario llevar una vida más fácil y cómoda en un aparato no más grande que la palma de la mano y como consecuencia de ello cada día más gente y empresas dependen de los codiciados Smartphones.

En este proyecto nos centraremos únicamente en uno de los dos sistemas móviles y en un tipo de terminales concretos, pues como hemos podido observar la variedad es muy grande y una buena estrategia suele ser focalizarse sobre un sector. El sistema elegido es iOS de Apple y más concretamente en la versión para teléfonos móviles, los *iPhone*[2].

1.1. Descripción del proyecto

Este Proyecto de Final de Carrera pretende ampliar las funcionalidades de la aplicación **Racó Mobile**, que es la aplicación oficial de la Facultad de Informática de Barcelona (FIB) que da acceso a la intranet y a otros recursos como noticias, horarios, etcétera. Con el rápido crecimiento de las tecnologías móviles han ido surgiendo nuevas necesidades que la aplicación de la facultad debería poder cubrir, más concretamente debería permitir la comunicación rápida y efectiva entre los diferentes integrantes de la institución que principalmente son estudiantes y profesores.

1.1.1. Descripción y objetivos

Como ya hemos comentado el principal objetivo de este proyecto es el de ampliar las funcionalidades ofrecidas por la aplicación Racó Mobile, más concretamente lo se quiere añadir es un sistema de comunicación entre estudiantes y/o profesores para facilitar tareas como trabajos en grupo, notificación de avisos o cualquier otra que precise el envío de mensajes.

De este modo se creará un entorno de mensajería para que todo aquel con acceso al *Racó*[3] pueda comunicarse con sus compañeros y profesores. Dicho entorno funcionará como un cliente de mensajería instantánea tradicional como por ejemplo **MSN Messenger**, pero además tendrá las características de un sistema de mensajería pensado para Smartphones como los mensajes offline y la creación de grupos persistentes en el tiempo, igual que los ofrecidos actualmente por la famosa

[2]Teléfono inteligente fabricado y diseñado por la empresa Apple
[3]Es el nombre que recibe la intranet de la Facultad de Informática de Barcelona

aplicación **Whatsapp**.

Este objetivo va directamente ligado a la implantación de notificaciones Push. Estas notificaciones sirven para indicar que ha habido algún tipo de cambio en el estado de los datos almacenados en el servidor de la aplicación, por ejemplo en nuestro caso indicarán que un usuario nos ha enviado un mensaje. Así que deberemos construir un sistema que nos permita generar notificaciones Push cuando los usuarios se manden mensajes entre ellos y esto lo conseguiremos implementando un plugin dentro del mismo servidor de mensajería.

Al igual que las telecomunicaciones han ido avanzando también lo han hecho los servicios relacionados y por lo tanto cada vez existen más empresas y entidades que nos permiten acceder a nuestros datos de manera remota. Esto también ha sucedido en la FIB, buen ejemplo de ellos es la versión existente de la aplicación Racó Mobile donde podemos acceder a nuestros datos del Racó, verlos y gestionarlos mediante nuestro terminal. Sin embargo resulta tedioso y poco seguro el almacenamiento y utilización de parejas usuario-password, por este motivo uno de los objetivos más importantes del proyecto será sustituir el sistema de claves actual por el protocolo *Open Authorization* más conocido como OAUTH.

OAUTH es un protocolo abierto y estandarizado para la autorización del acceso a APIs de terceros de un modo seguro. Uno de los principales motivos por el que se creó OAUTH fue para proteger los datos del usuario, por ejemplo si nuestra aplicación Racó Mobile desease acceder a la información de un usuario guardada en laboratorio de cálculo(LCFIB) lo que haría sería pedir permiso al LCFIB para acceder a dichos datos, entonces el dueño de esos datos (usuario) deberá conceder permisos dentro de los sistemas del LCFIB, así nuestra aplicación podrá conseguir únicamente los datos que autorice el usuario y todo ello sin necesidad de introducir dentro de la propia aplicación ningún tipo de código personal haciendo el proceso más seguro y fiable.

Como objetivo final, pero no por ello menos importante, se intentará crear algún elemento distintivo para que la interfaz de usuario parezca más moderna y elegante, no debemos olvidar que estamos trabajando en dispositivos de Apple y la gente suele esperar que los diseños de las aplicaciones sean muy visuales y fáciles de utilizar.

1.1.2. Motivaciones

Este proyecto se nos presenta como un gran reto y deberemos ser capaces de superarlo, para ello deberemos aprender muchas cosas que nos serán útiles para poder avanzar poco a poco, para poder construir día a día lo que al final deberá terminar siendo la aplicación Racó Mobile 2.0.

Además no podríamos haber elegido un momento mejor par la creación de este proyecto, pues tanto los terminales móviles como las aplicaciones de mensajería relacionadas cada día tienen más usuarios y su popularidad no para de crecer. Debemos ser capaces de crear un buen producto para que la gente tenga ganas de utilizarlo porqué realmente le haga la vida más fácil de uno u otro modo.

Así pues estamos tenemos una gran responsabilidad y debemos intentar conseguir cumplir todos y cada uno de los objetivos para que al final del proyecto seamos capaces de ver que el uso de la nueva aplicación se expande con cada oleada de nuevos alumnos que vayan llegando a la facultad.

1.2. Entorno de desarrollo

Todo sistema operativo ya sea móvil o de sobremesa necesita aplicaciones para ser útil y esas aplicaciones se deben desarrollar siguiendo unas pautas y convenciones impuestas por el mismo sistema y hardware en el que se vaya a ejecutar. Claros ejemplos de ello son la utilización de determinados lenguajes de programación o la compilación para una arquitectura determinada; esto podría parecer una dificultad en el momento de crear aplicaciones y servicios para cualquier entorno, sin embargo todo sistema debe tener ciertas limitaciones ya que uno totalmente universal podría volverse inestable o demasiado complejo tanto de utilizar como de mantener.

Partiendo de esta premisa observamos como la mayor parte de sistemas tiene uno o varios entornos de desarrollo específicos para crear componentes que sean compatibles con el sistema para el cual estemos realizando el desarrollo. Estos entornos de desarrollo nos proporcionan una serie de herramientas para facilitarnos las tareas tanto de diseño como de implementación y por lo tanto se convierten en una herramienta vital para el desarrollo.

1.2.1. iOS

El sistema iOS no es distinto de los demás y por lo tanto tendrá un entorno de desarrollo, pero antes sería apropiado explicar un poco de historia sobre este [4] [1] [8] [4].

La primera versión de iOS se llamaba iPhone OS y apareció en el primer trimestre de 2008, por aquel entonces aun siendo novedoso era muy cerrado ya que no se permitían aplicaciones de terceros, únicamente se podían ejecutar las aplicaciones preinstaladas en el terminal. Pronto se dieron cuenta de que la clave del éxito serían las aplicaciones y en eso se centró la segunda versión del sistema, pues con iPhone OS 2.0 llego la famosa App Store que permitía la descarga de aplicaciones externas a Apple hecho que provocó el crecimiento tanto de los usuarios del terminal como de la cantidad de desarrolladores interesados en crear aplicaciones para el mismo.

A principios de 2009 llegó la tercera versión del sistema: iPhone OS 3.0. No supuso una gran revolución a nivel tecnológico ya que la principal atracción fue el inicio de la integración con las notificaciones Push que permitían recibir avisos cuando las aplicaciones recibían alguna actualización de su estado mientras no estaban activas. esta funcionalidad fue duramente criticada ya que no se permitía que las aplicaciones se ejecutasen en segundo plano bajo el pretexto del consumo de batería.

En el año 2010 apareció el iPad, la primera tablet del mercado que trajo de la mano un cambio de nombre al sistema operativo ya que iPhone OS pasaría también a utilizarse en el iPad con lo cual el paso lógico fue lo que pasó a llamarse iOS 4.0. Con esta actualización llegó la tan ansiada multitarea pero algo descafeinada pues a Apple le seguía preocupando el consumo de batería que las aplicaciones en segundo plano podrían suponer y por lo tanto la multitarea se limitó únicamente a ciertos procesos muy limitados de algunas aplicaciones. También como actualización menor apareció la posible organización de los iconos de aplicación en carpetas.

Finalmente en 2011 aparecería iOS 5.0, una versión muy esperada por usuarios y desarrolladores pues incluiría una de las funcionalidades que la competencia (más concretamente Android) llevaba ya tiempo utilizando con éxito: el centro de notificaciones. También se incluyó el soporte para iCloud, el almacenamiento en la nube que prometía simplificar el almacenamiento de documentos en la red para poder acceder a ellos a través de nuestros distintos dispositivos. De nuevo las mejoras fueron bien recibidas pero muy criticadas por no dar la talla ya que el sistema de notificaciones de Android continuaba siendo mejor y iCloud estaba muy verde todavía pues sin ir

[4] Sistema Operativo

más lejos era imposible acceder a nuestros documentos desde un *mac*.

No sabemos lo que nos depara el futuro pero como podemos ver la evolución es constante y seguro que 2012 será un gran año para iOS al cual pese a ser un sistema muy avanzado todavía le queda mucho margen de mejora.

1.2.2. Cocoa touch, Xcode y iOS Simulador

Si bien hemos comentado la evolución de las distintas versiones del sistema debemos saber que todas ellas comparten algo en común: Cocoa touch.

Cocoa touch es el *framework*[5] utilizado para desarrollar las aplicaciones de iOS, desde su primera versión hasta la última. Basado en Cocoa (framework para programar aplicaciones para OS X) está escrito en lenguaje Objective-C [12] [7] y debemos ser capaces de dominarlo si queremos desarrollar aplicaciones para este sistema.

Cocoa touch nos ofrece una extensa API que nos permitirá interactuar con el sistema y , como no, recibir los eventos que produce el usuario al interactuar con la pantalla. Entre las diferentes funcionalidades que nos ofrece podríamos destacar:

- Core Data: Nos permite realizar una gestión eficiente y sencilla del almacenamiento de datos de nuestras aplicaciones.
- Core Audio: Con ello gestionaremos de un modo preciso la reproducción de fuentes sonoras, su procesado, etcétera.
- Webkit: Facilita el tratamiento y interacción con el entorno de las páginas web e internet

Éstas son solamente algunas de sus funcionalidades más populares, sin embargo existen infinidad de recursos que Cocoa touch pone a nuestra disposición para que seamos capaces de crear aplicaciones ricas en contenido y fluidas en ejecución con una sencillez difícil de encontrar en otros entornos.

Además debemos añadir que Cocoa touch incentiva la utilización de patrones de diseño [11], de hecho en algunos momentos incluso nos obliga. Esto se debe a que la construcción interna del propio framework está basada en patrones de diseño donde predomina el famoso MVC (Modelo - Vista - Controlador).

Para exprimir al máximo la potencia de este framework Apple nos ofrece el entorno de desarrollo Xcode que nos permitirá crear, compilar y probar nuestras aplicaciones. En este sentido el desarrollo para iOS es bastante cerrado ya que únicamente se pueden crear aplicaciones para este sistema utilizando Xcode que sólo está disponible para máquinas vendidas y fabricadas por Apple, hecho que le ha ganado una gran cantidad de críticas a la empresa aunque ello no le quita mérito al gran trabajo que han hecho con Xcode y su simulador que son herramientas muy superiores a las de Android (la competencia directa), tanto en agilidad como en facilidad de uso.

Al igual que las versiones del sistema operativo Xcode se ha ido actualizando, incluyendo las APIs y las nuevas características de cada nueva versión del sistema e integrando cada vez más recursos dentro del propio entorno de desarrollo. Por poner un ejemplo, el creador de interfaces (Interface Builder) inicialmente era una aplicación externa a Xcode, actualmente está integrada como parte del entorno de desarrollo hecho que facilita mucho las cosas. Además para probar nuestros desarrollos simplemente debemos pulsar sobre la opción de *Compilar y ejecutar* para que

[5] Estructura conceptual y tecnológica que permite la organización y desarrollo de software

automáticamente se lance el simulador, con la versión de iOS que tengamos configurada, y así poder ver y probar la mayor parte de funcionalidades que encontraríamos en uno de los teléfonos reales.

En principio esto es todo lo que necesitamos para crear aplicaciones para iOS si no tenemos en cuenta la generación de los gráficos que harán de nuestra aplicación algo más personal y elegante y que generalmente suelen construirse en programas especializados para dichos menesteres como por ejemplo Adobe Illustrator.

1.3. Patrones de diseño

Los patrones de diseño son un conjunto de pautas teóricas que nos ayudan a resolver y estructurar los problemas que aparecen comúnmente en el desarrollo de software. Existe una gran cantidad de ellos y se adaptan a múltiples situaciones e incluso a casos específicos de cada lenguaje o entorno de programación. Para entender mejor qué es un patrón de diseño describiremos brevemente su estructura general:

- Problema: Todo patrón de diseño parte de la existencia de un problema inicial al cual debemos dar solución, cada patrón suele estar asociado a un tipo concreto de problema, aunque en algunos casos un mismo patrón nos servirá para aplicarlo a una tipología de problemas más amplia
- Descripción: Para el problema presentado se describe un conjunto de reglas a seguir para solucionarlo de la mejor manera posible, esto representa realmente el *cuerpo y alma* de lo que es un patrón de diseño.
- Motivación: Cada patrón de diseño tiene unas características intrínsecas que lo hacen idóneo o mejor que otros para resolver un tipo de problema, la motivación para utilizar un patrón frente a un problema concreto son todas aquellas ventajas que nos proporciona el hecho de aplicarlo.
- Desmotivación: Como no todos los problemas son iguales, existen casos en que no es aconsejable utilizar algunos patrones que en lugar de ventajas nos proporcionaran dificultades.

La clave para utilizar correctamente los patrones en nuestro diseño es en primer lugar detectar que problemas nos presenta el diseño, a continuación debemos buscar la descripción de aquellos patrones que se puedan aplicar a nuestros problemas y finalmente valorar su aplicación realizando un balance entre los beneficios y pegas que supondrá para nuestro caso concreto. Si conseguimos realizar este proceso con éxito podemos estar seguros de que construiremos nuestro software sobre una base muy sólida.

Como ya hemos comentado este proyecto se basa en la ampliación de la primera versión del Racó Mobile que en su momento pasó por el proceso de desarrollo al completo y durante el cual se aplicaron patrones de diseño. Este hecho nos supondrá una gran ventaja ya que nos va a permitir realizar cambios y mejoras con mucha más facilidad que si se hubiese implementado de forma caótica o desestructurada.

A continuación procederemos a explicar brevemente los patrones de diseño que vamos a utilizar durante el proceso de desarrollo de la segunda versión del Racó Mobile comentando en cada caso en qué nos van a ayudar y el porqué de su elección.

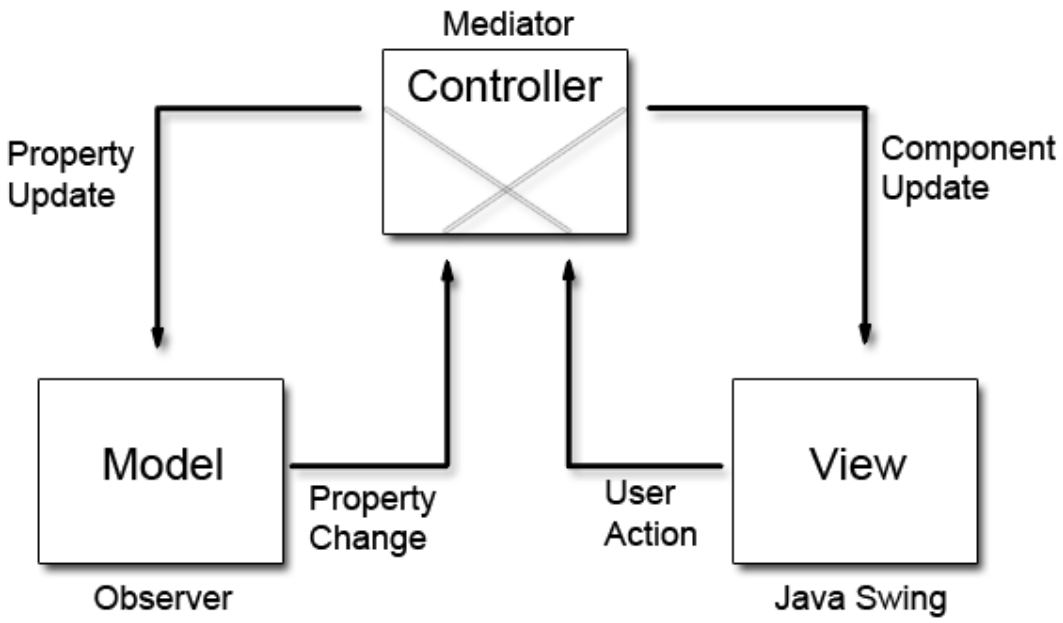

Figura 1.1: Flujo en el patrón MVC

1.3.1. Patrón Modelo-Vista-Controlador

También conocido por las siglas **MVC** es quizás el patrón arquitectónico más famoso y también el más utilizado. De hecho Cocoa touch está construido prácticamente alrededor de este patrón que se basa en la separación de la interfaz de la aplicación, la lógica de control y los datos de trabajo.

El patrón define esos tres bloques del siguiente modo:

- Modelo: El modelo se encarga de gestionar los datos del usuario así como las reglas de negocio establecidas.
- Vista: Se encarga exclusivamente de la interacción con el usuario, por lo general mostrando datos y recibiendo acciones o eventos.
- Controlador: Relaciona los dos anteriores y gestiona los eventos provocados por el usuario o el sistema.

Un ejemplo de flujo consistiría en: el usuario realiza una acción sobre una Vista, el Controlador detecta dicha acción y tramita la petición al Modelo que se encargará de actualizar y/o consultar los datos para devolver un resultado al Controlador que a su vez indicará a la Vista como debe actualizarse en caso de que sea necesario. Esto lo podemos ver perfectamente representado en la figura-1.1

Como vemos lo que pretende esta construcción es desacoplar los tres grandes bloques que componen prácticamente todas las aplicaciones buscando la independencia de cada uno de ellos para que un cambio en una parte del sistema no suponga tener que cambiar toda la otra.

En nuestra aplicación será utilizado como patrón general ya que prácticamente todo el sistema se podría dividir en tres grandes bloques donde interactúan controladores, vistas y el modelo de datos.

1.3.2. Patrón Two-stage creation

Este patrón viene forzado por la utilización de Cocoa touch ya que estamos obligados a utilizarlo, en cambio otros lenguajes como Java o C++ permiten su implementación pero no la obligan. Se basa en separar los procesos de reserva de memoria y inicialización en el momento de crear instancias.

Tener dos procesos separados tiene la ventaja de que podremos inicializar las instancias independientemente de cómo fue reservada la memoria para las mismas. Además nos permitirá una mejor gestión de la herencia ya que cada clase definirá un inicializador por defecto (que será el que sea capaz de inicializar todos los parámetros), entonces al crear una subclase simplemente debemos asegurarnos que llamamos en primer lugar al inicializador de la superclase para estar seguros de que no queda ninguna variable sin inicializar.

Como este patrón es de uso obligado estará presente cada vez que necesitemos crear una instancia de cualquier objeto que se utilice.

1.3.3. Patrón Singleton

Este patrón restringe la creación de un tipo de instancia, no permitiendo que exista más de una en el sistema. Esto representa un claro beneficio cuando necesitamos que un **mismo** objeto sea accesible desde muchos puntos distintos de la aplicación.

Lo utilizaremos principalmente en clases que requieren acceso desde múltiples lugares como por ejemplo las que gestionan la persistencia Core Data o las que permiten comunicarnos con el servidor de mensajería.

1.3.4. Patrón delegate

Es quizás junto a los tres anteriores el más utilizado en el desarrollo de aplicaciones en Cocoa y consiste en permitir que un objeto reaccione a cambios o acciones realizadas sobre otro objeto distinto.

Aclararemos el funcionamiento comentando un ejemplo sencillo en el que intervienen una vista y su controlador:

El controlador es un objeto más genérico y usualmente dedicado a tareas de gestión y que la vista realiza tareas más específicas. Supongamos que el controlador necesita actualizar un contador cuando la vista se haya cargado en pantalla, el controlador puede haber creado la vista pero no puede saber en qué momento se cargará en pantalla, el objeto vista en cambio sabe perfectamente cuando ha terminado de mostrarse, por lo tanto el objeto vista que mantendrá una referencia al objeto controlador mandará un mensaje a éste cuando se haya cargado por completo, entonces el controlador captará dicho mensaje y actualizará el contador.

La ventaja de todo esto la explotamos cuando se utilizan varios objetos a la vez; si no utilizásemos el patrón delegate se produciría una gran cantidad de acoplamientos entre los distintos objetos que se deberían avisar todos entre ellos tal y como se puede apreciar en la figura-1.2, en cambio si cada objeto utiliza el método delegado sobre un objeto centralizado se produce un desacoplamiento masivo tal y como podemos observar en las figuras 1.3 y 1.4.

