universidad del país vasco
Facultad de Informática

euskal herriko unibertsitatea
Informatika Fakultatea

TITULACIÓN: Ingeniería Informática

Cliente UCH para *smartphones* sobre Android

Alumno/a: D./Dna. Francisco Javier Logroño Aguirre
Director/a: D./Dna. Julio Abascal González

Proyecto Fin de Carrera, Julio de 2012

Agradecimientos

Agradecer a Borja Gamecho por todo el material proporcionado, por comunicarse con el director del proyecto cada vez que fuera necesario, por la rapidez de contestación de cada correo que le enviaba, por sus correcciones y propuestas.

Agradezco al Director del Proyecto, Julio Abascal por la libertad ofrecida en el desarrollo del proyecto, el material proporcionado, las correcciones e ideas, por permitirme y gestionarme un lugar de trabajo junto a su grupo de investigación, lugar en el cuál he estado muy cómodo gracias a este grupo.

También tengo que agradecer a mi pareja por escuchar tanto tiempo "Hoy no quedamos, tengo que hacer proyecto" y a mis padres que no paran de decirme "¿Aun no has terminado el proyecto?".

Gracias

Abstract

El objetivo del proyecto consiste en la creación de una aplicación cliente de UCH para Android, capaz de controlar "dispositivos objetivos". Teniendo como base el PFC de Borja Gamecho, titulado "Estudio del estándar URC y su aplicación a sistemas embebidos usando una implementación Java del UCH", proyecto que pertenece al campo de la inteligencia ambiental (AmI).

La construcción de un sistema capaz de controlar dispositivos de manera sencilla y ubicua fue la motivación para la realización de este proyecto, ya que con él, se puede conseguir un control universal de nuestro entorno, permitiendo eliminar barreras por ejemplo, para personas con discapacidad.

En este proyecto se ha estudiado el centro de control universal (UCH) junto con el protocolo URC-HTTP. También se ha investigado sobre la plataforma Android, para posteriormente usarlo como base para el desarrollo de un URC (cliente para el control del UCH y del entorno).

Como principal conclusión, el presente trabajo permite avanzar en la propuesta de encontrar un estándar capaz de controlar nuestro entorno desde cualquier *smartphone*, *tablet*,... de forma ubicua y sencilla.

Palabras clave

URC, UCH, Android, URC-HTTP

Tabla de Ilustraciones:

Capítulo 1 D.O.P.

1.1 Descripción del proyecto

El Centro de Control Universal o *Universal Control Hub* (UCH), es un *middleware* que permite la conexión de distintos dispositivos objetivos para ofrecer a un usuario una interfaz única desde la que controlarlos a todos. Se puede usar para la creación de escenarios ubicuos con multitud de dispositivos (cocina, luz, etc.) y servicios (información meteorológica, cita médica, etc.) que son accedidos por distintos usuarios desde sus propios terminales móviles.

Actualmente el *Universal Control Hub* (UCH) es capaz de generar una interfaz de usuario independiente del dispositivo de usuario a través de un navegador web. El PFC propone sustituir el navegador por una aplicación nativa de las plataformas Android (Google).

Para llevar a cabo el proyecto se debe usar el protocolo URC-HTTP, definido por los creadores de UCH, que permite acceder a la información de un determinado dispositivo compatible con URC mediante HTTP. Tras interpretar estas cabeceras se relacionan con los elementos de interfaz más apropiados de cada plataforma móvil para interactuar con los usuarios.

1.2 Objetivos del proyecto final de carrera

Objetivos principales:

- Creación de una aplicación que capture la información proporcionada por UCH.
- Mejora de la aplicación para darle un aspecto acorde con la plataforma móvil haciendo uso de los elementos de interfaz de usuario característicos.

Procedimientos:

- Obtener cabeceras HTTP desde un entorno con UCH.
- Conseguir tratar cabeceras HTTP desde una plataforma móvil.
- Estudio del protocolo URC-HTTP.
- Estudio de las interfaces de usuario en *smartphones*.
- Estudio de la plataforma ANDROID.

Objetivos secundarios:

1. Solución al descubrimiento de UCH's. Uso de un protocolo de descubrimiento por parte de la plataforma móvil.
2. Seguridad en las interfaces, actualmente no se da soporte a la seguridad en UCH.
3. Uso de HTTPS para asegurar las comunicaciones.

El proyecto se enmarca dentro de la Ingeniería Informática (15 créditos), deberá realizarse en 450 horas (15 créditos x 30 horas) cumpliendo los objetivos principales en el mejor de los casos aunque puede haber desviaciones que alarguen el número de horas. El proyecto deberá generar manuales y tutoriales sobre los temas propuestos y al menos una aplicación para una de las plataformas móviles tratadas.

1.3 Antecedentes

Se parte del proyecto fin de carrera de Borja Gamecho [Gamecho, 2009], quién realizó un estudio del estándar URC y su aplicación a sistemas embebidos usando una implementación en Java del UCH y la creación del *middleware* UCH, para trabajar a partir de este. En la búsqueda bibliográfica no se ha encontrado ningún sistema similar realizado anteriormente con *smartphones*, ninguna aplicación que use el protocolo URC-HTTP que se conecte a un *smartphone*. Además hay que tener en cuenta la dificultad añadida por la cantidad de modelos, versiones de S.O., resoluciones, puntos por pulgadas, etc. de la gran de variedad de *smartphones* basados en Android.

1.4 Método de trabajo

Procesos Tácticos

- El director del proyecto es Julio Abascal.
- La comunicación se realiza por correo electrónico, y como segunda opción se recurrirá a utilizar el teléfono móvil.

Procesos Operativos

- Para la elaboración del proyecto seguiremos el Proceso Unificado de Desarrollo de Software (PUD), utilizando un ciclo de vida iterativo e incremental, estableciendo las pautas de cada alteración en las reuniones pertinentes.
- Para la implementación se usará el entorno Eclipse.

1.5 Diagrama de descomposición de trabajo

1. Gestión:
    a. Objetivos: Definición de los distintos objetivos a cumplir en el proyecto final de carrera (ver 1.2)
    b. Planificación: (ver 1.6)
    c. Seguimiento
2. Formación:
    a. Estudio del protocolo URC-HTTP: Estudio del estándar URC y uso de la aplicación UCH.
    b. Estudio de la plataforma Android
3. Instalación y configuración:
    a. Instalación de UCH en un servidor si no lo está, así como todo lo necesario para poder tener el servidor con UCH.
    b. Instalación de Eclipse y de los *plugins* necesarios para poder desarrollar en Android.
4. Aplicación:
    a. Análisis y Diseño: Descripción de la aplicación a desarrollar.
    b. Desarrollo: Implementación de la aplicación
    c. Pruebas: Realización de pruebas de las características implementadas.
5. Cierre:
    a. Memoria: Elaboración de la memoria y manuales de uso.
    b. Presentación: Elaboración de la presentación para una defensa ante el tribunal.

Diagrama
E.D.T.

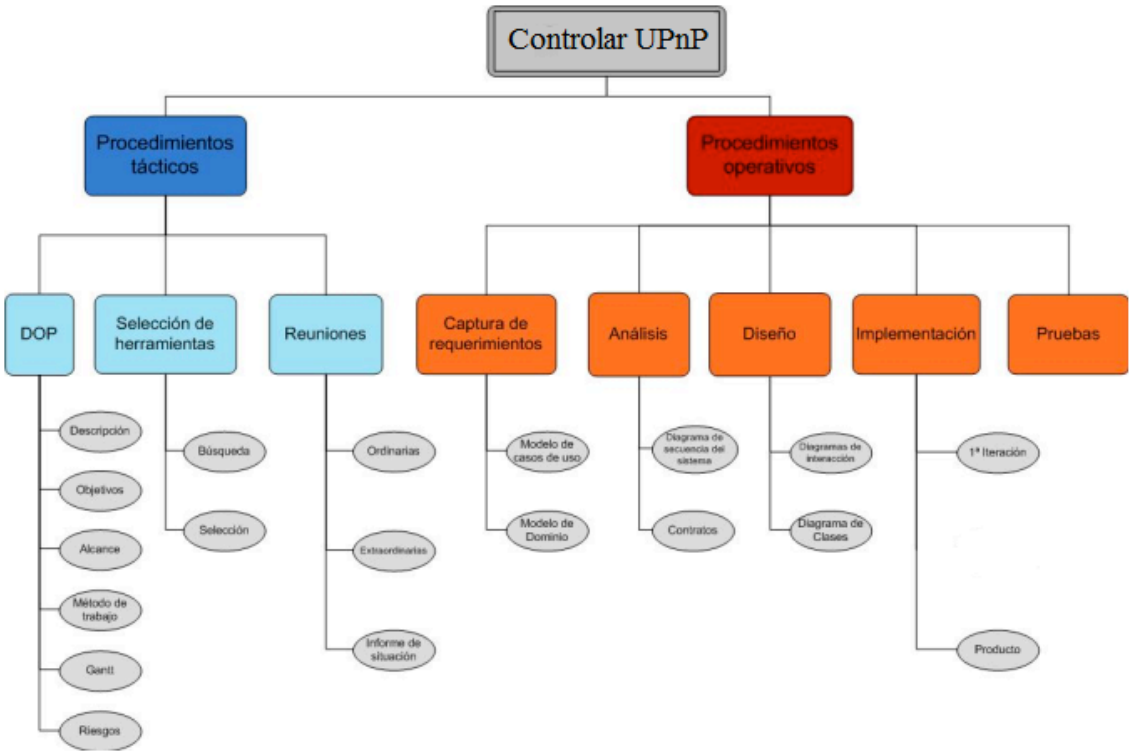


Ilustración 1 Diagrama E.D.T.

1.6 Alcance

1.6.1 Recursos

Recursos Humanos:

- El proyecto se enmarca dentro de la Ingeniería Informática (15 créditos), debería ser realizado en 450 horas.

Recursos Materiales:

- Los recursos materiales son los medios que se han utilizado para trabajar en el proyecto, es decir, diverso *software*, *hardware*, medios digitales y medios tradicionales. Ej. Servidor UCH, Eclipse,...

- Software de gestión de proyectos, como es el "GanntProyect"[1], y un editor de texto *"Open Source"* como es Open Office[2].

- El *hardware* son ordenadores de mesa y portátiles.

[1] http://www.microsoft.com/project/en-us/project-management.aspx
[2] http://www.ganttproject.biz/

- En cuanto a los medios digitales, estos son los *pendrives*, discos duros, extraíbles, CD-ROM y etc.

Recursos Formativos y Orientativos:

- Como estamos en periodo de formación, no nos podemos olvidar que se dispone de varios recursos de formación, como son los libros de la biblioteca, manuales de Internet apuntes de clase y etc. En cuanto a los recursos orientativos tenemos al profesor Julio Abascal y a Borja Gamecho, que nos pueden indicar si vamos por buen camino en el devenir del proyecto, sacarnos de las posibles dudas que podamos tener, etc.

1.6.2 Lista de entregas

El proyecto está planificado en torno a las siguientes entregas.

Fecha	Entrega	Documento
30/11/2010		DOP
31/12/2010		Captura de Requerimientos y Análisis
15/01/2011		Diseño
01/04/2011		Implementación y pruebas
15/06/2011		Presentación Pública y memoria

Como se puede apreciar, al final se ha alargado la entrega "Presentación pública y memoria" y una pequeña parte de la entrega "Implementación y pruebas".

1.7 Planificación temporal. Diagrama de Gantt

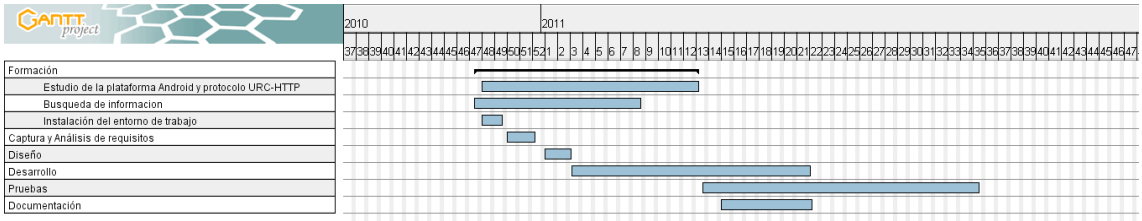


Ilustración 2 Diagrama de Gantt (esperado)

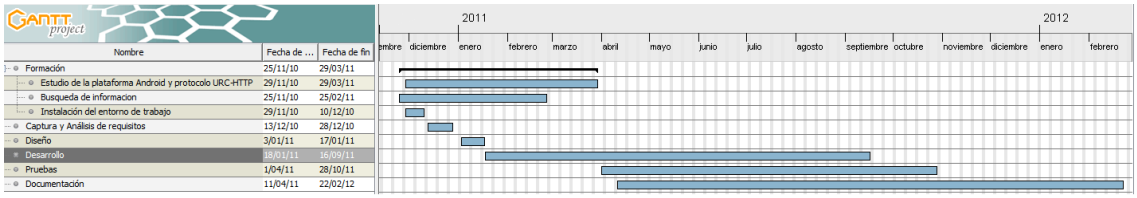


Ilustración 3Diagrama de Gantt (real)

Aproximadamente han sido un total de 550-600 horas, 100-150 más de las esperadas.

Capítulo 2 VIABILIDAD Y ALTERNATIVAS

Hasta que no se terminó el estudio de la plataforma Android no se sabía si el proyecto iba a ser viable o no. En ese punto nos dimos cuenta de que si era factible, se podía usar el UCH directamente con una aplicación nativa para Android

También debo mencionar que ya existía por parte de los creadores del estándar URC[3] una aplicación para iPhone, pero esta aplicación no es nativa para el sistema operativo iOS, ya que utiliza únicamente un navegador para comunicarse con UCH y éste con los dispositivos. Otra diferencia respecto a la aplicación desarrollada para iOS, es que no generan una interfaz genérica para todos los dispositivos, solo se puede mostrar interfaces de usuario para los dispositivos conocidos previamente.

Por lo tanto, ya que había una versión para iPhone, se decidió implementarlo para Android, pero de forma nativa, no desde un navegador y generando interfaces dinámicamente para cualquier dispositivo desconocido.

Como primer objetivo del proyecto nos planteamos obtener cabeceras HTTP desde un entorno con UCH, y como segundo objetivo tratar cabeceras HTTP desde una plataforma móvil. Pero para enlazar el primer objetivo con el segundo, se debe de haber realizado un estudio del protocolo URC-HTTP.

Tras ello se pasa al estudio de la plataforma Android para posteriormente crear la aplicación nativa que captura la información proporcionada por UCH.

Por tanto, para el desarrollo del proyecto se debe:

- Estudiar el funcionamiento de UCH y URC.
- Estudiar la plataforma Android.
- Crear una aplicación nativa para Android que implemente un URC capaz de controlar “dispositivos objetivos” comunicándose mediante el protocolo URC-HTTP al UCH y desde este con el mundo.

Otra alternativa viable y con futuro, dado que es soportada por distintos sistemas operativos, es la implementación del URC sobre HTML5, ya que existen herramientas que a partir de HTML5 encapsulan y crean la aplicación para el sistema operativo deseado, tal como por ejemplo PhoneGap oTitanium.

En cuanto a los objetivos secundarios, para el descubrimiento de UCH´s se puede lanzar un *script* para obtener las distintas redes (IP´s) detectadas por nuestro dispositivo (URC) y probar si tiene lanzado UCH. No necesitamos conocer más, con la IP nos basta para acceder y manejar los distintos dispositivos detectados por ese UCH.

Y como último objetivo secundario está la implantación de seguridad en UCH, que se puede conseguir mediante HTTPS configurando previamente Apache[4].

---

[3] http://myurc.org/

[4] Activar SSL en Tomcat (2010)
http://www.locualo.net/programacion/activar-ssl-tomcat-certificado-digital-prueba/00000081.aspx

Además, se puede mejorar con la implementación de un control de usuarios que consulte el fichero UIList y este a su vez consulte al *UCH Control Layer*, y decidir si está registrado o no el usuario.

Capítulo 3 CONTEXTO DE USO DE LA APLICACIÓN

El proyecto está enmarcado dentro del campo de la Inteligencia Ambiental o AmI. La Inteligencia ambiental se conoce por la integración de diferentes dispositivos u ordenadores en el entorno de una persona de manera transparente, permitiéndole interactuar con ellos de forma natural, respondiendo a nuestras acciones dependiendo de las necesidades.

Estas respuestas del sistema parten de mediciones tomadas de sensores, como por ejemplo las condiciones de luz, temperatura, localización, movimiento, señales vitales del cuerpo humano como la presión sanguínea, la frecuencia cardiaca, etc.

Se pretende crear interfaces sencillas e intuitivas para que no requieran un aprendizaje y permitan controlar un dispositivo. El sistema debe de ser embebido, sensible al contexto, y debe adaptarse o anticiparse a las necesidades del usuario.

Con todo esto se consigue un sistema que se adapte a las personas para mejorar la calidad de vida, más aún para personas ancianas o con algún tipo de discapacidad.

Se deben cumplir tres características:

- Interoperabilidad: Describe la capacidad que tiene el sistema de cooperar e intercambiar información entre los distintos elementos.
- Heterogeneidad: Este término describe la capacidad de ejecutar el mismo software en diferentes dispositivos.
- Dinámico: Es la capacidad de la red de adaptarse a los cambios. Por ejemplo cuando se conecta un nuevo dispositivo.

Nos referimos a Computación Ubicua o también conocida por *pervasive computing* cuando el sistema es capaz de acceder y tratar una gran cantidad de datos del entorno físico, humano y social, de forma independiente a la localización de los usuarios. Este sistema está formado por una red de ordenadores conectados entre ellos, embebidos o empotrados. Un término relacionado con la Computación Ubicua es el acceso ubicuo, que puede describirse como permitir el acceso al sistema ubicuo en cualquier sitio y desde cualquier dispositivo.

La Inteligencia Ambiental viene soportada por uno o varios *middlewares.* Estos abstraen algunas funcionalidades del sistema (sobre todo las relacionadas con la capa de red), formando una capa intermedia entre el sistema operativo y las aplicaciones ubicuas de forma transparente independientemente del protocolo de red usado por los distintos ordenadores

Características de un *middleware*[5] para Inteligencia Ambiental:

- Tratamiento dinámico (descubrimiento/instalación/actualización/borrado) de los dispositivos y aplicaciones.
- Soporte para múltiples usuarios y concurrencia.

[5]PFC Gamecho,2009

- Capaz de transmitir gran cantidad de información en escaso tiempo.

En el proyecto se ha desarrollado una aplicación cliente Android para que distintos usuarios puedan controlar un sistema con Inteligencia Ambiental soportado por el estándar URC como *middleware,* ofreciendo acceso ubicuo.

Capítulo 4 ESTUDIO DE UCH Y DEL PROTOCOLO URC-HTTP

4.1 Estructura del entorno y funcionamiento

El objetivo del proyecto es crear una aplicación nativa para Android que sea capaz de controlar "Dispositivos objetivo" o *targets* compatibles con UCH en *smartphones.*

UCH (*Universal Control Hub*) permite controlar los distintos dispositivos conectados a la red. UCH implementa el estándar URC (*Universal Remote Control*), permitiendo controlar remotamente los distintos dispositivos detectados por el mismo usando diferentes protocolos. Un cliente URC por ejemplo sería el *smartphone* o un PC, que a través de un *middleware* (UCH) permite controlar los dispositivos de un entorno.

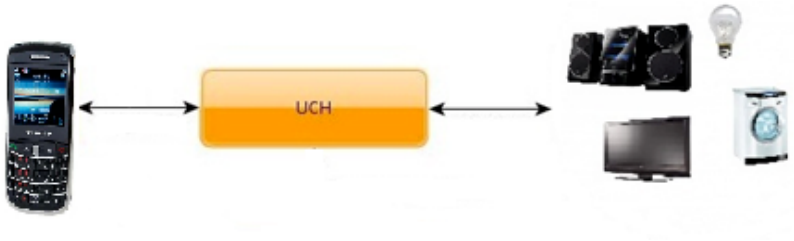


Ilustración 4 Entorno de la aplicación

4.1. 1 ¿Cómo está formado UCH?

UCH está formado por una serie de módulos, cada uno con una función específica, interactuando entre ellos, permitiendo la comunicación desde un *smartphone* con los distintos dispositivos a controlar. Estos módulos son:

UIList: Fichero que contiene las descripciones de los dispositivos (*targets*) descubiertos dinámicamente por UCH. Con tiempo real quiero decir que si un *target* se desconecta, UCH lo borrara de la lista en un periodo acotado de tiempo, al igual que si detecta uno nuevo lo añadirá. Se accede a este fichero mediante URL.

http://"IP"/UCH/GetCompatibleUIs Poniendo la IP donde está lanzado UCH.

UIPM: Es el módulo que permite enviar/recibir peticiones del usuario (terminal) a los dispositivos controlados. Utiliza protocolo URC-HTTP para comunicarse con el terminal.

