Universidad Autónoma de Madrid
Escuela Politécnica Superior

INTERFAZ DE CONTROL DE ENTORNOS DE INTELIGENCIA AMBIENTAL A TRAVÉS DE IPHONE

Ingeniería de Telecomunicación

David Matellano Criado
david.matellano@estudiante.uam.es
Diciembre 2010

INTERFAZ DE CONTROL DE ENTORNOS DE INTELIGENCIA AMBIENTAL A TRAVÉS DE IPHONE

AUTOR: David Matellano Criado
TUTOR: Javier Gómez Escribano
PONENTE: Germán Montoro Manrique

Grupo de Herramientas Interactivas Avanzadas (GHIA)
Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Diciembre 2010

Resumen y Palabras Clave

Resumen

Este proyecto parte de la necesidad de interactuar con entornos inteligentes a través de dispositivos móviles. Para ello, resulta necesario crear interfaces sencillas adaptadas al usuario, al terminal y al entorno en el que se ejecutan. En el proyecto se propone un sistema que, aprovechando el trabajo realizado previamente en el Laboratorio de Inteligencia Ambiental AmILab, genere automáticamente interfaces capaces de representar el entorno deseado.

Para ello, se ha creado una aplicación que, mediante la interpretación del sistema ontológico existente, crea una representación del espacio con los elementos contenidos en él. Estas representaciones se generan de forma automática y cambian en función de las modificaciones que se realicen sobre el entorno en tiempo de ejecución.

Para desarrollar todo esto, se eligió la plataforma de Apple: iOS. Con una especial relevancia dentro del mundo de los dispositivos táctiles, dispone de una gran colección de herramientas para la implementación de la aplicación. Además de una larga experiencia dentro del campo de interacción con el usuario.

Palabras Clave

Interfaces de usuario, Inteligencia Ambiental, dispositivos móviles, iOS, ontologia.

Abstract

This Project is driven by the need of interaction with intelligent environments through mobile devices. It is necessary to develop simple interfaces adapted to the user, the terminal and the environment in which they are executed. In this Project a system which automatically generates interfaces capable of representing the desired environment is proposed, using the previous work developed in the Environmental Intelligent Lab AmiLab.

To do so, an existent ontological system is used to create an application that produces a representation of the environment and the elements contained in it. These representations are automatically generated and change based on the modifications made to the space during the execution time.

The platform chosen to develop this Project is: iOS. Because of its relevance in the tactile devices world, this platform counts with a great collection of application implementation tools. It also provides with a long experience within the user interaction field.

Key words

User interfaces, ambient Intelligence, movile devices, iOS, ontology.

Agradecimientos

Quería dar las gracias en primer lugar a mi tutor, Javier, por haber estado siempre ahí en todo momento. Porque nunca me dio siquiera un "espera" por respuesta y siempre fue capaz de alegrarme en los momentos más duros de este trabajo. ¡Muchas gracias Nikonista!

También agradecer la dedicación de mi ponente, Germán, que fue mi segundo tutor a lo largo del trabajo y siempre estuvo presente a la hora de aportar ideas, apuntes y apoyo. Gracias Germán.

Lo mejor que me llevo de este proyecto son los nuevos amigos que me acompañaron dentro del AmILab. Gracias por hacer todo el trabajo mucho más ameno, por el apoyo recibido en todo momento y por la cantidad de cosas enseñadas. Sobre todo a Fer, que con su música hacía que todo fuera mucho mejor. Muchas gracias a todos.

No olvido tampoco a los amigos hechos a lo largo de toda la carrera. Compañeros que poco a poco fueron ocupando un lugar más importante en mi vida y con los que he compartido tantas clases, ratos en la cafetería y buenos momentos. Muchas gracias.

Que decir del Brujas y lo que a él acompaña. A mis amigos de toda la vida que siempre han estado ahí para cualquier cosa. Con los que he pasado tantas y tantas cosas... Jamás tendré suficientes palabras para expresaros lo importantes que sois en mi vida. Muchas gracias y ¡AUPA BRUJAS!

Mis Monis... Mis compañeros de alma... Nunca me arrepentiré de haber ido a aquel campamento que cambió mi vida. En el que conocí a nuevos amigos que más tarde se convirtieron en los mejores compañeros de trabajo que jamás podré tener. Con los que descubrí mi verdadera vocación y con los que siempre recordaré los mejores momentos de mi vida, descansando después de un duro día de trabajo jugando con esos pequeños genios que nos alegran la vida. En especial a ti, Julián, porque me diste la oportunidad de superarme y convertirme en gran parte de lo que hoy soy. No quiero ni pensar en lo que voy a echar esto de menos. Os quiero tanto... Gracias.

A ti Ana, porque fuiste a la persona que más he amado hasta ahora en mi vida. Porque me acompañaste durante la mayor parte de la carrera dándome el apoyo necesario para seguir avanzando día a día. Porque de ti solo guardo buenos recuerdos. Muchas gracias.

A esa pequeña lucecita que, aunque no lo sepa, o no lo quiera saber, ha hecho que mi vida vuelva a recuperar una ilusión que hacía tiempo que había perdido. Muchas gracias.

Y a ti, Laura. Porque desde que te conocí hace ya mucho tiempo, siempre has estado ahí. Por toda la cantidad de risas, alegrías, llantos y penas pasados. Porque eres cada vez más indispensable en mi vida y no veo el momento de separarme de ti. Nunca te podré agradecer suficiente todo lo que has hecho por mi.

Y por último, pero no menos importante, a mi familia. A los que están y a los que se han ido a lo largo de este camino que hoy pone un punto y a parte para pasar a una etapa mejor. Porque sois el mayor apoyo que he podido tener. Abuelos, tíos, primos... Gracias.

A ti, María. Por las risas compartidas, por las broncas tenidas, por las peleas de niños pasadas y por el saber aguantarme siempre. Porque te quiero mucho aunque no te lo diga, y cada día que pasa, más. Muchas gracias.

Y sobre todo, a ti Mamá y a ti Papá. Porque sois todo para mi. Porque después de luchar tanto juntos lo hemos conseguido. No tengo mayor objetivo en mi vida que llegar a formar algo como lo que vosotros habéis hecho. Con que estuvierais la mitad de orgulloso de mi de lo que yo lo estoy de vosotros me sobraría. Muchísimas, muchísimas gracias. Os quiero.

David Matellano
Diciembre 2010

A mis padres

Índice

Índice de Figuras

Índice de Tablas

Capítulo 1
Introducción

El concepto de entorno, al igual que el de tecnología, está en constante evolución. El número de construcciones que integran servicios domóticos para aportar mayor valor añadido a sus edificios aumenta y, dentro de poco, dotar de inteligencia a esos espacios será el nuevo objetivo a batir. El diseño de interfaces que permitan a los usuarios la interacción con el entorno será un punto clave para la perfecta integración de este nuevo concepto de edificio en la sociedad.

Los teléfonos móviles, muy presentes en nuestra vida cotidiana, podrían realizar dicha interacción gracias a la evolución que han sufrido a lo largo de los años. La nueva generación de "smartphones" tiene una alta capacidad de computación acompañada de la posibilidad de conectar con multitud de redes inalámbricas (Wi-Fi, UMTS, HSDPA, etc.) . Las cuales, por su parte, también han sufrido un gran avance, siendo capaces de proveer grandes anchos de banda en cualquier lugar.

El diseño de la interfaces en estos dispositivos es un factor clave para el correcto funcionamiento de éstas. En un teléfono móvil no disponemos de una gran pantalla, ni de un teclado o un ratón convencionales para la interacción con las aplicaciones. Por lo que la forma de presentar la información y de interactuar con el dispositivo debe de estar muy adaptada a las características de éste y del usuario.

Un buen ejemplo de interfaces adaptadas son las creadas por Apple en su plataforma de desarrollo: iOS. Basadas en un sistema operativo diseñado exclusivamente para sus terminales, la convierte en una de las más intuitivas en la actualidad. Si añadimos además que, desde el punto de vista del desarrollador, Apple ofrece un entorno de trabajo fuertemente equipado que cuenta con una alta penetración en el mercado, nos hace pensar que es una plataforma ideal para el desarrollo de aplicaciones para control de entornos inteligentes.

1.1. Motivación

Este proyecto nace con la idea de crear una aplicación capaz de representar y controlar un entorno inteligente dentro de un dispositivo móvil.

Las restricciones que tenían este tipo de dispositivos se han ido solventando con los avances tecnológicos permitiendo crear nuevas aplicaciones que, mediante el uso de interfaces adaptadas, permiten al usuario interactuar con diferentes sistemas de forma rápida y sencilla.

Por lo que la aplicación a desarrollar debe de ser capaz de adaptarse al entorno, al terminal así como a las necesidades del usuario. En este sentido, la plataforma iOS parece cumplir con los objetivos, ya que permite desarrollar aplicaciones con interfaces sencillas, intuitivas y, por lo tanto, cómodas para el usuario.

1.2. Objetivos

El objetivo de este proyecto es desarrollar una aplicación sobre la plataforma de Apple (iOS) que permita controlar el entorno inteligente integrado en el AmILab (Laboratorio de Inteligencia Ambiental de la Escuela Politécnica Superior de la UAM) a través de un iPhone o un iPod Touch desde cualquier lugar que disponga de conexión a Internet.

Para ello, se ha de crear un sistema de representación capaz de mostrar la información del espacio en tiempo real y, a su vez, que permita la interacción con éste mediante un sistema de interfaces dinámico adaptado al terminal, al usuario y al entorno en el que se ejecuta.

Como evaluación del sistema se realizarán diferentes pruebas dentro del Espacio Inteligente AmILab integrado en la UAM.

1.3. Organización de la memoria

En el primer capítulo se presenta el proyecto, la motivación y los objetivos del mismo.

En el segundo capítulo se hace un estudio sobre el estado del arte de los campos

asociados a este proyecto: plataformas móviles existentes, formas de generar interfaces y productos de mercado similares.

En el tercer capítulo se hace una descripción de las fases del diseño de la aplicación. En él aparece una breve descripción de los elementos en los que se basa la aplicación, así como las diferentes opciones durante el diseño y las elecciones tomadas según los criterios seguidos.

El cuarto capítulo trata sobre la fase de implementación y desarrollo de la aplicación.

En el quinto capítulo se presentan los ejemplos y las pruebas realizadas de la aplicación dentro de un entorno real con diferentes configuraciones y situaciones.

Las conclusiones obtenidas durante todo el desarrollo anterior aparecen en sexto capítulo, así como las posibles mejoras propuestas como trabajo futuro.

El Apéndice A es el presupuesto aproximado de los costes de este proyecto.

En el Apéndice B aparecen las condiciones legales que se han seguido en la realización del proyecto.

Dentro del Apéndice C se encuentra un fichero reducido de definición de entidades.

El Apéndice D contiene el póster informativo del seminario realizado para la introducción a la programación en iOS.

El Apéndice E contiene las transparencias utilizadas en dicha presentación.

Capítulo 2

Estado del Arte

2.1. Introducción

La Inteligencia Ambiental provee *"entornos altamente interactivos que usan computación embebida para observar y participar en las tareas cotidianas del mundo que les rodea"* [32] . La integración de estos servicios en casas u oficinas está cada vez más en auge, y la idea de poder interaccionar con los espacios desde cualquier lugar y en cualquier momento se hace cada vez más necesaria [41].

Las habitaciones, las oficinas, las casas y, en general, cualquier espacio que desee ofrecer este tipo de servicios, debe de estar equipado correctamente. De esta forma se podrá mejorar la calidad de vida de sus habitantes, ayudándolos, por ejemplo, en sus tareas diarias. La interacción entre el usuario y el espacio debe de ser sensible al contexto, adaptándose a la tarea, al entorno, los ocupantes y los recursos disponibles [46] [50].

Desarrollar estas formas de interacción [54], así como nuevos tipos de interfaces son las tareas que nos ofrecen estos espacios [53]. Teniendo siempre en cuenta que, tanto la representación como la interacción, se podrán realizar de forma convencional: a través de un PC (pantalla, teclado y ratón) [36], o a través de canales menos convencionales: mediante la voz [43], mediante mesas multi-táctiles [47] o a través de un terminal móvil [34].

Los objetos y personas contenidos en estos espacios no están fijos (Pueden moverse, relacionarse, desaparecer...). Por lo que nuestra forma de representación tiene que ser capaz de cambiar. Debido a ello, las interfaces se deben generar de forma automática y en tiempo real. Además, adaptarse al tipo de usuario que lo esté manejando, y no solo eso, sino también ser capaces de averiguar en qué entorno se está ejecutando para amoldarse a él, son necesidades que debemos satisfacer. Sin olvidar que la forma de

representación debe de ser clara, sencilla y, sobre todo, cómoda para el usuario.

Al igual que el contenido de los espacios se mueve, los usuarios también lo hacen y desean disponer de todos los servicios en cualquier momento y en cualquier lugar [41]. Gracias a las nueva generación de dispositivos móviles que tenemos en la actualidad y a la evolución de Internet Móvil, la idea de portar todos estos servicios se hace cada vez más atractiva.

Pero obtener la interfaz descrita anteriormente en este tipo de terminales solo es posible con una gran integración del software en nuestro dispositivo. Aprovechar al máximo los recursos, así como las formas de interactuar con el aparato serán las claves para que nuestra aplicación funcione [44].

El mercado ofrece multitud de dispositivos ("smartphones", "netbooks", "tablets"...) y plataformas (Android, Symbian, iOs...) diferentes, y elegir cuál de ellas se adapta mejor a nuestro Espacio Inteligente y, sobre todo, a las necesidades del usuario es un factor clave a la hora de diseñar e implementar las interfaces.

Cada una de estas plataformas posee diferentes características. Y no sólo a nivel de software, sino que también a nivel de hardware, ya que en la mayoría de los casos, cada plataforma va ligada a un tipo de dispositivo diferente. Esto dará lugar a que la forma de representar los elementos del entorno en la interfaz de usuario asociada varíe en función del tipo de terminal. Apreciándose claramente en el siguiente ejemplo: No es lo mismo representar el espacio inteligente en una pantalla táctil, donde al pulsar directamente sobre los elementos aparezca la información asociada a ellos, que representarla en un terminal que se maneje mediante botones físicos en la que tengamos que navegar a través de menús mediante los cursores correspondientes.

A continuación se presenta un repaso de las plataformas móviles más importantes de la actualidad (Sección 2.2) así como de programas capaces de generar interfaces de forma dinámicas (Sección 2.3) y una serie de ejemplos de representaciones ya creadas (Sección 2.4). Por último un apartado de conclusiones (Sección 2.5) en donde compararemos lo previamente descrito para argumentar la elección elegida.

2.2. Plataformas móviles

Si bien los dispositivos portátiles tienen limitaciones de recursos como la capacidad del procesador, el tamaño de la memoria o la forma de interactuar con él (tamaño de la pantalla, botones en vez de ratón...) [35] actualmente resultan muy atractivos para la investigación y desarrollo, ya que los avances que sufren últimamente ayudan a mitigar estas limitaciones. Además las plataformas de desarrollo móviles se han creado

con el propósito de aprovechar al máximo dichos recursos a la vez que favorezcan la interacción del usuario con el dispositivo. A continuación se comentarán las principales plataformas que conviven en la actualidad en los dispositivos móviles.

2.2.1. Java 2 Micro Edition (J2ME)

Es la plataforma desarrollada por Sun Microsystem [12] (Ahora propiedad de Oracle Corporation) que provee una colección de APIs ("Application Programming Interface") de desarrollo de software para terminales con recursos limitados.

Presentada en 1999 y, al igual que cualquier plataforma Java, permite la ejecución de aplicaciones a través de una máquina virtual (KVM: "Kilo Virtual Machine") consiguiendo así una alta portabilidad de aplicaciones independientemente del dispositivo a utilizar. Además, gracias a la gran capacidad de configuración que posee la KVM prácticamente casi cualquier terminal actual es capaz de cargar aplicaciones de este tipo [45]. Para hacernos una idea más clara de lo que es esta plataforma, podríamos hablar de que J2ME es una versión simplificada de la versión estándar (J2SE), en donde se ha reducido el API de Java para que ocupe menos espacio en el dispositivo [51].

La instalación de aplicaciones en esta plataforma es muy sencilla y rápida, tanto a nivel local como a través de un mecanismo OTA ("Over The Air": Descargar directamente a través del operador enviando un código desde el terminal) [49]. Sin embargo, carece de cualquier mercado común en donde aparezcan publicadas las aplicaciones para este tipo de plataforma, por lo que la descarga de la aplicación debe de ser a través de un servidor conocido previamente por el usuario o a través de la operadora móvil.

Los principales tipos de terminales asociados a esta tecnología son terminales antiguos o de "gama baja". En donde la interacción con el usuario se realiza a través de botones físicos y no a través de pantallas táctiles.

2.2.2. Symbian

Es la plataforma de desarrollo que surgió tras la alianza de varias empresas de telefonía móvil (Nokia, Sony Ericsson, Samsung, etc) con el objetivo de poder competir con Palm o Windows Mobile que, en la actualidad, es propiedad de Nokia [22].

A partir de Febrero de 2010, Symbian OS pasó a convertirse en Symbian Foundation, haciendo así que toda la plataforma pasase a ser "open source" [24]. El lenguaje base de programación es C++ aunque también acepta la programación en Java. La portabilidad

entre aplicaciones creadas con versiones anteriores de la plataforma está mal gestionada, siendo incluso necesario recompilar el código para la versión que vayamos a utilizar.

Estas aplicaciones pueden distribuirse mediante mecanismos OTA [30] como a través de la Ovi Store [23]. Dicha tienda posee al rededor de 28.000 aplicaciones disponibles así como una media de 1,7 millones de descargas por día [9].

Los terminales Nokia son los poseedores de esta plataforma por excelencia, sin embargo, otras marcas como LG o Samsung también utilizan este sistema operativo móvil.

2.2.3. PalmOS

Plataforma de desarrollo nacida en 1996 propiedad de la empresa Palm utilizada, sobre todo, para crear aplicaciones para PDAs [20].

El lenguaje más recomendado para la programación en esta plataforma es C, aunque también soporta Java o Visual Basic. Todas las aplicaciones desarrolladas para Palm que cumplan las especificaciones correspondientes pueden funcionar en cualquier versión anterior de la plataforma, por lo que la portabilidad dentro de este sistema es alta siempre y cuando se cumplan dichas especificaciones. Debido a su antigüedad, tiene una gran penetración en el mercado y un gran número de aplicaciones. La instalación de aplicaciones se realiza a través del PC.

Los terminales por excelencia asociados a este tipo de plataforma son las PDAs, aunque también puede aparecer en teléfonos móviles. Actualmente esta plataforma de desarrollo está evolucionando hacia webOS, la cual aporta mejoras en cuanto a adaptación a los nuevos terminales mediante soporte a pantallas táctiles, multitarea, así como la creación de una tienda online de aplicaciones llamada App Catalog [21]. Dicha tienda es una de las más jóvenes (creada el 6 de junio de 2009) y contiene unas 1.500 aplicaciones [9].

2.2.4. Blackberry

Es la plataforma de desarrollo creada por RIM ("Reserach In Motion") [6]. En 1999 comercializó su primer producto en el mercado y, además de ofrecer servicios típicos de los "smartphones" (agenda, calendario, calculadora, etc) introducía, como característica principal, la posibilidad de enviar y recibir correo electrónico a través de Internet. Esto iba acompañado (en la mayoría de los terminales de la plataforma) de un teclado "qwerty", lo que hacía que escribir emails fuera rápido y sencillo. Haciendo

así a esta plataforma popular en poco tiempo [52].

Enfocada siempre al ámbito de los negocios, permite también la navegación web y la descarga de archivos convirtiendo así a esta plataforma en una buena herramienta de trabajo. Las interfaces de usuario están creadas de forma que el terminal pueda ser manejado a través del teclado, o, a través de su "trackpad" central. En la actualidad dispositivos táctiles utilizan también esta plataforma, por ejemplo, la Blackberry Storm o híbridos con pantalla táctil y teclado como la Blackberry Magnum.

Esta plataforma también cuenta con una tienda de aplicaciones online: La Blackberry App World [7] que cuenta con 11.000 aplicaciones disponibles aproximadamente [9]. La instalación de éstas es muy sencilla y se puede realizar a través del terminal o mediante su previa descarga en un PC.

2.2.5. Android

Es la plataforma de desarrollo creada por Google y la Open Handset Alliance [18] para tratar de competir con las plataformas de desarrollo de Apple y Microsoft [1].

Su historia es bastante corta y comienza con el lanzamiento de su versión 1.0 en 2008. Desde entonces, no ha parado de evolucionar teniendo en la actualidad la versión 2.2 en el mercado y una futura versión 3.0 enfocada al mercado de las "Tablets". Bajo una licencia de código abierto trabaja sobre un núcleo basado en Linux. Su SDK (Software Development Kit) incluye un conjunto de APIs que permiten el acceso a todos los recursos disponibles del terminal, como Bluetooth, 3G, Wifi, cámara, GPS, acelerómetro, etc. También permite utilizar gráficos optimizados ya sean 2D o 3D basados en OpenGL. Además de las APIs previamente citadas, el SDK también incluye un emulador del dispositivo, herramientas de depuración, perfiles de memoria y rendimiento, así como un plugin para el entorno de desarrollo Eclipse. Todo ello se puede obtener de forma gratuita a través de descarga vía web [29].

Cuenta también con una tienda online en donde comercializar las aplicaciones: Android Market [2] que en la actualidad, cuenta con más de 85.000 aplicaciones publicadas [9].

Un gran número de terminales han aparecido desde que el G1 (primer móvil que trabajó bajo esta plataforma) entró a formar parte del mercado siendo así una plataforma de gran apuesta para los fabricantes de dispositivos móviles.

2.2.6. Windows Mobile

Es la plataforma de desarrollo de Microsoft diseñada para trabajar sobre dispositivos móviles nacida en el año 2003 [26].

Su núcleo se basa en el sistema operativo Windows CE y cuenta con una serie de funciones básicas utilizando la API de Microsoft Windows. Para crear aplicaciones se pueden utilizar una amplia gama de leguajes: Visual C++, Visual Basic .NET, C# dentro del entorno de desarrollo Visual Studio [30]. En la última versión de la plataforma se incluyó una tienda online en la que publicar las aplicaciones [27], pero que en la actualidad, no cuenta todavía con el volumen de aplicaciones que tienen el resto de plataformas (aproximadamente 1.000 aplicaciones [9]).

En cuanto a lo que a terminales se refiere, cuenta con más presencia en el mercado de las PDAs que en el de la telefonía móvil. Sin embargo, su gran objetivo es penetrar mucho más en este mercado con su nueva versión de la plataforma: Windows Phone 7.

2.2.7. iOS

Antes llamado iPhone OS, es la plataforma de desarrollo de Apple [3]. Creada en un principio para iPhone, también la usan otros dispositivos como iPod Touch o iPad.

Su desarrollo comienza con la aparición del iPhone en 2007, y desde entonces, al igual que el terminal al que acompaña, ha sido actualizada y renovada para ofrecer nuevas prestaciones. El lenguaje de desarrollo de esta plataforma es Objective C [17], y todo lo necesario para crear aplicaciones viene incluido en el SDK (Compilador, depurador, emulador del dispositivo, etc). La portabilidad entre las aplicaciones es alta y los desarrolladores solo deben de tener en cuenta las especificaciones detalladas en las guías de programación para crear aplicaciones que funcionen en cualquier dispositivo de Apple. Aunque si se desea aprovechar al máximo las nuevas capacidades de los últimos dispositivos se debe sacrificar funcionalidad en los modelos anteriores.

Cuenta con la mayor tienda online de todas las plataformas presentadas, la App Store [5], con más de 300.000 aplicaciones publicadas, así como con el mayor volumen de ventas [9].

El dispositivo por excelencia de esta plataforma es el iPhone, que, actualmente, cuenta con 4 versiones diferentes en el mercado. Pero, como se ha comentado anteriormente, también es la plataforma de otros dispositivos como iPod Touch y iPad.

2.3. Generación de interfaces de forma automática

La generación de interfaces de forma automática, como hemos dicho antes, es necesaria para la representación de nuestro Espacio Inteligente. No es concebible la idea de que los objetos y las personas cambien de lugar en el espacio y que nuestra representación no refleje dicho cambio.

Para definir dichas Interfaces de Usuario (UIs, "User Interfaces") se suele utilizar Lenguajes de Descripción de Interfaces de Usuario (UIDL, "User Interface Description Language") para poder automatizar el proceso de creación y volverlo así más estándar y sencillo. A continuación se presentarán una serie de sistemas que hacen uso de UIDLs para observar el tipo de interfaces que son capaces de generar:

2.3.1. Interfaz creada a partir de MIMIC

MIMIC [48] es un lenguaje que utiliza un sistema de creación de interfaces centrado en tres componentes: el modelo de la plataforma, el de representación y el de la tarea. Un ejemplo de aplicación que utiliza este UIDL es MANNA ("'The Map ANNOtation Assistant") [35]. Es una aplicación hipotética que permite añadir anotaciones a mapas geográficos que puede ejecutarse en diferentes plataformas. A continuación vemos en la Figura 2.1 las interfaces generadas para PDA y teléfono móvil mediante MIMIC.

Figura 2.1: Interfaz generada por MIMIC en PDA y teléfono móvil.

A la hora de adaptar la interfaz se tienen en cuenta las capacidades del terminal: resolución, profundidad de color, mecanismo de entrada (teclado querty, numérico, pantalla táctil...), etc.

2.3.2. Interfaz creada a partir de UIML

"User Interface Markup Language" genera UIs multiplataforma basado en XML [33] para diferentes dispositivos y aplicaciones. Un ejemplo de utilización de este lenguaje se aprecia en la creación de una aplicación genérica [31]. Como se aprecia en la Figura 2.2, gracias a UIML se consigue representar la misma interfaz para diferentes plataformas.

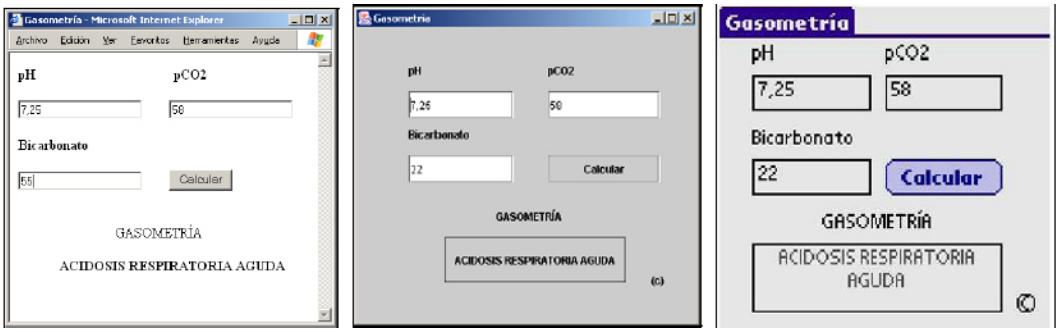


Figura 2.2: Interfaz generada con UIML adaptada a HTML, Java y J2ME.

2.3.3. Interfaz creada a partir de XAML

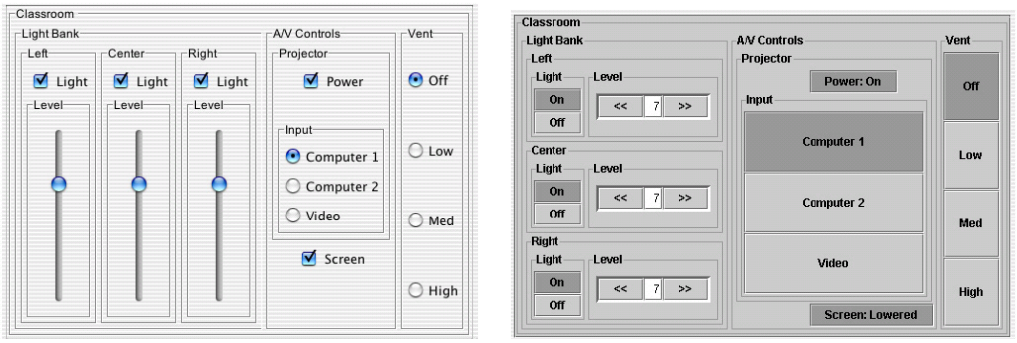
"eXtensible Application Markup Language" [28], creado por Microsoft, es un lenguaje basado en XML que define objetos y sus componentes. Trata de definir UI para la "Windows Presentation Foundation" (WPF) independizando la generación de la interfaz del código de la aplicación. CARUSO ("Context-sensitive ARchitecture for Unified Supervision and cOntrol") [42] es un ejemplo de arquitectura que utiliza este lenguaje en un entorno inteligente doméstico. En la Figura 2.3 se puede ver un ejemplo de esta arquitectura generada con XAML que adapta el estado del sistema, pantalla, método de entrada, etc al dispositivo utilizado.

2.3.4. SUPPLE

SUUPLE [37] es una aplicación que genera automáticamente interfaces, independientemente del tipo de dispositivo. SUPPLE utiliza un sistema de optimización teórica para representar la interfaz, a partir de una especificación abstracta de funcionalidad y un modelo de dispositivo. Además de poder utilizar información del modelo de usuario para adaptar las interfaces a las diferentes tareas y modelos de trabajo. En la Figura 2.4 se puede ver un ejemplo de interfaz creada con SUPPLE adaptada en función del

Figura 2.3: Interfaz generada con XAML.

tipo de interacción. En la imagen (a) la interfaz estaría adaptada al uso del ratón, mientras que en la imagen (b) estaría optimizada para ser una interfaz táctil (Se puede ver, entre otras cosas, que el tamaño de los objetos es mayor).



(a) (b)

Figura 2.4: Interfaz generada con SUPPLE para diferentes modos de interacción.

2.3.5. OpenRemote Boss

Es un producto desarrollado por OpenRemote [19] capaz de crear interfaces de control de espacios domóticos e inteligentes para dispositivos portátiles. Soporta una gran variedad de protocolos de comunicación (X10, IR, KNX, TCP/IP, etc) y diferentes tipos de plataformas. A continuación, en la Figura 2.5 podemos apreciar tanto

el entorno de desarrollo como un par de ejemplos de interfaz, para iPhone y Android respectivamente.

Figura 2.5: Ejemplo de entorno de desarrollo e interfaces con OpenRemote Boss.

Además, en la Figura 2.6 podemos ver un ejemplo de interfaz creada con dicho software para iOS. Es una aplicación que sirve para manejar espacios inteligentes y domóticos que funciona a través de EIB/KNX.

(a) (b) (c)

Figura 2.6: Ejemplo de aplicación con OpenRemote Boss.

En la imagen (a) se puede ver como se representa una habitación, en la (b) la interfaz generada para un objeto de esa habitación y en la (c) el monitor que nos da la información de los sensores de la casa.

2.4. Productos de mercado

En el mercado actual ya existen productos que hacen uso de plataformas móviles para el control de Entornos Domóticos y Espacios Inteligentes. En este apartado nos centraremos en el estudio de algunos de ellos.

2.4.1. Multidomo

Multidomo [15] es una empresa que lleva a cabo la instalación de entornos digitales. Estos espacios se pueden controlar y gestionar a través de diferentes dispositivos con conexión a Internet. Además de poder utilizar PCs y ordenadores portátiles para dicho manejo, existe la posibilidad de utilizar terminales móviles. El tipo de plataforma utilizada en los terminales de tipo PDAs es la de Microsoft. En el caso del los teléfonos móviles J2ME e iOS además de Windows Mobile. En la Figura 2.7 vemos los ejemplos de interfaces que proporciona el fabricante. En la imagen aparecen ejemplos de iOS, Windows Mobile y J2ME.

Figura 2.7: Ejemplo de interfaces de Multidomo.

2.4.2. Vantage

Empresa fundada a principios de los noventa que ofrece soluciones residenciales y comerciales para sistemas de control de espacios [25]. Mediante una solución software

llamada Webpoint Lite ofrece la posibilidad de manejar estos espacios mediante dispositivos móviles con conexión a internet. Está disponible para las plataformas iOS, Blackberry, Android, y Windows Mobile. En la Figura 2.8 podemos ver ejemplos de interfaces de Android, Blackberry, iPhone y iPad.

Figura 2.8: Ejemplo de interfaces de Vantage.

2.4.3. WL3

Es una solución software creada por HAI (Home Automation, Inc.) [11] que permite controlar y monitorizar espacios controlados por Windows Home Server a través de dispositivos móviles que tengan un navegador de Internet. Esto hace que se pueda instalar en la mayor parte de las plataformas previamente descritas: Windows Mobile, Blackberry, iOS, Android, etc. En la Figura 2.9 vemos ejemplos de este producto para iPhone y Blackberry.

2.4.4. Control de electrodomésticos

Los ejemplos anteriormente mostrados se encargan de manejar un entorno completo. Sin embargo también existen otras pequeñas soluciones que se encargan de manejar pequeños electrodomésticos por separado. Son productos mucho más enfocados a una solución concreta para un terminal determinado, sin embargo, observar la forma de representación de los elementos del electrodoméstico así como la interacción con éste puede ser muy interesante de cara a que la aplicación a desarrollar también debe manejar este tipo de objetos. Un ejemplo de este tipo de aplicaciones es i Got Control.

Figura 2.9: Ejemplo de interfaces de WL3.

i Got Control [16] es una aplicación para iOS que, mediante la incorporación de un puerto IRDA ("Infrared Data Associatio"), convierte el terminal en un mando remoto universal. La interfaz es capaz de cambiar en función de los dispositivos que el usuario quiera controlar así como el aspecto deseado por el usuario. A continuación, en la Figura 2.10, podemos ver un ejemplo de la interfaz del programa para diferentes tipos de dispositivos (Izquierda una televisión y derecha un dispositivo multimedia) así como un ejemplo de la interfaz de configuración de la aplicación (Figura 2.11):

Figura 2.10: Ejemplos de interfaces de i Got Control.

Figura 2.11: Ejemplos de configuración de interfaces de i Got Control.

2.5. Conclusiones

Tras haber observado las diferentes plataformas móviles que conviven en la actualidad así como diferentes ejemplos de representación de Espacios Inteligentes y creación de interfaces, se finalizará este capítulo con un apartado de conclusiones en donde se defenderá la propuesta elegida.

2.5.1. Plataforma elegida

Aunque desde el comienzo del proyecto se sabía que la plataforma de desarrollo sería iOS, razones no faltan para saber que es una de las más recomendadas para desarrollar.

Descartar plataformas como J2SE, Windows Mobile o PalmOS es muy sencillo ya que las herramientas que proveen para crear interfaces no se adaptan a la idea de representación que queremos tomar: poca configurabilidad, interfaces gráficas muy sencillas, falta de soporte multi-táctil... Windows Mobile o PalmOS son un poco más flexibles que J2SE pero no consiguen llegar al mismo nivel que el resto de competidores.

Por el último motivo nombrado también es fácil de descartar a Blackberry y a Symbian que, aunque son plataformas capaces de soportar interfaces táctiles, solo unos pocos de sus terminales lo utilizan, por lo que la adaptación no está tan integrada como en las plataformas restantes.

La comparación entre Android y iOS es la más igualada. Y decidir para qué plataforma desarrollar se centraría más en aspectos comerciales o de distribución que en aspectos técnicos. Ambas ofrecen un SDK muy completo, APIs para acceder a los

componentes del dispositivo así como soporte para pantallas multitáctiles. La mayoría de los terminales asociados a estas plataformas, como se ha dicho anteriormente, son smartphones con conexión a Internet, gran capacidad de procesamiento y pantallas lo suficientemente grandes para representar nuestro espacios de forma clara y sencilla. Por lo que nos debería hacernos decantar por una u otra son el volumen de ventas de terminales asociados, el volumen de aplicaciones de sus respectivos mercados online o la fragmentación asociada a los dispositivos.

En el momento que se presentó este proyecto (Septiembre 2009) la plataforma más fuerte en ese sentido era iOS. Por este motivo, ésta fue la elegida para el desarrollo del proyecto.

A partir de ahora nos referiremos a iOS como plataforma elegida y a iPhone o iPod como dispositivo de pruebas.

2.5.2. Modelo de representación elegido

A la hora de representar el espacio en nuestro dispositivo no se utilizará ninguno de los sistemas vistos anteriormente de generación de interfaces dinámicas, sino que se desarrollará uno propio diseñado específicamente para nuestro modelo de representación. El principal motivo es que el Lenguaje de Descripción de Interfaces de Usuario utilizado para representar el espacio no se corresponde con ninguno de ellos. Por lo que adaptar el UIDL utilizado a uno de los anteriores para poder utilizarlo sería mucho más costoso y mucho menos flexible en cuanto adaptación a nuestro entorno se refiere.

En cuanto a la forma de representarlo, se optará por un sistema que utilice las capacidades de iOS y que respete, en todo lo posible, la guía de creación de interfaces humanas de Apple [4]. Gracias a esto, nos aseguraremos que nuestra aplicación va a aprovechar al máximo los recursos del dispositivo, que la interfaz estará adaptada a nuestro terminal y que será sencilla de manejar de cara al usuario.

Por último, decir que los ejemplos de producto vistos, han aportado muchas ideas en cuanto al aspecto que podrían adoptar las interfaces generadas. Sin embargo, gracias a la potencia de nuestro modelo de representación, podemos aspirar a una aplicación más ambiciosa que, por ejemplo, represente a las personas que se encuentran dentro del espacio, pueda añadir nuevos tipos objetos al entorno sin la necesidad de modificar el software del dispositivo, o, simplemente, pueda representar entidades con solo añadirlas al servidor.

Capítulo 3

Diseño del sistema

3.1. Introducción

Este proyecto toma como base el entorno de Inteligencia Ambiental AmILab (ver Sección 3.2), por lo que se parte de un gran trabajo previo que se aprovechó y amplió durante la realización del mismo.

El objetivo principal de este trabajo es desarrollar una aplicación capaz de interactuar con este entorno a través de un iPhone. En él ya existían otras formas de interacción desarrolladas: A través de la voz, a través del PC o a través de la web. Complementarlo con el manejo mediante un dispositivo móvil dotaba de un mayor valor añadido y accesibilidad al laboratorio y a los elementos que lo componen.

Para el diseño de este proyecto se han seguido una serie de criterios, tanto a nivel de interpretación y utilización del trabajo previo, como de desarrollo y creación del soporte para la nueva plataforma:

- Se deben aprovechar al máximo los recursos que el entorno posea.
- Para interactuar sobre los dispositivos se hace uso de la capa intermedia desarrollada (Pizarra: Sección 3.3) así como de las librerías provistas.
- Las interfaces se generarán de forma automática a partir de la información contenida en el esquema de la ontología y de las entidades que conforman el entorno.
- La aplicación debe de estar adaptada lo máximo posible al dispositivo en el que se está ejecutando, es decir, al iPhone y aprovechar las ventajas que éste ofrece.
- La aplicación debe de ser capaz de adaptarse al entorno en el que se está utilizando y actuar en función de éste para una mejor experiencia del usuario.

- La aplicación debe adaptarse a las preferencias del usuario, por lo que debe de ser configurable en función de las necesidades de éste.

Una vez conocidos los requisitos del sistema, se debe estudiar qué métodos y herramientas son más adecuadas para satisfacerlos. A lo largo de este capítulo se describirán las bases en las que se asienta este proyecto: El entorno (Sección 3.2), el Trabajo Previo (Sección 3.3), la Arquitectura del sistema (Sección 3.3.1), la Ontología (Sección 3.3.2) y el Sistema de Representación de Interfaces (Sección 3.3.3); así como las fases de diseño del proyecto: Definición de las funciones de la aplicación (Sección 3.4) y la Interfaz (Sección 3.5).

3.2. El entorno:AmILab

En la Escuela Politécnica Superior (EPS) de la Universidad Autónoma de Madrid se encuentra instalado el laboratorio de inteligencia ambiental AmILab. Equipado con una sala de estar que simula un entorno real, cuenta con electrodomésticos, sensores y equipos audiovisuales con la idea de hacerlo lo más similar posible a un hogar cualquiera. En la Figura 3.1 podemos observar dicho entorno así como diferentes componentes del mismo.

Figura 3.1: Laboratorio de Inteligencia Ambiental AmILab.

Gracias a él, se pueden llevar al mundo real todas las ideas desarrolladas sobre entornos inteligentes y, además, sirve como banco de pruebas de las aplicaciones e interfaces que interactúen con el mismo.

Las principales características del entorno son:

- Contiene una capa física que hace posible el control de los dispositivos: El bus domótico EIB ("European Installation Bus"), actualmente denominado KNX [13].
- Una red Ethernet para el flujo de información multimedia.
- La definición del entorno y los elementos que forman parte de él se realiza de forma estándar y sencilla, basándose en su propio lenguaje de descripción de ontologías (Ver Sección 3.3.2).
- La información sobre los dispositivos y el resto de los elementos que componen el espacio sigue un modelo ontológico y está contenida en "Pizarra" [40], un middleware centralizado encargado de encapsular la interacción de las aplicaciones e interfaces a alto nivel para manejar los elementos físicos del entorno de una forma rápida y sencilla.
- La información física de los dispositivos del entorno se encuentra encapsulada en esta capa, Pizarra, de tal modo que se pueden cambiar unos dispositivos por otros de igual funcionalidad sin variar las capas superiores.
- Para obtener o modificar el estado de cualquier elemento del entorno bastará con enviar mensajes estándar de alto nivel a Pizarra.

3.3. Trabajo Previo

Un entorno inteligente puede estar compuesto por elementos físicos y virtuales así como aplicaciones. En AmILab, la comunicación entre los elementos (físicos y virtuales) y las aplicaciones se da mediante una capa intermedia que la facilita: La Pizarra.

La metáfora de Pizarra plantea que todos los intercambios de información se realicen a través de un módulo lógico centralizado, con el que los desarrolladores puedan interactuar para manejar ambas partes de forma sencilla.

La representación de la información relativa al mundo, que es independiente de la fuente que la genera, se divide en dos partes claramente diferenciadas pero estrechamente relacionadas y se representa mediante un modelo ontológico (Sección 3.3.2):

- **Esquema:** Contiene la descripción del mundo en términos de clases (Sección 3.3.2.1), propiedades y capacidades.
- **Repositorio:** Contiene las entidades (Sección 3.3.2.2), es decir, las realizaciones de las clases.

3.3.1. Arquitectura

Al igual que la representación de la información está dividida para una mejor comprensión, el sistema también está separado en diferentes dominios. Gracias a esto, se mejora la escalabilidad del sistema. A su vez, cada dominio está compuesto por una serie de servidores y de drivers, de forma que el sistema está centralizado desde el punto de vista lógico, pero físicamente distribuido.

Mientras que el modelo del esquema es compartido por todos los elementos del dominio, el repositorio puede estar dividido en diferentes partes. Además, en la arquitectura del sistema aparecen módulos adicionales: un servidor de autentificación, un servidor de resolución de nombres y los drivers [39].

Estos últimos son los encargados del acceso a los dispositivos físicos o a los datos de la memoria. En la Figura 3.2 podemos ver representados a los diferentes servidores así como las relaciones entre ellos. A continuación, se describirán todos los servidores que aparecen dicha figura:

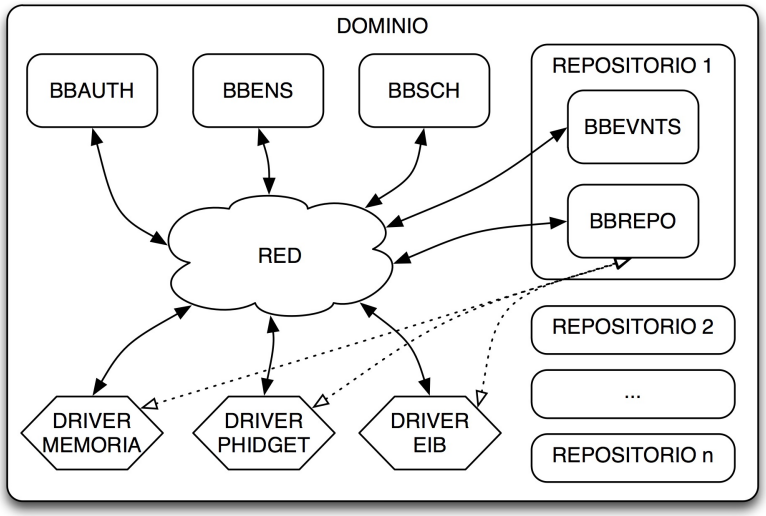


Figura 3.2: Arquitectura del sistema.

- **BBAUTH:** Es el servidor encargado de la autentificación en el sistema. Además, actúa como DNS (Domain Name Solver) que resuelve las direcciones del resto de los servidores que componen el domino.
- **BBENS:** Este es el servidor DNS de las entidades. Cada vez que llega una petición preguntado por cualquier entidad, suministra la información necesaria para conectar con el servidor (repositorio específico) que contiene la entidad.
- **BBSCH:** Este es el modelo del esquema. Es único para todo el sistema y todos los repositorios comparen la información que este servidor provee.
- **REPOSITORIOS:** Son los servidores que contienen a las entidades. Puede haber uno o varios en el sistema. Está a su vez compuesto por dos servidores: El repositorio en sí y otro encargado de notificar los cambios producidos sobre las entidades (o sus capacidades, propiedades o relaciones).
- **DRIVERS:** Son los encargados del acceso a los dispositivos físicos o a los datos de memoria almacenada. Hay tres tipos de drivers implementados:
  - Memoria: Dan acceso al los datos almacenados en memoria.
  - Phidget: Permite el acceso a los sensores y actuadores phidget.
  - EIB: Permite acceder a los elementos que se comunican a través del Bus EIB.

3.3.2. La ontología

Dentro del espacio inteligente encontramos una gran cantidad de información. Toda ella se encuentra ordenada y esquematizada dentro de un modelo ontológico. Gracias a él, la comunicación y el intercambio de información se realiza de forma estándar y sencilla.

Ver cómo se estructura la información en la ontología se hace necesario para poder entender cómo representamos, modificamos y configuramos los elementos del espacio en nuestra interfaz. Como se dijo anteriormente, las herramientas para representar la información se dividen en dos partes claramente diferenciadas:

- Un lenguaje que describe las características y relaciones de los objetos que pueden existir en el entorno, denominado esquema del modelo. Su elemento principal son las clases (representan categorías de la realidad).
- Un lenguaje que describe cada uno de los objetos asociados al entorno. Es decir, un lenguaje que represente a las realizaciones o instancias de las clases, denominadas entidades.

3.3.2.1. Clases

Una clase se corresponde con una categoría de la realidad, es decir, define un conjunto de individuos que comparten un conjunto de características comunes. Toda clase está compuesta de:

- **Nombre:** Cadena alfanumérica única para todo el sistema.
- **Propiedades:** Conjunto de pares nombre-valor que modelan características intrínsecas de la clase.
- **Capacidades:** Representación de las características funcionales asociadas a la clase a la que pertenecen.
- **Relaciones:** Conexiones unidireccionales entre clases.

En la Figura 3.3 está definida la sintaxis utilizada para definir una clase.

```
class [ abstract ] <class name> [ extends <parent class name> ]
[ "{ "
        [ must property prop ]
        [ may property prop ]
        [ must – list ]
        [ may – list ]
        [ cap – list ]
"} " ]

must – list: must properties "{" prop – list "}"
may – list: may properties "{" prop – list "}"
prop – list: name ";" [prop – list ]
cap – list: capability [ cap – list ]
```

Figura 3.3: Sintaxis de la definición de una clase.

Si profundizamos un poco más en esta descripción, vemos diferentes tipos de etiquetas que modelan a la clase: la etiqueta **extends** define si la clase es hija de una clase padre. Gracias a la herencia una clase puede adquirir todas las propiedades y capacidades de otra. Esta herencia es de tipo simple, por lo que una clase hija solo puede tener un padre mientras que una clase padre puede tener múltiples hijas.

La etiqueta **abstract**, si aparece, indica que esa clase es abstracta por lo que no se pueden realizar instancias de la misma.

Las etiquetas **must** y **may** describen la lista de propiedades que contiene la clase. La primera lista representa a las características que deben estar presentes en todas las instancias, mientras que la segunda representa las opcionales.

En la Figura 3.4 aparecen dos ejemplos de definición de clases concretas. En la imagen (a) se ha definido la clase "Light", hija de "Actuator". Tiene una propiedad obligatoria llamada "name" cuyo valor por defecto es "Luz" y una capacidad llamada "isSwitchable". En la imagen (b), se define la clase "DimmableLight", que es hija de la clase anterior ("Light"). Esto quiere decir que hereda sus propiedades y capacidades, y, a su vez, añade nuevas: "isDimmable".

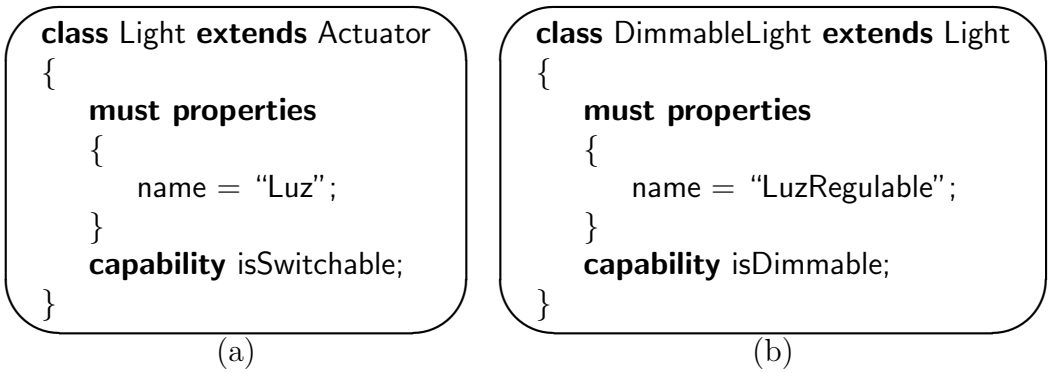


Figura 3.4: Ejemplos de clases extraídos de la ontología.

3.3.2.2. Entidades

Las entidades (o instancias) son las realizaciones particulares de cada clase. Todas las que pertenecen a la misma clase comparten propiedades y características, pero los valores asociados a ellas variarán de una entidad a otra.

La descripción de las entidades está separada de la de las clases. Poseen una sintaxis diferente que podemos apreciar en la Figura 3.5.

Para comprender mejor la sintaxis de las entidades, se continuará el ejemplo anterior con la descripción una entidad de tipo Luz Regulable en la Figura 3.6. En esta figura podemos ver como se asocian los valores correspondientes a las propiedades, capacidades y relaciones descritas en la definición de la clase.

3.3.3. Representación de interfaces

En el Capítulo 2 estudiamos diferentes formas de describir interfaces de usuario (UI) a través de Lenguajes de Descripción de Interfaces de Usuario (UIDLs) y, como

```
entity <class name>:<entity name>@<domain name>
{
    properties
    {
        [ prop – list]
    }
    capabilites
    {
        [ cap – list]
    }
    relations
    {
        [ rel – list]
    }
}

prop – list: prop – def ";" [prop – list];
prop – def: <property name> = "< value >";
cap – list: cap – def ";" [cap – list];
cap – def: <capability name> = "< value >";
rel – list: rel – def ";" [rel – list];
rel – def: <relation name> = "< related entity >";
```

Figura 3.5: Sintaxis para la definción de una entidad.

en el apartado anterior, se hace necesario también hablar de cómo se describen las interfaces dentro de la ontología.

Las interfaces de usuario tratan de mostrar al usuario las propiedades de la entidad a representar, así como las acciones que se puedan realizar sobre ella (capacidades). Es decir, el usuario querrá saber qué entidad está manejando y qué puede hacer con ella.

Para poder crear una interfaz de usuario para cualquier entidad es necesario añadir una serie de propiedades de representación a la clase a la que pertenece. Y, posteriormente, modificar los valores de representación asociados para dar la apariencia deseada a cada elemento del entorno.

Dichas propiedades se aplican tanto a la entidad en sí, como a las propiedades de las capacidades de ésta, ya que la interfaz de usuario se basará en las acciones que la entidad pueda realizar. Las propiedades de representación se encuentran dentro de una estructura llamada "Represetation", como se puede apreciar en la Figura 3.7.

Esta estructura tiene una serie de propiedades obligatorias y otras opcionales:

```
entity Light:lamp1@amilab
{
    properties
    {
        representable = true;
    }
    capabilies
    {
        isDimmable
        {
            level = {
                numericValue = 0;
                maxValue = 100;
                minValue = 0;
                step = 5;
            };
        }
        isSwitchable
        {
            status = {
                binaryValue = 0;
            };
        }
    }
    relations
    {
        locatedAt = Room:labB403Living@amilab;
    }
}
```

Figura 3.6: Ejemplo de entidad extraído de la ontología.

- **iName:** Texto con nombre representativo de la propiedad.
- **iText:** Texto con las diferentes acciones que puede realizar.
- **iEnable:** Indica si el objeto está habilitado o no.

iName e iText son, a su vez, estructuras de datos, ya que la ontología está diseñada para trabajar en diferentes lenguajes, por lo que dentro de cada uno de estas propiedades se encuentran los textos correspondientes a diferentes idiomas (Español, inglés y alemán).

Mientras que la opcional, "graphics", solo las poseen aquellas que tengan una determinada forma de representación. Dentro de esta propiedad, aparece una que debemos

```
struct Representation
{
    must properties
    {
        iName;
        iText;
        iEnable;
    }
    may properties
    {
        graphics;
    }
}
```

Figura 3.7: Definición de la estructura de representación.

comentar obligatoriamente para poder entender las representaciones de la aplicación: "iFaceType".

iFaceType es una propiedad obligatoria que han de poseer todos los elementos que tengan una representación gráfica. Es de tipo enumerado "iFaceWidget" y contiene los nombres de los posibles objetos gráficos que se pueden utilizar. La descripción de éstos se encuentra en la Tabla 3.1

Button	Botón.
CheckBox	Caja de selección.
ColouredLabel	Etiqueta con texto y color de fondo.
List	Lista desplegable de selección.
RadioButton	Conjunto de CheckBox en el que sólo uno podrá ser marcado.
Slider	Barra de desplazamiento (regulador).
Spinner	Contador.
TextBox	Caja contenedora de texto.

Tabla 3.1: Elementos gráficos posibles de "iFaceType".

En función del tipo de propiedad que estemos representando, el iFaceType utilizado deberá de adaptarse a la acción que desea realizar para que la interacción con el usuario sea lo más natural posible. Por lo tanto, a cada propiedad se le asociará un tipo de iFaceType que dará lugar a una UI diferente.

Antes de cerrar este apartado de representación, comentar que, además de la propiedad "graphics" que contiene cada clase, existen una serie de propiedades que siempre debemos representar (siempre que dicha entidad sea representable). Se encuentran en la clase "Root", de la cual heredan todas las demás, y son un par de propiedades que

nos darán información muy útil a la hora de crear la representación: La propiedad iIcon y la propiedad iName.

Ambas son de tipo opcional y contienen la información relativa a la imagen asociada a la entidad (el icono que la representará) y una cadena de texto que contiene el nombre descriptivo de la instancia. Ambas se han agregado directamente a la clase y no a las propiedades de representación (No confundir el iName de las capacidades con este otro que representa a al entidad en sí) ya que es una propiedad cuya información está asociada a la entidad y no a ninguna capacidad.

3.4. Funciones de la aplicación

A la hora de crear una aplicación, saber qué tipo de necesidades se deben cubrir es vital para saber qué funciones se deben realizar. Las necesidades que se desean cubrir desde un terminal móvil son diferentes a las que se pueden dar en otro tipo de dispositivos por lo que acotarlas y definirlas correctamente es necesario para una alta satisfacción por parte del usuario. Las funciones que la aplicación debe realizar son:

- Representar e interactuar con el entorno.
- Adaptarse al entorno.
- Gestionar correctamente los recursos disponibles.
- Adaptarse al usuario.

3.4.1. Representación e interactuación con el entorno

Es la principal necesidad que se debe satisfacer: Mostrar al usuario el entorno y los objetos contenidos en él, así como permitirle una buena interactuación con el espacio. Todo ello, como se ha dicho anteriormente, debe de hacerse a través de una interfaz clara, sencilla y fácil de manejar. Es decir, una interfaz “amigable”.

A la hora de mostrar cualquier objeto, independientemente del tipo que sea, la representación debe crearse de forma automática, por lo que la forma de mostrar las entidades en la interfaz debe de ser lo más genérica posible. Tras analizar los diferentes tipos de entidades existentes, se ha llegado a la conclusión de que no todas se debían representar de la misma forma. Éstas se podían clasificar en tres categorías diferentes:

- **Clases de tipo localización (Location, Building, Floor...):** Las instancias de este tipo de clase representan un lugar y las relaciones que tienen con el resto de entidades son del tipo "contains" (contienen otras entidades). Las entidades que contienen también son de tipo localización, por lo que las representaciones de este tipo de instancias se utilizarían para la navegación entre lugares dentro de la aplicación. Un claro ejemplo de esto es el siguiente: La entidad Edificio B contiene a todas las plantas del edificio B (Figura 3.8).

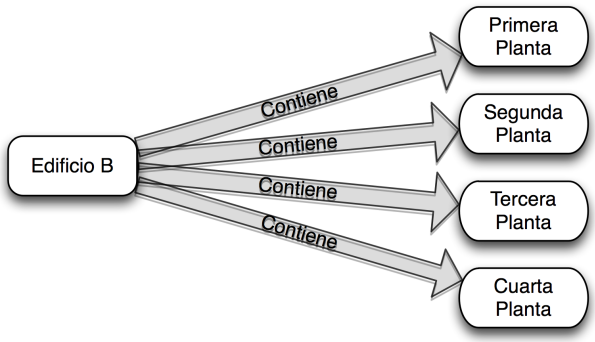


Figura 3.8: Ejemplo de entidad tipo localización.

- **Clases de tipo habitación (Room):** Al igual que las anteriores, las relaciones que tienen este tipo de entidades con las otras son del tipo "contains". Pero en este caso, las entidades contienen solo objetos. Un ejemplo es el siguiente: La entidad Living del Laboratorio contiene: luces, electrodomésticos, personas, etc (Figura 3.9). La representación de este tipo de entidades también se utilizará para navegar dentro de la aplicación, tratando de mostrar a los objetos que contenga de la forma más ordenada posible.

- **Clases de tipo objeto (Light, Door, Person...):** Este tipo de instancias son las que representan entidades con propiedades modificables. Es decir, con las que el usuario iteractúa para modificar valores del entorno. Por ejemplo: las luces, la puerta, las personas, etc.

La forma de representar cada entidad varía en función del tipo que sea y el tipo de interacción asociado a cada una de ellas es diferente: Las entidades de tipo localización o habitación se utilizarán para crear interfaces de navegación mientras que las de tipo objeto crearán interfaces con la información de la entidad y sus capacidades asociadas.

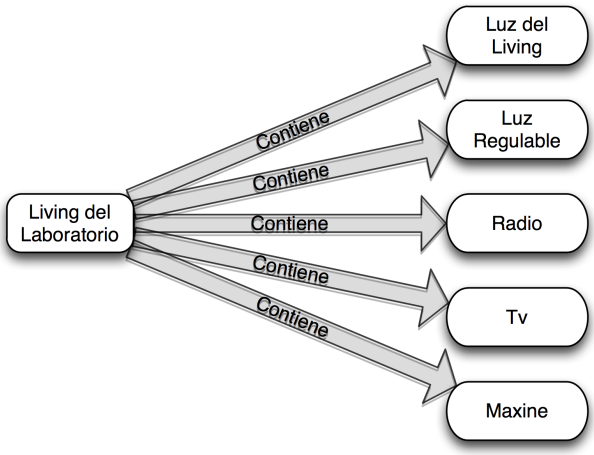


Figura 3.9: Ejemplo de entidad tipo habitación.

3.4.2. Adaptación al entorno

De la misma forma, la aplicación también debe ser capaz de averiguar en qué contexto se encuentra para poder adaptarse a él de forma correcta. Por ello, cada vez que se genera la interfaz de cada entidad se toman los valores de estado real de las propiedades mediante conexiones a pizarra.

Además, en el caso de dispositivos móviles, se desea averiguar qué tipo de red de acceso está disponible para, en función de la que se disponga, tratar de mostrar más o menos información para mejorar los tiempos de espera y el consumo de datos.

3.4.3. Gestión de los recursos disponibles

Además de representar las entidades adaptándose al usuario y al entorno, la aplicación debe de ser capaz de aprovechar al máximo todos los recursos que tiene a su alrededor. Ya sean del entorno (AmILab) como del terminal.

Con respecto al terminal, se deben aprovechar algunas funciones como la búsqueda, la conexión con otras aplicaciones como el email o el teléfono, la capacidad multimedia para reproducir el vídeo procedente de las cámaras del espacio o el uso de la cámara del dispositivo para poder utilizar otras tecnologías como los QRCode[1].

[1]The word “QR Code” is registered trademark of DENSO WAVE INCORPORATED.

3.4.4. Adaptación al usuario

En el Capítulo 2 se habló de la importancia de la adaptación de las interfaces de usuario a la persona que las manejaba: el idioma de la aplicación, la accesibilidad o la manejabilidad. Por lo que la aplicación desarrollada también debe ser capaz de cambiar conforme a las necesidades o preferencias del usuario.

Todos estos cambios se deben poder realizar por usuarios por medio de opciones claras que sean posibles de entender sin un alto conocimiento técnico de la aplicación. Siendo la mejor opción que el propio terminal sea capaz de modificar estos parámetros de forma automática con la información establecida por el usuario en la configuración del dispositivo previa.

3.5. La interfaz

El éxito o el fracaso de una aplicación, muchas veces, viene dada porque el usuario sea capaz de utilizarla y porque la información aparezca de una forma clara y ordenada, es decir, por la interfaz que se utilice y la forma de trabajar con ella.

En este aspecto, Apple ha avanzado muchísimo y, además de dotar de una amplia documentación [4] de cómo deben de ser las interfaces para que cumplan su cometido de la mejor forma posible, también suministra las herramientas necesarias en su SDK para llevarlas a cabo.

La aplicación a desarrollar realiza funciones claramente diferenciadas, por lo que la separación de éstas debe de estar latente en el diseño de la aplicación. Poder navegar de una función a otra debe de ser rápido, sencillo y, a la vez, mantener una distancia suficiente entre ambas para saber que se están realizando acciones diferentes. Por ello, la interfaz del programa se puede dividir en diferentes partes:

- Separación de funciones.
- Menú de entidades.
- Menú de búsqueda.
- Menú de cámaras.
- Menú de QRCodes.
- Configuración.

3.5.1. Separación de funciones

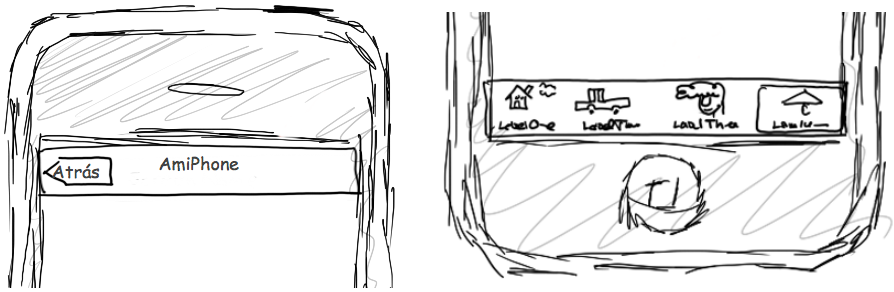
Como se ha comentado anteriormente, separar unas funciones de otras lo suficiente es esencial para que el usuario sepa qué está haciendo en cada momento. Sin embargo, al tratarse de un dispositivo con una pantalla pequeña (en comparación con un PC tradicional) separar una función de otra es bastante complejo.

Cada función se debe representar en diferentes apartados separados por los que se pueda navegar de una forma sencilla. Además, nunca se debe olvidar que tanto la navegación como el manejo de la aplicación se debe realizar a través de los dedos, por lo que todo lo que se diseñe debe de estar adaptado para ese tipo de interacción.

A la hora de estudiar, previo al diseño, qué formas de navegación ofrecía iOS se observó que las que las más comunes eran: A través de "Navigation Bars" o a través de "Tab Bars".

- **Navigation Bars:** Esta barra es la que se encuentra en la parte superior de la pantalla. Suele mostrar el título de la vista y se utiliza para navegar entre menús con cierta jerarquía. En la imagen (a) de la Figura 3.10 vemos un ejemplo de este tipo de barras de navegación.

- **Tab Bars:** Este tipo de barras se encuentran en la parte inferior de la pantalla. Se utilizan para cambiar vistas de la aplicación cuando éstas no suelen tener relación. En la imagen (b) de la Figura 3.10 vemos un ejemplo de este tipo de barras.



(a) (b)

Figura 3.10: Ejemplo de NavBar (a) y Tab Bar (b) de iOS.

Como las funciones de la aplicación no tienen una conexión jerárquica es mejor utilizar el sistema de navegación basado en Tab Bar. Gracias a ella podremos acceder fácilmente a cualquiera de ellas de forma rápida y sencilla.

3.5.2. Menú de entidades

Al igual que hay diferentes tipos de entidades, según la clasificación hecha en la Sección 3.4.1, también tendremos diferentes formas de representarlas. Señalar también que la relación entre entidades, al contrario que la relación entre funciones, es de tipo jerárquica: Las entidades tipo localización contienen entidades de tipo localización o habitación y las de habitación, entidades tipo objeto.

A continuación veremos qué ideas surgieron a la hora de representar cada una de ellas, así como la elección tomada y el motivo.

Representación de entidades tipo localización

Como se ha dicho en la Sección 3.4.1, las entidades de tipo localización son aquellas que contienen entidades del mismo tipo. Dentro de la aplicación se utilizarán para indicar al usuario dónde se encuentra navegando y mostrar hacia qué entidades puede viajar.

Dado que mantienen, como se ha dicho previamente, una clara jerarquización, la representación por medio de Nav Bars es la más adecuada para este tipo de entidades. Gracias a ella, el usuario sabrá en todo momento en qué lugar se encuentra (Título de la barra de navegación), de dónde viene (Botón de atrás) y qué hay dentro de dicha localización (Objetos que aparezcan dentro de la vista). En la Figura 3.11 podemos ver un esquema de esta representación.

Para representar los objetos contenidos dentro de la localización se barajaron diferentes opciones:

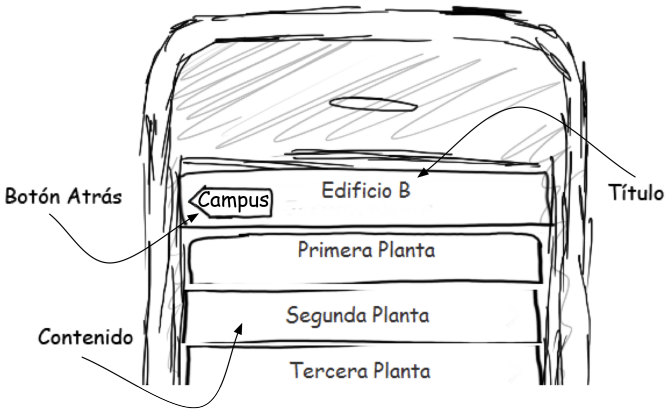
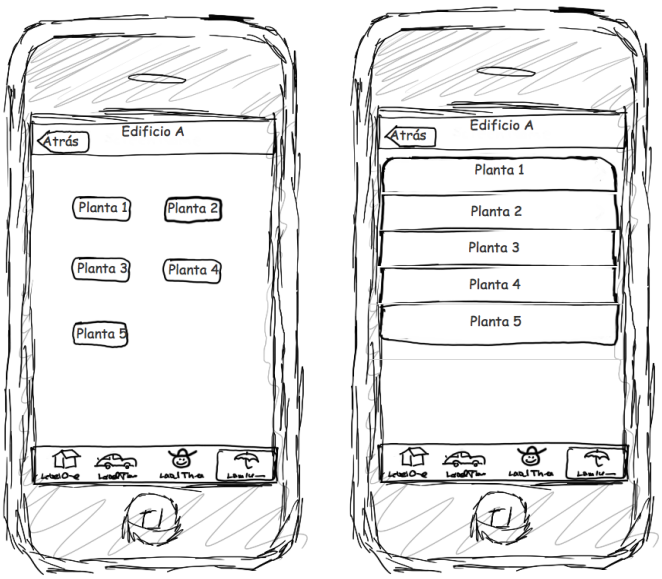


Figura 3.11: Esquema de representación mediante Nav Bar.

- **Navegación a través de botones:** En la barra de navegación aparecería el nombre de la entidad en la que nos encontramos y cada botón que apareciese en la vista se correspondería con las que contiene. Para pasar de un nivel a otro de la jerarquía se tendría que pulsar en los botones correspondientes. Un boceto de este sistema de representación se puede apreciar en la imagen (a) de la Figura 3.12.

- **Navegación a través de tablas:** En este caso, en cada celda de la vista aparecerían las entidades contenidas dentro de la localización correspondiente. Para pasar de un nivel a otro de la jerarquía se pulsaría en la celda deseada y aparecería el menú siguiente. Un ejemplo de este tipo de diseño aparece en la imagen (b) de la Figura 3.12.



(a) (b)

Figura 3.12: Ejemplos de menús con botones (a) y tablas (b).

Tras ver diferentes ejemplos de aplicaciones que utilizaban los diferentes sistemas de navegación se decidió utilizar la segunda porque:

- Las celdas pueden representar más información que los botones (Título, subtítulo e imagen).

- La navegación es más intutiva.

- La información queda representada de una forma más clara y sencilla.

Representación de entidades tipo habitación

Las entidades de tipo habitación contienen directamente entidades de tipo objeto, por lo que son el paso previo a la representación de la entidad a manejar. Debido a eso, son un caso especial dentro de las entidades de tipo localización.

Representar los objetos que aparecen dentro de las habitaciones es el cometido de estas interfaces y, de nuevo, aparece la duda de si representar las entidades objeto de la habitación por medio de botones o por medio de tablas. Resaltar que para este tipo de representación interesa que las entidades estén agrupadas de alguna forma para que sean más sencillas de localizar.

Con los tipos de representación previamente mostrados (tablas o botones) se necesita un nivel más de profundidad en nuestro sistema de navegación:

Habtiación - Clase del objeto - Objeto

Una posible solución que resuelve dicho problema, sin tener que crear un nuevo nivel de navegación, es mediante tablas agrupadas. Una tabla agrupada es similar a una tabla normal salvo que puede agrupar las celdas en función de algún tipo de característica.

Por lo tanto, se podrán representar los objetos de la habitación agrupados por tipo de clase. En la Figura 3.13 podemos ver un ejemplo de cómo es una tabla agrupada. Como se puede apreciar claramente, las entidades objetos aparecen separadas en función de la clase a la que pertenecen y no es necesario cambiar de menú para ello.

Comentar en este apartado que también se ha barajado la posibilidad de representar los objetos de dos formas diferentes, según su clase, o según su localización:

- **Representarlas en función de la clase:** Si se representan las entidades en función del tipo, una vez se llegue a una localización, se puede acceder a los objetos de ésta a través de los tipos de las clases.
- **Representarlas en función de su localización:** Si se representaban las entidades en función de su localización se debía llegar primero a la localización exacta, para, posteriormente, representar los elementos contenidos en ella.

Como se puede ver en la Figura 3.13, la solución mixta obtenida mediante las tablas agrupadas es capaz de aprovechar las ventajas de esas dos formas de clasificación de objetos.

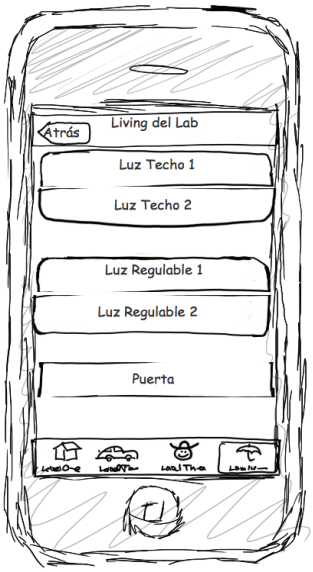


Figura 3.13: Ejemplo de tabla agrupada.

Representación de entidades tipo objeto

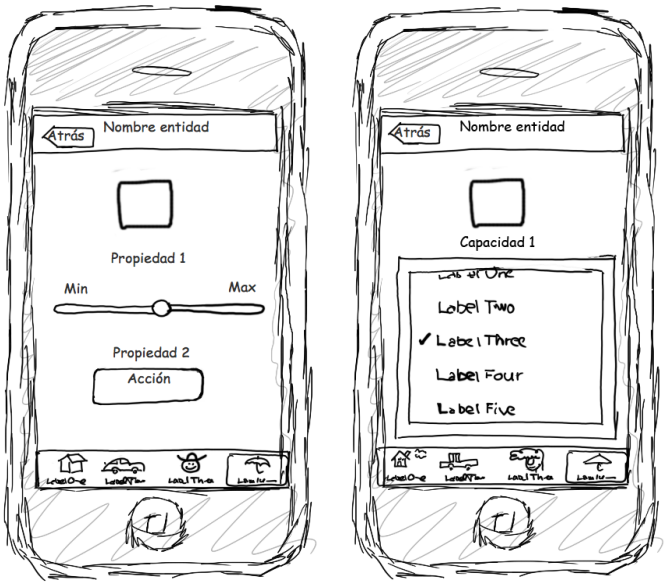
En el último nivel de representación tenemos a las entidades de tipo objeto. Éstas deben mostrar la información de la entidad así como los elementos capaces de modificar las capacidades de éstas. Además, como se ha dicho anteriormente, este menú debe de ser lo suficientemente genérico para que, independientemente del tipo de objeto a mostrar, sea capaz de representarlo.

La información a mostrar se obtendrá de la ontología, tanto de la estructura de representación como de los parámetros de la entidad tratando de colocarse de la mejor manera posible en la pantalla del dispositivo. En la Figura 3.14 se pueden apreciar dos bocetos con ejemplos de la interfaz para este tipo de entidades.

En la imagen (a) estaría representada una entidad con un botón y un slider y en la imagen (b) un entidad con una lista. Como el número de capacidades que pueden poseer las entidades es variable, también es necesario introducir scroll en las interfaces para poder representar a todas ellas.

3.5.3. Menú de búsqueda

Además de poder representar la información de forma jerárquica, tener un menú de búsqueda de ésta es muy útil, dado que hace que encontremos la entidad deseada en un tiempo mucho menor, al no tener que navegar a través de todas las entidades de



(a) (b)

Figura 3.14: Ejemplo: Entidad con Botón y Slider (a) y entidad con Lista (b).

tipos anteriores.

Mediante una barra de búsqueda y un sistema de representación de tablas similar al utilizado en las entidades de tipo localización se consigue una representación sencilla que cumple perfectamente con su cometido. Además, una vez que elijamos el valor deseado de la tabla podemos navegar a través de éstas mediante la interfaz generada para la representación de entidades. En la Figura 3.15 podemos ver un boceto de la interfaz de búsqueda.

3.5.4. Menú de cámaras

Gracias a la disponibilidad de cámaras dentro de AmILab y a la capacidad de reproducción multimedia de iOS se puede mostrar el entorno en tiempo real, dotando a la aplicación de mayor valor añadido.

Mediante un sistema de tablas se representan las cámaras existentes. Tras pulsar sobre la deseada la aplicación muestra una vista con el vídeo captado por ésta. En la Figura 3.16 está dibujado el boceto correspondiente a dicha vista.

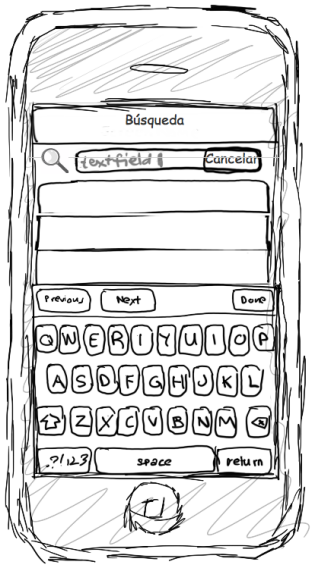
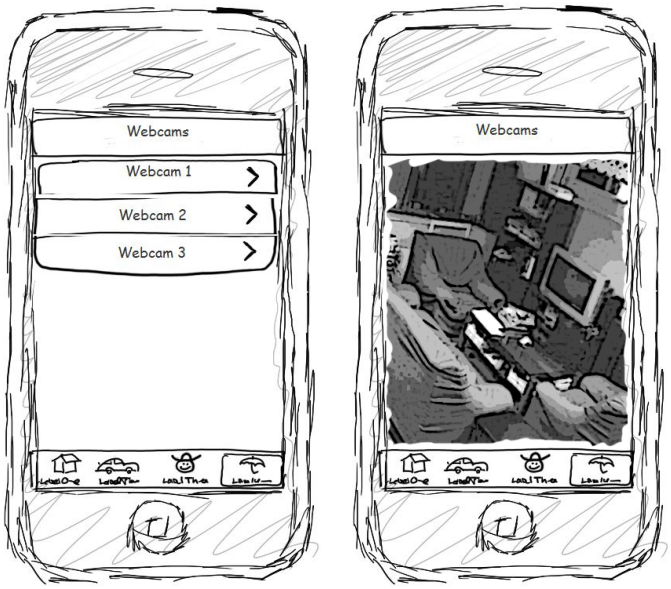


Figura 3.15: Ejemplo de menú de búsqueda.



(a) (b)

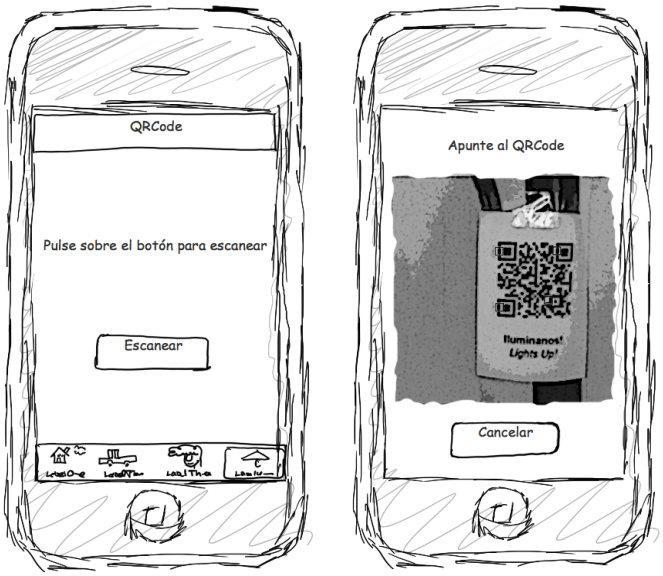
Figura 3.16: Ejemplo de menú de cámaras.

3.5.5. Menú de QRCodes

Dentro del AmILab todos los objetos están etiquetados mediante un sistema de representación basado QRCodes [38]. Aprovechar esta capacidad del entorno para dotar a la aplicación de una nueva forma de búsqueda de las entidades, de forma rápida a través de la cámara del terminal, es una idea muy interesante para mejorar las

prestaciones de la aplicación. Cada entidad está identificada de forma única y estándar.

El funcionamiento sería el siguiente: El usuario debe de escanear con el terminal la etiqueta. Para ello, tan solo tendría que enfocar con la cámara al QRCode y, automáticamente, el terminal se encarga de representar la entidad señalada. Un boceto de este menú se puede ver en la Figura 3.17.



(a) (b)

Figura 3.17: Ejemplo de menú QRCode.

3.5.6. Configuración

En la Sección 3.4 se ha hablado de que la aplicación debe de ser capaz de ser configurada por el usuario. Por lo tanto es necesario tener un menú de configuración dentro de la aplicación. A la hora de aprovechar las capacidades que iOS ofrecía, también se añadió la posibilidad de poder incluir dicho menú dentro de los ajustes del teléfono, por lo que desde un comienzo se debía de implementar según las especificaciones de iOS para aprovechar dicha capacidad. En la Figura 3.18 se puede ver el boceto realizado para dicho menú.

Si se observa la interfaz diseñada utiliza solo interfaces de usuario disponibles dentro del menú ajustes del teléfono para poder compartir dicha representación.

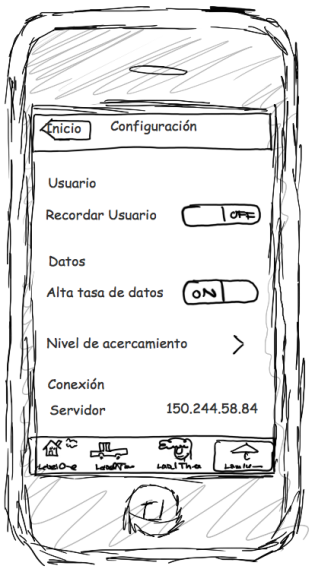


Figura 3.18: Ejemplo de menú de configuración.

Capítulo 4

Implementación

4.1. Introducción

Una vez conocidos los objetivos, requisitos y diseño de la aplicación se pasó a la implementación. En éste capítulo se describen las diferentes fases pasadas a lo largo de ésta. Antes de comenzar a profundizar más en cada uno de los apartados, se realizará una pasada general del trabajo realizado para tener así una visión global de éste.

En primer lugar, se realiza una descripción de la estructura de una aplicación de iOS (Sección 4.2) en la que se describen las partes básicas de un proyecto de esta plataforma. Una vez visto esto, se pasa a la implementación en sí de la aplicación.

La primera parte realizada fue el desarrollo de la librería (Libbb) que permite la comunicación con la capa intermedia, Pizarra, para el lenguaje nativo de iOS: Objective-C (Sección 4.3). Dicha librería estaba previamente implementada en Java pero, al no ser compatible con la plataforma iOS, se tuvo que desarrollar para iPhone.

Una vez realizada la librería, se implementaron una serie de programas de pruebas para observar tanto el funcionamiento de la Libbb, como para profundizar en el nuevo lenguaje de programación (Sección 4.4).

Como última parte de este apartado veremos el desarrollo de la aplicación final: AmiPhone. Se realizará una completa descripción de todas las partes de la aplicación así como los problemas surgidos durante su implementación (Sección 4.5).

4.2. Estructura de una aplicación iOS

Toda aplicación diseñada para iOS se basa en, al menos, una vista con su controlador correspondiente. A partir de ésto, puede complicarse todo lo que el desarrollador desee añadiendo barras de navegación, interfaces de usuario, tablas, etc.

4.2.1. Vistas y controladores de vistas

Las representación de las aplicaciones de iOS se realiza mediante un sistema de vistas. Puede contar únicamente con una sola vista (Por ejemplo la aplicación del Tiempo de Apple) o con un conjunto de éstas (Aplicación del Email de Apple). Cualquier vista se puede dividir en dos partes: la vista en sí y su controlador asociado.

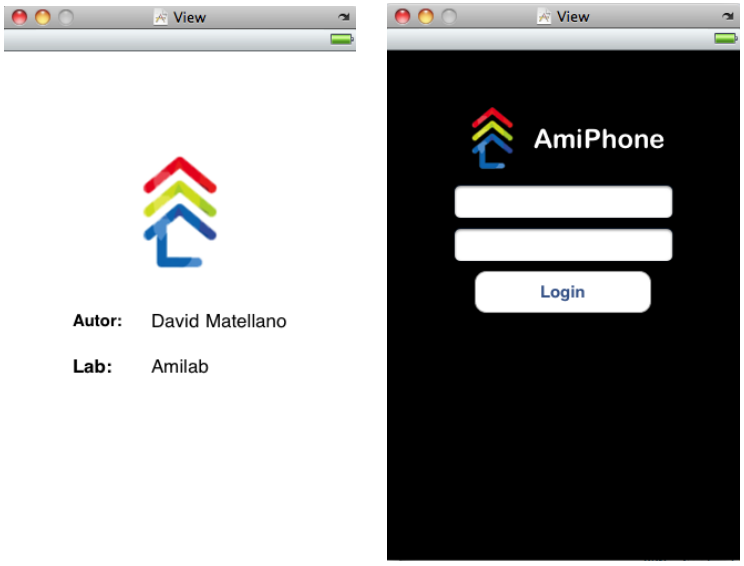
- **Vista:** Es la representación de la interfaz que se quiere construir. Contiene los elementos necesarios para la interactuación con el usuario (Textos, botones, sliders, etc) así como otras vistas. Pueden crearse mediante el editor de interfaces del SDK (Interface Builder) o a través de código.
- **Controlador:** Es el encargado de realizar las acciones pertinentes cuando se interactúa con su vista asociada. Definido mediante código, es quién controla a la vista.

Dos ejemplos de vistas serían las contenidas en la Figura 4.1. En la imagen (a), vemos una de las vistas que aparecen en el inicio de la aplicación. Consta de una imagen y una serie de etiquetas con texto. En la imagen (b) vemos una vista compuesta de dos campos de texto, un botón, una etiqueta y una imagen. Su controlador asociado se encarga de que cuando pulsemos sobre los campos de texto aparezca el teclado en la pantalla, que se llame a la función encargada de realizar la autentificación del usuario al pulsar sobre el botón de login y, además, de que el texto que aparece dentro de los campos esté en un idioma u otro en función de la configuración del usuario.

Este sistema de objeto y controlador del objeto se utiliza con todas las interfaces complejas de iOS: las barras de navegación, las tablas, los selectores, etc.

4.2.2. Barras de navegación

Definidas ya en el Capítulo 3, nos sirven para navegar de una vista a otra dentro de nuestra aplicación. A continuación, hablaremos sobre la jerarquía de las barras y de



(a) (b)

Figura 4.1: Ejemplos de Vistas.

cómo se deben programar los controladores de éstas para que todo funcione correctamente.

En el caso de las Tab Bar, la jerarquía es muy sencilla: todos los elementos que se introduzcan dentro del a barra se representarán dentro de ésta en el orden en el que se añadieron. Esto se realiza a través de su controlador. Para apreciar de forma gráfica la diferencia entre la barra y su controlador, así como con las vistas que posea, aparece el ejemplo de la Figura 4.2.

Vemos como el controlador y la Tab Bar son elementos diferentes. El controlador se encarga de gestionar la Tab Bar y la vista que se tiene que mostrar en ese momento en la pantalla mientras que la Tab Bar es, sencillamente, el elemento de interacción del usuario con imágenes y textos.

Las barras de navegación funcionan de forma diferente. Al poseer una jerarquía en la navegación, dentro del controlador de la barra se indica cuál es la vista más alta de la jerarquía que se debe cargar dentro de ella. Una vez mostrada en la pantalla, dentro de los controladores de las vistas posteriores se indica a qué otras vistas se deben llamar al realizar las acciones pertinentes. Éstas son almacenadas dentro de la pila de la Nav Bar para poder viajar de una a otra a través de los botones de navegación.

En la Figura 4.3 podemos ver una representación similar a la anterior con un sistema de NavBar. De nuevo recalcar la importancia en separar la Nav Bar de la vista que contiene como de su controlador de vista asociado.

Figura 4.2: Representación de una Tab Bar.

Figura 4.3: Representación de una Nav Bar.

4.2.3. Descriptores de vistas

Los ficheros .nib (NeXT Interface Builder) son archivos que contienen las información de las vistas generadas a través del Interface Builder. Escritos en XML almacenan todo lo necesario para poder recrear toda la representación creada: color de la vista, objetos incluidos, posición y tamaño de éstos, así como las conexiones realizadas al controlador de la vista.

Son de carácter opcional ya que toda la información que contienen se puede crear mediante código dentro del controlador, pero ahorran mucho trabajo cuando las interfaces que se desean generar son sencillas y estáticas.

4.3. Libbb

En el Capítulo 3 se presentó la arquitectura del AmILab, así como la capa intermedia utilizada para comunicarse con los elementos del entorno. Gracias a ella el nivel de abstracción a la hora de crear aplicaciones para el desarrollador era mucho mayor.

La Libbb es la librería que implementa el acceso a esta capa intermedia. Se encarga de realizar las llamadas y acciones pertinentes con los distintos servidores haciendo transparente al desarrollador dicha tarea.

La librería original está implementada en Java (al igual que el resto de la arquitectura). Este lenguaje es compatible con un gran número de plataformas y dispositivos, pero no con iOS. Debido a ello, y a la gran utilidad de ésta, se decidió desarrollar una nueva Libbb específica para iOS escrita en Objective-C.

Además, aprovechando que se estaba desarrollando para una plataforma enfocada al entorno de los terminales móviles, la librería original fue modificada para adaptarla mejor a dicha plataforma: adaptación de funciones existentes, eliminación de las no utilizadas y creación de una serie de funciones auxiliares que podían servir de utilidad.

Durante el desarrollo de la librería se prestó mucha atención a que tanto la forma como el contenido de la nueva librería fuera lo más parecida a la original. Respetando siempre el modelo de datos se ha conseguido tener una gran homogeneidad entre ambas librerías, por lo que la única diferencia a la hora de desarrollar en una plataforma u otra con la Libbb será prácticamente de sintaxis.

Muchas de las funciones implementadas no se utilizaron durante el desarrollo del proyecto. Esto se debe a que la implementación de la librería se realizó de forma ex-

haustiva con vistas al futuro esperando aprovecharla al máximo en trabajos posteriores.

4.3.1. Estructura de la Libbb

Las funciones que se pueden realizar con la librería se pueden dividir en diferentes bloques.

- **Gets:** Son las funciones encargadas de recuperar los elementos de los repositorios (entidades, propiedades, capacidades o relaciones), así como los valores asociados a éstas. Esta parte de la librería está implementada en su totalidad debido a la gran utilidad que aporta.
- **Filters:** Aunque este tipo de funciones son de tipo "Gets" ya que obtienen elementos de los repositorios, debido a su importancia caben dentro de otro apartado. Son funciones que obtienen elementos de los repositorios por medio de un filtrado previo dotando a la librería de una mayor potencia. Ejemplos posibles de filtrado pueden ser: obtener una lista de entidades en función de si poseen cierta capacidad, cierta propiedad de una capacidad o una relación concreta.
- **Sets:** Son las funciones encargadas de modificar el entorno cambiando el valor de las propiedades. Al igual que los Gets, se ha implementado la totalidad de este tipo de funciones.
- **Otros:** Comentar también que existen otros tipos de funciones que no se pueden englobar en los grupos anteriores. Son las encargadas de realizar la conexión y desconexión con los servidores así como el trabajo con cachés de memoria.

En la librería original existían más tipos de funciones como los Adds y los Removes. Desde el punto de vista de un terminal móvil no parecía útil poder añadir o eliminar elementos al entorno. Por lo que la gran mayoría de estas funciones no se implementaron y, por eso, tampoco tienen lugar en la estructura de la Libbb de iOS.

4.3.2. Potencia de la librería

Como se ha comentado previamente, la librería se ha implementado para iOS debido a su gran utilidad a la hora de desarrollar aplicaciones que utilicen la capa intermedia Pizarra. Para observar la simplicidad que ésta aporta se analizará, como ejemplo, la reducción de los pasos necesarios para obtener una entidad cualquiera de un repositorio.

En la Figura 4.4 aparecen todos los pasos que realiza la Libbb para devolver una entidad. Este proceso se debería repetir cada vez que se quisiera obtener un elemento de la Pizarra.

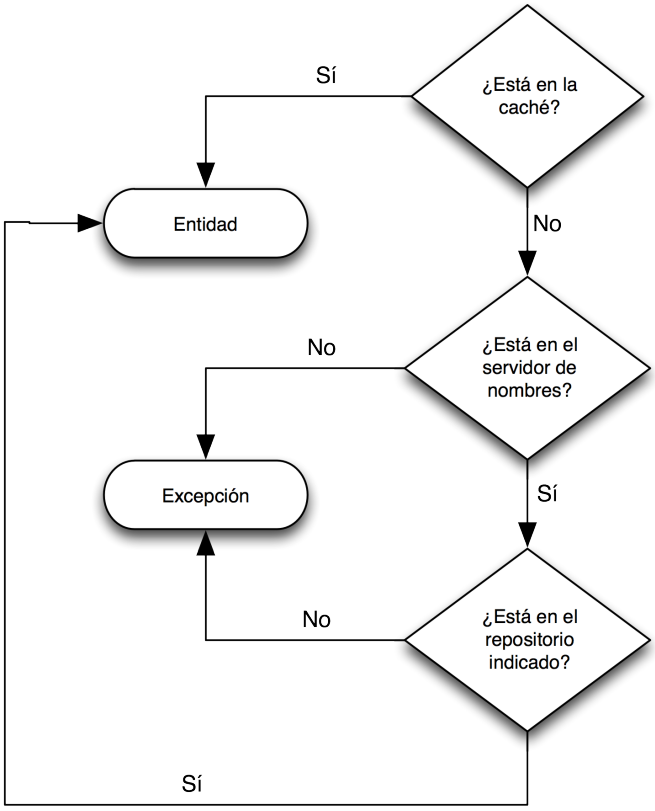


Figura 4.4: Pasos que realiza la Libbb para obtener una entidad.

Sin embargo, en la Figura 4.5 vemos como se simplifica dicha tarea con el uso de una única función de la librería.

Al comparar ambas figuras se aprecia la reducción en el número de pasos necesarios para obtener un simple objeto de la Pizarra. Si a esto se le suman las mayores reducciones que se producen en los filtrados o en la modificación de los parámetros, vemos la rapidez y comodidad que aporta esta librería a la hora de crear aplicaciones.

Para finalizar, y para remarcar más aun la potencia de esta librería, añadir que posee un alto número de funciones diferentes para obtener los elementos de Pizarra de múltiples formas y formatos distintos, así como una gran cantidad de filtros disponibles que harán el trabajo mucho más rápido y sencillo.

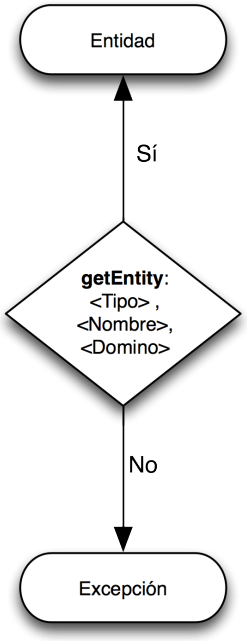


Figura 4.5: Pasos para obtener una entidad con la Libbb.

4.4. Aplicaciones previas

Antes de pasar al desarrollo de la aplicación principal, se desarrollaron una serie de aplicaciones para aprender a manejar correctamente tanto el SDK como para realizar las pruebas pertinentes de la librería y, así, asegurar su correcto funcionamiento.

Esta aplicaciones realizadas hacían uso de los elementos suministrados por el SDK de Apple y de los diferentes elementos del terminal: conexión a la red, cámara, acelerómetro, etc. En este apartado se comentará brevemente un par de aplicaciones destacadas que utilizaban tanto las interfaces de usuario como la Libbb: AmiPrueba y AmiLuces.

4.4.1. AmiPrueba

Fue la primera aplicación que hizo uso de la librería y del entorno con el objetivo de interactuar con una serie de elementos fijos de éste. Su cometido lo conseguía a través de una interfaz estática compuesta de botones y campos de texto. Era capaz de encender y apagar luces, abrir la puerta y mandar mensajes orales a través del avatar del laboratorio.

En la Figura 4.6 se puede observar el aspecto de la aplicación que, aunque cumpliera su cometido, poseía una interfaz estática y poco precisa:

- No mostraba el estado de ninguno de los elementos que podía manejar.
- No situaba a los objetos dentro del espacio por lo que el usuario no podía saber cual de ellos manejaba a no ser que se encontrase en el entorno.
- No tenía ni un aspecto ni una forma cuidados para una correcta interacción con el usuario.

Figura 4.6: AmiPrueba.

Como primer paso era una aplicación correcta pero llegar a la deseada todavía se necesitaban hacer muchos cambios. En posteriores aplicaciones, ya se fueron introduciendo muchos de ellos.

4.4.2. AmiLuces

Otra aplicación que se debe comentar es AmiLuces. Desarrollada para utilizarla como ejemplo durante uno de los seminarios impartidos como introducción a la programación en iOS, llegaba un paso más que la anterior ya que poseía cierta "inteligencia". La aplicación encendía y apagaba una luz del AmILab en función del estado en el

que ésta estuviera. A su vez, la interfaz mostrada al usuario también cambiaba con las acciones realizadas por éste. En la Figura 4.7 se pueden ver unos pantallazos de la aplicación.

(a) (b)

Figura 4.7: Amiluces.

En la imagen (a) vemos como el icono de la bombilla aparece apagado y el texto que aparece en el botón es encender ya que lee previamente el estado en el que se encontraba la bombilla. Una vez encendida, la interfaz cambia y muestra una bombilla encendida con el texto apagar (Imagen (b)).

En esta aplicación también se utilizaron otras características del terminal como el acelerómetro. El dispositivo era capaz de modificar la interfaz en función de la orientación de éste. En la Figura 4.8 vemos un ejemplo de cómo varía la intefaz en función de la posición del dispositivo.

Tanto esta aplicación, como la anterior todavía no hacían uso de la estructura de representación de la ontología, sino que se basaban en textos e imágenes fijas guardadas en la memoria del dispositivo. Fue posteriormente cuando se dotó a la aplicación de dinamismo y adaptación.

Figura 4.8: AmiLuces en posición horizontal.

4.5. AmiPhone

Una vez adquiridos los conocimientos necesarios sobre la plataforma y sobre la ontología se comenzó con el desarrollo de la aplicación final: AmiPhone. Siguiendo siempre los requisitos de diseño se realizaron las tareas pertinentes para que ésta cumpliera los objetivos para los que fue diseñada de la mejor manera posible. A continuación se hablará de las diferentes partes del desarrollo de la aplicación.

4.5.1. Interfaces de Usuario

Hasta ahora, cuando hablábamos de interfaces de usuario nos referíamos a menús o vistas con las que el usuario interactúa. A partir de este momento, dotaremos de un nuevo significado al término interfaz de usuario. Además del significado anterior, también utilizaremos este término para referirnos a todos los elementos suministrados por el SDK del Apple para interactuar con el dispositivo (UIButtons, UISliders, UIPickerView, UILabel, etc).

En el Capítulo 3 se definían los tipos de iFaceType que existían dentro de la estructura de representación. A continuación en la (Tabla 4.1) vemos las relaciones que se hacen entre éstas y las interfaces utilizadas para representarlas en el terminal.

Cada vez que el usuario quiera cambiar el estado de algún objeto de la habitación debía interactuar con una de ellas en el dispositivo. Pero estas interfaces suministradas no eran suficientes para conseguir el funcionamiento deseado de la aplicación.

Una vez pulsada, el programa llamaba a la función que la interfaz tenía asociada. El problema era que dicha función solo podía poseer como parámetro a la interfaz en sí, cuando, además de ésta, se necesitaba más información como la entidad, la propiedad

Pizarra	iOS
Button	UIButton
CheckBox	UISwitch
Label	UILabel
List	UIPicker
RadioButton	UIPicker
Slider	UISlider
Spinner	UIPicker
TextBox	UITextField

Tabla 4.1: Relaciones entre los "iFaceType" de la ontología y los UI de iPhone.

y la capacidad sobre las que se había generado el evento. Por lo que hubo que crear interfaces de usuario propias, que heredan de las suministradas por el SDK. En la Tabla 4.2 se pueden ver la lista de variables que se añadieron a cada interfaz de usuario.

Variables	
Entidad	Entidad a la que representa.
Propiedad	Propiedad a la que representa.
Capacidad	Capacidad a la que representa.
Enable	Dice si la interfaz está activada o no.
Representable	Dice si la interfaz es visible o no.

Tabla 4.2: Variables introducidas en las interfaces de usuario.

A continuación, se muestran todas las interfaces nuevas creadas para la aplicación:

- **Button:** Objeto que hereda de UIButton que, como su nombre indica, es un botón. Con ellos se realizan acciones binarias como encender y apagar (Figura 4.9).

Figura 4.9: Button.

- **Label:** Objeto que hereda de UILabel y sirve para mostrar etiquetas dentro de las vistas (Figura 4.10).
- **SelectableList:** Objeto que hereda de UIPickerView. Se utiliza para mostrar listas de opciones dentro de la aplicación como, por ejemplo, los canales de la televisión o de la radio (Figura 4.11).

Figura 4.10: Label.

Figura 4.11: SelectableList.

- **Slider:** Objeto que hereda de UISlider. Son barras desplazables que permiten elegir entre un valor mínimo y máximo con cierto paso entre ellos (Figura 4.12).

Figura 4.12: Slider.

- **Switch:** Objeto que hereda de UISwitch que permite elecciones binarias. Son similares a un CheckBox (Figura 4.13).

Figura 4.13: Switch.

- **Text:** Objeto que hereda de UITextField que permite introducir campos de texto dentro de él (Figura 4.14).

Figura 4.14: Text.

Como se ha visto en las figuras correspondientes a cada interfaz, la forma de éstas no cambia en cuanto a la igualdad con las del SDK. Sin embargo, la información que poseen aumentó para pasar a contener toda la necesaria para la implementación de la aplicación.

Si comparamos estas nuevas interfaces creadas con las relaciones que se hicieron en la Tabla 4.1, podemos realizar una nueva asociación en la Tabla 4.3.

Pizarra	iOS	UIExtendidas
Button	UIButton	Button
CheckBox	UISwitch	Switch
Label	UILabel	Label
List	UIPicker	SelectableList
RadioButton	UIPicker	SelectableList
Slider	UISlider	Slider
Spinner	UIPicker	SelectableList
TextBox	UITextField	Text

Tabla 4.3: Relaciones entre los "iFaceType" de la ontología, los UI de iPhone y las interfaces creadas.

4.5.2. Arquitectura de la aplicación

Dentro de AmiPhone aparecen diferentes bloques que interaccionan entre sí. Descritos en los puntos anteriores, en la Figura 4.15 se muestra mediante un diagrama la forma que tienen de comunicarse unos con otros.

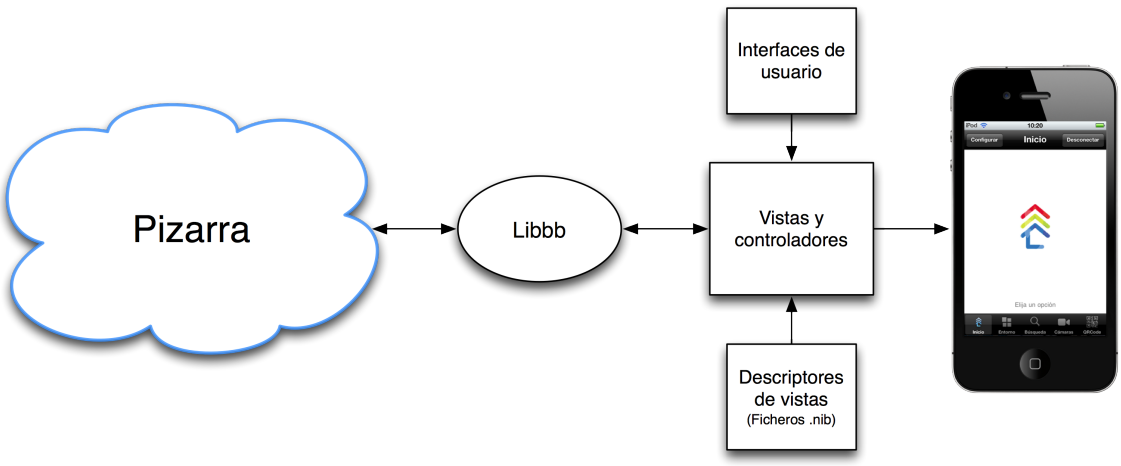


Figura 4.15: Arquitectura de AmiPhone.

En todo momento, las vistas y los controladores de éstas (Sección 4.2.1) son las encargadas de mostrar la información en la pantalla del dispositivo. A su vez, éstas

pueden estar realizadas mediante ficheros descriptores de vistas (Sección 4.2.3) o mediante código. Contienen las interfaces de usuario (Sección 4.5.1) descritas en Pizarra (Sección 3.3) obtenidas a través de la Libbb (Sección 4.3).

4.5.3. Jerarquía de la aplicación

Como se ha dicho en la Sección 4.2.3 la jerarquía de las barras de navegación es muy importante para saber qué y quién debe mostrar la vista en la pantalla del dispositivo.

Debido a la naturaleza de la aplicación definida en el diseño, esta puede cambiar de un menú a otro en cualquier momento y, dentro de cada menú, se navega de una vista a otra de forma jerárquica en la mayoría de los casos. En la Figura 4.16 se puede ver la estructura de la aplicación.

Figura 4.16: Esquema de vistas de AmiPhone.

Disponemos de tres elementos mostrados en todo momento en la pantalla. Una Tab Bar que contiene una Nav Bar que, a su vez, contiene una jerarquía de vistas. Es muy importante saber separar unas de otras ya que trabajan de forma independiente a través de su controlador correspondiente.

El esquema de esta jerarquía se puede ver de una forma gráfica en la Figura 4.17. En ella, se aprecia claramente como la Tab Bar contiene a las barras de navegación de

cada menú que, a su vez, contienen al resto de las vistas de la aplicación.

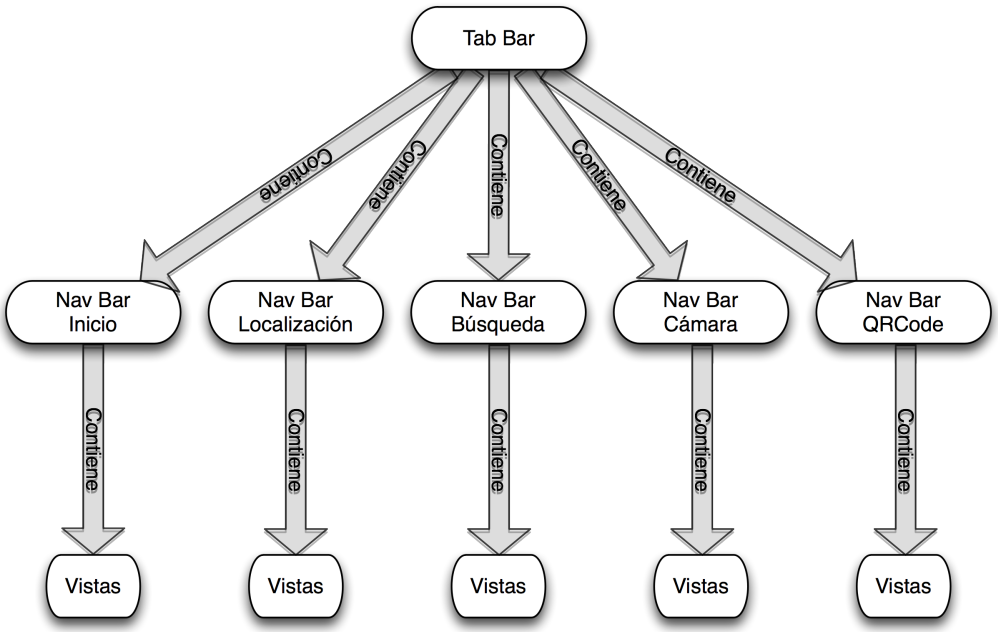


Figura 4.17: Jerarquía de las barras y vistas de AmiPhone.

Una vez comprendida la jerarquía de la aplicación es más sencillo aún imaginarse el sistema de navegación dentro de ella.

4.5.4. Inicio

Una vez vistos los elementos previos necesarios y la jerarquía de la aplicación, pasaremos a describir los diferentes menús de la aplicación según el orden que tienen dentro de la Tab Bar.

En primer lugar tenemos el menú de Inicio, el cual se dividirá en la vista de Login y la de Inicio. Según arranca la aplicación la vista de Login será la primera en aparecer y, una vez autentificado el usuario, se pasaría al propio menú de Inicio. A partir de ese momento, el usuario tiene control sobre la aplicación y puede navegar a cualquier lugar de ésta.

Login

Al arrancar la aplicación se abre la vista de Login (Figura 4.18). Como todas las pantallas de autentificación, pide al usuario los datos necesarios (usuario y contraseña) para que la aplicación sea capaz de identificarlo.

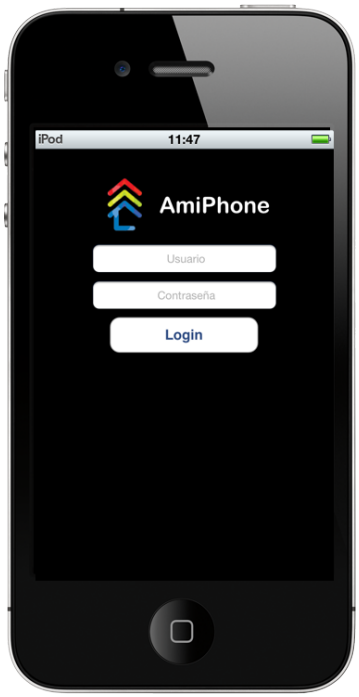


Figura 4.18: Vista de Logueo de AmiPhone.

Lo primero que se hace en esta vista es comprobar si el terminal tiene conexión a Internet. En caso afirmativo se continúa con la ejecución del programa pero, en caso contrario, se comunica al usuario que no existe conexión disponible por medio de una vista de alerta y termina la ejecución del programa.

Tras este paso, se comprueba que el usuario sea correcto por medio de llamadas a Pizarra a través de la Libbb. En el caso de que los datos introducidos sean incorrectos se avisa de nuevo al usuario por medio de otra vista de alerta (Figura 4.19).

Figura 4.19: Vista de alerta de AmiPhone.

Las vistas de alerta (UIAlertView) se utilizan para mostrar mensajes de alerta al

usuario. Estos mensajes están compuestos de un título, un texto descriptivo y una serie de botones. Bloquean el resto de las vistas hasta que el usuario interactúe con ellos y, por lo tanto, son muy útiles a la hora de controlar errores y mostrar mensajes al usuario.

Inicio

Tras pasar la vista de Login, aparece la vista de inicio de la aplicación. Como cada menú, consta de una Nav Bar que contiene las vistas que se desean representar. Esta barra de navegación permite al usuario llegar al menú de Configuración y de Login a través de los botones correspondientes. En la Figura 4.20 podemos ver unas capturas de este menú.

(a) (b)

Figura 4.20: Vista de inicio de AmiPhone.

Para pasar de la vista de la imagen (a) a la de la imagen (b) se debe arrastrar sobre la pantalla del dispositivo. Mediante un Scroll View que contiene a las dos vistas conseguimos este efecto de desplazamiento y, en función de hasta qué punto las arrastremos pondrá una u otra en la pantalla del terminal. La configuración se comentará de forma más detallada posteriormente, en la Sección 4.5.10.

4.5.5. Entorno

El menú de entorno es el encargado de mostrar la información contenida dentro del espacio inteligente. Es, por tanto, la parte principal de la aplicación en donde se verán representados todos los elementos que existan dentro de los repositorios. Mediante un sistema de navegación a través de tablas, podemos profundizar todo lo que deseemos dentro de la estructura jerarquía basada en localizaciones hasta llegar a los objetos deseados.

Esta estructura jerárquica de navegación se consigue gracias a las relaciones que existen entre las entidades. Una relación crea una conexión entre dos entidades. Las conexiones que se buscan para realizar el sistema de navegación son las de tipo "contains" y ´´locatedAt".

- **Contains:** Las entidades de tipo localización y habitación contienen a otras entidades. Gracias a esta relación podemos saber cuáles son para poder representarlas.
- **LocatedAt:** Relación inversa a la anterior que poseen todas las entidades contenidas en algún lugar.

En el capítulo anterior, se habló de los diferentes tipos de entidades según su representación. Ahora se verá la implementación de cada uno de ellos.

Representación de entidades de localización

La navegación entre tipo de entidades se realiza por un sistema Nav Bar y de tablas. Comenzamos desde un nivel inicial (por defecto Campus, pero modificable por el usuario) que muestra todos los elementos de su interior (Se llama a las entidades filtradas por la relación "contains"). Cada celda de la tabla se corresponde con cada uno de estos elementos y, tras pulsar sobre uno de ellos, viajamos a dicho lugar.

Los elementos a los que se pueden llamar pueden ser de tres tipos diferentes: localización, habitación u objeto. Para averiguar dicho tipo y, por lo tanto, crear un tipo de interfaz u otra, se observan las relaciones que tienen las entidades de su interior. Por ejemplo, si las entidades del interior solo poseen la relación "locatedAt" es que son entidades tipo objeto y su interfaz asociada será de tipo objeto.

A la hora de crear este tipo de entidades se realizan diferentes llamadas a Pizarra. En la Figura 4.21 se puede ver un esquema del orden y del tipo de éstas. Tras el último

paso de la figura se obtendría la interfaz que se puede ver en la imagen (b) de la Figura 4.22.

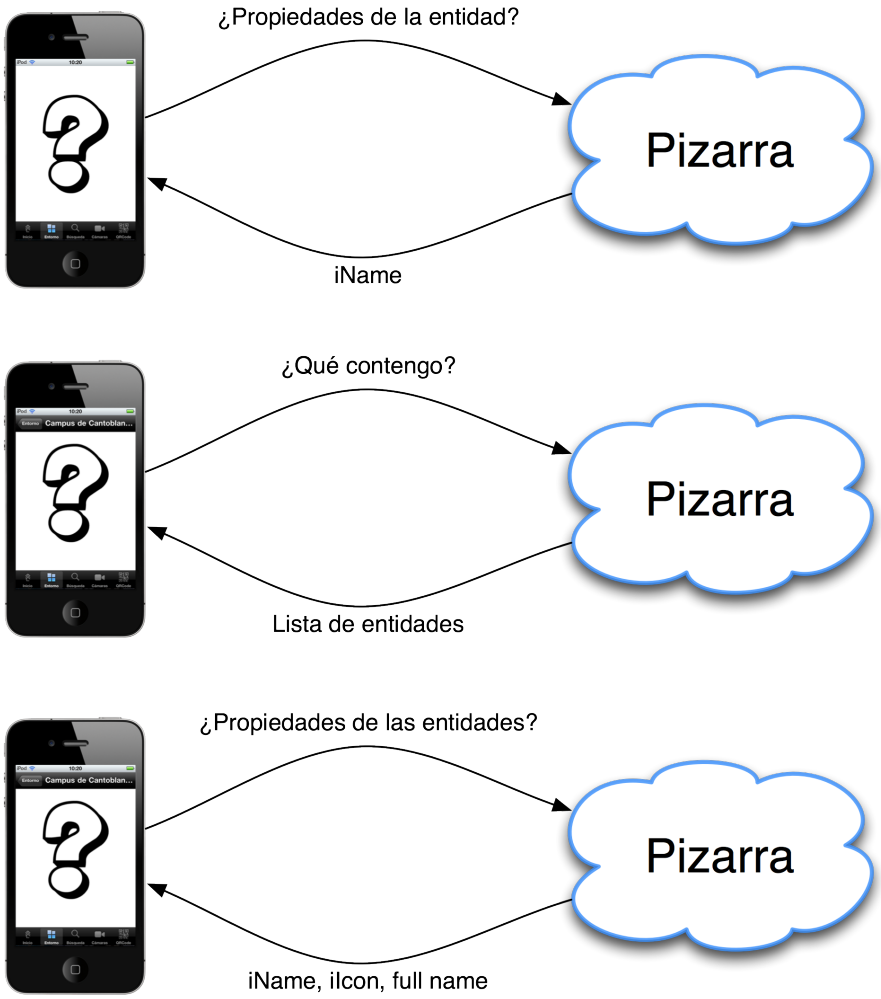


Figura 4.21: Generación de la entidad de una entidad tipo localización.

En la Figura 4.22 se pueden ver diferentes ejemplos de interfaces de localización a distintos niveles. La forma de rellenar las celdas, la barra de navegación y los botones de ésta es siempre la misma. En la imagen (a) vemos que la única entidad que posee el primer nivel de navegación es el campus. La celda que representa a esta entidad contiene la siguiente información:

- Icono representativo de la entidad (iIcon).
- Título con el iName de la entidad en el idioma con el que esté configurado el terminal.

(a) (b) (c)

Figura 4.22: Interfaces de entidades de localización.

- Subtítulo con el "full name" de la entidad. Este parámetro está compuesto por el tipo de la entidad, el nombre de ésta y el domino al que pertenece con la siguiente sintaxis:

<Class name>:<Entity name>@<domain>

Si observamos la imagen (b) vemos como en título de la Nav Bar se corresponde con el iName de la entidad en la que nos encontramos y, finalmente, en la imagen (c) vemos como el botón de la barra para volver al nivel anterior se corresponde con el iName de la entidad de procedencia.

Un detalle a destacar durante la implementación de las interfaces de este tipo de entidades fue que algunas entidades de localización contenían entidades objeto dentro de ellas. Por ejemplo, la entidad Laboratorio de Inteligencia AmILab contiene a dos entidades de tipo habitación: Living del Laboratorio y Office del Laboratorio y, a su vez, contiene la entidad objeto Puerta de entrada. Para una correcta representación, la puerta debía aparecer al mismo nivel que las otras dos entidades por lo que se modificó la representación de objetos para poder representarlos dentro de las entidades de localización también. Esto se puede apreciar en la Figura 4.23.

Figura 4.23: Ejemplo de entidad de localización con entidad objeto.

Representación de entidades de habitación

La representación de este tipo de entidades se consiguió mediante un sistema de tablas agrupadas que mostraban los elementos contenidos en la habitación ordenados por tipo. Sirve como paso intermedio entre las entidades de localización y los objetos del espacio.

La diferencia en la implementación de este tipo de entidades y las anteriores se basa en que una vez recibida la lista de entidades se deben agrupar en función de la clase de éstas. Ejemplos de este tipo de interfaces se pueden ver en la Figura 4.24.

En la Figura 4.24 podemos ver dos ejemplos de interfaces te tipo habitación. En la imagen (a) se puede apreciar el uso de las tablas agrupadas. Si observamos la representación de los diferentes objetos contenidos en el Living, vemos como en función del tipo que tienen se agrupan o no. En la imagen (b) además de ver también la agrupación por tipo también se puede observar qué ocurre cuando algún dato necesitado por la aplicación no se encuentra contenido en los servidores. Si se observa el icono que acompaña a la persona Nacho no existe porque no se encuentra en el servidor, por lo que la aplicación lanza una excepción y no muestra dicha información.

(a) (b)

Figura 4.24: Interfaces de entidades de habitación.

Representación de entidades de objeto

De acuerdo a lo diseñado en el Capítulo 3, la interfaz encargada de representar los objetos de los espacios debía de ser genérica y válida para cualquier tipo de objeto.

A la hora de representar los parámetros de una entidad, se recorrerá la estructura de representación (Sección 3.3.3) perteneciente a cada entidad mostrando la información en la pantalla del dispositivo. Además, también se recuperará la información correspondiente al icono y al nombre representativo de la entidad. En la Figura 4.25 se puede ver mediante un diagrama cómo se realiza dicho proceso.

Tras obtener la información de la estructura de representación para cada capacidad, la aplicación las muestra por pantalla según indica la siguitene enumeración (El resultado final se puede ver en la imagen (c) de la Figura 4.26):

- **iName de la entidad:** Se colocará en la barra de navegación.
- **iIcon:** Será el primer elemento de la vista.
- **Capacidades:** Tras el icono vendrán todas las capacidades que contenga la entidad. Cada una de ellas se dibujará de la siguiente forma:

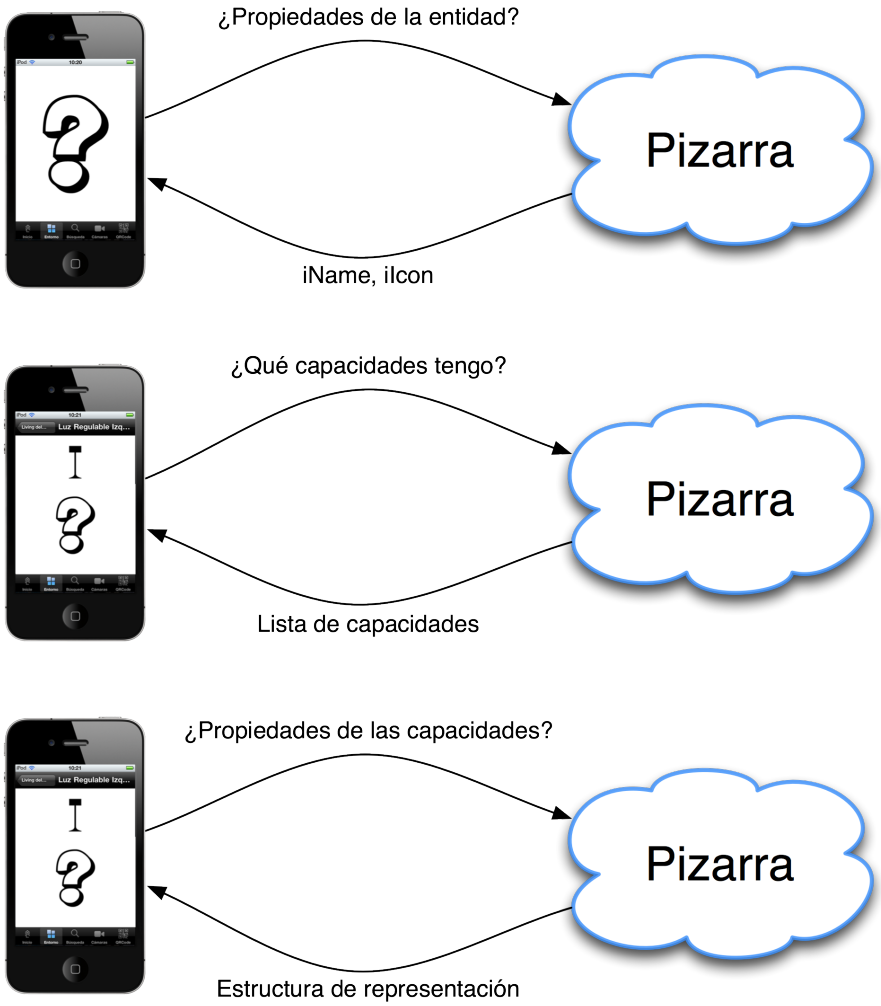
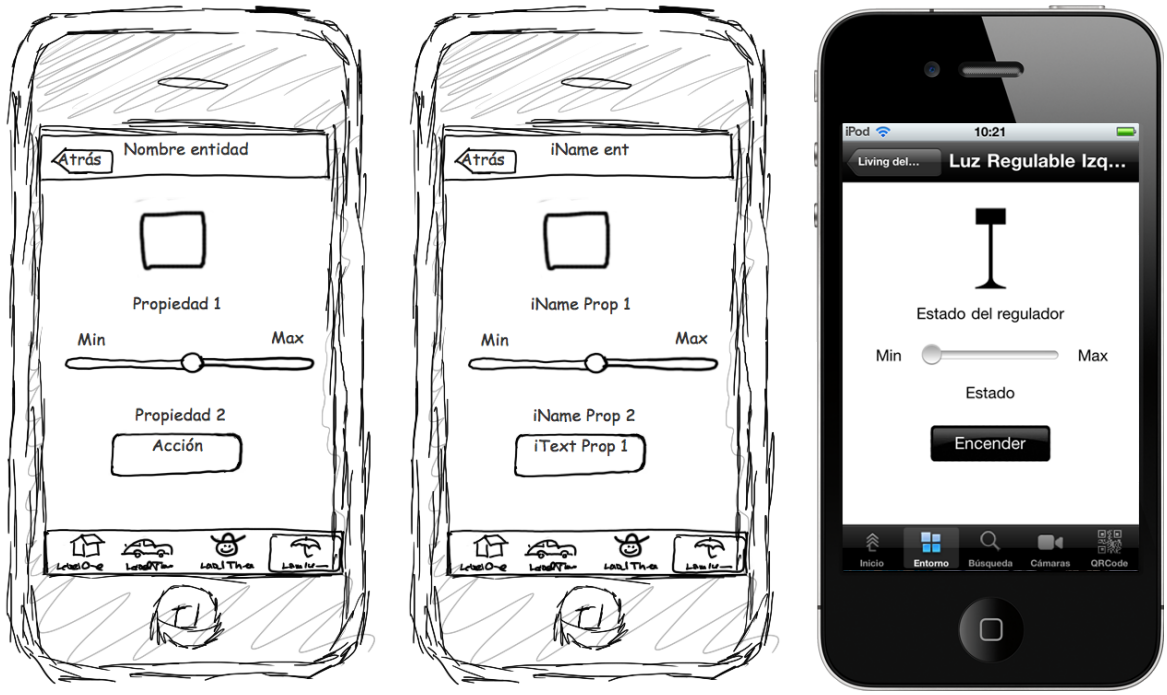


Figura 4.25: Generación de la entidad de una entidad tipo objeto.

- **Propiedad iName de la capacidad:** Se colocará en primer lugar. Encima de la UI correspondiente.
- **Propiedad iText de la capacidad:** En función del tipo de UI se colocará en la UI en si o cercano a ella.
- **Propiedad UI de la capacidad:** Justo debajo de lo anterior se encontrará la UI que se utilizará en función de la iFaceType.

La aplicación debía ser capaz de representar todas las iFaceType (Sección 3.3.3) que nuestro sistema de representación podía enviar. Por lo que a cada una de ellas se le asoció un tipo de interfaz de usuario diferente del sistema iOS. En la Figura 4.26 se puede ver la transición entre el boceto del diseño y la representación real de una entidad de tipo objeto perteneciente a una luz regulable.



(a) (b) (c)

Figura 4.26: Interfaces de entidades genéricas.

La imagen (a) se corresponde con el boceto del diseño. En la imagen (b) sin embargo, se llega a un nivel mayor de profundización ya que se asocia el boceto anterior con los parámetros que se deben obtener de la ontología. Por último, en la imagen (c) vemos como sería la implementación de una entidad concreta que sigue este modelo de representación.

Interacción con las entidades

En las entidades de tipo localización y habitación la interacción es muy sencilla de imaginar: se pulsa sobre la celda deseada y la aplicación muestra la entidad seleccionada. Sin embargo, dentro de las entidades objetos la interacción debe ser detallada un poco más.

Para cada objeto se utiliza la interfaz de usuario asociada según la estructura de representación. Cada una de estas interfaces lleva asociado un método diferente que se encarga de realizar la acción deseada en el entorno a través de la Libbb. Por lo que, en función de qué tipo de interfaz de usuario pulsemos, llamaremos a una u otra. En la Figura 4.27 podemos ver un esquema de cómo sería dicha interacción.

La diferencia entre los métodos de interacción radica en que la acción realizada en

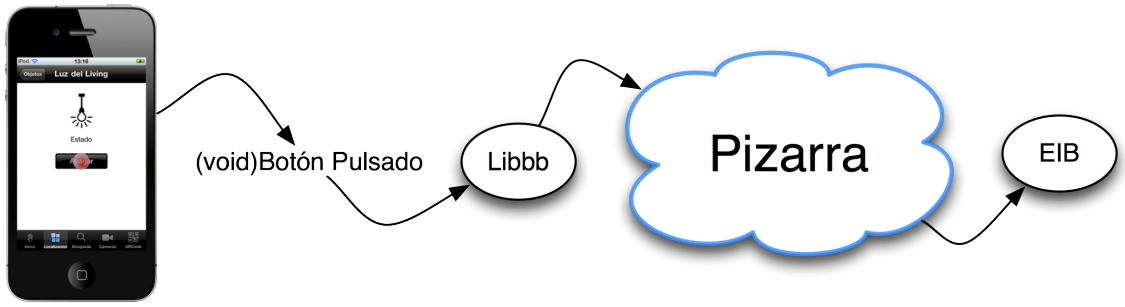


Figura 4.27: Esquema de la interacción con el entorno.

cada uno de ellos es diferente. Por ejemplo, cuando se pulsa sobre un botón se realiza una acción de "Touch up inside" sin embargo, cuando se modifica el valor de un slider se genera otra llamada "Event value changed". Por eso se crearon diferentes funciones en vez de una única que contuviera todo.

4.5.6. Búsqueda

Como se dijo en el Capítulo 3 es necesario una forma de buscar entidades de forma rápida y eficaz dentro de la aplicación. Para ello, iOS ofrece un conjunto de herramientas que lo facilitan: interfaces de usuario (barra de búsqueda) así como un motor de búsqueda integrado en iOS.

El menú de búsqueda permite encontrar resultados en función de tres parámetros diferentes:

- **Nombre de la entidad:** Cada entidad tiene un nombre único por el cual podemos encontrarla.
- **iName de la entidad:** También se podrá encontrar a la entidad por su iName. Es el nombre representativo que aparece en la barra de navegación y cambia en función del idioma.
- **Clase:** Búsqueda que permite filtrar a los elementos según su clase o tipo.

Para ello, cada vez que se pulsa el botón se realizan tres búsquedas correspondientes en una serie de tablas indexadas. Cada una de ellas contiene los diferentes parámetros por los que se debe buscar y se cargan al inicio de la aplicación para obtener un mayor rendimiento durante las búsquedas. Si se obtienen aciertos en las búsquedas, la

aplicación rellena la tabla con las entidades encontradas y, si se pulsa sobre cualquiera de ellas se pasa a generar la interfaz correspondiente.

A la hora de buscar no es necesario poner el texto completo que estemos buscando. Se barajó la posibilidad de mostrar los resultados de la búsqueda según las pulsaciones de las letras pero se descartó debido a su lento funcionamiento.

En la Figura 4.28 podemos ver ejemplos de los tres tipos de búsqueda de la aplicación:

(a) (b) (c)

Figura 4.28: Interfaces del menú de búsqueda.

En la imagen (a) se puede ver una búsqueda en función del nombre de la entidad. Además, se aprecia también como no hace falta poner el texto completo para poder encontrarla. En la imagen (b) se busca por iName de la entidad apareciendo los diferentes aciertos de la búsqueda y, en la (c), aparece el ejemplo de búsqueda por clase.

4.5.7. Cámaras

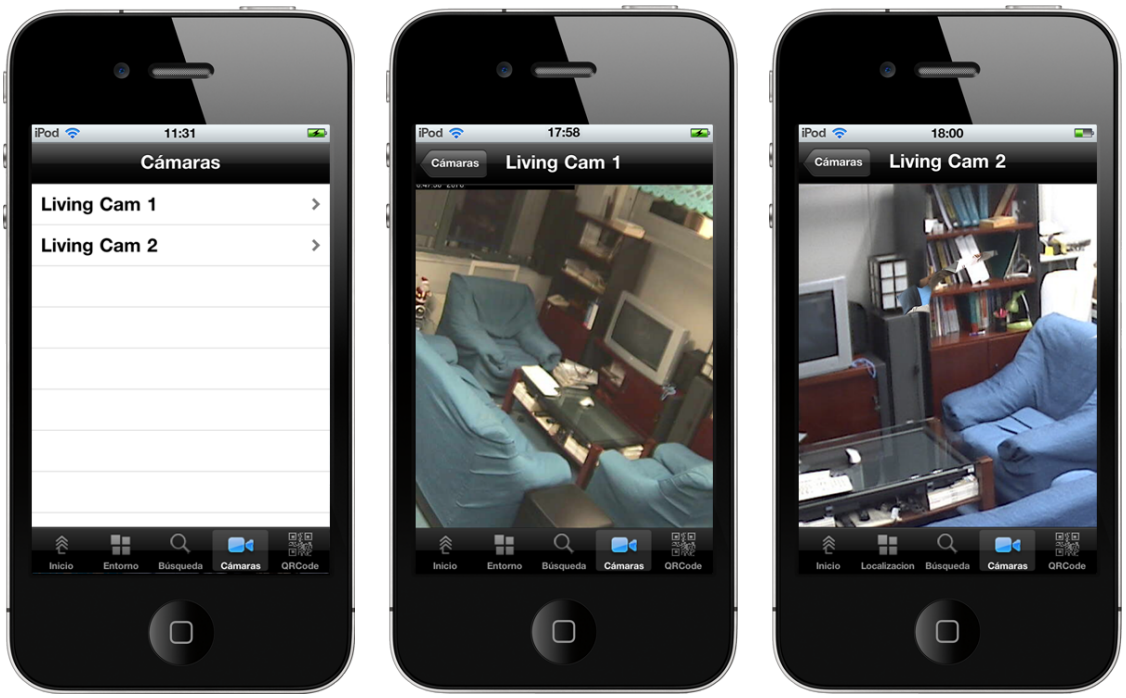
Dentro de AmILab existen una serie de cámaras que permiten la visualización del entorno en todo momento. Utilizarlas para poder observar el espacio desde cualquier lugar se convertía en un complemento ideal para la aplicación.

El sistema de cámaras no está incluido dentro de Pizarra, por lo que introducirlas dentro del sistema general de representación u obtener el vídeo a través de ésta era imposible. Con vistas a crear un sistema de representación que se pudiera ampliar de una forma sencilla se buscó una solución escalable que permitiese la modificación de los datos a través de ficheros plist.

Los ficheros plist ("Property list format") [8] son archivos XML que contienen información organizada mediante pares clave-valor utilizados en la plataforma de Apple. Se guarda el nombre de la cámara junto a su dirección ip para, posteriormente, representar y utilizar esos valores para realizar conexiones a éstas.

A la hora de representarlas, almacenamos todas las cámaras disponibles dentro de un sistema de navegación mediante Nav Bar y tablas. Una vez elegida la cámara se abre una nueva vista con un visor web que reproduce el contenido de la página, es decir, el vídeo de la cámara.

En la Figura 4.29 se puede ver tanto la representación de la tabla (imagen (a)) que contiene a las cámaras así como una imagen de la cámara en funcionamiento (imágenes (b) y (c)).



(a) (b) (c)

Figura 4.29: Interfaces del menú de cámaras.

Por último añadir que dentro del vídeo de la cámara se pueden realizar acciones como zoom y desplazamiento a través de gestos con los dedos.

4.5.8. QRCodes

Como se he dicho en en Capítulo 3 el entorno está marcado con una serie de etiquetas con QRCodes. Cada entidad objeto lleva una asociada y, gracias a ellas, se puede crear una forma de representación muy rápida siempre que se esté dentro del espacio. Bastaría con escanear la etiqueta y el sistema generaría la interfaz correspondiente.

A la hora de integrar esta función en el terminal se utilizó la librería ZXing [14]. Es una librería de procesado de códigos de barras implementada en Java que tiene soporte para diferentes tipos de plataformas. Entre ellas, iOS.

Tras incluirla en el proyecto se añadieron las funciones necesarias en el menú de QRCode para que, al pulsar sobre el botón de escanear, el programa utilizase tanto la cámara como las librerías para escanear los códigos. En la Figura 4.30 se pueden ver capturas de este menú:

(a) (b)

Figura 4.30: Interfaces del menú de QRCodes.

Cuando pulsamos sobre el botón de QRCodes de la Tab Bar aparece el menú de la imagen (a) que nos indica que pulsemos sobre el botón para escanear los códigos. Una vez pulsado, aparece el visor del escáner (Imagen (b)) en donde debemos colocar la etiqueta a escanear.

Una vez leída, las funciones devuelven el texto contenido dentro del QRCode, es

decir, el "full name" de la entidad asociada. Gracias a él, generamos la interfaz correspondiente de igual manera que en el menú de entorno.

4.5.9. Idiomas

Otro de los requisitos de la aplicación era que se adaptara a las necesidades del usuario. La lengua en la que ésta aparece debe ser capaz de cambiar y, además, de forma automática en función de los parámetros del terminal. Para que Apple apruebe aplicaciones multi-língües deben de cumplir una serie de requisitos descrios en su guía de traducción de aplicaciones [10]. Todos ellos fueron seguidos durante el desarrollo de la aplicación y los principales son los siguientes:

- Las vistas, menús e interfaces deben de estar configuradas para poder variar sus textos en función del idioma.
- Los textos estáticos deben estar traducidos.
- Los iconos y dibujos deben adaptarse a las culturas a las que se traduce la aplicación.
- Los sonidos de la aplicación deben de estar traducidos y adaptarse a la cultura.

Para la traducción de los textos, iOS cuenta con una serie de funciones que permiten modificar los textos de las interfaces siempre y cuando se creen una serie de ficheros de traducción dentro del proyecto. Estos ficheros contienen pares clave-valor con las traducciones de todas las palabras necesarias. Una vez creados, dentro de la aplicación había que sustituir todos los textos existentes por referencias a estos ficheros para que, en función del idioma del terminal, la aplicación tomase unas cadenas de caracteres u otras.

En cuanto a los valores procedentes de Pizarra, se indica al programa qué valores debe de coger de las estructuras de iName e iText haciendo así que todos los campos de la aplicación queden traducidos. En la Figura 4.31 se puede ver la misma interfaz de la aplicación en diferentes idiomas (Español e inglés).

4.5.10. Menú de ajustes

A través de la barra de navegación del menú de Inicio se llega al menú de ajustes de la aplicación. Al igual que en el caso del idioma, este menú está generado de forma

(a) (b)

Figura 4.31: Ejemplo de interfaces con cambio de idioma.

especial para que se pueda utilizar dentro de la aplicación como a través del menú de "Ajustes" del teléfono.

Creado mediante un paquete de configuración (Settings bundle) se construye dicho menú a través de una serie de ficheros plist en los que definimos el tipo de interfaces que deben aparecer. Los valores de los parámetros de configuración se almacenan automáticamente en la memoria del teléfono y se cargan al comienzo de la aplicación. En la Figura 4.32 podemos ver unas capturas de este menú.

Los parámetros que podemos modificar mediante este menú (Imagen (a)) son los siguientes:

- **Recordar usuario:** Con esta opción, se idica al sistema si debe o no recordar el usuario y la contraseña cada vez que arranque la aplicación. Para ello se utiliza un Switch.
- **Alta tasa de datos:** Con esta opción se indica a la aplicación si debe ahorrar en el consumo de datos. Si se elige el modo de ahorro no se cargan las imágenes reduciendo tanto el número de llamadas a Pizarra como la cantidad de datos a utilizar. De nuevo se utiliza un Switch para configurar dicha opción.
- **Nivel de acercamiento:** Con esta opción el usuario elige el primer nivel de

(a) (b)

Figura 4.32: Configuración de AmiPhone.

representación dentro de la Localización. Para ello se utiliza un sistema de tablas en un nivel (Imagen (b)).

- **Conexión:** Por último, mediante campos de texto se pueden modificar los parámetros de conexión correspondientes a Pizarra.

Capítulo 5

Ejemplos y pruebas realizadas en un entorno real

El objetivo de este capítulo es presentar los resultados de las diferentes pruebas realizadas en un espacio inteligente real: el AmILab.

A lo largo de éste, se verá la representación del entorno (Sección 5.1), los cambios en la aplicación para adaptarse al éste (Sección 5.3), la traducción de la aplicación a los diferentes lenguajes (Sección 5.4), las modificaciones que un usuario avanzado podría realizar en las interfaces (Sección 5.5) y los tipos de terminales en los que se ha probado la aplicación (Sección 5.6).

5.1. Representación del entorno

A lo largo de toda la memoria se han visto capturas de diferentes interfaces. Sin embargo, nunca se ha mostrado cómo se generan éstas a través de una definición de entidades. En esta sección veremos cómo se realiza dicha tarea además de ver diferentes ejemplo de navegación.

5.1.1. Ejemplo reducido de representación

Este ejemplo describe cómo a partir de un archivo de definición de entidades la aplicación crea las interfaces correspondientes. Para ello, se ha creado un nuevo fichero simplificado en el que solo aparecen tres entidades: Una de tipo de habitación y dos de tipo objeto. La definición de este archivo se puede ver en el Anexo C.

Una vez arrancada la aplicación, al pulsar sobre el botón de entorno lo primero que hace es crear la interfaz correspondiente al máximo nivel de la jerarquía, en este caso, la entidad de tipo habitación Living del Laboratorio. La primera vista que se genera es estándar y contiene una tabla con las entidades de dicho nivel. A la hora de representar los datos de la entidad se siguen los pasos definidos en la Sección 4.5.5. Para ello, se obtienen los parámetros del fichero de entidades de la siguiente manera:

- **Título de la barra:** Para este primer nivel se coloca el nombre del menú en el que estamos: Entorno.
- **Contenido de la tabla:** Esta tabla contiene a las entidades del primer nivel de representación de la jerarquía. Por lo que solamente aparece la entidad del Living.
- **Contenido de las celadas:** Las celdas se rellenan con las propiedades de la entidad: iName, iIcon y fullname. Como se aprecia en el fichero, iName es una estructura con los nombres de la entidad en diferentes idiomas, la aplicación, en función del idioma de configuración del teléfono, elige uno u otro. iIcon es una URL donde se encuentra contenida la imagen asociada a la entidad. El programa se encarga de leerla y cargar el contenido en la pantalla.

El resultado obtenido es el que se muestra en la Figura 5.1.

Figura 5.1: Ejemplo de representación sencillo: Localización.

Tras pulsar sobre la celda que contiene a la entidad, se crea una vista que genera una interfaz de una entidad tipo habitación. En ella se representa la información de la manera siguiente:

- **Título de la barra:** En este caso el título de la barra de navegación es el iName de la entidad en la que nos encontramos. A su vez, aparece el botón de retorno a su lado para poder volver al nivel superior.
- **Contenido de la tabla:** En ella, aparecen todas las entidades que están contenidas en el Living por medio de una relación. Si observamos el archivo, en la entidad Living no aparece definida ninguna relación. Sin embargo, las de las luces si que contienen la relación "locatedAt". Esto hace que Pizarra genere la relación opuesta dentro del servidor del repositorio consiguiendo así saber qué entidades están contenidas en el Living.
- **Contenido de las celdas:** Una vez que tenemos las entidades contenidas en el Living, obtenemos las propiedades necesarias para rellenar la celda (iName, iIcon y fullName).

En este caso, el la interfaz generada es la que aparece en la Figura 5.2

Figura 5.2: Ejemplo de representación sencillo: Habitación.

Por último, al pulsar sobre cualquier entidad de esta representación se genera una interfaz para entidades tipo objeto. En este ejemplo tomaremos la entidad Luz del Living.

Lo primero que se obtiene son las propiedades iName e iIcon (si no hay reducción de la tasa de datos) de la entidad. El iName se coloca en el título de la barra y la imagen obtenida de la URL se coloca en la parte superior de la vista.

Después de esto, se obtienen todas las capacidades de la entidad, en este caso, la entidad solo posee la capacidad "isSwitchable". Y, para cada una de ellas, se obtienen todas las propiedades para poder representarla:

- **binaryValue:** Representa el estado de la luz. Gracias a dicho valor sabremos qué texto poner dentro del botón (Encender o apagar).
- **represetation:** Estructura que contiene los parámetros para la representación de la capacidad:
  - **iName:** El nombre de la capacidad se escribe en una etiqueta que se situará sobre la interfaz de usuario utilizada.
  - **iText:** Es el texto que aparece dentro de la interfaz de usuario utilizada. iText, al igual que iName, posee las diferentes traducciones en función del idioma y un valor numérico que nos indica cuántos textos contiene. En el caso del botón solo son dos (Encender o Apagar) pero cuando representemos los canales de la televisión este número será superior. En función del valor de la propiedad binaryValue se utiliza un texto u otro.
  - **graphics:** Por último, aparece la propiedad graphics que contiene el tipo de interfaz que se debe utilizar para representar dicha capacidad. En este caso una de tipo Button.

Por último, dentro de la definición de la entidad en el fichero aparecen las relaciones que esta posee. Como se ha dicho antes, se puede apreciar como aparece la relación con el Living. El resultado tras leer esta entidad es el que aparece en la Figura 5.3.

5.1.2. Ejemplo de navegación completa

El siguiente ejemplo describe la navegación más genérica posible. En él, aparecen entidades de todos tipos (Localización, habitación y objeto) así como todos los niveles dentro de la jerarquía del AmILab.

En la Figura 5.4 se puede apreciar dicho ejemplo. El objetivo de éste es representar todos los pasos posibles para llegar a una entidad de tipo objeto (En este caso la televisión) desde el más alto nivel de representación posible (El nivel de Campus).

Figura 5.3: Ejemplo de representación sencillo: Objeto.

Si observamos las capturas, como se dijo en el Capítulo 4, vemos como los títulos de las barras de navegación, el botón de ésta y el contenido de las tablas cambia en función de la entidad en la que nos encontremos. Además, también se ve la diferencia entre las tablas agregadas de las entidades de tipo habitación y las tablas normales del las entidades de tipo localización.

La entidad objeto elegida para representar es la televisión. En la representación se modelan los parámetros que puede tener cualquier televisión: encendido, volumen y canales. Pero está contenida en el repositorio de memoria y no el de EIB, por lo que los cambios realizados en la interfaz no producirán ningún efecto en el mundo real. Sin embargo, bastaría solo con instalar el driver pertinente y añadir la entidad al repositorio del EIB para que funcionara como el resto de las entidades.

La imagen que se corresponde con la interfaz de la televisión está retocada de forma que pueda representar todos los elementos que contiene. La representación en el terminal se realizaría por medio de un scroll que permite subir y bajar dentro de la vista para poder acceder a todas las interfaces de usuario. Sin embargo, para que se pudiera apreciar correctamente en la memoria se realizaron los cambios pertinentes.

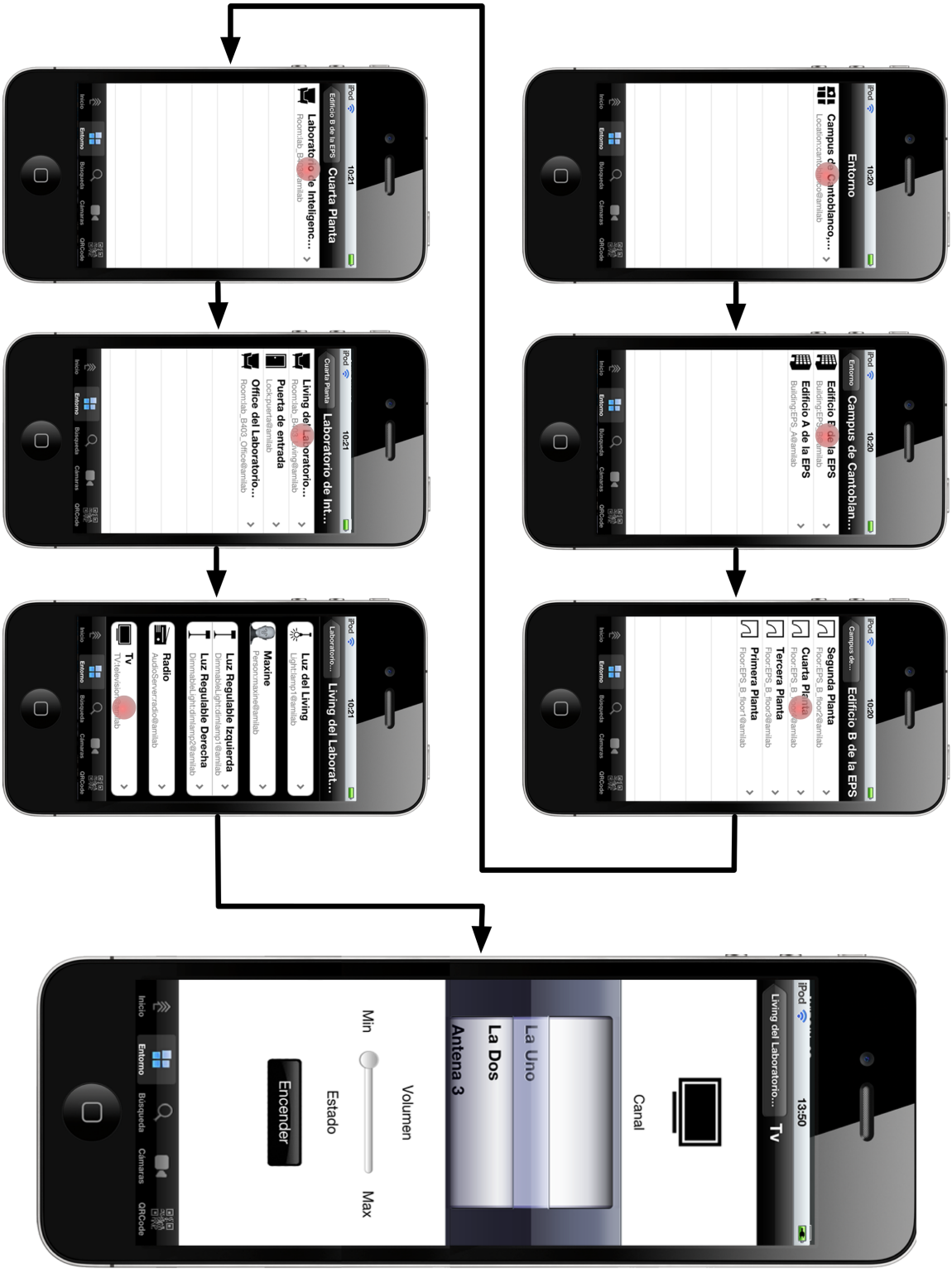


Figura 5.4: Ejemplo de navegación con todos los niveles.

5.1.3. Ejemplo de navegación parcial

En el menú de configuración el usuario puede elegir el nivel inicial de navegación. Con ello, consigue llegar a las entidades de tipo objeto con un mayor o menor número de saltos entre entidades. En función de la opción elegida, el usuario sacrifica parte de la jerarquía para ganar mayor comodidad y velocidad.

Sin embargo, el usuario solo deja de acceder a los niveles superiores al inicial, pudiendo acceder a cualquier entidad (de cualquier tipo) inferior al nivel que ha tomado como comienzo.

En la Figura 5.5 se puede apreciar todo esto. Como se puede observar, el número de saltos a realizar para llegar a la entidad de tipo objeto se ha reducido drasticamente y, además, como se representan todos los elementos disponibles en el nivel inicial de navegación, se puede acceder a cualquier entidad deseada de un nivel inferior a éste.

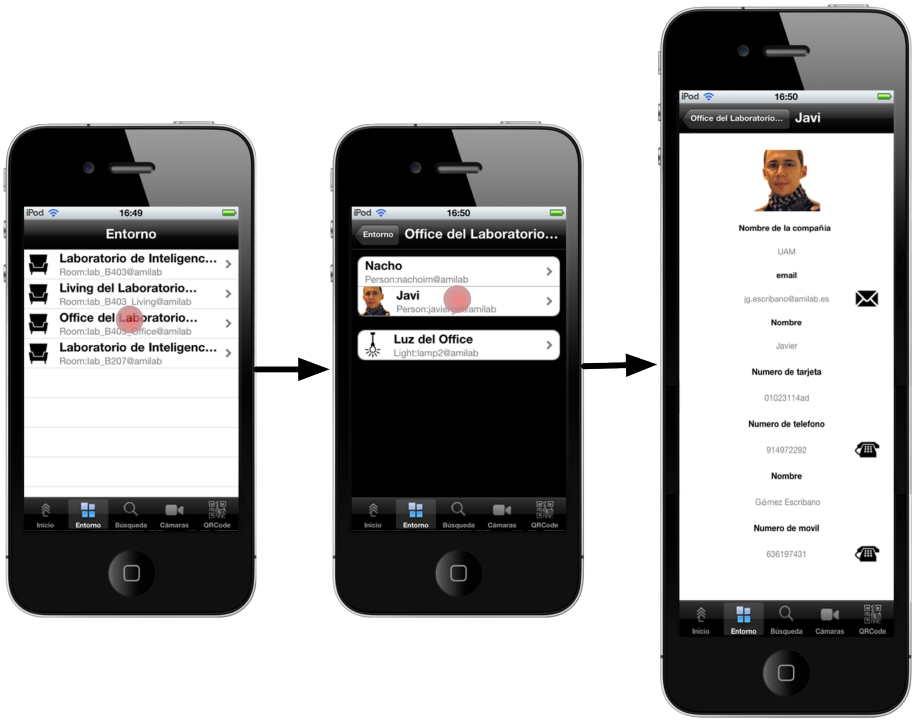


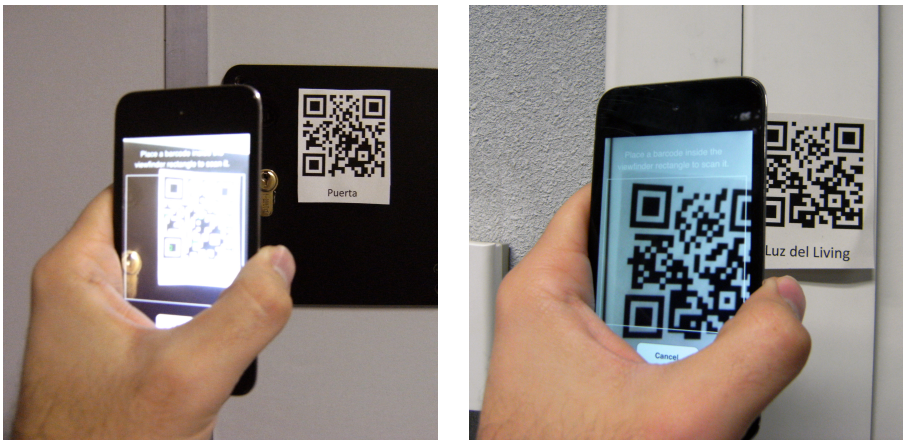
Figura 5.5: Ejemplo de navegación desde nivel de Habitación.

Destacar también en este caso la representación de entidad objeto elegida. Es una entidad de tipo persona. Contiene toda la información asociada a ella y, además, gracias a la conexión con el resto de aplicaciones incluidas en el teléfono podemos abrir la aplicación de mail y de teléfono desde esta interfaz. Con esto aprovechamos las posibilidades que el terminal nos ofrece aumentando las acciones que puede realizar el

usuario con la aplicación.

5.2. Navegación por QRCodes

Descrita en el Capítulo 4, la interacción con el entorno por medio de QRCodes es posible en la aplicación gracias a la librerías de ZXing. En la Figura 5.6 se puede ver un ejemplo del uso de este tipo de tecnología.



(a) (b)

Figura 5.6: Ejemplo de navegación con QRCodes.

Los QRCodes se encuentran situados cerca de la entidad a la que representan. El usuario, debe enfocar con la cámara del dispositivo a dichas etiquetas para que se realice su lectura. Una vez realizada, la aplicación muestra la información de la entidad asociada a la etiqueta escaneada.

5.3. Adaptación al entorno

Para mejorar el retardo dentro de la aplicación, sobre todo cuando se está accediendo a Internet a través de una red 3G, la aplicación es capaz de reducir el número de llamadas a Pizarra así como de disminuir el tráfico producido.

Para ello modificando la opción de “Alta tasa de datos” del menú de configuración, la aplicación cargará o no imágenes en las representaciones. En la Figura 5.7 se aprecia como cambian los distintos tipos de entidad cuando éstas no se cargan. En la imagen (a) se ve un ejemplo de entidad de tipo localización, en la (b) una de tipo habitación y en la (c) una de tipo objeto.

(a) (b) (c)

Figura 5.7: Ejemplo de modo ahorro de datos.

El ahorro conseguido es considerable. En la Figura 4.4 de la Sección 4.3.2 se veía el número de llamadas a los servidores que realizaba una simple llamada a Pizarra. Por lo que, cada vez que se decide no cargar una imagen, además del reducir el tráfico generado por la carga, se suprimen un gran número de llamadas a los servidores. Si esto se aplica a todas las celdas de una entidad de tipo localización hace que se reduzcan en gran manera los tiempos de carga.

Para observar la diferencia de tiempos real se realizó la siguiente prueba: En un iPhone 3GS conectado a una red Wifi se cargaba la interfaz correspondiente a la entidad habitación del Living en modo normal y en modo de ahorro de datos.

La entidad a cargar poseía seis entidades en su interior. Por lo que la diferencia en los tiempos de carga se debía a la carga de estas seis imágenes y al tráfico que se generaba al averiguar cuáles eran. La diferencia de tiempos se pueden apreciar en la Tabla 5.1.

Vemos como el modo de ahorro de datos funciona llegando a reducir el tiempo de espera en prácticamente un 26 % cada vez que se crea la interfaz correspondiente.

Con imágenes	Tiempo (s)
Prueba 1	4,329
Prueba 2	4,4
Prueba 3	4,32
Prueba 4	4,4
Prueba 5	4,406
Media	4,371

Sin imágenes	Tiempo (s)
Prueba 1	3,266
Prueba 2	3,237
Prueba 3	3,243
Prueba 4	3,223
Prueba 5	3,195
Media	3,232

Tabla 5.1: Comparación de tiempo de carga en modo ahorro de datos en redes Wifi.

5.4. Idioma

A la hora de adaptar la aplicación al usuario hacer que ésta cambie en función de su idioma es un factor fundamental.

En función del idioma en el que el usuario tenga configurado el terminal, si éste se encuentra dentro de las traducciones del programa, los textos varían mostrando el idioma configurado por el usuario.

La aplicación está traducida a tres idiomas (español, inglés y alemán) y aumentar el número de éstos es muy sencillo. Para ello, tan solo habría que traducir un fichero con los textos estáticos de la aplicación (Localizable.string) y, posteriormente, traducir los textos de las entidades en los repositorios.

En la Figura 5.8 vemos ejemplos de interfaces en diferentes idiomas. En la imagen (a), vemos el menú de configuración en inglés. Los textos que se encuentran en la Tab Bar y en el menú se corresponden con los traducidos dentro del fichero Localizable.

En la imagen (b) podemos ver la interfaz producida cuando se viaja a una entidad que no contiene nada en su interior. En vez de mostrar un menú vació se muestra una vista de alerta en la que se indica al usuario que la vista a la que quiere acceder está vacía. En este caso se aprecia como está traducida al alemán. Las traducciones de estos textos están también incluidas dentro del fichero Localizable.

Por último, en la imagen (c) vemos una representación de tipo objeto traducida. En este caso señalar que la información mostrada de la entidad, al ser dinámica, viene traducida de Pizarra.

(a) (b) (c)

Figura 5.8: Ejemplo de traducción de idiomas.

5.5. Modificación de las interfaces de usuario

Otro parámetro a modificar dentro de la aplicación son las interfaces de usuario que representan a cada capacidad. Enfocado para un tipo de usuario más avanzado, mediante cambios en las entidades dentro de los ficheros del repositorio, la aplicación soporta ciertas modificaciones.

En la Figura 5.9 podemos ver diferentes formas de representar la capacidad isSwitchable. Para ello se han utilizado tres tipos de interfaces diferentes:

- **Imagen (a):** En ella vemos la interfaz como es por defecto, es decir, la representación mediante un botón.
- **Imagen (b):** En este caso se utiliza un switch para realizar cambios sobre la luz. Este tipo de interfaz también es muy intuitiva a la hora de representar capacidades de este tipo.
- **Imagen (c):** Por último, se representa la capacidad por medio de un SelectableList. Aunque el uso de este tipo de interfaces no es el más intuitivo a la hora de representar este tipo de capacidades, gracias a este ejemplo apreciamos la flexibilidad que posee la aplicación.

(a) (b) (c)

Figura 5.9: Ejemplo de cambio de interfaces.

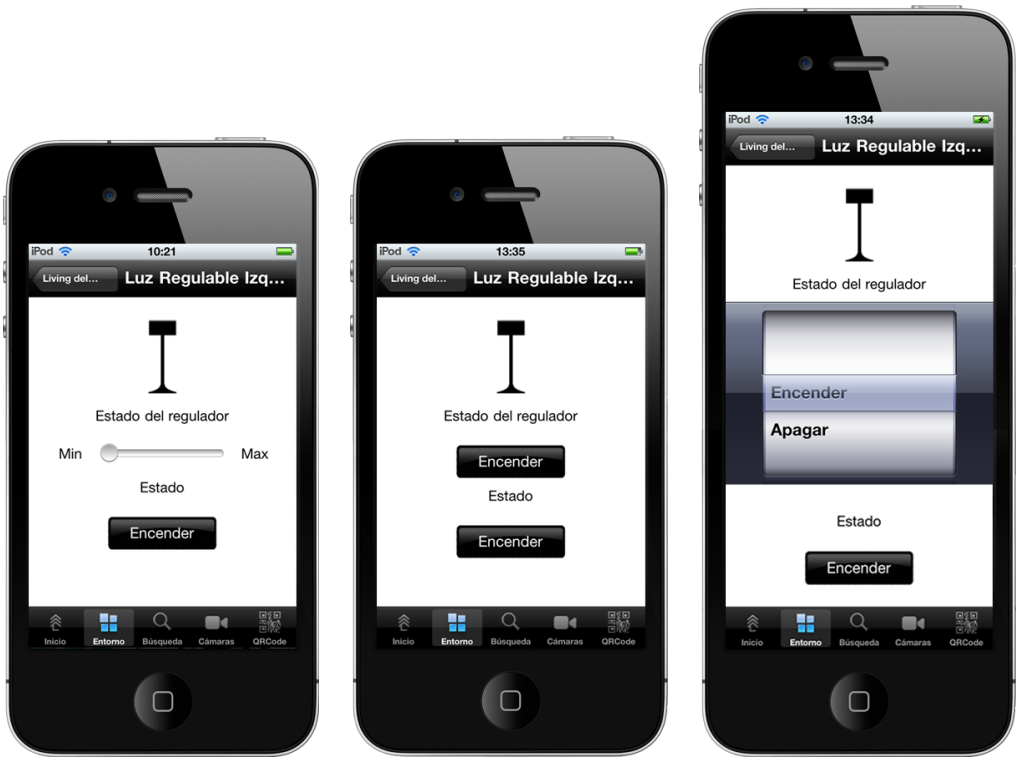
A la hora de hacer este tipo de cambios tiene que ser lógico el tipo de interfaz utilizada, ya que no todas ellas podrán realizar la acción que deberían correctamente. Por ejemplo, en la Figura 5.10 se aprecia la interfaz de la luz regulable con las interfaces modificadas. En vez de poner un slider (Imagen (a)) se cambian por un botón (Imagen (b)) o una lista ((Imagen (c)).

Si bien la aplicación es capaz de soportar dichos cambios, las interfaces no pueden realizar las acciones correctamente. Por lo que, como se ha dicho antes, los cambios se deberán realizar con conocimiento de la acción a realizar y las interfaces disponibles.

Por último, comentar que todos los cambios de las interfaces se pueden realizar en tiempo real. Es decir, la aplicación no necesita ser reiniciada para modificar la interfaz, sino que solo se necesita modificar los valores procedentes de Pizarra para que se aprecien en ésta.

5.6. Agregar o eliminar elementos al entorno

Al igual que en tiempo de ejecución se pueden modificar los tipos de interfaces en la representación de las entidades, también es posible agregar, eliminar y modificar entidades de los repositorios y ver los cambios reflejados en la aplicación en tiempo real.



(a) (b) (c)

Figura 5.10: Ejemplo de cambio de interfaces.

Esta modificación se puede realizar de diferentes maneras:

- **Modificando los ficheros del repositorio:** Modificando los archivos que contienen la definición de las entidades y reiniciando su repositorio asociado.
- **Añadiendo entidades en tiempo de ejecución:** No es necesario reiniciar los repositorios para añadir o eliminar entidades. La Libbb está preparada para conseguir estas funciones y, a través de aplicaciones externas, se pueden modificar las entidades existentes en el entorno. Resaltar que estas aplicaciones pueden utilizar tanto la Libbb original como la implementada en este proyecto.

5.7. Pruebas en diferentes terminales

Además de todas las pruebas previamente realizadas, la aplicación se probó en diferentes versiones de iOS y de terminales. Gracias a la capacidad de configuración del SDK se podía variar el tipo de ejecución y así probarlo en diferentes versiones de la plataforma.

En cuanto a los terminales físicos, en función de las características de cada uno la interacción del usuario cambiaba y algunas de las funciones no eran compatibles.

- **iPod Touch:** Se probó en diferentes generaciones de iPod Touch. Las funciones de teléfono no están disponibles es este tipo de dispositivo ya que carece de éste. Y en cuanto a la representación mediante QRCodes, solo está disponible en el iPod Touch de última generación que es el que incluye cámara al dispositivo. Decir también, que cuanto más moderna era la generación del iPod, mayor era la velocidad a la que la aplicación corría mejorando, por tanto, la interacción con usuario.
- **iPhone:** También se probó la aplicación en el teléfono de Apple. En este dispositivo es donde la aplicación aprovecha al máximo los recursos disponibles debido a que posee tanto cámara como teléfono.
- **iPad:** Por último, se probó en el tablet de Apple. En este caso, la forma de interacción no estaba adaptada correctamente al dispositivo por lo que la navegabilidad conseguida no es la óptima para este tipo de terminal. Sin embargo, la aplicación funcionaba y cumplía su cometido.

Para ver el rendimiento de la aplicación en función del tipo de terminal se realizó la siguiente prueba: Medir los tiempos de carga de la aplicación en diferentes dispositivos.

Durante la carga, la aplicación inicia las variables globales utilizadas y rellena una serie de tablas con las entidades del entorno para acelerar los procesos de búsqueda. Se introdujo un sistema para medir el tiempo transcurrido en este proceso en diferentes situaciones:

- Cuando el terminal accedía a la red mediante una red wifi (Tabla 5.2).
- Cuando el terminal accedía a la red mediante una red 3G (Tabla 5.3).

Vemos como el simulador es el que realiza la tarea más rápidamente debido a que utiliza el procesador del ordenador. Al comparar los dispositivos móviles, vemos como tanto el iPod como el iPad tienen prácticamente los mismos tiempos de carga. Esto se debe a que ambos tienen la nueva generación de procesadores incluidos por Apple dentro de sus dispositivos de última generación. El más lento de todos ellos es el iPhone 3GS que carece de dicho procesador.

Si comparamos estos tiempos con los obtenidos en las redes 3G vemos como estos tiempos aumentan debido a la velocidad de esta red.

Simulador	Tiempo (s)
Prueba 1	1,289
Prueba 2	1,234
Prueba 3	1,18
Prueba 4	1,242
Prueba 5	1,26
Media	1,241

iPod	Tiempo (s)
Prueba 1	3,49
Prueba 2	3,409
Prueba 3	3,39
Prueba 4	3,407
Prueba 5	3,39
Media	3,417

iPhone 3GS	Tiempo (s)
Prueba 1	7,049
Prueba 2	6,977
Prueba 3	6,962
Prueba 4	7,051
Prueba 5	7,983
Media	7,204

iPad	Tiempo (s)
Prueba 1	3,753
Prueba 2	3,75
Prueba 3	4,003
Prueba 4	4,626
Prueba 5	3,649
Media	3,956

Tabla 5.2: Comparación de tiempos de carga en redes Wifi.

iPhone 3GS	Tiempo (s)
Prueba 1	169,225
Prueba 2	100,818
Prueba 3	147,505
Prueba 4	139,614
Prueba 5	111,907
Media	133,813

Tabla 5.3: Comparación de tiempos de carga en redes 3G.

Luego, para concluir este apartado se puede ver como cuanto mayor sea la generación del dispositivo utilizado más rápidamente correrá la aplicación. Además, el ancho de banda disponible también acepta al funcionamiento de ésta por lo que se hará necesario poseer una rápida red de acceso para un correcto funcionamiento.

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

El concepto de entorno está cambiando con la llegada de las nuevas tecnologías. En las nuevas construcciones la dómotica es algo habitual y el paso hacia una nueva generación de edificios inteligentes está, poco a poco, penetrando cada vez más dentro de la sociedad actual. La forma de interactuar con estos espacios inteligentes es una actividad vital que debe de ser clara, sencilla y transparente para el usuario de a pie. Además, debe permitir acceder a la información del entorno desde cualquier lugar, así como la capacidad de interactuar con él.

En este Proyecto Fin de Carrera se ha presentado AmiPhone, que es una aplicación desarrollada para la plataforma iOS que satisface parte de esas necesidades tomando como base el trabajo realizado en el Espacio Inteligente AmILab. Diseñada para que cualquier tipo de usuario, es capaz de cumplir su cometido por medio de una interfaz generada de forma dinámica, clara, sencilla y fácil de manejar.

En el Capítulo 2 se vio como la plataforma de Apple era una de las más poderosas a la hora de crear aplicaciones para dispositivos móviles. Además, se vieron un conjunto de aplicaciones que, si bien aportaron ideas para el diseño de AmiPhone, no conseguían implementar todas las funciones que se podían realizar gracias al potente sistema creado previamente.

La aplicación cumple los requisitos propuestos en el Capítulo 3: representa los elementos del entorno así como su estado, por medio de un sistema jerárquico de navegación que permite una fácil localización de los objetos; aprovecha las cámaras del espacio, como la capacidad multimedia del dispositivo, para mostrar al usuario el lugar en todo momento; busca los objetos de forma rápida y eficaz mediante diferentes tipos de búsqueda, así como por medio de etiquetas con QRCodes que representan a los

elementos del AmILab y permite que el usuario configure una serie de parámetros para una mayor adaptación.

Para realizar todas estas tareas, la aplicación se apoya en una librería cuya implementación ocupó una parte importante del proyecto: La Libbb. El desarrollo de esa librería ha hecho posible que la implementación de esta aplicación haya sido mucho más sencilla, rápida e intuitiva. Y servirá como base de futuros trabajos realizados en Objective-C relacionados con iOS y el laboratorio.

En el Capítulo 5 se vio como las interfaces mostradas al usuario cambian en función de los cambios producidos en el entorno en tiempo de ejecución. Ya sea añadiendo o eliminando componentes del entorno, como modificándolos, la aplicación es capaz de generar una nueva interfaz que se adapte a dichos cambios. Así como las diferencias producidas en la aplicación al ejecutarla en diferentes terminales y redes de acceso.

Comentar también que AmiPhone cumple con las especificaciones dictadas por Apple en cuanto a la generación de interfaces de usuario, la traducción de la aplicación y la creación de un menú de configuración para una mayor integración en el dispositivo. Por lo que, la adaptación conseguida dentro del dispositivo es tal que no hace falta ejecutar la aplicación para poder modificar sus parámetros.

Gracias a los conocimientos adquiridos durante la realización de este trabajo se pudieron impartir una serie de seminarios para introducir a los asistentes a la programación en iOS. Uno para los miembros del Grupo de Herramientas Interactivas Avanzadas (GHIA) que, además de tratar los aspectos básicos de la programación en iOS, hablaba sobre la integración de la Libbb dentro de las aplicaciones para interactuar con espacios inteligentes. Y un segundo, abierto para toda la comunidad de la EPS que se basaba en los inicios de la programación para la plataforma de Apple.

En este último, se hacía mayor incapié en el uso de las APIs incluidas dentro del SDK y en la facilidad de programación para esta plataforma. Dicho seminario tuvo una gran acogida por parte de la EPS con más de cincuenta asistentes al evento. En el Anexo D se puede ver el cartel informativo del evento y en el Anexo E las transparencias de la presentación realizada en ambos seminarios.

6.2. Trabajo futuro

AmiPhone es una aplicación base a la que pueden agregarse mejoras en un futuro. Una de ellas, por ejemplo, es introducir una nueva forma de representación de la jerarquía por medio de botones (Descrita en la Sección 3.5.2) y dar al usuario la opción de que elija la que más se adapta a sus gustos.

También se podría introducir la opción de modificar dicha jerarquía así como los elementos que la contienen. Con esta nueva opción, el usuario no tendría que acceder a los repositorios para modificar parámetros como el nombre del objeto o su localización y, mediante sencillos pasos dentro de la aplicación, el usuario sería capaz de cambiar dichos parámetros.

Otra mejora posible es la integración de las suscripciones en la Libbb de iOS. Con ellas, las interfaces podrían representar los cambios realizados en las entidades sin la necesidad de volver a ser regeneradas.

Una mejora posible en la representación de objetos es dotar de mayor dinamismo a las interfaces mediante, por ejemplo, animaciones que varíen en función de los cambios producidos en las propiedades de éste. Consiguiendo así un mayor acercamiento de la acción al usuario de la aplicación.

Por útlimo, añadir mejoras enfocadas al ámbito comercial como aumentar el número de idiomas que puede soportar la aplicación. Como se comenta en la Sección 5.4 es una tarea sencilla con la que se aumentaría el número de posibles usuarios de ésta.

Bibliografía

[1] Android developer. http://developer.android.com/, 2010.

[2] Android market. http://www.android.com/market/, 2010.

[3] Apple developer. http://developer.apple.com/, 2010.

[4] Apple human interface guidelines. http://developer.apple.com/library/mac/#documentation/UserExperience/Conceptual/AppleHIGuidelines/XHIGIntro/XHIGIntro.html, 2010.

[5] Apple store. http://www.apple.com/es/iphone/apps-for-iphone/, 2010.

[6] Blackberry developer. http://na.blackberry.com/eng/developers/, 2010.

[7] Blackberry store. http://na.blackberry.com/eng/services/appworld/, 2010.

[8] Definición de plist. http://developer.apple.com/library/mac/#documentation/Darwin/Reference/ManPages/man5/plist.5.html, 2010.

[9] Distimo app store analytics. http://www.distimo.com/, 2010.

[10] Guía de traducción de aplicaciones de apple. http://developer.apple.com/library/ios/#documentation/MacOSX/Conceptual/BPInternational/BPInternational.html, 2010.

[11] Home automation, inc. http://www.homeauto.com/, 2010.

[12] Java me. http://java.sun.com/javame, 2010.

[13] Knx. http://www.knx.org/, 2010.

[14] Librería de zxing. http://code.google.com/p/zxing/, 2010.

[15] Multidomo. http://www.multidomo.com/, 2010.

[16] Multidomo. http://www.i-got-it.com/, 2010.

[17] Objective c. `http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/`, 2010.

[18] Open handset alliance. `http://www.openhandsetalliance.com/`, 2010.

[19] Open remote. `http://www.openremote.org/`, 2010.

[20] Palm developer. `http://developer.palm.com/`, 2010.

[21] Palm store. `http://www.palm.com/us/products/software/mobile-applications.html`, 2010.

[22] Symbian developer. `http://developer.symbian.org/`, 2010.

[23] Symbian store. `https://store.ovi.com/`, 2010.

[24] Symbian web. `http://www.symbian.org/`, 2010.

[25] Vantage. `http://www.vantagecontrols.com/`, 2010.

[26] Windows developer. `http://msdn.microsoft.com/en-us/windowsmobile/`, 2010.

[27] Windows marketplace. `http://marketplace.windowsphone.com/`, 2010.

[28] Xaml. `http://www.xaml.net/`, 2010.

[29] J. Balaguero Peña. Estudio de la plataforma Android. 2008.

[30] H.R.G. Brito. Estudios de las plataformas principales en la informática móvil. 2006.

[31] F. Casado, T. Navarrete, S. Sayago, and J. Blat. Una herramienta para la creación de interfaces multiplataforma con UIML. In *VI Congreso Español de Interacción Persona-Ordenador (INTERACCION'2005)*, 2005.

[32] M.H. Coen et al. Design principles for intelligent environments. In *Proceedings of the National Conference on Artificial Intelligence*, pages 547–554. JOHN WILEY & SONS LTD, 1998.

[33] S. Edwards, M.B. Rosson, R.C. Williges, and C. Phanouriou. UIML: A Device-Independent User Interface Markup Language. 2000.

[34] J. Eisenstein, J. Vanderdonckt, and A. Puerta. Adapting to mobile contexts with user-interface modeling. In *wmcsa*, page 83. Published by the IEEE Computer Society, 2000.

[35] J. Eisenstein, J. Vanderdonckt, and A. Puerta. Applying model-based techniques to the development of UIs for mobile computers. In *Proceedings of the 6th international conference on Intelligent user interfaces*, pages 69–76. ACM, 2001.

[36] J.G. Escribano, G.M. Manrique, and P.A.H. Coll. Ifaces: Adaptative user interfaces for Ambient Intelligence. *IADIS International Conference Interfaces and Human Computer Interaction*, pages 133 – 140, Julio 2008.

[37] K. Gajos and D.S. Weld. SUPPLE: automatically generating user interfaces. In *Proceedings of the 9th international conference on Intelligent user interfaces*, pages 93–100. ACM, 2004.

[38] Javier Gómez, Germán Montoro, Pablo A. Haya, and Xavier Alamán. Using 2d codes for creating ubiquitous user interfaces for ambient intelligence environments. In *1st International Workshop on Human-Centric Interfaces for Ambient Intelligence, (Intelligent Environments'10)*, July 2010.

[39] Javier Gómez, Germán Montoro, Pablo A. Haya, Manuel García-Herranz, and Xavier Alamán. Easing the integration and communication in ambient intelligence. *International Journal of Ambient Computing and Intelligence (IJACI) (ISSN: 1941-8647),*, 1 (3):53 – 65, 2009.

[40] P.A. Haya, G. Montoro, and X. Alamán. A prototype of a context-based architecture for intelligent home environments. *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, pages 477–491, 2004.

[41] J.M. Hullot. Ubiquitous Devices, Mobility and Context Awareness. *Sustainable Internet*, pages 123–124, 2007.

[42] R. Kistler, S. Knauth, D. Kaslin, and A. Klapproth. CARUSO-Towards a context-sensitive architecture for unified supervision and control. In *Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on*, pages 1445–1448. IEEE, 2008.

[43] G. Montoro, P.A. Haya, and X. Alamán. Interacción adaptada al contexto en una interfaz de diálogos orales para entornos inteligentes.

[44] Brad A. Myers. Mobile devices for control. *Human Computer Interaction with Mobile Devices*, Volume 2411/2002:569–579, 2002.

[45] R.R. Oskui. Especificación Formal en Coq del Módulo de Control de Acceso de MIDP 2.0 para Dispositivos Móviles Interactivos. 2008.

[46] F. Paterno and C. Santoro. One model, many interfaces. In *Computer-Aided Design of User Interfaces III: Proceedings of the Fourth International Conference on Computer-Aided Design of User Interfaces, 15-17 May 2002, Valenciennes, France*, page 143. Kluwer Academic Pub, 2002.

[47] P. Haya M. García-Herranz P.Llinás, G. Montoro and X. Alamán. Interacción gestual sobre superficies horizontales multi-contacto para un control natural. *XI Congreso Internacional de Interacción Persona-Ordenador, Interacción 2010 (CEDI 2010)*, 2010.

[48] A. Puerta. The MECANO project: comprehensive and integrated support for Model-Based Interface development. In *Computer-Aided Design of User Interfaces*, pages 19–36. Citeseer, 1996.

[49] O. Rashid, R. Thompson, P. Coulton, and R. Edwards. A comparative study of mobile application development in Symbian and J2ME using example of a live football results service operating over GPRS. In *IEEE International Symposium on Consumer Electronics, Reading, UK*, pages 203–207, 2004.

[50] M. Rayner, I. Lewin, G. Gorrell, and J. Boye. Plug and play speech understanding. In *Proceedings of the Second SIGdial Workshop on Discourse and Dialogue-Volume 16*, pages 1–10. Association for Computational Linguistics, 2001.

[51] S.G. Rojas and L.O. Díaz. *Java a Tope: J2me (java 2 Micro Edition).* Sergio Gálvez Rojas.

[52] P. Saco et al. La telefonía celular en la era blackberry. *Negotium*, 5(13):71–79, 2009.

[53] S.A.N. Shafer, B. Brumitt, and JJ Cadiz. Interaction issues in context-aware intelligent environments. *Human–Computer Interaction*, 16(2):363–378, 2001.

[54] M. Weiser. The world is not a desktop. *interactions*, 1(1):7–8, 1994.

Apéndice A

Presupuesto

1) Ejecución Material

- Compra de ordenador personal (Software incluido) 2000 €
- Alquiler de impresora láser durante 6 meses 50 €
- Material de oficina 150 €
- Total de ejecución material 2200 €

2) Gastos generales

- sobre Ejecución Material 352 €

3) Beneficio Industrial

- 6 % sobre Ejecución Material 132 €

4) Honorarios Proyecto

- 640 horas a 15 €/ hora 9600 €

5) Material fungible

- Gastos de impresión 60 €
- Encuadernación 200 €

6) Subtotal del presupuesto

- Subtotal Presupuesto 12060 €

7) I.V.A. aplicable

- 16 % Subtotal Presupuesto 1929.6 €

8) Total presupuesto

- Total Presupuesto 13989.6 €

Madrid, Diciembre 2010

El Ingeniero Jefe de Proyecto

Fdo.: David Matellano Criado

Ingeniero Superior de Telecomunicación

Apéndice B

Pliego de condiciones

Pliego de condiciones

Este documento contiene las condiciones legales que guiarán la realización, en este proyecto, de un *sistema de generación de interfaces de usuario en el campo de la Inteligencia Ambiental*. En lo que sigue, se supondrá que el proyecto ha sido encargado por una empresa cliente a una empresa consultora con la finalidad de realizar dicho sistema. Dicha empresa ha debido desarrollar una línea de investigación con objeto de elaborar el proyecto. Esta línea de investigación, junto con el posterior desarrollo de los programas está amparada por las condiciones particulares del siguiente pliego.

Supuesto que la utilización industrial de los métodos recogidos en el presente proyecto ha sido decidida por parte de la empresa cliente o de otras, la obra a realizar se regulará por las siguientes:

Condiciones generales.

1. La modalidad de contratación será el concurso. La adjudicación se hará, por tanto, a la proposición más favorable sin atender exclusivamente al valor económico, dependiendo de las mayores garantías ofrecidas. La empresa que somete el proyecto a concurso se reserva el derecho a declararlo desierto.
2. El montaje y mecanización completa de los equipos que intervengan será realizado totalmente por la empresa licitadora.
3. En la oferta, se hará constar el precio total por el que se compromete a realizar la obra y el tanto por ciento de baja que supone este precio en relación con un importe límite si este se hubiera fijado.
4. La obra se realizará bajo la dirección técnica de un Ingeniero Superior de Telecomunicación, auxiliado por el número de Ingenieros Técnicos y Programadores que se estime preciso para el desarrollo de la misma.

5. Aparte del Ingeniero Director, el contratista tendrá derecho a contratar al resto del personal, pudiendo ceder esta prerrogativa a favor del Ingeniero Director, quien no estará obligado a aceptarla.

6. El contratista tiene derecho a sacar copias a su costa de los planos, pliego de condiciones y presupuestos. El Ingeniero autor del proyecto autorizará con su firma las copias solicitadas por el contratista después de confrontarlas.

7. Se abonará al contratista la obra que realmente ejecute con sujeción al proyecto que sirvió de base para la contratación, a las modificaciones autorizadas por la superioridad o a las órdenes que con arreglo a sus facultades le hayan comunicado por escrito al Ingeniero Director de obras siempre que dicha obra se haya ajustado a los preceptos de los pliegos de condiciones, con arreglo a los cuales, se harán las modificaciones y la valoración de las diversas unidades sin que el importe total pueda exceder de los presupuestos aprobados. Por consiguiente, el número de unidades que se consignan en el proyecto o en el presupuesto, no podrá servirle de fundamento para entablar reclamaciones de ninguna clase, salvo en los casos de rescisión.

8. Tanto en las certificaciones de obras como en la liquidación final, se abonarán los trabajos realizados por el contratista a los precios de ejecución material que figuran en el presupuesto para cada unidad de la obra.

9. Si excepcionalmente se hubiera ejecutado algún trabajo que no se ajustase a las condiciones de la contrata pero que sin embargo es admisible a juicio del Ingeniero Director de obras, se dará conocimiento a la Dirección, proponiendo a la vez la rebaja de precios que el Ingeniero estime justa y si la Dirección resolviera aceptar la obra, quedará el contratista obligado a conformarse con la rebaja acordada.

10. Cuando se juzgue necesario emplear materiales o ejecutar obras que no figuren en el presupuesto de la contrata, se evaluará su importe a los precios asignados a otras obras o materiales análogos si los hubiere y cuando no, se discutirán entre el Ingeniero Director y el contratista, sometiéndolos a la aprobación de la Dirección. Los nuevos precios convenidos por uno u otro procedimiento, se sujetarán siempre al establecido en el punto anterior.

11. Cuando el contratista, con autorización del Ingeniero Director de obras, emplee materiales de calidad más elevada o de mayores dimensiones de lo estipulado en el proyecto, o sustituya una clase de fabricación por otra que tenga asignado mayor precio o ejecute con mayores dimensiones cualquier otra parte de las obras, o en general, introduzca en ellas cualquier modificación que sea beneficiosa a juicio del Ingeniero Director de obras, no tendrá derecho sin embargo, sino a lo que le correspondería si hubiera realizado la obra con estricta sujeción a lo proyectado y contratado.

12. Las cantidades calculadas para obras accesorias, aunque figuren por partida alzada en el presupuesto final (general), no serán abonadas sino a los precios de la contrata, según las condiciones de la misma y los proyectos particulares que para ellas se formen, o en su defecto, por lo que resulte de su medición final.

13. El contratista queda obligado a abonar al Ingeniero autor del proyecto y director de obras así como a los Ingenieros Técnicos, el importe de sus respectivos honorarios

facultativos por formación del proyecto, dirección técnica y administración en su caso, con arreglo a las tarifas y honorarios vigentes.

14. Concluida la ejecución de la obra, será reconocida por el Ingeniero Director que a tal efecto designe la empresa.

15. La garantía definitiva será del 4

16. La forma de pago será por certificaciones mensuales de la obra ejecutada, de acuerdo con los precios del presupuesto, deducida la baja si la hubiera.

17. La fecha de comienzo de las obras será a partir de los 15 días naturales del replanteo oficial de las mismas y la definitiva, al año de haber ejecutado la provisional, procediéndose si no existe reclamación alguna, a la reclamación de la fianza.

18. Si el contratista al efectuar el replanteo, observase algún error en el proyecto, deberá comunicarlo en el plazo de quince días al Ingeniero Director de obras, pues transcurrido ese plazo será responsable de la exactitud del proyecto.

19. El contratista está obligado a designar una persona responsable que se entenderá con el Ingeniero Director de obras, o con el delegado que éste designe, para todo relacionado con ella. Al ser el Ingeniero Director de obras el que interpreta el proyecto, el contratista deberá consultarle cualquier duda que surja en su realización.

20. Durante la realización de la obra, se girarán visitas de inspección por personal facultativo de la empresa cliente, para hacer las comprobaciones que se crean oportunas. Es obligación del contratista, la conservación de la obra ya ejecutada hasta la recepción de la misma, por lo que el deterioro parcial o total de ella, aunque sea por agentes atmosféricos u otras causas, deberá ser reparado o reconstruido por su cuenta.

21. El contratista, deberá realizar la obra en el plazo mencionado a partir de la fecha del contrato, incurriendo en multa, por retraso de la ejecución siempre que éste no sea debido a causas de fuerza mayor. A la terminación de la obra, se hará una recepción provisional previo reconocimiento y examen por la dirección técnica, el depositario de efectos, el interventor y el jefe de servicio o un representante, estampando su conformidad el contratista.

22. Hecha la recepción provisional, se certificará al contratista el resto de la obra, reservándose la administración el importe de los gastos de conservación de la misma hasta su recepción definitiva y la fianza durante el tiempo señalado como plazo de garantía. La recepción definitiva se hará en las mismas condiciones que la provisional, extendiéndose el acta correspondiente. El Director Técnico propondrá a la Junta Económica la devolución de la fianza al contratista de acuerdo con las condiciones económicas legales establecidas.

23. Las tarifas para la determinación de honorarios, reguladas por orden de la Presidencia del Gobierno el 19 de Octubre de 1961, se aplicarán sobre el denominado en la actualidad "Presupuesto de Ejecución de Contratаz anteriormente llamado "Presupuesto de Ejecución Material"que hoy designa otro concepto.

Condiciones particulares.

La empresa consultora, que ha desarrollado el presente proyecto, lo entregará a la empresa cliente bajo las condiciones generales ya formuladas, debiendo añadirse las siguientes condiciones particulares:

1. La propiedad intelectual de los procesos descritos y analizados en el presente trabajo, pertenece por entero a la empresa consultora representada por el Ingeniero Director del Proyecto.

2. La empresa consultora se reserva el derecho a la utilización total o parcial de los resultados de la investigación realizada para desarrollar el siguiente proyecto, bien para su publicación o bien para su uso en trabajos o proyectos posteriores, para la misma empresa cliente o para otra.

3. Cualquier tipo de reproducción aparte de las reseñadas en las condiciones generales, bien sea para uso particular de la empresa cliente, o para cualquier otra aplicación, contará con autorización expresa y por escrito del Ingeniero Director del Proyecto, que actuará en representación de la empresa consultora.

4. En la autorización se ha de hacer constar la aplicación a que se destinan sus reproducciones así como su cantidad.

5. En todas las reproducciones se indicará su procedencia, explicitando el nombre del proyecto, nombre del Ingeniero Director y de la empresa consultora.

6. Si el proyecto pasa la etapa de desarrollo, cualquier modificación que se realice sobre él, deberá ser notificada al Ingeniero Director del Proyecto y a criterio de éste, la empresa consultora decidirá aceptar o no la modificación propuesta.

7. Si la modificación se acepta, la empresa consultora se hará responsable al mismo nivel que el proyecto inicial del que resulta el añadirla.

8. Si la modificación no es aceptada, por el contrario, la empresa consultora declinará toda responsabilidad que se derive de la aplicación o influencia de la misma.

9. Si la empresa cliente decide desarrollar industrialmente uno o varios productos en los que resulte parcial o totalmente aplicable el estudio de este proyecto, deberá comunicarlo a la empresa consultora.

10. La empresa consultora no se responsabiliza de los efectos laterales que se puedan producir en el momento en que se utilice la herramienta objeto del presente proyecto para la realización de otras aplicaciones.

11. La empresa consultora tendrá prioridad respecto a otras en la elaboración de los proyectos auxiliares que fuese necesario desarrollar para dicha aplicación industrial, siempre que no haga explícita renuncia a este hecho. En este caso, deberá autorizar expresamente los proyectos presentados por otros.

12. El Ingeniero Director del presente proyecto, será el responsable de la dirección de la aplicación industrial siempre que la empresa consultora lo estime oportuno. En caso contrario, la persona designada deberá contar con la autorización del mismo, quien delegará en él las responsabilidades que ostente.

Apéndice C

Definición de un conjunto de entidades sencillo para demostración

```
entity Room:lab_B403_Living@amilab{
  properties {
    representable = true;
    inhabitants = 2;
    iName = {
      iNameSpanish="Living del Laboratorio AmILab, en el B-403";
      iNameEnglish = "Living of the Laboratory AmILab, in the B-403";
      iNameGerman = "Wohnen im Labor";
};
    iIcon={
iconSource="http://belisama.ii.uam.es/amiphone/images/habitacion.png";
      x=5;
      y=10;
      resolution={
        width=100;
        height=50;
      };
    };
    coordinate = {latitude=156;longitude=259;};
  }
}


entity Light:lamp1@amilab
{
  properties
  {
```

```
    representable =true;
      iName={
        iNameSpanish = "Luz del Living";
        iNameEnglish = "Living Light";
        iNameGerman = "Licht Raum";
      };
      iIcon={
iconSource="http://belisama.ii.uam.es/amiphone/images/luz_techo.png";
        x=5;
        y=10;
        resolution={
          width=100;
          height=50;
      };
    };

  }
  capabilities
  {
    isSwitchable
    {
      status ={
        binaryValue=0;
        representation={
                iName ={
                iNameSpanish = "Estado";
                iNameEnglish = "Status";
                iNameGerman = "Zustand";
        };
        iText = {
                iTextSpanish={
                        texto=("Encender","Apagar");
                        numValues=2;
                };
                iTextEnglish={
                        texto=("Turn on","Turn off");
                        numValues=2;
                };
                iTextGerman={
                        texto=("Einschalten","Ausschalten");
                        numValues=2;
                };
        };
        graphics = {
                        iFaceType = "Button";
        };
```

```
      };
    };
   }
  }
  relations
  {
    locatedAt = Room:lab_B403_Living@amilab{
                        fineLocation = "living";
                        place = "techo";
    }
  }
}


entity DimmableLight:dimlamp2@amilab
{
  properties
  {
    representable = true;
      iName={
        iNameSpanish = "Luz Regulable Derecha";
        iNameEnglish = "Adjustable Right Light";
        iNameGerman = "Licht Einstellbar";
    };
      iIcon={
iconSource="http://belisama.ii.uam.es/amiphone/images/luz_pie.png";
        x=5;
        y=10;
        resolution={
                width=100;
                height=50;
      };
    };
  }
  capabilities
  {
    isDimmable
    {
      level ={
        representation={
          iName ={
            iNameSpanish = "Estado del regulador";
            iNameEnglish = "State Controller";
            iNameGerman = "Zustand Regler";
          };
          iText = {
```

```
            iTextSpanish={
              texto=("Encender","Apagar");
              numValues=2;
            };
            iTextEnglish={
              texto=("Turn on","Turn off");
              numValues=2;
            };
            iTextGerman={
              texto=("Einschalten","Ausschalten");
              numValues=2;
            };
          };
          graphics={
            iFaceType = "Slider";
          };
        };
      numericValue=0;
      maxValue=100;
      minValue=0;
      step=5;
    };
  }
  isSwitchable
    {
      status ={
        binaryValue=0;
        representation={
                iName ={
                iNameSpanish = "Estado";
                iNameEnglish = "Status";
                iNameGerman = "Zustand";
        };
        iText = {
                iTextSpanish={
                        texto=("Encender","Apagar");
                        numValues=2;
                };
                iTextEnglish={
                        texto=("Turn on","Turn off");
                        numValues=2;
                };
                iTextGerman={
                        texto=("Einschalten","Ausschalten");
                        numValues=2;
                };
```

```
        };
        graphics = {
                        iFaceType = "Button";
        };
      };
    };
   }
  }
  relations
  {
    locatedAt = Room:lab_B403_Living@amilab{
                        fineLocation = "living";
                        place = "izquierda";
    }
  }
}
```

Apéndice D

Cartel informativo del seminario

Figura D.1: Cartel informativo del seminario.

Apéndice E

Transparencias del seminario