UNIVERSIDAD POLITÉCNICA DE VALENCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Proyecto Fin de Carrera

Diseño e implementación de una aplicación domótica en Android reconfigurable a partir de un XML

Autor:

Raúl Cano Calabuig

Director:

Ismael Torres

Diciembre 2010

A mis Padres, amigos y Maribel

Agradecimientos

Quiero mostrar mi agradecimiento a todas las personas que han hecho posible este proyecto. En primer lugar al personal del el Grupo de sistemas pervasivos del centro de investigación en métodos de producción de software (ProS) de la UPV, cuya colaboración ha resultado facilitadora en la elaboración de este trabajo.

En segundo lugar agradecer su colaboración y disposición a mi director de proyecto Ismael Torres y Nacho Mansanet por todo el apoyo en la realización de este proyecto de final de carrera.

Finalmente agradecer a todas las personas que han estado ahí durante todo el tiempo dedicado a este proyecto, especialmente a mi familia y a mis amigos, que me han apoyado en todo momento.

Índice de Contenido

Índice de Ilustraciones

1. Introducción

El proyecto “Diseño e implementación de una aplicación domótica en Android reconfigurable a partir de un XML” surge con la intención de crear una aplicación para la mencionada plataforma móvil que sea capaz de procesar y mostrar mediante interfaces sencillas, ciertos escenarios domotizados y debidamente descritos en un documento XML que obtiene la aplicación a través de la red de manera dinámica.

1.1. Motivación

Tecnologías móviles

En poco tiempo, los terminales móviles se han convertido en un elemento indispensable en nuestra vida cotidiana, siendo esta una de las razones por las cuales se ha producido una evolución del sector tan acelerada. La incorporación a los móviles de nuevas tecnologías, que nos permiten por ejemplo el acceso a internet, reproductor multimedia, fotografía digital, etc., dan al usuario muchas posibilidades de uso de su terminal.

Con las tendencias actuales de un mercado que, cada vez más, se decanta por los Smartphone, encontramos tecnologías en estos mismos terminales, que hace poco tiempo eran novedosas en el sector de la informática. El hardware implantado en los mismos, es mucho más avanzado en algunos casos, que el instalado en una ordenador de hace pocos años. El tamaño ocupado los chips y componentes se ha reducido considerablemente, gracias a la evolución de las tecnologías de fabricación.

Paralelamente a la evolución hardware que han sufrido los terminales móviles, nos encontramos con un software cada vez más avanzado, que nos permite aprovechar al máximo las características de estos nuevos terminales.

Como bien venimos viendo a través de los años, hay gran variedad de terminales móviles y con ellos gran variedad de sistemas, como es el caso de iOS,

Symbian, Windows Mobile, Android, etc. Como en todo sector, compiten por obtener o captar usuarios. Cada uno de estos sistemas tiene unas características que pueden ser apropiadas para unas u otras tareas.

En el caso de Android, como explicaremos extensamente con posterioridad en el documento, es un Sistema Operativo con licencias de software libre y código abierto. El desarrollo de aplicaciones sobre esta plataforma se realiza mediante la SDK, de una forma no muy compleja.

Domótica

La Domótica es un conjunto de sistemas capaces de automatizar una vivienda, aportando servicios de gestión energética, seguridad, bienestar y comunicación. Con este objetivo, el Grupo de sistemas pervasivos del centro de investigación en métodos de producción de software (ProS) de la UPV, desarrollan tecnología que permita una utilización más eficiente de los recursos energéticos disponibles por parte de los hogares y por lo tanto contribuya al desarrollo sostenible de la sociedad.

1.2. Objetivos del proyecto

Utilizando esta plataforma domótica desarrollada en el ProS, crearemos una aplicación móvil con interfaz en el sistema operativo Andorid, con el objetivo de gestionar la información recibida sobre la plataforma, así como intercambiar información con la misma a través de la comunicación mediante HTTP y mostrar al usuario una interfaz sencilla y apropiada de la misma.

El primer objetivo que se presenta, se refiere a la motivación de la creación de la aplicación en una plataforma que se encuentra bien valorada en el mercado y está en alza. El desarrollo de lo que será un prototipo de aplicación domótica en Android reconfigurable a partir de un XML, que muestre una interfaz gráfica atractiva.

Otro objetivo del proyecto, ha sido la puesta a prueba de los conocimientos y aptitudes adquiridas por parte del alumno a lo largo de la titulación universitaria, estas habilidades, han permitido la realización de esta aplicación.

1.3. Estructura de la memoria

En este punto describiremos brevemente que temas se expondrán y tratarán en las diferentes secciones del documento.

Para empezar, en el primer capítulo, encontramos una pequeña introducción al contenido del documento, seguidamente se presenta la motivación de la siguiente memoria y se expondrán los objetivos perseguidos con la realización del proyecto. Por último, se realiza un repaso a los contenidos del documento.

En el capítulo titulado el Propuesta de aplicación, se describe los objetivos de la aplicación, las características y necesidades que se pretenden cubrir con la misma, veremos que líneas de investigación hemos seguido para lograr nuestros objetivos y explicaremos con detalle cada una. Presentaremos el contexto tecnológico en el cual se desarrolla nuestra aplicación, dando una visión amplia tanto de Android, como de otras tecnologías con las que trataremos.

Posteriormente presentaremos la estructura del escenario al cual se enfoca nuestra aplicación, exponiendo las diferentes características de la misma y sus funciones. A continuación, procederemos a explicar las posibles tecnologías que utilizaremos en la gestión del escenario, así como los puntos importantes que contendrá el documento de entrada.

Conociendo el contexto, nos centraremos en el diseño y estructura de la aplicación realizada, explicando cada uno de los diferentes apartados de la misma. Se mostrará la implementación de las distintas características que vamos a encontrar en la aplicación.

Una vez hemos mostrado todas las características de la aplicación, se presentará el prototipo realizado. Se podrán observar las diferentes posibilidades que nos aporta, así como los resultados de las pruebas que se han realizado con el mismo.

Para finalizar el documento describiremos las conclusiones que hemos alcanzado en la realización de este proyecto de final de carrera.

2. Propuesta de aplicación

2.1. Objetivo de la aplicación

Este punto expone los objetivos que la aplicación desarrollada para el proyecto debe ofrecer.

Ofreceremos una aplicación dinámica en la obtención de datos y la representación de los mismos. La interfaz debe ser sencilla, amigable e intuitiva, para que el usuario pueda navegar entre las diferentes capas de la estructura, de una forma rápida.

Otra de las características que debe ofrecer nuestro interfaz, es el soporte de los diferentes dispositivos y localizaciones que contenga la estructura personalizada para cada clase de dispositivo, que nos permite la interactuación con el mismo, pudiendo así utilizar las funciones que nos ofrece dicho elemento.

En cuanto a la obtención de los datos, la aplicación debe disponer de un apartado que permita al usuario la introducción de la dirección del servidor (URL) que referencie a un documento XML que contiene la información necesaria para formar una estructura correctamente, seguidamente la aplicación procesará la información y mostrará mediante una interfaz gráfica, al usuario, los diferentes dispositivos, permitiendo la navegabilidad entre ellos y el entorno en el que se encuentran.

En el punto Descripción del escenario y entrada de datos, explicaremos más detalladamente, los métodos utilizados para la obtención de estas características y funciones de la aplicación.

2.2. Líneas de investigación

Para la realización de este proyecto, han sido necesarios una serie de conocimientos, algunos adquiridos en el desarrollo del mismo. En este punto, veremos una pequeña muestra de las líneas de investigación mediante las cuales hemos conseguido adquirir esos conocimientos necesarios con el fin de realizar una aplicación acorde a los mismos.

Las dos líneas de investigación principales seguidas por el alumno y que exponemos seguidamente son el desarrollo de aplicaciones en Android y dentro de la entrada de datos, la parte que incluye el procesado de un documento XML.

2.2.1. Desarrollo de aplicaciones en Android

Como es habitual en todas las plataformas, para lograr una aplicación aceptable, primero debemos familiarizarnos con el entorno, para ello, debemos estudiar y explorar dicha plataforma, con tal de adquirir los conocimientos que nos serán necesarios para nuestros intereses.

El primer paso en esta línea de investigación, es conocer sus características principales, en este caso, la aplicación se desarrollan en Android versión 1.6, también conocida como “Donut”.

Una vez conocemos las características principales de la misma y conocemos su funcionamiento sobre un terminal móvil, presentamos el entorno donde podremos proceder al desarrollo de aplicaciones, hemos elegido el entorno más difundido para crear aplicaciones en Android como es el caso de Eclipse, junto al SDK de Android que podemos encontrar en la página oficial. Teniendo configuradas las dos partes del entorno, realizaremos una pequeña aplicación que nos servirá para familiarizarnos con el funcionamiento del SDK y el entorno. Podemos encontrar en el mismo SDK ejemplos que nos familiarizan con los métodos de creación y funcionamiento de aplicaciones básicas en Android.

Como último punto de esta línea de investigación, se realizaron una serie de ejemplos de aplicaciones, para adquirir conocimientos tanto del desarrollo de interfaces gráficas como de componentes, gestión de recursos y estructuración de las aplicaciones.

Para el estudio de esta plataforma se utilizaron libros mencionados en el punto

Referencias, que sirvieron de guía para la instalación del entorno de desarrollo y descripción de las características de la versión, así como, el desarrollo de aplicaciones ejemplo básicas.

2.2.2. Procesado de un documento XML

Nuestra aplicación, tiene como principal entrada de datos, la recuperación de un documento en XML, que se aloja en un servidor, cuya URL, se debe proporcionar a la misma. Una vez obtenido este documento debemos extraer de él la información que necesita la aplicación para formar una estructura que alimentará la aplicación. Para ello necesitamos un recurso que nos permita realizar esta tarea.

Sabiendo que las aplicaciones se realizan en el lenguaje de programación Java, y que este dispone de recursos para obtener información de un documento XML, se investigaron las diferentes posibilidades que ofrece Java para el tratado de XML. Debemos estudiar las posibilidades que ofrece cada uno de estos *Parser*, estudiando su compatibilidad con la plataforma y sus características de extracción de datos.

Finalmente, se comparan y se decide que proceso será el más beneficioso para nuestra aplicación, valorando diferentes aspectos, como puede ser la sencillez para el programador, la adaptabilidad a la plataforma o el consumo de recursos.

2.3. Contexto tecnológico

2.3.1. Android

En este punto haremos un pequeño análisis de Android, mencionamos un poco su historia, su situación de mercado y la estructura básica sus aplicaciones, desde el punto de vista del desarrollador y del usuario de la plataforma.

2.3.1.1. Introducción

Desde su lanzamiento en octubre del 2008 y considerado un sistema joven, Android se ha convertido en una gran plataforma en el mercado. Tal como se observa en la tabla de la Ilustración 2.1, se sitúa en dos años con una cuota de mercado de 14.7% a la altura de otros importantes sistemas operativos, según un informe de la IDC.

Worldwide Converged Mobile Device Operating System Market Shares and 2010-2014 Growth

Operating System	2010 Market Share	2014 Market Share	2014/2010 Change
Symbian	40.1%	32.9%	-18.0%
BlackBerry OS	17.9%	17.3%	-3.5%
Android	16.3%	24.6%	51.2%
IOS	14.7%	10.9%	-25.8%
Windows Mobile	6.8%	9.8%	43.3%
Others	4.2%	4.5%	8.3%
Total	100.0%	100.0%	

Ilustración 2.1. Posición en el mercado de los SO móviles.

Posee varias características que son claves para este éxito, entre ellas encontramos la versatilidad del sistema, adaptándose rápidamente a los cambios de hardware y avances tecnológicos actuales. Destacamos además la política de admisión de aplicaciones de *Android Market*, con esto han conseguido que muchos

desarrolladores creen aplicaciones para este sistema operativo, que aunque relativamente nuevo, es muy accesible.

El desarrollo de aplicaciones en Android se realiza principalmente en el lenguaje de programación orientada a objetos Java en un entorno parecido a Java SE 5, sin embargo, la implementación no es oficial de SUN por lo que tiene ciertos matices al estar basado realmente en el proyecto *Harmony* de la fundación Apache, adaptado por Google para emplearlo en Android. El entorno de desarrollo para aplicaciones Android más difundido es Eclipse junto al *plugin ADT* (*Android developement tools*) mantenido por los creadores de la plataforma por lo que será el escogido para la realización del proyecto.

Estas características junto a muchas otras hacen de Android un sistema operativo que cumple los requisitos mínimos para el desarrollo de nuestra aplicación, aplicando así los conocimientos adquiridos de los lenguajes de programación utilizados por ese SO durante los cursos anteriores como son Java y XML.