UCH (*Socket Layer*): Es el núcleo de UCH. Se encarga de crear y gestionar el fichero UIList y de lanzar los módulos TA, TDM Y UIPM en tiempo de ejecución, escuchando estos módulos para manejar los cambios en los dispositivos.

TDM: Este módulo es el encargado de detectar nuevos dispositivos, avisando a UCH (*Socket Layer*), que además de actualizar el UIList con el nuevo dispositivo detectado, lanza el módulo TA correspondiente al nuevo *target*.

TA: Es el módulo encargado de interactuar con el dispositivo asociado.

En el estándar URC, aparte de los módulos explicados, se usan una serie de documentos que especifican el estándar URC, y estos son:

URC *Framework*: Especificación del marco de trabajo del URC.

User Interface Socket Description (UISD): Representa la funcionalidad y estado de un dispositivo o *target*. Cada *target* tendrá al menos un fichero .uis. Se usa para construir la interfaz gráfica ya que permite conocer las distintas acciones sobre un dispositivo, una descripción de cada acción y el estado de esta.

Resource Description (RSHEET): Se usa para crear una interfaz de usuario a partir de una plantilla asociada a cada *target*. Son ficheros .rsheet. No se ha usado en el proyecto, ya que se han generado las interfaces a partir de los ficheros .uis.

Target Description (TD): Contiene la descripción del *target* permitiendo controlarlo en una sesión. Aparte de la descripción, aparece información como la localización del .uis y .rsheet del *target* que le corresponde. Tampoco se ha usado en el proyecto ya que se conoce la localización del .uis gracias al UIList.

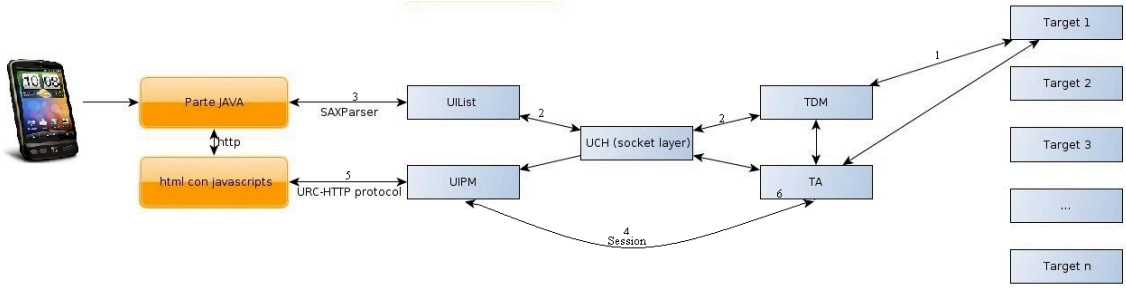


Ilustración 5 Módulos UCH

Conocidos los distintos módulos, voy a describir cómo interactúan estos elementos de la ilustración:

Para poder manejar los dispositivos desde el *smartphone*, UCH debe estar lanzado en un servidor que permita el uso de "*servlets*[6]". Para el proyecto se ha usado Tomcat. Una vez lanzado UCH (tiene los módulos cargados):

El módulo TDM comienza el descubrimiento de los dispositivos de la red. Este módulo está continuamente escuchando la red, detectando altas y bajas de dispositivos detectados dinámicamente.

Cuando el TDM detecta un nuevo dispositivo, analiza el fichero .td y comunica al UIList los *sockets* que le corresponden al nuevo dispositivo (puede haber más de uno por dispositivo). Cuando detecta que un dispositivo se ha desconectado, lo borra del UIList.

Cuando el usuario (URC) desea controlar un dispositivo, se realiza una consulta al UIList (mediante URL) para conocer el *socket* que le corresponde al dispositivo.

Se lanzan los módulos UIMP y TA, y se crea un objeto *Session* que permite la comunicación del usuario con el dispositivo a través de una sesión enlazando los módulos UIMP y TA. El objeto *Session* contiene información del usuario, el *target* y el *socket* usado por este. Por lo tanto podremos tener varias sesiones por distintos usuarios, *targets* o *sockets*

[6] Objetos capaces de ofrecer información dinámicamente, alojados en un servidor web.

Cuando el usuario realiza una petición, por ejemplo encender una bombilla, se realiza sobre el módulo UIMP que a través del objeto *Session* se comunica con TA y este con el dispositivo. Esta comunicación se realiza usando URC-HTTP. Como el estado de la bombilla ha cambiado se actualiza el estado de esta en el módulo TA y en todas las sesiones relacionadas con el *target*.

4.1.2 URC-HTTP

El URC-HTTP se utiliza para comunicar el terminal (URC) con UCH sobre el protocolo HTTP, por lo cual, el URC debe de usar un navegador para poder utilizar el protocolo.

Como en el proyecto se ha intentado realizar con código nativo de Android, en lugar de optar por cargar en un navegador (*WebKit*) una página HTML que contenga: la interfaz de los dispositivos a controlar, la lógica de negocio y cargue los *scripts* que forman el protocolo URC-HTTP para comunicarse con UIMP, se optó por separar la interfaz gráfica y la lógica de negocio para implementarlo como aplicación nativa de Android, con el uso de Java para la lógica de negocio y los XML para las interfaces gráficas.

Como el protocolo URC-HTTP funciona sobre un navegador, se ha creado un fichero HTML que carga los *scripts* que forman el protocolo URC-HTTP que permite la comunicación con el módulo UIMP de UCH. Este HTML se carga en un *WebView* sin mostrarlo en pantalla.

Teniendo la estructura, por ejemplo, para el ejemplo de una luz que se enciende y se apaga, basta con llamar desde java al *script* "encender" contenido en el fichero HTML y este se encarga de comunicarse con UCH mediante URC-HTTP, protocolo generado mediante unos *scripts*. Estos *scripts* o librerías son:

lib.js: Se utiliza para trabajar con ficheros XML, muy usados en los módulos de UCH. Realiza funciones como pasar ficheros XML a *Strings* y viceversa, manipulación de los ficheros XML, etc.

schema.js: Usa los ficheros lib.js y urc-http.js. Es una biblioteca JavaScript para clientes web genéricos. Define la función constructora y los métodos para la clase de esquema (para estructurar la información). Se utiliza en la librería socket.js.

socket.js: Utiliza las librerías lib.js y URC-HTTP. Es una biblioteca JavaScript que permite la construcción del objeto *socket* y métodos para su gestión.

webclient.js: Utiliza las librerías socket.js, urchttp.js y lib.js. Se encarga de inicializar todo lo necesario para la comunicación del terminal con UCH, de gestionar las sesiones, etc. La llamada que he comentado sobre la bombilla en el *script* "encender" se realiza así:

```
webclient.setValue("http://pfc.cybergar.upnp/light/controlSocket1#/1")
```

Este método *setValue()* contiene una llamada al *script* urchttp:

```
urchttp.setValue(session.protocolUrl, session.sessionId, path, value);
```

urchttp.js: Librería JavaScript usada para comunicarse con el cliente sobre el protocolo

URC-HTTP. Permite abrir/cerrar sesiones con UCH, obtener o establecer valores en UCH, enviar comandos, recibir actualizaciones de UCH, etc. Utiliza la librería lib.js

4.2 Cómo añadir un nuevo *target* en UCH

Como se ha mencionado en el apartado 4.1, si queremos disponer de una interfaz específica para un dispositivo concreto, UCH deberá tener información previa sobre el dispositivo concreto a controlar. Por ejemplo, para una televisión, no podremos generar dinámicamente una interfaz que represente un mando a distancia real, a no ser que se conozca previamente qué tipos de funciones controlaremos.

En el proyecto se han añadido 4 *targets* específicos: bombilla, satelliteBox, lavadora y MP3. Por lo tanto, para cada uno de ellos hemos tenido que cumplir los siguientes pasos:

El fichero /UCH/WEB-INF/uch.config contiene tdmList, taList y uipmList, las propiedades de los distintos *target*s y otros datos de los distintos dispositivos que vamos a controlar mediante UCH.

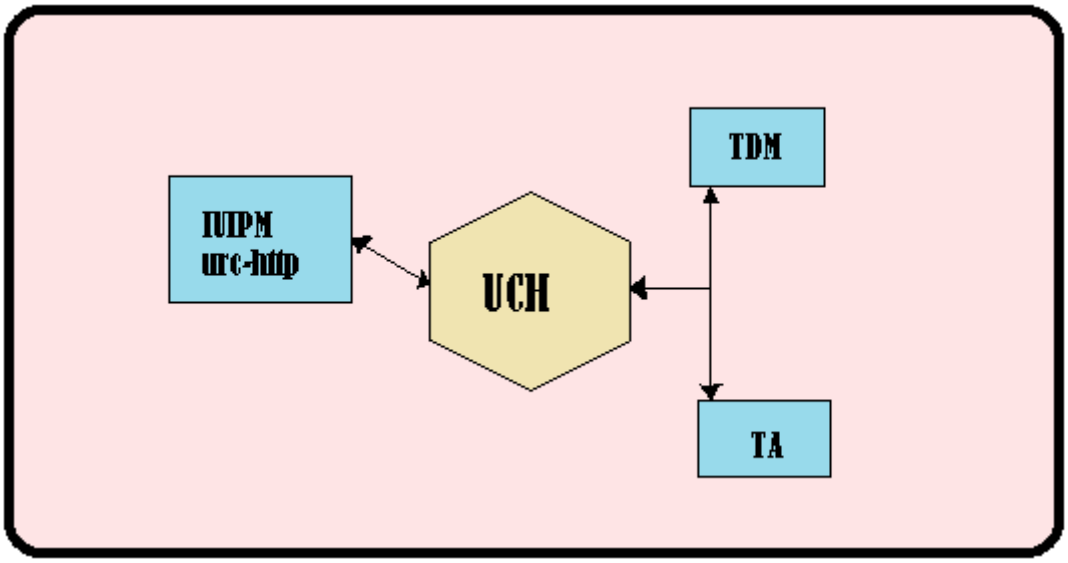


Ilustración 6 Iteración con UCH

En concreto, taList contiene información sobre los dispositivos, tal cómo dónde está alojada la clase TaFacade (clase que implementa la interfaz del TA y gestiona las sesiones) o dónde se encuentra el fichero .td (que contiene la descripción del *target* y permite controlarlo) de cada dispositivo. La estructura del fichero uch.config es la siguiente:

```
<uchConfig>
<uch>…
</uch>
<tdmList>…
</tdmList>
<taList>
<ta>#descripción
bombilla…</ta>
<ta>#descripción
SatelliteBox…</ta>
<ta>#descripción
MP3…</ta>
…
</taList>
<uipmList>…
</uipmList>
</uchConfig>
```

Por lo tanto, si queremos añadir un nuevo *target* deberemos incluirlo en el taList. Para el ejemplo de la bombilla se ha añadido un <ta>…</ta> siguiendo la estructura del fichero uch.config:

```
<ta>
  <prop>
     <name>http://myurc.org/ns/res#dynamicLibClass</name>
     <value>pfc.upnpCybergarageLight.UpnpLightTAFacade</value>
  </prop>
  <prop>
     <name>http://myurc.org/ns/res#type</name>
     <value>http://myurc.org/restypes#ta</value>
  </prop>
  <prop>
     <name>http://purl.org/dc/terms/conformsTo</name>
     <value>http://myurc.org/TR/uch-20080811/</value>
  </prop>
  <prop>
     <name>http://myurc.org/ns/res#deviceType</name>
     <value>urn:schemas-upnp-org:device:light:1</value>
  </prop>
  <prop>
     <name>http://myurc.org/ns/res#devicePlatform</name>
     <value>UPnP</value>
  </prop>
  <prop>
     <name>http://purl.org/dc/terms/modified</name>
```

```
      <value>2008-11-15T00:00:00Z</value>
    </prop>
    <prop>
      <name>http://myurc.org/ns/res#tdUri</name>
      <value>pfc/ta/upnp/light/upnpLight.td</value>
    </prop>
  </ta>
```

Para cada *target* a introducir debemos de alojar en UCH/web-inf/lib un fichero .jar, que contiene los ficheros TaDetails, TA, TAFacade, entre otros. Este fichero .jar se ha exportado desde Eclipse.

Lo que se exporta es un módulo creado en Eclipse que implementa a org.myurc.uch.TA, por lo tanto debemos rellenar la estructura con los datos de nuestro *target*. Tras realizarlo ya podremos exportar.

En UCH/pfc/ta/upnp debemos de añadir una carpeta que contenga los ficheros .td, .rsheet y .uis correspondientes al *target* que deseamos incluir. Para el caso de la bombilla, se ha añadido la carpeta "light" al directorio UPnP con los 3 ficheros.

4.2.1 Fichero upnpLight.uis

Aparecen las diferentes acciones disponibles (variables) para el *target*. En el caso de la bombilla solo hay una, que permite conocer el estado de la bombilla. También aparece el identificador del *socket* sobre el que se realizará la conexión de la sesión para este *target*.

```
<uiSocket about="http://pfc...upnp/light/controlSocket1"id="controlSocket1" …
<variable id="estado" type="xsd:int" includeRes="true">
<dc:description xml:lang="en">
  Turns on,off the ligth.
</dc:description>
</variable>
</uiSocket>
```

4.2.2 Fichero upnpLight.td

Es el fichero que contiene la información que permite controlar el *target* y su localización se descubre gracias al UIList. Contiene información de la localización de los ficheros upnpLight.rsheet y upnpLight.uis, el *socket* a usar, etc.

```
<target about="http://pfc.cybergarage.upnp/light" id="target" … >   …
<socket id="controlSocket1" name="http://pfc.cy....upnp/light/controlSocket1">
<socketDescriptionLocalAt>upnpLight.uis</socketDescriptionLocalAt>
</socket>
<rdf:RDF>
<res:ResDir>
<res:ResSheet rdf:about="http://pfc.cybergarage.upnp/light/rsheet.rdf">
```

```
<res:forLang>en</res:forLang>
<dcterms:conformsTo>…</dcterms:conformsTo>
<res:localAt>upnpLight.rsheet</res:localAt>
<dc:type>Text</dc:type>
<res:aResDescForDomain
rdf:resource="http://pfc.cybergarage.upnp/light/controlSocket1"/>
</res:ResSheet>
</res:ResDir>
</rdf:RDF>
</target>
```

4.2.3 Fichero upnpLight.rsheet

Añaden información sobre los elementos del fichero .uis y sirven de ayuda para generar la interfaz del usuario:

```
<ResSheet rdf:about="http://pfc.cybergarage.upnp/light/rsheet.rdf">
<dcterms:conformsTo>…</dcterms:conformsTo>
<dc:type>Text</dc:type>
<dc:format>text/xml</dc:format>
<dc:publisher>Urc Consortium</dc:publisher>
<dc:title xml:lang="en">…</dc:title>
<dcterms:modified>2009-07-12</dcterms:modified>
<aResDes…  rdf:resource="http://pfc.cybergarage.upnp/light/controlSocket1"/>
<forLang>en</forLang>
<resItems rdf:parseType="Collection">
<AResDesc rdf:about="http://pfc.cybe....upnp/light/rsheet.rdf#socket_label">
<content rdf:parseType="Lit" xml:lang="en">Socket_pfc_upnpLight</content>
<useFor rdf:parseType="Collection">
<Context>
<eltRef rdf:resource="http://pfc....upnp/light/controlSocket1#controlSocket1"/>
<role rdf:resource="http://myurc.org/ns/res#label"/>
</Context>
</useFor>
</AResDesc>
…
</resItems>
</ResSheet>
```

Una vez realizados estos pasos, necesitamos una interfaz de usuario para poder controlar el *target*. En el apartado siguiente (4.3), se indican los pasos a seguir para la construcción de la interfaz y en el capítulo 5.11 el ejemplo concreto para la bombilla.

4.3 Cómo crear una interfaz específica para un dispositivo concreto

Introducción

La aplicación ControlUCH tiene implementadas una serie de interfaces para unos dispositivos concretos. Estos son: Una bombilla, un reproductor MP3, el SatelliteBox y una lavadora.

Siempre no conoceremos las características de cada dispositivo a controlar, por eso, aparte de las interfaces de usuario conocidas se ha implementado el descubrimiento y generación de interfaces para cualquier dispositivo desconocido, pero estas interfaces desconocidas o genéricas, a pesar de ser bastante intuitivas no son muy practicas, ya que por cada acción del dispositivo que queramos realizar, va a generar una *Activity* (pantalla) diferente. El problema de generar muchas actividades radica en que se complica mucho la manejabilidad de los dispositivos, muchas pantallas o actividades para un único dispositivo.

Por ejemplo para el SatelliteBox, al seleccionar el dispositivo, se lanza la *Activity* en la que aparece una lista con todas las acciones posibles para el SatelliteBox. Por cada acción que se selecciona, se lanza otra actividad, que contiene una descripción de la acción seleccionada, un campo editable con el valor actual de dicha acción y dos botones. Uno para enviar el nuevo valor que le indiquemos a la acción en el campo editable y otro para volver a la lista de acciones del dispositivo elegido, en este ejemplo a la lista de acciones del SatelliteBox.

En muchos casos se crean abundantes actividades innecesarias e ineficaces. Por ejemplo, los canales de televisión no siempre tienen el mismo volumen. Si tenemos la televisión a un volumen bajo para no molestar, y cambiamos a un canal que tenga el volumen más alto y deseemos bajar el volumen rápidamente, será complicado realizarlo en un tiempo corto porque hay que pasar por 4 actividades diferentes para conseguir todo el proceso. La primera actividad es la que selecciona la acción cambiar canal, se crea la segunda actividad con la información de la acción “cambio canal”, la tercera actividad se crea al cargar de nuevo la lista de acciones para poder seleccionar la acción volumen, y la cuarta actividad que muestra la información de la acción volumen.

Esto no es muy práctico si se va a controlar regularmente el dispositivo. Por ello, debemos crear una interfaz concreta para un dispositivo concreto para conseguir una funcionalidad más concisa y eficaz. En este apartado se muestra cómo podemos diseñar nuestra propia interfaz y con su correspondiente funcionalidad para un dispositivo concreto.

Estructura de una interfaz concreta

Antes de realizar los pasos descritos a continuación, deberemos tener cargado el dispositivo o *target* en el servidor UCH.

Por cada interfaz que deseemos crear para un dispositivo específico, tenemos que seguir el siguiente procedimiento:

En la carpeta Res dentro del proyecto Android, hay una carpeta llamada *layout*. Dentro de esta crearemos un fichero XML que contendrá el diseño de nuestra interfaz. Podemos especificar el diseño mediante código o utilizando en entorno gráfico que nos proporciona el *plugin* para Eclipse (com.android.ide.eclipse). Aquí hay que tener en cuenta para qué versión de Android se va a desarrollar, porque dependiendo de la versión se usan unos u otros controles y la apariencia en pantalla no será la misma.

Clase Java que extiende de *Activity*. A cada clase Java le corresponde una actividad. En esta clase se implementa la lógica del dispositivo que queremos controlar, relacionándola con los componentes de la interfaz gráfica (fichero XML del punto 1).

Fichero HTML localizado en la carpeta *assets* de nuestro proyecto Android, que contendrá los métodos llamados desde Java mediante *scripting*. Estos métodos realizan llamadas al servidor UCH mediante el protocolo URC-HTTP.

Modificación del fichero AndroidManifest.xml. Toda actividad se declara en este fichero. Se le añade una línea para que reconozca la nueva actividad. En este fichero también se le indican los permisos necesarios si no estuvieran incluidos.

```
<activity android:name="NombreActividad"android:label="TextoConTituloActividad"></activity>
```

Editar la actividad menuUisDetectadas, añadiendo una línea que indique la localización del fichero .uis que le corresponde al dispositivo para el que deseemos crear la interfaz. Esta línea se añadirá en el método cargarDispositivos (localizado al final del fichero)

```
Ej.dispositivos.add("+IP+/UCH/SatelliteBoxNQT/satellitebox.uis");
```

En el apartado 5.11 se puede ver el ejemplo de la bombilla usando la interfaz concreta.

Capítulo 5 DESARROLLO DE LA APLICACIÓN

5.1 Introducción

El proyecto realizado permite controlar desde un terminal Android, los distintos dispositivos de un entorno ubicuo, de la manera más transparente posible para el usuario. Está enmarcado dentro del campo de los servicios para la inteligencia ambiental (AmI).

La aplicación permite controlar los distintos dispositivos de un entorno gracias a UCH. El terminal móvil realiza peticiones HTTP usando un protocolo llamado URC-HTTP a UCH, y este se encarga de interactuar con los distintos dispositivos usando una serie de módulos que se encargan de descubrirlos y mantenerlos conectados.

La aplicación se puede dividir en dos partes. Una contiene la parte puramente nativa (Java) de Android y la otra, las comunicaciones con UCH y de éste con el resto. Para la unión de estas dos partes, se usa un fichero HTML que carga unos ficheros JavaScript, encargándose de la comunicación con UCH.