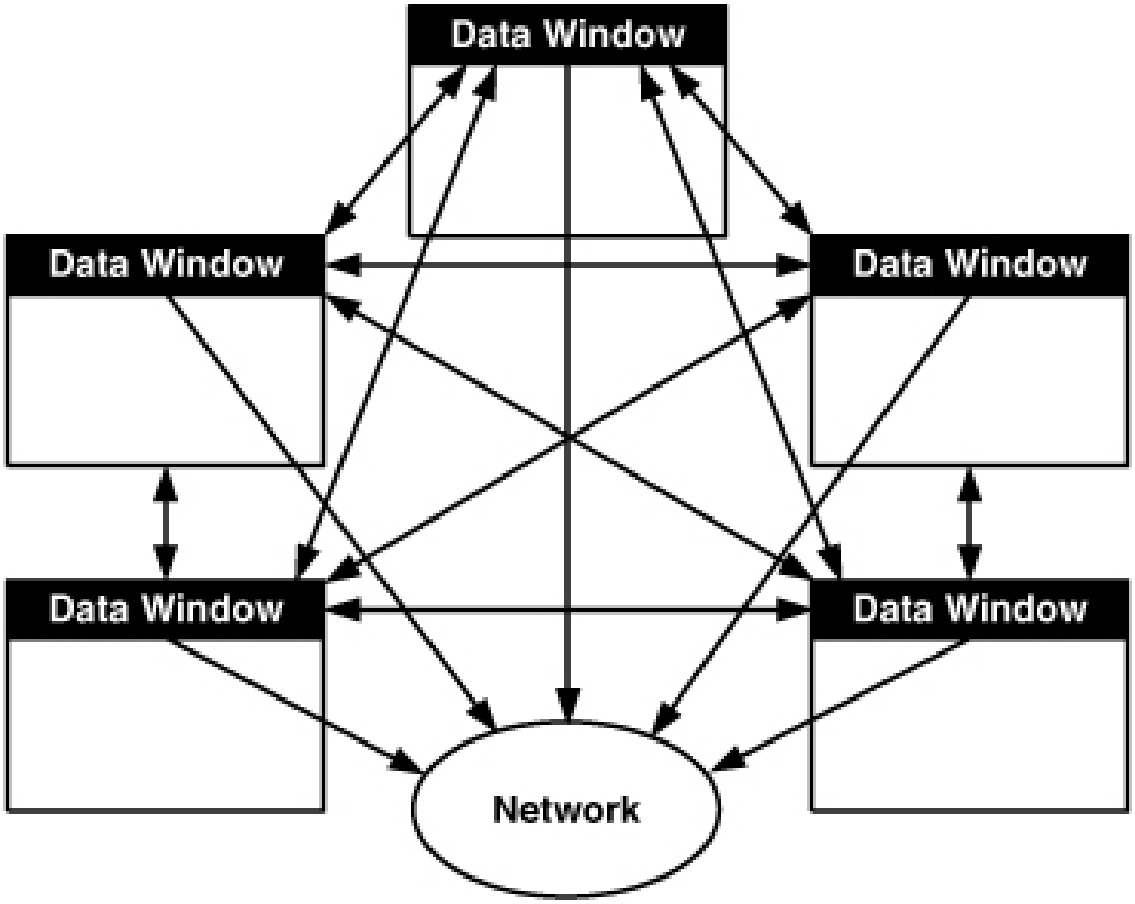

Figura 1.2: Diagrama de un sistema que no utiliza el patrón Delegate [11]

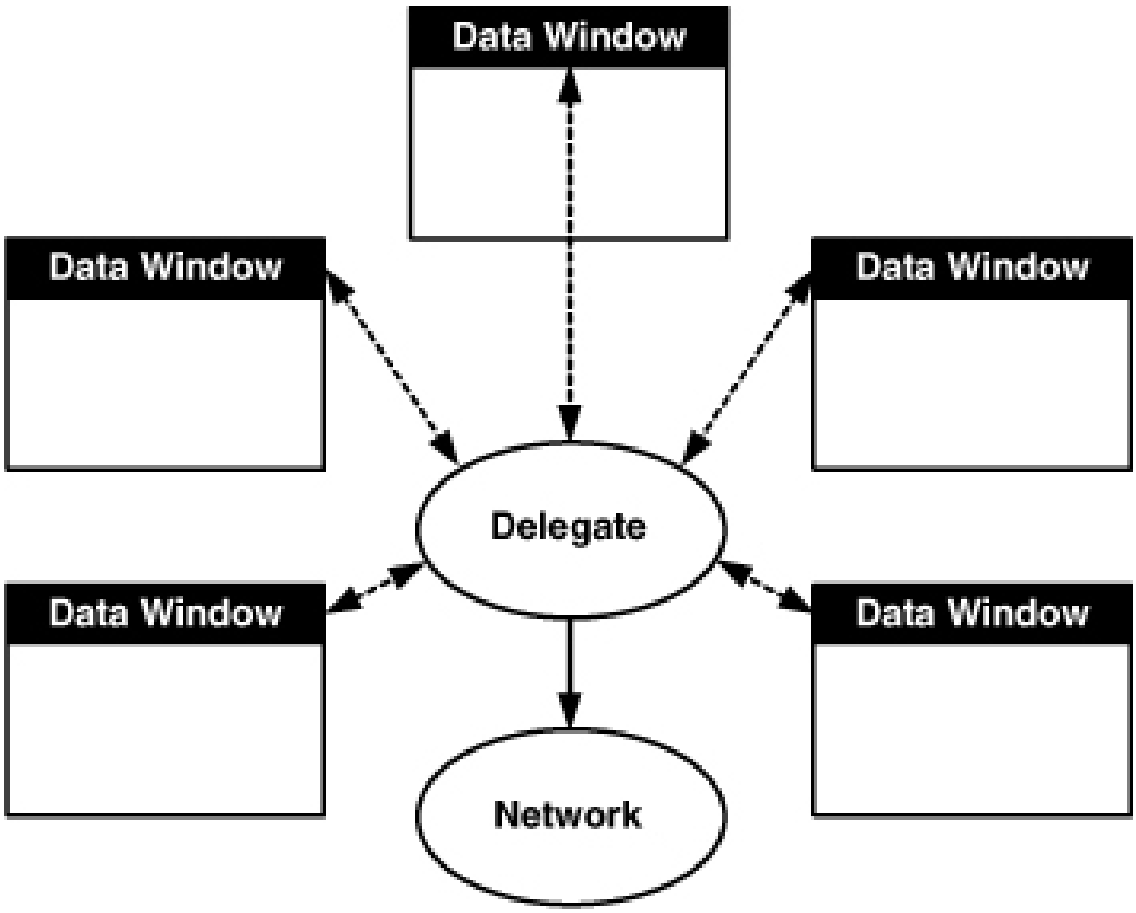

Figura 1.3: Diagrama de un sistema que utiliza el patrón Delegate [11]

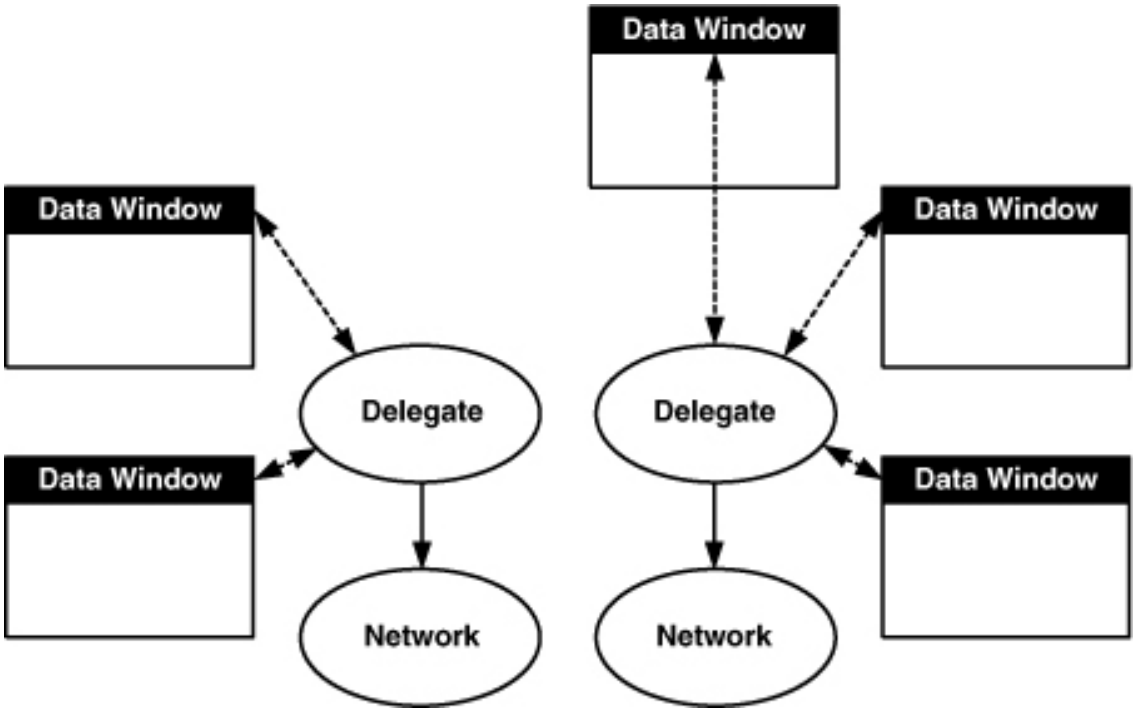

Figura 1.4: Diagrama de un sistema que utiliza el patrón Delegate múltiples veces [11]

Además existe una variante de este patrón que podemos ver representado la figura-1.5, que es el multicast delegate, el principio es el mismo sin embargo en lugar de existir un método delegado existen N métodos delegado independientes entre sí. En el ejemplo de antes la vista en lugar de avisar únicamente al controlador avisaría a varios y todos realizarían las acciones que tuviesen asignadas al recibir el mensaje de la vista. Esto será muy útil cuándo un evento se debe reflejar en varios sitios a la vez.

En nuestra aplicación le sacaremos partido en muchas ocasiones, sobre todo en el momento de realizar comunicación por red, al recibir mensajes de chat, descargar datos etcétera.

1.3.5. Patrón Controlador

Como ya hemos comentado en el patrón MVC los controladores se utilizan para recibir eventos y lanzar acciones en consecuencia hacia el destino adecuado, que normalmente suelen ser las vistas o el modelo de negocio. En el sistema iOS lo encontraremos presente en prácticamente toda la aplicación, ya que por norma general una pantalla implica la existencia de un controlador para dicha pantalla, además también existen controladores orientados gestionar los datos o las comunicaciones.

Gracias al uso de este patrón somos capaces de mantener el sistema desacoplado lo que minimizará el impacto de los cambios, propiciando la reusabilidad y facilitando el mantenimiento del código.

1.3.6. Patrón Fachada

En los sistemas grandes hay grandes existe una gran cantidad de métodos y funciones que se invocan cuando realizamos una acción, sin embargo es muy probable que para nosotros esa acción no represente un conjunto tan elevado de tareas entrelazadas. El patrón fachada se basa en esta

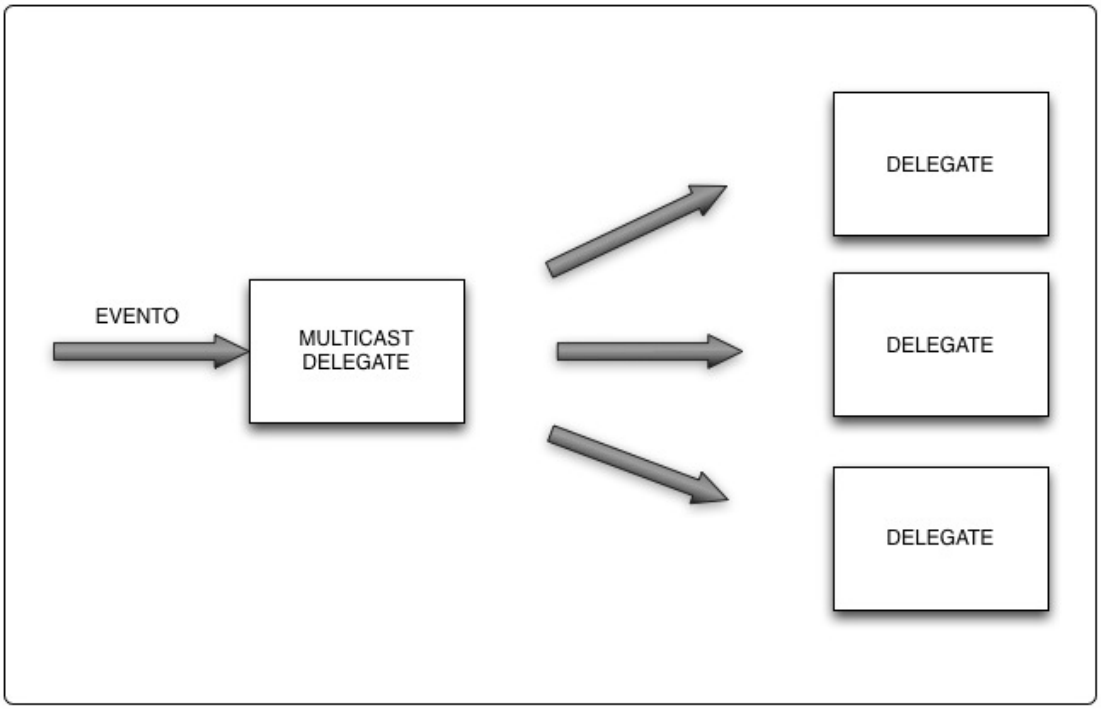

Figura 1.5: Diagrama de un sistema que utiliza el patrón Multicast Delegate

premisa, consiste en proporcionar una interfaz que recoja las funcionalidades de un grupo de clases pero que a la vez esconda toda la complejidad del proceso. Esto beneficia la utilización de dichas funcionalidades que únicamente serán invocadas sobre el objeto que implementa la fachada y éste será el que se encargue de realizar la acción con toda la complejidad que ello conlleva. Por su naturaleza este patrón se suele combinar con el patrón singleton.

Un ejemplo muy típico para explicar este patrón de diseño es el de la pizzería representado en la figura-1.6. Nosotros como clientes probablemente hablemos con el operador que recoge nuestro pedido, nuestro objetivo es pedir una pizza y el operador será el que mueva los hilos para conseguirlo, pues en el proceso (que nosotros como clientes no vemos) interviene más gente además del operador como por ejemplo cocineros, repartidores y proveedores.

En nuestra aplicación este patrón será realmente útil al utilizar las funcionalidades relacionadas con la conexión al servidor de mensajería o a la gestión de la persistencia entre otras.

1.3.7. Patrón Outlets and Actions

Este patrón está basado directamente en la creación de las vistas y sus controladores y la solución que nos propone es la utilización de **Outlets** y **Actions**. Los Outlets son objetos que apuntan directamente a un elemento de la interfaz gráfica, estos elementos se declaran como variables dentro de los controladores de las vistas y se enlazan mediante código o utilizando el *Interface Builder*[6] con el archivo que describe la vista concreta. Las Actions en cambio son lo contrario, son las acciones que se disparan cuando se provoca un cambio en una vista y pueden ser enlazadas con múltiples componentes de las mismas así como ser reutilizados.

[6]Constructor de interfaces para aplicaciones iOS integrado con la herramienta Xcode

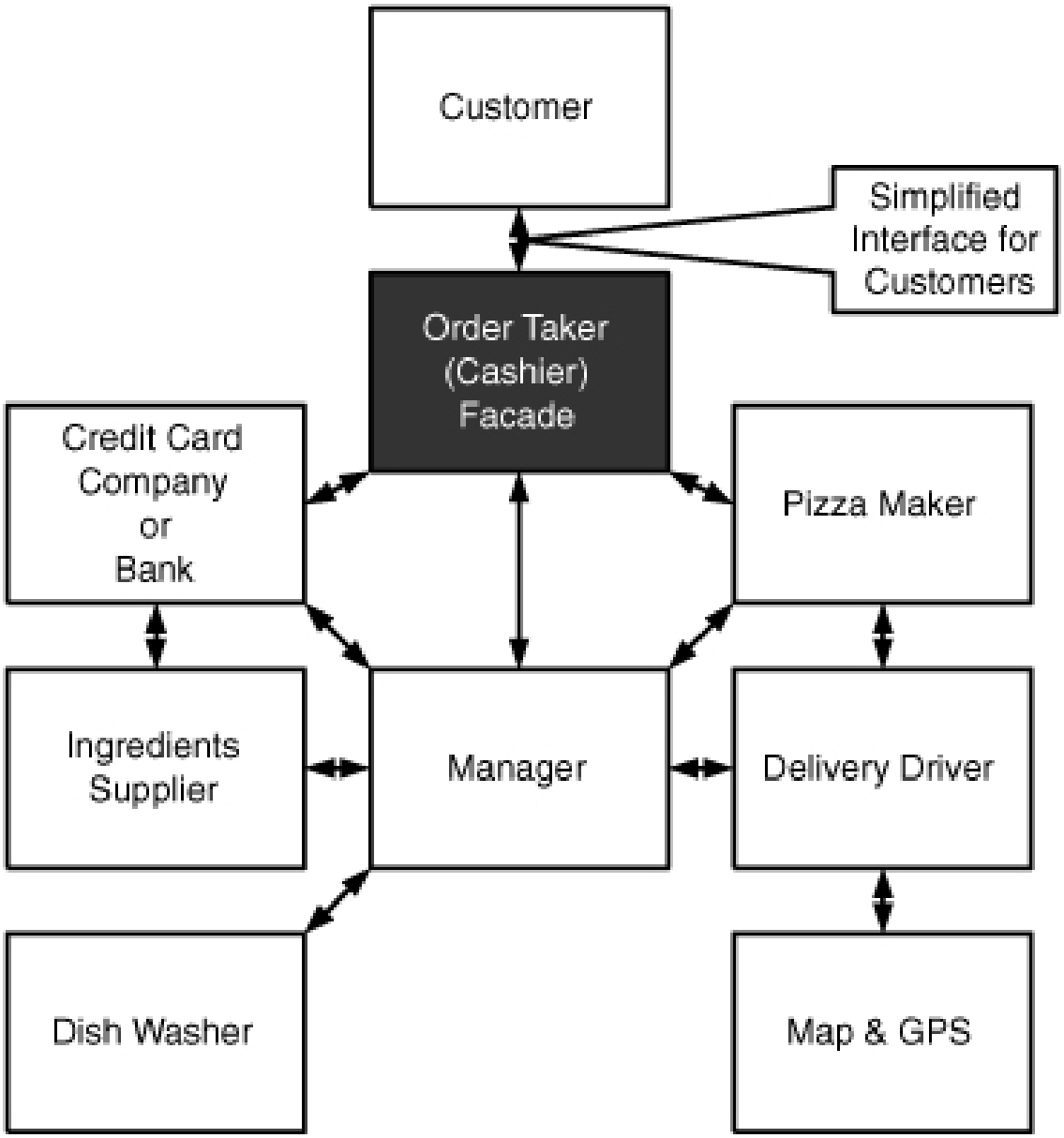

Figura 1.6: Ejemplo Pizzería: Diagrama de un sistema que utiliza el patrón Fachada [11]

Siguiendo este patrón nos ahorramos el acoplamiento entre la interfaz gráfica y el comportamiento específico de nuestra aplicación permitiendo la reutilización de las vistas y minimizando la duplicación de código.

1.4. Organización de la memoria

En este primer capítulo se ha realizado una aproximación al entorno que rodea a los dispositivos móviles en la actualidad, también se ha descrito en qué consiste el proyecto explicando de modo general las diferentes tareas que se llevarán a cabo y las motivaciones que nos han llevado a elegir este tipo de proyecto.

En los siguientes capítulos se desarrollará la evolución y construcción del proyecto.

El segundo capítulo trata sobre el análisis del proyecto donde se expone **qué** se va a realizar sin entrar en detalles sobre **cómo** se va a realizar, se explican las decisiones importantes que afectan al proyecto y se exponen los requisitos funcionales y no funcionales.

El tercer capítulo expone el diseño del proyecto, aquí si explicaremos el **cómo** se realizará la construcción de la aplicación. Se incluyen los diversos diagramas y esquemas conceptuales en UML, gráficos diversos, explicaciones sobre el flujo de datos y demás detalles relevantes para la realización de la implementación del sistema.

En el cuarto capítulo se tratarán los detalles más relevantes sobre la implementación y por lo tanto se explicarán en detalle la construcción de diversos elementos que forman parte de la aplicación como por ejemplo el nuevo *widget*[7] gráfico o el sistema de mensajería.

El quinto capítulo se exponen algunos datos sobre el testeo de la aplicación y las mejoras que se podrían realizar en un futuro.

En el sexto capítulo se extraerán las conclusiones de la realización de este proyecto.

Además los datos más pesados de leer se han incluido en los anexos.

[7]Pequeña aplicación, programa o vista

Capítulo 2

Análisis

El proceso de análisis es el primer paso en el momento de desarrollar software ya que nos delimitará el alcance del proyecto y gracias a ello podremos estructurar las siguientes fases de desarrollo de una forma ordenada y eficaz. Sin embargo esto no debería ser un proceso cerrado, si bien es cierto que debemos ceñirnos al resultado del análisis no deberíamos cerrar las puertas a cambios futuros que sean beneficiosos par al aplicación siempre y cuando no afecten de forma drástica al desarrollo del proyecto como por ejemplo que se extienda demasiado el tiempo de creación o que el presupuesto se fuese de madre (en caso de estar sujetos a un presupuesto).

2.1. Requisitos funcionales

Los requisitos funcionales son los que definen el comportamiento de nuestro sistema, explicitan qué podrá hacer el sistema y constituyen la base de prácticamente cualquier desarrollo de software. Como estamos realizando la ampliación de un proyecto anterior solamente vamos a incluir los requisitos funcionales que afectan directamente a nuestro proyecto obviando los que ya están implementados por la versión anterior de la aplicación.

2.1.1. Autenticación OAUTH

Es uno de los requisitos más importantes. La aplicación debe ofrecer al usuario el acceso a la API del Laboratorio de Calculo (LCFIB) que es necesaria para obtener los datos privados (Agenda, Avisos de asignaturas, Calendario), el acceso a la nueva API se realiza mediante el protocolo OAUTH y por lo tanto nuestra aplicación debe soportar dicho protocolo hecho que implica la implementación de un conjunto de funcionalidades específicas para la autenticación OAUTH. A continuación tenemos el listado de estas funcionalidades:

- Solicitud del *Request Token*[1]
- Validación del Request Token
- Obtención del *Access Token*
- Almacenado del *Access Token*
- Realización de accesos a las URL de la API firmadas con el *Access Token* y las claves de la aplicación
- Acceso al servidor de mensajería

[1] Cadena de caracteres

A excepción de la validación del request token todos los demás procesos son invisibles para el usuario, sin embargo para dicha validación el sistema ofrecerá una interfaz Web donde el usuario se conectará al sistema del LCFIB para otorgar o denegar permisos a nuestra aplicación para acceder a sus datos remotos del LCFIB. En caso que no se otorguen los permisos la aplicación no podrá ofrecer gran parte de sus funcionalidades.

2.1.2. Mensajería

Junto con la autenticación OAUTH es el requisito más importante del sistema, ya que es el eje principal de la ampliación del Racó Mobile. Nuestro sistema deberá proporcionar un sistema de mensajería integral que cubra las necesidades comunicativas surgidas de la realización de prácticas en grupo y otras tareas derivadas de la facultad.

Independientemente de la naturaleza de los motivos comunicativos nuestra aplicación debe ofrecer un conjunto de servicios que permitan al usuario comunicarse con el resto de un modo fluido y ágil a la vez que cómodo. Para que esto sea posible la aplicación deberá disponer de las siguientres funcionalidades:

- Comunicación con el servidor de mensajería
- Un listado de todos los chats iniciados y no eliminados, con distintivos para diferenciar los chats normales de los grupales
- Un sistema de eliminación de chats iniciados
- Un listado de los mensajes enviados y recibidos desde donde se podrán enviar más mensajes
- Un listado actualizado de los usuarios *amigos*
- Un sistema para añadir nuevos usuarios *amigos*
- Un sistema para eliminar usuarios *amigos*
- Un sistema de creación de chats tanto individuales como grupales
- Control del estado de la conexión activando y desactivando opciones según el estado de la red

Todo ello permitirá al usuario interactuar con el cliente de mensajería de forma dinámica y ágil.

2.1.3. Notificaciones

Este requisito funcional servirá como complemento del anterior ya que están estrechamente relacionados. Las notificaciones informarán al usuario en los siguientes casos:

- Información de la recepción de un mensaje de chat (texto + sonido opcional)
- Información de la recepción de un mensaje de chat grupal (texto + sonido opcional)

Este sistema deberá ser escalable para poder añadir en un futuro notificaciones de otro tipo no relacionadas exclusivamente con la mensajería.

2.1.4. Persistencia de datos

Los datos generados por el chat deberán almacenarse de un modo eficiente para satisfacer las necesidades de la mensajería, por lo tanto deberán ser accesibles de un modo ágil, ordenables, actualizables y se deberán poder eliminar tanto de modo individual como en conjunto. Se guardara información relativa a los mensajes siguiendo las siguientes directrices:

- Quien nos ha enviado el mensaje o a quien se lo hemos enviado
- Cuando se ha enviado
- Cuando se ha recibido (solo los mensajes propios)
- Que texto se ha enviado
- Que texto se ha recibido

En lo relativo a otros datos como por ejemplo los horarios, asignaturas, noticias, avisos.... de esto se encargaba el sistema ya existente en la primera versión de la aplicación y por lo tanto se va mantener. Sin embargo se añadirá funcionalidades que permitan el reseteo de dicha información.

2.1.5. Interfaz

La interfaz va a adaptarse tanto para acoger las nuevas funcionalidades como para modernizar la apariencia de la aplicación, para ello nuestra aplicación debe contar con elementos novedosos que se ajusten a las necesidades del usuario. Para ello se ha elaborado un listado con las acciones que se deben llevar a cabo para que la aplicación se ajuste al estilo de interfaz que espera encontrar el usuario:

- Se creará un widget similar a la famosa *Clickwheel*[2] para pasar entre las distintas secciones
- Se creará una pantalla para mostrar las conversaciones de forma agradable, con globos de texto de colores
- Se crearán pantallas para acoger el resto de listados (lista amigos, chats....)
- Se eliminará la función de correo de todas las pantallas ya que no se utiliza

2.2. Requisitos no funcionales

Los requisitos no funcionales son las características que deberá tener nuestro proyecto pero que no están relacionadas de manera directa con las funcionalidades del mismo. Se podría decir que garantizan la calidad de nuestro producto y que en un principio se presuponen como presentes en el sistema, sin embargo es mejor dejar constancia de qué va a ofrecer nuestra aplicación y del motivo por el que lo va a hacer.

En este proyecto los requisitos no funcionales detectados han sido los siguientes:

- Eficiencia: Los terminales iPhone son por definición potentes, sin embargo eso no quiere decir que debamos descuidar la eficiencia en el momento de desarrollar el sistema, entre otros factores cuanto menos eficiente sea un programa más batería consumirá ya que ejecutará una mayor cantidad de operaciones y transmitirá más información que la estrictamente necesaria. así que intentaremos minimizar la ejecución de operaciones innecesarias y transmitiremos los datos esenciales para el buen funcionamiento de la aplicación.

[2]Interfaz táctil en forma de rueda que permitía controlar los Ipod clásicos

- Escalabilidad: El sistema debe mantenerse escalable, debe tener la posibilidad de crecer y expandir sus funcionalidades sin un esfuerzo extra excesivo. Nuestro proyecto parte ya de una aplicación existente y el hecho de que sea escalable nos facilita el trabajo y la ampliación de la misma, es por ello que es uno de los factores más importantes en una aplicación que tiende a evolucionar con el paso del tiempo.
- Mantenibilidad: El sistema debe tener la cualidad de ser mantenible, se debe poder solucionar la mayor parte de los problemas que surjan o bien incluir cambios solicitados por el cliente o demandado por los usuarios. Si el sistema es demasiado complejo de mantener esto puede provocar que los usuarios dejen de utilizar la aplicación por una respuesta demasiado lenta por parte del equipo de mantenimiento.
- Integridad: La seguridad de los datos almacenados en el dispositivo o bien los que se transmiten a través de la red deben permanecer seguros en todo momento, ello otorga un valor añadido a la aplicación y aumenta la confianza del usuario quien puede utilizar la aplicación con total tranquilidad al saber que sus datos no se van a ver comprometidos.
- Robustez: El sistema debe ser sólido y estable. Un sistema que falla constantemente no nos da seguridad ni comodidad, por lo tanto debemos dotar el sistema con respuestas fiables antes posibles fallos, tanto de la propia aplicación como de la transmisión de datos.
- Actual: Las tecnologías aplicadas en el proyecto deberían ser las más modernas de las que podamos disponer en el momento. Construir un proyecto sobre tecnologías antiguas u obsoletas (como versiones anteriores del sistema operativo) puede provocar agujeros de seguridad, falta de rendimiento o comportamientos anormales en el flujo normal de la aplicación. Por lo tanto trataremos de utilizar las versiones más recientes y fiables que podamos encontrar.
- Reutilizable: Esto no es algo indispensable pero si muy recomendable. Los elementos generados durante el desarrollo del proyecto deberían poderse reutilizar en un futuro, o bien para mejorar el propio proyecto o bien para formar parte de un proyecto totalmente nuevo. Por lo tanto intentaremos realizar el proyecto de forma modular para que los diferentes componentes se puedan reutilizar en proyectos futuros.
- Usabilidad: La aplicación debe ser útil y fácil de utilizar, de cualquier otro modo la aplicación no será utilizada y por lo tanto el proyecto podrá considerar que ha sido un fracaso ya que una aplicación que nadie utiliza no sirven para nada. así pues trataremos de realizar una aplicación intuitiva, que se pueda aprender a usar rápidamente y que realmente sirva al usuario.

Si conseguimos que nuestro sistema tenga todos estos elementos podremos estar seguros de que hemos realizado un producto de calidad y que muy probablemente sea utilizado por los usuarios a los que va dirigido.

Estos no son todos los requisitos funcionales, son solo los referidos a la calidad del software, también tenemos los requisitos económicos, estructurales y políticos que podemos resumir en el siguiente listado:

- El diseño debe guardar relación con la Facultad de Informática
- La aplicación debe ofrecerse de manera gratuita sin cobro alguno por sus servicios
- La aplicación debe ser lo más simple posible siguiendo el famoso patrón KIS (Keep It Simple)

2.3. Decisiones de diseño

Es importante que antes de entrar de lleno en la especificación y el diseño de la aplicación se comenten algunas decisiones importantes que van a afectar directamente esas dos etapas del proyecto.

En primera instancia debo matizar que la versión de OAUTH utilizada en esté proyecto será la *RFC 5849: The OAuth 1.0 Protocol* de Abril del 2010 principalmente por dos motivos, la versión 2.0 todavía no es un estándar aprobado y ésta es la versión para la cual están preparadas las API del Racó que nos permitirán obtener los datos de los usuarios.

Para la parte de mensajería se utilizará también un protocolo estandarizado llamado XMPP [13] [9], más comúnmente conocido por el nombre de Jabber. Esta elección se sustenta en que Jabber es uno de los protocolos de mensajería más utilizados (aunque se utilice para mensajería en realidad se puede utilizar para cualquier fin que implique el intercambio de mensajes), el servicio gmail sin ir más lejos utiliza este protocolo, además existen varias soluciones libres del lado del servidor, es capaz de comunicarse con otros servicios como MSN o AIM, esta basado en XML y es multiplataforma. De entre las opciones disponibles para el servidor hemos elegido **Openfire** [6], tanto por ser libre como por permitir la creación de plugins y complementos que nos permitirán remodelar el servidor para adaptarlo plenamente a nuestras necesidades.

Como último detalle comentar que la aplicación estará realizada en base al último sistema iOS en el mercado, la versión 5.0.

Capítulo 3

Especificación y diseño

Esta es una de las secciones más importantes de este documento ya que en ella describiremos la estructura interna que va a seguir nuestro sistema, desde la arquitectura hasta los casos de uso particulares. Todo ello nos ayudará más adelante cuando implementemos el proyecto ya que nos proporcionará una guía a seguir en la construcción de cada uno de los elementos y artefactos que unidos formarán el sistema real.

3.1. Arquitectura del sistema

Nuestra aplicación no puede existir sin conexión a internet, ya que los datos que se recogen y almacenan provienen todos de servicios que están disponibles en la red, ya sea de forma pública o privada y por lo tanto nuestro sistema no estará formado únicamente por la aplicación sino que además deberemos añadir todos aquellos recursos a los que accedemos en un momento u otro.

Como ya hemos indicado que el proyecto es la ampliación de uno existente y por lo tanto partimos de una base tanto a nivel funcional como a nivel arquitectónico, partiendo de esa base hemos ampliado los componentes que conforman el sistema para adecuarse a las nuevas funcionalidades a la vez que hemos minimizado el impacto sobre los componentes existentes mientras maximizábamos el aprovechamiento de los recursos disponibles. En la figura-3.1 se puede ver una comparativa entre la arquitectura existente al inicio del proyecto y la arquitectura utilizada por la nueva versión del mismo, debo decir que únicamente se han añadido los componentes propios, ya que por ejemplo el servidor de mensajería se conecta al servidor *APNS*[1] de Apple para mandar las notificaciones Push.

A continuación pasaremos a describir y justificar porqué hemos optado por esta configuración y porqué creemos que es mejor que otras alternativas.

La arquitectura antigua estaba constituida por tres componentes, la aplicación, el servidor del Racó y la base de datos a la que accede dicho servidor. Esta estructura era la más lógica ya que la aplicación solicita al servidor los datos almacenados en el Racó, el servidor los lee de base de datos y se los manda. Se podría decir que no hay otra estructura más simple y eficaz para las necesidades que presentaba la aplicación y por lo tanto era una construcción sólida y robusta.

Con la inclusión de las nuevas funcionalidades la cosa cambia, de hecho es el chat lo que hace que todo cambie pues no es suficiente tener acceso a los datos disponibles mediante el servidor del Racó sino que ahora necesitamos un sistema que gestione el servicio de mensajería y se encargue de mandar las notificaciones Push relacionadas y como hemos comentado en la sección 2.3, este

[1] Apple Push Notification Service

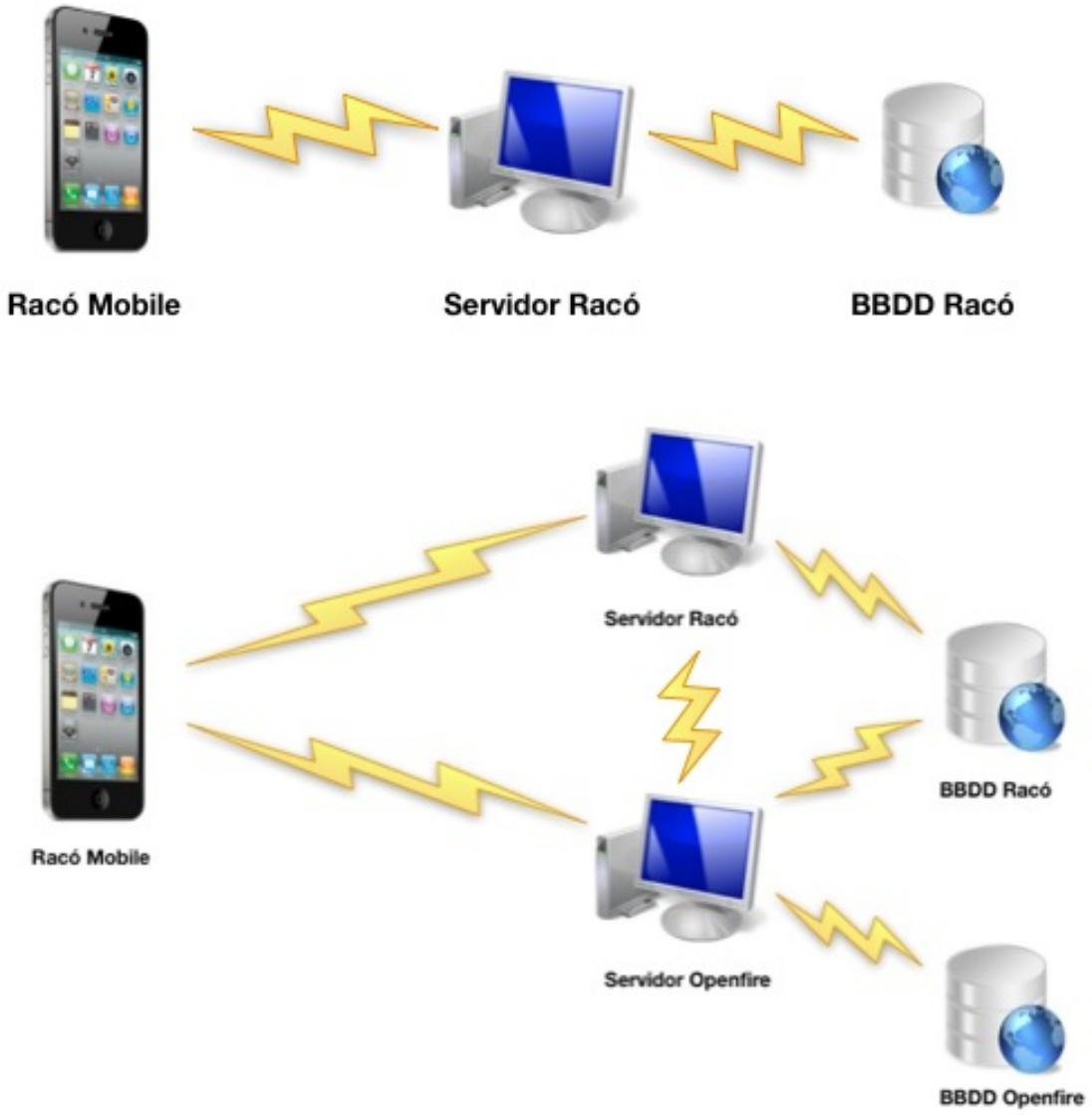

Figura 3.1: Comparativa arquitectura del sistema versión 1 (arriba) y versión 2 (abajo)

sistema será el servidor XMPP Openfire.

Si echamos un vistazo a la nueva arquitectura veremos que ahora disponemos de la misma estructura antigua a la que se ha añadido el servidor de mensajería que accederá a su propia base de datos en la que gestiona los mensajes no enviados, logs y las relaciones entre usuarios además de muchos otros aspectos y también accederá a la base de datos del Racó para poder obtener el listado completo de usuarios sin tener que darlos de alta uno por uno. Se debería aclarar que el servidor Openfire accederá a la base de datos del Racó en modo de **sólo lectura** por lo que será imposible que se realice ninguna modificación intencionada o accidental a los datos almacenados en esa BBDD. Vemos que el servidor Openfire también se conecta con el servidor del Racó, lo que ocurre en realidad es que es el plugin encargado de mandar las Push el que se conecta al servidor del Racó que es donde están almacenados los tokens de los dispositivos iOS a los que se deben mandar las notificaciones.

Los motivos que nos han llevado a de elegir esta arquitectura han sido una vez más la simplicidad, la eficacia y la minimización del impacto de las nuevas funcionalidades sobre el sistema existente. Hemos aprovechado todas las características que nos brinda openfire, por una parte lo hemos podido conectar sin problemas a la base de datos de usuarios del Racó y por otro hemos desarrollado un plugin para Openfire que se encarga de enviar las notificaciones push, pues es el mejor lugar para generar las notificaciones Push ya que el plugin tiene acceso a toda la información del servicio de mensajería hecho que nos permite construir y enviar las notificaciones de un modo rápido y preciso.

así pues esta arquitectura nos brinda todas las funcionalidades que necesitamos para poder cubrir el alcance de nuestro proyecto a la par que no alteramos el equilibrio que existía antes de esta nueva versión por lo que consideramos que es una estructura sólida y viable que sería capaz de soportar la introducción de nuevos cambios y componentes en el futuro si fuese necesario.

3.2. Mapa navegacional

La llegada de la rueda táctil a la interfaz ha provocado que el mapa navegacional de la primera versión (figura-3.2) quedase desfasado y por lo tanto se ha tenido que diseñar uno nuevo, sin embargo se ha aprovechado algunas características del antiguo como veremos más adelante.

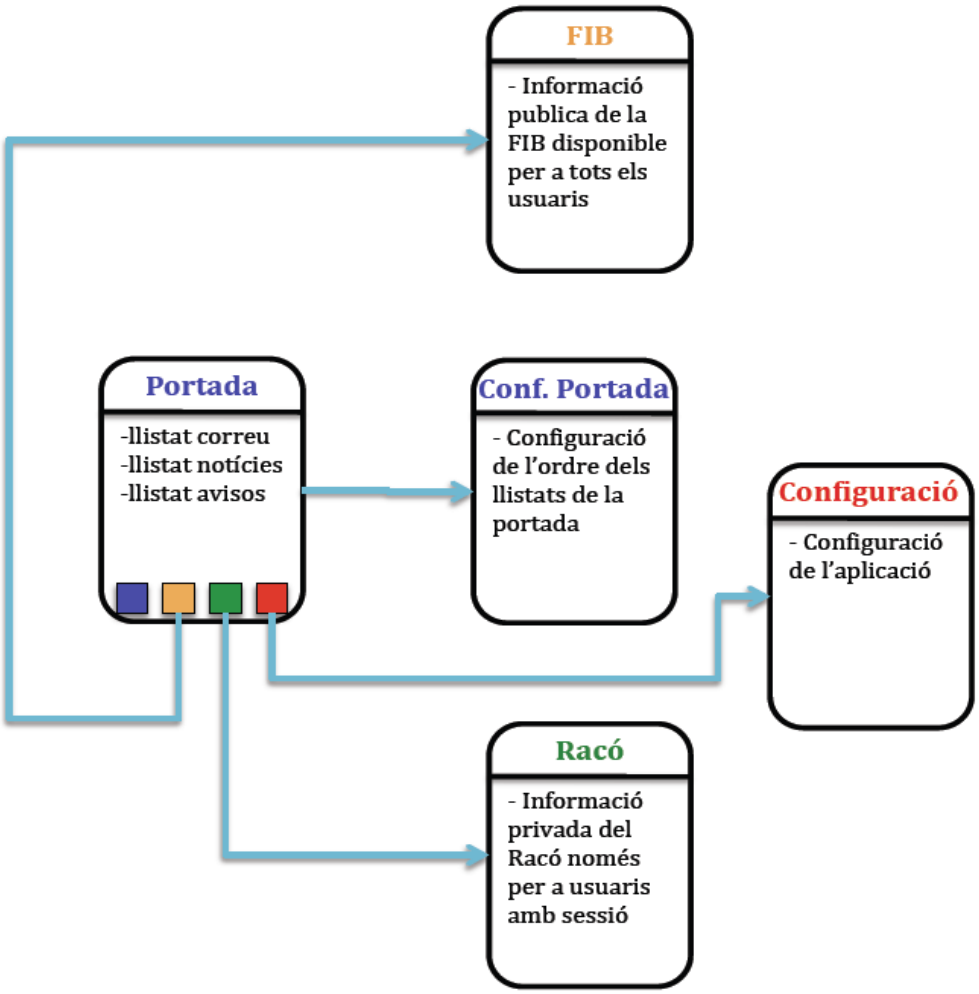

Figura 3.2: Mapa navegacional de la primera versión del Racó Mobile [10]

La figura-3.3 muestra las vistas principales accesibles desde el nuevo widget de la rueda y la relación entre unas y otras. Una de las desventajas de la nueva rueda respecto a la anterior barra es que antes se podía acceder a cualquiera de las vistas pulsando el icono correspondiente en la barra, ahora estamos obligados a pasar por una o más vistas hasta llegar a la sección deseada. Esto es debido a una decisión estética, pues si reducimos el tamaño de los segmentos de la rueda podemos saltar fácilmente entre una y otra sección sin tener que pasar por las contiguas pero la sensación gráfica sería tan agradable.

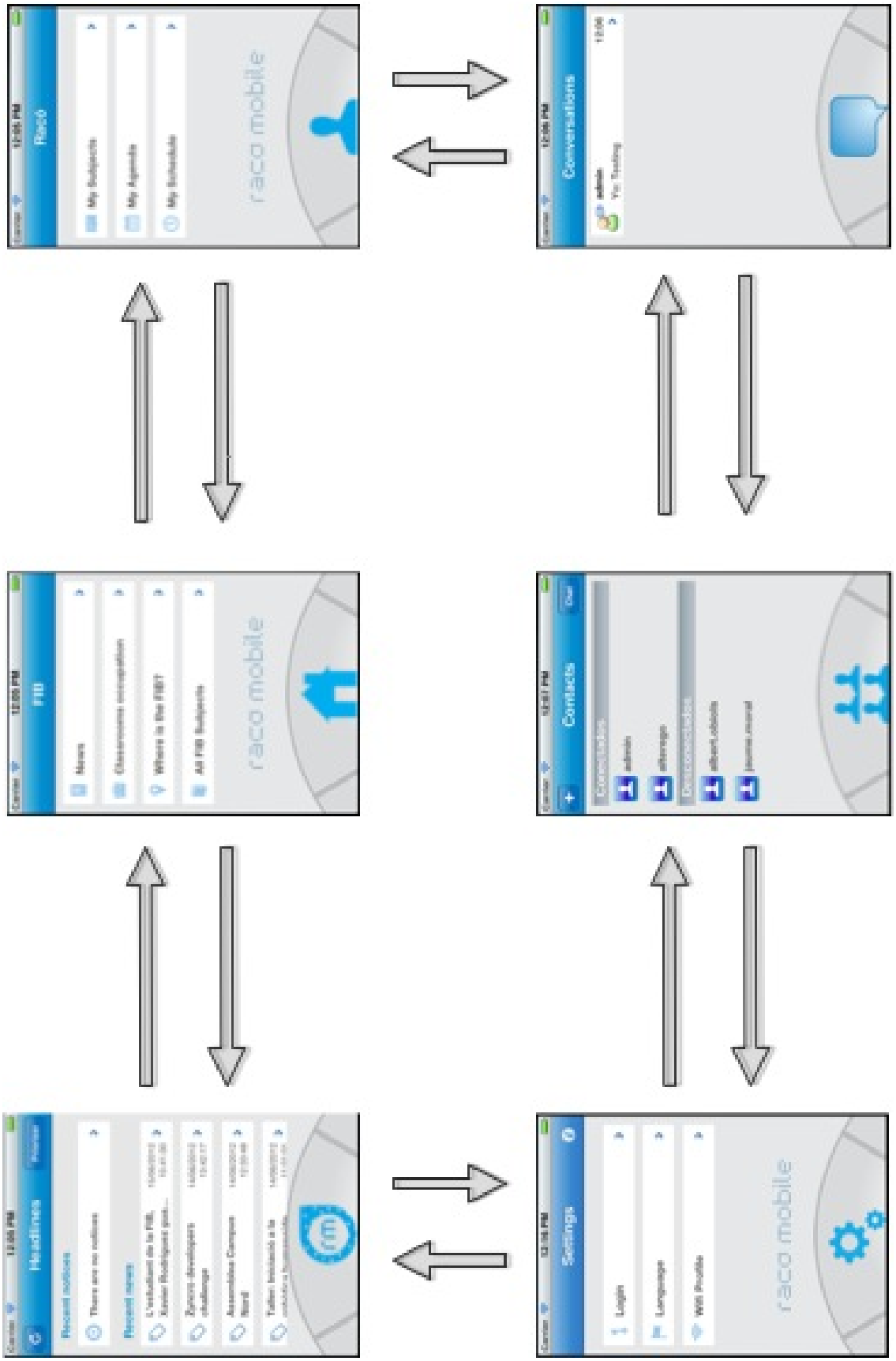

Figura 3.3: Mapa navegacional de la Rueda Táctil

La navegabilidad una vez dentro de cada sección se ha mantenido como en la versión anterior, se esconde la rueda y se da paso a la navegación entre las distintas vistas de un modo lineal, o dicho de otro modo una vista lanza otra vista interna que a su vez puede lanzar otra vista interna y así hasta las que sean necesarias, todas estas pantallas tendrán un botón atrás o equivalente que permita volver a la pantalla desde la cual proceden, una vez se vuelve a la pantalla original la rueda táctil se vuelve a mostrar para habilitar el cambio entre secciones. En la figura-3.4 podemos observar este comportamiento en las secciones de Chat y Contactos.

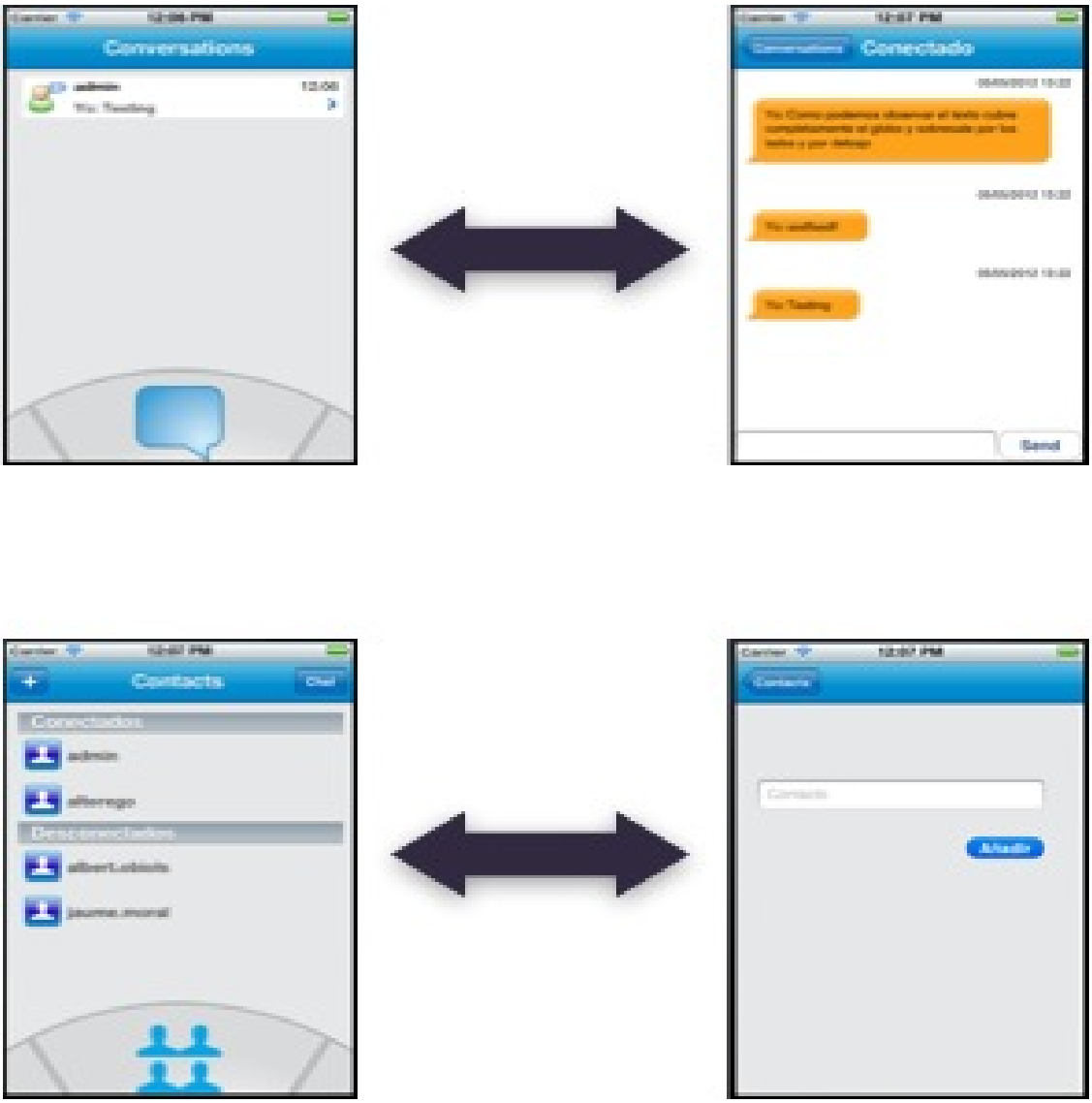

Figura 3.4: Mapa navegacional de las pantallas de Chat y Contactos

También se ha decidido mantener la aparición de la ventana de login (en este caso de obtención de token OAUTH) si un usuario que no tiene un token OAUTH intenta acceder a alguna de las secciones que lo requieren, además de la opción particular en el menú de la sección de configuración. Esto queda ilustrado por la figura-3.5.

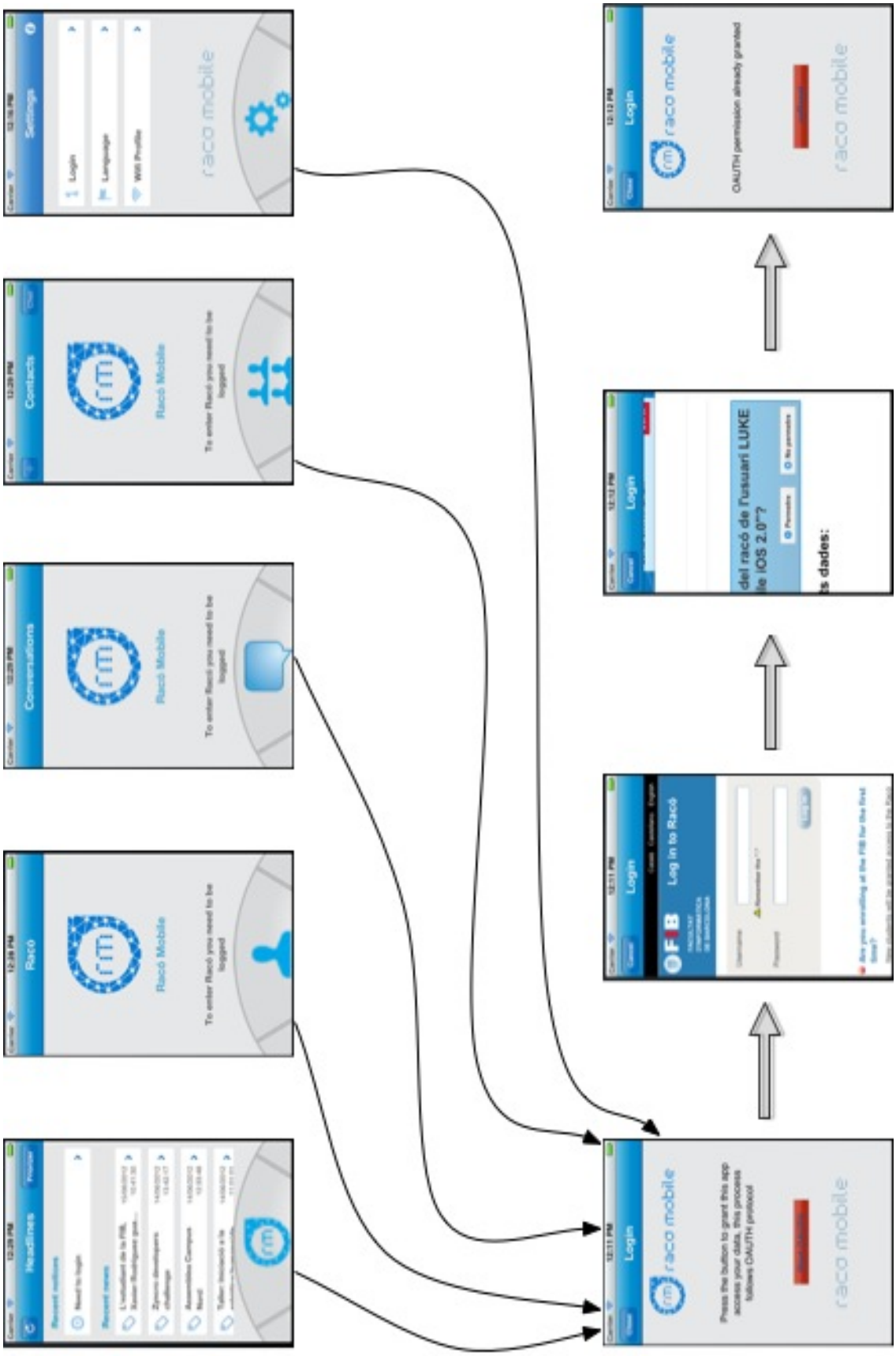

Figura 3.5: Mapa navegacional de la obtención del Token OAUTH

3.3. Actores

Un actor es cualquier entidad externa que tiene relación con algún caso de uso presente en el sistema y que interviene activa o pasivamente en uno o más escenarios.

Como este proyecto es el resultado de la ampliación de un proyecto existente es necesario comentar los actores definidos durante la creación de la primera versión del Racó Mobile. En la documentación se indica de forma clara la existencia de un actor, el Usuario, el cual se puede subdividir en dos tipos más concretos el **usuario con sesión** y el **usuario sin sesión**. Los primeros representan a aquellos que utilizan la aplicación habiéndose autenticado como alumnos/profesores del Racó de la FIB, mientras que los segundos son todos aquellos que no lo han hecho, tanto usuarios externos sin acceso alguno al Racó como usuarios que por algún motivo no han decidido autenticarse, podemos ver esta distribución en el esquema original representado en la figura-3.6.

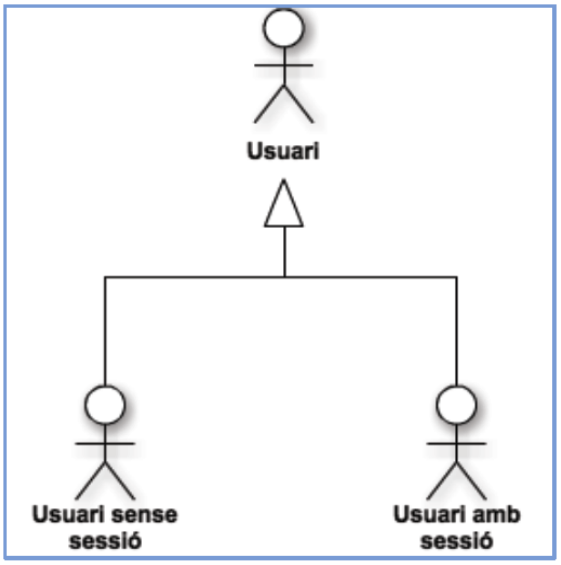

Figura 3.6: Esquema original de los actores en Racó Mobile 1.0 [10]

Durante el desarrollo de la segunda versión del proyecto no se ha considerado necesario añadir ningún tipo de actor más ya que las funcionalidades añadidas no dan pie a la aparición de dichas entidades, aunque tras analizar los cambios realizados en la autenticación de los usuarios que ahora se realiza por medio de OAUTH se ha decidido cambiar la terminología y en lugar de **usuarios con sesión** y **usuarios sin sesión** ahora pasaremos a hablar de **usuarios con token** y **usuarios sin token**.

3.4. Definición de los casos de uso

Los casos de uso representan una descripción detallada de las funcionalidades del sistema y la interacción de los usuarios con estas funcionalidades. A continuación se presentará una descripción detallada de cada caso de uso presente en el sistema con la siguiente estructura:

- Condiciones previas: Detalla las condiciones que se deben cumplir para poder llevar a cabo el escenario principal del caso de uso.
- Descripción detallada: Se explica con detalle en que consiste el caso de uso para cada uno de los actores que intervengan

- Escenario principal: Se detalla paso por paso en qué consiste el escenario ideal
- Escenarios alternativos: Se detallan los escenarios que complementan al principal en caso que la ejecución no sea idílica.
- Diagrama de secuencia: Descripción gráfica del caso de uso

Todos los casos de uso se presentarán con la descripción y el escenario principal como mínimo, los escenarios alternativos se añadirán en caso de que existan y los diagramas de secuencia únicamente se realizarán para los casos que resulten más interesantes ya que todos han seguido una pauta similar en el momento de ser diseñados.

3.4.1. Caso de uso: Obtención de token OAUTH

- Condiciones previas: El dispositivo debe disponer de una conexión a internet activa y el usuario debe disponer de acceso al Racó.
- Descripción detallada: El usuario será capaz de obtener un token OAUTH que le permitirá acceder a las secciones restringidas de la aplicación que son el Racó, el sistema de chat y el listado de contactos. A partir de este momento también será capaz de recibir notificaciones remotas.
- Escenario principal:
  1. El usuario accede al menú login desde configuración o bien trata de acceder a una zona restringida
  2. La aplicación presenta la pantalla de obtención del token OAUTH
  3. El usuario hace click sobre el único botón de la pantalla que inicia todo el proceso
  4. La aplicación realiza la petición del request token
  5. La aplicación recibe un request token del servidor
  6. La aplicación muestra una webView con la pantalla de autorización del Racó
  7. El usuario introduce su login y password y pulsa el botón de login
  8. Se presenta ante el usuario un formulario que indica la concesión de permisos a los datos por parte del Racó Mobile
  9. El usuario pula el botón *Permitir*
  10. La aplicación utiliza el request token (ya autorizado) para pedir al servidor un access token
  11. El servidor devuelve un access token a la aplicación
  12. La aplicación registrará el identificador del dispositivo en la base de datos del servidor Push
  13. La aplicación obtendrá el nombre de usuario mediante la API del Racó
  14. La aplicación mostrará un mensaje indicando que el proceso ha finalizado correctamente y cargará los datos restringidos
- Escenarios alternativos:
  1. El usuario accede al menú login desde configuración habiendo obtenido ya un access token
  2. La aplicación muestra un texto explicativo indicando que ya se dispone de un token y un botón Reset
  3. El usuario pulsa el botón reset
  4. La aplicación muestra un aviso sobre el borrado de datos que se producirá

5. El usuario pulsa el botón aceptar
6. La aplicación borra los datos del usuario y bloquea la pantalla avisando de la necesidad de reinicio de la misma

- Diagrama de secuencia: A continuación en la Figura-3.7 presentamos un esquema gráfico del flujo provocado por este caso de uso. El esquema ha sido algo simplificado por temas de espacio ya que se han eliminado las tareas realizadas por *JSONParser*[2] y por *RacoMobileAppDelegate*[3] que en este caso no resultan muy relevantes.

[2]Clase encargada de obtener objetos en formato JSON y leerlos correctamente
[3]Clase principal del sistema

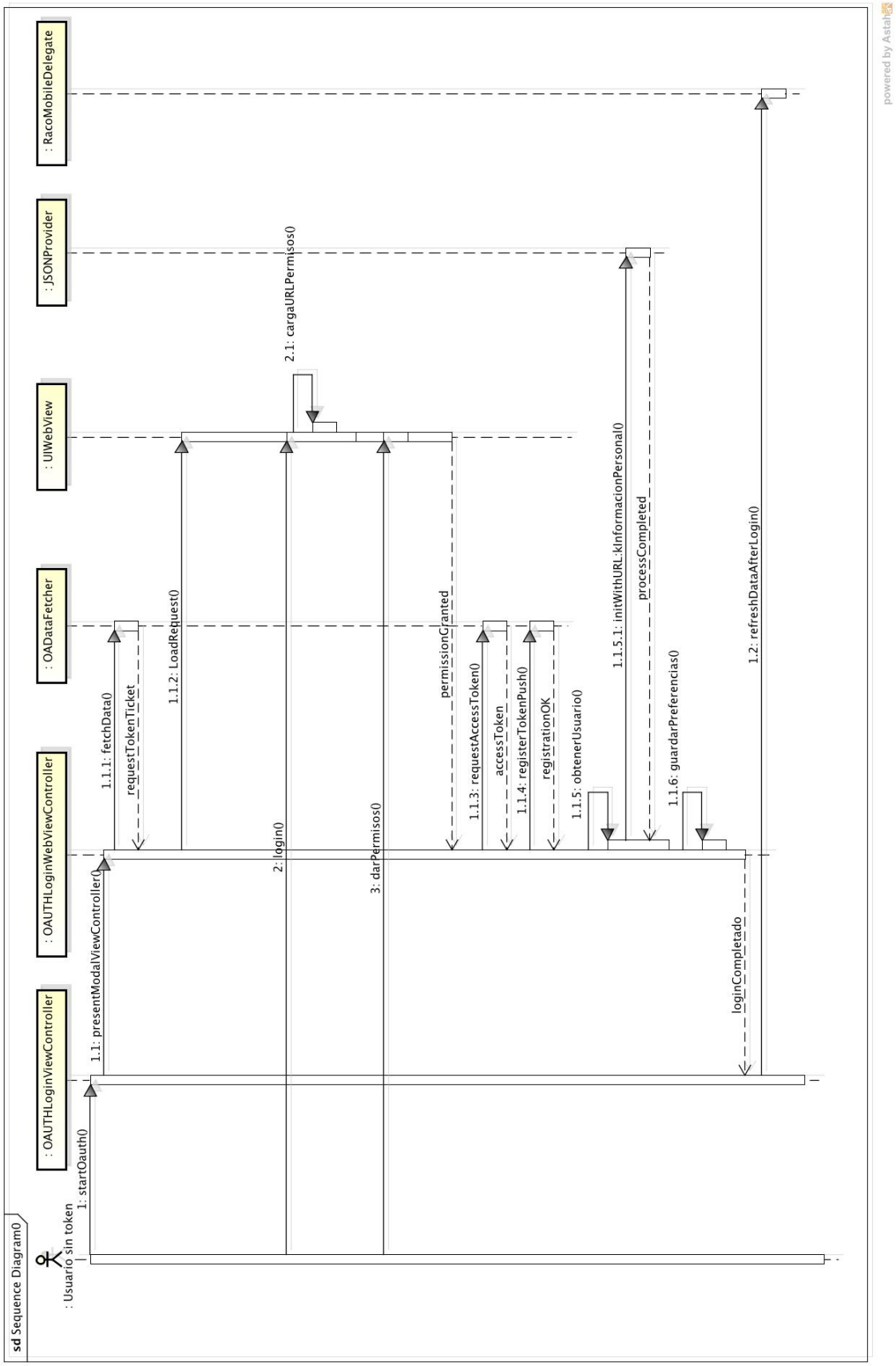

Figura 3.7: Diagrama de secuencia de la obtención de un Token OAUTH

3.4.2. Caso de uso: Consulta del listado de chats

- Condiciones previas: El usuario debe haber obtenido un token OAUTH.
- Descripción detallada: Accediendo a la sección de listado de chats de la aplicación el usuario podrá visualizar los chats activos en el sistema, pudiendo ver para cada uno la siguiente información:
  - Un icono indicando si el chat es normal o grupal
  - Nombre del usuario o nombre del grupo del chat
  - Fecha del último mensaje recibido/enviado
  - Texto y remitente del último mensaje recibido/enviado
  - Cantidad de mensajes sin leer
- Escenario principal:
  1. El usuario accede a la pantalla de chat
  2. La aplicación consulta los chats activos en el sistema de persistencia de datos
  3. La aplicación carga y formatea los datos de los chats activos en la tabla correspondiente
- Escenarios alternativos:
  1. El usuario sin token accede a la pantalla de chat
  2. La aplicación muestra la pantalla de obtención de token y se continúa con el flujo de la sección 3.4.1

3.4.3. Caso de uso: Consulta mensajes

- Condiciones previas: El usuario debe haber obtenido un token OAUTH.
- Descripción detallada: Accediendo a la sección de listado de mensajes de un chat activo el usuario podrá visualizar los mensajes enviados y recibidos en dicho chat.
- Escenario principal:
  1. El usuario accede a la pantalla de chat
  2. La aplicación consulta los chats activos en el sistema de persistencia de datos
  3. La aplicación carga y formatea los datos de los chats activos en la tabla correspondiente

3.4.4. Caso de uso: Consulta del listado de contactos

- Condiciones previas: El usuario debe haber obtenido un token OAUTH.
- Descripción detallada: Accediendo a la sección de contactos de la aplicación, el usuario podrá visualizar un listado con los contactos añadidos al sistema de mensajería, además si tiene conexión activa a internet se mostrará el estado de conexión de cada contacto aunque únicamente recibiremos notificaciones de conexión de usuarios que hayan aceptado nuestras peticiones de amistad.
- Escenario principal:
  1. El usuario accede a la pantalla de contactos
  2. La aplicación consulta los contactos existentes en el sistema de persistencia de datos
  3. La aplicación carga y formatea los datos de los contactos y los muestra en la tabla correspondiente

- Escenarios alternativos:
  1. El usuario sin token accede a la pantalla de contactos
  2. La aplicación muestra la pantalla de obtención de token y se continúa con el flujo de la sección 3.4.3

3.4.5. Caso de uso: Añadir contactos

- Condiciones previas: El usuario debe haber obtenido un token OAUTH y disponer de conexión a internet activa
- Descripción detallada: Al acceder a la pantalla de añadir contactos (accesible desde la pantalla de listado de contactos) el usuario podrá enviar invitaciones a otros usuarios introduciendo su identificador del Racó.
- Escenario principal:
  1. El usuario accede a la pantalla de añadir contactos
  2. El usuario escribe el identificador de otro usuario y pulsa el botón *Añadir*
  3. La aplicación manda una invitación de amistad al contacto indicado por el usuario
  4. La aplicación añade al contacto al listado de contactos e indica que la petición se ha realizado con éxito
  5. El usuario puede continuar mandando más invitaciones o cerrar la pantalla
- Diagrama de secuencia: En el esquema de la figura-3.8 se muestra el flujo de información cuando el usuario manda una petición, claramente podemos observar como la clase XMPP-Framework [5] está actuando siguiendo el patrón Fachada.

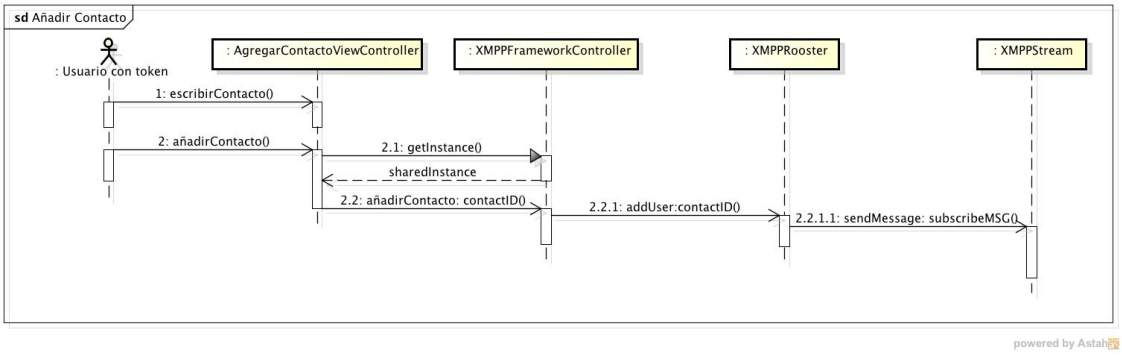

Figura 3.8: Diagrama de secuencia del caso de uso Añadir Contactos

3.4.6. Caso de uso: Eliminar contacto

- Condiciones previas: El usuario debe haber obtenido un token OAUTH y disponer de conexión a internet activa y algún contacto agregado.
- Descripción detallada: Al acceder a la pantalla de contactos el usuario podrá eliminar a sus contactos del listado realizando un movimiento de arrastre sobre el contacto y pulsando el botón aceptar.
- Escenario principal:

1. El usuario accede a la pantalla de listado de contactos
2. La aplicación consulta los contactos existentes en el sistema y muestra un listado actualizado
3. El usuario realiza un movimiento de arrastre sobre el contacto que desea eliminar
4. La aplicación entre en modo edición y muestra un botón *Eliminar* sobre dicho contacto
5. El usuario pulsa el botón eliminar
6. La aplicación envía la petición al servidor y elimina el contacto de persistencia

3.4.7. Caso de uso: Gestionar una petición de amistad

- Condiciones previas: El usuario debe haber obtenido un token OAUTH y disponer de conexión a internet activa.
- Descripción detallada: Cuando un usuario decide mandarnos una invitación de amistad el usuario debe ser capaz de aceptar o denegar dicha solicitud.
- Escenario principal:
  1. La aplicación se conecta al servidor de mensajería
  2. El servidor manda a la aplicación una invitación de amistad dirigida a nuestro usuario
  3. La aplicación muestra un aviso indicando la recepción de la invitación
  4. El usuario decide si aceptar o rechazar la invitación
     a) El usuario acepta la solicitud de amistad
        1) La aplicación añade el usuario que nos manda la petición al listado de contactos
     b) El usuario rechaza la solicitud de amistad

3.4.8. Caso de uso: Crear un chat convencional

- Condiciones previas: El usuario debe haber obtenido un token OAUTH y disponer de conexión a internet activa y usuarios añadidos en la lista de contactos.
- Descripción detallada: Accediendo al listado de contactos el usuario puede seleccionar cualquiera de ellos de la lista e iniciar una conversación con el, tanto si el otro usuario se encuentra conectado como si se encuentra desconectado. En caso de encontrarse desconectado los mensajes le llegarían cuando se volviese a conectar.
- Escenario principal:
  1. El usuario accede a la pantalla de contactos
  2. La aplicación muestra el listado de contactos actualizado
  3. El usuario selecciona un usuario de entre los disponibles en el del listado de contactos
  4. El usuario pulsa el botón *Chat*
  5. La aplicación crea un nuevo chat en caso de no existir uno con el mismo usuario
  6. La aplicación redirige al usuario al listado de chats donde puede acceder al chat creado que aparecerá el primero de la lista.
- Diagrama de secuencia: En el esquema de la figura-3.9 se muestra el flujo de información cuando el usuario decide iniciar un chat. Se puede apreciar claramente como se utiliza el patrón MVC.

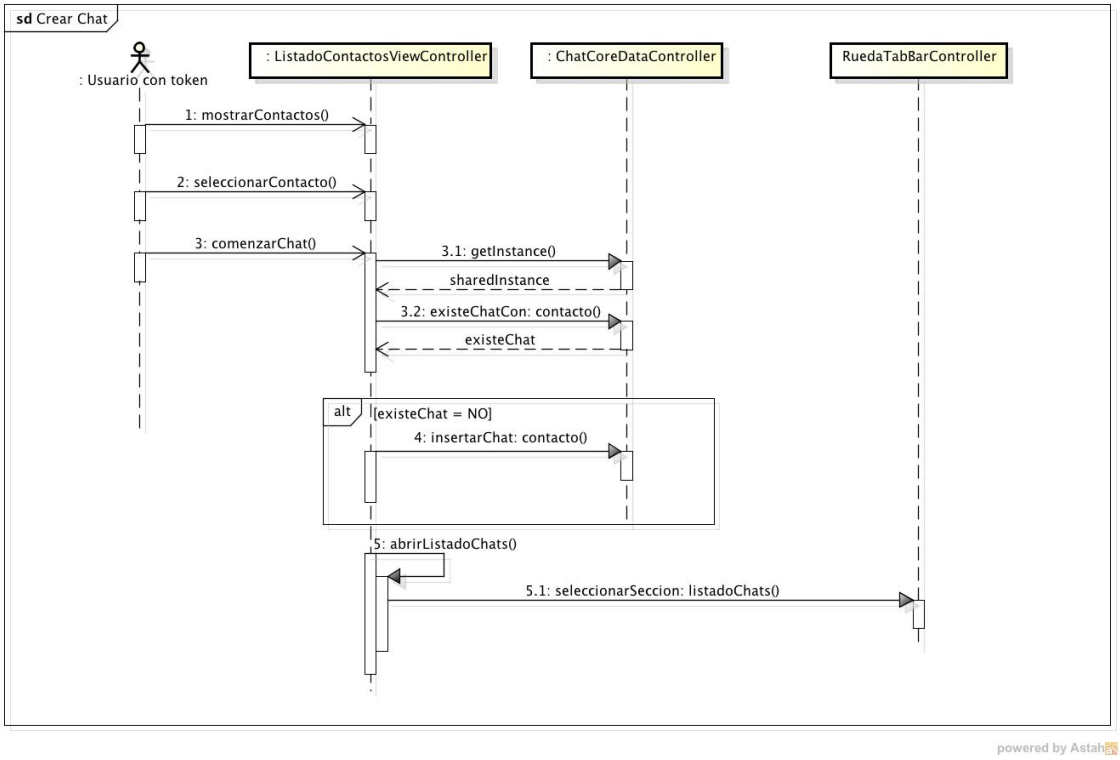

Figura 3.9: Diagrama de secuencia del caso de uso Crear un chat convencional

3.4.9. Caso de uso: Crear un chat multiusuario

- Condiciones previas: El usuario debe haber obtenido un token OAUTH y disponer de conexión a internet activa y usuarios añadidos en la lista de contactos.
- Descripción detallada: Accediendo al listado de contactos el usuario puede seleccionar hasta un límite de 29 contactos distintos para crear una sala de chat multiusuario, además tendrá la potestad de asignar un nombre identificativo para la sala.
- Escenario principal:
  1. El usuario accede a la pantalla de contactos
  2. La aplicación muestra el listado de contactos actualizado
  3. El usuario selecciona más de un usuario de entre los disponibles en el listado de contactos.
  4. El usuario pulsa el botón *Chat*
  5. La aplicación muestra un recuadro solicitando un nombre para la sala que se va a crear
  6. El usuario introduce el nombre deseado y pulsa el botón aceptar
  7. La aplicación genera un identificador único para la sala
  8. La aplicación crea la sala en el servidor de mensajería
  9. La aplicación se une a la sala recién creada
  10. La aplicación añade todos los contactos seleccionados a la lista de propietarios de la sala
  11. La aplicación envía un invitación a cada contacto añadido a la sala

12. La aplicación añade la sala a la persistencia de nuestro sistema
13. La aplicación redirige al usuario al listado de chats donde puede acceder al chat multiusuario acabado de crear

- Diagrama de secuencia: En el esquema de la figura-3.10 se muestra el flujo de información cuando el usuario decide iniciar un chat multiusuario. Se puede apreciar que aún siendo el proceso similar al de la creación de un chat convencional, los pasos a seguir y las acciones a realizar son mucho más complejas. De nuevo el patrón Fachada nos vuelve a resultar de gran utilidad.

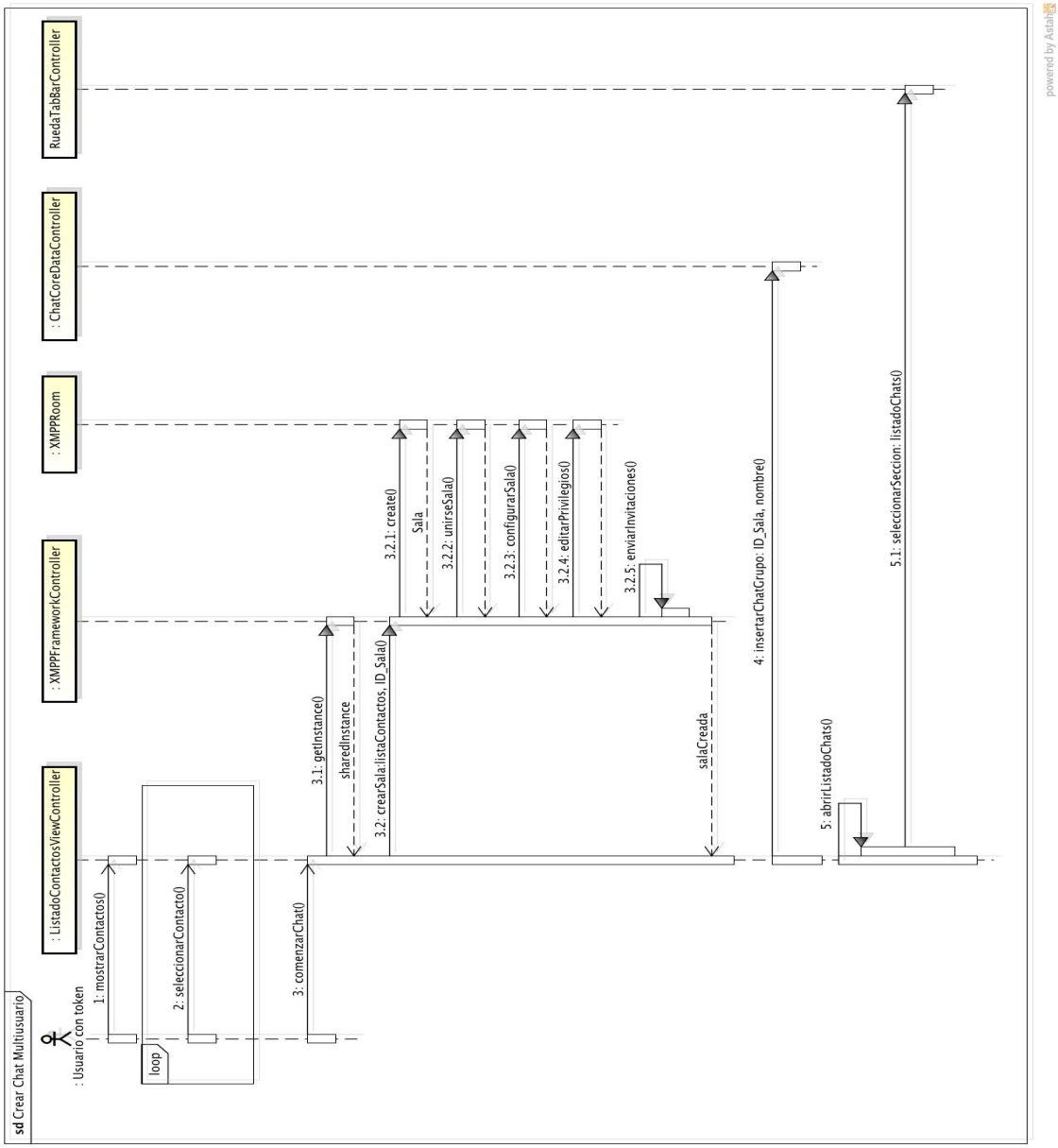

Figura 3.10: Diagrama de secuencia del caso de uso Crear un chat multiusuario

3.4.10. Caso de uso: Eliminar chat convencional

- Condiciones previas: El usuario debe haber obtenido un token OAUTH.
- Descripción detallada: El usuario podrá eliminar de la persistencia y por lo tanto del listado de chats activos un chat convencional en cualquier momento, tenga o no conexión a internet.
- Escenario principal:
  1. El usuario accede al listado de chats activos
  2. La aplicación carga a partir de los datos de persistencia el listado de chats activos actualizado
  3. El usuario realiza el gesto de arrastre sobre el chat convencional
  4. La aplicación entra en modo edición mostrando un botón *Eliminar* sobre la fila seleccionada
  5. El usuario pulsa el botón eliminar
  6. La aplicación se encarga de eliminar el chat del sistema de persistencia y recarga los datos del listado

3.4.11. Caso de uso: Eliminar chat grupo

- Condiciones previas: El usuario debe haber obtenido un token OAUTH, disponer de conexión a internet activa y tener como mínimo un chat convencional activo.
- Descripción detallada: El usuario podrá eliminar un chat de grupo del listado de chats activos, hecho que provocará la salida de dicha sala de conversación multiusuario, dejando de recibir mensajes y notificaciones.
- Escenario principal:
  1. El usuario accede al listado de chats activos
  2. La aplicación carga a partir de los datos de persistencia el listado de chats activos actualizado
  3. El usuario realiza el gesto de arrastre sobre el chat multiusuario que desea eliminar
  4. La aplicación entra en modo edición mostrando un botón *Eliminar* sobre la fila seleccionada
  5. El usuario pulsa el botón eliminar
  6. La aplicación consulta el listado de propietarios de la sala
     a) Si somos el único propietario
        1) La aplicación abandona la sala de chat multiusuario e indica al servidor que debe ser destruida
     b) si existe más de un propietarios
        1) La aplicación elimina al usuario del listado de propietarios
        2) La aplicación abandona la sala de chat multiusuario
  7. La aplicación se encarga de eliminar el chat del sistema de persistencia y recarga los datos del listado
- Diagrama de secuencia: A continuación se puede apreciar en la figura-3.11 que el proceso de eliminar un chat multiusuario implica una gestión mucho más compleja que la de eliminar un chat simple ya que debe existir comunicación activa con el servidor.

3.4.12. Caso de uso: Enviar un mensaje

- Condiciones previas: El usuario debe haber obtenido un token OAUTH, disponer de conexión a internet activa y tener como mínimo un chat activo de cualquier tipo.
- Descripción detallada: Al entrar en la pantalla de listado de mensajes de un chat convencional o multiusuario el usuario podrá redactar y enviar un mensaje al destinatario o destinatarios.
- Escenario principal:
  1. El usuario accede al listado de mensajes de un chat activo
  2. La aplicación carga a partir de los datos de persistencia el listado de mensajes del chat activo actualizado
  3. La aplicación resetea el contador de mensajes sin leer
  4. El usuario redacta un mensaje y pulsa el botón enviar
  5. La aplicación recoge el mensaje redactado y lo envía al servidor
  6. La aplicación guarda en persistencia los datos del mensaje
  7. La aplicación actualiza el listado de mensajes del chat activo

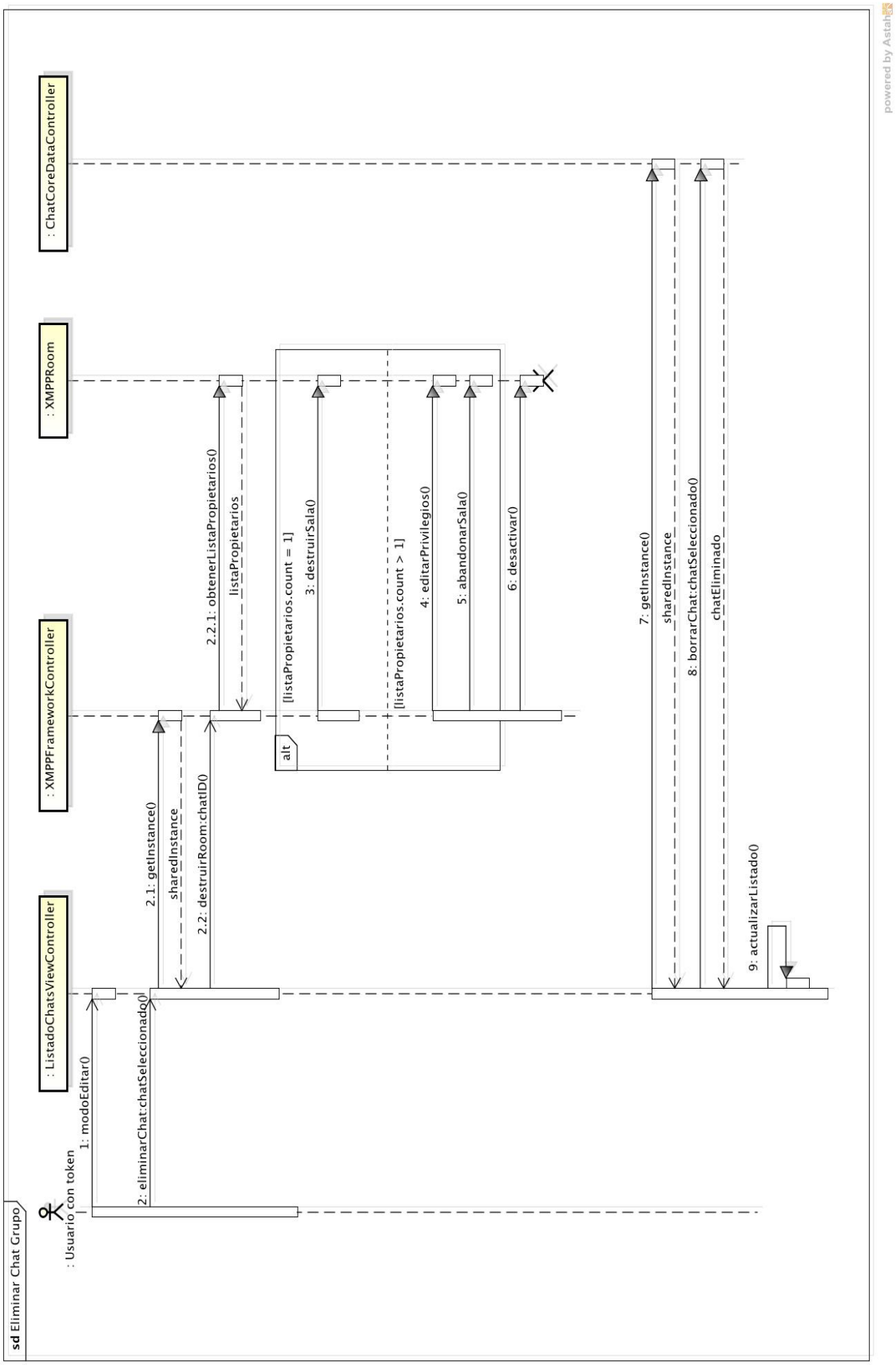

Figura 3.11: Diagrama de secuencia del caso de uso Eliminar un chat multiusuario

3.5. Modelo conceptual

El modelo conceptual es un artefacto generado durante la fase de diseño de un proyecto que intenta representar las entidades del mundo real que van a aparecer en nuestro proyecto y que acabarán formando parte de nuestro modelo de negocio. Para construirlos se suele utilizar el lenguaje *UML*[4]. En la figura-3.12 podemos ver el esquema realizado para la primera versión de la aplicación, nosotros partiremos de esa base, la modificaremos y la ampliaremos según las necesidades de las nuevas funcionalidades.

Podemos ver la representación gráfica del nuevo modelo conceptual en la figura-3.13, se ha decidido eliminar la información almacenada por cada clase ya que a continuación vamos a explicar cada una de las modificaciones que se han realizado sobre el modelo antiguo y detallaremos los datos que incluye cada una de ellas, en caso de no mencionarse lo contrario se mantienen los datos presentes en la figura-3.12.

3.5.1. Modificaciones del modelo anterior

La primera modificación que hemos realizado ha sido la eliminación de la clase Correo esto se debe a que la funcionalidad de acceso al correo electrónico se elimina de la aplicación por los motivos siguientes:

- El prácticamente nulo índice de uso de esta funcionalidad
- La no adaptación al protocolo OAUTH de esta funcionalidad

Por lo tanto y en consenso con el director y el co-director del proyecto se decidió suprimir por completo la sección de correo en la aplicación hecho que implicará la desaparición de los elementos que representan el correo tanto en los esquemas como en los modelos.

La otra modificación sobre el modelo de la versión 1.0 la provoca la inclusión del protocolo OAUTH que nos permitirá no tener que almacenar en nuestro sistema el binomio usuario-password, sino que lo podremos sustituir directamente por el token de acceso.

[4]Unified Modeling Language, lenguaje para el modelado de sistemas software

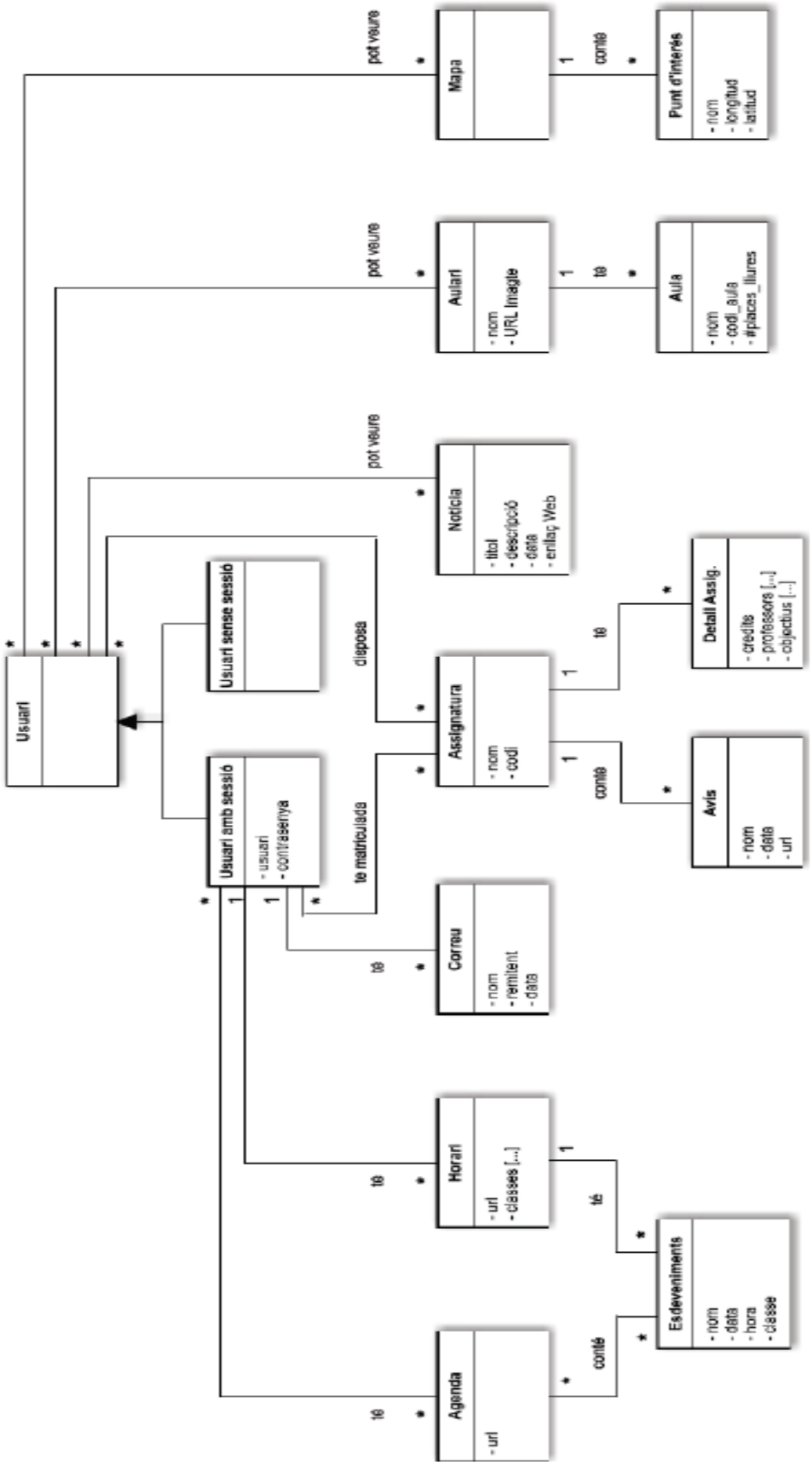

Figura 3.12: Modelo conceptual del Racó Mobile 1.0 [10]

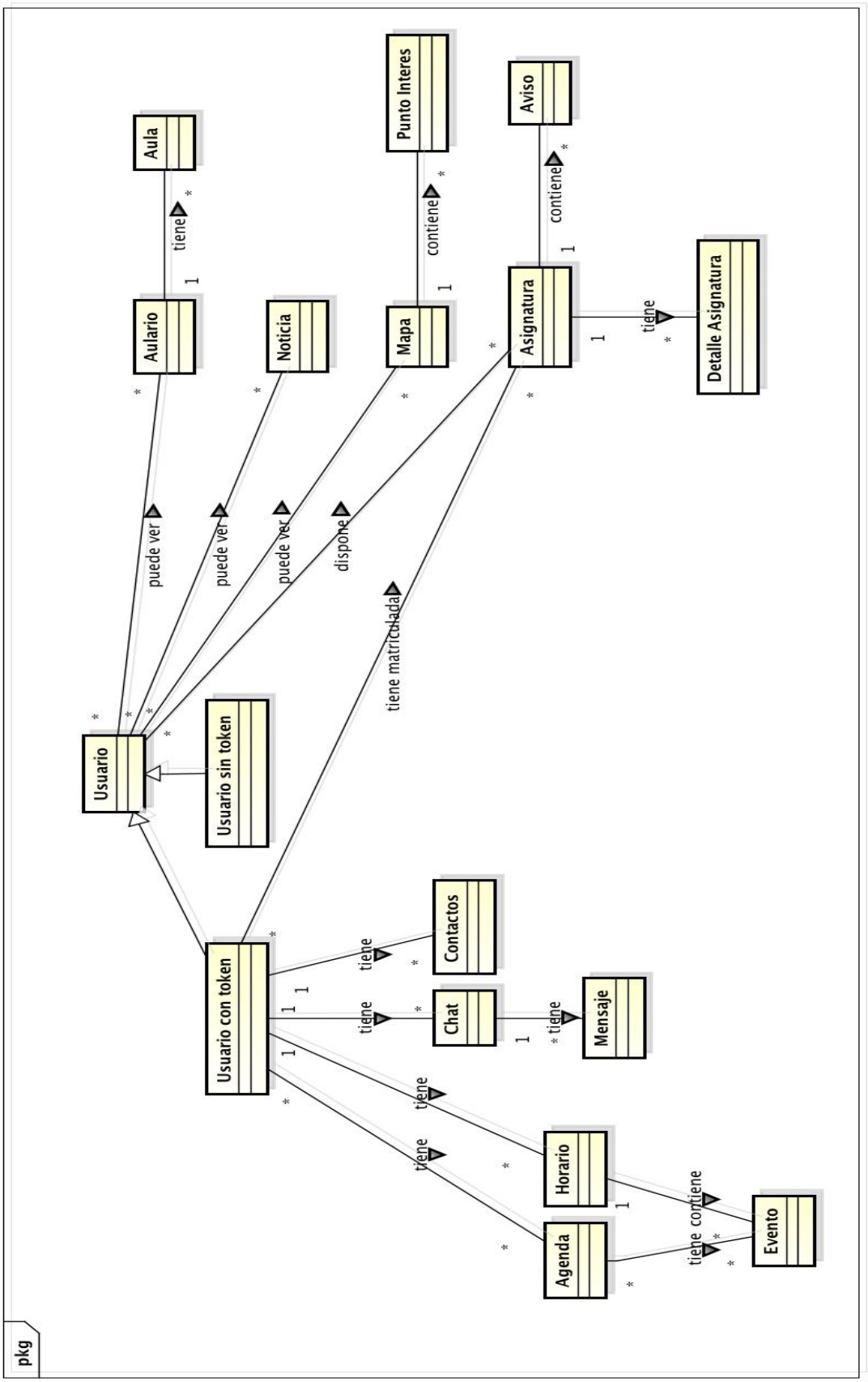

Figura 3.13: Modelo conceptual del Racó Mobile 2.0

3.5.2. Chat y mensajes

Con la introducción del sistema de mensajería aparecen nuevos datos que el sistema deberá tener en cuenta. En primer lugar sabemos que será una de las funciones privadas por lo que únicamente tendrán acceso al chat aquellos usuarios que hayan conseguido un token OAUTH. Si continuamos analizando las necesidades del sistema de mensajería pronto nos daremos cuenta que los mensajes los debemos almacenar ya que se trata de un sistema de chat persistente y como todo mensaje sucede en un contexto determinado también debemos guardar dicho contexto, así nacen las clases *Chat* y *Mensaje* en nuestro modelo conceptual.

Los datos que debe tratar cada clase son los siguientes:

- Clase Chat
  - esGrupo: Indicará si el chat es convencional o multiusuario
  - fechaActualizacion: Indica la fecha de la última actualización del chat
  - jid: Es el identificador del chat
  - noLeidos: Cantidad de mensajes no leídos
  - nombre: Indica el nombre a mostrar como identificador del chat en el listado
  - ultimoMensaje: Cuerpo del último mensaje enviado/recibido
- Clase Mensaje
  - cuerpo: Texto del mensaje
  - esMio: Indica si el mensaje es nuestro o de otro usuario
  - jidMsg: Indica el identificador del remitente del mensaje
  - remitente: Indica el nombre del remitente del mensaje
  - timestampLocal: Indica la fecha en que se ha recibido el mensaje
  - timestampRemoto: Indica la fecha en que se ha enviado el mensaje
  - tipo: Tipo de mensaje

Estos son los datos que la aplicación guardará mediante Core Data.

3.5.3. Contactos

Al igual que en el caso de las clases Chat y mensajes la clase Contactos también aparece como consecuencia de la existencia del sistema de mensajería ya que necesitaremos poder acceder a los usuarios que hemos añadido a nuestra lista para poder iniciar sesiones de chat.
En este caso los datos que debe tratar esta clase son los siguientes:

- Clase Chat
  - nombre: Indicará el nombre a mostrar del contacto
  - jid: Indica el identificador del contacto
  - suscripción: Indica el tipo de relación que tenemos el contacto

Con esta información tenemos suficiente para gestionar nuestro listado de contactos que igual que los mensajes será almacenada utilizando Core Data.

3.6. Vistas

No debemos olvidar que estamos trabajando aplicando el patrón MVC y por lo tanto debemos estudiar y definir el Modelo, las vistas y los controladores. En la sección anterior hemos descrito el modelo, a continuación pasaremos a diseñar las vistas.

En el sistema iOS las vistas se pueden definir utilizando ficheros .xib que se construyen mediante la utilización del *Interface Builder* que viene integrado en el Xcode o bien las podemos crear mediante código. En este proyecto se han utilizado ambos métodos para conseguir crear vistas lo más concretas posibles para adaptarse a las necesidades específicas de los nuevos componentes de la aplicación. En la figura-3.14 se puede ver un esquema de las nuevas vistas, las antiguas no se han incluido en el esquema por temas de espacio y simplificación, únicamente debería matizar que no se han visto alteradas y han mantenido su aspecto original.

Tanto si hemos creado las vistas con un método o con el otro los componentes disponibles para generarlas son los mismos. Tras un estudio exhaustivo de las características de las vistas de nuestro proyecto pudimos concluir que necesitaríamos los siguientes elementos:

- UILabel: Muestra texto no editable en pantalla
- UIButton: Muestra un botón en pantalla
- UIWebView: Muestra una ventana en pantalla donde se puede cargar contenido web
- UIImageView: Muestra una imagen en pantalla
- UITableView: Muestra un listado en pantalla
- UITableViewCell: Representa una celda de un listado
- UITextField: Muestra un recuadro de texto editable
- UIBarButtonItem: Muestra botones en la barra superior de navegación

En la sección 4 se explica con detalle como se han personalizado algunos de estos elementos para poder conseguir efectos visuales más avanzados, aunque en la mayor parte de los casos ha sido suficiente con utilizar los elementos básicos ajustando sus características a la apariencia de nuestra aplicación.

También comentar que igual que sucede con el modelo, en el caso de las vistas también se han eliminado aquellas relacionadas con la recepción y envío de correos.

3.7. Controladores

Los controladores se podría decir que son el corazón del sistema ya que ellos son los que organizan y mueven todo el flujo tanto de datos como de eventos y vistas. Sin los controladores sería imposible enlazar las vistas con el modelo de un modo limpio y eficaz, por lo tanto debemos diseñar su distribución con meticulosidad pues si esta parte falla la aplicación se volverá inestable y poco fiable.

En iOS podemos encontrar dos tipos de controladores: los controladores asociados a vistas y el resto. El por qué de esta diferenciación vuelve a ser Cocoa touch, pues prácticamente estamos obligados a utilizar un controlador por cada vista que tengamos, esto no es que sea algo negativo más bien al contrario ya que nos permitirá tener un control minucioso de cada una de nuestras vistas para poder actuar con la mayor precisión posible.

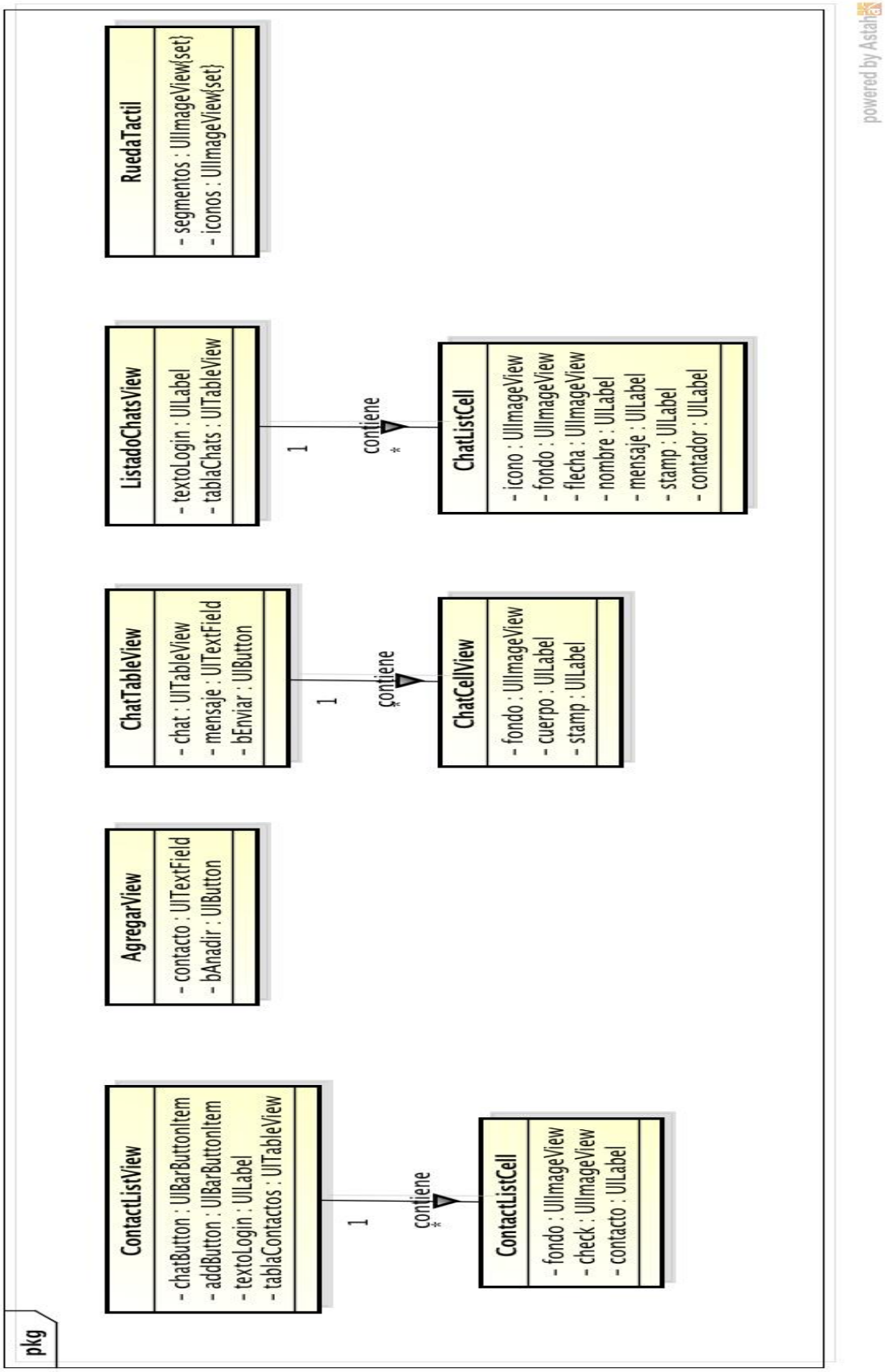

Figura 3.14: Esquema de las vistas del Racó Mobile 2.0

Así pues como se observa en la figura-3.15 tenemos un controlador por cada vista que se ha añadido al sistema a excepción de las vistas utilizadas dentro de otras como es el caso de las *UITableViewCell*[5] o de la Rueda Táctil. También se puede ver que tenemos un controlador encargado del conjunto de las vistas que sustituye al antiguo *UITabBarController*[6] que era el encargado de gestionar la barra de secciones y las vistas asociadas a cada sección en la primera versión de la aplicación, este nuevo controlador incluye la gestión de la rueda táctil y de las secciones de la versión actual.

Por otro lado observamos el controlador encargado de gestionar las conexiones con el servidor de mensajería, el XMPPFrameworkController y el ChatCoreDataController que es el responsable de tratar la persistencia relacionada con el chat. Como nota adicional debo decir que en el diagrama no se muestran todos los controladores nuevos añadidos, ya que muchos de ellos pertenecen a las librerías y frameworks utilizados para las conexiones OAUTH y las conexiones al servidor Openfire.

Igual que en los apartados 3.6 y 3.5 anteriores debemos indicar que los controladores relacionados con el correo han sido eliminados.

[5] Clase que representa una fila en un listado
[6] Clase que agrupa varias vistas y las hace accesibles desde una barra con icono

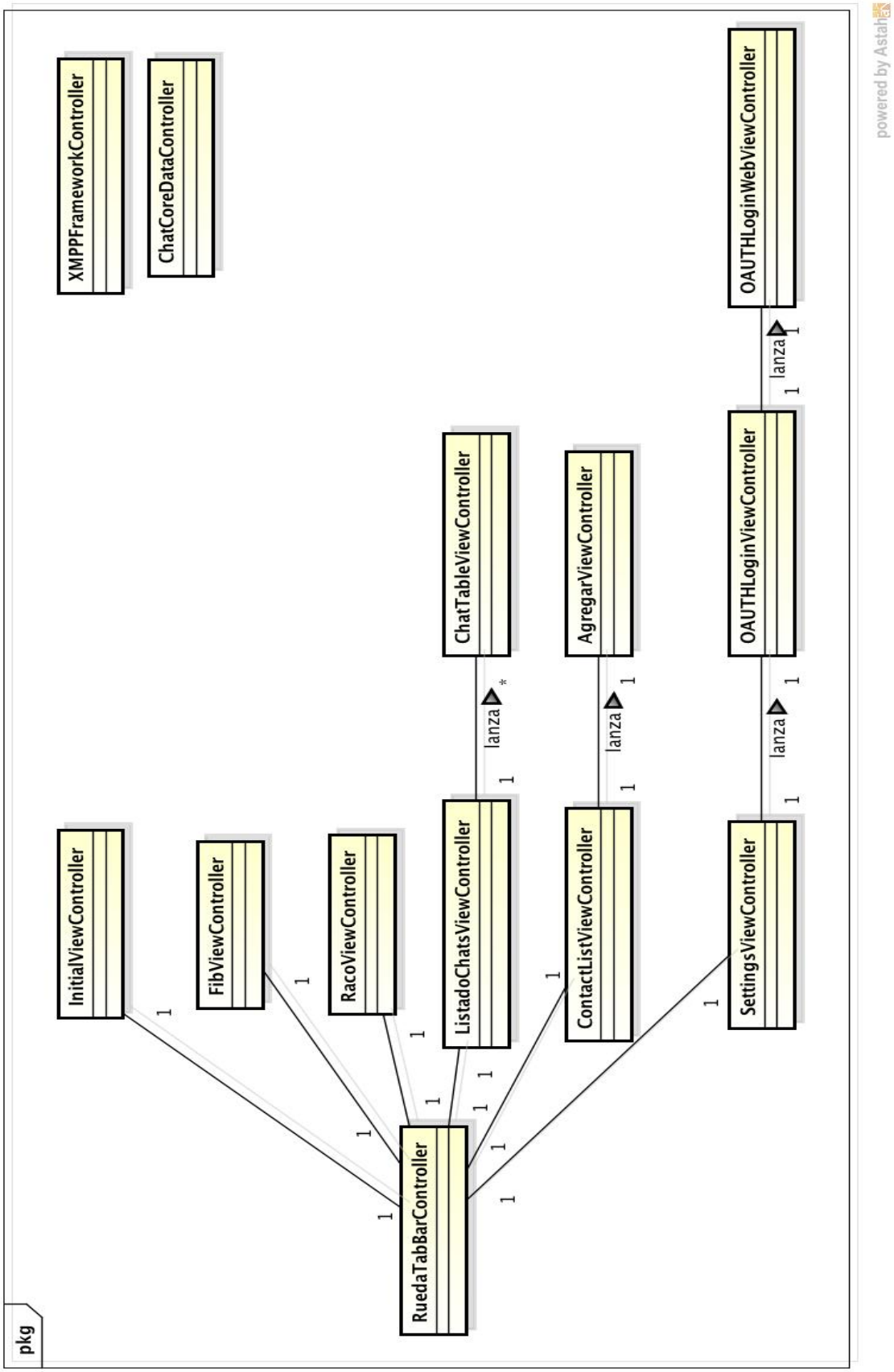

Figura 3.15: Esquema de los controladores del Racó Mobile 2.0

3.8. Planificación

Una de las partes más importantes en un proyecto es la organización temporal de las tareas que se deben realizar para llevarlo a cabo. Sin una buena estructuración es imposible que se pueda realizar el proyecto en el tiempo estipulado ya que incluso habiendo realizado una planificación extremadamente meticulosa y precisa es muy probable que durante el desarrollo surjan contratiempos de cualquier tipo ya sean personales o técnicos.

Es por eso que debemos intentar realizar un buen trabajo organizativo para poder realizar las distintas tareas a tiempo, además una buena planificación también nos servirá para ver el estado del proyecto en cada momento y la cantidad de trabajo que falta por realizar.

Para realizar esta tarea debemos tener presente el alcance de nuestro proyecto, una vez sabemos exactamente qué debemos hacer y el tiempo que disponemos para ello pasaremos a clasificar las tareas según su importancia dentro del proyecto, la carga de trabajo que suponga y su dependencia respecto a otras tareas. Hecho esto deberíamos ser capaces de realizar una organización preliminar del trabajo, para ello nos será muy útil realizar un diagrama de Gantt.

Cuando hayamos terminado procederemos con una segunda reorganización de las tareas, esta vez intentando tener en cuenta fechas que sepamos de antemano que nuestra productividad no será la esperada por uno u otro motivo. Con esto conseguiremos una buena estructura de trabajo que deberemos intentar seguir durante todo el proyecto.

Es importante que si en algún momento aparecen problemas que conlleven un retraso considerable nos replanteemos las tareas pendientes y volvamos a reorganizar el trabajo. El resultado final de este proceso aplicado a nuestro proyecto lo podemos ver a gráficamente en el diagrama de Gantt de la figura-3.16. Para hacerse una idea concreta de las horas realizadas se debe tener en cuenta que durante la realización del proyecto se he estado trabajando 9 horas diarias y por lo tanto la media de horas durante los días laborables representados en el Gantt debe tomarse por unas 4 horas, en cambio los fines de semana deberían contarse aproximadamente por unas 8 horas.

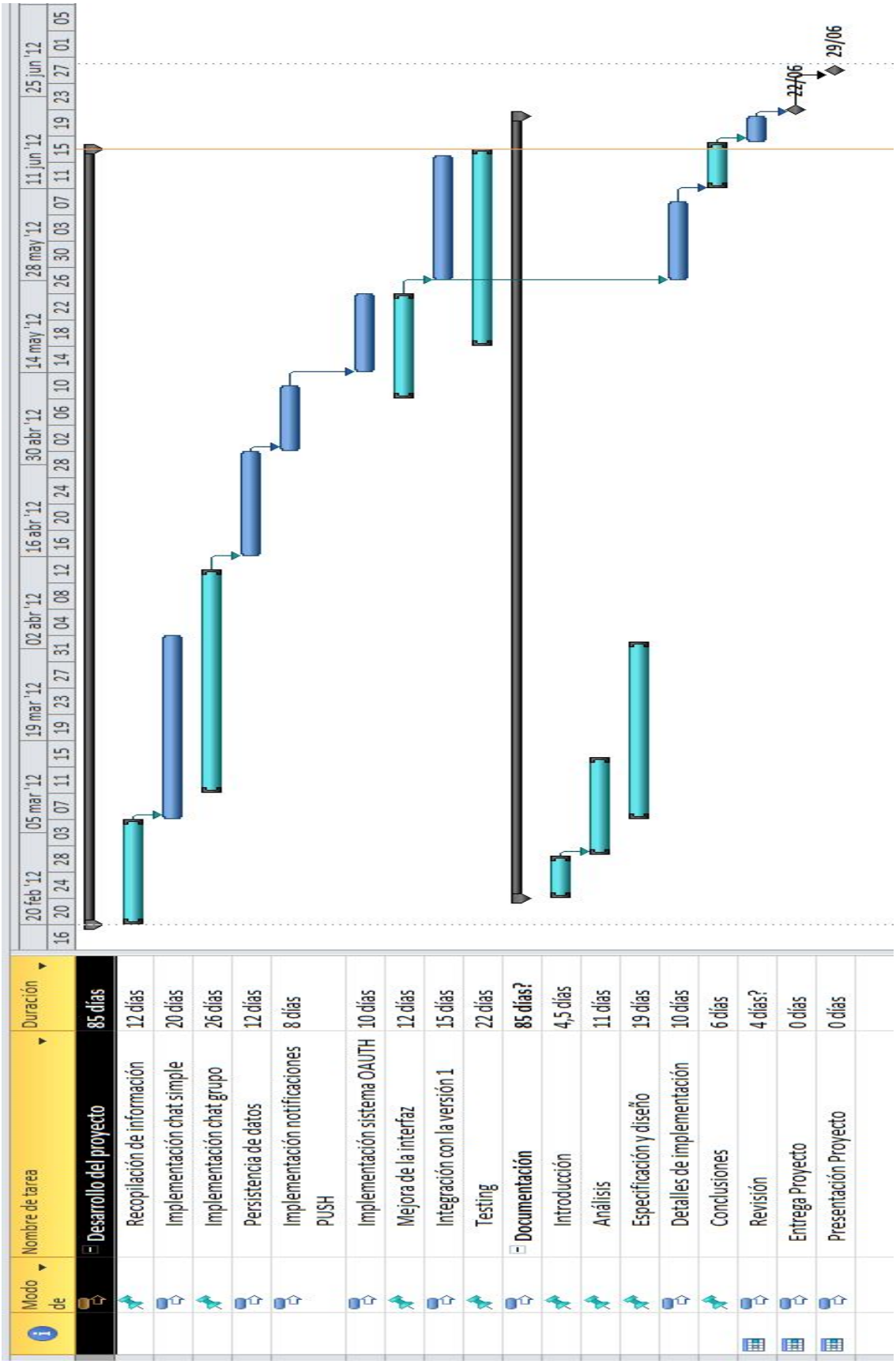

Figura 3.16: Planificación de tareas: Diagrama de Gantt del proyecto

Capítulo 4

Detalles de implementación

Durante el proceso de implementación del Racó Mobile tal y como hemos detallado en el capítulo se han construido varios elementos vitales para la consecución de los objetivos planteados al inicio del proyecto, la mayor parte de los elementos son bastante complejos y por eso intentaremos desgranar el proceso de construcción e implementación de las partes más importantes de nuestra aplicación.

4.1. Notificaciones

Las notificaciones permiten la comunicación de eventos, locales o remotos ,que están relacionados directamente con nuestra aplicación. Racó Mobile utiliza principalmente las notificaciones remotas ya que los cambios que provocan las notificaciones suceden directamente en el servidor de mensajería.

Pasaremos a realizar una breve introducción al sistema de notificaciones que nos proporciona Apple antes de profundizar en nuestra implementación.

Como ya hemos comentado las notificaciones se dividen en dos grupos según dónde se generan:

- Notificaciones locales: Son aquellas que programa directamente la aplicación y que son lanzadas internamente por el sistema operativo
- Notificaciones push: También conocidas como notificaciones remotas son aquellas enviadas por el servidor de la aplicación, pero no se envían directamente a nuestros terminales sino que en primera instancia se mandan al APNS que será el encargado de sincronizarse con nuestro teléfono y mandarnos la notificación siempre y cuando tengamos la aplicación instalada.

Como podemos observar son dos procedimientos distintos aunque el resultado final de cara al usuario es el mismo; se recibirá un mensaje de aviso siempre y cuando la aplicación esté cerrada o funcionando en segundo plano, ya que de otro modo se presupone que la aplicación recibirá directamente el mensaje que el servidor le quiere comunicar y lo mostrará por pantalla siendo innecesario realizar ningún tipo de advertencia. Tal y como podemos ver en la figura-4.1 las notificaciones se pueden presentar de dos formas según la configuración del usuario, su función será la misma, dar un aviso de texto y dos posibles acciones a realizar una de ellas lanzará la aplicación y la otra simplemente cerrará la notificación.

Las notificaciones no están pensadas para mandar mensajes muy largos, de hecho se puede enviar tres tipos de información (en una misma notificación):

Figura 4.1: Tipos de notificaciones Push según la configuración

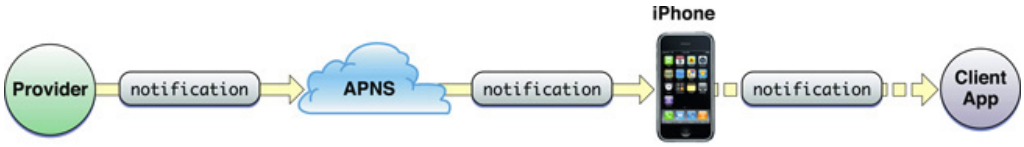

Figura 4.2: Flujo de datos entre el servidor de la aplicación, el servidor de Apple y nuestra aplicación [3]

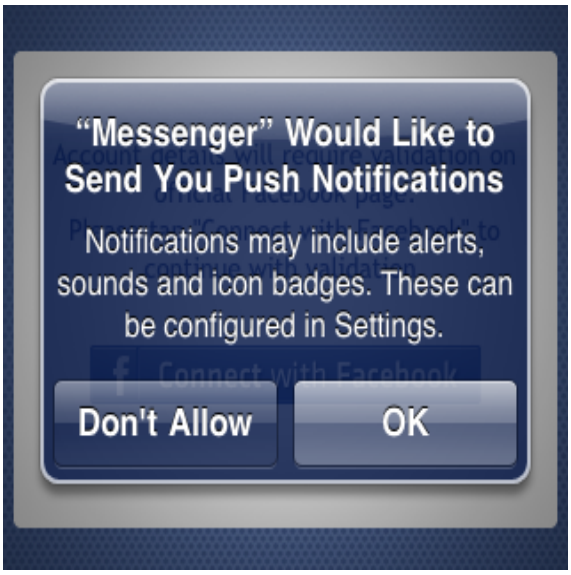

Figura 4.3: Mensaje emergente de permisos para notificaciones

- Texto: Es la información que se mostrará al usuario una vez se presente la notificación, además también se puede mandar el texto del botón de acción (el que abre la aplicación).
- Icono: Es el número que aparecerá en el icono de nuestra aplicación. No se puede incrementar ni decrementar el número que actualmente aparezca en el icono así que si se quiere mostrar un número acorde a la cantidad de notificaciones recibidas se deberá llevar la cuenta en el lado del servidor.
- Sonido: Es el sonido a reproducir en el momento de recibir la notificación. Este sonido se buscará dentro de la aplicación y en caso de no encontrarse se reproducirá el sonido de alertas que el usuario haya configurado por defecto.

Durante el desarrollo de este servicio deberemos tener en cuenta que el envío de notificaciones remotas no tiene garantías de entrega, hecho que implica que los avisos puedan llegar en un orden distinto en el que fueron enviados o incluso no llegar nunca, por ello la información que descargue nuestra aplicación nunca debería presuponer que nos ha llegado el aviso correspondiente.

También deberemos tener en cuenta el flujo de información entre nuestro dispositivo, el servidor Apple y el servidor de nuestra aplicación. Tal y como se muestra en la figura-4.2 el flujo transcurre del siguiente modo:

La primera vez que abrimos la aplicación, nos aparecerá un mensaje similar al de la figura-4.3 para permitir o denegar la utilización de notificaciones en dicha aplicación.

Suponiendo que el usuario acepta recibir notificaciones remotas la aplicación procederá a conectarse con el servidor APNs PIE, la conexión inicialmente realizará la validación del dispositivo

Figura 4.4: Validación de los certificados dispositivo-servidor [3]

mediante autenticación TLS donde tanto el servidor como el dispositivo se intercambias y validan los respectivos certificados tal y como vemos en la figura-4.4. Este tipo de autenticación también tendrá lugar entre el servidor de la aplicación y el servidor APNs.

Tras establecer la conexión, el dispositivo procederá a registrarse con el servidor APNs. La aplicación enviará una petición de registro indicando entre otras cosas el tipo de notificaciones que desea recibir (texto/audio/icono), el servidor APNs procesará la petición y generará un Token que devuelve a la aplicación quien se encargará de hacerla llegar hasta el servidor de la aplicación tal y como ilustra la figura-4.5. A partir de este momento cada vez que el dispositivo se conecte con el servidor APNs le mandará el Token como identificación.

Finalmente, cada vez que el servidor de la aplicación necesite mandar una notificación a un dispositivo concreto se conectará con el servidor APNs y le mandará tanto el contenido de la notificación como el Token del dispositivo al que se la quiere mandar. A partir de ahí el servidor APNs se la hará llegar al dispositivo tan pronto como éste conecte con él.

Después esta explicación del funcionamiento del sistema de notificaciones explicaremos como lo utilizamos en Racó Mobile.

La idea básica consiste en mandar una notificación siempre que un usuario reciba un mensaje, tanto si es un mensaje directo como si se trata de un mensaje grupal. La notificación será del tipo texto y audio prescindiendo del indicador en el icono ya que no se ha implementado ningún componente adicional dedicado a rastrear la cantidad de notificaciones pendientes por leer por parte de la aplicación, aunque esto se podría añadir en un futuro sin la necesidad de realizar demasiados cambios.

En nuestro caso el encargado de enviar notificaciones a los dispositivos será un plugin instalado en el servidor de mensajería. Este plugin lo hemos diseñado utilizando tres tipos de API: la del servidor Openfire, la del Racó y la de Java PNS. El funcionamiento es simple, se capturan los mensajes que salen del servidor, entonces utilizando la primera API se descompone el mensaje para obtener el remitente y el destinatario, a continuación con esa información accedemos, mediante la API del Racó, al webservice encargado de almacenar los Tokens de cada usuario, una vez

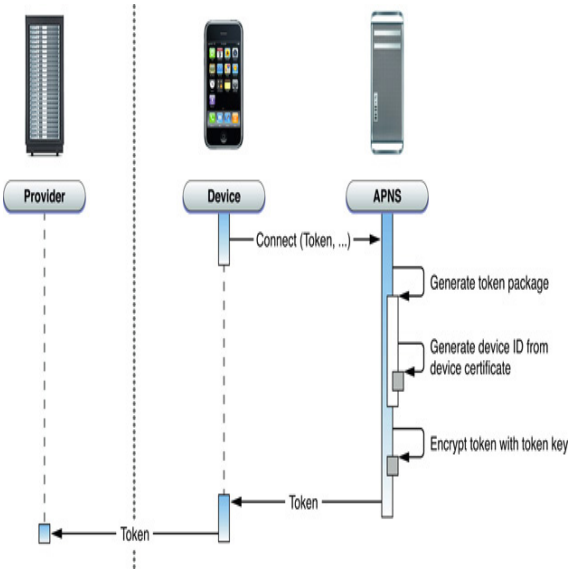

Figura 4.5: Obtención y envío del token identificador del dispositivo [3]

obtenida respuesta se monta el contenido de la notificación Remitente: Texto del mensaje que se lo pasaremos a la API JavaPNS [2] que se encargará de realizar la comunicación con el servidor APNs al que se le trasmitirá el Token del usuario destino, la notificación y el certificado de nuestro servidor, con ello finaliza el proceso. Podemos ver un ejemplo gráfico este proceso en la figura-4.6

En el caso de las conversaciones usuario a usuario el flujo de acción del plugin se desarrolla como acabamos de comentar, sin embargo para las conversaciones multiusuario realizaremos un poco de trabajo extra, en primer lugar a partir del destinatario deberemos conseguir la sala de conversa multiusuario a la cual va dirigido el mensaje, de ahi obtendremos la lista de propietarios de la sala (todos nuestros usuarios serán propietarios una sala en la que participan) y mandaremos una notificación con el contenido del mensaje y el nombre del remitente a cada uno de los usuarios que no sean el remitente.

Con esto ya tenemos el sistema de notificaciones activo y funcionando tal como queremos.

4.2. Elementos de navegación personalizados

Probablemente uno de los elementos que hacen la interfaz más moderna y elegante es la rueda táctil que en esta nueva versión del Racó Mobile ha pasado a sustituir la típica barra de navegación inferior como podemos ver en la comparativa de la figura-4.7. Este nuevo widget no es un uno de los elementos básicos que nos ofrece el SDK para el desarrollo de interfaces y por lo tanto debemos construirlo con técnicas algo más complejas.

En primer lugar dividiremos el problema en dos bloques independientes: la rueda táctil y el gestor de navegación.

4.2.1. La rueda táctil

La idea principal tras este componente radica en construir una rueda giratoria dividida en tantas secciones como tenga la aplicación, que en nuestro caso son 6. Si repasamos los widgets básicos que nos ofrece Cocoa touch no encontraremos ninguno que a priori nos sirva para realizar

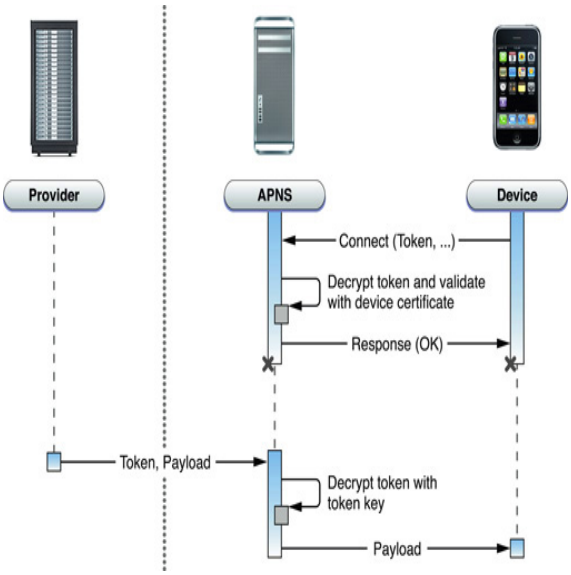

Figura 4.6: Flujo completo de envío y recepción de una notificación Push [3]

Figura 4.7: Comparativa de la barra de la versión 1.0 (izq.) y 2.0 (der.)

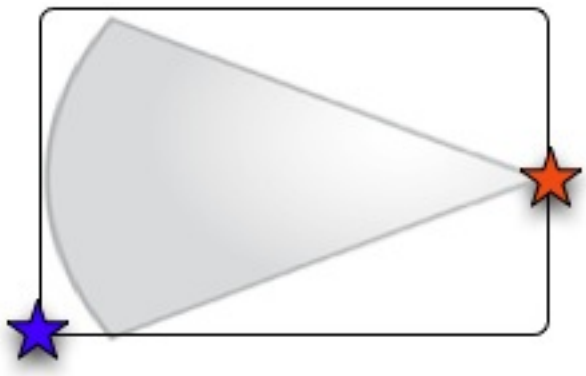

Figura 4.8: Punto de punto de origen (azul) y rotación (rojo)

nuestro cometido, sin embargo la clave reside en dividir el problema en elementos más simples.

Podemos observar que un círculo dividido en 6 partes iguales se puede conseguir o bien partiendo el círculo en 6 o bien juntando seis partes iguales alrededor de un punto central. Como queremos tener control sobre los distintos elementos y además realizar un trabajo simple y limpio optaremos por la segunda opción.

Para representar cada uno de los segmentos utilizaremos una imagen construida para representar la sexta parte de un círculo, a diferencia de lo que se podría pensar en primer momento no vamos a utilizar seis imágenes distintas dibujadas en distintos ángulos, utilizaremos las transformaciones afines para conseguir el mismo resultado. Para ello necesitamos la librería gráfica QuartzCore incluida en Cocoa touch, con ella tendremos acceso a las llamadas **CALayers** que son los componentes sobre los cuales trabajan todos los elementos visuales en iOS y que dicho de un modo simple representan el rectángulo donde se dibujan los elementos en cuestión. La ventaja de poder acceder a esta capa es que tiene múltiples propiedades que nos resultarán útiles para construir nuestra rueda táctil como veremos a continuación.

Lo que vamos a hacer en primera instancia es generar las seis imágenes que representan nuestro círculo y a cada una de ellas le aplicaremos una transformación afín distinta para conseguir distribuirlas correctamente en formación circular. Como las transformaciones afines funcionan en radianes deberemos rotar cada imagen un múltiplo de 2PI / 6 rad (una sexta parte de 360 º), hay que recordar que como queremos el primer segmento colocado perpendicularmente respecto a la horizontal deberemos comenzar a colocar los segmentos a partir de PI/2 rad (90º).

Hay un pequeño detalle a tener en cuenta y es que por defecto la rotación de una imagen se realiza desde el punto central de la capa que contiene la vista, por eso para hacer más sencilla la colocación de cada segmento las CALayers nos permiten mover el punto de rotación según nos convenga, así que lo moveremos al extremo derecho centrado de la imagen como indica la figura-4.8.

Conforme vayamos colocando los segmentos añadiremos a cada uno de ellos la imagen correspondiente a la sección que representan, los iconos se deberán que rotar -PI/2(90º a la izquierda) ya que se insertan dentro de los segmentos siguiendo el sistema de referencia de cada imagen (tienen la posición 0,0 en la esquina inferior izquierda, figura-4.8) y sin olvidar que el primer segmento debe contener el icono en estado activo (marcado de color azul).

Con esto ya tenemos construido el círculo con los iconos y segmentos de cada sección, ahora

vamos a añadirle la rotación. Como estamos trabajando con un conjunto de *UIImageViews*[1] y éstas son subclases de *UIView*[2] podemos añadir las imágenes que hemos generado en un contenedor que las englobe a todas ellas ya que lo que haremos será rotar el conjunto de los segmentos en lugar de cada segmento por separado.

Para poder gestionar el movimiento de rotación debemos calcular el ángulo que dibujará el usuario al tocar la rueda y aplicar esa rotación al conjunto de segmentos, para ello nos bastará con capturar dos tipos de evento: el inicio del toque y el arrastre del mismo.

La secuencia es sencilla, al recibir un evento de inicio de toque sobre la rueda obtendremos el punto inicial y calcularemos el ángulo que forma con la horizontal de la rueda, a partir de este momento cada vez que se arrastra el dedo por la rueda se captura un evento de arrastre, cada vez que se reciba este tipo de evento volveremos a obtener el punto de la rueda donde se haya producido el arrastre y calcularemos de nuevo el ángulo que forma con la horizontal, con ello podremos calcular la diferencia entre el ángulo del toque inicial y el del toque de arrastre, esa diferencia será la rotación que aplicaremos al conjunto, si hacemos esto cada vez que se produce un arrastre conseguiremos una rotación fluida que seguirá el gesto realizado con el dedo sobre la superficie de la rueda.

Con esto ya tenemos el funcionamiento básico de la rueda implementado, sin embargo necesitamos poder saber cual es el segmento activo una vez el usuario deja de arrastrar la rueda y levanta el dedo de la pantalla, en ese momento calcularemos en que segmento nos encontramos (que será equivalente a una sección única de nuestra aplicación) y aprovecharemos para centrar la rueda en dicho segmento de manera que quede perpendicular a la pantalla, además el icono de dicho segmento se sustituirá por el de color azul para indicar que la sección está activa.

Para conseguir este efecto una vez se levante el dedo podremos calcular la rotación total aplicada al conjunto de segmentos accediendo a los datos de la transformación afín que se está aplicando en ese momento. Una vez tengamos el ángulo de rotación necesitamos saber a qué segmento equivale ese ángulo, para ello prepararemos una estructura de datos que contenga los valores de los ángulos entre los que se encuentra cada segmento así como el punto intermedio que nos servirá para centrar la rueda. así cada vez que se produzca una secuencia de toque - arrastre - levantamiento buscaremos en que sección pertenece el ángulo de la rotación actual, calcularemos la diferencia entre la rotación actual y el punto central del segmento actual y rotaremos esa cantidad de radianes consiguiendo el efecto que buscábamos.

Para finalizar únicamente debemos hacer que cada vez que se cambie el segmento activo se dispare un evento que avise de este cambio y permita que cambiemos de pantalla, para ello utilizaremos el gestor de navegación explicado en la siguiente sección.

4.2.2. El gestor de navegación

Apple pone a nuestra disposición varios elementos que nos permiten generar y controlar la navegación entre las distintas pantallas de nuestras aplicaciones. Estos elementos son los llamados *View Controllers*[3] y existen varios tipos que vienen por defecto con el entorno de desarrollo y se ajustan a una amplia variedad de situaciones, por ejemplo entre los más básicos podemos encontrar el *UIViewController*[4] que es el más genérico de todos y que se encarga de mostrar una pantalla simple, también podemos encontrar el *UITableViewController* que se encarga de mostrar y gestionar los datos sobre una tabla; si buscamos algo más complejo tenemos el *UINavigationController* que en este caso se encarga de mantener un conjunto de pantallas que se muestran

[1] Clase que permite representar una imagen
[2] Clase genérica para representar vistas de la cual heredan el resto de componentes visuales de iOS
[3] Controladores de vista
[4] Clase que representa un controlador de vista genérico

de manera ordenada y generalmente en cascada, nos ofrece alguna característica extra como por ejemplo una barra de navegación superior que nos permite saltar entre las distintas vistas.

Finalmente encontramos el UITabBarController que es el que se utilizaba en la primera versión del Racó Mobile, este controlador se basa en una barra inferior con hasta cinco secciones, cada sección corresponde a una pantalla distinta y para navegar entre ellas simplemente debemos pulsar sobre los iconos de la barra.

Nosotros lo que queremos es situar nuestra rueda táctil en la parte inferior de la pantalla y que al ir seleccionando los distintos segmentos de la rueda se nos vaya mostrando la pantalla que cada uno de ellos los tiene asociado. No hay ningún tipo básico de View Controller que nos permita hacer esto, por suerte podemos extender y modificar las funcionalidades de los modelos simples para generar uno que se adapte plenamente a nuestras necesidades.

En nuestro caso partiremos del UITabBarController por tres motivos, el primero es que la primera versión se construyó sobre este tipo de controlador y por lo tanto dispondremos de una base sólida por donde comenzar, el segundo es que ya nos proporciona una estructura que nos resulta útil ya que es capaz de gestionar distintas pantallas mostrando cada vez la que está seleccionada y el tercer motivo es que la posición del elemento de control es la misma que la que necesitamos, la parte inferior de la pantalla.

Lo que haremos en primer lugar es deshacernos de la barra por defecto, para ello simplemente indicaremos a nuestro controlador personalizado que debe mantener la barra oculta, esta es una de las propiedades que se puede asignar a este tipo de controladores. Acto seguido incluiremos nuestra rueda táctil y la situaremos en la parte inferior de la pantalla de modo que se vea únicamente la parte superior de la misma para hacer más cómoda su utilización, además añadiremos un método que nos permita recibir los eventos que lanza la rueda cada vez que cambia de sección lo que nos permitirá controlar la pantalla activa.

Como estamos ampliando las funcionalidades de un UITabBarController necesitamos saber cómo gestiona sus pantallas, en este caso lo que mantiene en todo momento es un *Array*[5] o conjunto con todas y cada una de las pantallas asignadas a las secciones de la barra, pero existe una pega y es que en caso de tener 6 pantallas o más dentro de dicho Array de la primera a la quinta se mostrarán como de costumbre, sin embargo de la sexta en adelante se mostrará un listado con todas ellas para que el usuario pueda escoger, esto se debe a que en la barra original únicamente se tiene espacio para 5 elementos y el resto se juntan en el último que da acceso a este listado, con la rueda táctil no tenemos este inconveniente pero como estamos partiendo de la base de un UITabBarController deberemos realizar algunos cambios para que obtengamos el efecto deseado que no es otro que mostrar una pantalla para cada sección evitando el molesto listado que aparece tras tener seis o más pantallas asociadas a una misma barra.

Para conseguir esto asociaremos un nuevo Array a nuestro controlador personalizado y en el momento de la creación del controlador que es cuando se rellena el Array original lo que haremos será mover todo el contenido a la nueva estructura y haremos creer al controlador que únicamente tiene una pantalla que será la que carguemos en el Array original cada vez que la rueda táctil cambie de sección, podemos ver una representación gráfica de este comportamiento en la figura-4.9.

Con todo esto ya tenemos listo el componente que nos ayudará a navegar de un sitio a otro de nuestra aplicación de un modo elegante y sofisticado.

[5]Estructura de almacenamiento de datos con estructura vectorial o matricial

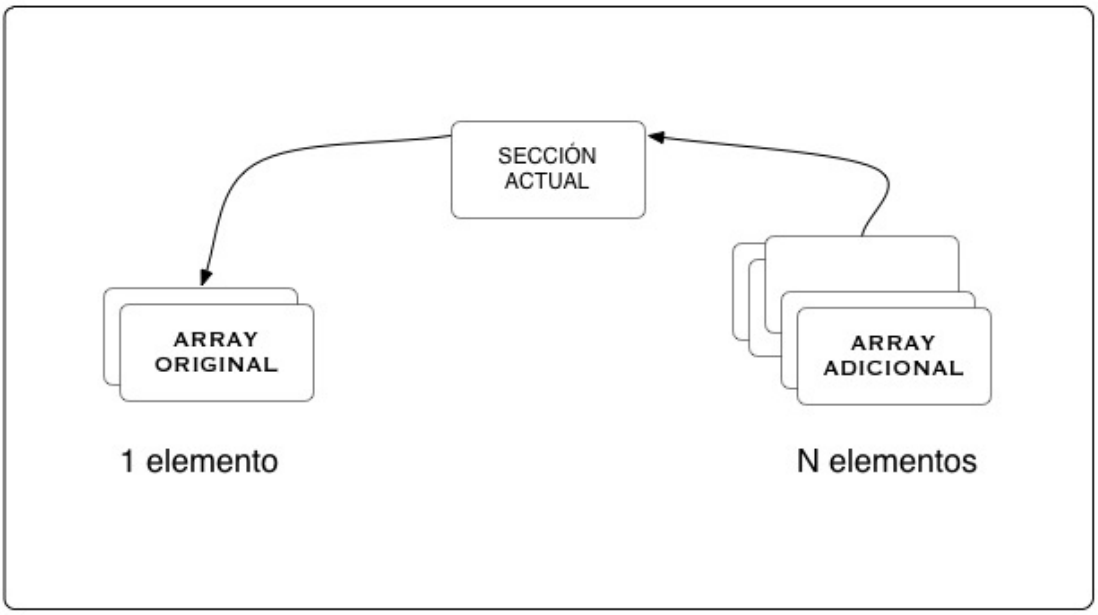

Figura 4.9: Esquema de la utilización de un array adicional

4.3. Chat

Uno de los objetivos principales del proyecto era crear un sistema de chat multiusuario tal y como hemos descrito en el apartado 1.1. Para ello además del cliente y del servidor como se ha explicado con anterioridad necesitaremos un sistema de comunicación que ambos entiendan, el sistema elegido es el XMPP (o Jabber) que está estandarizado desde hace tiempo. Además nuestro cliente deberá ser capaz de transmitirnos la información de una manera visualmente agradable. A continuación explicaremos con algo más de detalle las particularidades del binomio cliente-servidor y de la construcción de la pantalla de chat.

4.3.1. Comunicación iOS - Openfire

Una de las partes más complejas del proyecto ha sido desarrollar el sistema de comunicación entre el dispositivo y Openfire, el servidor de mensajería. Como ya hemos COMENTADO la principal dificultad ha sido que Openfire no implementa el estándar en algunos de sus bloques como por ejemplo la extensión MUC (Multi User Chat) que ha sido particularmente compleja de adaptar y es la que explicaremos más detalladamente.

En Openfire el chat multiusuario se centra en el subsistema de conferencias. Las conferencias tienen como pilar básico clave la creación de salas a las cuales se pueden unir más de dos personas para mantener una conversación, cada sala tiene un identificador único y está regida por distintos tipos de usuario que son los propietarios, los administradores, los moderadores y los ocupantes.

Cada rol tienes una serie de permisos y capacidades que lo distinguen del resto y la idea inicial era que el creador de la sala se convirtiese en propietario único y el resto fuesen ocupantes normales, el problema radica en que Openfire no es capaz de mandar mensajes offline a los integrantes de las salas, esto es consecuencia de que la lista de ocupantes no es permanente sino que se va actualizando en función de si se está conectado o no a la sala, hecho que dificulta tanto el envío de notificaciones Push como la entrega de mensajes multiusuario a un usuario que está desconectado.

Otro problema derivado de esta estructura del servicio de conferencias es que si uno de los ocupantes sale de la sala el chat continúa vivo, pero si el propietario abandona la sala ésta es borrada del sistema y el resto de integrantes se quedan sin ningún método de comunicación. Además de estos inconvenientes debemos añadir el hecho que la creación de una sala cuando uno de los usuario está desconectado suponía la creación de dicha sala sin el usuario que está offline en ese momento ya que las salas funcionan por invitación directa que tampoco son entregadas a usuarios offline.

La solución a todo esto acabó llegando tras darle muchas vueltas al problema y pensar en una organización que permitiese subsanar y rodear todos los problemas presentados por la implementación de Openfire, la solución consistió eliminar las invitaciones directas e ignorar el rol de ocupante asignando a todo usuario de una sala el rol propietario. Voy a explicar esto más a fondo ya que ambos cambios van estrechamente ligados.

En el momento de crear una sala de chat se deciden los contactos con los que se quiere realizar el grupo y un nombre para dicho grupo, se crea la sala en el servidor y en lugar de mandar invitaciones individuales a cada integrante (que de estar desconectados no les llegaría nunca) lo que haremos será modificar la lista de propietarios de la sala añadiendo a todos los usuarios que habíamos seleccionado para incluir en el chat grupal. Debo decir que tanto las listas de propietarios como las de administradores y moderadores a diferencia de las de ocupantes sí son persistentes, utilizaremos la de propietarios porque si un propietario se va de la sala y no hay más propietarios ésta queda destruida y estaríamos en la misma situación que con los ocupantes.

Una vez tenemos en la lista de propietarios a todos los integrantes del grupo simplemente debemos mandarles un mensaje normal (usuario - usuario), que sí se mandan a usuarios offline, añadiendo un campo nuevo a la cabecera que indique el identificador de la sala y como cuerpo del mensaje le pasaremos el nombre que le hemos dado al grupo. Además solucionamos también el problema de abandonar una sala, ahora cuando un usuario salga del grupo lo que hará será editar el listado de propietarios y eliminarse a sí mismo, en caso de ser el único mandará la orden de destruir la sala .

Todo esto no nos permite solucionar el tema de los mensajes online, sin embargo Jabber permite que al conectarse a una sala podemos solicitar los mensajes escritos en una sala desde una fecha, entonces cada vez que nos conectemos a una sala mandaremos una petición para que nos manden los últimos mensajes con fecha posterior al último mensaje que hayamos recibido.

Finalmente me gustaría añadir que gracias al este diseño también se permite que el plugin encargado de mandar las notificaciones Push funcione correctamente estén los usuario en línea o desconectados ya que cuando se manda un mensaje a una sala, se leerá el listado de propietarios y se mandará una notificación push con el mensaje a cada uno de ellos excepto al remitente del mensaje.

4.3.2. Pantalla

Para presentar la información de una conversación electrónica un listado de los mensajes enviados y recibidos ordenados cronológicamente debería ser suficiente, sin embargo presentar así la información resulta poco elegante e incluso desagradable a la vista, sobretodo si la cantidad de texto es grande. Para mejorar la estética del chat nos seguiremos basando en ese principio de *lista de texto* ordenada pero intentaremos presentar la información utilizando más elementos gráficos que facilitarán la lectura de la conversación.

La idea principal de nuestro diseño es la de incluir los típicos globos de texto de distintos colores para distinguir los mensajes que mandamos nosotros de los que recibimos hecho que reforzaremos mostrando los globos de texto a izquierda o derecha de la pantalla para dar un efecto

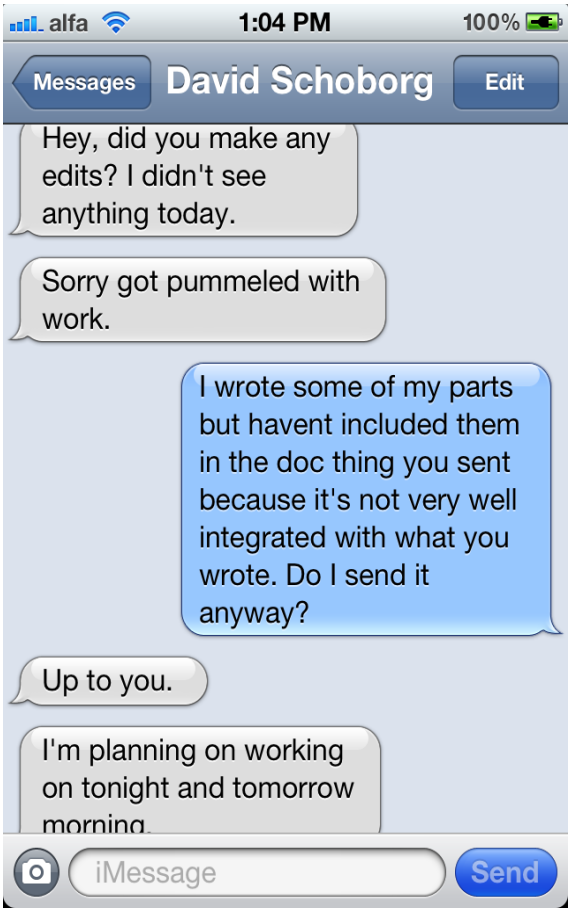

Figura 4.10: Captura de la aplicación de mensajería iMessage para iOS

similar al de las conversaciones de los cómics. También añadiremos otros elementos que facilitarán el seguimiento de conversaciones discontinuas en el tiempo como pueden ser la fecha en la que se envió el mensaje y qué usuario lo mandó, hecho bastante importante en las salas de chat que impliquen más de dos personas.

De nuevo nos encontramos ante un pequeño reto ya que los elementos que nos ofrece el framework gráfico de desarrollo no dispone directamente de globos de texto que aparezcan dónde y cómo nosotros queramos, por lo tanto nos tocará volver a implementar nuestros propios elementos gráficos personalizados partiendo de la sólida base que se nos ofrece con los elementos predeterminados.

Antes de entrar en detalle explicaremos de que elementos se compone una UITableView que es el nombre que reciben los elementos que muestran un listado en iOS, estos widgets se componen básicamente de celdas, de hecho podríamos decir que no son más que un conjunto de celdas puestas una encima de otra cosa que podemos aprovechar directamente sin realizar ningún tipo de cambio, al fin y al cabo como ilustra la figura-4.10, un chat es un listado ordenado de las frases intercambiada entre los diversos interlocutores.

Estas celdas que hemos comentado constituyen por sí mismas entidades independientes llamadas UITableViewCell que será el elemento que queremos personalizar para poder manejar y distribuir el contenido que se inserta en cada una de las celdas y que será mostrado como una lista por la UITableView. Las celdas básicas que nos ofrece el sistema están divididas por defecto en VARIAS sub vistas tal y como se puede apreciar en la figura-4.11, pero como no se ajustan

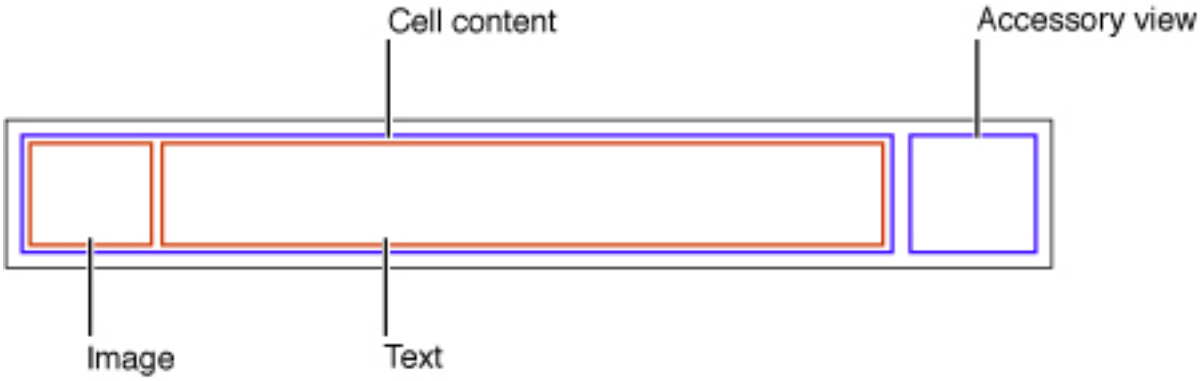

Figura 4.11: Esquema de las vistas que componen una celda básica

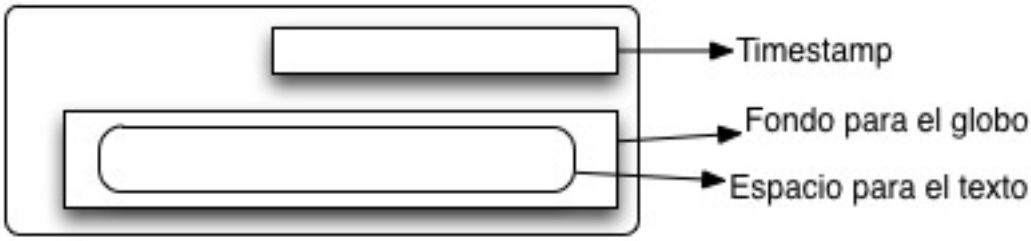

Figura 4.12: Esquema de las vistas que componen nuestra celda personalizada

a nuestras necesidades lo que haremos será cambiar las subvistas que forman la celda incluyendo aquellas que nos sean útiles para mostrar todas las imágenes y la información que queremos, la distribución final de los componentes de la nueva celda se puede apreciar en el esquema de la figura-4.12.

Como vemos necesitaremos una imagen que se dibujará en el fondo, esta imagen será el globo que aparece desde la izquierda que será el que utilicemos para mostrar mensajes enviados por nosotros o bien el globo que aparece desde la derecha que utilizaremos para mostrar los mensajes que nosotros recibimos; sobre la imagen del globo mostraremos el nombre del usuario que ha escrito el mensaje (o la palabra *Yo* si somos nosotros el remitente), el cuerpo del mensaje en sí y el momento en el que se envió el mensaje. Todos los elementos excepto el globo de texto son *UILabels* o etiquetas y la más interesante es la que contiene el texto del mensaje como veremos en unos instantes.

Si simplemente componemos cada celda como acabamos de describir lo que tendríamos sería un fondo fijo que mostraría un pequeño globo de texto y un texto que en la mayor parte de los casos cubriría por completo la imagen del globo de texto tal y como se muestra en la figura-4.13. Lo que queremos es que el globo de texto se ajuste a las medidas de nuestro texto o viceversa, como jugar a hacer grande y pequeño el tamaño de la letra para introducirlo por completo en la imagen no facilitaría la lectura en absoluto lo que haremos será ajustar la imagen de fondo al tamaño del bloque de texto.

Para conseguir este propósito utilizaremos dos herramientas muy útiles del sistema, la primera de ellas es capaz de dado un texto decirnos el espacio que va a ocupar tanto en altura como en longitud, la otra lo que nos permite es replicar secciones verticales u horizontales de una misma imagen cosa que será perfecta para poder construir nuestro globo de texto definitivo.

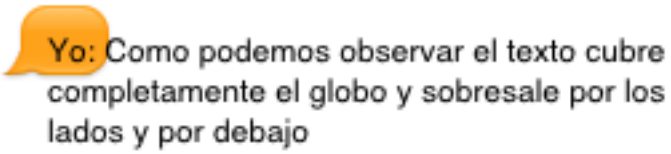

Figura 4.13: Texto cubriendo por completo la imagen del globo

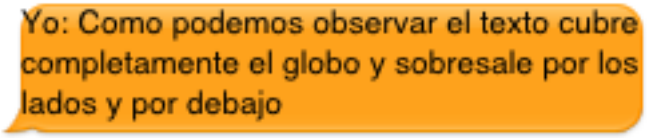

Figura 4.14: Texto tocando los márgenes del globo

Teniendo en cuenta lo que acabamos de explicar la construcción se consistirá en:

- Obtenemos las medidas del texto a mostrar
- Dada la morfología de la imagen utilizada para el globo nos damos cuenta de que si eliminamos los extremos redondeados el centro no es más que un rectángulo y por lo tanto replicando secciones verticales del centro de la imagen conseguiremos que se alargue mientras que si replicamos secciones horizontales lo haremos más alto y como disponemos de las medidas exactas que ocupará el texto únicamente debemos añadir un pequeño margen para no dejar el texto demasiado concentrado tal y como se puede apreciar en la figura-4.14.

Con esto ya tenemos la parte más complicada hecha ya que únicamente debemos añadir el resto de etiquetas con la fecha y el usuario y ya tendremos la estructura de cada celda como queremos. Eso sí, no debemos olvidar que esto hará que la altura de las celdas sea variable, pero como sabemos la altura de los componentes no será un problema indicarle al sistema el tamaño de cada celda de modo individual mediante las herramientas que nos ofrece *de serie* la UITableView. El resultado final lo podemos ver en la figura-4.15.

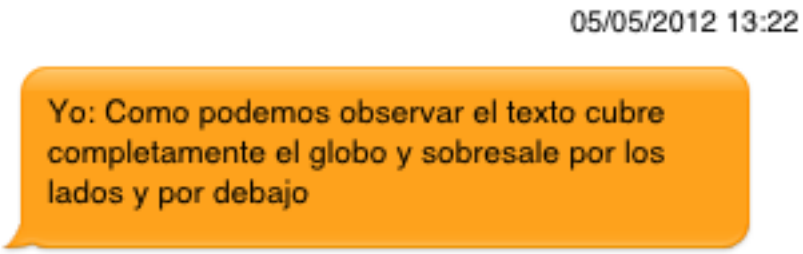

Figura 4.15: Globo de texto con márgenes y el timestamp

Capítulo 5

Conclusiones y ampliaciones

Una vez finalizado el proceso de desarrollo debemos realizar una valoración general del proyecto, repasando los objetivos del proyecto descritos a lo largo de la sección 1.1, viendo lo que se ha conseguido al final y también aportando mi opinión personal sobre lo que el proyecto ha supuesto para mí. También añadiré una pequeña sección con comentarios sobre posibles mejoras que se pueden realizar a la aplicación que todavía tiene un buen margen de crecimiento.

5.1. Objetivos

Una vez hemos llegado a este punto es obligado hacer un repaso al planteamiento inicial del proyecto para poder ver qué hemos conseguido tras más de cuatro meses de duro trabajo.

Podemos resumir los objetivos principales del proyecto en los siguientes puntos:

- Creación de un sistema de mensajería
- Adaptación al protocolo OAUTH
- Creación de elementos distintivos para la interfaz
- Creación de un sistema de avisos Push
- Integración con el sistema actual

Si vamos punto por punto podemos ver que se ha conseguido crear un sistema de mensajería accesible por todo aquel estudiante y profesor con acceso al Racó, implementando las bondades de los chats de antaño y añadiendo algunas de las funcionalidades de los chats modernos adaptados al entorno móvil como pueden ser la recepción de mensajes offline y la creación de charlas grupales persistentes en el tiempo. Así que podemos considerar que el objetivo principal del proyecto ha sido realizado con éxito. Y como característica adicional podemos decir que el sistema de chat creado sería plenamente integrable en cualquier aplicación creada para iOS, gracias a la modularidad con la que se ha diseñado.

Otro de los puntos clave del proyecto era la integración del protocolo OAUTH, tanto para acceder a la parte privada de la aplicación (la que contiene datos personales residentes en el sistema del Racó) como para utilizar el sistema de mensajería comentado en el punto anterior. Podemos decir que la aplicación ha sido capaz de integrar el protocolo OAUTH con un éxito rotundo, garantizando una mayor comodidad y seguridad para todos los usuarios del Racó Mobile. Además la inclusión en un futuro de nuevos servicios que requieran de este protocolo se podrán introducir en la aplicación sin apenas complicaciones gracias a una implementación limpia y clara.

También hemos conseguido crear un elemento distintivo capaz de diferenciar la interfaz de nuestra aplicación de la gran mayoría que podamos encontrar en la *App Store*[1], nos referimos a la rueda táctil que ha sustituido en el Racó Mobile a la típica barra de iconos. Además de dar un toque moderno y fresco a la aplicación es posible que siente las bases para un rediseño completo de la interfaz que aunque no resulte necesario a los usuarios les gusta ver cambios radicales de vez en cuando. Debo añadir que este componente pese a estar personalizado para nuestra aplicación, sería totalmente reutilizable en cualquier aplicación que originalmente utilizase las barras de control clásicas.

Las notificaciones Push también soy a día de hoy una realidad, están estrechamente ligadas al sistema de mensajería de nuestra aplicación y funciona perfectamente, tanto para los chats convencionales como para los chats multiusuario ya que el núcleo que genera el envío de estas notificaciones está integrado dentro de un plugin en nuestro servidor de mensajería Openfire.

Y llegamos a uno de los puntos críticos, se debía introducir todo esto minimizando al máximo los cambios en el sistema actual y podemos afirmar que también lo hemos conseguido. Los cambios han sido mínimos o incluso inexistentes para las funcionalidades y estructuras que ya estaban funcionando. En realidad únicamente se ha tenido que instalar y configurar el servidor de mensajería Openfire, sustituir el sistema de claves por el sistema OAUTH e introducir las nuevas características dentro de la antigua aplicación, el impacto no podría haber sido menor.

Sintetizando podemos ver como los objetivos han sido cubiertos satisfactoriamente, dejando una aplicación renovada y lista para seguir creciendo en el futuro.

5.2. Valoración general del proyecto

Podríamos decir que el proyecto ha resultado muy interesante en muchos niveles, tanto a nivel de estructura, como a nivel de aprendizaje pues el alcance del proyecto ha permitido al equipo de proyecto formarse en nuevas tecnologías para ser capaces de superar el reto que se presentaba al inicio.

Ser capaces de hacer crecer la aplicación día a día ha supuesto un esfuerzo considerable, sin embargo todo esfuerzo tiene su recompensa, pues no hay mejor recompensa que ver un trabajo terminado después de largas jornadas de duro trabajo, además las ventajas de las nuevas tecnologías es que se expanden rápidamente y por lo tanto podemos estar prácticamente seguros de que el software va a llegar a una cantidad considerable de terminales hecho que ya de por sí representará un indicador del éxito del proyecto.

Esperamos pues haber contribuido al crecimiento de la comunidad de la Facultad de Informática de Barcelona, queríamos ser capaces de conectar a nuestros compañeros, profesores y alumnos de algún modo y lo hemos conseguido. Esperemos que sean esos mismo alumnos y profesores los que continuen en un futuro con la labor de mejorar y hacer crecer este proyecto que de algún modo nos pertenece un poco a todos.

5.3. Valoración económica

Toda realización de un proyecto conlleva un coste tanto en materiales como en personal, en este caso al tratarse de un proyecto de final de carrera no existe este factor económico pero siempre es positivo realizar una valoración del coste real del trabajo realizado.

[1] Tienda de aplicaciones de Apple

En primer lugar calcularemos el coste del trabajo realizado por el personal que en este caso es la misma persona que ha desarrollado las tareas necesarias en cada momento. Dividiremos las tareas según perfiles reales de facturación y tomaremos los precios aproximados de mercado para cada uno de ellos.

El proyecto se inició el día 20/02/2012 y se entrega el día 22/06/2012, en dicho período de tiempo se comprenden 19 semanas y 6 días de vacaciones, si realizamos los cálculos nos salen 89 días laborables y 38 días en fin de semana. Los días laborables la carga media de trabajo es de aproximadamente unas 4 h/día, durante los fines de semana la media queda en 8 h/día lo cual nos da un resultado total de 660 horas. En la figura-5.1 observamos la distribución de las horas según el perfil técnico y el cálculo de los totales.

Además de los costes de personal un proyecto real también suele tener un coste en materiales, en el cual se comprendería el software, la licencia de desarrollo de Apple y los costes de servidores y equipos de desarrollo (iPhone y macbook), pero en nuestro caso el software utilizado no tiene coste alguno, pues Xcode es gratuito, la licencia de desarrollo nos la ha prestado la universidad durante el proyecto y los servidores utilizados son los que ya había en la facultad. Por lo tanto no vamos a prescindir de los costes derivados de los materiales.

Perfil técnico	Tareas	Precio/hora	Horas	Total
Jefe de proyecto	Análisis inicial Requisitos Planificación	42€	110	4.620€
Analista/Diseñador	Especificación Diseño Arquitectura Documentación	30€	175	5.250€
Programador	Implementación Documentación	22€	375	8.250€
Total				18.120 €

Figura 5.1: Resumen del coste en recursos humanos del proyecto

5.4. Ampliaciones

Una aplicación de este tipo está destinada a crecer de un modo u otro pues hay que tener en cuenta que pertenece a la Facultad de Informática y como tal debería ser un buen ejemplo de lo que se puede conseguir aplicando ideas innovadoras, conocimientos de ingeniería y las últimas técnicas innovadoras.

Sin ir más lejos quizás un día la FIB o la UPC acaben creando algún tipo de red social, el Racó Mobile sería la plataforma ideal para integrar dicha red dentro del entorno móvil iOS además de poder aprovechar algunas de sus características como por ejemplo el nuevo sistema de mensajería. Ligado al factor social creo que sería una buena idea poder incluir una sección con acceso al foro de la facultad, para ello habría que adaptar el login al protocolo OAUTH y quizás construir alguna API encargada de descargar e insertar información en el foro.

Por lo que respecta al sistema de mensajería que es una de las características que se han añadido durante este proyecto también se puede mejorar añadiéndole funciones adicionales que se han quedado fuera del alcance de este desarrollo como por ejemplo la transferencia de archivos, tanto

de datos como de audio o la inclusión de avatares y estados personalizables.

La parte de OAUTH es muy posible que cambie en un futuro ya que como hemos comentado actualmente se está construyendo el estándar de su segunda versión y lo más probable es que la API del Racó acabe por actualizarse y por lo tanto el Racó Mobile también deberá hacerlo.

En lo referente a las notificaciones Push, no sería complicado implementar este sistema no para que únicamente de avisos del sistema de mensajería sino que también se podría construir un sistema ligado a la publicación de noticias y avisos que mandase una notificación al usuario cada vez que un profesor realizase un nuevo aviso en el Racó o cada vez que se suba una nueva noticia. Todo ello además podría ser configurable dentro de la aplicación.

Como apunte final comentar que actualmente la aplicación funciona correctamente en dispositivos iPhone, sin embargo cada vez existen nuevos gadgets como el iPad que están empezando a sustituir los equipos portátiles, sobre todo para realizar consultas mediante la red o el acceso a redes sociales, estos dispositivos trabajan a resoluciones distintas pero utilizan la misma base de código y por lo tanto sería realmente interesante ver el Racó Mobile convertido en una aplicación capaz de ejecutarse en un iPad de manera cómoda y elegante.

Esto han sido solamente algunas ideas que han ido surgiendo durante el desarrollo de esta nueva versión de la aplicación, lo más probable es que existan mejoras y ampliaciones que ni siquiera se han contemplado y que supondrían cambios impactantes. Esperemos que las generaciones futuras que continúen este proyecto que no para de crecer nos sorprendan con nuevas e impactantes funcionalidades.

Capítulo 6

Bibliografía

[1] http://appleweblog.com.

[2] http://code.google.com/p/javapns/.

[3] http://developer.apple.com.

[4] http://en.wikipedia.org/.

[5] https://github.com/robbiehanson/xmppframework.

[6] http://www.igniterealtime.org/projects/openfire/.

[7] http://www.stanford.edu/class/cs193p/cgi-bin/drupal/.

[8] http://www.theverge.com.

[9] http://xmpp.org/.

[10] Racó mobile: extensió de la intranet de l'estudiant de la fib per a dispositius mòbils ios.

[11] Erik M. Buck and Donald A. Yacktman. *Cocoa Design Patterns*. Addisson-Wesley, 2009.

[12] Fernando López Hernández. *El lenguaje Objective-C para programadores C++ y Java*. MacProgramadores, 2008.

[13] Kevin Smith Peter Saint-Andre and Remko Tronçon. *XMPP The Definitive Guide*. O'Reilly, 2009.