2.3.1.2. Arquitectura

Con el objetivo de comprender mejor Android, echaremos un pequeño vistazo a su arquitectura y las características de la misma, en este apartado conoceremos de qué se compone el sistema operativo con mayor detalle, describiendo cada una de las partes.

El sistema operativo Android se compone de las siguientes partes:

Aplicaciones: Todas las aplicaciones escritas en el lenguaje de programación Java, este es el objetivo principal del proyecto. Entre otras, el propio sistema contiene algunas aplicaciones básicas, como pueden ser, cliente de email, programa de SMS, calendario, contactos, etc.

Framework de aplicaciones: Los desarrolladores tienen acceso completo a los mismos APIs del framework usados por las aplicaciones base. La arquitectura está

diseñada para simplificar el reúso de componentes. Éste mismo mecanismo permite que los componentes sean reemplazados por el usuario.

Librerías: Android incluye un set de librerías C/C++ utilizadas por varios componentes del sistema Android. Estas capacidades se exponen a los desarrolladores a través del framework de aplicaciones de Android. Algunas son: *System C library* (implementación librería C Standard), librerías de medios, librerías de gráficos, 3d, *SQLite*, entre otras.

Runtime de Android: Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual *Dalvik*.

Núcleo - Linux: Android depende de un Linux versión 2.6 para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, etc. El núcleo también actúa como una capa de abstracción entre el hardware y el software.

2.3.1.3. Jerarquía Visual

La principal clase de Android es *Activity*, un objeto de la clase **android.app.*Activity***. Una actividad hace multitud de cosas, pero por ella misma no se presenta nada por pantalla. Para conseguirlo es necesario diseñar el UI, con *View* y *Viewgroup*, que son las clases que se utilizan para crear la interfaz entre el usuario y la plataforma Android.

View*:** Una *View* es un objeto cuya clase es **android.*View*.*View. Es una estructura de datos cuyas propiedades contienen los datos de la capa y la información específica del área rectangular de la pantalla. La clase *View* es útil como clase base para los *Widgets*, que son unas subclases ya implementadas que dibujan los elementos en la pantalla. Los *Widgets* contienen sus propias medidas. La lista de *Widgets* es amplia e incluyen entre otros: *Text*, *EditText*, *InputMethod*, *MovementMethod*, *Button*, etc.

***View*groups:** Un *View*group es un objeto de la clase **android.*View*.*View*group**, como su propio nombre indica, un *View*group es un objeto especial de *View* cuya función es contener y controlar la lista de *View*s y de otros *View*groups. Los *View*groups te permiten añadir estructuras a la interfaz y acumular complejos elementos en la pantalla que son diseccionados por una sola entidad. La clase *View*group es útil como base de la clase *Layouts*, que son subclases implementadas que proveen los tipos más comunes de los *Layouts* de pantalla. Los *Layouts* proporcionan una manera de construir una estructura para una lista de *View*.

Árbol estructurado de la interfaz UI: En la plataforma Android tú defines una *Activity* del UI usando un árbol de nodos *View* y *View*groups, tal como se muestra en la Ilustración 2.2. El árbol puede ser tan simple o complejo como necesites hacerlo, y se puede desarrollar usando los *Widgets* y *Layouts* que Android proporciona o creando tus propias *View*s.

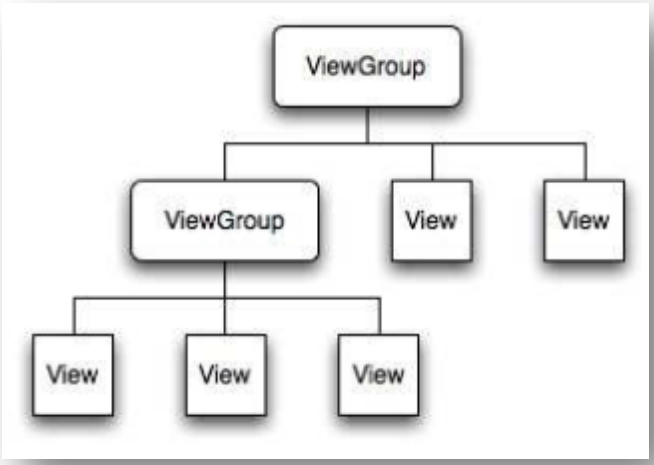


Ilustración 2.2. Árbol de Interfaz

Para añadir el árbol a la pantalla, la *Activity* llama al método "**setContent*View***" y pasa una referencia al objeto nodo principal. Una vez que el sistema Android ha referenciado el objeto nodo principal ya puede trabajar directamente con el nodo para anular, medir y dibujar el árbol.

Como se ha dicho anteriormente, cada *Viewgroup* es el responsable de tomar medidas sobre el espacio que tienen, preparando a sus hijos y llamando a "**Draw**" por cada hijo que se muestra a sí mismo. El hijo hace una petición sobre el tamaño y la localización del padre, pero el objeto padre toma la última decisión sobre el tamaño que cada hijo puede tener.

LayoutParams: Cómo un hijo especifica su posición y su tamaño. Todos los *Viewgroup* utilizan como clase anidada una extensión de ***View*Group.LayoutParams**. Esta subclase contiene los tipos de propiedades que definen la posición y el tamaño de un hijo, en propiedades apropiadas para la clase de grupo de clases.

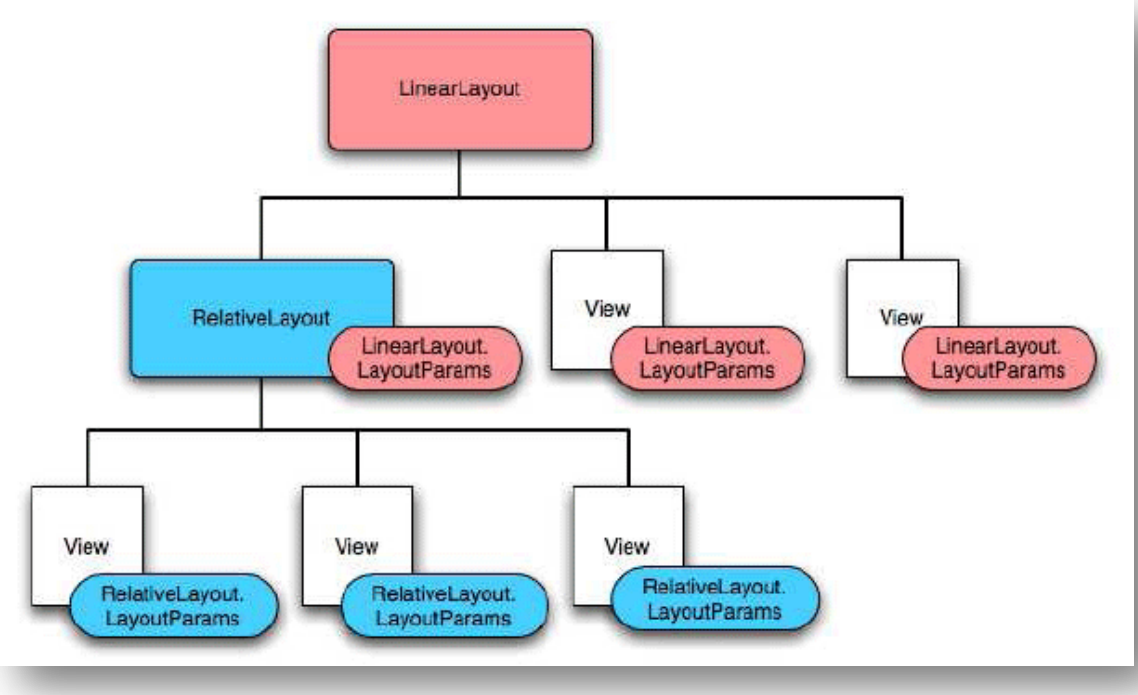


Ilustración 2.3. Ejemplo Layout

Hay que destacar que cada subclase *LayoutParams* tiene su propia sintaxis para cambiar los valores. Cada elemento hijo debe definir unos *LayoutParams* que sean apropiados para su padre, aunque se podrían definir diferentes *LayoutParams* para sus hijos.

Todos los *Viewgroup* incluyen anchura y altura, muchos también incluyen márgenes y bordes. Se puede indicar que el *View* tenga las dimensiones del tamaño de su contenedor, o que llegue a ser tan grande como el contenedor le permita.

2.3.2. XML (Extensible Markup Language)

Este metalenguaje extensible está muy presente en el trascurso de nuestro proyecto, como ya sabemos, puede tener muchas funciones.

En nuestro caso lo encontramos como intermediario para la comunicación de datos, tal y como venimos señalando durante el trascurso de este documento. Además Android hace un uso extensivo del estándar XML para la definición de las interfaces gráficas y del manifiesto que cada una de las aplicaciones Android posee.

Existen varias formas de procesar documentos XML en Java, necesitaremos conocerlas e implantar la más satisfactoria al proyecto, con la finalidad de aplicarla en el documento de entrada de datos de la aplicación. En el próximo capítulo veremos las diferentes opciones de las que disponemos en este aspecto y cual escogeremos finalmente para implantarla.

3. Descripción del escenario y entrada de datos

3.1. Estructura del escenario utilizado

En este punto, enumeraremos los diferentes dispositivos que ofrece a fecha de hoy el escenario y que dispositivos de este escenario soporta la aplicación. Podemos encontrar más detalladamente explicados en el proyecto de final de carrera "Implementación de aspectos de eficiencia energética en el Hogar Digital: un caso práctico" desarrollado por Miriam Gil Pascual y dirigido por el profesor Joan Fons i Cors.

La aplicación desarrollada deberá comunicarse con los dispositivos que a continuación se enumeran a través de un *Servlet* y que se encuentran en el laboratorio del ProS:

- AngularDimmer
- HexDimmer
- PercentualDimmer
- Pulse
- Sensor
- Switch
- VerticalMovement

Cada unos de estos dispositivos, tendrá una ubicación asociada a él, y esta misma ubicación puede contener más de un dispositivo, por lo tanto se convierten en otro punto importante de la estructura.

En resumen en la estructura podemos encontrar ubicaciones (**Location**) y dispositivos (**Device**), como ya hemos visto existen distintos tipos de dispositivos según sea su función final, por tanto, deberemos distinguir en todas las posibilidades existentes.

3.2. Gestión y Procesado del escenario

Una de las características principales de la aplicación desarrollada es la entrada de datos, como se menciona anteriormente, debe tratar y analizar el documento recibido mediante la URL proporcionada por el usuario. Con tal de cumplir estas características se deben contemplar varios puntos.

En primer lugar, y como explicamos en el sub-apartado Presentación del escenario, debemos conocer y comprender el documento de entrada, estudiando el formato de los datos y las partes que utilizará la aplicación.

Una vez estudiado el documento de entrada, debemos realizar la estructura que albergará los datos que serán leídos del mismo documento, este punto implica conocer la jerarquía de los datos y conocer sus atributos y características, para poder modelar una estructura correcta.

Para terminar con la obtención de datos de la entrada, estudiamos cada uno de los procesadores de documentos XML ofrecidos por Java y compatibles con Android, comparamos las posibilidades que nos ofrecen e implantamos el elegido en la aplicación, adaptando sus funciones a nuestras necesidades. En el sub-apartado Procesado del escenario encontramos las diferentes tecnologías ofrecidas por el lenguaje, así como los problemas surgidos para implantarlas.

3.2.1. Presentación del escenario

El escenario que recibirá nuestra aplicación se encuentra expresado en XML, que posee una estructura determinada. En el caso de nuestra aplicación, solo debemos prestar atención a los sectores que nos interesan, ya que este mismo documento, contiene información que es redundante para nosotros y en cambio es importante para otro tipo de proyectos de la plataforma.

Dicho esto, vamos a exponer las partes del documento que son primordiales para nosotros, de las cuales obtendremos información y analizaremos con el sistema de tratado de documentos XML que hemos elegido.

```
−<Location>
    <ID ID="FirstFloor"/>
    <Name name="The First Floor"/>
    <SuperLocationIDRef superLocationID="Home"/>
    <Image URL="firstFloor.jpg" PosX="20px" PosY="10px" Width="700px" Height="500px"/>
  −<Sublocations>
    −<Sublocation>
        <LocationIDRef locationID="Kitchen"/>
        <ClickableArea PosX="25px" PosY="25px" Width="200px" Height="150px"/>
        <Thumbnail URL="kitchen_thumb.jpg" PosX="20px" PosY="20px" Width="210px" Height="160px"/>
      </Sublocation>
    </Sublocations>
    <LocationDevices/>
  </Location>
−<Location>
```

Ilustración 3.1 Ejemplo de la sección Location del documento de entrada

Siguiendo un orden secuencial del documento, el primer objeto que nos encontramos son las diferentes ubicaciones, que presentan diferentes características.