UCH permite descubrir sistemas UPnP, *Jini, Web Services* o cualquier *middleware* o protocolo similar (Bluetooth, TCP/IP, SOAP, etc.)

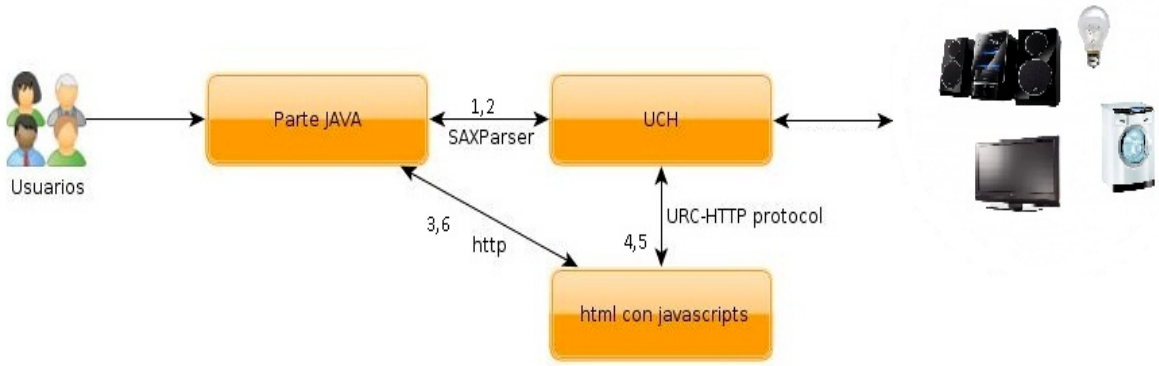


Ilustración 7 Iteración de partes de aplicación

1. La aplicación accede a `http://"+IP+"/UCH/GetCompatibleUIs` que mediante un *SAXParser*, usado como lector de documentos XML, realiza petición para obtener la información del fichero GetCompatibleUIs que contiene la descripción de los dispositivos detectados por UCH. Parte del fichero UIList:

```
<uilist xmlns="urn:schemas-upnp-org:remoteui:uilist-1-0">
<ui>
<uiID>UPnP-UES</uiID>
<name>UES</name>
<protocol shortName="URC-HTTP">
<uri>http://192.168.0.13:8080/UCH/urchttp/UPnP-UES/level1socket</uri>
<protocolInfo>
<socketName>http://res.myurc.org/upnp/av/play-1.uis</socketName>
<targetDescriptionAt>
http://192.168.0.13:8080/UCH/upnpav/UpnpEntertainmentSystem.td
</targetDescriptionAt>
<conformsTo>http://myurc.org/TR/urc-http-protocol-20080627</conformsTo>
<targetId>UPnP-UES</targetId>
<socketDescriptionAt>http://192.168.0.13:8080/UCH/upnpav/play-
1.uis</socketDescriptionAt>
<targetName>http://res.myurc.org/upnp/av</targetName>
</protocolInfo>
</protocol>
<protocol shortName="URC-HTTP">...</protocol>
</ui>
<ui>...</ui>
</uilist>
```

2. Gracias a *XMLHandler*, recibe los datos de la petición sobre el fichero GetCompatibleUIs, formando una lista de dispositivos detectados, *List<myParsedXMLDataSet>* donde cada *myParsedXMLDataSet* representa a un dispositivo (*target*) y contendrá:

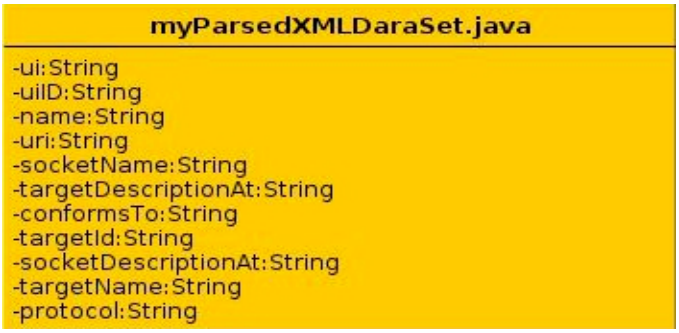


Ilustración 8 myParsedXMLDataSet.java

3. Una vez conocidos los distintos *targets* detectados, se puede realizar peticiones sobre estos. Como directamente desde Java no se puede comunicar con UCH, se carga en un *WebView* el HTML que se encarga de las comunicaciones con UCH.

```
mWebView.loadUrl("file:///android_asset/indexGenerico.html");
```

Este fichero HTML carga una serie de *scripts* que juntos formar el protocolo URC-HTTP, permitiendo llamarlos desde el código Java. Ahora ya se puede realizar peticiones a dispositivos ya que conocemos sus datos (*ui, name, uri, socket*,…):

```
// Iniciar el socket del target conocido
mWebView.loadUrl("javascript:init('"+socketIniciar+"')");

mWebView.loadUrl("javascript:enviarVal('"+socketCon+"','"+valEnvi+"')"
);
```

4. Comunicación mediante JavaScripts:

Envío:

webclient.setValue('http://res…org/upnp/demo/satelliteBox/socket#/volume', curVol)

5. Comunicación mediante JavaScripts:

Respuesta:

var curVol =org.myurc.webclient.getValue("http://pfc..p/light/controlSocket1#/estado")

6. Para obtener la respuesta a las peticiones JavaScript a UCH y recibirlas en la actividad (Java) usamos:

```
WebView.addJavascriptInterface(new DemoJavaScriptInterface(), "demo");
```

Gracias a esto, podremos llamar desde JavaScript a métodos contenidos en *DemoJavaScriptInterface* y así obtener el resultado mediante un evento.

Entrando en más detalle con los módulos de UCH:

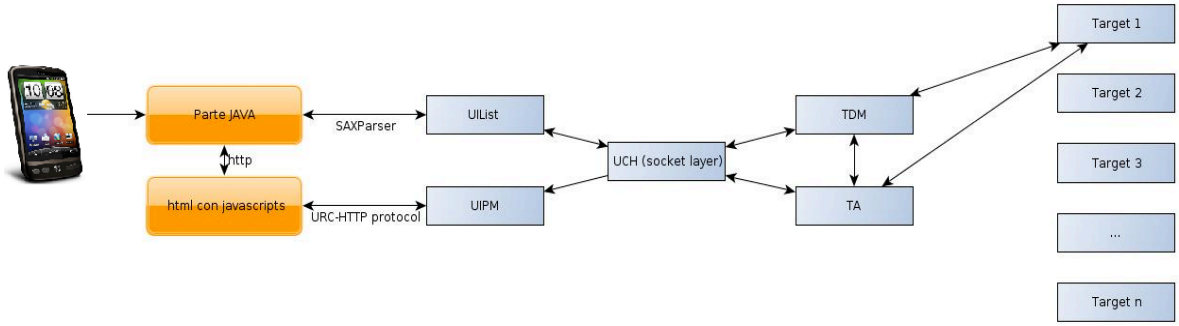


Ilustración 9 Relaciones entre la aplicación, UCH y los dispositivos (*targets*)

UIList: Fichero que contiene las descripciones de los dispositivos (*targets*) descubiertos en tiempo real por UCH. Con tiempo real quiero decir que si un *target* se desconecta, UCH lo borrara de la lista en un tiempo acotado de tiempo, al igual que si detecta uno nuevo lo añadirá. Se accede a este fichero mediante URL.

http://+IP+/UCH/GetCompatibleUIs poniendo la IP donde está lanzado UCH.

UIPM: Es el módulo que permite enviar/recibir peticiones del usuario (terminal) a los dispositivos controlados.

UCH (*Socket Layer*): Es el núcleo de UCH. Se encarga de crear y gestionar el fichero UIList y de lanzar los módulos TA, TDM Y UIPM en tiempo de ejecución, escuchando estos módulos para manejar los cambios en los dispositivos.

TDM: Este módulo es el encargado de detectar nuevos dispositivos, avisando a UCH (*Socket Layer*), que además de actualizar el UIList con el nuevo dispositivo detectado, lanza el módulo TA correspondiente al nuevo *target*.

TA: Es el modulo encargado de interactuar con el dispositivo asociado.

En el estándar URC, aparte de los módulos explicados, se usan una serie de documentos que especifican el estándar URC, y estos son:

URC *Framework*: Especificación del marco de trabajo del estándar URC.

***User Interface Socket Description* (UISD):** Representan la funcionalidad y estado de un dispositivo o *target*. Cada *target* tendrá al menos un fichero .uis. Se ha usado para construir la interfaz gráfica ya que permite conocer las distintas acciones sobre un dispositivo, una descripción de cada acción y el estado de esta.

***Resource Description* (RSHEET):** Se usa para crear una interfaz de usuario a partir de una plantilla asociada a cada *target*. Son ficheros .rsheet. No usado en el proyecto, ya que he generado las interfaces a partir de los ficheros .uis.

***Target Description* (TD):** Contiene la descripción del *target* permitiendo controlarlo en una sesión. Aparte de la descripción, aparece información como la localización del .uis y .rsheet del *target* que le corresponde. Tampoco se he usado en el proyecto ya que se conoce la localización del .uis gracias al UIList.

Una vez conocida la estructura del proyecto, vemos el funcionamiento de la aplicación en más profundidad. La aplicación está formada por una serie de pantallas o también llamadas actividades. Cada una de estas actividades está relacionada con una de las interfaces gráficas de la aplicación y son ficheros Java que extienden la clase *Activity* de Android. Las actividades contenidas en la aplicación son: Principal.java, Manual.java, ConfigurarUCH.java, Idioms.java, menuUisDetectadas.java, ListaAcciones.java, UiSeleccionada.java y RealizaAccion.java.

Cada una de estas actividades contienen la funcionalidad que se le va a dar a los elementos gráficos que se ven en la pantalla del terminal móvil en cada pantalla, pero no está especificado la composición de la interfaz gráfica, es decir, en las actividades no se especifica que tipos de elementos aparecerán en la pantalla, ni donde estará localizado cada componente, entre otras cosas.

Existe independencia entre la capa de presentación (interfaz gráfica) y la lógica de negocio con las ventajas que ello conlleva. La especificación de la interfaz gráfica se construye mediante ficheros XML. Cada actividad lleva asociado al menos un fichero XML que contiene la interfaz gráfica correspondiente a la actividad. En la siguiente

tabla se puede observar que fichero de interfaz gráfica (XML) corresponde a cada actividad en este proyecto.

Nombre Actividad	Nombre Interfaz gráfica
Principal.java	layoutprincipal.xml
Manual.java	layoutmanual.xml
MenuUisDetectadas.java	menu.xml
ListaAcciones.java	lista.xml
UiSeleccionada.java	layoutuiseleccionada.xml y tvlayout.xml
RealizarAccion.java	accion.xml
ConfigurarUCH.java	layoutip.xml
Idioms.java	layoutidiomas.xml

El nombre de las actividades comenzará por mayúsculas y el de los ficheros que contengan los elementos gráficos comenzará por minúsculas.

Toda la aplicación se mostrará en el idioma por defecto del terminal, a no ser que se seleccione otro idioma en la Actividad Idioms.java. Los idiomas para los cuales está disponible la aplicación son: Castellano, Inglés, Francés, Alemán, Portugués, Italiano Chino, Japonés, Coreano, Árabe, Euskera, Gallego y Catalán.

La aplicación aparte de permitir controlar dispositivos o *target*s desconocidos, se han implementado interfaces para 3 dispositivos o *target* específicos, ya que son más vistosas y funcionales. Estos 3 dispositivos son una televisión (SatelliteBox), una bombilla, y un MP3.

Como se ha comentado, cuando el dispositivo es conocido es mejor implementar su interfaz. Veamos porque:

Para el ejemplo de la televisión, sin la interfaz específica, mostraría una lista larga de acciones (volumen, encendido, brillo, cambiar canal,….), seleccionaríamos una acción y pasaríamos a otra pantalla para introducir manualmente el nuevo valor. Esto es un poco tedioso. Por ello, con las interfaces específicas conseguimos una interfaz que simula a un mando real, con todos sus botones.

Para la bombilla con la interfaz específica, mostramos una bombilla, que al tocar la pantalla del terminal se enciende o se apaga cambiando la imagen (sin usar textos ni listas).

Para el MP3 se ha implementado una interfaz con los botones necesarios para manejarlo.

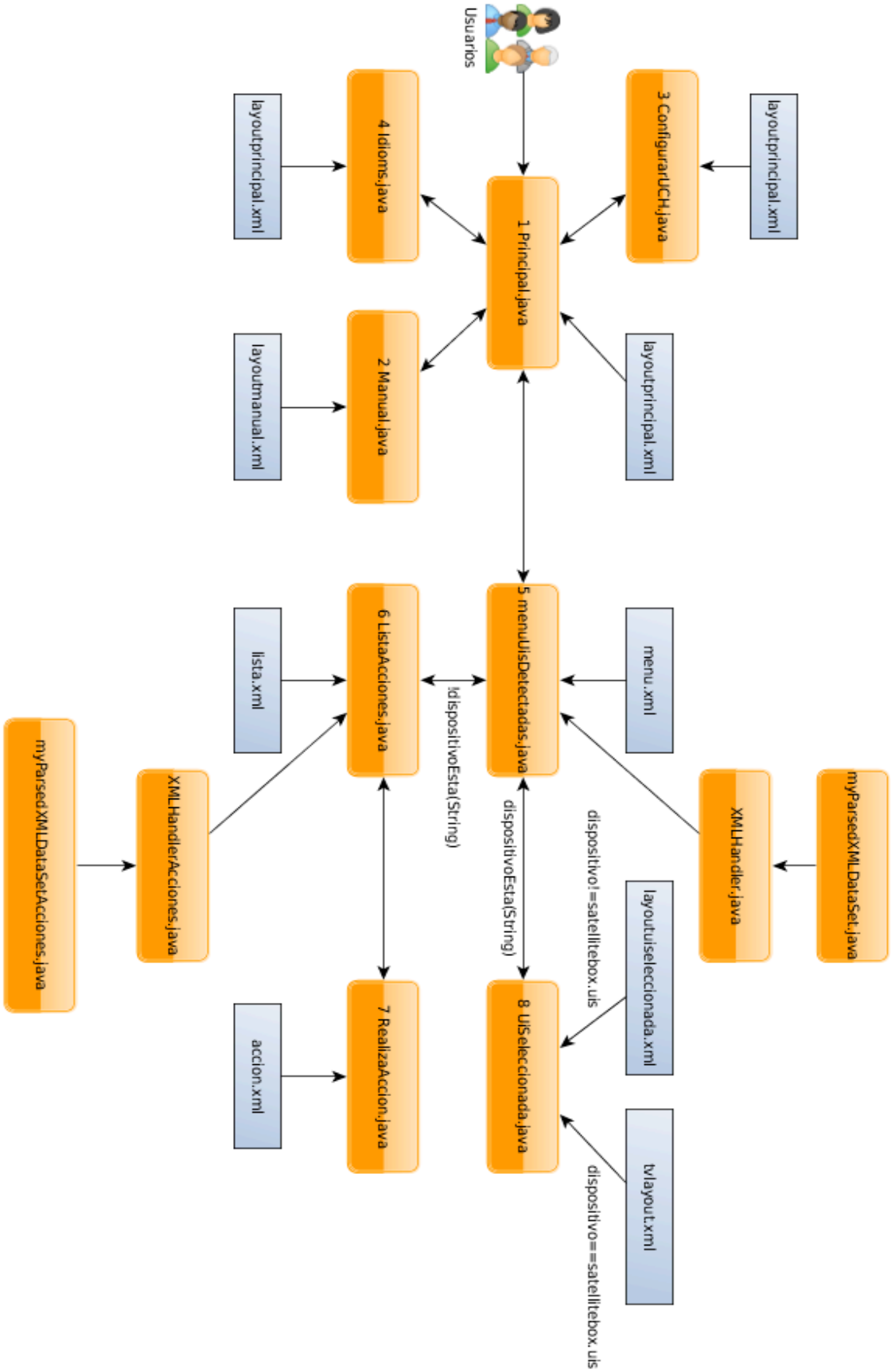


Ilustración 10 Diagrama de la aplicación

5.2Principal.java

La aplicación comienza con la Actividad Principal.java, como en Android no disponemos del método *main()*, deberemos indicar en el AndroidManifest.xml que actividad será la que arranque primero. Esto se indica así:

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

Esta actividad (Principal.java), gracias al fichero layouprincipal.xml se presenta el menú principal, que se compone de una serie de botones que enlazan con el resto de actividades de la aplicación. Estos botones son 5. 4 de ellos llevan a las actividades ConfigurarUCH.java, Manual.java, Idioms.java y menuUisDetectadas,java, mientras que el último botón se usará para salir de la aplicación.

Ilustración 11 Pantalla de Principal.java

Tras los botones aparece un *TextView* indicándonos que IP tenemos seleccionada, deberá ser la que esté lanzado el servidor UCH añadiéndole el puerto en el que está corriendo Tomcat. También aparece en la parte inferior la barra de publicidad de AdMob que se explica en el capítulo de insertar publicidad en la aplicación. Esta actividad tiene la apariencia de la ilustración 12.

MY_AD_UNIT_ID contiene un identificador único para cada aplicación Android, y es usado para poder incrustar publicidad de AdMob en la aplicación. Se carga en el *adView*.

PREF es usado para guardar preferencias de la aplicación y recordarlas la siguiente vez que ejecutemos.

En el método *onCreate()* se carga el fichero layoutprincipal.xml.

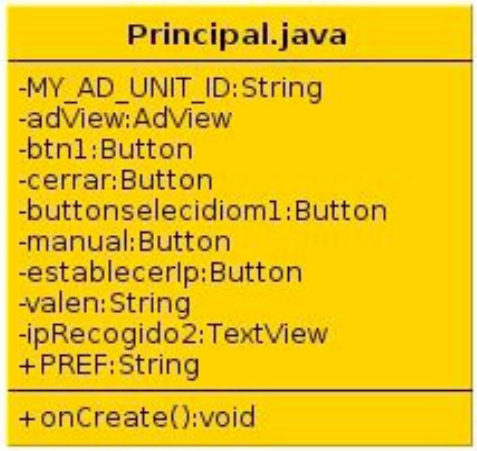


Ilustración 12 Clase Principal.java

5.3 Manual

El fichero Manual.java carga el fichero layoutmanual.xml en el método *onCreate* de la actividad. Este fichero XML contiene dos *TextViews*, uno con una pequeña descripción de la aplicación y otro, invitándonos a introducir un mail valido en el componente *EditText* situado en el mismo fichero layoutmanual.xml. También disponemos de dos botones, uno para enviar el correo y el otro, para volver a Principal.java. Al final se muestra la barra de publicidad.

Ilustración 13 Pantalla de Manual.java

La actividad Manual.java se utiliza para enviar por correo la información necesaria para que funcione la aplicación, ayudándose de la clase GMailSender.java. La actividad Manual.java tendrá la siguiente apariencia:

Al pulsar en enviar de la actividad Manual.java, se carga la cuenta Gmail, se construirá el mensaje y se enviará:

```
GMailSender s = new
GMailSender("nombre@gmail.com",
"contraseña");//cuenta Gmail
s.sendMail(titulo,cuerpo,origen,destinatari
o, buscando,msj1);
```

El correo que recibiremos (llega en el idioma por defecto del terminal) será algo así:

Step 1: Download and install Tomcat.
Step 2: Download the file from the link below and paste it into the Tomcat webapps folder.
Link: http://depositfiles.com/files/2ygjvc7vh (fichero UCH)

Note: By default the server is launched in 192.168.1.35:8080 (8080 port of Tomcat), as well as the application, which searches for devices in the direction 192.168.1.35:8080 (configurable in the application). In order to change the server IP just open with winrar or winzip the file downloaded, go to the folder WEB-INF and edit the web.xml file and replace the IP with the port desired.
Example: 192.168.1.35:8080
Restart Tomcat.

Remember that the IP that appears in the web.xml should be the same as appears by clicking on "Set IP of server UCH" main menu of the application.
Example: 192.168.1.35:8080

Ilustración 14 Clases que intervienen en el envío de correo

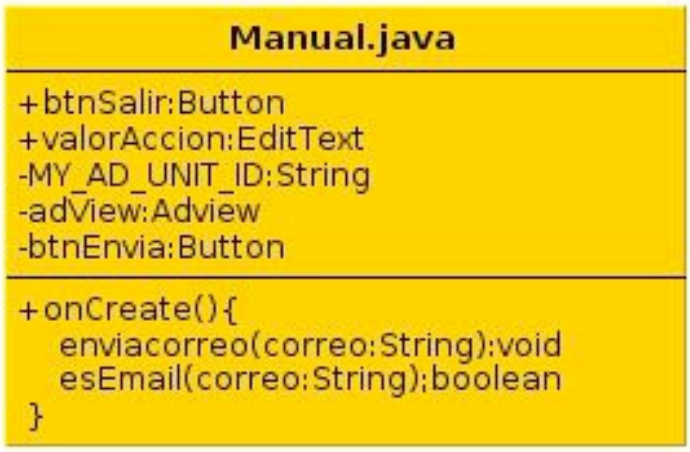


Ilustración 15 Clase manual.java

Ilustración 16 Clase GMailSender.java

La clase GMailSender.java se encarga de configurar una cuenta Gmail para poder enviar emails a los destinatarios necesarios ayudándose de la clase JSSEProvider.java.

Ilustración 17 Clase JSSEProvider.java

5.4 ConfigurarUCH

La actividad ConfigurarUCH.java tiene como función establecer la IP en la que está lanzado UCH en Tomcat. También permite establecer el puerto.

La interfaz se carga a partir del fichero layoutip.xml. Este fichero contiene 4 *TextViews*, uno indica el titulo, otro la palabra "Actual" en su respectivo idioma, otro con la IP actual mas el puerto y el cuarto contiene "Ej:192.168.1.35:8080" como modelo para completar el *EditText*.

También dispone de dos botones. Uno para establecer la IP con el texto introducido en el *EditText* y meterlo en una variable global para la aplicación (así se recuerda cada vez que se ejecuta la aplicación) y otro, para volver al menú principal Principal.java.

Ilustración 18 Pantalla de ConfigurarUCH.java

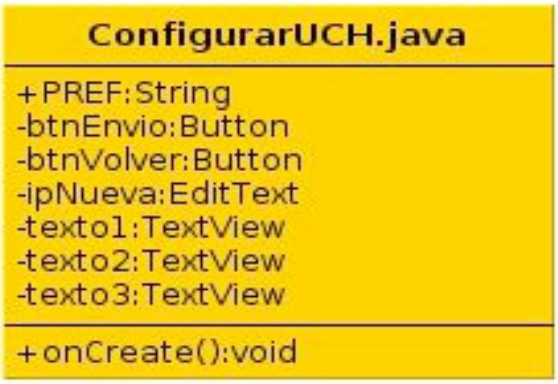


Ilustración 19 Clase ConfigurarUCH.java

5.5 Idioms

La actividad Idioms.java se utiliza para cambiar de idioma. Se carga el fichero layouidiomas.xml en el método *onCreate()*. Este fichero XML contiene un *ListView* (lista de *Views*), formado por 14 elementos, los 13 idiomas disponibles y el último elemento con el texto "Salir" que permite regresar a la pantalla principal, Principal.java.

Ilustración 20 Pantalla de Idioms.java

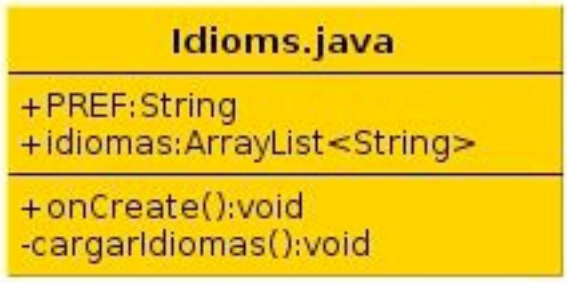


Ilustración 21 Clase Idioma.java

Pese a que a los usuarios se les muestra el texto del idioma por completo, se usan identificadores de idiomas para relacionarlo con su carpeta values correspondiente (explicado en Anexo 7.5, Localización).

Euskera → eu	Gallego → gl	Catalán → ca	Inglés → en
Japonés → ja	Francés → fr	Alemán → de	Chino → zh
Coreano → ko	Portugués → pt	Árabe → ar	Italiano → it

5.6 MenuUisDetectadas

La actividad menuUisDetectadas.java carga el fichero menú.xml en el método *onCreate* de la actividad. menú.xml contiene una lista de objetos. Cada objeto que aparezca en la lista, representa a un dispositivo detectado.

Ilustración 22 Clases utilizadas en MenuUisDetectadas.java

Los elementos de la lista de MenuUisDetectadas se obtienen realizando una consulta al fichero UIList del servidor UCH, y estructura los resultados gracias a dos ficheros. Un fichero se llama XMLHandler.java, éste se encarga de obtener la información de los distintos dispositivos que detecta el servidor UCH consultando el UIList del servidor, obtendrá una lista de objetos. Cada uno de estos objetos son instancias de la clase myParsedXMLDataSet (el segundo fichero), que contienen información variada del dispositivo detectado, desde el nombre del dispositivo hasta el *socket* entre otros muchos parámetros. La lista de myParsedXMLDataSet se obtiene en el método *onCreate()*.

Ilustración 24 Clase MenuUisDetectadas.java

Ilustración 23 Pantalla de menuUisDetectadas.java

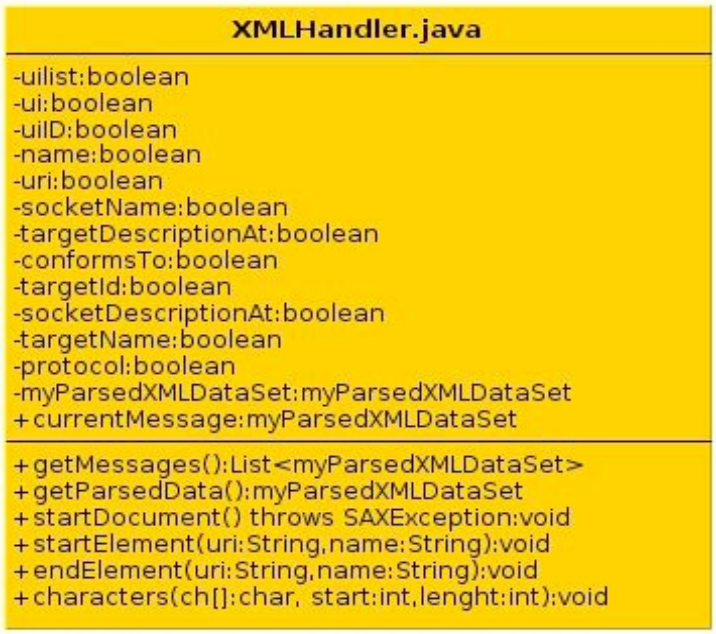


Ilustración 25 Clase XMLHandler.java

Ilustración 26 Clase myParsedXMLDataSet.java

5.7 ListaAcciones

La aplicación, tras mostrar los distintos dispositivos que ha reconocido, se podrá seleccionar uno. Si el dispositivo seleccionado posee su propia interfaz gráfica específica, la aplicación nos lleva hasta la actividad UiSeleccionada.java, si no está implementada previamente su interfaz, nos lleva a la actividad ListaAcciones.java.

La actividad ListaAcciones.java al igual que menuUisDetectadas.java, gracias a dos ficheros obtiene la información del servidor UCH, dentro del método *onCreate()*. El fichero XMLHandlerAcciones.java accede al fichero .uis del dispositivo seleccionado, introduciendo la información de cada acción en un *myParsedXMLDataSetAcciones* y así obtener una lista de acciones (lista de *myParsedXMLDataSetAcciones*) con toda la información correspondiente a cada acción. La forma de estructurar la información es similar a la que se realiza con el fichero de lista de dispositivos detectados por UCH, explicado en el punto 5.6.

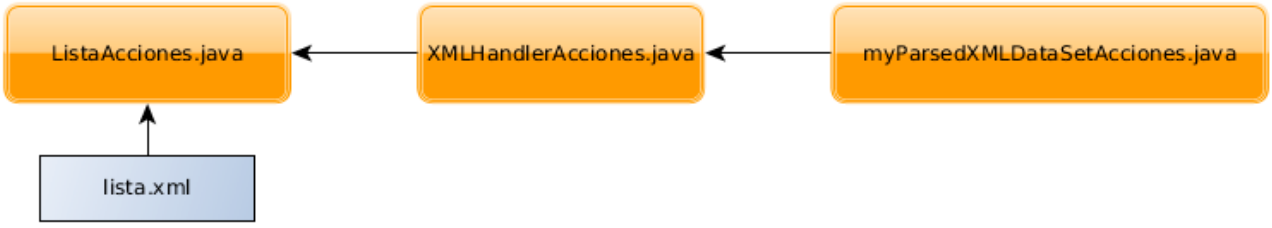


Ilustración 27 Clases utilizadas en ListaAcciones.java

Ilustración 28 Clase XMLHandlerAcciones.java

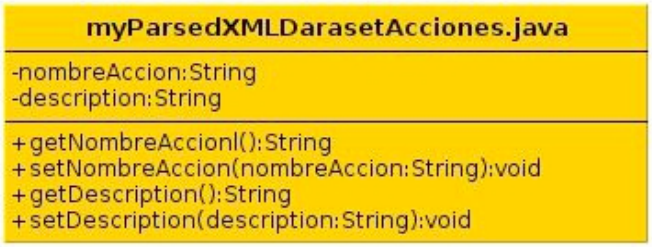


Ilustración 29 Clase myParsedXMLDataSet.java

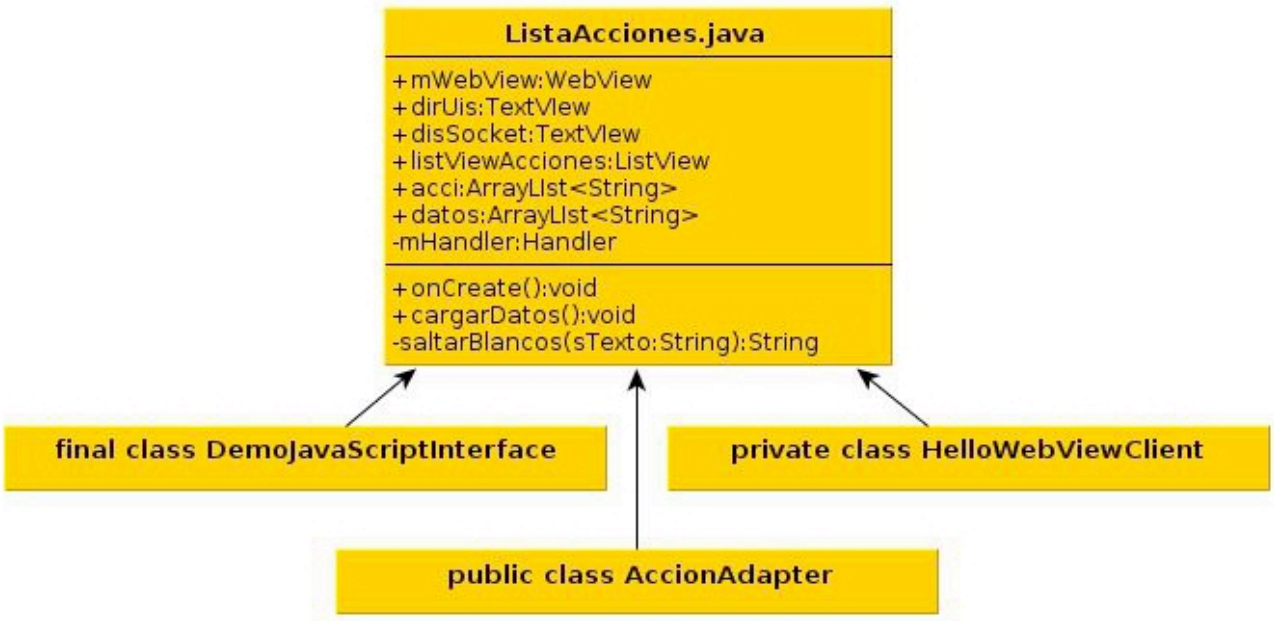


Ilustración 30 Otras clases utilizadas en ListaAcciones.java

La interfaz gráfica de la actividad ListaAcciones.java se carga del fichero lista.xml que únicamente contiene una lista de objetos, cada objeto representa al nombre de una acción.

Ilustración 31 Pantalla de ListaAcciones.java

5.8 Otras clases

Tanto RealizarAccion.java como SatelliteBoxSeleccionado.java usan dos clases, DemoJavaScriptInterface y HelloWebViewCliente. Las actividades ListaAcciones.java y UiSeleccionada.java, además de usar las 2 clases anteriores, también usan la clase AccionAdapter.

5.8.1 JavaScriptInterface

Es una clase final utilizada para obtener datos de las llamadas que se producen a los métodos JavaScript contenidos en los ficheros HTML.

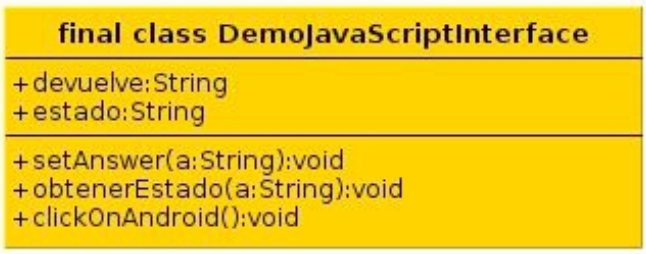


Ilustración 32 Clase DemoJavaScriptInterface.java

Pasos y explicación:

Activar la ejecución de JavaScript en nuestro *WebView*:

```
WebView.getSettings().setJavaScriptEnabled(true);
```

Se añade la clase en el *WebView*, se usa para realizar la conexión con UCH, le ponemos un nombre cualquiera, en este caso "demo", para luego poder referenciarlo dentro del HTML:

```
WebView.addJavascriptInterface(new DemoJavaScriptInterface(), "demo");
```

Cargamos el fichero HTML en el *WebView*:

```
WebView.loadUrl("file:///android_asset/indexGenerico.html");
```

Normalmente un *WebView* se muestra en pantalla como un navegador, para nuestras necesidades no se muestra, ya que se usa como un intermediario entre el código Java y los ficheros JavaScript. Si se desea que aparezca el *WebView*, bastaría con añadir un *WebView* en el fichero XML (interfaz gráfica) que le corresponde a la actividad y referenciarlo en el código Java.

Ahora ya se puede ejecutar los métodos JavaScript de nuestro HTML. Se lanza el método *init* de indexGenerico.html:

```
WebView.loadUrl("javascript:init('"+IP+"')");
```

Porción de código de indexGenerico.html:

```
1. <script language="javascript" type="text/javascript">
2.   updateInterval = 2000;
3.    function init(thisValue){
4.       try {
5.         var str= thisValue;
6.         org.myurc.urchttp.cargar(str);
7.         window.demo.setAnswer(str.toString());
8.         org.myurc.webclient.init(thisValue, updateInterval);
9.         }catch (ex) { alert(typeof ex…essage);}
10.   }
11. </script>
```

Se puede observar en la línea 7 que se llama al método *setAnswer()* de la interfaz JavaScript (*DemoJavaScriptInterface*) que hemos llamado demo.

```
public void setAnswer(String a){   devuel=a;}
```

Como *DemoJavaScriptInterface* es una clase instanciada dentro de la actividad, se puede pasar parámetros desde los ficheros JavaScript al código java de la actividad.

Ej. Cuando se ejecuta *setAnswer*, actualiza la variable global de la actividad con el valor del parámetro pasado desde el código JavaScript.

5.8.2 WebViewClient

Es una clase privada y extiende a *WebViewClient*. Es una especie de cliente del *WebView* que se encarga de la gestión de las peticiones a URL.

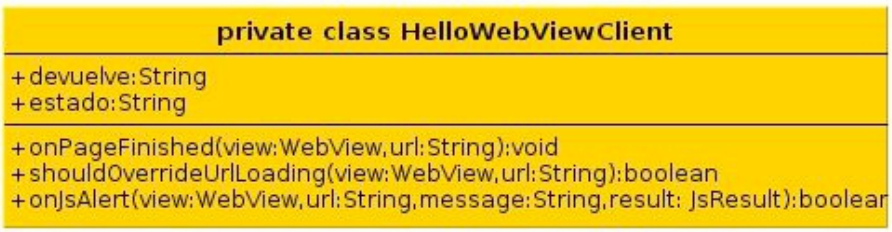


Ilustración 33 Clase HelloWebViewClient

onPageFinished(): Este método se ejecuta cuando la página ha terminado de cargarse en el *WebView*.

shouldOverrideUrlLoading(): Carga una nueva URL en el *WebViewClient.*

onJsAlert(): Lanza un mensaje de alerta JavaScript.

El cliente se carga de la siguiente manera:

```
mWebView.setWebViewClient(new WebViewClient());
```

Tras ello se puede cargar la URL:

```
mWebView.loadUrl("file:///android_asset/indexLuz.html");
```

Ej. Al terminar de cargarse la página indexLuz.html, se ejecutará el siguiente método:

```
public void onPageFinished(WebView view, String url) {
   mWebView.loadUrl("javascript:init('"+tempip+"')");
}
```

5.8.3 AccionAdapter

Es una clase pública que extiende a ArraAdapter<myParsedXMLDatsetAcciones>.

Se encarga de cargar una lista de myParsedXMLdataSetAcciones en un adaptador para luego cargarlo en la vista y poder ver la lista en pantalla.

```
final ListView ListaAcciones=new ListView(this);
// Se le pasa el tipo de lista que tiene que rellenar y los elementos que debe contener.
AccionAdapter adaptador =new AccionAdapter(this,R.layout.lista_item,
parsedExampleDataSet2);
// asocia el adaptador con los datos al ListView que se verá en pantalla.
ListaAcciones.setAdapter(adaptador);
```

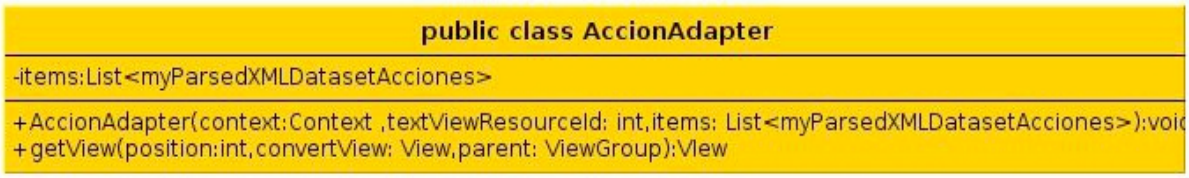


Ilustración 34 Clase AccionAdapter

5.9 RealizarAccion

Al pulsar en una acción, la aplicación muestra una nueva actividad llamada RealizaAccion.java. Esta actividad contiene la información correspondiente a la acción seleccionada. La interfaz grafica se carga a partir del fichero acción.xml, mostrando una explicación de la acción, un campo editable con el valor del parámetro de la acción y dos botones. Uno para enviar el nuevo valor de la acción para el dispositivo seleccionado y el otro botón, para volver al menú anterior. Aparte aparece la publicidad.

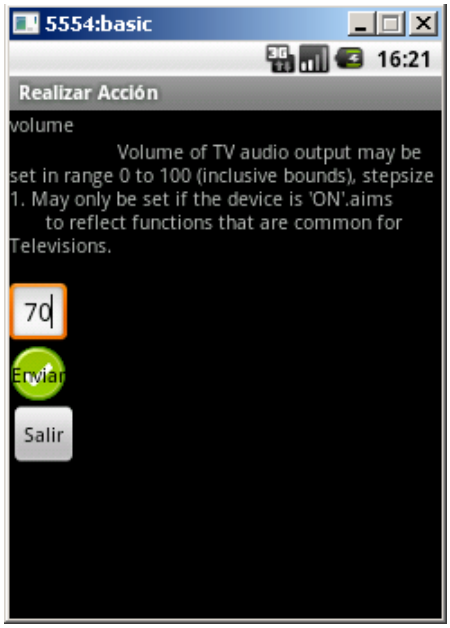


Ilustración 35 Pantalla de RealizarAccion.java

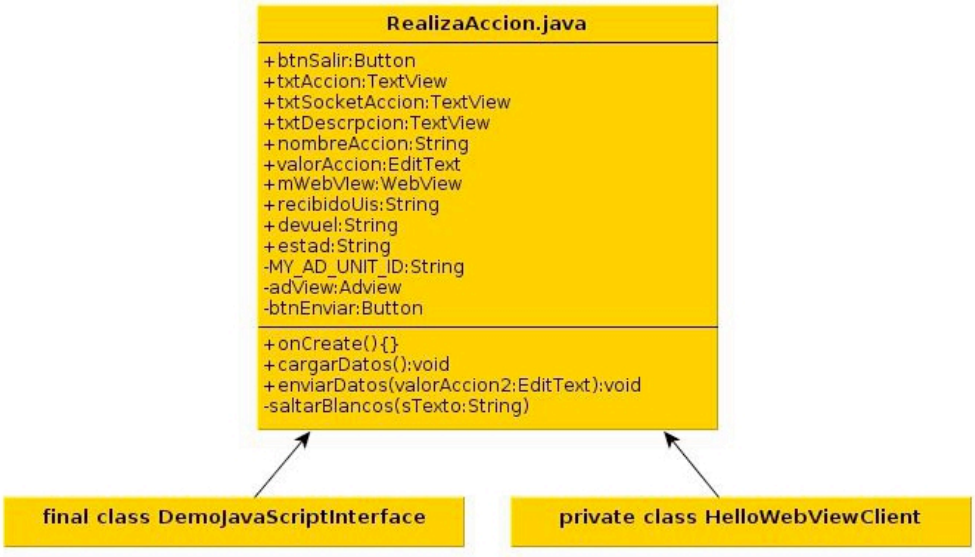


Ilustración 36 Clases utilizadas en RealizarAccion.java

5.10 UiSeleccionada

Tanto la actividad UiSeleccionada.java, como la actividad RealizarAccion.java están formados por dos partes: una parte nativa (Java) y otra parte web (*WebView*[7]).

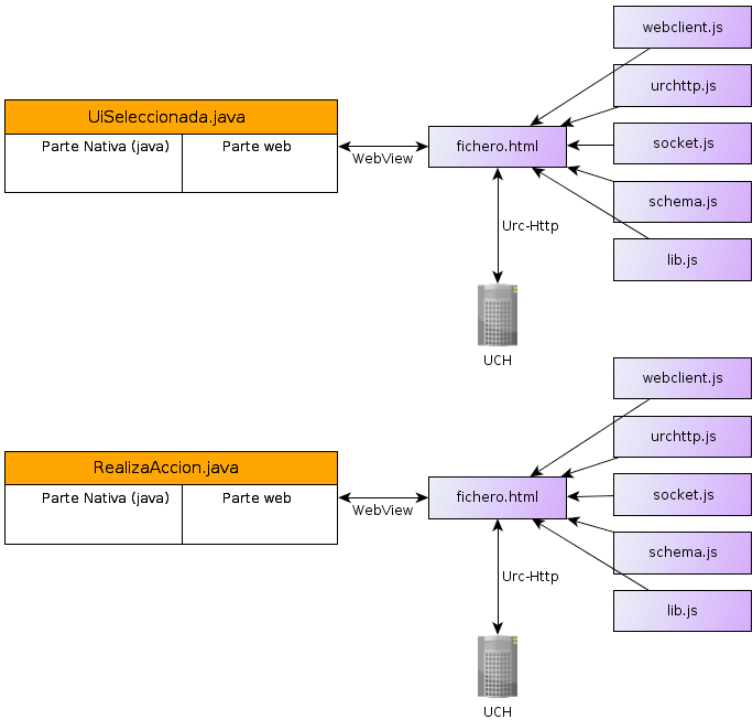


Ilustración 37 Conexión de la aplicación con UCH

En la clase RealizarAccion.java mediante la función:

```
mWebView.loadUrl ("file:///android_asset/indexGenerico.html" ")
```

Carga el fichero HTML en un *WebView*. Este fichero HTML contiene una serie de métodos que se llaman mediante la función:

```
mWebView.loadUrl("javascript:nombreMetodo('"+parametro1+"','"+parametr
o2+"'…)");
```

En el caso del fichero indexGenerico.html, se carga en la actividad RealizarAccion.java y contiene dos métodos.

El primero se encarga de inicializar la conexión de la sesión para poder usar el protocolo URC-HTTP, y de obtener el valor del parámetro de la acción seleccionada para mostrar el valor actual de la acción. Se usa al comienzo del método *onCreate()* de la actividad RealizarAccion.java, es decir, nada más cargar la actividad mediante la

[7] Un *WebView* es un objeto que se usa para cargar una página HTML entre otras funciones.

siguiente línea:

```
// pasamos como parámetro el socket.
mWebView.loadUrl ("javascript:init('"+so+"')");
```

El segundo método se encarga de establecer el nuevo valor para la acción seleccionada. Se lanza cuando se presiona en el botón de enviar, es decir, tiene un evento btnEnviar.setOnClickListener() sobre el botón de enviar de la vista o actividad actual que llama al siguiente método:

```
//  Pasamos la dirección del socket junto con el nombre de la variable y el nuevo valor
para la variable indicada.
mWebView.loadUrl("javascript:enviarValor('"+socketConect+"','"+valEnvi
+"')");
```

Si en la actividad menuUisDetectadas.java se selecciona un dispositivo para el cual tengamos diseñada su interfaz, la aplicación nos guiará hasta la actividad UiSeleccionada.java.

Como se ha comentado, esta pantalla es una actividad hibrida entre parte nativa y parte web como la clase RealizaAccion.java, pero en este caso no disponemos únicamente de un *layout* o interfaz gráfica, dependerá del dispositivo que se haya seleccionado.

En este proyecto, por ejemplo, se han diseñado tres interfaces para tres dispositivos. La interfaz de la luz y la del MP3 se ha implementado en un mismo fichero, layoutuiseleccionada.xml, ocultando los elementos gráficos pertinentes, y el satelliteBox en un único fichero, tvlayout.xml. Dependiendo de si se ha seleccionado un dispositivo u otro, se cargará layoutuiseleccionada.xml o tvlayout.xml.

Aparte de los ficheros para cargar la interfaz gráfica dependiendo del dispositivo seleccionado, como cada uno tiene distintas variables, se tiene un fichero HTML para cada dispositivo implementado. Aunque se podría hacer todo en un mismo fichero llamando a las funciones que les tocase, por claridad, se ha optado por separar en varios.

Ej.

```
if (socket.equals("http://192.168.1.33/UCH/pfc/ta/upnp/light/upnpLight.uis")
     setContentView(R.layout.layoutuiseleccionada);
mWebView.loadUrl("file:///android_asset/indexLuz.html");
…
mWebView.loadUrl("javascript:mostrar()");
…
else if (socket.equals("http://192.168.1.33/UCH/pfc/ta/upnp/mp3/upnpMp3.uis")){
setContentView(R.layout.layoutuiseleccionada);
     mWebView.loadUrl("file:///android_asset/indexMp3.html");
    …
     mWebView.loadUrl("javascript:funcPlay()");
    …
else if(socket==http://192.168.1.33/UCH/SatelliteBoxNQT/satellitebox.uis"))
     setContentView(R.layout.tvlayout);
    mWebView.loadUrl("file:///android_asset/index.html");
    …
mWebView.loadUrl("javascript:obtenerVolume()");
…
```

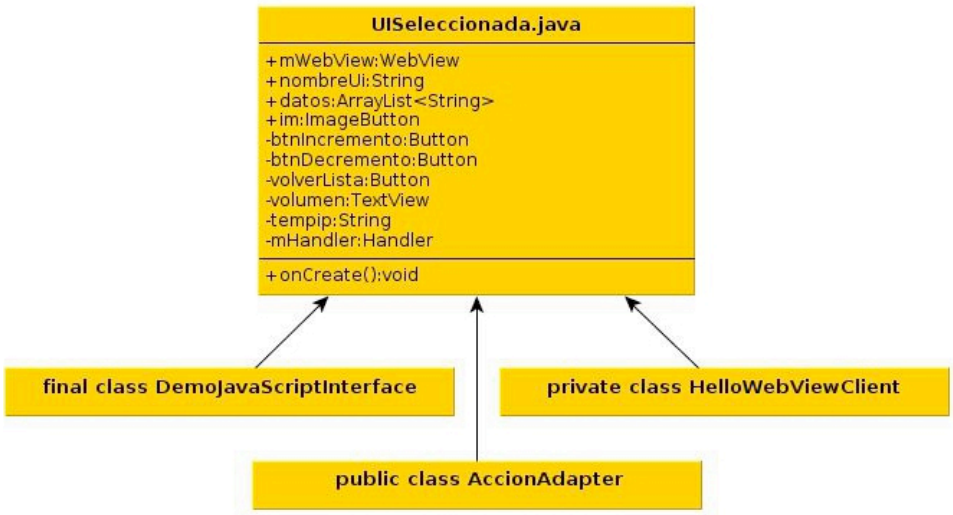


Ilustración 38 Clases utilizadas en UiSeleccionada.java

5.11 Ejemplo concreto. Interfaces específicas: Bombilla.

Como se ha descrito en este capítulo, hay dos tipos de interfaces diferentes dependiendo de si el dispositivo o *target* a controlar es conocido previamente o no. Si es conocido previamente se ha creado una interfaz concreta para ese dispositivo. Al comienzo del proyecto se ha realizado todos así, es decir, se crea una interfaz específica para cada dispositivo que se fuera a utilizar. Se ha diseñado interfaces específicas para la televisión (SatelliteBox), la bombilla, lavadora y MP3.

Pero cuando se desea controlar un dispositivo que no se conoce previamente, se genera dinámicamente la interfaz gracias a ListaAcciones.java, que obtiene las distintas acciones de un *target* desconocido, y a realizarAccion.java que muestra una descripción de la acción y su valor, y permite modificar el valor de la acción seleccionada (parte 7 y 8 de este tema). Las llamaremos interfaz genérica. Dado que con las interfaces genéricas están más explicadas en este tema, a continuación se indica un ejemplo de interfaz específica.

Tras arrancar la aplicación se puede observar una lista con una serie de elementos (a esta pantalla le corresponde la actividad Princial.java), pulsar en "VerTerminales". Esto nos lleva a la siguiente actividad (MenuUisDetectadas.java), en la cual aparecen todos los dispositivos que ha descubierto la aplicación. En esta actividad se selecciona upnpLight.uis, que es la que le corresponde a la bombilla. Tras seleccionarla, la aplicación comprueba si se tiene implementada la interfaz para el dispositivo seleccionado. Si lo está, se pasa a la actividad UiSelecionada.java, pasándole el nombre del dispositivo que se desea controlar en una variable llamada uiseleccionada (Línea 4 de la porción de código indicada abajo). Si no, se pasa a la actividad ListaAcciones que se explica en el apartado 7 de este capítulo.

Porción de código de MenuUisDetectadas.java:

```
String direcc2=ListaDispositivos.getItemAtPosition(arg2).toString();
if(dispositivoEsta(direcc2)){ //direcc2 valdrá upnpLight.uis
  Intent intent = new Intent(menuUisDetectadas.this, UiSeleccionada.class);
  intent.putExtra("uiseleccionada", direcc2);
  finish(); //finalizamos la actividad menuUisDetectadas
  startActivity(intent); // lanza la actividad UiSeleccionada
}
```

Ilustración 39 Captura de pantalla de MenuUisDetectadas.java

Como la bombilla está implementada, se pasa a la actividad UiSeleccionada. Al cargar esta actividad (en el método *onCreate*), se carga el *layout* (interfaz) que le corresponde. En este caso se carga layoutuiseleccionada con la siguiente línea de código:

```
setContentView(R.layout.layoutuiseleccionada);
```

Se debe crear tantas variables en la clase java (de la actividad UiSeleccionada), como elementos de la interfaz gráfica que vayamos a usar (layoutuiseleccionada), es decir, se establece una relación entre la interfaz gráfica y la lógica de negocio de la aplicación. Ej.

```
TextView nombreUi; // como variable global
nombreUi = (TextView)findViewById(R.id.textViewUiSeleccionada); // relación
ToggleButton btnBoton2 = (ToggleButton)findViewById(R.id.BtnBotonLuz);//relación
Bundle recibido = getIntent().getExtras(); //para recibir las variables enviadas
if(recibido !=null) nombreUi.setText(recibido.getString("uiseleccionada"));
```

Tras todo esto se carga el fichero HTML con contenido JavaScript que le corresponde a la bombilla (indexLux.html).

```
mWebView.loadUrl("file:///android_asset/indexLuz.html");
```

Este fichero HTML tiene una serie de *scripts* que se encargan de cargar otros ficheros JavaScripts (.js). Estos últimos ficheros son los que implementan el protocolo URC-HTTP, por lo tanto, al cargarlos se puede usar los métodos de conexión de dicho protocolo. También se dispone de un *script* que contiene un método llamado *init*(), que se encarga de realizar la conexión entre la aplicación y el servidor UCH mediante el protocolo URC-HTTP (usando los ficheros cargados mediante los *scripts* mencionados anteriormente), y obtiene el valor actual de la bombilla. A un *script* se le llama de la

siguiente manera:

mWebView.loadUrl("javascript:nombreScript()");

Ej.

mWebView.loadUrl("javascript:init()");

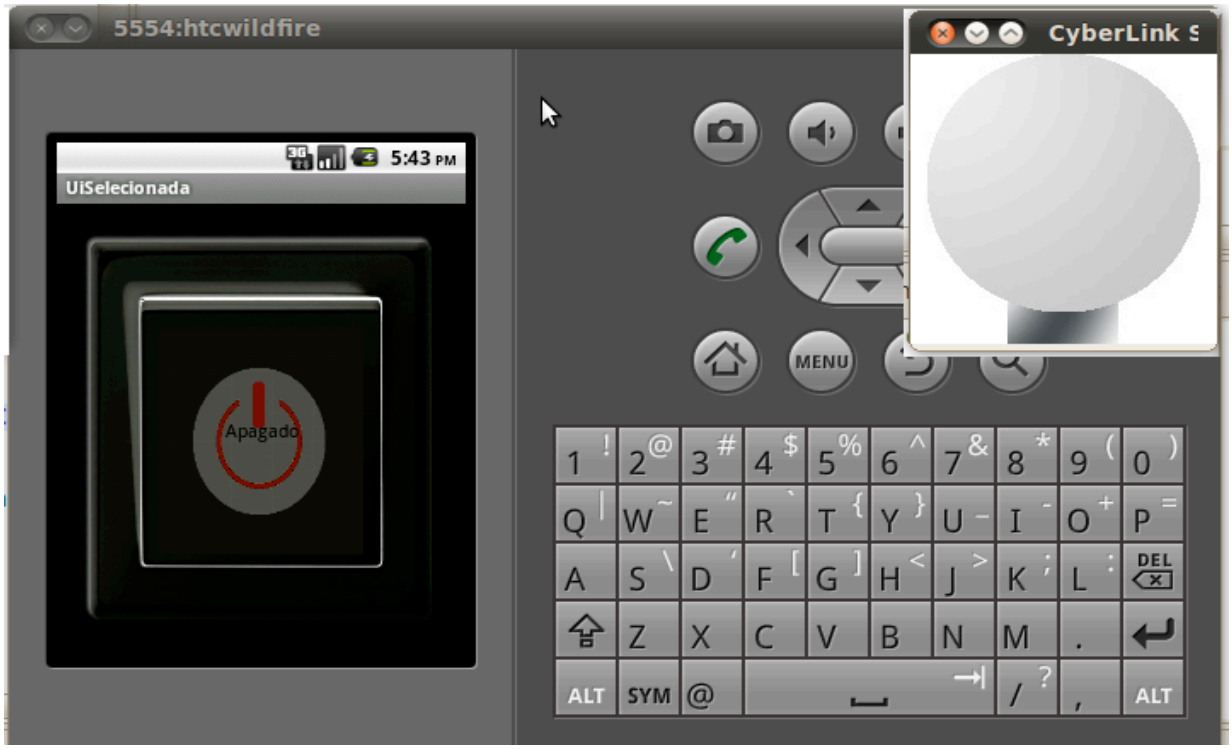


Ilustración 40 Captura de pantalla de UiSelecionada.java con la bombilla apagada

También se dispone de un *script* para cambiar el estado de la bombilla:

```
<script language="javascript" type="text/javascript">
var estadoActual=
org.myurc.webclient.getValue("http://pfc.cybergarage.upnp/light/controlSocket1#/estado");;
    function cambiar(){
        if(estadoActual=="0"){
                estado="1";
        }else{
                estado="0";
        }
org.myurc.webclient.setValue('http://...upnp/light/controlSocket1#/estado', estado);
        }
</script>
```

Cada dispositivo tiene una serie de variables para poder controlarlo, en este caso, la bombilla solo tiene una variable llamada estado, que representa con valor 0 si la bombilla está apagada y con valor 1 si esta está encendida.

Por último, se tiene el botón btnBoton2 que tiene un evento que cada vez que se pulsa, cambia el estado de la bombilla llamando al servidor UCH con la siguiente línea:

```
mWebView.loadUrl("javascript:cambiar()");
```

Ilustración 41 Captura de pantalla de UiSeleccionada.java con la bombilla encendida

Capítulo 6 CONCLUSIONES Y LÍNEAS DE TRABAJO FUTURAS

6.1 Resumen de los logros

El objetivo principal del proyecto era la construcción de una aplicación para *smartphones* que fuera capaz de controlar dispositivos compatibles con UCH.

Primero se estudió el estándar URC, la estructura y el funcionamiento de UCH y el protocolo URC-HTTP. En mi opinión, es difícil estandarizar el protocolo para todos dispositivos. Las compañías productoras de estos dispositivos son las primeras a las que no les interesa. Se piensa que en unos pocos años todos los dispositivos tendrán un acceso Wi-Fi, con lo que el control total de ellos estará a disposición de nuestro terminal.

Tras ello se realizó un estudio sobre el mercado de los *smartphones*, en el que cada día se realizan más avances. Existe una fuerte lucha por dominar el sector. No solo están iPhone y Android, recientemente Nokia se está recuperando con la presentación de su *smartphone* "Lumia" en 3D. Hace unos pocos años un terminal móvil solo servía para llamar o enviar sms, hoy en día se usa para todo (GPS, MP3, video, internet, juegos con alto nivel gráfico,…) creando una dependencia entre el usuario y el *smartphone*.

En el siguiente paso se decidió sobre qué sistema operativo se iba a desarrollar el cliente para UCH. Se eligió Android por ser libre y menos restrictivo que para iPhone (hace falta un Mac para desarrollar aplicaciones entre otras desventajas), además ya existía una aplicación (no nativa) para iPhone de los creadores del estándar URC (http://myurc.org/). Por estos motivos nos decidimos por Android. La versión para iPhone no es nativa para iOS, ya que utiliza únicamente un navegador para comunicarse con UCH y este con los dispositivos. Otra diferencia respecto a la aplicación desarrollada para iOS es que no generan una interfaz genérica para todos los dispositivos, solo se puede mostrar interfaces de control para los dispositivos conocidos previamente.

Una vez elegido sobre qué sistema operativo iba a realizar la aplicación, me puse a estudiarlo. Entender la estructura y el funcionamiento de Android, desde su núcleo *kernel* hasta su nivel más alto. Antes de comenzar a desarrollar, tuve que aprender la estructura básica de un proyecto Android y su funcionamiento.

En este punto se encontró un problema. La aplicación realizada en el proyecto debía de conectar con el servidor UCH a través del protocolo URC-HTTP[8]. Los ficheros que lo componen son ficheros JavaScript (.js), por lo tanto se deseaba rescribir el protocolo a Java. Entonces se comenzó a pasar el código, pero es un protocolo que entre los 5 ficheros que lo componen reúnen más de 3.000 líneas de código. Ya que se salía del objetivo del proyecto y dado el excesivo trabajo sin saber si luego podría funcionar, se decidió investigar si se podía realizar *scripting*, es decir, cargar un fichero HTML que lance una serie de *script* que contengan a su vez ficheros JavaScript.

Se hubiera ahorrado mucho trabajo si directamente se hubiera cargado en un

[8] El borrador del protocolo puede descargarse de http://myurc.org

WebView un fichero HTML, que contenga la interfaz gráfica de la aplicación y se encargue de la comunicación con UCH mediante el protocolo URC-HTTP, protocolo que está formado por una serie de ficheros JavaScript que se cargan en este HTML. Este fichero HTML podría estar colgado en el servidor junto a UCH.

Lo único que se tendría que implementar de la parte de Android sería el menú principal y la carga de un fichero HTML en un *WebView*. Pero como un objetivo del proyecto era realizar la aplicación de la aplicación de la manera más nativa posible (Java) y no que pareciera únicamente una aplicación web en Android. Por eso se optó por realizar una aplicación mixta, parte web y parte nativa.

Además, disponer de una arquitectura mixta (aplicación nativa + una aplicación web) tiene ventajas como:

- Ahorro de tiempo de desarrollo.
- Permite combinar o comunicar elementos web con los elementos nativos de Android.
- No hay que distribuir la aplicación web (HTML, JavaScript, CSS, etc.) cuando hay un cambio en la misma, con cambiar el código en el servidor bastaría.

Se ha implementado las interfaces gráficas en nativo, así como las comunicaciones con UCH usando el protocolo URC-HTTP (cargado en la aplicación web).

Tras realizar la primera versión de la aplicación, la cual permitía detectar y controlar dispositivos específicos conocidos, pasé a implementar una segunda versión que fuera capaz de detectar y controlar cualquier dispositivo sin previo conocimiento de sus funcionalidades.

6.2 Conclusión

El área de la domótica[9] nació hace muchos años, pero hay un problema, que existen muchos estándares como X-10 (1978), KNX, C-Bus, InstabusETB(1984), LonWorks(1992), EHS(1987), Batibus, etc. Pero todavía no existe un estándar universal que facilite la competencia entre fabricantes y la libre elección de distintos elementos por parte de los clientes. Dada la gran variedad de dispositivos, sensores y actuadores que posee una casa, es compleja la búsqueda de un estándar capaz de controlarlos todos.

Una línea por la que se podría conducir el proyecto sería la construcción de un sistema domótico estándar, universal y ubicuo, permitiendo la monitorización de los dispositivos de la vivienda de forma remota desde nuestro teléfono Android (u otro sistema operativo). También se podría disponer de un *tablet* fijo en la vivienda para mayor comodidad y un control dentro de la vivienda y no depender únicamente del terminal móvil.

En mayo de 2011 Google presento Android@Home, que engloba todos los protocolos necesarios para poder controlar nuestra vivienda. Busca la estandarización de todos los sistemas.

En mi opinión es un gran paso hacia la estandarización de la domótica y poco a poco los fabricantes deberán de adaptarse a los requisitos, pero eso depende de la aceptación por parte de los usuarios.

Mi idea es bastante similar, un punto de control de sistemas UPnP que hace de servidor UCH y dispositivos con sistema UPnP para cada uno. Así se podría llegar a controlar fácilmente cualquier dispositivo aunque previamente no lo conozcamos (ubicuidad).

[9] La domótica es el campo que se encarga del estudio e implantación de sistemas automáticos de los sistemas eléctricos en casas o edificios.

6.3 Líneas Futuras

Dadas las posibilidades que ofrecen la tecnología hoy en día, existen muchas líneas por las que se podría seguir trabajando relacionadas con este proyecto. Algunas de estas ideas son:

- Otra alternativa viable y con futuro dado que es soportado por distintos sistemas operativos, es la implementación del URC sobre HTML5, ya que existen herramientas que a partir de HTML5 encapsula y crea la aplicación para el sistema operativo deseado. Ej. Titanium, Phonegap, etc.
- Mejora del descubrimiento de UCH´s, lanzando un *script* para obtener las distintas redes (IP´s) detectadas por nuestro dispositivo (URC) y probar si tiene lanzado UCH. No necesitamos conocer más, con la IP nos basta para acceder y manejar los distintos dispositivos detectados por ese UCH.
- Mejora de la seguridad en UCH. Esto se puede conseguir mediante HTTPS configurando previamente apache.
- Además, se puede mejorar con la implementación de un control de usuarios en Android, que consulte el fichero UIList y este a su vez consulte al *UCH Control Layer*, y decidir si esta registrado o no el usuario.
- La construcción de *Widgets* en Android permitiendo así un acceso más rápido a los dispositivos detectados.
- Ampliación de la funcionalidad de la aplicación mostrando los usuarios conectados al UCH o la localización de los mismos mediante el GPS o Bluetooth.
- Posibilidad de añadir notificaciones, como por ejemplo, para avisar al usuario del estado de un dispositivo o programar tareas sobre los dispositivos a una determinada hora o periodo. Esta implementación correrá en segundo plano y simplemente nos informará con un mensaje de notificación, sin interrumpir nuestra actividad.
- Implementar en Android la conexión con los dispositivos mediante Bluetooth, ya que gasta menos batería que con Wi-Fi, y ese es uno de los puntos flacos de los *smartphones* actuales.
- Adaptación de la aplicación para personas ciegas con el uso de voiceXML.
- Creación de interfaces adaptadas según el usuario.
- Añadir funcionalidad a la aplicación para que se permita descargar interfaces diseñadas por otros usuarios.
- Mejorar la aplicación permitiendo acceder a los ficheros HTML desde internet o en el servidor junto a UCH, sin la necesidad de tener que introducirlos en el *assert* (carpeta de un proyecto Android). Este HTML contendría la interfaz específica y control para dispositivos desconocidos que deseamos tener para cada usuario. Entonces, con una comunidad de usuarios podrían disponer de distintas interfaces, para que según los requisitos o necesidades de los usuarios dispongamos de una interfaz u otra, independientemente del sistema operativo del *smartphone* usado.

Capítulo 7 BIBLIOGRAFÍA

- [Gamecho,2009] B.Gamecho. Estudio del Estándar URC y su Aplicación a Sistemas Embebidos Usando una Implementación en JAVA del UCH. Proyecto final de carrera.

- Documentación de peticiones HTTP de Android a Apache:

http://developer.android.com/reference/org/apache/http/package-summary.html

- Manuales de Android:

http://www.softwarelibre.ulpgc.es/cursos/android
http://www.sgoliver.net/blog/?p=1313

- Foros de desarrolladores Android:

http://groups.google.com/group/desarrolladores-android?pli=1
http://www.android.es/category/aplicaciones#axzz1bhygzcQS

- Uso de HTTP desde Android:

http://www.android-spa.com/viewtopic.php?p=49440&sid=857b00397b61ddd25af47b0a4463a789

- Ejemplos de clientes HTTP para Apache:

http://hc.apache.org/httpcomponents-client-ga/examples.html

- Exportación de una aplicación Android:

http://www.htcmania.com/archive/index.php/t-127196.html

- Especificación del protocolo URC-HTTP:

http://myurc.org/TR/urc-http-protocol2.0-20091103/
http://myurc.org/TR/urc-http-protocol/

- Documentación de UCH:

http://myurc.org/TR/uch/

- Uso de JavaScript y *WebView*:

http://www.android-spa.com/viewtopic.php?t=7497&highlight=
http://developer.android.com/guide/webapps/webview.html
http://developer.android.com/resources/articles/using-webviews.html

http://www.androidizados.com/desarrollo/2011/04/26/admob-para-desarrolladores-ingresos-mediante-publicidad/

- Instalar entorno Android en Eclipse:

http://developer.android.com/sdk/installing.html

- Proyecto UHC 2.0:

http://sourceforge.net/projects/traceurcsdk/files/UCHe/2.0/UCHe.2-0.tar.gz/download

- Pasos de instalación UCH:

http://myurc.org/tools/UCHj/Installation.php

- Inteligencia ambiental:

[Aarts 2009] E. Aarts and R. Wichert, "Ambient intelligence," in *Technology Guide*, vol. 3, no. 3, 2009, pp. 244 - 249.

- Sistemas ubicuos:

[Posland 2009] S. Posland, *Ubiquitous Computing Smart Devices, Environments and Interactions*. Wiley, 2009, p. 473.

CAPÍTULO 8 ANEXOS

Anexo 1 Estudio de la plataforma Android

Arquitectura, Estructura y Funciones

En este anexo se presenta las características principales de la plataforma, explicando detalladamente su arquitectura: familia de procesadores, la estructura de sus aplicaciones, cómo funcionan sus comunicaciones con el exterior, cómo es el diseño de las aplicaciones y qué posibilidades gráficas tiene.

1.1 Familia de procesadores para Android

En los dos últimos años está habiendo un crecimiento en cuanto a la venta de smartphones y se prevé que en 2013 se vendan 750 millones de ellos. Antes Qualcomm dominaba el sector de fabricación de procesadores para Android, ahora (2011), otras compañías también se han metido en este mercado.

La competencia que existe, se debe también en gran medida a las prestaciones que tienen que soportar, como video en HD a 1080p, Adobe Flash, juegos con gráficos potentes, internet más rápido, captura y reproducción de imágenes en 3D, etc.

Actualmente los procesadores se ajustan a dos tipos de arquitecturas ARM (RISC) y x86(CISC). ARM ha tenido el monopolio de implantación de núcleos para dispositivos Android durante muchos años. Actualmente ARM se encuentra en los procesadores QualComm, nVidia y Samsung, mientras que Intel que comenzó también con ARM, se está decantando por x86.

Veamos someramente los procesadores de cada una de las 4 compañías que dominan el mercado. Muchos de los procesadores están montados sobre el mismo núcleo, pero la diferencia entre unos y otros radica en otros componentes denominados SOC*(system on a chip*), como son *chips* gráficos (GPU, procesador gráfico), comunicaciones extra, etc.

QualComm:

Procesadores basados en la arquitectura ARM. En el año 2007, se lanzó el procesador QualComm MSM7200A utilizado en el HTC Hero, LG GW620, Motorola Backflip y Samsung Galaxy. Alcanza frecuencias de 528 MHz.

A finales de 2008 apareció su procesador más famoso, el SnapDragon, con una frecuencia de 1Ghz usado en casi todos los terminales más avanzados de 2010. Diseñado para decodificar video de alta definición (720p) y dar gran rendimiento en gráficos 3D. En junio de 2011 salió al mercado el modelo HTC Sensation. Este modelo cuenta con un procesador 1,2 Ghz Dual Core QualComm SnapDragon.

El futuro de QualComm se llama SnapDragon S3, alcanza 3,4Ghz (1,7Ghzx2) gracias a su doble núcleo, y es capaz de reproducir 1080p. Gráficamente llevan incorporados un coprocesador (GPU) que podría llegar a 80 Mt/s[7].

nVidia:

Usa una plataforma llamada Tegra basada también en ARM. En 2010 no estaba disponible en *smartphones*, pero fue integrada en el Adam Tablet equipado con Tegra 2 (2 núcleos) y en la nueva Nintendo Ds, alcanzando velocidades de 1,5Ghz-2Ghz.

También Acer usa el procesador Tegra 2 (diseñado para *smartphones*) en un ultra portátil de 10 pulgadas.

En 2011 apareció el Motorola Atrix y el LG Optimus 2X que cuenta con un procesador 1Ghz Dual Core Nvidia Tegra 2.

En el año 2011 según Nvidia, casi el 10% de los nuevos dispositivos con sistema operativo Android (tabletas + *smartphones*) cuentan con su *chip* Tegra 2.

Su futuro es un procesador con cuatro núcleos, proyecto llamado Kal-El.

Samsung:

Anteriormente usaba procesadores QualComm, pero ya para el Samsung Galaxy S Pro incorpora un procesador basado en ARM, pero fabricado por Samsung. Su nombre es Hummingbird y está montado sobre el núcleo ARM Cortex-A8, el mismo sobre el cual está montado el Tegra de Nvidia y el SnapDragon de QualComm, pero aseguran que tiene un rendimiento 300% mejor que este último. Alcanza los 90 Mt/s[10].

El *smartphone* más alto de la gama, sacado en Junio de 2011, se llama Samsung Galaxy S2 y cuenta con un procesador 1,2Ghz Dual Core Samsung Exynos.

Este año (2012) ha salido el Samsung Galaxy S3, que utiliza un procesador Exynos de de cuatro núcleos a 1,4Ghz.

Intel:

El procesador Z6XX es el primero que permite Android alcanzando 100 fps en el juego Quake III (usado habitualmente para medir el rendimiento gráfico/computacional de los nuevos dispositivos). También es compatible con Meego y Windows 7. Posee un núcleo a 1,5Ghz acompañada de una GPU que da soporte a DX9 y a OpenGl 2.1. Pero a pesar de tener un solo núcleo, reproduce video a 1080p con salidas HDMI.

Este año Intel no tiene previsto sacar ningún *smartphone* con sus procesadores. Aunque si en *tablets* pero a finales de 2012. Uno de los mayores problemas de los *smartphones* es la corta duración de las baterías, pero Intel ha conseguido apagar partes del procesador para ahorrar energía. Permitirá ejecutar aplicaciones x86.

[10] MegaTexels/seg es la cantidad de texturas que puede mostrar en pantalla

Conclusión objetiva en cuanto a procesadores:

Se puede observar que la tendencia está hiendo por aumentar el número de núcleos, pero el problema de la corta duración de las baterías sigue estando presente. Además, *underclockean*[11] los núcleos para que consuman menos y obtengan mejor rendimiento gracias al paralelismo.

Intel lo tiene más complicado que ARM ya que la arquitectura CISC x86 de Intel por su arquitectura, necesita más consumo aunque iguale a potencia a ARM.

1.2 Arquitectura

Una de las características principales de Android es que está creado para que cada aplicación se ejecute en un proceso independiente, con una instancia de la maquina virtual propia y con un UID (identificador de usuario) propio. Con ello se consigue concurrencia e interoperabilidad de procesos, compartiendo información entre ellos.

En la siguiente imagen podemos apreciar la arquitectura de Android:

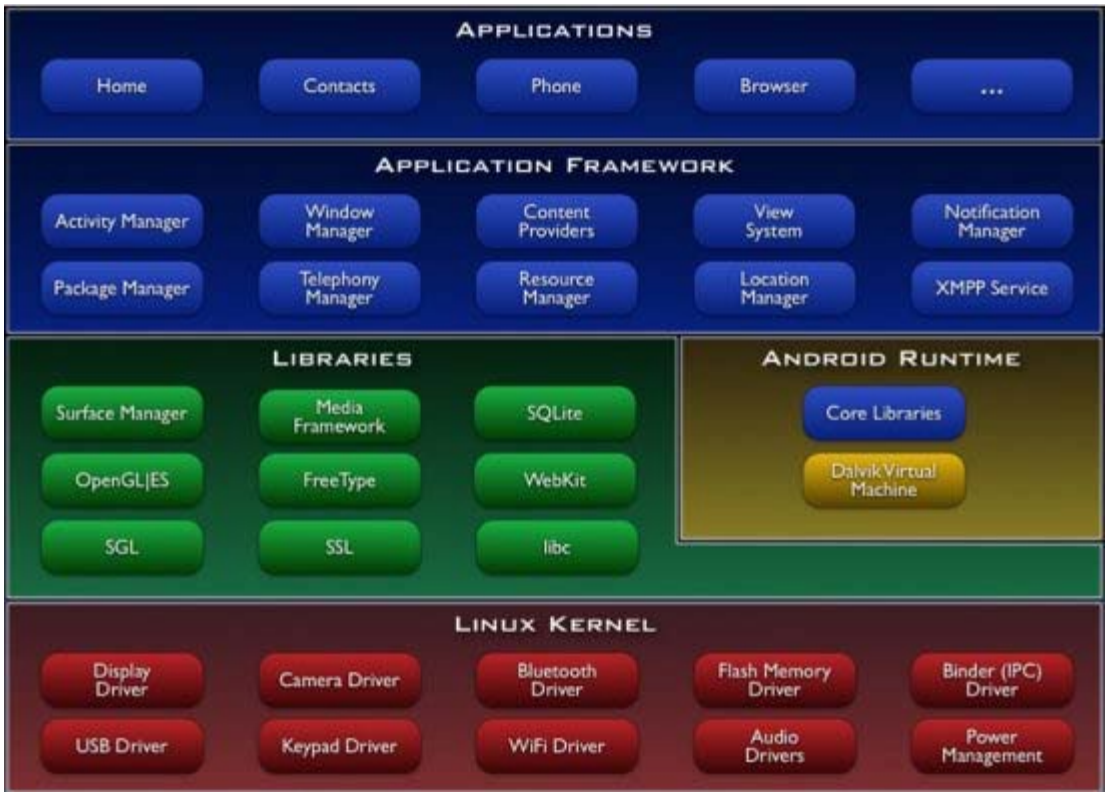


Ilustración 42 Arquitectura Android http://es.wikipedia.org/wiki/Android

Cómo se puede ver se ha construido a partir de un *kernel* base hacia las capas superiores:

[11] Bajar la velocidad de reloj de la CPU

- Linux *Kernel*: Linux 2.6 (control drivers, periféricos, Servicios: seguridad, gestión de memoria, procesos).

- Bibliotecas: Escritas en C/C++ (Gráficos 2D, 3D) reproducción/grabación archivos media, mapa de bits, BBDD.

- *Runtime*: Sobre la maquina virtual de Dalvik (archivos .dex optimizan necesidades de memoria, batería y procesado).

- Framework de Aplicaciones: Acceso completo a APIs (Vistas (Listas, *Grids*, botones, etc.) *Content provider* (gestión acceso) y varios *Managers (Resource, Notification, Activity))*.

- Aplicaciones: Conjunto de aplicaciones (E-mail, calendario, mapas, navegador, contactos, etc.).

En los siguientes apartados se muestra con más detalle cada una de las partes anteriores, además del *hardware* utilizado por los terminales actuales.

1.2.2 Linux

Linux es el núcleo o *Kernel* del Sistema Operativo libre denominado GNU/Linux. Lanzado bajo la licencia pública general (GPL - *General Public License*) de GNU y desarrollado gracias a contribuciones provenientes de todo el mundo.

Usan la versión 2.6 de Linux y es la capa encargada de interconectar el *hardware* del dispositivo con las aplicaciones y todo el *software* disponible utilizando el *hardware* del dispositivo.

Ilustración 43 Linux *kernel*. Funciones principales.

1.2.3 *Runtime* Android

Android incluye un conjunto de librerías que proporciona la mayoría de la funcionalidad, disponible en las principales librerías que están programadas en Java. Por cada aplicación utiliza un proceso, con su propia instancia de la máquina virtual Dalvik.

Ilustración 44 Runtime Android

Dalvik permite ejecutar múltiples MVs (máquinas virtuales). La MV Dalvik ejecuta ficheros en el formato Dalvik ejecutable (.dex) que está optimizado para el mínimo gasto de memoria. La MV está basada en registros y ejecuta clases compiladas mediante un compilador de lenguaje Java. Son transformadas

en el formato .dex mediante la herramienta "dx" incluida en el SDK.

Uno de los motivos por el cual Android se decantó por el uso de Dalvik fue evitar depender de la máquina virtual de Java de Sun.

1.2.4 Librerías

A un nivel superior al *kernel* de Linux contamos con un conjunto de librerías disponibles para el desarrollador de aplicaciones que podemos ver en la siguiente imagen:

Ilustración 45 Librerías

- SGL (*Skia Graphics Library*): Se trata de un motor gráfico implementado en C++ y que puede lanzarse en varios entornos, entre ellos en dispositivos móviles. Está orientado al desarrollo de 2D con la utilización de líneas, rectángulos, óvalos, polígonos y trayectorias.

- SSL (*Secure Socket Layer*): El protocolo SSL es un sistema diseñado y propuesto por Netscape Communications Corporation. Se encuentra en la pila OSI entre los niveles de TCP/IP y de los protocolos HTTP, FTP, SMTP, etc. Proporciona sus servicios de seguridad cifrando los datos intercambiados entre el servidor y el cliente con un algoritmo de cifrado simétrico, el RC4 o IDEA, y cifrando la clave de sesión de RC4 o IDEA mediante un algoritmo de cifrado de clave pública, el RSA. Android utiliza SSLv3.0 o SSLv2.0

- Libc: Consiste en una implementación derivada de BSD de la librería estándar C del sistema (libc), adaptada para dispositivos embebidos basados en Linux. Google la llamó con el nombre de Bionic e implementa servicios como la gestión de excepciones de c++, gestión de propiedades del sistema, sistema de *logs*, etc.

- OpenGL|ES(OpenGL *for Embedded Systems*): es una variante simplificada de la API gráfica OpenGL diseñada para dispositivos integrados tales como teléfonos móviles, PDAs y consolas de videojuegos. La define y promueve el Grupo Khronos, un consorcio de empresas dedicadas a *hardware* y *software* gráfico interesadas en APIs gráficas y multimedia.

- FreeType: Librerías que se encargan de mapas de bits y vectores.

- WebKit: Es un marco de trabajo para aplicaciones que funciona como base para el navegador web Safari y Google Chrome. Está basado originalmente en el motor KHTML del navegador Web del proyecto KDE: Konqueror.

- *Surface Manager*: Es la librería encargada de la gestión de superficies que gestiona el acceso al subsistema de la pantalla y compone capas de gráficos 2D y 3D desde múltiples aplicaciones.

- *Media Framework*: Son las librerías multimedia basadas en OpenCORE PacketVideo que soportan reproducción y grabación de múltiples formatos de audio y video, así también como ficheros de imagen estáticos, incluyendo MPEG4, H.264, MP3, AAC, AMR, JPG, y PNG.

- SQLite: Es un poderoso y ligero motor de bases de datos relacionales que está disponible para almacenamiento de datos en todas las aplicaciones.

1.2.5 *Framework* de Aplicaciones

En esta capa se encuentran utilidades con que cuenta el terminal como el manejador de vistas, manejador de ventanas, notificador de eventos, localizador:

Ilustración 46 Framework de Aplicaciones

- Activity Manager: Conjunto de APIs que controlan el ciclo de vida de aplicaciones y mantiene un "*Backstack*" para que el usuario pueda utilizar otras aplicaciones mientras algunas siguen ejecutándose.

- *Windows Manager*: Gestor de ventanas

- *Content Providers*: Estos objetos encapsulan los datos que hay que compartidos entre las aplicaciones, como los contactos.

- *View System*: Sistema encargado de controlar las vistas de las aplicaciones.

- *Notification Manager*: Eventos llamados *intents*, como mensajes recibidos, citas e infinidad de eventos que se presentan de una manera discreta para el usuario.

- *Package Manager*: Gestor de paquetes para instalar/desinstalar aplicaciones.

- *Telephony Manager*: Es el gestor del teléfono (llamadas, mensajes,…).

- *Resource Manager*: Los recursos son algo que van con su respectivo programa, no es código.

- *Location Manager*: Corresponde al gestor para controlar tu localización y posicionamiento.

- XMPP *Service*: Ofrece servicio para poder utilizar el protocolo de intercambio de mensajes.

1.2.6 Aplicaciones

Esta capa se sitúa en el nivel más superior, aquí se encuentran todas las aplicaciones a las cuales tiene acceso el usuario desde su terminal como pueden ser los contactos, telefonía, navegador, juegos, reproductor multimedia, etc.

Ilustración 47 Capa aplicaciones

1.3 Comunicaciones con Android

La clase padre de las comunicaciones con Android es android.net, encargada de:

- Supervisar las conexiones de red (Wi-Fi, GPRS, UMTS, etc.)

- Enviar intentos de conexión de red.

- Intentar la reconexión a otra red cuando la conectividad a la red se pierde.

- Proporcionar una API que permite a las aplicaciones la consulta del estado de las redes disponibles. Las vías de comunicación con las que cuenta Android dependen directamente del *hardware* que lleve cada terminal, pero a nivel más alto podemos destacar estos dos tipos:

Conexiones Wi-Fi:

Es un sistema de envío de datos sobre redes computaciones que utiliza ondas de radiofrecuencia. En Android estas conexiones se realizan mediante el paquete Android.net.wifi, que cuenta con clases para detectar puntos de acceso, para configurar una conexión Wi-Fi, para configurar métodos de acceso mediante clave o cifrados, para conocer el estado de la conexión o el protocolo. Las clases son las siguientes:

- ScanResult
- WifiConfiguration
- WifiConfiguration.AuthAlgorithm
- WifiConfiguration.GroupCipher
- WifiConfiguration.KeyMgmt
- WifiConfiguration.PairwiseCipher
- WifiConfiguration.Protocol
- WifiConfiguration.Status
- WifiInfo
- WifiManager
- WifiManager.WifiLock

Conexiones Bluetooth:

Es una especificación industrial para Redes Inalámbricas de Área Personal (WPANs) que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia segura y globalmente libre (2,4 GHz.). Actualmente aún no están disponibles (en el último SDK) las clases para poder implementar funcionalidades Bluetooth en las aplicaciones, aunque existen algunas aplicaciones que lo hacen, pero

para su funcionamiento deberemos *rootear*[12] el terminal.

1.4 Estructura de aplicaciones

1.4.1 Definición

Una aplicación Android cuenta con las siguientes características:

Inicialmente cada aplicación tiene su propio proceso Linux. Android inicia el proceso de cualquier aplicación cuando es necesario y termina el proceso cuando ya no se utiliza.

Cada proceso tiene su propia instancia de máquina virtual Java, con el fin de ejecutarla de manera aislada a las demás aplicaciones.

A cada solicitud se le asigna un único identificador de usuario de Linux con los permisos asignados de tal manera que los archivos solo son visibles para el usuario y solo cuando son requeridos, aunque estos permisos son modificables.

Una característica de Android es que una aplicación puede hacer uso de elementos de otras aplicaciones (siempre que esas aplicaciones lo permitan). Por ejemplo, si una aplicación necesita mostrar una lista de desplazamiento de imágenes y otra aplicación se ha desarrollado un *Scroll* dinámico de texto y es pública, se puede llamar al *Scroll* para hacer el trabajo, en lugar de desarrollar uno propio. Esta solicitud no implica incorporar el código sino que simplemente se inicia la aplicación cuando surge la necesidad de ejecutarla. Para que esto funcione, el sistema debe ser capaz de iniciar un proceso de solicitud, cuando cualquier parte de ella se necesita, y la instancia de objetos Java se encargara de invocarla.

A diferencia de la mayoría de las aplicaciones en otros sistemas, las aplicaciones de Android no disponen de un único punto de entrada para toda la aplicación (no hay función main (), por ejemplo). Por el contrario, el sistema puede crear una instancia y ejecutar, según sea necesario.

[12] Proceso por el cual se obtienen permisos de administrador, sin tener impuestas las restricciones de fabricantes.

1.4.2 Componentes

Toda aplicación desarrollada con Android se basa en cuatro bloques principales:

Activity (Actividad):

Las actividades son extensiones de la clase Java *Activity* del paquete android.app. Por lo general, cada *Activity* está relacionada con una pantalla visible (UI) y responde a eventos iniciados por el sistema o por el usuario. Por ejemplo, una actividad podría presentar una lista de los elementos del menú donde el usuario puede elegir o puede mostrar fotografías junto con sus leyendas. Una aplicación de mensajería de texto puede tener una actividad que muestra una lista de contactos enviar mensajes a una segunda actividad como escribir mensajes a los contactos elegidos, y otras actividades para revisar los mensajes antiguos o cambiar los ajustes. En resumen se denomina actividad a toda translación de vista en la pantalla. A pesar de que las actividades trabajan juntas para formar una coherente interfaz de usuario, cada actividad es independiente de los demás.

Broadcast Intent Receiver (Receptor de radiodifusión):

No dispone de una interfaz gráfica propia y reacciona a la emisión de eventos. Muchos eventos se originan en el sistema de código, por ejemplo, anuncia que ha cambiado la zona horaria, que la batería está baja, que una imagen se ha tomado, o que el usuario cambia el idioma de preferencia. Las solicitudes también pueden iniciar los eventos, por ejemplo, para que otras aplicaciones controlen que algunos datos han sido descargados en el dispositivo y estén disponibles para su uso. Una aplicación puede tener cualquier número de receptores de radiodifusión para responder a cualquier anuncio que considera importante. Las notificaciones que el usuario puede percibir se muestran de distintas maneras, por ejemplo se pude llamar a parpadear la luz de fondo, ejecutar el dispositivo de vibración, un sonido, y así sucesivamente. Normalmente se utiliza un persistente icono en la barra de estado, que los usuarios pueden abrir para obtener el mensaje.

Service (Servicio):

Un servicio no tiene un interfaz de usuario visual, sino que se ejecuta en segundo plano por un período indefinido de tiempo. Por ejemplo, un reproductor de música que el usuario está escuchando mientras esta interactuando con el terminal, realizando otras acciones como enviar mensajes. El sistema mantiene el funcionamiento del servicio de reproducción de música, incluso después de que comience la actividad y que salga de la pantalla. Pero pueden tener una interfaz para poder controlar el servicio, por ejemplo para el caso anterior del reproductor, tiene una interfaz con la que se puede detener, rebobinar, detener y reiniciar la reproducción. Al igual que las actividades y el resto de los componentes, los servicios se ejecutan en el hilo principal del proceso de solicitud.

Content Provider (Proveedor de contenidos):

Un proveedor de contenidos, como el nombre indica, provee un conjunto de datos a disposición de otras aplicaciones. Los datos pueden ser almacenados en el sistema de archivos, en una base de datos SQLite, o en cualquier otra forma de almacenamiento. El proveedor de contenidos extiende *ContentProvider*, la clase base para implementar un conjunto de métodos que permitan a otras aplicaciones recuperar y almacenar datos. Sin embargo, las aplicaciones no llaman a estos métodos directamente. En su lugar utiliza un objeto *ContentResolver* y sus métodos. Un *ContentResolver* puede hablar con cualquier proveedor de contenidos, y colabora con el proveedor para la gestión de toda la comunicación.

1.4.3 *Intents*

Si las actividades son básicamente pantallas, las "intenciones" o *Intents* son la manera de invocar estas *Activitys*. Es una clase que permite especificar una *Activity* a ejecutar, llamando a uno de los métodos de la clase *Activity* con ese *Intent* de parámetro. Son mensajes del sistema lanzados en el interior del dispositivo.

Por ejemplo, en la aplicación se han usado para pasar de una actividad a otra y pasar una serie de parámetros al mismo tiempo:

```
Intent intent = new Intent(Class1.this, Class2.class);
intent.putExtra("envi", valen); // meter var valen en "envi"
```

Luego para recuperar el valor en la actividad destino:

```
Bundle recibido = getIntent().getExtras();
String temp=recibido.getString("envi");
```

Existe otra clase denominada *IntentFilter* que está muy relaciona con *Intent*, mientras que *Intent* es una petición para hacer alguna acción, un *IntentFilter* es una descripción de lo que los I*ntent* de una actividad es capaz de soportar. Las actividades publican sus *IntentFilter* en el fichero AndroidManifest.xml

1.4.4 Archivo AndroidManifest.xml

El archivo AndroidManifest.xml está presente en todas las aplicaciones Android, contiene los componentes de la aplicación, así como la configuración global de la misma. Como ejemplo se muestra el archivo AndroidManifest.xml de la aplicación desarrollada:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3        android:versionCode="3"
4        android:versionName="3.0" package="javier.com">
5      <uses-sdk android:minSdkVersion="4" />
6  <uses-permission
7          android:name="android.permission.INTERNET" />
8          <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
9      <application android:icon="@drawable/icon" android:label="@string/ClienteUCH">
10         <activity android:name=".Principal"
11                   android:label="@string/app_name">
12             <intent-filter>
13                 <action android:name="android.intent.action.MAIN" />
14                 <category android:name="android.intent.category.LAUNCHER" />
15             </intent-filter>
16         </activity>
17 <activity android:name="menuUisDetectadas" android:label="@string/ListaDispositivos"></activity>
18 <activity android:name="UiSeleccionada" android:label="@string/UiSeleccionada"></activity>
19 <activity android:name="ListaAcciones" android:label="@string/ListaAcciones"></activity>
20 <activity android:name="RealizaAccion" android:label="@string/RealizaAccion"></activity>
21 <activity android:name="ConfigurarUCH" android:label="@string/ConfigurarUCH"></activity>
22 <activity android:name="Idioms" android:configChanges="locale" android:label="@string/Idioma">
23 </activity>
24 <activity android:name="Manual" android:label="@string/Manual"></activity>
25 <activity android:name="com.google.ads.AdActivity"
26               android:configChanges="keyboard|keyboardHidden|orientation"/>
27     </application>
28 </manifest>
```

Ilustración 48 Captura del Manifest.xml

Dentro de este archivo hay un elemento <application>, dentro del cual habrá uno o varios elementos <activity>. Cada uno de estos elementos supone una interacción con el usuario (generalmente una ventana), y se corresponde con una clase que hereda de la clase *Activity*. Cuando una actividad necesita efectuar una acción se crea un objeto de tipo *Intent*.

La mayoría de implementaciones del fichero AndroidManifest.xml incluye la declaración del *namespace* que se realiza según la siguiente sintaxis:

xmlns:android=http://schemas.android.com/apk/res/Android

La raiz <manifest> describe el contenido completo de los paquetes bajo el cual se engloba el resto de la aplicación y también indica la versión y nombre del código. Dentro tendremos estructuras, como por ejemplo <user-permission> o <permission>, encargadas de los permisos tales como acceso a internet, o acceso a la vibración del terminal. También cuenta con otras estructuras como <instrumentation> que se encarga

de declarar el código de un componente de instrumentación.

<intent-filter> declara un conjunto de valores de *Intent* que un componente dado puede soportar, los atributos pueden ser colocados bajo este elemento y pueden referenciar a etiquetas, iconos o cualquier otra información que la acción vaya a describir:

<action> Representa un intento de acción que el componente soporta.

<category> Representa un intento de categoría que el componente soporta.

<data> Son los intentos denominados *data MIME type, data URI scheme, data URI authority o data URI path* que los componentes soportan.

La estructura <meta-data> sirve para adicción de nuevos meta datos a la actividad.

Como ultimas etiquetas de importancia tenemos a <receiver>, <service> y <provider>, las cuales tienen las función de permitir a una aplicación ser llamada para cambiar los datos o acciones que ocurren, poder ejecutar un componente en segundo plano y controlar la persistencia de datos y sus publicaciones.

Los diferentes elementos que pueden aparecer en este archivo son los siguientes:

<action>	<activity>
<activity-alias>	<application>
<category>	<data>
<grant-uri-permission>	<instrumentation>
<intent-filter>	<manifest>
<meta-data>	<permission>
<permission-group>	<permission-tree>
<provider>	<receiver>
<service>	<uses-library>
<uses-permission>	<uses-sdk>

1.4.5 Ciclo de vida de una actividad

Una actividad puede estar ejecutándose por primera vez (*onCreate()*), o se retoma su ejecución (*onRestart()*), en pausa (*onPause()*) o detenida (*onStop()*). En todos estos casos la clase mantiene su información.

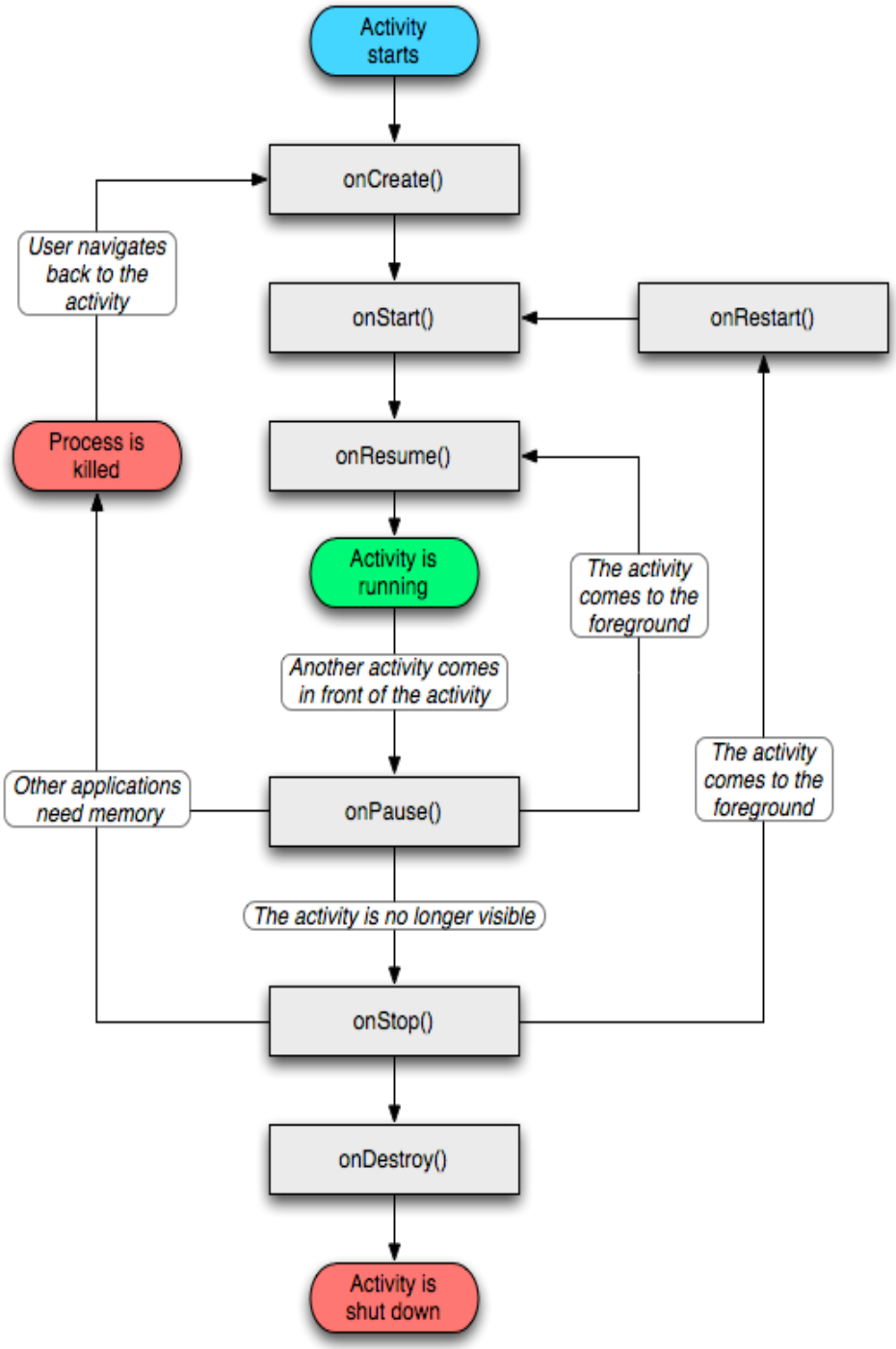


Ilustración 49 http://developer.android.com/images/activity_lifecycle.png

En el anterior gráfico se ven los estados por los que puede pasar una actividad (los óvalos oscuros) y los eventos que se disparan en dichos estados (los rectángulos más claros):

- ***onCreate()*** se invoca al crear la actividad. Aquí se realiza la configuración. Este evento sólo se invoca la primera vez que se llama a una actividad, o bien cuando se llama después de que el sistema haya tenido que eliminarla por falta de recursos, en este caso, volver al estado almacenado previamente mediante *Bundle*.

- ***onStart()*** es invocado cada vez que la actividad es visible al usuario. Es decir, la primera vez que se muestra, y las veces que en las que vuelve a aparecer tras haber estado oculta.

- ***onFreeze()*** y ***onPause()*** se invocan cuando detenemos la actividad o al reclamar CPU u otros recursos. Tras *onPause()*, la actividad permanece en un estado de espera en el que puede ocurrir que la aplicación sea destruida, por lo que estos eventos se usan para consolidar la información que no queremos que se pierda. Si la actividad no se destruye volverá al primer plano con el evento ***onResume()***.

Una actividad que esté pausada o detenida (tras *onPause()* u *onStop()*) puede ser destruida por el sistema sin previo aviso, por lo que deberemos encargarnos de guardar antes la información necesaria para poder comenzar con el mismo estado con el que finalizó.

- ***onDestroy()*** se invoca al eliminar una actividad de la memoria del sistema. Se produce al lanzar ***onFinish()*** o cuando el sistema necesita detener la actividad para liberar recursos.

1.5 Diseño de aplicaciones

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 en la pantalla. Para conseguir que aparezca algo en la pantalla es necesario diseñar el UI, con *Views* y *ViewGroups*, que son las clases que se usan para crear la interfaz entre el usuario y la plataforma Android.

Views: o también llamado vista, es la clase base de todos los *Widgets* (subclases ya implementadas que dibujan los elementos en la pantalla), cuyas propiedades contienen los datos de la capa y la información específica del área rectangular de la pantalla, pudiendo gestionar el tamaño, el cambio de foco y los gestos del área que representan. Una *View* tiene: *layout, drawing, focus change, scrolling*, etc.

La lista de widgets usables incluye *Text, EditText, InputMethod, MovementMethod, Button, RadioButton, CheckBox, y ScrollView.*

ViewGroups: Es la clase base de los *layouts* (subclases implementadas que proveen los tipos más comunes de los *layouts* de pantalla) y vistas contenedoras. Un *ViewGroup* es un objeto de la clase android.view.Viewgroup, cuya función es organizar los *views* y *ViewGroups* de nuestra interfaz. Los *ViewGroups* permiten añadir estructuras a la interfaz formando un árbol cuya raíz es un *ViewGroup* y contiene más *ViewGroup* y/o más Views.

Árbol estructurado de la interfaz UI: Cada *Activity* del UI usa el árbol comentado en *ViewGroup*, como apreciamos en la imagen de abajo. El árbol puede ser tan simple o complejo como se necesite, y se puede desarrollar usando los *Widgets* y *layouts* que Android proporciona o creando tus propias *Views*.

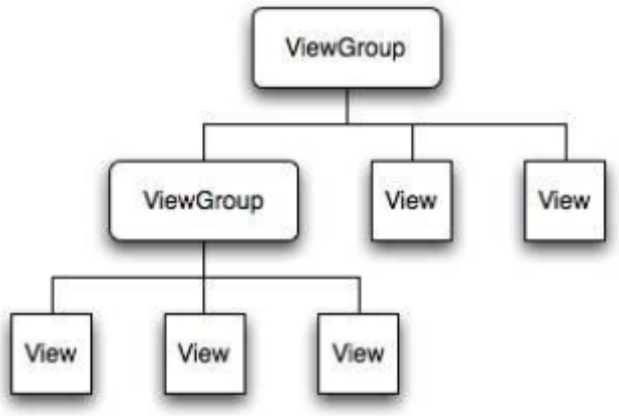


Ilustración 50 Representación de *ViewGroup*[13]

Para añadir el árbol a la pantalla, *Activity* deberá llamar al método setContentView(). Cuando tu *Activity* está activo y recibe el foco el sistema notifica y pide al nodo

[13] Imagen procedente de http://developer.android.com/images/viewgroup.png

principal medidas y dibuja el árbol. El nodo principal entonces pide que sus nodos hijos se dibujen a sí mismos, a partir de ese momento cada nodo *ViewGroup* del árbol es responsable de pintar sus hijos directos.

Cada *ViewGroup* se encarga de su dibujado, preparando a sus hijos y llamando a Draw() por cada hijo que se dibuje a sí mismo.

LayoutParams: Un hijo especifica su posición y su tamaño. Todos los *ViewGroup* usan como clase anidada una extensión de ViewGroup.LayoutParams. Esta subclase contiene los tipos de propiedades que definen la posición y el tamaño de un hijo, en propiedades para la clase de grupo de clases.

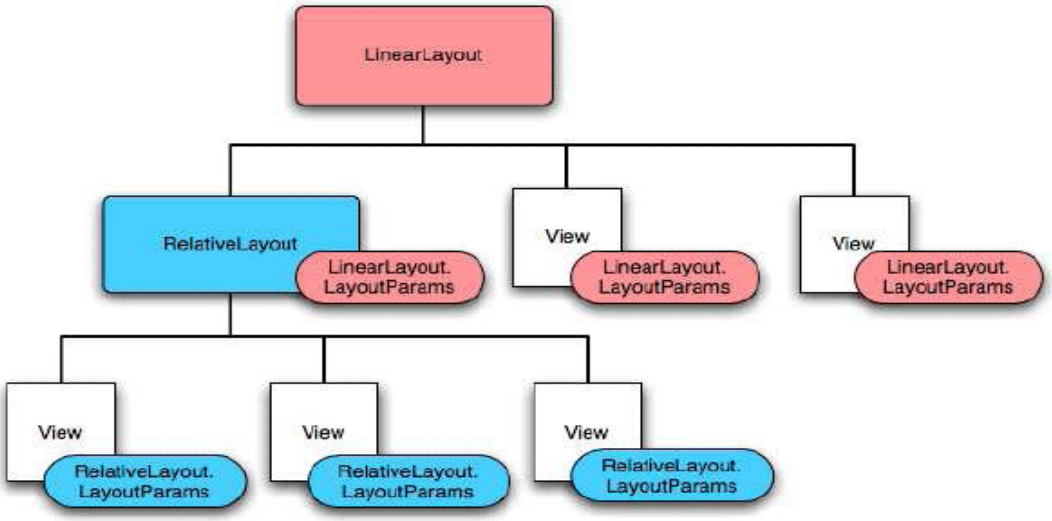


Ilustración 51 Representación de layout[14]

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 *ViewGroups* incluyen anchura y altura. Muchos también incluyen márgenes y bordes. Puedes especificar exactamente la altura y la anchura. Frecuentemente habrá que indicar a la *View* que tenga las dimensiones del tamaño de su contenedor, o que llegue a ser tan grande como el contenedor le permita.

[14] Imagen procedente de http://developer.android.com/images/layoutparams.png

1.6 Posibilidades gráficas

Para el desarrollo de juegos y aplicaciones con alto contenido gráfico, Android cuenta con OpenGL ES 1.0 de alto rendimiento para gráficos 3D. El API OpenGL está disponible en el paquete khronos de OpenGL ES, además de algunos servicios públicos Android OpenGL.

Al iniciar un proyecto, es importante tener en cuenta exactamente lo que se necesita a nivel gráfico. Dependiendo del objetivo se utilizaran diferentes técnicas en su proceso de elaboración. Por ejemplo, los gráficos y animaciones para una aplicación estática, son muy distintos a los gráficos y animaciones para un juego interactivo o 3D.

Este capítulo no se desarrolla más porque no se ha utilizado en el proyecto.

Para hacer un simple diseño de la interfaz gráfica puede escribirse en XML, sin utilizar el API de OpenGL:

```
<shape xmlns:android=”http://schemas.android.com/apk/res/android”>
<solid android:color=”#FF0000FF”/>
</shape>
```

Anexo 2 Instalación del entorno de desarrollo UCH,Eclipse,pluginAndroid

Para conseguir el entorno se necesita un IDE de desarrollo para implementar las conexiones entre el terminal y UCH, para ello se ha elegido Eclipse. Por otra parte, se necesita un servidor donde lanzar UCH, esto se consigue con Tomcat, alojándolo a su vez en Eclipse.

Para terminar de montar el entorno, necesitamos instalar el SDK de Android para poder desarrollar aplicaciones móviles y poder emular en un terminal virtual personalizable.

2.1 Instalar Eclipse y Tomcat

En la web http://www.eclipse.org/downloads/ ir a "Windows 32 Bit" de *Eclipse IDE for Java EE Developers* (recuadro rojo de la siguiente ilustración)

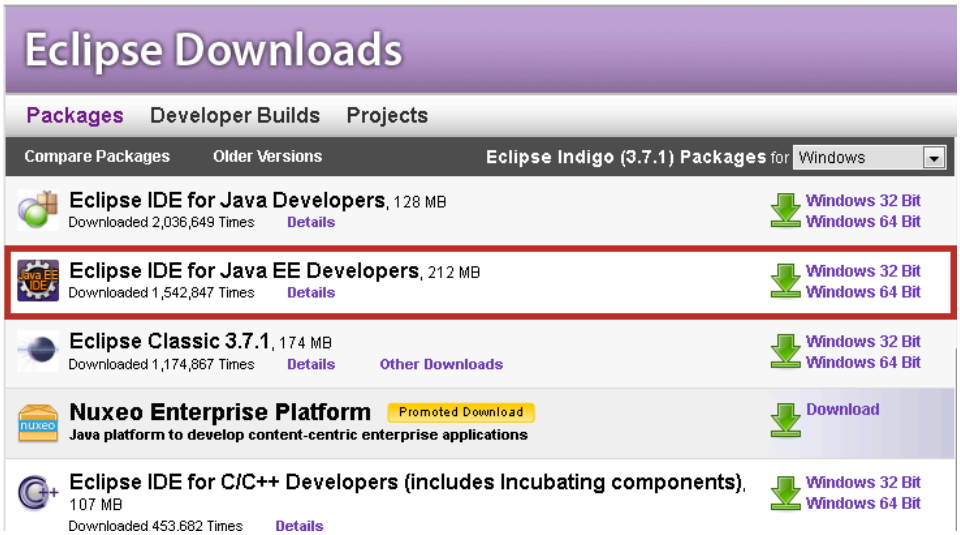


Ilustración 52 Captura de la web de www.eclipse.org

Tras ello, descargar haciendo clic en la flecha verde:

Ilustración 53 Captura de la zona de descargas de la web de Eclipse

Tras la descarga, descomprimir el fichero alojándolo en el disco duro.

Instalación Tomcat:

Descargar la versión 6.0 de Tomcat, pinchando en "zip" de *core* de la distribución binaria. http://tomcat.apache.org/download-60.cgi :

Ilustración 54 Captura de pantalla de http://tomcat.apache.org/download-60.cgi

Para lanzar Tomcat, arrancamos Eclipse y vamos a *Window -> Preferences -> Server -> Runtime Environment*, pulsamos *Add* y seleccionamos nuestra versión de Tomcat. Marcamos *Create a new local server* si no está seleccionado.

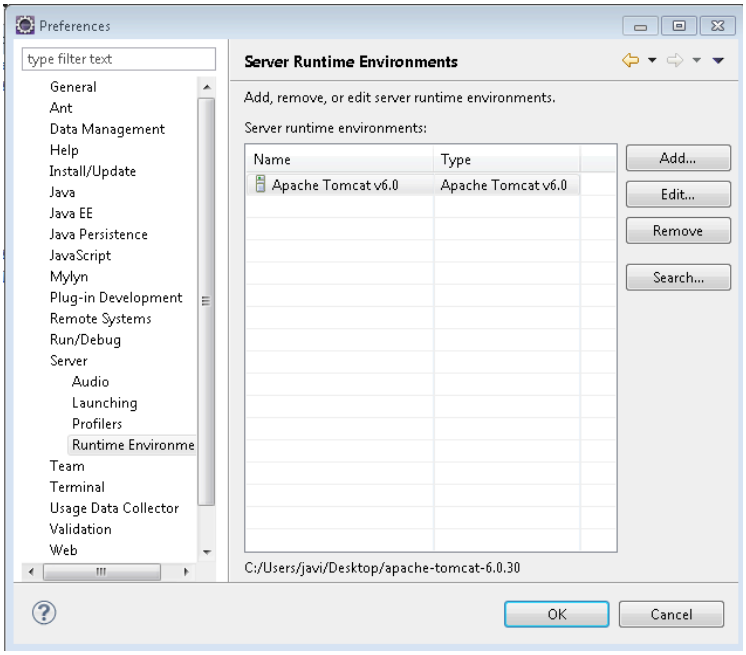


Ilustración 55 Captura de pantalla de las preferencias de Eclipse

Pulsamos *Next* y buscamos el directorio en el que instalamos Tomcat. Pulsamos *Finish* y *OK*.

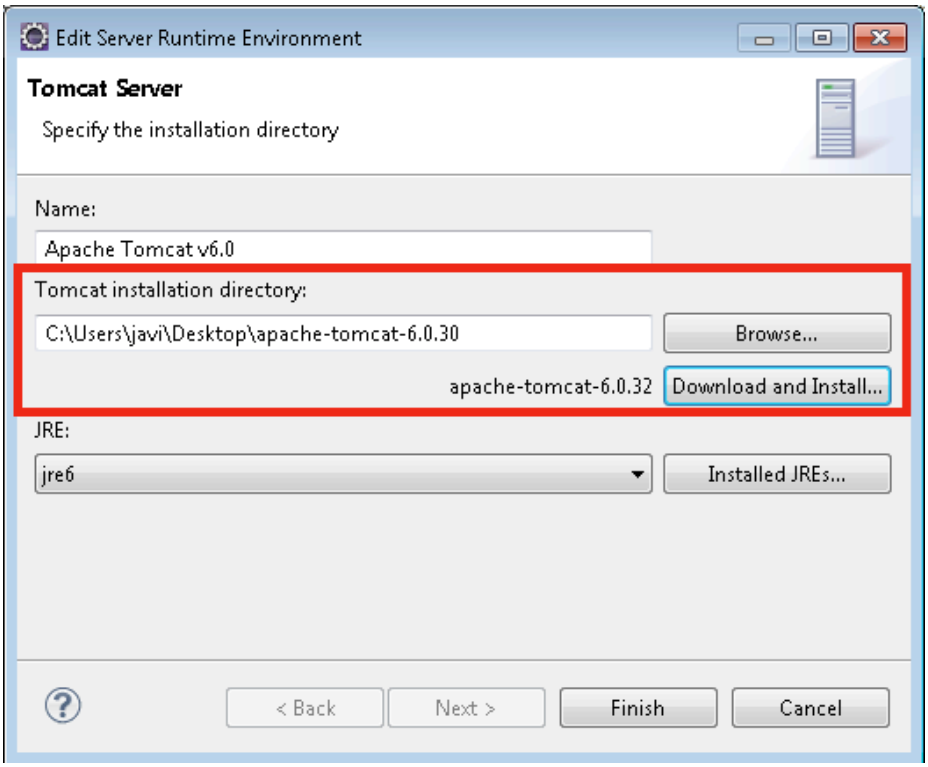


Ilustración 56 Captura de pantalla de la configuración de Tomcat en eclipse

Como se puede apreciar en la ilustración anterior, podemos evitar el paso de descargar Tomcat desde la web pinchando en "*Download and Install…*"

El nuevo servidor debería mostrarse en la pestaña *Servers*. También se habrá creado un proyecto nuevo llamado "Servers" con los archivos de configuración de nuestra instancia de Tomcat.

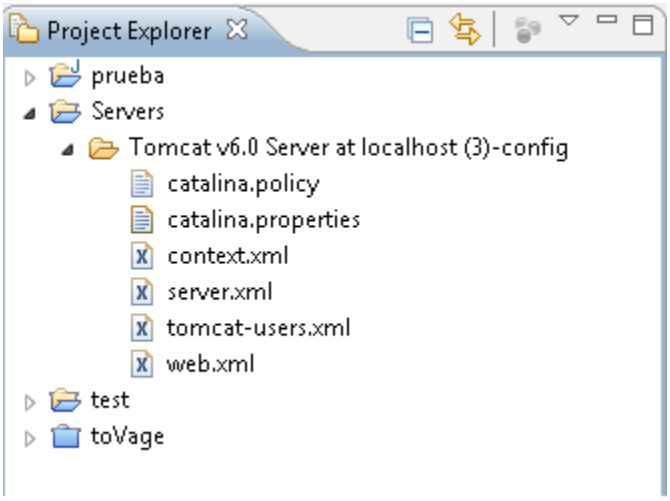


Ilustración 57 Captura de pantalla del proyecto del servidor Tomcat en Eclipse

2.2 Instalar UCH en Tomcat

Lo primero, descargar el UCHj, que es la implementación Java del Centro de Control Universal (UCH). Descargar de:

http://sourceforge.net/projects/traceurcsdk/files/UCHj/2-0a/

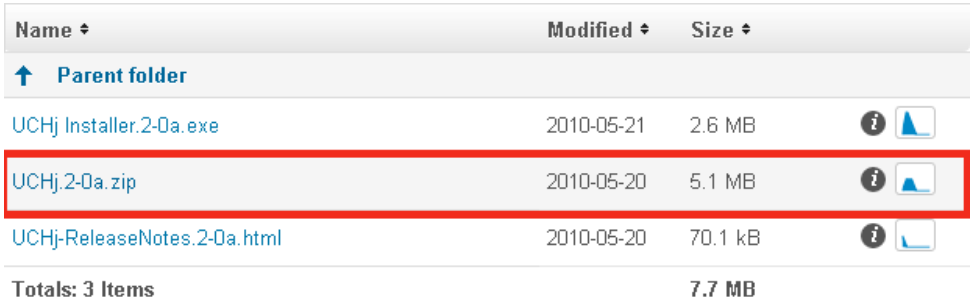


Ilustración 58 Captura de pantalla de los ficheros disponibles para descargar

Solo interesa el fichero UCH.war alojado en la carpeta /bin del fichero descargado. Lo extraemos y vamos a Eclipse. Vamos a File/import y seleccionar importar ficheros war.

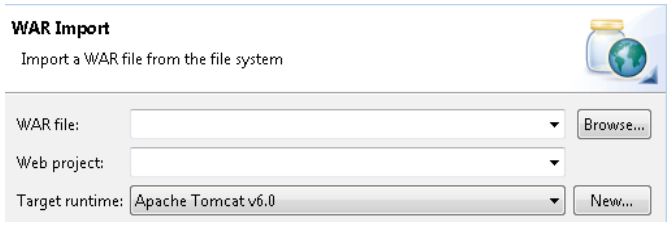


Ilustración 59 Captura de pantalla de la importación de un war en eclipse (paso 1)

Accedemos a *Browse* y seleccionar el fichero war de donde esté alojado.

Pulsar en *next*.

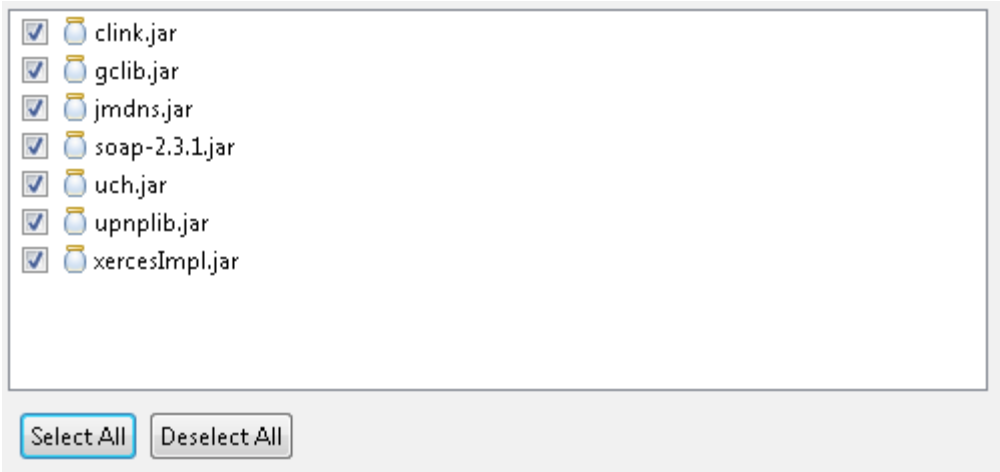


Ilustración 60 Captura de pantalla de la importación de un war en eclipse (paso 2)

Seleccionar todas las librerías y pulsar *Finish*. Ahora ya podemos lanzar UCH lanzándolo como (Run as) servidor (Apache server).

Hay otra forma más sencilla de lanzar UCH en Tomcat sin la necesidad de meter Tomcat en Eclipse, tras descargar Tomcat, alojar el fichero war dentro de la carpeta *webapps* de la carpeta de Tomcat descargada. Tras reiniciar Tomcat se despliega UCH.war:

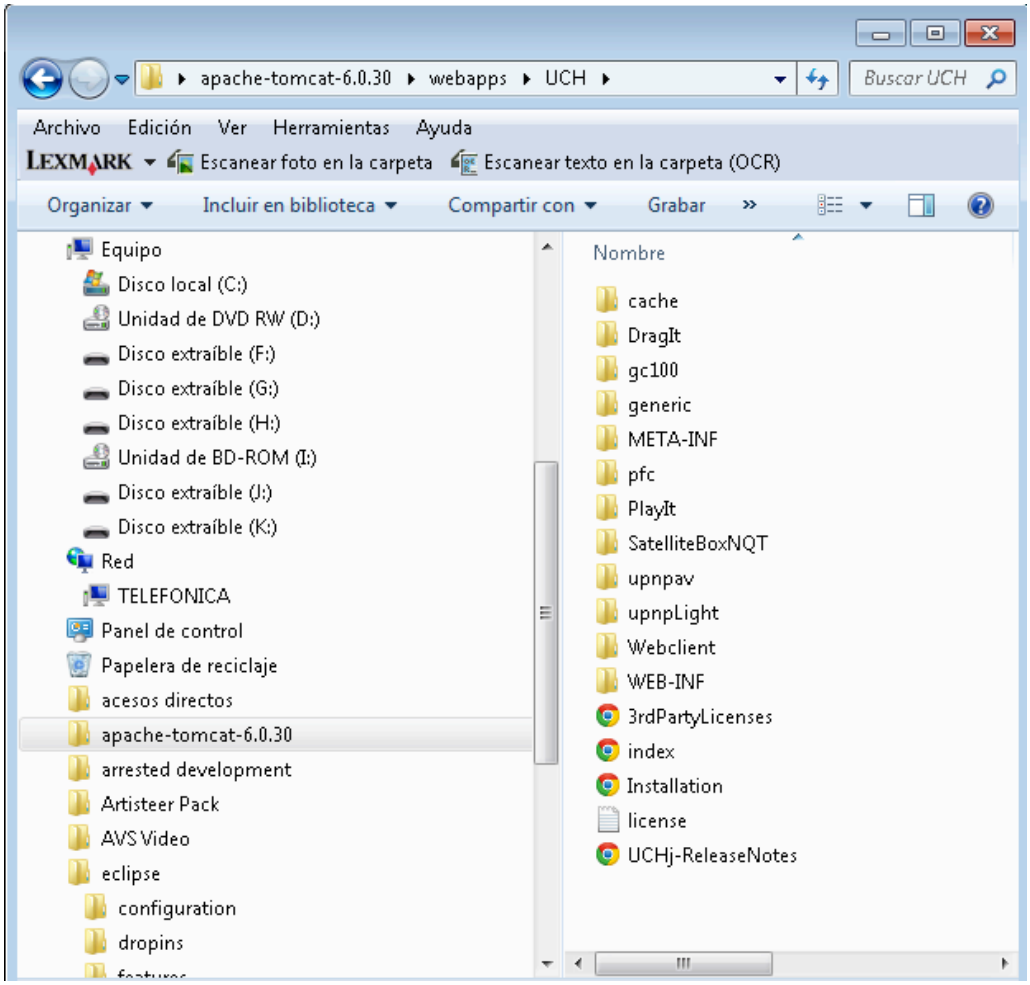


Ilustración 61 Captura de pantalla de los ficheros desplegados de UCH al lanzarse UCH.war en tomcat

Tanto la primera vez, como cada vez que se cambie la IP de nuestro PC, se debe indicar nuestra IP a UCH, esto se consigue editando el fichero apache-tomcat-6.0.30/webapps/UCH/WEB-INF/web.xml, y sustituyendo 82.130.206.133:8080 por vuestra IP.

Para lanzarlo, basta con abrir el fichero bootstrap de la carpeta /bin de Tomcat.

Una vez se tenga UCH dentro de Tomcat, se debe de configurar Tomcat para que funcione el servidor UCH. Se necesita cambiar dos ficheros alojados en la carpeta conf de Tomcat:

Fichero server.xml: Cambiar el Original por lo nuevo

Original:

```
<Connector port="8080" maxThreads="150" minSpareThreads="25"
maxSpareThreads="75"enableLookups="false" redirectPort="8443"
acceptCount="100"connectionTimeout="20000"
disableUploadTimeout="true" />
```

Nuevo:

```
<Connector port="80" maxThreads="150" minSpareThreads="25"
maxSpareThreads="75" enableLookups="false" redirectPort="8443"
acceptCount="100" connectionTimeout="20000"
disableUploadTimeout="true" />
```

Fichero web.xml: Se debe añadir las extensiones de los ficheros de UCH, para que los entienda Tomcat. