Enero | 2010

Desarrollo de una aplicación iPhone para interactuar con una vivienda domótica.

Carlos Serrano Galiana

Proyecto Final de Carrera dirigido por Joan Fons i Cors.

Escuela Técnica Superior de Ingeniería Informática. UPV.

Tabla de contenido

1. Introducción

El presente documento constituye la memoria del Proyecto Final de Carrera realizado por Carlos Serrano Galiana, alumno de la antigua Facultad de Informática de la Universidad Politécnica de Valencia (nueva Escuela Técnica Superior de Ingeniería Informática).

El PFC desarrollado tiene como título "Desarrollo de una aplicación iPhone para interactuar con una vivienda domótica" y ha sido dirigido por Joan Fons i Cors, profesor adherido al Departamento de Sistemas Informáticos y Computación.

1.1. Motivación

La selección de este proyecto como mi PFC viene derivada de mis intereses personales tanto por la tecnología involucrada como por la relación con el mundo de la domótica.

Poseo un terminal iPhone y siento una gran curiosidad por su tecnología y sus posibilidades técnicas. Así mismo, tuve la oportunidad de realizar un Erasmus Intensive Programme sobre inteligencia ambiental que me introdujo en el área de la domótica.

Aunando estos dos intereses, compartidos por mi director de proyecto, nació la propuesta de este PFC, con el que pretendo iniciarme en el desarrollo de aplicaciones para dispositivos móviles, obtener una visión del iPhone desde la posición del desarrollador y ahondar en las motivaciones y posibilidades de la domótica.

1.2. Contexto y tecnología

1.2.1. Domótica

La Wikipedia define la domótica como el conjunto de sistemas capaces de automatizar una vivienda, aportando servicios de gestión energética, seguridad, bienestar y comunicación. La domótica no es si no una rama más de la ciencia y de la tecnología al servicio del hombre,

que pretenden facilitar o hacer más cómoda su vida diaria. Desde hace años se vienen realizando progresos en este área, proponiendo nuevas tecnologías y implantando estándares para que, poco a poco, se convierta en una realidad extendida al ámbito cotidiano.

En el presente proyecto, se ha hecho uso de la experiencia del director en este ámbito. Se ha utilizado una maqueta de una vivienda eficiente desarrollada con el estándar europeo para domótica, EIB/KNX, como escenario de simulación de una vivienda rural, y la gestión de la misma desde una aplicación servidor desarrollada en un laboratorio de investigación del DSIC.

1.2.2. Terminal iPhone

El iPhone es un dispositivo móvil desarrollado por la multinacional Apple que combina tres productos en uno: un teléfono móvil, un reproductor de música y vídeo y un dispositivo de acceso a Internet. El terminal, por su cuidado diseño, su sencillez de manejo, sus características técnicas (incluida su pantalla táctil) y sus innumerables funcionalidades se ha convertido rápidamente en un referente en el mercado tecnológico.

Así mismo, el desarrollo de aplicaciones para este terminal está a la orden del día. Un kit de desarrollo software (SDK) propio de la marca permite a un programador crear aplicaciones que corran bajo el sistema operativo iPhone OS que utiliza el dispositivo. La propia casa permite comercializar a través de su tienda online (App Store) las aplicaciones desarrolladas por terceros, lo que dota de un dinamismo y una profundidad al mercado que lo hacen muy interesante.

En concreto, la programación para esta terminal se realiza utilizando el entorno de desarrollo Xcode y el lenguaje de programación Objective-C.

1.3. Propósitos y objetivos

El proyecto consiste en el desarrollo de una aplicación de control para vivienda domótica para un dispositivo móvil, pero con la dificultad y posibilidades añadidas que ofrece la programación para el terminal móvil iPhone de Apple.

En este proyecto se desarrollará una infraestructura que permita crear interfaces de usuario para controlar sistemas domóticos, en los que la interfaz permitirá ubicar los servicios y dispositivos de la(s) vivienda(s), así como utilizarlos y definir las diferentes ubicaciones o sectores de la vivienda para acceder a estos recursos.

El usuario debe poder pues, crear los entornos (viviendas) domóticos, con los diferentes elementos y distribución que considere oportunos, y utilizar la aplicación para la navegación intuitiva, sencilla y completa que permita el uso de los diferentes dispositivos y servicios.

No obstante, cabe destacar que este proyecto no se centra en la realización de la aplicación en sí, si no que se lleva a cabo con la finalidad de aprender a desarrollar aplicaciones para dispositivos móviles y explorar las posibilidades y características de la tecnología iPhone y iPhone OS. La aplicación desarrollada sirve pues como prototipo y primer paso para el desarrollo de aplicaciones relacionadas con la domótica para dispositivos móviles iPhone.

2. Entorno tecnológico

2.1. Desarrollo para dispositivos móviles.

Los dispositivos móviles han evolucionado enormemente en los últimos tiempos, desde PDAs hasta Smart Phones, estamos siendo testigos del boom de las tecnologías móviles. Esta evolución ha permitido y provocado el crecimiento exponencial de las aplicaciones desarrolladas específicamente para estos sectores, y el mercado de las mismas. Sin embargo, se trata de un sector muy específico, con unas posibilidades y limitaciones propias.

El desarrollo de aplicaciones para dispositivos móviles supone un reto para cualquier programador acostumbrado a la programación de aplicaciones de escritorio, o incluso de aplicaciones web. Las diferencias entre las capacidades técnicas son abismales, y esto supone una serie de factores a tener en cuenta a la hora de plantearse el desarrollo de una aplicación móvil. Lo más importantes son:

- Memoria y procesamiento

  Es imprescindible tener en cuenta las limitaciones de los dispositivos móviles en estos aspectos. No disponemos de los mismos recursos que nos encontraríamos en un ordenador tradicional, y por tanto a la hora de la programación hay que tener cuidado de no sobrepasar las capacidades técnicas de nuestro terminal. Las aplicaciones móviles tienden a ser más sencillas y con procesos más concretos que sus homólogas de escritorio. Una palabra define a la perfección este concepto: optimizar.

- Interfaz

  El tamaño de los dispositivos y de las pantallas de los terminales móviles ha de ser un factor considerado a la hora de diseñar la interfaz de nuestra aplicación. Un entorno completo y bien desarrollado para un escritorio puede ser totalmente

inviable para un terminal móvil. La sencillez en el diseño y la intuición del uso son requisitos para este tipo de aplicaciones. Por otra parte, algunos terminales como el iPhone brindan nuevas posibilidades como las pantallas multitáctiles o los acelerómetros.

- Almacenamiento

  Pese a los avances en este aspecto en los últimos tiempos, los dispositivos móviles todavía tienen una restricción notable en cuanto a espacio de almacenamiento disponible. Hay que cuidar en el desarrollo la cantidad de información a almacenar en el terminal, y buscar siempre posibles alternativas para reducir estas necesidades.

2.2. iPhone OS

iPhone OS comprende el sistema operativo y las tecnologías utilizadas para correr aplicaciones nativas en los dispositivos iPhone y iPod Couch. Muy relacionado con Mac OS X, iPhone OS fue diseñado para cumplir con las necesidades de un entorno móvil, donde las necesidades del usuario son ligeramente diferentes.

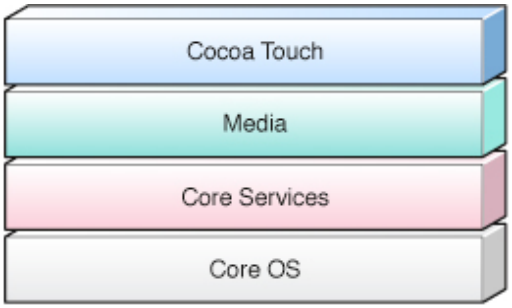
2.2.1. Arquitectura de capas del iPhone OS

En iPhone OS, la arquitectura de sistema subyacente y muchas de las tecnologías son similares a las encontradas en Mac OS X. El kernel en iPhone OS está basado en una variante de la misma base que el kernel de Mac OS X. Por encima de este kernel están las capas de servicios utilizadas para implementar aplicaciones en la plataforma.

Esta estructura de capas expande las posibilidades al implementar código. Por ejemplo, las capas Core OS y Core Services contienen las interfaces fundamentales para IPhone OS, incluyendo aquellas utilizadas para acceder a ficheros, tipos de datos de bajo nivel, sockets de red, etc. Estas interfaces están basadas en C y incluyen tecnologías como Core Foundation, SQLite y acceso a hilos POSIX y sockets UNIX entre otros.

Conforme nos desplazamos a niveles superiores, podemos encontrar tecnologías más avanzadas que utilizan una mezcla de interfaces basadas en C y Objective-C. Por ejemplo, la capa Media contiene las tecnologías fundamentales utilizadas para dar soporte al dibujo 2D y 3D, audio y vídeo. Esta capa incluye las tecnologías basadas en C OpenGL, Quartz y Core Audio. También contiene el motor de animación Core Animation.

En la capa Cocoa Touch, la mayoría de las tecnologías utilizan Objective-C. Los frameworks en estos niveles proporcionan la infraestructura fundamental utilizada por cualquier aplicación. Por ejemplo, el Framework Foundation proporciona soporte de orientación a objetos para colecciones, gestión de ficheros, operaciones de red y mucho más. El Framework UIKit proporciona la infraestructura visual para nuestras aplicaciones, incluidas clases para ventanas, vistas, controles y los controladores que gestionan estos objetos. Otros frameworks a este nivel dan acceso a la información de contactos e imágenes del usuario, acelerómetro y otras características hardware del dispositivo.



El punto de partida para cualquier proyecto es la capa Cocoa Touch y el Framework UIKit en particular. Cuando se plantea qué tecnologías adicionales utilizar, se recomienda empezar por los frameworks de nivel más alto e ir descendiendo por las capas conforme sea necesario. Los frameworks de más alto nivel hacen sencillo dar soporte a comportamientos del sistema estándar, con el mínimo esfuerzo por parte del desarrollador.

Podemos encontrar más información sobre la tecnología iPhone OS en el siguiente enlace.

2.2.2. Desarrollo para iPhone OS

La aplicaciones creadas por el programador reside en el Home del usuario, junto a otras aplicaciones del sistema como Photos, Weather o Clock. Una vez la aplicación es lanzada, aparte del kernel y algunos demonios de bajo nivel, la aplicación es la única corriendo en el

sistema. Mientras se ejecuta, la aplicación ocupa la pantalla completa y es el foco de atención del usuario. Cuando el usuario aprieta el botón Home, la aplicación finaliza y el sistema vuelve a la pantalla Home. Se puede tomar ventaja del hardware integrado como los acelerómetros o la gestión de gráficos para ejecutar nuestro propio código.

La manera con la que los usuarios interactúan con los dispositivos iPhone y iPod Couch es fundamentalmente diferente a como los usuarios interactúan con Mac OS X, y por tanto la manera de desarrollar las aplicaciones también debe serlo. En una aplicación iPhone, no hay concepto de ventanas de documentos para mostrar contenido. En su lugar, toda la información de la aplicación es mostrada en una única ventana. Esto ha llevado a la creación de nuevas vistas y controles que permitan presentar la información de la aplicación de forma organizada. Además, algunos de los elementos más estándares pueden comportarse de una manera ligeramente diferente. Estas diferencias son transparentes, pero requieren en ocasiones replantearse la forma de organizar y presentar el contenido de nuestra aplicación.

El modelo de manejo de eventos en iPhone OS representa una desviación de la las aplicaciones tradicionales de escritorio. En lugar de depender de eventos de teclado o ratón, iPhone OS introduce la idea de eventos táctiles (touch events). Un touch event puede ocurrir en cualquier momento y en combinación con uno o más eventos adicionales. Los toques pueden ser usados desde para detectar interacciones simples con el contenido, como seleccionar o arrastrar ítems, o hasta complejos gestos como los realizados para hacer funciones de zoom.

Más allá de considerar la estructura básica de la aplicación, hay que pensar en cómo van a utilizarla los usuarios. Las aplicaciones iPhone deben ser limpias y centradas en lo que los usuarios necesitan en cada momento. Cabe recordar que muchos de los usuarios de estos dispositivos necesitan acceder a información rápidamente y no perder mucho tiempo navegando entre capas y capas de la aplicación. Un diseño simple que destaque la información esencial que necesita el usuario es importante.

Como ya hemos mencionado, los frameworks que se utilizan al iniciar el desarrollo de una aplicación son Foundation y UIKit, que proveen los servicios básicos utilizados por todas las aplicaciones iPhone. Conforme se refina la aplicación, se pueden ir utilizando otros frameworks que ofrezcan los servicios que necesitamos.

En la web de desarrolladores de apple podemos encontrar más información sobre Foundation Framework Reference y sobre UIKit Framework Reference.

2.3. Herramientas de desarrollo: iPhone SDK

El iPhone Software Developement kit es un paquete para desarrolladores, preparado por Apple y distribuido gratuitamente con todo lo necesario para que cualquier programador pueda desarrollar sus propias aplicaciones para iPhone OS.

El iPhone SDK viene con todas las interfaces, herramientas y recursos necesarios para desarrollar aplicaciones iPhone desde cualquier ordenador Macintosh.

Apple distribuye la mayoría de sus interfaces de sistema en paquetes especiales llamados frameworks. Un Framework es un directorio que contiene una librería compartida dinámicamente y los recursos (tales como ficheros de cabecera, imágenes, aplicaciones de ayuda, etc.) necesarios para apoyar esa librería. Para utilizar frameworks, se enlazan en el proyecto de la aplicación como se haría con cualquier librería común. Enlazándolos al proyecto obtenemos acceso a todas las características del Framework y también permite a las herramientas de desarrollo saber dónde encontrar los ficheros de las cabeceras y otros recursos.

Aparte de frameworks, Apple también distribuye algunas tecnologías en forma de librerías compartidas estándar. Como iPhone OS está basado en UNIX, muchas de las tecnologías que forman los niveles más bajos del sistema operativo son derivadas de tecnologías open-source. Las interfaces para estas tecnologías están por tanto disponibles en las librerías estándar y los directorios de interfaz.

Algunos de los componentes claves del Software Developement Kit serían:

- Xcode Tools

  Proporciona las herramientas para permitir el desarrollo de aplicaciones iPhone, incluyendo los siguientes elementos:

  - Xcode

    Entorno de desarrollo integrado que gestiona los proyectos de aplicaciones y permite editar, compilar, lanzar y depurar el código. Xcode está integrado con muchas otras herramientas y es la aplicación de uso principal durante el desarrollo.

  - Interface Builder

    Una herramienta que se utiliza para montar la interfaz de usuario visualmente. Los objetos de interfaz que se crean son guardados en un fichero con formato especial y cargados en la aplicación en tiempo de ejecución.

  - Instruments

    Una aplicación de análisis de comportamiento y de debugging. Se puede utilizar para reunir información sobre el comportamiento en tiempo de ejecución de la aplicación y identificar potenciales problemas.

- iPhone Simulator

  Una aplicación para Mac OS X que simula la pila de tecnología iPhone, permitiendo probar las aplicaciones iPhone localmente en un ordenador Macintosh.

- iPhone Reference Library

  El SDK incluye documentación de referencia para iPhone OS por defecto y se pueden descargar versiones más completas, incluyendo ejemplos de código.

Aunque el SDK proporciona el software necesario para desarrollar aplicaciones, Xcode y Instruments también permiten interactuar directamente con un dispositivo adjunto para ejecutar y depurar el código en el terminal físico. El desarrollo en un dispositivo físico requiere registrarse en el programa de pago de Apple iPhone Developer Program y configurar el terminar para estos propósitos.

Más información sobre el iPhone Developer Program.

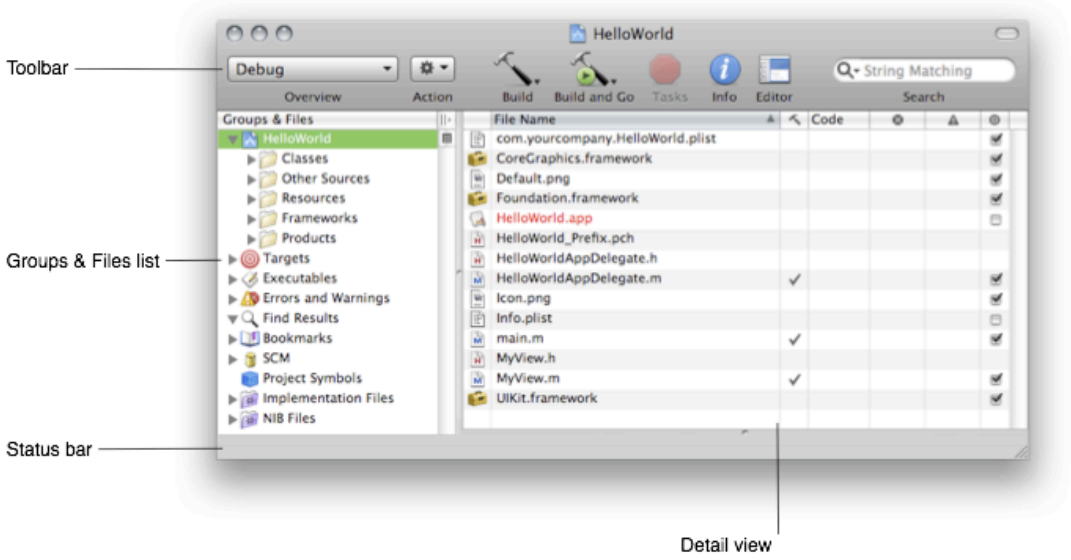
2.3.1. Xcode

El centro de la experiencia de desarrollo es la aplicación Xcode. Xcode es un entorno integrado de desarrollo que provee todas las herramientas necesarias para crear y gestionar los proyectos iPhone y sus ficheros fuente, compilar el código en un ejecutable, lanzar y depurar el código bien en el simulador iPhone o en un dispositivo. Xcode incorpora un gran número de características para hacer más fácil el desarrollo de aplicaciones iPhone, como son:

- Un sistema de gestión de proyectos para definir productos software.
- Un sistema de edición de código que incluye características como el coloreado de sintaxis, autocompletado de código y indexado.
- Un visor de documentación avanzado con sistema de búsqueda.
- Compiladores GCC para C, C++, Objective-C, Objective-C++ y Objective-C 2.0.
- Depuración a nivel de fuente utilizando GDB.
- Compilación predictiva que acelera el proceso.
- Soporte para snapshots de proyectos, que permiten una forma ligera de gestionar el código fuente.
- Herramientas de análisis de las aplicaciones.

Para crear una nueva aplicación iPhone se empieza por crear un nuevo proyecto en Xcode. Un proyecto gestiona toda la información asociada con la aplicación, incluyendo los ficheros fuente, la configuración de compilación y todos los elementos para juntar las piezas. El corazón de cada proyecto Xcode es la ventana del proyecto. Esta ventana proporciona acceso rápido a todos los elementos claves de nuestra aplicación. En la lista de grupos y ficheros se gestionan los ficheros de nuestro proyecto, incluyendo el código fuente. En la barra

de herramientas, se accede a los comandos y herramientas más comunes. En el panel de detalles, se puede configurar un espacio para trabajar en el proyecto.

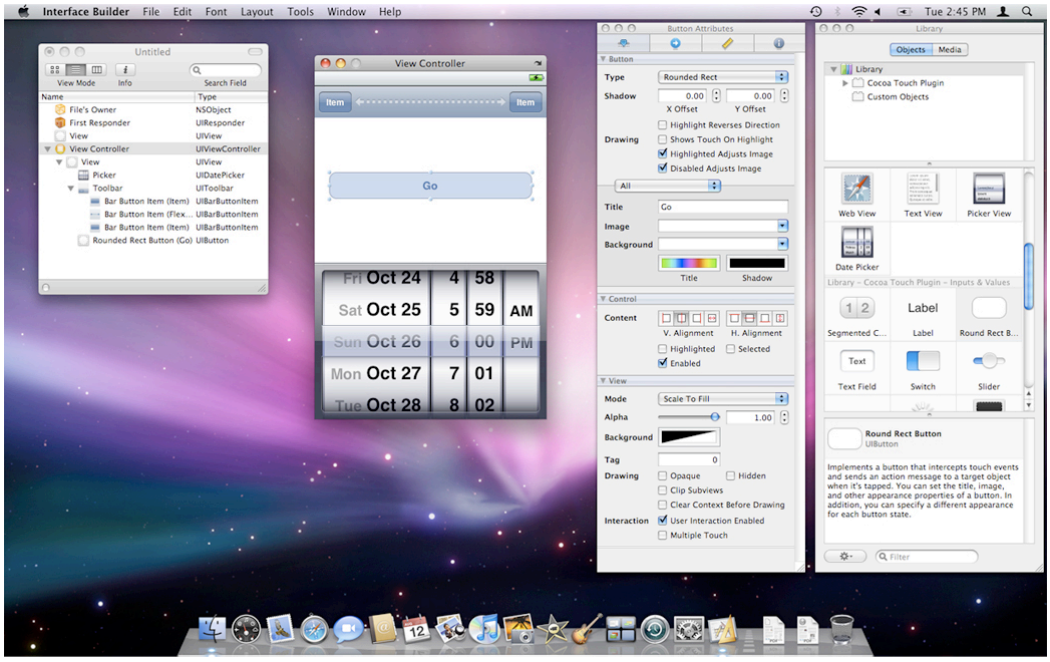


Cuando se compila una aplicación para Xcode, tienes la posibilidad de elegir compilarla para el simulador iPhone o para un dispositivo. El simulador proporciona un entorno local para testear las aplicaciones y asegurarse de que se comportan de manera esperada. Una vez se está satisfecho con el comportamiento, se puede compilar la aplicación y lanzarla en un iPhone o iPod touch conectado al ordenador. Correr la aplicación en un dispositivo físico proporciona un entorno de test definitivo, y Xcode permite depurar la aplicación corriendo en él.

2.3.2. Interface Builder

Interface Builder es la herramienta a utilizar para diseñar visualmente la interfaz de usuario de nuestra aplicación. Utilizando Interface Builder, podemos montar la ventana de nuestra aplicación arrastrando y soltando componentes preconfigurados. Se incluyen una serie de controles como switches, text fields o buttons.

Una vez se han colocado los componentes en la superficie de nuestra ventana, podemos posicionarlos a nuestro parecer, configurar sus atributos utilizando el inspector, y establecer las relaciones entre esos objetos y nuestro código. Cuando la interfaz está terminada, se guarda en un fichero nib, formato propio para las interfaces.



Los ficheros nib que se crean en Interface Builder contienen toda la información que el Framework UIKit necesita para recrear los objetos en tiempo de ejecución de la aplicación. Al cargar un fichero nib durante la ejecución esto crea una instancia de todos los objetos almacenados en el mismo, con la configuración que le dimos en el Interface Builder. También utiliza la información de conexiones especificadas para crear las conexiones entre las instancias creadas y cualquier objeto existente en nuestra aplicación. Estas conexiones proporcionan a nuestro código punteros a los objetos del fichero nib, y también se la proporcionan a los propios objetos para comunicar acciones de usuario al código.

En general, utilizar Interface Builder ahorra mucho tiempo en cuanto a crear la interfaz de usuario de la aplicación. Interface Builder elimina el código necesario para crear, configurar y posicionar los objetos que conforman nuestra interface. Por ser un editor visual, se consigue obtener en tiempo de desarrollo una visión exacta del aspecto que tendrá la interfaz en tiempo de ejecución, lo que lo convierte en una potente y conveniente herramienta durante nuestro desarrollo.

2.3.3. Instruments

El entorno de Instruments permite analizar el comportamiento de nuestras aplicaciones iPhone mientras corren en el simulador o en un dispositivo. Instruments recoge información de la aplicación en

ejecución y la presenta gráficamente en un timeline. Se puede obtener información sobre el uso de memoria, actividad de disco, actividad de red o comportamiento gráfico. El timeline puede mostrar todo tipo de información simultáneamente, para relacionar los datos obtenidos.

Además de esta vista del timeline, Instruments proporciona herramientas para analizar el comportamiento de nuestra aplicación en el tiempo. Por ejemplo, podemos salvar la información de múltiples ejecuciones para ver el desarrollo del comportamiento, compararlo o ayudarnos a evaluar la necesidad de revisar alguno de estos aspectos.

3. Caso de estudio: Análisis.

Como ya hemos mencionado, la finalidad de la realización de este proyecto no es el desarrollo de una aplicación concreta, sino el aprendizaje del desarrollo de aplicaciones para dispositivos móviles, y la exploración de la tecnología iPhone y iPhone OS. Así pues, el contenido exacto se fue definiendo mientras se avanzaba en el aprendizaje y en las primeras versiones del prototipo.

Si se trazaron unas líneas maestras para el prototipo a desarrollar, así como una serie de funcionalidades o requisitos que podían llegar a implementarse. A continuación se presenta el análisis correspondiente al prototipo que finalmente se ha desarrollado.

3.1. Objetivos y requisitos de la aplicación.

3.1.1. Objetivo básico de la aplicación.

Se pretende el desarrollo de una aplicación para el terminal móvil iPhone de Apple que permita la navegación entre los dispositivos o elementos de una vivienda domótica previamente definida y modelada, así como la utilización o activación de dichos recursos a través de una conexión vía Internet con un servidor.

3.1.2. Especificaciones, requisitos y funcionalidades.

- La aplicación deberá permitir una navegación sencilla, cómoda e intuitiva entre los diferentes elementos domóticos incluidos en las viviendas que hayan sido previamente modeladas y configuradas. Se decide utilizar listados jerarquizados para presentar la información.
- La aplicación deberá permitir la carga o definición de diferentes modelos de viviendas o entornos domóticos. En dicha definición, se deberán poder especificar indefinidos niveles de agrupación (de ahora en adelante llamados sectores), que podrán contener a

su vez indefinidos elementos domóticos de cualquier tipo, así como otros sectores.

- Se definirán dos tipos de elementos domóticos: los dispositivos y los servicios. Un dispositivo corresponderá a un elemento físico utilizable, mientras que un servicio corresponderá a cualquier función domótica que no corresponda físicamente con un único elemento.

- Los dispositivos y servicios podrán ser de un único o varios tipos diferentes previamente definidos. En cualquier caso se deberá poder acceder al uso de cualquiera de las funcionalidades definidas para dichos tipos.

- La aplicación debe de poder comunicarse a través de Internet con un servidor que gestione la activación o uso de los diferentes dispositivos definidos, de manera que la activación en la aplicación suponga un cambio real en la vivienda modelada.

3.2. Casos de uso

Los casos de uso son un modelo enmarcado en el lenguaje gráfico UML (Lenguaje Unificado de Modelado) que nos permite, una vez capturados los requisitos, representar gráficamente cómo debería interactuar la aplicación con los usuarios.

Por lo tanto, los casos de uso están formados por actores (usuarios), por los propios casos de uso (funcionalidades) y por un diagrama de caso de uso (relaciones entre los usuarios y la aplicación).

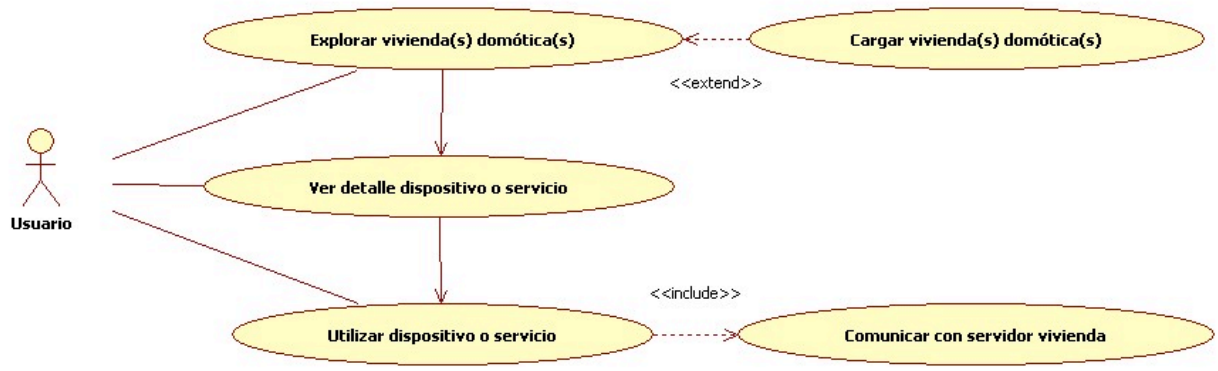
Entre los diferentes elementos de un diagrama de casos de uso pueden presentarse diferentes tipos de relación, de los que hemos utilizado:

- <<comunicate>> define una relación entre un actor y un caso de uso. Generalmente la relación únicamente se indica con una línea continua.

- <<include>> define una relación de dependencia entre dos casos de uso en el que un caso de uso A incluye a un caso de uso B, realizando una instancia de A todos los eventos descritos en B.

- <<extend>> define una relación de dependencia entre dos casos de uso en el que un caso de uso B es una especialización de un caso de uso A, es decir, una relación equivalente a una inclusión más una condición.

En nuestro proyecto, presentamos un diagrama de casos de uso sencillo, en el que se observan las principales interacciones que el usuario va a realizar con la aplicación.



Cabe destacar de nuevo que en las aplicaciones orientadas a dispositivos móviles se busca la sencillez, la comodidad y la rapidez de interacción entre el usuario y la funcionalidad. Es por eso que los casos de uso son tan sencillos como se pueda imaginar, para facilitar la política "on-the-go" de este tipo de aplicaciones, que busca que el usuario no necesite centrar su atención en el dispositivo más que unos segundos para obtener la funcionalidad deseada.

Vamos a ver a continuación una descripción de los casos de uso presentados en el diagrama:

- Explorar vivienda(s) domótica(s)

  Cualquier usuario que lance la aplicación estará comenzando a explorar las viviendas domóticas que hayan sido cargadas (ver caso de uso correspondiente). A través de un sistema de navegación entre tablas jerarquizadas, el usuario podrá explorar el contenido de las viviendas e ir viendo el contenido de cada sector de manera ordenada y clasificada.

- Cargar vivienda(s) domótica(s)

Al lanzarse la aplicación, como hemos dicho, el usuario comienza a explorar las viviendas domóticas que la aplicación haya cargado (ver apartado de carga de información dentro de la fase de diseño). Esta carga se produce de manera automática cada vez que se comienza a explorar las viviendas. Se carga toda la información a memoria y la aplicación trabaja con ella hasta que se cierra. Es por esto que la carga de la información se realiza cuando se comienza a explorar, pero una única vez.

- Ver detalle dispositivo o servicio

  Durante la navegación el usuario puede seleccionar un dispositivo o servicio (y en el futuro cualquier elemento domótico que se implementara) para observar su detalle (a través de la carga de su interfaz que veremos en el apartado de diseño). Esto mostrará toda la información relevante y proporcionará la posibilidad de hacer uso de sus funcionalidades asociadas. En caso de elementos de varios tipos (por ejemplo una bombilla que se pueda apagar/encender o regular), se presentará la posibilidad de acceder a tantas vistas de control como sea necesario una vez se haya seleccionado el elemento.

- Utilizar dispositivo o servicio

  Una vez el usuario se encuentre en la pantalla de detalle de un elemento domótico, ésta mostrará los controles asociados al tipo correspondiente del elemento, permitiendo al usuario accionar o hacer uso de las funcionalidades asociadas ha dicho elemento según el tipo que corresponda (ver apartado de diseño).

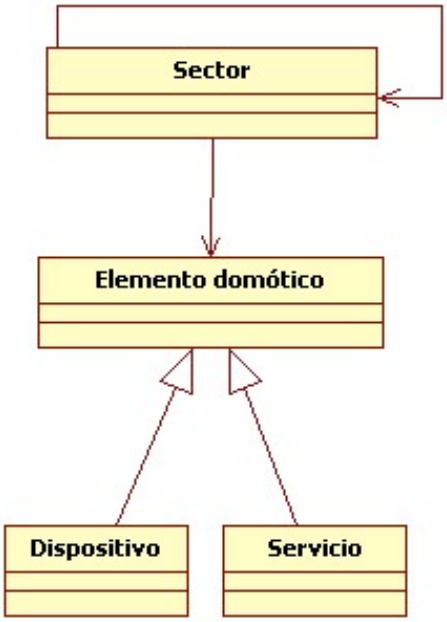
- Comunicar con servidor vivienda

  Cada vez que el usuario proceda a utilizar un elemento domótico, esta acción resultará en una comunicación directa con el servidor de la vivienda definido, a través de una señal vía Internet que proceda a realizar la función deseada en la vivienda.

3.3. Diagrama de clases

Haremos uso de los diagramas de clases, un tipo de diagrama estático de UML que nos permitirá describir y visualizar las clases que componen el sistema y las relaciones que existen entre ellas.

Se define una clase como una unidad básica que encapsula la información de un objeto, y se muestran atributos y métodos de las mismas.



Así pues, y desde un punto de vista del análisis de la aplicación, sólo hemos incluido las clases que almacenan información sobre objetos, y no aquellas clases que han sido necesarias para la gestión de la información (controladores de vistas, delegado de la aplicación, XML Parser…). Obtendremos más información sobre estas clases y las funciones que proporcionan a la aplicación en apartados posteriores de la presente memoria, limitándonos en este diagrama a las clases contempladas en la fase de análisis, sin además especificar las propiedades de las mismas.

4. Diseño

Vamos a ver a continuación las características del diseño de la aplicación. Para ello, presentaremos un diagrama de clases de la aplicación desarrollada en detalle, con todos las clases y relaciones definidas en este proyecto, como refinación del presentado en la fase de análisis. Realizaremos una descripción de las funcionalidades de cada clase, así como de sus propiedades. Veremos también la definición del esquema XML utilizado para la carga de información en la aplicación, y por último detalles de la interfaz gráfica desarrollada para cumplir con las especificaciones propuestas en la fase de análisis.

4.1. Diagrama de clases ampliado.

A continuación vamos a ver una ampliación del diagrama de clases presentado en el apartado de análisis.

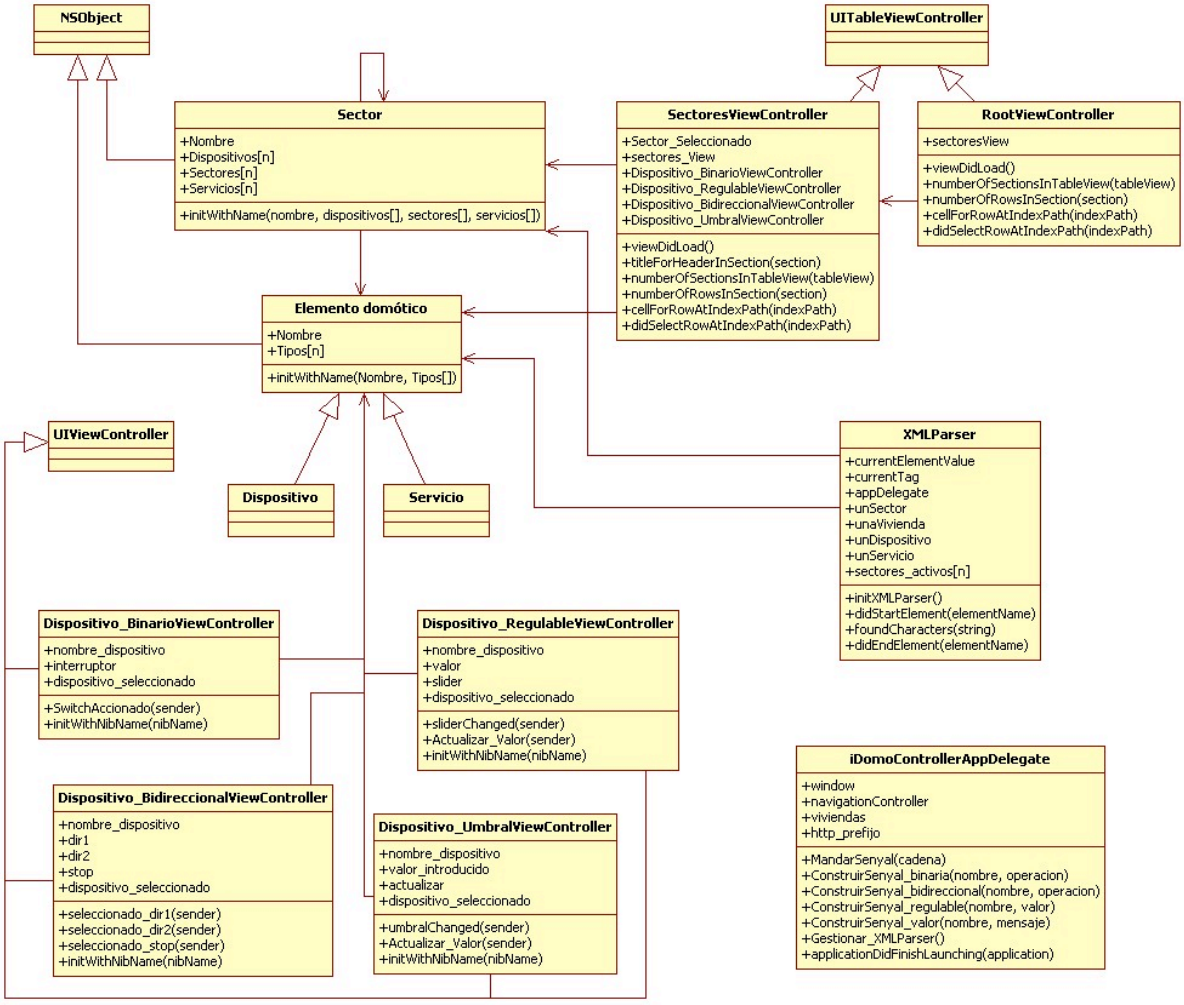
En el diagrama de clases de la fase de análisis presentamos únicamente las clases necesarias para modelizar la información manejada en la aplicación, sin entrar en detalle en su funcionamiento o finalidad y limitándonos a este concepto. Además de éstas, existen en el proyecto muchas otras clases utilizadas para otros menesteres, tales como los controladores de vistas, la clase de recorrido del XML o el delegado de la aplicación.

En las clases presentadas no se han implementado ningún método del tipo “set” o “get”, que nos permiten dar valor a los atributos de una instancia de una clase o recuperarlo, ya que éstos son implementados de manera automática al definir los atributos como propiedades. Estos métodos, pese a no haber sido necesario implementarlos, sí han sido utilizados y cabe destacar que por defecto se han definido todos los atributos de las diferentes clases como propiedades con la finalidad de poder utilizar los getters y setters en cualquier momento.

```
@property(nonatomic, retain) NSString *nombre;

@synthesize nombre;
```

Además, en el diagrama de clases presentado, por motivos de espacio, no se han incluido las relaciones entre la clase "iDomoControllerAppDelegate" y el resto, ya que queda relacionado con todas ellas, así como algunas otras relaciones, que iremos viendo en detalle a continuación. Veamos el diagrama.



4.1.1. Clases de modelización de información.

Las clases "Sector", "Elemento domótico", "Dispositivo" y "Servicio", todas heredadas de la clase interna de Foundation "NSObject", son las diseñadas con la finalidad de modelizar la información en nuestro proyecto. Con ellas podemos almacenar y trabajar con nuestros entornos domóticos. Veamos en detalle cada una de ellas.

- Sector

  La clase Sector representa toda unidad de agrupación de elementos domóticos. Así pues, un sector podría ser una habitación, una planta o incluso una vivienda. Un sector puede contener otros sectores y/o elementos domóticos tales como los dispositivos o los servicios. Estas relaciones de asociación de la clase Sector consigo misma y con los diferentes elementos domóticos se puede apreciar en el diagrama.

  Un sector poseerá como atributos un nombre, una lista de los sectores que están contenidos en esta agrupación, una lista de los dispositivos que contiene y otra lista con los servicios que agrupa.

  Se ha definido un método de inicialización para los objetos de la clase Sector que, recibiendo toda la información sobre sus atributos, cree y inicialice la instancia.

- Elemento Domótico

  La clase Elemento Domótico es una superclase de la que heredan las clases servicio y dispositivo. Pese a que habitualmente no se instanciarán objetos de esta clase, nos sirve para agrupar todos los elementos domóticos que podamos utilizar en un futuro, y para estructurar el código de manera que no sea necesario repetir definiciones de atributos y métodos comunes a estas otras clases.

  Un Elemento Domótico (y por tanto cualquiera de sus clases hijos) poseerá como atributos un nombre y una lista con los tipos de dicho elemento.

  Se ha definido también un método de inicialización para esta clase, que dado un nombre y una lista de tipos, cree y inicialice el objeto.

- Servicio

  Un servicio es una subclase de un Elemento Domótico. Se entiende por servicio cualquier función domótica que puede estar presente en una vivienda y no corresponda directamente con la activación de un único elemento físico.

Tal y como finalmente se ha decidido el funcionamiento y funcionalidades de la aplicación, no posee ningún atributo ni método propio, pero el hecho de existir una clase para estos objetos nos permite concretar de manera aislada su comportamiento y facilita desarrollos futuros sobre la base de esta aplicación.

- Dispositivo

  Un dispositivo es una subclase de un Elemento Domótico. Se entiende por dispositivo cualquier elemento físico que tenga una funcionalidad controlable asociada, como por ejemplo una bombilla que se enciende y se apaga.

  Al igual que los servicios, se ha creado una clase específica para su tratamiento que por el momento no tiene atributos o métodos propios.

4.1.2. Clases controladores de vistas

En nuestro proyecto se han definido 6 clases que actúan como controladores de vistas. Estas clases son las que nos permiten gestionar las diferentes interfaces gráficas definidas, tanto el contenido que se muestra, como la capa de negocio con la que la interfaz debe comunicar al usuario. Diferenciamos entre dos subtipos. Primero las clases heredadas de la clase interna del UIKit "UIViewController", donde enmarcamos las cuatro clases definidas como controladores de las cuatro interfaces de dispositivos. Segundo las clases heredadas de la clase interna del UIKit "UITableViewController", donde enmarcamos las dos clases definidas como controladores de la interfaz de navegación. Veamos los detalles de cada controlador.

- RootViewController

  Clase creada como controlador de la interfaz de navegación para la gestión de la vista principal, en la que se representen las viviendas cargadas en nuestra aplicación. Como atributo se establece la relación con la clase SectoresViewController y como métodos encontramos refinamientos de los métodos propios de su clase padre, propios de la gestión de una interfaz de tipo navegación. Entre ellos encontramos métodos como los que definen lo que sucede cuando termina de cargarse la interfaz, el

número de filas de cada sección de la tabla, el contenido de las celdas o lo que sucede cuando el usuario selecciona una de ellas.

- SectoresViewController

  Esta clase es muy similar a la anterior, ya que también se utiliza como controlador de la interfaz de navegación. Esta clase, en concreto, gestionará la navegación a nivel de sectores, mientras que la anterior se reservaba para el nivel de viviendas. Se establece esta diferencia para poder diferenciar entre ambos y permitir una mayor flexibilidad en comportamiento y apariencia de la aplicación para estos dos niveles.

  Esta clase se relaciona directamente con todas las clases de modelización de la información presentadas, así como con los cuatro controladores de vistas de interfaz, relación que no se ha representado por claridad en el diseño. Los métodos implementados son aproximadamente los mismos que en la anterior clase, pero con comportamientos diferentes.

- Dispositivo_BinarioViewController

  Clase controlador de vista que hereda de la clase interna UIViewController. Utilizada para controlar la interfaz de elementos domóticos de tipo binario y gestionar la interacción de ésta con la capa de negocio.

  Los atributos definidos corresponden mayormente a los elementos gráficos de la interfaz, para permitir la gestión desde la clase de los mismos. Entre los métodos encontramos el método de inicialización y el que gestiona la activación del switch de la interfaz para construir la señal a mandar al servidor.

- Dispositivo_RegulableViewController

  Clase controlador de vista que hereda de la clase interna UIViewController. Utilizada para controlar la interfaz de elementos domóticos de tipo regulable y gestionar la interacción de ésta con la capa de negocio.

  Los atributos definidos corresponden mayormente a los elementos gráficos de la interfaz, para permitir la gestión desde la clase de los mismos. Vemos entre los métodos el que gestiona las acciones a tomar cuando se modifica el “slider” de la interfaz, así como el que se encarga de construir la señal de envío cuando se pulsa el botón correspondiente.

- Dispositivo_BidireccionalViewController

  Clase controlador de vista que hereda de la clase interna UIViewController. Utilizada para controlar la interfaz de elementos domóticos de tipo bidireccional y gestionar la interacción de ésta con la capa de negocio.

  Los atributos definidos corresponden mayormente a los elementos gráficos de la interfaz, para permitir la gestión desde la clase de los mismos. Se definen métodos para tomar acciones para cada uno de los posibles botones pulsados, gestionando la señal a enviar al servidor según se seleccione una de las direcciones o la parada del elemento.

- Dispositivo_UmbralViewController

  Clase controlador de vista que hereda de la clase interna UIViewController. Utilizada para controlar la interfaz de elementos domóticos de tipo umbral o de valor y gestionar la interacción de ésta con la capa de negocio.

  Los atributos definidos corresponden mayormente a los elementos gráficos de la interfaz, para permitir la gestión desde la clase de los mismos. Como método principal destacar la gestión del valor introducido para enviar al servidor la señal correspondiente.

4.1.3. Clase de recorrido del XML

La clase XMLParser es la clase definida para el recorrido y carga de la información almacenada en el fichero XML de carga (ver apartados al respecto). Esta clase se instancia desde el delegado de la aplicación al lanzar el programa, y se encarga de recorrer el fichero e ir creando las estructuras necesarias (clases de modelización de información) para almacenar la información que después pueda ser presentada y tratada.

Hereda de la clase interna NSObject y los atributos y métodos definidos en ella son los necesarios para cumplir con la finalidad expuesta.

4.1.4. Clase delegado de la aplicación

Esta clase, que hereda también de la clase interna NSObject, es la clase principal de la aplicación. iDomoControllerAppDelegate actúa como delegado de la aplicación en su ejecución, lo que significa que gestiona desde que se lanza hasta que se cierra el flujo del programa. Se podría comparar con la tradicional clase main, que si bien existe en este proyecto, únicamente da paso al delegado, que es quien realmente gestiona la ejecución.

Esta clase, pese a que no se ha representado en el diagrama por claridad, tiene relaciones con casi todas las clases diseñadas. Desde multitud de ellas se crean instancias del delegado para ejecutar los métodos en él definidos, y es el mismo delegado el que crea la instancia del XMLParser que permite la carga de la información. Además, también centraliza la formación y envío de señales al servidor que permiten que la aplicación interactúe con los entornos domóticos físicos.

4.2. Carga de datos: Esquema XML

Ante el problema de el almacenamiento y carga de información en la aplicación se barajaron diferentes posibilidades.

La primera consistía en cargar la información relativa a las viviendas a partir de un fichero que previamente debe haber sido almacenado en el terminal. Fue también la primera en ser desechada, pues almacenar un fichero en el terminal físico supone un gasto de espacio innecesario, Además, obviamente habría que generar primero el archivo y cargarlo en la terminal, lo que puede ser un impedimento a la hora de cargar nuevas viviendas o modificar las existentes.

La segunda opción barajada fue crear una base de datos, que iPhone OS soporta a través de SQL Lite. También fue desechada, puesto que si bien es una manera ordenada de almacenar y cargar la información, seguimos teniendo el problema de cómo un usuario podría definir una nueva carga de datos.

Estas dos primeras opciones podrían haber sido válidas si se hubiera desarrollado la aplicación de modo que el usuario definiera su vivienda a través de la interfaz de la propia aplicación. No obstante,

esto no era parte de la finalidad del proyecto, además de que si bien podría ser una buena manera de hacer cambios sobre viviendas ya creadas, definir una nueva vivienda al completo podría ser interminable, y como ya hemos mencionado en otros puntos, se busca la sencillez y rapidez de interacción entre el usuario y la aplicación.

Así pues, llegamos a la tercera y última opción, que fue finalmente implementada: La carga de datos a través de un fichero XML.

Esta opción no tiene inconveniente alguno. Para hacer uso de la aplicación debemos tener conexión vía Internet con el servidor de nuestra vivienda domótica, con lo que podemos utilizar éste mismo para almacenar el fichero XML con los datos de la vivienda a cargar. Esto nos exime de almacenar ninguna información en el terminal, ya que ésta se encuentra en la red y se carga cada vez que se lanza la aplicación, con lo que se puede modificar fácilmente el XML para reflejar por ejemplo un nuevo dispositivo, o para definir nuevos sectores o viviendas.

El modo de funcionamiento es el siguiente: Se implementa un XML Parser, es decir, una clase que gestiona cómo se recorrerá el fichero XML a cargar. Se define una ruta donde está almacenado el fichero (que puede ser, por ejemplo, la ruta del servidor con el que se comunica la aplicación para interactuar con la vivienda), y en la carga de la aplicación se le solicita al XML Parser que lo cargue y lo recorra, de modo que sean los métodos definidos en la propia clase los que recojan todo lo que el Parser lea al recorrer el fichero.

Así pues, se diseña una estructura para el fichero XML que se desee utilizar como fuente de datos para la aplicación. Vamos a continuación a definir los elementos de la estructura:

- Elementos domóticos

  Dispositivos o servicios. Se definen con la etiqueta <Dispositivo> o <Servicio> respectivamente. En su interior, se debe indicar el nombre, indicado con la etiqueta <Nombre>, y 1 o más tipos de elementos domóticos, con sucesivas apariciones de la etiqueta <Tipo> y el código numérico asociado al tipo en nuestra aplicación.

```
<Dispositivo>
    <Nombre>Dispositivo A</Nombre>
```

```
            <Tipo>3</Tipo>
            <Tipo>2</Tipo>
            <Tipo>1</Tipo>
        </Dispositivo>
```

- Sector o Vivienda

Se definen con la etiqueta <Sector> o <Vivienda> respectivamente. Si bien diferenciamos entre ambos conceptos, cabe recordar que una vivienda no es más que un sector de primer nivel, que almacena el resto de sectores y que también puede contener elementos domóticos. Así pues, dentro de un <Sector> o <Vivienda> debemos definir primeramente su <Nombre>, y a continuación, si procede, podemos definir tantos elementos <Servicio>, <Dispositivo> y <Sector> como corresponda. Es importante respetar este orden, para que el Parser sepa distinguir a qué sector pertenece los elementos que estamos indicando en todo momento.

```
<Sector>
        <Nombre>Sector 3</Nombre>
        <Dispositivo>
            ..
        </Dispositivo>
        <Servicio>
            ..
        </Servicio>
        <Sector>
            <Nombre>Sector3.1</Nombre>
            <Dispositivo>
                ..
            </Dispositivo>
        </Sector>
```

```
</Sector>
```

- Viviendas

  Es simplemente el tag que identifica el principio de la lista de viviendas. En su interior podemos encontrar diferentes viviendas definidas, ya que la aplicación permite la carga de un número indefinido de elementos <Vivienda>.

```
<Viviendas>
        <Vivienda>
                <Nombre>Vivienda 1</Nombre>
                ..
        </Vivienda>
        <Vivienda>
                <Nombre>Vivienda 2</Nombre>
                ..
        </Vivienda>
</Viviendas>
```

4.3. Interacción con las viviendas físicas.

Esta aplicación está pensada como una interfaz móvil de interacción entre el usuario que desea controlar su vivienda o entorno domótico desde cualquier lugar y el servidor de dicha vivienda que es el que gestiona la efectuación física de esas modificaciones.

Así pues, la aplicación se deberá comunicar con el servidor para que sea éste el que ejecute las acciones. En nuestro caso, y tal como veremos en el apartado de descripción del prototipo implementado, nos comunicaremos con un servidor situado en el DSIC que controla la maqueta de una vivienda domótica. Cabe destacar en esta fase de diseño que la comunicación se realizará a través de órdenes http gracias al servicio post implementado en el mismo.

La interacción con la vivienda física requiere la formación de una señal, que en este caso se trata de la creación de una cadena url concreta según dispositivo y acción, y el envío de dicha url al servidor a través de un post http.

Las señales a enviar en este proyecto tenían como característica que se diferenciaban según tipo de elemento domótico, y incluían en la ruta el nombre del elemento, así como una codificación de la acción a realizar. Por esto, los métodos de formación y envío de estas señales se han implementado a nivel de delegado de aplicación, mientras que en cada controlador de vistas de cada tipo de elemento, es donde se crea la instancia del delegado, se llama al método de construcción de señal y al de envío, según corresponda con la acción realizada por el usuario en la interfaz (y por tanto, capturada por este controlador).

En el delegado de la aplicación es donde se definen los métodos de construcción de señales (uno por cada tipo de elemento domótico, ya que como hemos dicho las señales se forman de manera diferente), teniendo en cuenta como parámetros nombre y acción a realizar para la señal a construir. También es aquí donde encontramos el método que gestiona el envío de la señal al servidor. Este método recibe como parámetro la cadena de conexión que previamente ha sido formada con la casuística indicada, la convierte en un objeto del tipo NSURL, crea una señal de post http y utilizando la conectividad que gestiona el propio iPhone OS, fuerza el envío de la señal. Este post http será gestionado por el servidor, que efectuará las acciones asociadas al mismo, ejecutando físicamente lo que el usuario ha solicitado en el dispositivo.

4.4. Interfaz gráfica

Hay que tener en cuenta que al final, es la interfaz gráfica lo que el usuario ve de la aplicación. Dada la importancia de la claridad, simplicidad y sencillez de uso en la aplicaciones móviles, es precisamente en esta fase del diseño donde hay que tenerlo en cuenta. Así pues, se define una única interfaz (y decimos interfaz puesto que el concepto vista es diferente en iPhone Os, donde una interfaz puede tener diferentes vistas) para la navegación por las viviendas cargadas. Se define también una interfaz por cada tipo de elemento domótico, en nuestro caso 4 interfaces que corresponden a los tipos de elementos definidos (ver prototipo).

5. Prototipo y implementación

Vamos a centrarnos en este apartado en el prototipo de aplicación para terminal iPhone que se ha desarrollado como parte de este proyecto. Vamos a ver cómo se han llevado a cabo aspectos del diseño de aplicación y cómo en concreto se efectúan todas las acciones que se han definido en anteriores apartados.

5.1. El prototipo: Una vivienda eficiente.

En el laboratorio del DSIC dirigido por el director de este proyecto, Joan Fons i Cors, han construido una maqueta que representa una vivienda eficiente y un servidor que gestiona la interacción con la misma a través de comunicaciones post por http. Es precisamente este entorno el que se ha utilizado en este proyecto para el desarrollo de un prototipo con los requisitos y el diseño detallados en la presente memoria.

5.1.1. Tipos de elementos domóticos.

Se definen cuatro tipos de elementos domóticos (ya sean dispositivos o servicios) en nuestra vivienda eficiente, que presentamos a continuación (ver apartado de interfaces de usuario):

- Binario

  Los elementos de tipo binario son los más sencillos que vamos a encontrar. Tienen dos estados (on/off, encendido/apagado, activado/desactivado…) y de ahí el nombre de los mismos. Por ejemplo una bombilla.

- Regulable

  Son elementos regulables aquellos que son susceptibles de ser controlados de manera regulada para cualquiera de sus aspectos. Por ejemplo, consideramos regulable una bombilla cuya intensidad se puede controlar representando la misma como un porcentaje.

- Bidireccional

  Son elementos que tienen dos posibles interacciones opuestas (y opcionalmente un tercer estado de reposo). Por ejemplo una cortina que pueda subirse y bajarse, o un depósito que se pueda llenar o vaciar.

- De valor

  Cualquier elemento con el que se pueda interactuar con un valor, en un concepto más amplio que los elementos regulables, y no necesariamente con valores de porcentaje o numéricos. Sería un ejemplo un climatizador que aceptara como input la temperatura a conseguir.

5.1.2. Elementos domóticos y distribución.

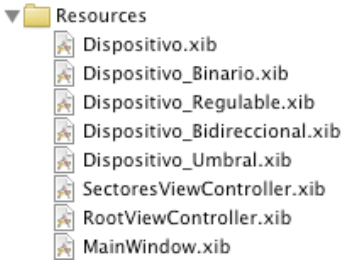
A continuación vamos a profundizar en cómo es nuestra vivienda eficiente. Se trata, como ya hemos introducido, de una maqueta que representa una vivienda con una serie de elementos domóticos diseñados con la eficiencia energética y la comodidad que ofrece la domótica como premisas.

Vamos a revisar todos los elementos domóticos de los que consta nuestra vivienda eficiente, así como su distribución en la misma.

Nombre	Tipo Elemento	Sector	Descripción
Flooding Security	Servicio binario	Vivienda	Servicio dedicado al control de inundaciones.
Water Supply	Servicio binario	Vivienda	Servicio que controla la provisión de agua corriente.
Home Security	Servicio binario	Vivienda	Activación de elementos tales como verjas, detectores, alarmas..
Home Messaging	Servicio de valor	Vivienda	Usos como displays de mensajes.
Buzzer	Dispositivo binario	Vivienda	Alarma o zumbador.
General Light	Dispositivo binario	Vivienda	Luz general de la vivienda.
Visual Notification	Dispositivo binario	Vivienda	Dispositivo de notificación.
Climated Room	Servicio binario	Dormitorio	Climatización del dormitorio. Controla temperatura.
Room Lighting	Dispositivo binario	Dormitorio	Iluminación del dormitorio.
Air Conditioned	Dispositivo binario	Dormitorio	Controla el aparato de aire acondicionado.
Kitchen Light By Presence	Servicio binario	Cocina	Encendido automático de iluminación en cocina.
Light By Light Intensity	Servicio binario	Cocina	Iluminación según iluminación natural.
Kitchen Lighting	Dispositivo binario	Cocina	Iluminación manual de cocina.
Louver	Dispositivo bidireccional	Salón	Persiana del salón.
Gradual Lighting	Dispositivo binario y regulable	Salón	Luz regulable del salón.

5.2. Interfaces de usuario

A continuación vamos a describir las interfaces de usuario diseñadas y implementadas con el Interface Builder para nuestro prototipo, que son la interfaz de navegación y las cuatro interfaces correspondientes a los cuatro tipos de elementos domóticos.



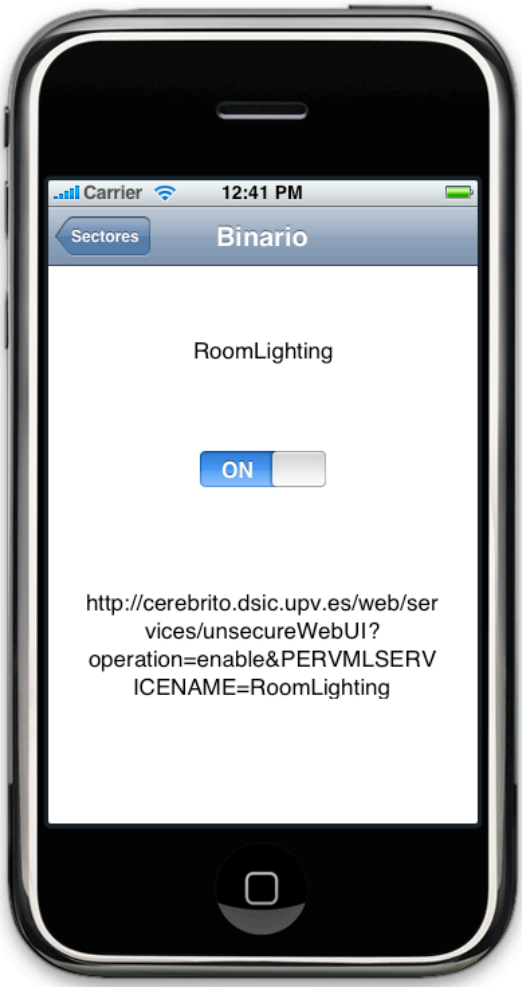
- Navegación

  El diseño de la interfaz de navegación lo proporciona el propio Interface Builder al utilizar una tabla para representar los datos. Como parte de una Navigation Based Application (aplicación basada en navegación) se añade una barra de navegación (Navigation Bar) en la parte superior, cuyo contenido es configurable. Se puede observar que se ha configurado el título de la misma, así como la aparición de un botón para regresar a la anterior pantalla de la navegación.

Esta interfaz permite pues navegar por todos los elementos domóticos y sectores de nuestra vivienda, de manera rápida e intuitiva. Además, al mostrar el contenido de un sector, se diferencia entre otros sectores, dispositivos y servicios, para una mayor claridad.

❖ Elemento binario

El diseño de la interfaz para elementos domóticos de tipo binario (recordemos que eran aquellos con dos estados), presenta un botón deslizante que permite al usuario con un sencillo gesto cambiar el estado del elemento. Esta interacción del usuario provoca el envío de la señal correspondiente al servidor para que se produzca le cambio. Dicha señal, y de manera provisional, se muestra en el inferior de la interfaz.



❖ Elemento regulable

El diseño de la interfaz para elementos domóticos de tipo regulable presenta un elemento gráfico muy característico, el "slider" (barra de deslizamiento), que nos permite mostrar un elemento con el que el usuario puede interactuar para indicar un valor dentro de un rango. Se añade también un botón para enviar la señal con el valor seleccionado cuando esté listo.

Nótese en la parte inferior de la imagen que éste es uno de los casos en los que un elemento domótico es de más de un tipo diferente. Por eso aparece en la interfaz una barra con tantas divisiones como de diferentes tipos sea el elemento, seleccionables para cargar la interfaz correspondiente a dicho tipo y así permitir cualquier interacción posible con el elemento.

❖ Elemento bidireccional

El diseño de la interfaz para elementos bidireccionales cuenta con tres botones. Dos botones corresponderían a las dos direcciones o estados contrapuestos que caracterizan este tipo de elementos, mientras que el tercer botón serviría para entrar en un tercer estado, de reposo, si existiera la posibilidad.

En nuestro prototipo el reposo de este tipo de elementos domóticos estaba programado en el servidor con un temporizador tras activar una de las dos direcciones, no existiendo la posibilidad de mandar una señal de parada, con lo cual la funcionalidad del botón de parada no se implementó. No obstante, pensando en la flexibilidad del proyecto desarrollado y en posibles ampliaciones o reutilizaciones, se diseñó el botón para poder implementar su funcionalidad si corresponde.

❖ Elemento de valor

El diseño de la interfaz para elementos de valor consta como el resto de las interfaces de una etiqueta con el nombre del elemento domótico que estamos manipulando. Así mismo, contiene una entrada de texto, que permite que el usuario introduzca cualquier valor deseado a través del teclado que IPhone OS despliega automáticamente cuando el usuario pulsa dentro del cuadro de texto. Por último, y pese a que se podría implementar el envío de la señal a través de la tecla correspondiente del teclado desplegado, he optado por añadir el mismo botón que en las otras interfaces por coherencia, que como en los otros casos, envía el valor establecido (en este caso una cadena) al servidor.

5.3. Carga del prototipo. Definición en XML

Como ya se introdujo en el apartado de diseño, la carga de datos en la aplicación se realiza mediante la lectura de un fichero XML

alojado en la red. Definimos también la estructura que debía cumplir el XML para ser leído por la aplicación. Vamos a ver a continuación en el caso concreto de nuestra vivienda eficiente cómo ha sido definido ese XML .

```
<Viviendas>
    <Vivienda>
        <Nombre>Vivienda eficiente</Nombre>
        <Servicio>
            <Nombre>FloodingSecurity</Nombre>
            <Tipo>1</Tipo>
        </Servicio>
        <Servicio>
            <Nombre>WaterSupply</Nombre>
            <Tipo>1</Tipo>
        </Servicio>
        <Servicio>
            <Nombre>HomeSecurity</Nombre>
            <Tipo>1</Tipo>
        </Servicio>
        <Servicio>
            <Nombre>HomeMessaging</Nombre>
            <Tipo>4</Tipo>
        </Servicio>
        <Dispositivo>
            <Nombre>Buzzer</Nombre>
            <Tipo>1</Tipo>
        </Dispositivo>
        <Dispositivo>
            <Nombre>GeneralLight</Nombre>
            <Tipo>1</Tipo>
        </Dispositivo>
        <Dispositivo>
            <Nombre>VisualNotification</Nombre>
            <Tipo>1</Tipo>
        </Dispositivo>
         <Sector>
             <Nombre>Dormitorio</Nombre>
            <Servicio>
                <Nombre>ClimatedRoom</Nombre>
                <Tipo>1</Tipo>
            </Servicio>
            <Dispositivo>
                 <Nombre>RoomLighting</Nombre>
                <Tipo>1</Tipo>
             </Dispositivo>
            <Dispositivo>
```

```
                <Nombre>AirConditioned</Nombre>
                <Tipo>1</Tipo>
            </Dispositivo>
        </Sector>
        <Sector>
            <Nombre>Cocina</Nombre>
            <Servicio>
                <Nombre>KitchenLightByPresence</Nombre>
                <Tipo>1</Tipo>
            </Servicio>
            <Servicio>
                <Nombre>LightByLightIntensity</Nombre>
                <Tipo>1</Tipo>
            </Servicio>
            <Dispositivo>
                <Nombre>KitchenLighting</Nombre>
                <Tipo>1</Tipo>
            </Dispositivo>
        </Sector>
        <Sector>
            <Nombre>Salon</Nombre>
            <Dispositivo>
                <Nombre>Louver</Nombre>
                <Tipo>3</Tipo>
            </Dispositivo>
            <Dispositivo>
                <Nombre>GradualLighting</Nombre>
                <Tipo>2</Tipo>
                <Tipo>1</Tipo>
            </Dispositivo>
        </Sector>
    </Vivienda>
</Viviendas>
```

El recorrido y interpretación de este XML es gestionado por la clase XMLParser, que ya hemos introducido y que veremos con detalle en el siguiente punto.

5.4. Descripción de la implementación

Vamos a ver, de una manera general, como se ha estructurado la implementación del proyecto. A parte de las interfaces diseñadas con el Interface Builder, se han programado una serie de clases que nos

sirven tanto para modelar la información que maneja la aplicación como para controlar el contenido visible por el usuario en dichas interfaces. Se han programado las siguientes clases:

- Clases de objetos modelados

  Las clases "Sector", "Dispositivo" y "Servicio" son las tres clases implementadas para almacenar y gestionar la información de los diferentes elementos que componen las viviendas en nuestra aplicación. Cada elemento domótico o sector de nuestras viviendas es una instancia de una de estas clases.

  En la fase de diseño vimos los atributos y métodos definidos en estas clases, que permiten la estructuración de la información a tratar en la aplicación.

  Como ejemplos de la implementación podemos ver por ejemplo el fichero de cabecera de la clase "Sector". Observamos la definición de atributos y de métodos. Tratándose de un sector, definimos su nombre y referencias a dispositivos, servicios y otros sectores contenidos. En cuanto a los métodos, únicamente se define la inicialización del sector.

```
#import <Foundation/Foundation.h>

@interface Sector : NSObject {
      NSString *nombre;
      NSMutableArray *dispositivos;
      NSMutableArray *sectores;
      NSMutableArray *servicios;
}

@property(nonatomic, retain) NSString *nombre;
@property(nonatomic, retain) NSMutableArray *dispositivos;
@property(nonatomic, retain) NSMutableArray *sectores;
@property(nonatomic, retain) NSMutableArray *servicios;

-(id)initWithName:(NSString *)n dispositivos:(NSMutableArray
*)disp sectores:(NSMutableArray *)sect servicios:(NSMutableArray
*)serv;

@end
```

  A continuación, como ejemplo de implementación de uno de los métodos, la implementación del método de inicialización de un servicio, donde vemos que se inicializan los atributos del mismo.

```
#import "Servicio.h"

@implementation Servicio
@synthesize nombre, tipo, tipos;

-(id)initWithName:(NSString *)n tipos:(NSMutableArray
*)kind_list{
      self.nombre = n;
      self.tipos = [[NSMutableArray alloc] init];
      for (int i=0; i<[kind_list count]; i++) {
            [self.tipos addObject:[kind_list objectAtIndex:i]];
      }
      return self;
}

@end
```

❖ Clases controladores de interfaces

Las clases "Root View Controller" y "Sectores View Controller", así como las clases "Dispositivo Binario View Controller", "Dispositivo Regulable View Controller", "Dispositivo Bidireccional View Controller" y "Dispositivo Umbral View Controller" han sido programadas como controladores de las 5 interfaces que se han definido. Estas clases nos permiten interactuar con dichas interfaces, y por tanto, con el usuario. Se gestiona en ellas el contenido e información que aparece en cada momento en la pantalla del dispositivo, así como lo que sucede cuando el usuario interactúa con él de cualquier modo.

Vemos a continuación la cabecera de la clase controladora de dispositivos bidireccionales.

```
#import <UIKit/UIKit.h>
#import "Dispositivo.h"

@interface Dispositivo_BidireccionalViewController :
UIViewController {
      IBOutlet UILabel *nombre_dispositivo;
      IBOutlet UIButton *dir1;
      IBOutlet UIButton *dir2;
      IBOutlet UIButton *stop;
      IBOutlet UITextView *aux;
      Dispositivo *dispositivo_seleccionado;
}

@property (nonatomic, retain) IBOutlet UILabel
*nombre_dispositivo;
```

```
@property (nonatomic, retain) IBOutlet UIButton *dir1;
@property (nonatomic, retain) IBOutlet UIButton *dir2;
@property (nonatomic, retain) IBOutlet UIButton *stop;
@property (nonatomic, retain) IBOutlet UITextView *aux;
@property (nonatomic, retain) Dispositivo
*dispositivo_seleccionado;

-(IBAction)seleccionado_dir1:(id)sender;
-(IBAction)seleccionado_dir2:(id)sender;
-(IBAction)seleccionado_stop:(id)sender;

@end
```

En este fragmento de código vemos la declaración de las propiedades de la clase. Recalcar que el tipo IBOutlet nos permite definir propiedades a nivel de codificación que después pueden ser, a través del Interface Builder, asociadas a objetos en el diseño. De este modo, se puede codificar acciones y propiedades de los objetos diseñados en el IB que son instanciados en tiempo de ejecución.

En la declaración de los métodos, observamos tres métodos que corresponden a las tres posibles interacciones que puede realizar el usuario con un elemento de tipo bidireccional. Nótese también que el prefijo IBAction establece el método, de forma análoga a lo visto en las propiedades, como interconectable con el Interface Builder. Así, se establecen las relaciones entre la interacción del usuario y las acciones a realizar en el Interface Builder, pudiendo implementar a nivel de codificación dichas acciones.

A continuación vemos la implementación de uno de estos métodos, en concreto el que gestiona las acciones derivadas de que el usuario seleccione el botón de una de las direcciones sobre un elemento bidireccional.

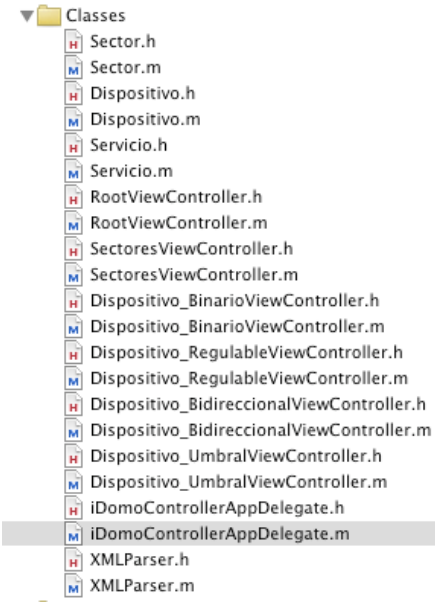
```
-(IBAction)seleccionado_dir1:(id)sender{
      iDomoControllerAppDelegate *appDelegate =
      (iDomoControllerAppDelegate *)[[UIApplication
      sharedApplication] delegate];
      NSString *cadena = [appDelegate
      ConstruirSenyal_bidireccional:self.nombre_dispositivo.text
      operation:@"enable"];
      aux.text = cadena;
      [appDelegate MandarSenyal:cadena];
}
```

Se observa, como ya se especificó en la fase de diseño, que estos métodos incluidos en los controladores son los que, al recibir el evento que realiza el usuario sobre el terminal, instancian al delegado de la aplicación y construyen la señal y la envían al servidor, produciendo el efecto deseado en la vivienda física.

❖ Clase XMLParser

La clase "XMLParser" se encarga de gestionar el recorrido del fichero XML, así como la interpretación y mapeado del mismo en estructuras definidas en nuestro proyecto (clases).

Ya sabemos que se lanza el parseado del fichero XML situado en una ruta definida, se va recorriendo y cada elemento que se encuentra (etiqueta, valor, fin de etiqueta…) es interpretado. Se programan las acciones a tomar en caso de encontrar cada uno de estos elementos de modo que una vez finalice la lectura del XML obtengamos una serie de instancias de las clases que hemos definido, organizadas en vectores de referencias que permiten "almacenar" la estructura de nuestra vivienda o viviendas sin gasto de espacio en disco.



Los métodos han sido implementados con mucho cuidado para que, cumpliendo con la estructura definida para la carga de ficheros XML, se permita cualquier número de niveles (sectores anidados), cualquier número de elementos domóticos y de

cualquier tipo. Veamos un ejemplo de la implementación de uno de estos métodos.

```
- (void)parser:(NSXMLParser *)parser didStartElement:(NSString
*)elementName namespaceURI:(NSString *)namespaceURI
qualifiedName: (NSString *)qualifiedName
attributes:(NSDictionary *) attributeDict {
      if([elementName isEqualToString:@"Viviendas"]) {
            appDelegate.viviendas = [[NSMutableArray alloc]
            init];
      } else if([elementName isEqualToString:@"Vivienda"]) {
            unaVivienda = [[Sector alloc] initWithName:@"prueba"
            dispositivos:[[NSMutableArray alloc] init]
            sectores:[[NSMutableArray alloc]init]
            servicios:[[NSMutableArray alloc]init]];
            sectores_activos = [[NSMutableArray alloc]
            initWithObjects:unaVivienda,nil];
            [unaVivienda release];
            unaVivienda = nil;
            currentTag = @"Vivienda";
      } else if([elementName isEqualToString:@"Sector"]){
            unSector = [[Sector alloc] initWithName:@"prueba"
            dispositivos:[[NSMutableArray alloc] init]
            sectores:[[NSMutableArray alloc] init]
            servicios:[[NSMutableArray alloc] init]];
            [sectores_activos addObject:unSector];
            currentTag = @"Sector";
            [unSector release];
            unSector = nil;
      } else if([elementName isEqualToString:@"Dispositivo"]){
            unDispositivo = [[Dispositivo alloc]
            initWithName:@"prueba" tipos:[[NSMutableArray alloc]
            init]];
            [[[sectores_activos lastObject]
            dispositivos]addObject:unDispositivo];
            currentTag = @"Dispositivo";
            [unDispositivo release];
            unDispositivo = nil;
      } else if([elementName isEqualToString:@"Servicio"]){
            unServicio = [[Servicio alloc] initWithName:@"prueba"
            tipos:[[NSMutableArray alloc] init]];
            [[[sectores_activos lastObject]
            servicios]addObject:unServicio];
            currentTag = @"Servicio";
            [unServicio release];
            unServicio = nil;
      }
}
```

❖ Delegado de la aplicación.

Existe una clase principal en los proyectos para iPhone OS, llamada en este caso "iDomoControllerAppDelegate", que actúa, como su nombre indica, como delegado de la aplicación. Esta clase recibe los eventos generales de la aplicación cuando esta se inicia. Así pues podemos encontrar métodos que responden a eventos tales como la finalización del lanzamiento de la aplicación, lo que nos permite manejar exactamente en todo momento el flujo de nuestra aplicación, y, entre otras cosas, lo que debe suceder antes de que el usuario vea la primera interfaz cargada. Es precisamente aquí donde podemos programar el recorrido del fichero xml.

Veamos algunos de los métodos implementados en esta clase, el método de construcción de una señal para un elemento de tipo binario, y el método encargado de enviar dicha señal.

```
-(NSString *)ConstruirSenyal_binaria:(NSString *)nombre
operation:(NSString *)operacion{
      NSString *aux=self.http_prefijo;
      aux = [aux stringByAppendingString:@"operation="];
      aux = [aux stringByAppendingString:operacion];
      aux = [aux stringByAppendingString:@"&PERVMLSERVICENAME="];
      aux = [aux stringByAppendingString:nombre];
      return aux;
}
-(void)MandarSenyal:(NSString *)cadena{
      NSURL *url = [[NSURL alloc] initWithString:cadena];
      NSMutableURLRequest *theRequest = [[NSMutableURLRequest
      alloc] initWithURL:url];
      [theRequest setHTTPMethod:@"POST"];
      NSURLConnection *theConnection = [[NSURLConnection alloc]
      initWithRequest:theRequest delegate:nil];
      [theConnection start];
}
```

Se puede observar en esta implementación lo especificado en fase de diseño en cuanto a la construcción de la señal. Una posible señal enviada al servidor en nuestro prototipo sería:

```
http://cerebrito.dsic.upv.es/web/services/unsecureWebUI?ope
ration=enable&PERVMLSERVICENAME=Buzzer
```

6. Problemas encontrados y desarrollos futuros.

6.1. Dificultades surgidas

Vamos a ver ahora algunos de los puntos que mayor dificultad o reflexión crearon durante la realización de este proyecto, ya sea por su dificultad, por los problemas creados o por lo comprometido de una decisión.

6.1.1. La creación y almacenamiento de la información

Nuestra aplicación trata de proporcionar una herramienta para el usuario interesado en interactuar con su vivienda o elementos domóticos. Esto hace que el hecho de configurar, definir, crear o cargar la información sobre esa vivienda o elemento sea un punto crucial a tener en cuenta.

Varias opciones fueron apareciendo sobre este tema. En un principio, con una mentalidad que tras el paso del tiempo acabé viendo como totalmente desenfocada, pensé que lo correcto es que la propia aplicación presentara unas pantallas de edición para poder ir creando, editando y borrando cada uno de los elementos que el usuario deseara. Este concepto, tan instaurado en la costumbre de desarrollar aplicaciones de escritorio, no tenía cabida en la programación de una aplicación como esta para un terminal móvil. Vuelvo a destacar, que la clave de una aplicación para un dispositivo móvil, y en concreto para el iPhone, está en la sencillez y claridad de la misma. Para empezar el hardware no está diseñado para trabajar con él durante un largo periodo de tiempo, mirando una pantalla de apenas unas pulgadas y tratando de escribir en un teclado "minúsculo". Pero más allá de esto, lo verdaderamente importante es que el usuario no desea pasar horas configurando el acceso a lo que verdaderamente le interesa, el control de su vivienda.

Por otra parte, era un claro requisito fundamental poder definir aquello susceptible de ser controlado por la aplicación. Barajé la idea de almacenar en un fichero binario en el terminal el contenido de la vivienda, que previamente tendría que ser definido… ¡dentro de la aplicación! No tenía sentido. Estaba ante el mismo problema, o aún

peor, tendría que definir la vivienda dentro del código, con lo que cualquier cambio supondría modificar la propia aplicación. Inviable.

Fue entonces cuando opté por la decisión que se ha implementado, almacenar ese fichero en el exterior y leerlo desde la aplicación. Y qué mejor manera que utilizar el lenguaje XML y almacenar el fichero en un servidor. Ya hemos visto como hemos definido la estructura a utilizar en la definición de nuestra vivienda y cómo se lee, se recorre y se interpreta su contenido. Una carga rápida, efectiva, sin necesidad de almacenar la información en el terminal, flexible y escalable… problema resuelto.

6.1.2. Elementos de varios tipos

En un momento dado, los requisitos iniciales se ampliaron y se contempló la posibilidad de que un mismo elemento pudiera pertenecer a más de un tipo. Esto tiene mucho sentido, ya que por ejemplo una bombilla regulable, puede ser controlada con una función de apagado y encendido (tipo binario) o puede ser controlada su intensidad progresivamente (tipo regulable). Como este, podrían presentarse muchos otros casos.

A nivel de capa de negocio y de datos no supuso demasiado problema. Se adaptó el desarrollo para que se permitiera tanto en la definición XML, como en la definición de los elementos domóticos (clases) y se gestionara adecuadamente. El problema llegó a la hora de representarlo visualmente, en el diseño de las interfaces.

La aplicación está basada en la navegación, lo que quiere decir que existe una pila de navegación, donde se pueden apilar o desapilar controladores de vistas, que son las clases que gestionan cada una de las interfaces. Así, cuando se selecciona un elemento, se apila en la "navigation stack" el controlador correspondiente al tipo del mismo. Así mismo, cada interfaz, que puede tener n vistas, tiene una vista por defecto, que se define en el Interface Builder. También se pueden apilar vistas de modo que cuando un controlador está el primero en la pila de navegación, es la vista que está la primera en la pila de vistas la que se visualiza.

La idea era crear un "tab bar", que es un objeto de interfaz que muestra una barra con diferentes pestañas y que nos permitiría seleccionar, una vez cargada una interfaz de un elemento, el resto de

interfaces de dicho elemento correspondientes a sus tipos. Fue aquí donde surgieron los problemas. Primero que Apple desaconseja el uso de este tipo de objetos dentro de aplicaciones de navegación, por problemas en el manejo de las pilas. El segundo, es que al haber definido una interfaz por cada elemento, el objeto tab bar, que se incluye en el diseño de una interfaz y se gestiona desde un controlador, no podía estar cargado en todo momento, puesto que para cambiar de un tipo a otro hacía falta apilar un nuevo controlador, y se pierde el control sobre el anterior. Todo este lío hizo casi imposible cumplir con este requisito. Finalmente y de forma provisional, conseguí implementar el tab bar como algo "pintado" encima de cada vista en cada momento que se cargaba la misma, pero resultó ser una manera engorrosísima de manejar, sucia de programar y no conseguí que esta dificultad fuera transparente para el usuario ya que quedaban residuos de este lío de interfaces, vistas, controladores y pilas que se producía, pese a que se consiguió que funcionara.

Es probable que el hecho de haber optado por definir una interfaz por cada tipo de elemento domótico sea un punto crítico en este aspecto. Si en lugar de 4 interfaces diferentes se hubiera creado una única interfaz con 4 vistas diferentes, sí podríamos haber insertado en la interfaz un tab bar, gestionado por un único controlador, que fuera manejando la pila de vistas una vez se carga cualquier elemento de cualquier tipo.

6.2. Posibles mejoras y desarrollos futuros

Además de el cambio de interfaces por vistas nombrado en el punto anterior, durante la realización del proyecto han ido surgiendo ideas de posibles mejoras, modificaciones o desarrollos futuros que se podrían llevar a cabo para mejorarlo, modificarlo o ampliarlo. Vamos a ver alguno de ellos.

6.2.1. Información almacenada en bases de datos

No considero esto una mejor, sino una variación o simplemente una alternativa al desarrollo planteado. La información sobre las viviendas, podría almacenarse en una base de datos implementada en el propio terminal. IPhone OS soporta el motor de bases de datos SQLite, lo que permitiría almacenar la información cargada en una base de datos, pudiendo por ejemplo implementar métodos y interfaces para hacer modificaciones, o un histórico de viviendas para poder ir haciendo "cargas" desde ficheros xml una única vez.

6.2.2. Información de estado de los elementos

En este proyecto hemos diseñado y desarrollado la interacción del usuario con su vivienda a través de un servidor que recibe peticiones POST http para modificar el estado de un determinado elemento domótico. Sin embargo, no obtenemos ninguna información al respecto. Por ejemplo, al seleccionar un dispositivo de tipo binario, se nos presenta la interfaz correspondiente, con el botón deslizante en estado "off". Esto se hace por defecto, y de poder consultar el estado del elemento podríamos situarlo en el estado que realmente estuviese en ese momento. Además, cuando nosotros mandamos una señal de cambio de estado, no obtenemos información sobre si dicho cambio se ha llevado a cabo o no. Así pues, lo representado en la interfaz y lo que en la realidad sucede puede estar desacompasado, cosa que resultaría incomodo y confuso para el usuario.

Esta modificación requeriría, además de los cambios de programación correspondientes en nuestra aplicación, que el propio servidor tuviera una metodología implementada para estas consultas, ya fuera devolviendo una respuesta con dicha información tras una petición http o de cualquier otra forma.

6.2.3. Interfaz gráfica alternativa

En las primeras reuniones que definieron a grandes rasgos el contenido del presente proyecto entre el director y yo, se puso sobre la mesa la posibilidad de desarrollar una interfaz gráfica que representara la vivienda eficiente de nuestro prototipo, y que

permitiera al usuario interactuar con la misma de una manera aún más intuitiva, utilizando una imagen en lugar de texto para mostrar todo el contenido de nuestro viviendo.

Las ventajas están claras. Aún más comodidad. El usuario podría simplemente tocar con su dedo encima del dispositivo deseado, o de una habitación (sector) para ver respectivamente la interfaz del elemento seleccionado o un listado como los que en el desarrollo actual se utilizan.

Sin embargo, esta idea puede tener también algunos problemas. El primero es que esto no permitiría o dificultaría la existencia de sectores anidados, ya que si se presenta una imagen de una vivienda, y esta tiene un sector "2ª planta", que a su vez tiene otras habitaciones y elementos, tendríamos que incluir en nuestra imagen de la vivienda un acceso a esa segunda imagen, y se complica más si pensamos en más niveles.

El segundo problema, mucho más grave todavía que este primero, es que esto no permitiría el uso de la aplicación para cualquier vivienda, ya que digitalizar una vivienda supondría la definición de esta imagen (no sólo la imagen en sí, si no toda la configuración de las áreas de la misma con la que el usuario puede interactuar). Se perdería así toda la flexibilidad de la aplicación, limitándola a su uso para una vivienda o entorno previamente fijado y desarrollado.

Lo considero pues una posible ampliación, que habría que sopesar y razonar previamente para solventar los problemas derivados expuestos. Además, se podría aprovechar el acelerómetro que posee el terminal iPhone para combinar la representación actual de texto con la representación propuesta de imagen. Por ejemplo, si el usuario mantiene el dispositivo en posición vertical, navegaría por la aplicación tal cual se hace en el presente desarrollo, mientras que si tumba el dispositivo situándolo en horizontal, pasaría a obtener una navegación con imagen. Sin duda, muy atractivo.

6.2.4. Pantalla de configuración

He hablado en múltiples ocasiones de la importancia de la flexibilidad de la aplicación. Sin embargo, en el presente desarrollo la comunicación con el servidor es de un determinado modo, y eso limita la utilidad de la aplicación. Sería muy interesante que se pudieran configurar algunos de estos parámetros, tales como la dirección del servidor con el que nos vamos a comunicar para interactuar con los elementos, o la ruta donde se encuentra el xml a cargar. Esto permitiría la posibilidad de que con la aplicación se pudieran controlar no sólo diferentes entornos domóticos, sino incluso entornos controlados por diferentes servidores.

Se podría diseñar una interfaz de configuración que se pudiera cargar desde la pantalla principal (donde se listan las viviendas cargadas) para configurar estos parámetros. Se plantea también como una posible ampliación.

7. ANEXO 1. Por dónde empezar

Empezar no sólo es el primer paso, sino que suele ser el más complicado. En el caso de desarrollo para iPhone OS, sobretodo si no se está habituado al desarrollo para entornos Mac o dispositivos móviles, esto puede ser aún mayor.

El primer paso, es ir a la web de desarrollo para IPhone de Apple (http://developer.apple.com/iphone/index.action) y registrarse con una cuenta de desarrollador. El registro gratuito nos da acceso a todas las herramientas y documentación existentes. El registro de pago como desarrollador nos permitiría probar nuestras aplicaciones en terminales iPhone y publicarlas en la App Store, pero esto ya se escapa de lo necesario.

En esa misma web, encontraremos un montón de documentación, pero antes de comenzar a leer, debemos descargarnos el SDK (Software Developement Kit) que nos proporciona Apple. Como ya vimos en los primeros apartados de esta memoria, incluye todas las herramientas necesarias para el desarrollo de aplicaciones para iPhone OS.

Hasta aquí los pasos básicos indispensables. Crear aplicaciones para iPhone OS es relativamente simple con las plantillas que incorporadas en el Xcode, pero crear aplicaciones que hacen algo útil y que tengan un aspecto agradable requieren mucho más tiempo de lectura de documentación. Recomendaría revisar la documentación propuesta por Apple en esta misma web, que nos da una idea general de cómo se fundamenta iPhone Os (si bien la lectura de esta memoria podría valer como un buen punto de partida).

Tras esta primera fase de lectura y como se suele decir habitualmente, la práctica hace al maestro. Mi recomendación es buscar webs con ejemplos y tutoriales, y comenzar a "trastear" con ellos, tratando de reproducirlos, y así, poco a poco, ir descubriendo todos los mecanismos, complejidades y posibilidades que nos brinda la programación para iPhone. Recomiendo entre otras:

- http://cocoadevcentral.com/d/learn_objectivec/
- http://www.iphonesoftware.es/
- http://icodeblog.com/

- http://www.iphonesdkarticles.com/

También recomiendo el curso de programación para iPhone que se llevó a cabo en la universidad Stanford, y que podemos encontrar íntegramente grabado en la red.

Por último, no conviene olvidar que existen múltiples foros de programación y en concreto dedicados al desarrollo de aplicaciones para iPhone OS que podemos encontrar muy fácilmente y que nos proporcionarán una ayuda inestimable cuando todo lo demás falle. Una búsqueda en cualquier navegador web nos devolverá decenas de resultados interesantes al respecto.