Como vemos en la Ilustración 3.1 dentro del tag *Location* encontramos los atributos de la misma, podemos distinguir varios campos, los más importantes para nosotros son: el campo bajo la etiqueta ID que será el identificador de la localización, *Name* que contiene el nombre completo de la misma, *SuperLocationIDRef* donde encontramos la ID de la localización que estará por encima de la actual, es decir la localización padre o *SuperLocation*, la etiqueta *Sublocations* hace referencia a las localizaciones que existen dentro de la misma, por último *LocationDevice* contiene los dispositivos que están ligados a esta localización.

Prosiguiendo con la lectura del documento, la siguiente parte relevante, la encontramos en los dispositivos (“**Device**”), en cuyo apartado encontramos diferentes características y descripción del mismo, así como que posición y en que ubicación se encuentran.

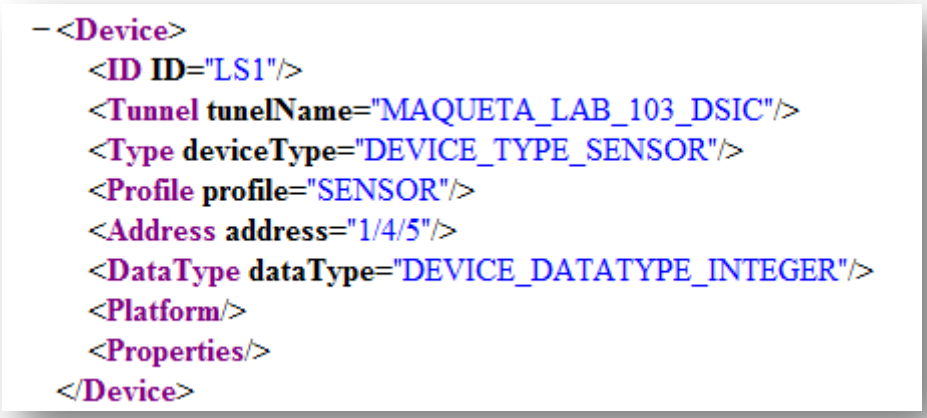


Ilustración 3.2 Ejemplo de la sección Device del documento de entrada

Tal como se observa en la Ilustración 3.2, al igual que en *Location*, dentro del tag *Device*, se encuentran diferentes etiquetas como son: ID que contiene el identificador del dispositivo, *Type* y *DataType* que nos indican el tipo del dispositivo, la etiqueta *Properties* puede contener información sobre el nombre completo del elemento y otras características de menor utilidad.

Ahora conocemos la distribución de la información que nos es necesaria para el desarrollo correcto de la aplicación, el siguiente paso es crear una serie de clases que contendrán dicha información que seguidamente extraeremos.

3.2.2. Estructura e implementación de las clases necesarias

En este apartado, describiremos las clases que componen la estructura creada para albergar la información recibida de la lectura y procesado del documento de entrada. Dicha estructura se compone de tres clases importantes y diferenciadas: *Installation*, *Location* y *Device*. A continuación describiremos las relaciones entre estas clases y la composición de las mismas con la ayuda de un diagrama de clases.

Installation

Comenzando por la parte superior, la clase padre es la llamada *Installation*, la cual contiene la referencia a la ubicación padre ("*ParentLocation*") y los dispositivos ("*Device*") de los que se compone la estructura, así como el nombre e identificador de la misma. En esta clase encontraremos métodos, constructores, de consulta (*get*) y modificación (*set*) de los atributos.

Location

La clase Location, contiene una referencia a su *ParentLocation*, que en caso de ser la raíz, tendrá el valor de la misma, como hemos visto en el punto anterior, esta referencia será extraída de la etiqueta *SuperLocationIDRef*. Además una ubicación puede contener cualquier número de dispositivos en su interior y de ubicaciones hijo, es decir, una relación entre clases de 0 a muchos. Encontramos otros atributos como el *ID* de la localización, importante para su identificación y el nombre de la misma, así como otros atributos.

Como métodos de esta clase, encontramos métodos que nos permitirán consultar la ubicación padre de la misma y los diferentes dispositivos que contiene (tanto por referencia al mismo por identificador, como por referencia a la estructura que los contiene). Lógicamente, encontramos también, métodos que nos permiten modificar los atributos y constructores.

Device

Por último, la clase *Device*, es parte de la *Installation* y debe pertenecer a una ubicación. Tanto en la definición de la ubicación, como en el mismo dispositivo, se hace referencia al lugar al que pertenece. En la clase, podemos encontrar también otros atributos como el identificador y el nombre del dispositivo, y otros que son características del mismo dispositivo.

Al igual que anteriormente, en otras clases, encontramos métodos que nos permiten, la creación de un objeto de la clase, así como la modificación y consulta de los atributos del mismo.

Puesto que los dispositivos soportados por la aplicación tienen varios tipos y para hacer más reutilizable la estructura que se ha creado, se crea una clase extensión de Device para cada dispositivo con el nombre del mismo, así pues, estas clases contendrán los atributos y métodos de su clase padre ("Device"). Además en esta clase se introducen distintos métodos que nos permiten la comunicación con el dispositivo real, mediante una serie de protocolos que más adelante se explicarán con más detalle.

3.2.3. Procesado del escenario

En este capítulo profundizaremos en la obtención y procesado de información del documento XML que corresponde a la estructura domótica que se introduzca en la interfaz.

El procesado de un documento, consiste en procesar la información del mismo con tal de obtener los datos que alberga. En Java existen varios métodos que nos sirven a este fin, entre ellos encontramos SAX, DOM, JDOM, etc.

3.2.3.1. Manipuladores de datos XML

Como hemos dicho existen varios manipuladores de datos XML en lenguaje java. En este punto expondremos las características de cada uno de ellos y finalmente estudiando su adaptabilidad a la plataforma y beneficios para nosotros, nos decantaremos por uno.

DOM

Document Object Model o *DOM*, fue creado para ser un lenguaje neutral e inicialmente usado para manipulación de páginas HTML con *JavaScript*, proporciona un conjunto estándar de objetos para representar documentos HTML y XML, genera un árbol jerárquico en memoria de la información del documento. A través del *DOM*, los programas pueden acceder y modificar el contenido, estructura y estilo de los documentos HTML y XML, que es para lo que se diseñó principalmente.

Este procesador de documentos está más orientado a la posibilidad de agregar, eliminar o modificar cualquier punto del árbol. Por tanto y ya que nuestro principal objetivo es la extracción de datos del documento y no la modificación del mismo. Este procesador proporciona posibilidades que no van a ser utilizadas, las cuales hacen más costosa la aplicación en tiempo y memoria utilizada.

SAX

SAX ("Simple API for XML") procesa el documento o información en XML de una manera muy diferente a *DOM*, *SAX* procesa la información por eventos. A diferencia de *DOM* que genera un árbol jerárquico en memoria, *SAX* procesa la información en XML conforme esta sea presentada (evento por evento), efectivamente manipulando cada elemento a un determinado tiempo, sin incurrir en uso excesivo de memoria. Por lo tanto puede notar las siguientes características:

- *SAX* es un "*Parser*" ideal para manipular archivos de gran tamaño, ya que no ocupa generar un árbol en memoria como es requerido en *DOM*, como es nuestro caso el documento podría tener una extensión considerable.

- Es más rápido que utilizar *DOM*, no requiere un uso excesivo de memoria por lo tanto nuestra aplicación será más liviana y rápida.

- El inconveniente más relevante en nuestro uso de esta tecnología es el modo en el cual *SAX* procesa la información, debido a que *SAX* funciona por eventos no es posible manipular información una vez procesada, en *DOM* no existe esta limitación

ya que se genera el árbol jerárquico en memoria y es posible regresar a modificar nodos.

Para solventar este último inconveniente, optamos por crear la estructura anteriormente expuesta, en la cual introduciremos los datos que en cada evento vayamos recibiendo del documento.

JDOM

A pesar de existir diversos "*Parser*" para ambientes Java las metodologías *SAX* y *DOM* no fueron diseñadas con Java en mente, con la intención de proporcionar una metodología para procesar XML más acorde con el mundo Java surgió *JDOM* o **J**ava **D**ocument **O**bject **M**odel. Ha causado un gran interés debido a la *sencillez* de procesar XML con Java comparado con *DOM* o *SAX*.

Debido a estas características y ya que es muy intuitivo y sencillo para el programador, en principio nos decantamos por implantar este método para manipular XML. Sin embargo, pese a ser la mejor opción, en su versión 1.0 (versión más actual en el momento del desarrollo de la aplicación) no soportaba Android. Por lo tanto se desestima esta opción como Parser de la entrada de datos.

Una vez hemos investigado las opciones que tenemos para la tarea a realizar, debemos proceder a elegir el recurso que cumpla los requisitos. Como hemos mencionado a pesar de ser la mejor opción *JDOM* queda fuera por incompatibilidad con la plataforma, por tanto, se ha elegido *SAX* para llevar a cabo esta tarea, ya que frente a *DOM*, es más rápido y menos pesado, además es un Procesador de documentos por eventos lo cual se ajusta más a nuestras necesidades.

3.2.3.2. Manejador del escenario con SAX (*Handler*)

Para implantar SAX a nuestra aplicación, debemos crear un manejador del escenario o *Handler*, será el encargado de responder debidamente a los eventos que se produzcan sobre el documento de entrada al ser leído.

El *DefaultHandler* es una interfaz que responde a eventos. Por lo que se llamarán los métodos especificados cuando suceda algo en concreto. Debemos extender esta clase y reescribir los métodos principales para nuestros fines. Para ello crearemos la clase *HousedHandler*, los métodos principales son:

startDocument: Que se invoca cuando se detecta que el documento empieza. Como vemos en la Ilustración 3.3**Error! Reference source not found.**, utilizamos el inicio del documento para inicializar las variables que vamos a utilizar, en este caso, inicializamos la estructura creada y las ArrayList auxiliares que utilizaremos para el proceso de extracción de la información del documento.

```
public  void startDocument() throws SAXException
{
    inst = new Installation();
    locations = new ArrayList<Location>();
    devices = new ArrayList<Device>();

}
```

Ilustración 3.3 Método startDocument del Handler en SAX

endDocument: Que se invoca cuando detecta que el documento ha acabado. En este caso, cuando se inicie este evento, revisaremos y terminaremos de crear la estructura que alberga la información del XML, comprobando su correcta formación y estado.

```
public void endDocument() throws SAXException
```

Ilustración 3.4 Método endDocument del Handler en SAX

startElement: Que se invoca cuando encuentra un nuevo elemento o tag. Cuando se active este evento, se comprueba, el nombre de la etiqueta y las variables estado, para saber en qué momento de la estructura nos encontramos y poder obtener los datos del documento de manera correcta.

endElement: Que se invoca con el cierre del tag, debemos devolver el valor inicial a las variables estado e introducir el elemento leído en la estructura de manera correcta.

3.2.3.3. Inicialización y utilización del manejador

En el punto anterior, se ha explicado el contenido del manejador del escenario, con el cual se realizara la gestión y corrección del documento XML, pero, este manejador, debe ser introducido en la aplicación, en concreto en el intérprete o "Parser". Como vemos en la imagen posterior, seguiremos ciertos pasos para la introducción y correcto funcionamiento del manejador.

```
//Creamos una nueva Factoria SAXParser.
SAXParserFactory spf = SAXParserFactory.newInstance();
//Creamos el SAXParser
SAXParser sp;
sp = spf.newSAXParser();
//Obtenemos el XML Reader
XMLReader xr = sp.getXMLReader();
//Iniciamos nuestro handler;
HousedHandler myHandler = new HousedHandler();
```

Ilustración 3.5 Configuración e inicialización del Parser SAX

El primer paso, es para obtener el proveedor de intérpretes, seguidamente a inicializar este proveedor, obtendremos el intérprete con la función ***newSAXParser.*** Como vemos debemos inicializar nuestro manejador citado anteriormente, con tal de introducirlo en la especificación del intérprete.

```
URL path = new URL (path_string);

xr.setContentHandler(myHandler);
xr.parse(new InputSource(path.openStream()));
```

Ilustración 3.6 Invocando al Parser

Una vez tenemos la dirección del documento a analizar, introducimos el manejador y ordenamos al intérprete que obtenga el documento y lo analice. Al finalizar el análisis, podremos proceder a obtener los datos ya estructurados y correctos.

En caso de ocurrir una excepción en la llamada, se capturará y se mostrará al usuario por interfaz gráfica la naturaleza de dicha excepción.

4. Arquitectura de la aplicación

El siguiente apartado tiene como objetivo servir de la base para presentar la arquitectura propuesta para el prototipo de la aplicación móvil con el objetivo de crear una aplicación más sostenible. Conseguimos esto con el empleo de buenas prácticas de programación, separando los interfaces y la implementación.

4.1. Arquitectura por capas

La arquitectura con la que se desarrolla la aplicación móvil en Android es multicapa, dedicando así a cada una de las capas una funcionalidad concreta y bien definida.

4.1.1. Capa interfaz

La capa de interfaz o de presentación de la aplicación está compuesta por las vistas definidas en ficheros XML. Tiene un papel muy importante en la aplicación, ya que se trata de la presentación de la aplicación al usuario. En la Ilustración 4.1 podemos ver un fragmento de los ficheros XML que implementan una de las vistas.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/LinearLayout01"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical">

<TableLayout
 android:id="@+id/TableLayout01"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
 android:layout_weight="2">
```

Ilustración 4.1 Fragmento del layout switch_device.xml

4.1.2. Capa de modelo de negocio

Esta capa es la encargada de manejar los datos enviados por anterior, es decir, en función de los eventos recibidos por la capa de presentación deberá tomar decisiones de modificar el modelo y mostrar los resultados. Aquí se implementa toda la lógica de negocio. En la aplicación, esta capa se implementa en Java.

4.1.3. Capa de datos

Es la encargada de acceder a los datos según reciba las solicitudes de la capa anterior. Se obtendrán los datos del documento de entrada, mediante el tratado y extracción de datos del mismo.

4.2. Estructura de la aplicación

El objetivo del siguiente apartado es presentar la estructura interna de la aplicación realizada, en base a los conocimientos adquiridos sobre la tecnología utilizada y acorde a lo presentado en anteriores apartados.

4.2.1. Directorio "src"

Dentro de este directorio encontramos los paquetes Java que contiene la aplicación, a continuación, explicaremos con detalle el contenido de cada uno de ellos, esto ayudará a encontrar fácilmente el código encargado de cada funcionalidad.

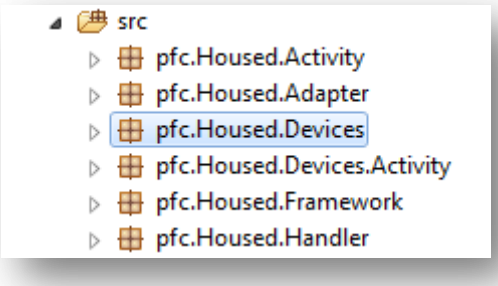


Ilustración 4.2 Directorio "src" de la aplicación

La funcionalidad distribuida en paquetes quedaría de esta forma:

pfc.Housed.*Activity*

En este paquete podemos encontrar la actividad principal de la aplicación, llamada "Housed", será la encargada de inicializar los componentes de la misma y cargar la interfaz textual principal. Junto a esta actividad, encontramos otra, llamada "ChangeUrl*Activity*", que nos permitirá mediante la carga de una interfaz diseñada para ello, cambiar la *Url* del archivo de entrada.

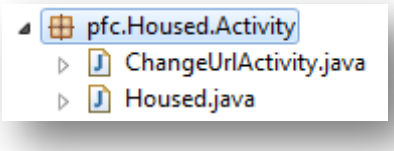


Ilustración 4.3 Paquete pfc.Housed.Activity

pfc.Housed.Adapter

Situado en este paquete encontramos el adaptador que nos permite rellenar el listado tratando uno a uno cada objeto a introducir en el. Esta clase hereda del adaptador por defecto llamado "BaseAdapter" y tiene como tareas, cargar las *View* de cada uno de los objetos que componen la fila, así como determinar que objetos pueden ser pulsados.

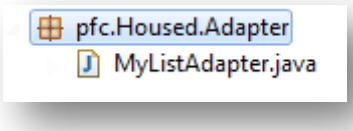


Ilustración 4.4 Paquete pfc.Housed.Adapter

pfc.Housed.Device

Este paquete contiene la implementación de los posibles dispositivos de los que dispondrá nuestra aplicación, dentro de las mismas clases encontramos métodos que permiten la comunicación con los dispositivos mediante llamadas HTTP por método GET al servidor que los maneja, dando la opción de obtener los diferentes estados que puede tener cada uno de los dispositivos.

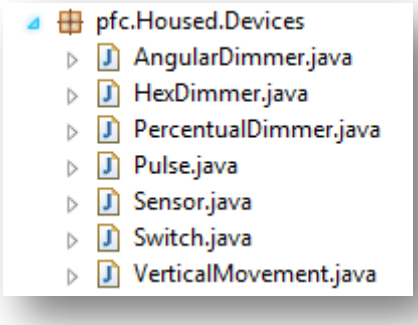


Ilustración 4.5 Paquete pfc.Housed.Devices

Todas las clases contendías en este paquete heredan de la clase “Device” contenida en el paquete **pfc.Housed.Framework**.

pfc.Housed.Devices.*Activity*

En este paquete se encontrarán las *Activity* encargadas del control del estado de los dispositivos, así como de la carga de las interfaces de cada uno de los tipos. Cada una de estas *Activity* representará un dispositivo de un tipo y en función de las acciones realizadas por el usuario utilizará los métodos de los objetos determinados.

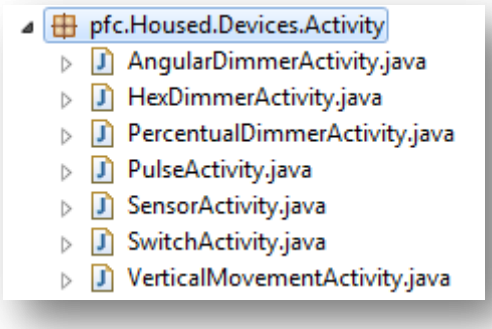


Ilustración 4.6 Paquete pfc.Housed.Devices.Activity

pfc.Housed.Framework

Encontramos dentro de este paquete, la implementación de la estructura que soporta la obtención de datos.

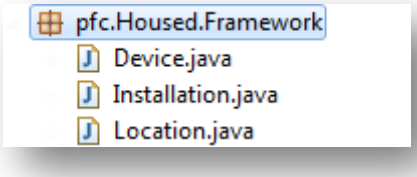


Ilustración 4.7 Paquete pfc.Housed.Framework

pfc.Housed.*Handler*

Por último, este paquete contiene el *Handler* o manejador de eventos del intérprete que utilizaremos para la entrada de datos de la aplicación. Esta clase hereda del manejador por defecto llamado "Default*Handler*" y reescribiendo sus métodos.

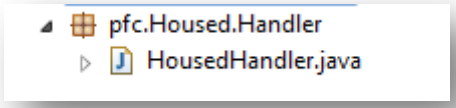


Ilustración 4.8 Paquete pfc.Housed.Handler

4.2.2. Directorio “res”

En este directorio almacenaremos todo tipo de recursos externos que utilizará la aplicación para su funcionamiento. Los más importantes que encontramos son las imágenes, almacenadas en el directorio “drawable” y los ficheros xml que contienen la definición de las vistas en “layout”.

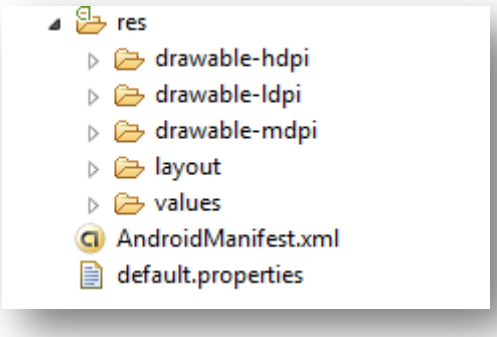


Ilustración 4.9 Directorio "res" de la aplicación

4.3. Implementación de actividades

Android basa sus aplicaciones en el uso de las *Activity*, relacionadas con una interfaz de usuario, son las encargadas de gestionar los eventos que sobre ella se produzcan. En este apartado, veremos las actividades que componen nuestra aplicación y que funciones tiene cada cual de ellas.

Cada una de las actividades creadas para la aplicación debe de ser especificada en el Manifiesto de actividades, donde podremos ver, las actividades que componen la aplicación, así como, las características y opciones de las mismas.

Como explicaremos a continuación, la aplicación se compone de una actividad principal, la cual será la encargada de crear sub-actividades y proporcionarles la información necesaria para su correcto funcionamiento, así como recibir información de las mismas cuando sea necesario.

4.3.1. Main Activity

La actividad principal de nuestra aplicación se sitúa en el paquete Java llamado **pfc.Housed.*Activity*** y la hemos llamado "Housed.java". Como actividad propia de la aplicación debe estar registrada en el Manifiesto de actividades ("AndroidManifest.xml"), en este registro, debe constar que dicha actividad es la actividad principal y tendrá como categoría "*Launcher*", esto nos permitirá iniciar la aplicación con esta actividad.

```
<activity android:name=".Activity.Housed"
          android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Ilustración 4.10 Fragmento de AndroidManifest.xml que hace referencia a MainActivity

En el punto anterior, se mencionó que cada actividad se relaciona con una interfaz de usuario, en este caso, nuestra actividad principal será la encargada de gestionar los eventos e introducir la información en nuestra interfaz de ubicaciones. Esta interfaz es de tipo lista de objetos, por tanto, esta característica implica que *Housed* herede de *ListActivity* y no de *Activity*.

```
public class Housed extends ListActivity {
```

Ilustración 4.11 Clase Housed

Otra de las funciones principales de esta actividad es gestionar los eventos que sobre la interfaz gráfica asociada se produzcan, para este fin, se crean métodos dentro de la clase, que permiten el manejo de los diferentes eventos que se pueden producir, como por ejemplo, clic en uno de los botones de la interfaz o en un objeto de la lista.

```
public void onListItemClick(ListView parent, View v, int position, long id)
```

Ilustración 4.12 Manejador de click en la lista de localizaciones

Con tal de gestionar este tipo de eventos hemos de asignar al elemento de la interfaz un manejador de eventos, que será el método que se inicie al producirse el evento.

```
Button  button_back,
```

Ilustración 4.13 Declaración de botón back

Primero declararemos el tipo de Widget que queremos manejar. Seguidamente a esto, vamos a referenciar el recurso desde la actividad y posteriormente asignaremos el manejador que vamos a utilizar.

```
button_back = (Button) findViewById(R.id.boton_atras);
```

Ilustración 4.14 Capturar referencia del botón

```
button_back.setOnClickListener(pressButton);
```

Ilustración 4.15 Instalación del manejador que utilizará

Por último declaramos el método que gestionará los eventos y dentro del mismo declaramos el método del evento que vamos a gestionar e implementaremos las acciones del mismo.

```
private OnClickListener pressButton = new OnClickListener()
{
    public void onClick(View v)
```

Ilustración 4.16 Método que gestiona los eventos producidos

MainActivity es también la encargada de cargar los datos de entrada y mostrarlos, para ello se declara un método **"Load"**, en el cual se inicializara y llamará a nuestro manejador, tal y como se explica en el punto anterior. Otra de las funciones de este método, es la declaración de la variable que recibirá dichos datos, así como, el manejo de las diferentes excepciones que puedan ocurrir en el proceso de obtención de datos de entrada.

Como se expuso en el punto anterior, la actividad principal tiene otra función principal, debe enviar y recibir datos de sus sub-actividades.

Para la obtención de datos de las sub-actividades, especialmente de la actividad *ChangeURLActivity*, se debe manejar este evento, existe un método "**on*Activity*Result**" que nos permite la realización de esta tarea de manera sencilla. En el podemos extraer desde el *Bundle* los datos que la sub-actividad nos envía. Lógicamente, antes de finalizar el método si la URL proporcionada por el cliente cambió se debe llamar al método "**Load**", que se encargará de extraer los datos de la entrada y mostrarlos correctamente como hemos explicado anteriormente.

En cuanto al envió de datos a las actividades, utilizaremos la clase *Intent* que nos permite insertar datos en su estructura y recuperarlos en la sub-actividad muy sencillamente, el único inconveniente de este sistema, es que solo permite enviar tipos primitivos. Por este motivo la comunicación entre *MainActivity* y las actividades de los dispositivos se realizarán mediante *deviceId* del elemento.

4.3.1.1. MyListAdapter

La función de esta clase es básicamente el manejo de la lista de objetos que se le envía, extiende de *BaseAdapter* y vamos a sobrescribir varios de los métodos, con tal de adaptarla a nuestras necesidades. Vamos a explicar varias de las modificaciones realizadas en dicha clase.

En primer lugar, nuestro *ListAdapter* recibe, al ser inicializado, datos tales como los elementos que debe mostrar en la lista y los objetos de nuestra estructura de entrada, para que pueda trabajar con ellos, estos datos están separados en dos variables (*map_device*, *map_items*) para agilizar la búsqueda.

Una vez, disponemos de los datos, otro de los métodos importantes que se deben reescribir es *getView*, donde nuestro Adaptador decide que *Layout* debe mostrarse en cada uno de los elementos de la lista. Como veremos más adelante, las localizaciones y dispositivos van acompañados de un icono a la izquierda, en cambio los elementos predeterminados como *Sublocation* o Device no tienen este icono. Disponemos pues de dos tipos de *Layout*, por tanto debemos realizar un filtro de elementos de la lista, para diferenciar los estáticos de los dinámicos.

Otra de las características de los elementos de la lista estáticos es que no deben ser clickables, para ello, existe un método en el Adaptador llamado *isEnable*, donde podemos deshabilitar los el click en los elementos estáticos de nuestra lista.

4.3.2. Change URL *Activity*

Esta actividad es la responsable de ofrecer al usuario de la aplicación, la posibilidad de cambiar de escenario o de estructura domótica. Para ello, se muestra una sencilla interfaz de usuario que da la opción de introducir el URL donde se aloja el fichero que contiene la estructura y que será la entrada de datos de la aplicación.

Por tanto, esta actividad extiende de *Activity* y su única labor es recoger los datos que el usuario introduce en el formulario, para posteriormente al terminar enviarlos de vuelta a su *MainActivity*.

4.3.3. Actividades de los diferentes dispositivos

Anteriormente, hemos enumerado los diferentes dispositivos que soporta la aplicación, como es lógico, cada uno de ellos tiene unas funciones y características. Con tal de satisfacerlas, se crea una actividad por cada uno de los diferentes dispositivos de los que disponemos.

La actividad principal o *MainActivity* será la encargada de iniciar una u otra actividad, según el dispositivo con el que el usuario desee interactuar.

Estas actividades reciben dos parámetros de la *MainActivity* antes mencionados (*deviceID* y *locName*), con estos datos, la actividad debe buscar el dispositivo del que se trata y mostrar la interfaz de usuario correspondiente, así como manejar las diferentes opciones de esta interfaz y coordinar el tipo de llamadas que se hacen sobre el dispositivo.

4.4. Implementación de interfaz de usuario

En este punto se explican las diferentes partes de la interfaz de usuario que componen la aplicación, veremos qué aspecto tiene cada una de las cuales y que actividades o clases las utilizarán y porque.

4.4.1. Implementación de interfaz de ubicaciones

Para el desarrollo de esta parte de la interfaz, hemos decidido mostrar los datos en una lista, además de esto se diferenciarán entre filas estáticas y dinámicas, siendo diferentes los aspectos que las componen.

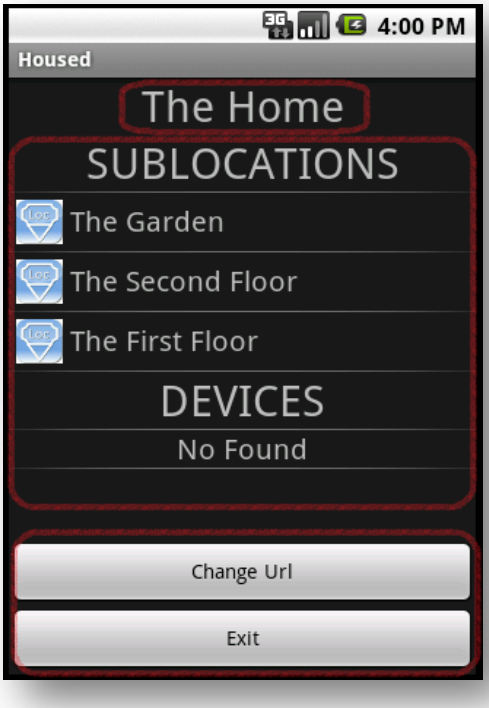


Ilustración 4.17 Interfaz de ubicaciones

Como se puede observar en la imagen, la *Layout* principal de esta interfaz se compone de: Un título, la lista de elementos y los dos botones. A esta *Layout* principal la hemos llamado *location.xml*.

Ahora bien, dentro de la lista de elementos, podemos observar que hay dos *Layout* diferentes para la composición de filas.

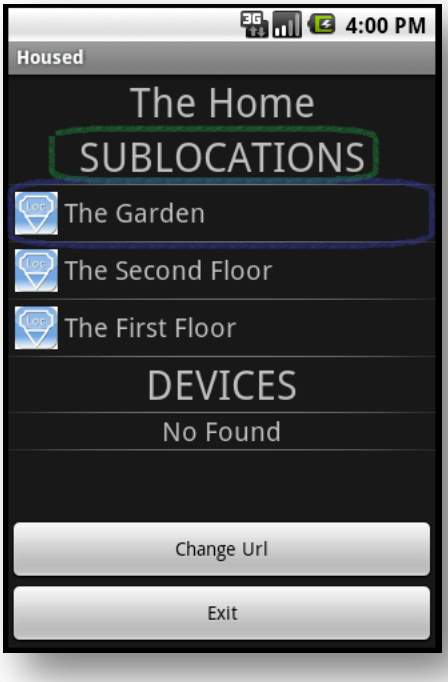


Ilustración 4.18 Elementos dinámicos y estáticos de la interfaz de ubicaciones

Para las filas estáticas, creamos un *Layout* llamado *fila.xml*, está formado por un simple *TextView*. En cambio, para las filas dinámicas que contendrán un icono según el tipo, creamos un *Layout* diferente llamado fila_icono.xml, en el cual dentro de un *LinearLayout*, tendremos un *ImageView* y un *TextView*.

En el caso del *Layout location.xml*, será utilizado e introducido por la *MainActivity*, en cambio, la decisión de que *Layout* de fila se debe mostrar caerá sobre el adaptador de la lista.

4.4.2. Implementación de interfaz de dispositivos

Teniendo en cuenta que cada uno de los dispositivos tiene diferentes opciones, debemos personalizar cada uno de ellos. Aun así, en esencia, la interfaz de usuario para los dispositivos comparte gran parte del total de la interfaz, ya que en todos debemos mostrar las características de los mismos dispositivos y la única diferencia son las acciones que se pueden realizar sobre ellos.

Ilustración 4.19 Interfaz de dispositivo Switch

Más adelante en el punto Interfaz de dispositivos, se expone de una manera más visual, el tipo de interfaz de cada uno de los dispositivos soportados por la aplicación, donde veremos los diferentes recursos de interfaz utilizados para la realización de las mismas.

4.4.3. Implementación de interfaz de cambio de ruta

En este caso la interfaz de usuario para esta función es bastante simple, se trata de un simple formulario, donde el usuario puede ver la ruta por defecto del archivo de entrada de datos y tiene la opción de cambiarlo.

Para ello se introducen dentro de un *LinearLayout* una tabla para organizar los datos y dentro de la misma, tanto *TextView* como EditText, además y ya fuera de la tabla, se introducen dos botones con opción para guardar los cambios o cancelarlos.

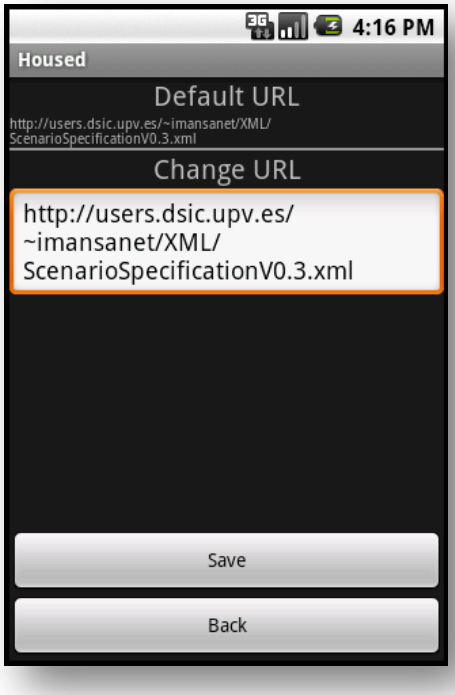


Ilustración 4.20 Interfaz de cambio de ruta

4.5. Comunicación con la plataforma domótica

Otra de las características de la aplicación es la posibilidad de interactuar con los dispositivos de la plataforma domótica que esta recibe, para ello se utilizan peticiones HTTP, las cuales serán interpretadas por un *Servlet* llamado *MuteServlet*, encargado de atenderlas y de la interacción con los dispositivos.

Para conseguir esta función, debemos introducir el protocolo en nuestra aplicación. Hemos creado métodos en cada una de las clases de los dispositivos soportados por la aplicación, mediante ellos podremos consultar el estado del dispositivo o cambiarlo. Lógicamente, estos métodos implementan el protocolo de consulta del *MuteServlet*.

```
public String switchOn()
```

Ilustración 4.21 Método switchOn de la clase Switch

La última parte para conseguir la comunicación con la plataforma domótica, es la utilización de los métodos creados. Al iniciar una actividad correspondiente a un dispositivo, mediante la selección del mismo en la interfaz de ubicaciones, esta actividad localiza el dispositivo que corresponde a los datos que recibe de la *MainActivity*, una vez localizado, se procede a consultar los datos y el estado del mismo. Los datos, se pueden encontrar localmente, ya que son proporcionados por la entrada de la aplicación, en cambio, para el estado del mismo se llamarán a los métodos creados para este fin y mencionados anteriormente.

Como vimos en el punto anterior la interfaz de dispositivos ofrece también la posibilidad, mediante varios mecanismos de cambiar el estado del mismo, según sea su clase. Los eventos producidos en estos mecanismos se manejan, dando la funcionalidad correspondiente con el uso de los métodos de consulta de estado y modificación.

4.6. Manifiesto de la aplicación

Este archivo es de importancia dentro de la aplicación, ya que en él se declararán tanto los componentes de Android utilizados como los recursos del sistema que la aplicación necesita.

```
<activity android:name=".Activity.Housed"
          android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Ilustración 4.22 Fragmento de AndroidManifest.xml

En nuestro caso, el manifiesto contendrá la declaración de las actividades de las que se compone la aplicación, indicando el modo y las características de la ejecución de cada una de ellas. Además, en la parte final se definen los permisos de los que dispone la aplicación, en este caso, el único permiso necesario para la correcta ejecución de la aplicación es la conexión a internet.

```
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
```

Ilustración 4.23 Fragmento de permisos del AndroidManifest.xml

4.7. Otras características implementadas

4.7.1. Captura de excepciones

Es necesario capturar y distinguir entre las diferentes excepciones que se pueden producir en la vida de la aplicación, para ello, en la *Main Activity* o *Housed.java*, se ha implementado la captura de las diferentes excepciones.

Brevemente enumeraremos las excepciones que tratan la aplicación y el resultado de las mismas en la interfaz.

Encontramos, excepciones recibidas de la entrada de datos, bien producidas por una estructuración errónea del documento de entrada, o por una dirección URL

errónea. El primer caso se capturara como una *IOException* y se mostrara por pantalla un mensaje informativo del tipo de excepción producida. En el caso de las excepciones producidas por una dirección de URL errónea introducida por el usuario, se capturarán como *FileNotFoundException* e igual que en la anterior, se muestra la naturaleza de la misma.

4.7.2. Inclusión de iconos dinámicos en la interfaz de ubicaciones

A lo largo de la memoria, se ha mencionado la inclusión de iconos en la lista de ubicaciones y dispositivos que harían más intuitiva la interfaz. Con esta pequeña variedad de iconos permiten al usuario distinguir rápidamente los elementos.

Una vez introducidos en el proyecto, en la sección de */res/drawable* ya se pueden referenciar desde cualquier lugar. En nuestro caso, la clase responsable de dibujar el icono correcto en cada uno de los elementos será, el adaptador de la lista de elementos (*MyListAdapter.java*). Cuando se recibe el elemento, se consulta el tipo del mismo y se dibuja el icono asociado en la fila correspondiente.

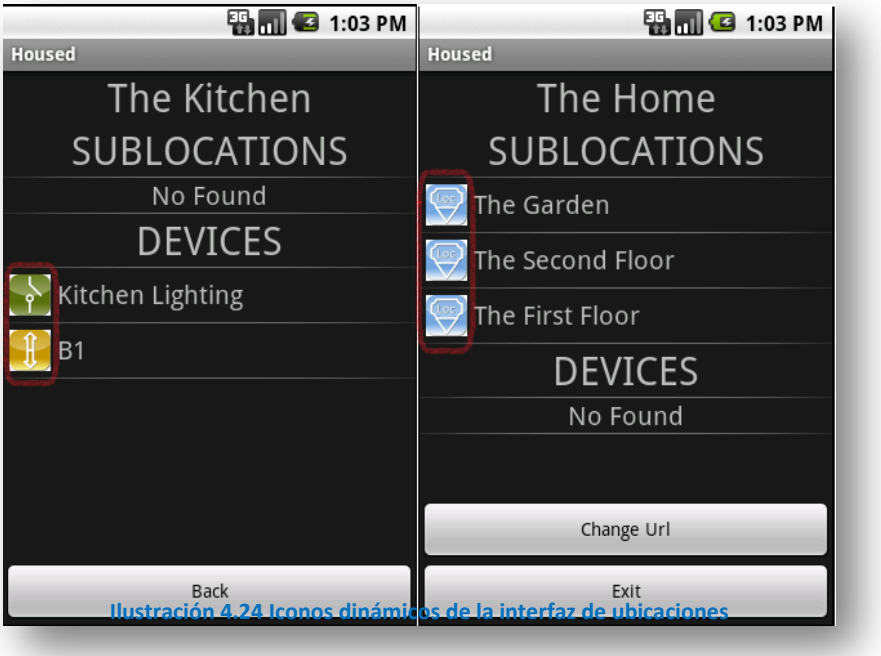


Ilustración 4.24 Iconos dinámicos de la interfaz de ubicaciones

4.7.3. Camino navegacional

Con la finalidad de que el usuario de la aplicación pueda navegar como el desee entre las localizaciones y dispositivos que presenta la estructura domótica hemos implementado un camino navegacional en la actividad principal, que tiene como objetivo seguir los pasos del usuario, para en cualquier momento poder volver sobre ellos.

Para ello, se crea un *ArrayList* donde según el botón de navegación pulsado se añadirán o eliminarán elementos de la lista. Lógicamente, estos elementos hacen referencia al dispositivo o localización pulsada.

Otra de las características que conseguimos con esta implementación, es poder mostrar al usuario distintas posibilidades según sea su situación en la estructura, de este modo, si el usuario se encuentra en el *ParentLocation* el botón *Back* se convierte en un botón de salida de aplicación y se muestra el botón de cambio de URL, que no se muestra en el resto de la estructura.

4.8. Mejoras aplicables

4.8.1. Lista de URL conocida

Actualmente, la aplicación solo recuerda la última URL introducida por el usuario en la sección *ChangeURL*, para mejorar este aspecto, una de las opciones propuestas por el autor es la inclusión de un historial local de URL visitadas, que sea recordado siempre. El se puede presentar como una lista de elementos para que el usuario pueda navegar por ellos y seleccionarlos. Con esta mejora tendríamos una aplicación más sencilla y usable.

4.8.2. Mejora en el tratado de excepciones de la entrada de datos

Gracias a la sencilla captura de excepciones implementada, podemos distinguir entre los errores que se producen en la entrada de datos. Con tal de dar mayor información al usuario, se puede mejorar este aspecto concretando más la naturaleza del error e informando al usuario donde se ha producido y que puede hacer para solucionarlo.

4.8.3. Autentificación

Uno de los aspectos más comunes de las aplicaciones es la autentificación, en este caso, tal y como se presenta la aplicación, no dispone de esta funcionalidad. La inclusión de esta funcionalidad proporcionaría ciertas ventajas para el usuario, ya que esto unido a la lista de URL conocidas, proporciona al usuario una aplicación más sencilla, segura y personalizada.

Los problemas que nos plantea esta funcionalidad, se refieren al cuando y como se debe implementar. En cuanto al cuando realizar la autentificación, la opción más común y cómoda es introducirla al inicio de la aplicación. Para la implementación,

pensamos en el formulario de autentificación, se puede hacer mediante un formulario que salte al inicio de la misma y donde el usuario debe rellenar los diferentes campos. En cambio, para que la aplicación verifique los datos necesitamos que sean persistentes, una de las opciones que Android nos brinda es la introducción de una pequeña base de datos.

Hay que tener en cuenta, que la implementación de esta mejora, haría más complicada y pesada la aplicación.

4.8.4. Interfaz visual

Para el usuario final es más atractiva una interfaz visual que una textual. Actualmente la aplicación dispone de una interfaz textual, por tanto, una de las mejoras que se podrían realizar es la inclusión de navegación por planos o imágenes, de forma que el usuario seleccionando partes de estas navegue por la estructura.

Para llevar a cabo esta mejora, debemos tener en cuenta los campos que referencian a las imágenes en el documento de entrada de la aplicación, crear nuevos *layout* donde aplicaremos las características necesarias a cada una de las imágenes mostradas.

Por último, se podría dar al usuario la opción de cambiar de interfaz textal a visual en cualquier momento de la navegación por la estructura de la aplicación.

5. Prototipo de aplicación

Una vez creado el prototipo de aplicación, en este capítulo presentaremos el resultado de la aplicación, veremos cada una de las interfaces de dispositivos y se realizan diferentes pruebas sobre ella con un cierto caso práctico.

5.1. Interfaz de dispositivos

Anteriormente, en el punto Implementación de interfaz de dispositivos se expone la necesidad de incluir una interfaz personalizada para cada uno de los dispositivos soportados por la aplicación. Ahora, veremos una serie de capturas que ilustrarán el resultado de estas interfaces de las que dispone el prototipo de aplicación.

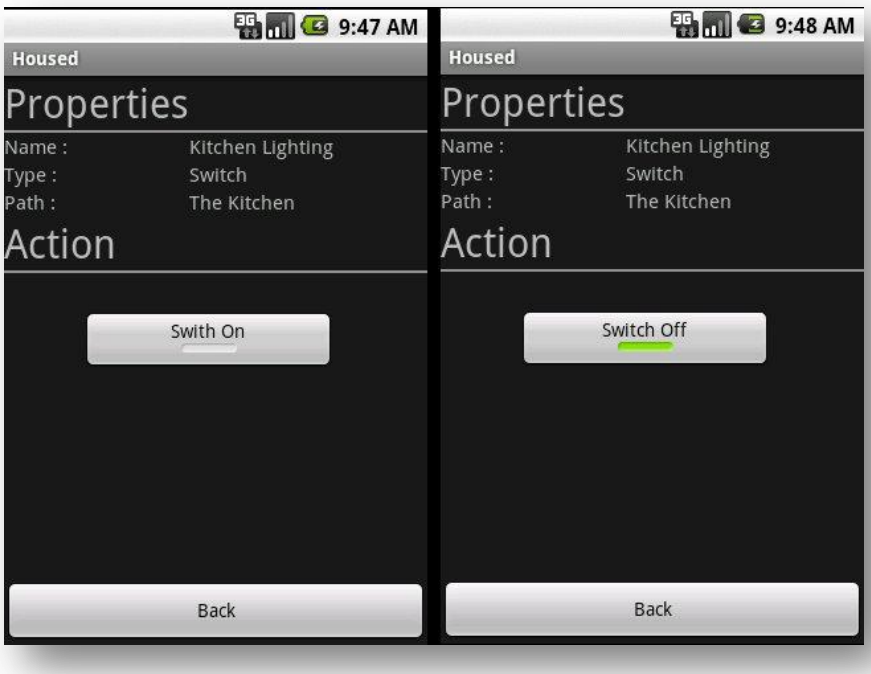


Ilustración 5.1. Interfaz del dispositivo Switch

Para empezar, en la Ilustración 5.1 podemos ver la interfaz del dispositivo Switch en los dos posibles estados del dispositivo. Estos estados se representan gracias al *ToggleButton* que declaramos en el *layout* que reside */res/layout/swtich_device.xml*.

En cuanto a los dispositivos *Dimmer*, disponen de una interfaz muy similar, su única diferencia es la configuración del *SeekBar* que muestra el progreso y el formato en el que se muestra este. En caso del dispositivo PercentualDimmer, el progreso es de 0-100 y se representa en tanto por cien, AngularDimmer abarca de 0-360 y se representa en grados, por ultimo HexDimmer tiene un rango de 0-255.

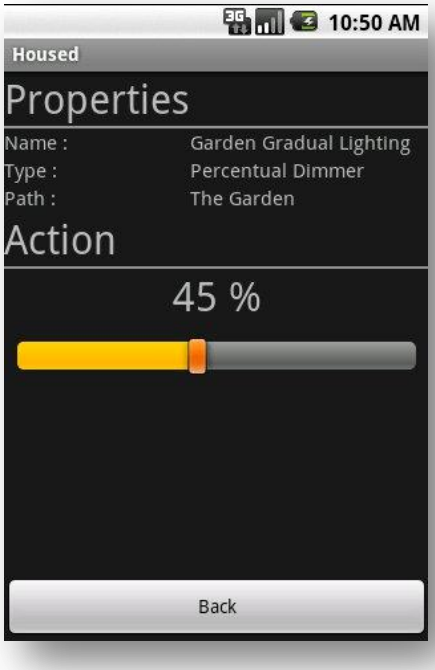


Ilustración 5.2 Interfaz del dispositivo PercentualDimmer

Seguimos con el dispositivo llamado Pulse, cuya interfaz está dotada de un Button con el texto "*Press*" el cual activa el dispositivo al pulsarlo, enviando una señal como se ha explicado a lo largo del documento.

Ilustración 5.3 Interfaz del Dispositvo Pulse

La interfaz del dispositivo Sensor dispone de un Text*View* que muestra textualmente el estado actual del sensor, además de esto, se añade un botón cuya función es actualizar el estado del dispositivo y reflejarlo en el Text*View*.

Ilustración 5.4 Interfaz del dispositivo Sensor

Por último, nos centramos en la interfaz del dispositivo VerticalMovement, distinguimos tres botones con funcionalidades distintas. Las acciones son Up, Stop, Down, cada cual interactúa con el controlador tal y como expresa el nombre del Button.

Ilustración 5.5 Interfaz del dispositivo VerticalMovement

5.2. Demostración del prototipo

En este punto veremos unos casos prácticos sobre el prototipo de aplicación desarrollada en el proyecto. Veremos las diferentes opciones de las que dispone el usuario y cuál es el aspecto final de las mismas.

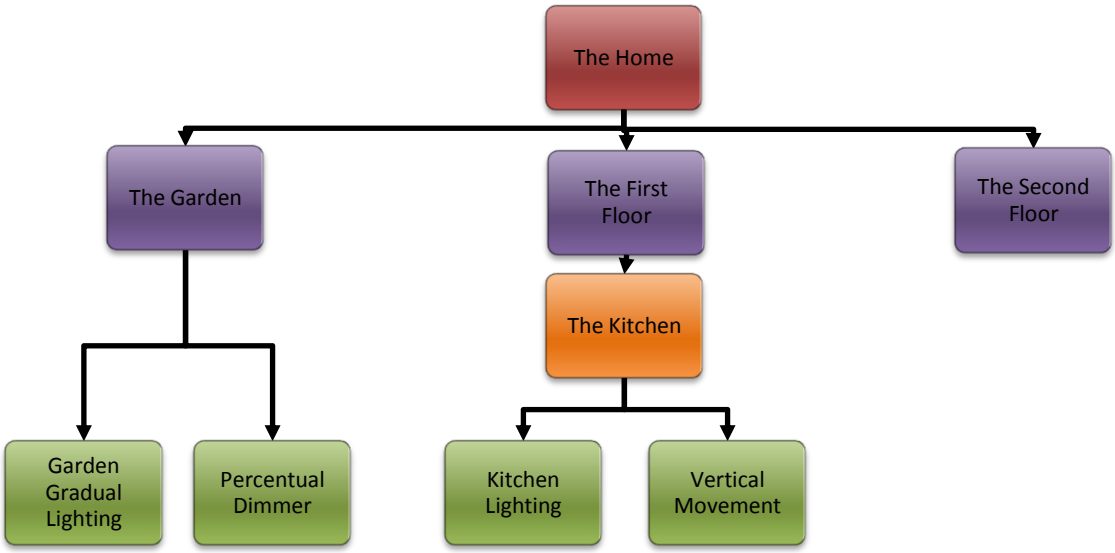


Ilustración 5.6 Estructura domótica ejemplo

Para estos casos prácticos, hemos utilizado el documento XML ScenarioSpecificationV0.3.xml, que se aloja un servidor proporcionado. Tiene una estructura como la que podemos observar en la imagen.

5.2.1. Navegación por ubicaciones y dispositivos

En este primer ejemplo, veremos la navegabilidad disponible entre los diferentes niveles de la estructura, en concreto vamos a consultar los dispositivos que se encuentran en la cocina.

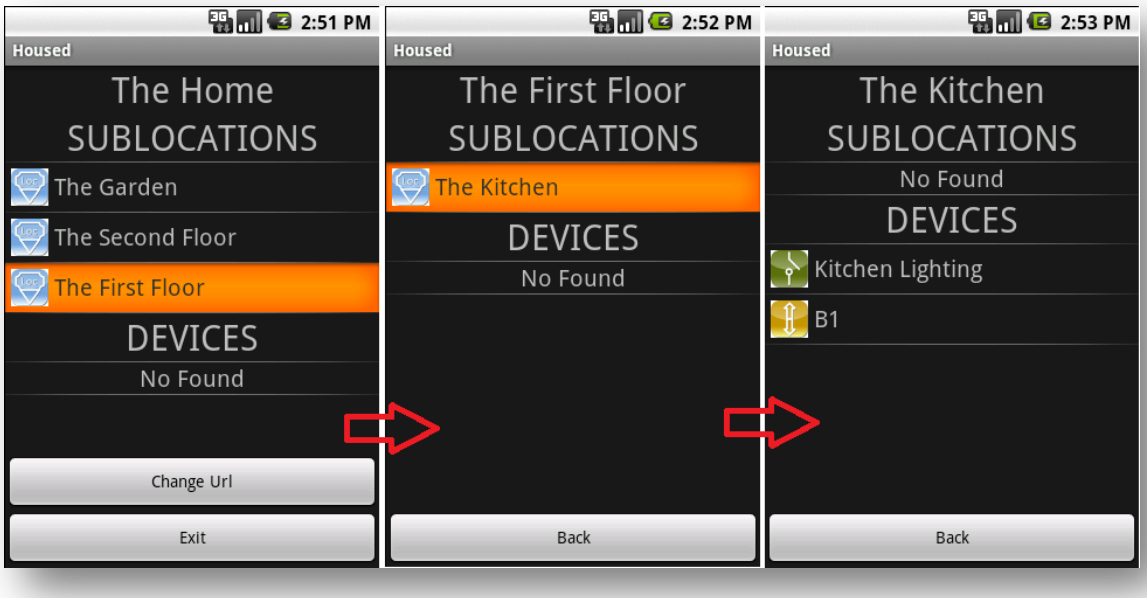


Ilustración 5.7 Ejemplo navegación entre localizaciones y dispositivos

Solo con iniciar la aplicación encontramos la lista de las ubicaciones y dispositivos de la *Location Parent*, como vemos en la figura de la derecha. Tras seleccionar el primer piso, encontramos las localizaciones y dispositivos que contiene. De la misma manera, al pulsar sobre la cocina, nos muestra que no dispone de sublocalizaciones y si de *Devices*, como son, un dispositivo Switch y un *Vertical Movement*.

Como vemos esta interfaz es fácil e intuitiva, por lo tanto, el usuario no debe tener demasiadas dificultades para navegar por su estructura domótica propia.

5.2.2. Interacción con dispositivos

En este segundo caso práctico vamos a mostrar como encender la luz de la cocina (*Switch Actuator*). Partiendo de la situación anterior, disponemos de dos dispositivos con los que podemos interactuar, en nuestro caso seleccionamos la luz de la cocina y tras estos, se cargará la interfaz del dispositivo seleccionado.

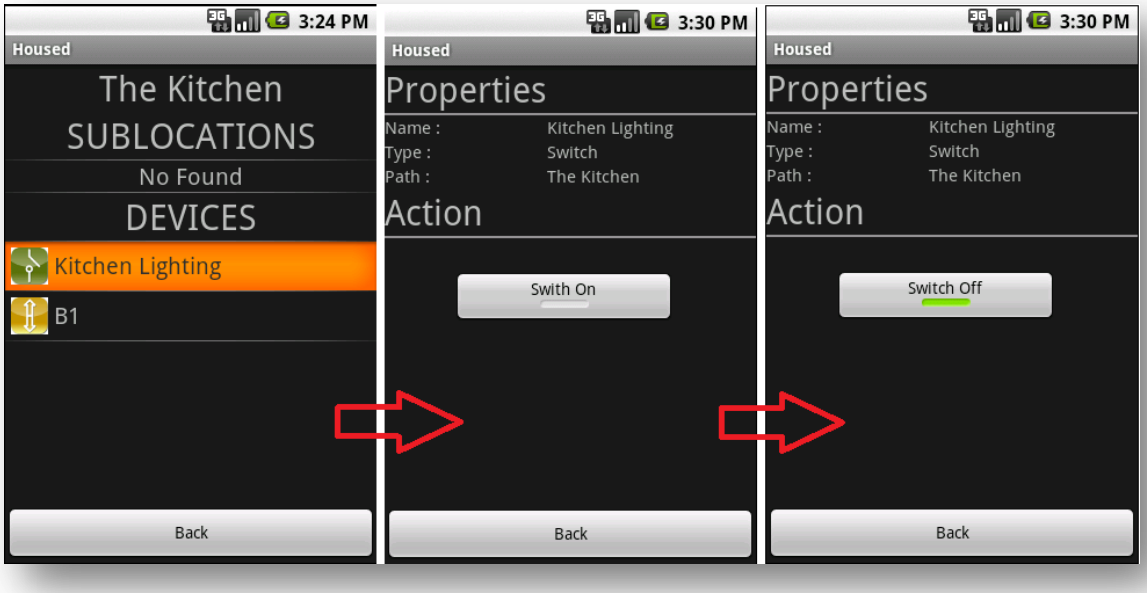


Ilustración 5.8 Ejemplo interacción con un dispositivo Switch

Como se muestra en la imagen, tenemos información sobre las características del dispositivo, además de ello en la parte media encontramos el botón para encender o apagar, pulsando sobre él se cambia el estado del dispositivo y se muestra el estado actual en dicho botón.

5.2.3. Cambio de URL de entrada

Para finalizar, veremos el camino que se debe seguir para cambiar el documento XML de entrada de la aplicación y poder guardarlo en la memoria de la aplicación.

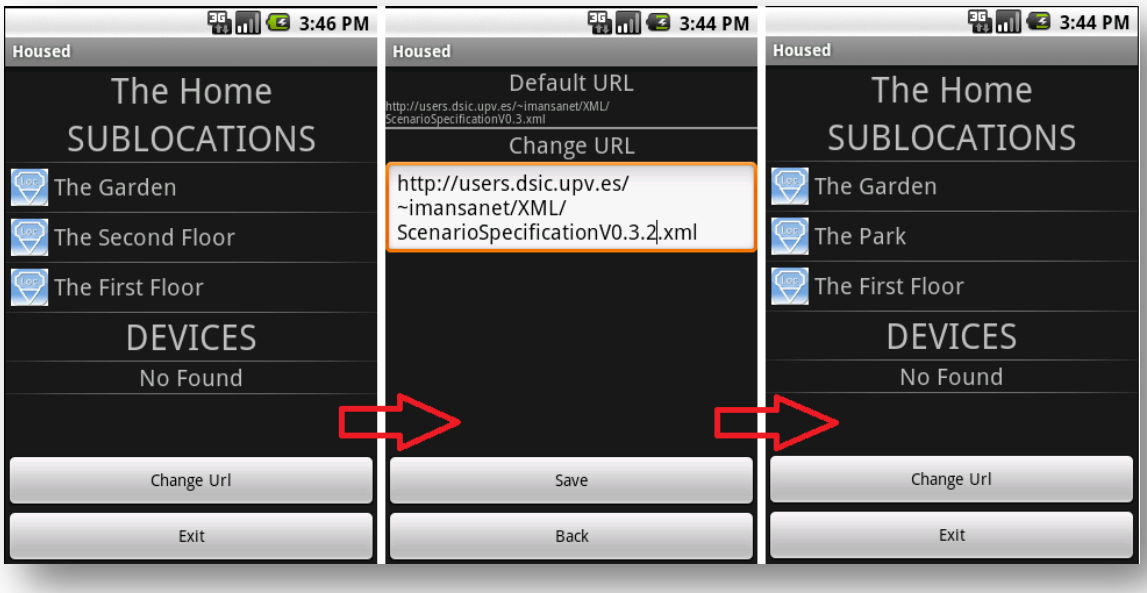


Ilustración 5.9 Ejemplo de cambio de dirección del documento de entrada

Al iniciar la aplicación encontramos el botón "*Change Url*", seleccionamos la opción y nos lleva a un pequeño formulario, donde se muestra el *url* actual y un pequeño recuadro para cambiar dicha dirección, una vez hechas las pertinentes modificaciones al seleccionar "*Save*", la aplicación procederá a analizar los datos y tomar esta dirección como dirección por defecto.

Como bien ilustra la imagen, una vez realizado el proceso, automáticamente nos encontramos ante la interfaz principal de la aplicación que muestra los datos del nuevo documento de entrada.

6. Conclusiones y trabajos futuros

En este apartado el alumno expone sus impresiones sobre la realización del proyecto final de carrera y como este ha contribuido a asentar conceptos y técnicas obtenidas a lo largo de la carrera de Ingeniería Técnica en Informática de Sistemas. Se expondrá el trabajo realizado, la valoración personal del mismo, así como trabajos futuros que se pueden realizar.

6.1. Valoración personal

El proyecto realizado ha contribuido a la introducción del alumno en diferentes campos que son importantes hoy en día. Por una parte, el hecho de desarrollar una aplicación en una de las plataformas móviles más importantes y jóvenes, como es Android, ha proporcionado la motivación necesaria para realizar el proyecto y después de esto seguir desarrollando e investigando sobre esta plataforma.

Por otra parte, se introduce al alumno en el campo de la domótica, una parte importante de la tecnología actual. Así como del tratamiento de documentos XML en lenguaje Java, parte importante por el hecho de que este metalenguaje se utiliza de manera frecuente en la comunicación entre diferentes aplicaciones o dentro de las mismas, como hemos visto en Android y sus interfaces.

Los obstáculos encontrados en la realización del proyecto, han sido superados gracias a la motivación del alumno al realizar un proyecto que fomentaba en él un gran interés y los conocimientos adquiridos a lo largo de la carrera.

6.2. Trabajos futuros

Este trabajo realizado se puede continuar completando funcionalidades que al final la aplicación no ha soportado. Así pues en un futuro, partiendo de la base de esta

aplicación, se podrían desarrollar algunas de las mejoras expuestas en el punto Mejoras aplicables, como pueden ser:

- Lista de URL conocida.
- Mejora en el tratado de excepciones de la entrada de datos.
- Autentificación.
- Interfaz visual.

Estos aspectos harían que la aplicación sea más madura, segura e intuitiva para el usuario. Como mejora adicional se contempla el hecho de actualizar la aplicación a las últimas versiones de la plataforma, ya que su evolución es constante y rápida, revisando el perfecto funcionamiento de todos los aspectos de la aplicación y manteniendo su total compatibilidad.

7. Referencias

Las siguientes fuentes de información fueron empleadas durante la realización del proyecto.

[1] *DiMarzio, J.F.* **"Android A Programmer's Guide"**. McGrawHill. United States of America, 2008.

[2] *Rogers, Rick.* **"Android Application Development"**. O'Reilly Series. United States of America, 2009.

[3] *Rogers, Reto.* **"Professional Android Application Development"**. Wrox. Canadá, 2009.

[4] *Gil Pascual, Miriam.* **"Implementación de aspectos de eficiencia energética en el hogar digital: un caso práctico"**. Valencia, 2008.

[5] http://developer.android.com

[6] http://source.android.com/

[7] http://www.android-spa.com/

Anexo A: Mute*Servlet*

Para la correcta comunicación entre la aplicación y los dispositivos se hace uso de *MuteServlet*. *MuteServlet* es un *Servlet* que atiende peticiones de interacción con los dispositivos del *tunnel* de *calimero*. Esta implementado como un *bundle OSGi* que se instalará y ejecutará en la misma configuración de ejecución que el resto de dispositivos del túnel.

Se utilizará de la siguiente manera.

En primer lugar crearemos un petición http (*http request*) a la dirección donde oye el *Servlet* pasándole como parámetros el ID y tipo del dispositivo, así como la acción a tomar sobre el dispositivo y el nuevo valor en caso que sea necesario.

A continuación se exponen los tipos de parámetros que admite el *Servlet*

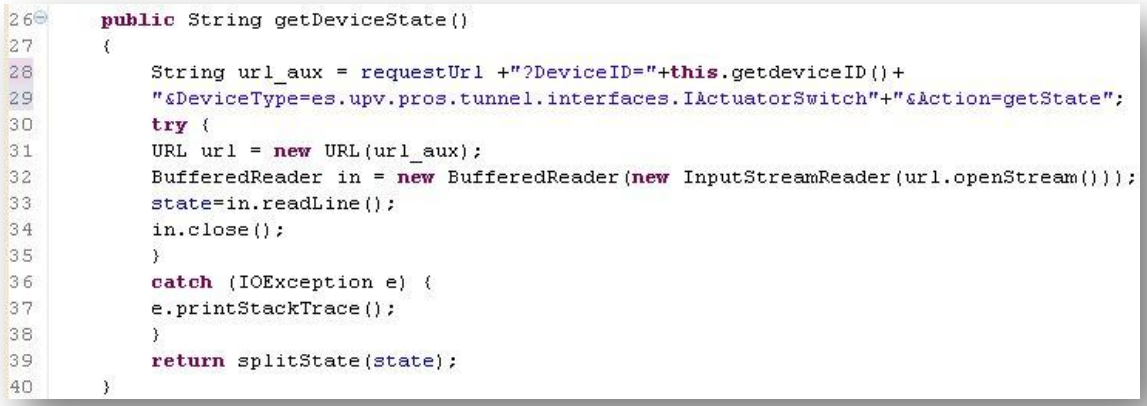
- DeviceID → El ID del dispositivo contra el que queremos interactuar.
- DeviceType →El tipo del dispositivo. Puede ser uno de los siguientes:
    - es.upv.pros.tunnel.interfaces.IActuatorSwitch
    - es.upv.pros.tunnel.interfaces.IActuatorPulse
    - es.upv.pros.tunnel.interfaces.IActuatorVerticalMovement
    - es.upv.pros.tunnel.interfaces.IActuatorPercentualDimmer
    - es.upv.pros.tunnel.interfaces.IActuatorAngularDimmer
    - es.upv.pros.tunnel.interfaces.IActuatorHexDimmer
    - es.upv.pros.tunnel.interfaces.ISensor
- Action →La operación a ejecutar. En función del tipo de dispositivo contra el que se actué, tendremos distintos tipos de acción:
    - es.upv.pros.tunnel.interfaces.IActuatorSwitch
        - switchOn
        - switchOff
    - es.upv.pros.tunnel.interfaces.IActuatorPulse
        - pulse
    - es.upv.pros.tunnel.interfaces.IActuatorVerticalMovement

        - moveUp
        - moveDown
        - stop
    - es.upv.pros.tunnel.interfaces.IActuatorPercentualDimmer
        - setPercentualValue
        - on
        - off
    - es.upv.pros.tunnel.interfaces.IActuatorAngularDimmer
        - setAngularValue
    - es.upv.pros.tunnel.interfaces.IActuatorHexDimmer
        - setHexValue
    - es.upv.pros.tunnel.interfaces.ISensor
        - getState
- Value → En los dispositivos de tipo dimmer, existen operaciones que requieren un valor. Este deberá ser de tipo entero :
    - es.upv.pros.tunnel.interfaces.IActuatorPercentualDimmer
        - de 0 a 100
    - es.upv.pros.tunnel.interfaces.IActuatorAngularDimmer
        - de 0 a 360
    - es.upv.pros.tunnel.interfaces.IActuatorHexDimmer
        - de 0 a 255

También cabe notar que el *Servlet* responderá con el valor actual del dispositivo con el estado tras la operación. La respuesta será del tipo "*state=VALOR_ACTUAL*"

Una vez sabemos cómo se utiliza, veremos la implementación de las peticiones y la opción de los datos de las respuestas del mismo.

En cada uno de los dispositivos soportados por la aplicación, en la clase que hace referencia a ellos, se implementan métodos que enviarán peticiones al *MuteServelt*. Para ello estos métodos construyen la URL de petición de esta manera:



```
26    public String getDeviceState()
27    {
28        String url_aux = requestUrl +"?DeviceID="+this.getdeviceID()+
29        "&DeviceType=es.upv.pros.tunnel.interfaces.IActuatorSwitch"+"&Action=getState";
30        try {
31        URL url = new URL(url_aux);
32        BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()));
33        state=in.readLine();
34        in.close();
35        }
36        catch (IOException e) {
37        e.printStackTrace();
38        }
39        return splitState(state);
40    }
```

Ilustración 0.1. Fragmento de código de la clase Switch, método getDeviceState

Como vemos en la imagen la URL está formada por las variables que obtenemos de la estructura de datos y que el *MuteServlet* necesita como es por ejemplo *DeviceID*. El método que arriba exponemos esta creado para solicitar el estado del dispositivo, por ello en la acción a realizar se introduce la opción *getState*. Este es el aspecto que tendría la URL una vez construida.

```
http://pervasive.dsic.upv.es:8080/muteServlet/?DeviceID=L1&DeviceType=
es.upv.pros.tunnel.interfaces.IActuatorSwitch&Action=getState
```

Una vez construida la URL, llamamos a la misma, según nos indica la documentación del mute *Servlet* y obtenemos mediante la variable state la información que nos devuelve. Ahora solo queda procesar esa información y mostrar en la interfaz el resultado de esta llamada.

Anexo B: MyListAdapter, Construcción de una lista personalizada

A lo largo de la memoria, se describe la importancia de esta clase en la construcción de la lista de ubicaciones que cargará la actividad principal y que muestra las ubicaciones y los dispositivos del escenario. En este anexo, veremos a nivel de código, como construimos dicha lista personalizada.

La clase *MyListAdapter* es una extensión de la clase *BaseAdapter*, en la cual vamos a extender los métodos de esta y adaptarla para lograr la funcionalidad deseada. Esta clase nos proporciona cada una de las filas de la lista. Para ello dispone de unos métodos, el más importante para nuestra función es el llamado *getView*.

Android llamará al método *getView* de nuestro adaptador para obtener la vista de cada una de las filas. En este caso, tendremos dos tipos de filas: fila con contenido e icono y otra con solo contenido. Previamente, se han creado dos *layout* que representan debidamente las filas, llamadas *fila.xml* y *fila_icono.xml* cuyo código podemos ver a continuación.

```
4  <TextView
5      xmlns:android="http://schemas.android.com/apk/res/android"
6      android:id="@+id/TextoDeFila"
7      android:textSize="30sp"
8      android:layout_width="fill_parent"
9      android:layout_height="fill_parent"
10     android:gravity="center"/>
11
12
```

Ilustración 0.1. Código fila.xml

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:id="@+id/LinearLayout01"
android:orientation="horizontal">

<ImageView
android:id="@+id/ImageView01"
android:layout_width="wrap_content"
android:layout_height="fill_parent"
android:layout_margin="5sp"
/>

<TextView
    android:id="@+id/TextoDeFila"
    android:textSize="20sp"
    android:gravity="center_vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"/>

</LinearLayout>
```

Ilustración 0.2. Código fila_icono.xml

En el código de fila.xml observamos el tipo Text*View* donde definimos su estilo y forma. De este modo la fila que solo contenga contenido se mostrará de este modo. En cambio, en fila_icono.xml encontramos la definición de un LinearLayout de orientación horizontal y dentro del mismo se introducen tanto un Image*View* donde mostraremos el icono correspondiente al dispositivo o localización y el Text*View* que contendrá el contenido.

```
64    public View getView(int position, View convertView, ViewGroup parent)
65    {
66        TextView tv;
67        View row_view;
68        String item = items.get(position);
69        if(!(items.get(position).equals("SUBLOCATIONS") || items.get(position).equals("DEVICES")
70            || items.get(position).equals("No Found")
71            || items.get(position).equals("Incorrect Syntaxis XML")
72            || items.get(position).equals("Wrong URL")))
73        {
74            row_view = View.inflate(context, R.layout.fila_icono, null);
75            tv = (TextView)row_view.findViewById(R.id.TextoDeFila);
76            ImageView icon=(ImageView)row_view.findViewById(R.id.ImageView01);
77            icon.setImageResource(R.drawable.device);
78            String id = items.get(position);
79            if(map_items.containsKey(id))
80            {
81                if(map_items.get(id).getlocationName().equals(""))
82                {tv.setText(id);}
83                else
84                {tv.setText(map_items.get(id).getlocationName());}
85                icon.setImageResource(R.drawable.location);
86            }
87            else if(map_devices.containsKey(id))
88            {
89                if(map_devices.get(id).getpropiety("NAME") != null)
90                {tv.setText(map_devices.get(id).getpropiety("NAME"));
91                }else
92                {tv.setText(id);}
93                /*Introducir un icono dependiendo del tipo de dispositivo*/
94                if(map_devices.get(id).getdeviceType().equals("DEVICE_TYPE_SWITCH"))
95                { icon.setImageResource(R.drawable.switch_actuator); }
96                else if(map_devices.get(id).getdeviceType().equals("DEVICE_TYPE_SENSOR"))
97                { icon.setImageResource(R.drawable.sensor);}
98                else if(map_devices.get(id).getdeviceType().equals("DEVICE_TYPE_PULSE"))
99                { icon.setImageResource(R.drawable.pulse_actuator);}
100               else if(map_devices.get(id).getdeviceType().equals("DEVICE_TYPE_PERCENTUAL_DIMMER"))
101               { icon.setImageResource(R.drawable.perc_dimmer_actuator);}
102               else if(map_devices.get(id).getdeviceType().equals("DEVICE_TYPE_ANGULAR_DIMMER"))
103               { icon.setImageResource(R.drawable.ang_dimmer_actuator);}
104               else if(map_devices.get(id).getdeviceType().equals("DEVICE_TYPE_HEX_DIMMER"))
105               { icon.setImageResource(R.drawable.hex_dimmer_actuator);}
106               else if(map_devices.get(id).getdeviceType().equals("DEVICE_TYPE_VM"))
107               { icon.setImageResource(R.drawable.vm_actuator);}
108               else
109               { icon.setImageResource(R.drawable.device);}
110           }else{tv.setText(item);}
111       }
112       else
113       {   row_view = View.inflate(context, R.layout.fila, null);
114           tv = (TextView)row_view.findViewById(R.id.TextoDeFila);
115           if(items.get(position).equals("No Found")
116                   || items.get(position).equals("Incorrect Syntaxis XML")
117                   || items.get(position).equals("Wrong URL"))
118           {tv.setTextSize(20);}
119           tv.setText(item);
120       }
121       return row_view;
122   }
```

Ilustración 0.3. Código del método getView la clase MyListAdapter

En primer lugar debemos comprobar qué clase de contenido estamos recuperando, para ello, en la sentencia condicional de la línea 69 se diferencia entre ellos, como vemos si se trata de un contenido estático, como pueden ser errores o cabeceras, se carga en el *View* el *layout* de la fila que solo contiene contenido (fila.xml) y se rellena el *TextView* como observamos en la línea 113. Además se comprueba si no

se trata de una cabecera, en ese caso se personaliza el tamaño y por último se retorna el *View* que espera la *MainActivity*.

Por otro lado, en caso de tratarse de una fila con contenido de dispositivos o localizaciones, comprobamos mediante condicionales de que tipo se trata, como vemos, tenemos a nuestra disposición dos variables que contienen todos los objetos que se pueden recuperar. Finalmente, si el objeto recuperado es un tipo de dispositivo, se comprueba de que tipo es este, como podemos ver en las líneas de la 94 a 107.

Con este método sobrescrito, conseguimos personalizar cada uno de los elementos que aparecen en la *ListView*, ajustando sus características al tipo de objeto que se trata. El resultado se puede observar en la Ilustración 4.24.

Anexo C: Guía de Puesta en marcha de la aplicación

En este anexo veremos una breve guía de los componentes y pasos que se precisan para la puesta en marcha de la aplicación realizada.

Entorno Eclipse

Nos centramos inicialmente en la instalación del entorno de desarrollo utilizado y la incorporación al mismo de los plugins de Android necesarios. El primer paso es descargar el entorno de desarrollo Eclipse, que podemos encontrar en su página oficial (www.eclipse.org/downloads) en el apartado de descargas, en nuestro caso, hemos utilizado Eclipse IDE for Java EE Developers. Una vez descargado y descomprimido en nuestro sistema, iniciamos la herramienta.

Instalación SDK

Para la instalación del SDK de Android, acudimos a la página para desarrolladores de Android (http://developer.android.com/sdk/index.html), en ella podemos obtener el SDK para los diferentes sistemas operativos que soporta. Mediante el SDK-Manager podemos seleccionar los paquetes que deseamos instalar, en nuestro caso, seleccionaremos la versión 1.6 de Android. Además crearemos desde el mismo las Android Virtual Device (AVD).

Plugin ADT

Iniciamos Eclipse y seleccionamos la opción ***Help*** > ***Install New Software***. En la ventana que ahora aparece ***Available Software***, hacer clic al boton ***Add…*** Aparece la ventana ***Add Site***, en el cuadro ***Name*** puede poner por ejemplo "*Android plugin*", en el cuadro ***Locatión*** ponga la siguiente *URL:* https://dl-ssl.google.com/android/eclipse/. Una vez instalado el plugin, pasamos a configurarlo, nos dirigimos al menú ***Windows*** > ***preferences*** y dentro del mismo encontraremos la sección ***Android***, donde referenciamos el directorio donde el SDK ha sido instalado en nuestro sistema.

Ejecutar aplicación

Una vez tenemos el entorno instalado y debidamente configurado. Procedemos a ejecutar la aplicación, tenemos dos opciones, emular desde eclipse creando el ADV desde el ADV Manager o desde eclipse, o introduciendo la aplicación en un terminal Android.

En el primer caso, la emulación desde Eclipse es sencilla, una vez creado el ADV mediante ***Run As...*** seleccionamos Android Application. Con ello tendremos la aplicación funcionando en el ADV creado anteriormente.

Por la otra línea, la puesta en marcha en un terminal Android conlleva la configuración de ese dispositivo, para ello debemos instalar el driver del dispositivo en el sistema y seleccionar en el terminal el modo depuración. Una vez configurado de tal manera, podremos asignar la aplicación al terminal mediante Run.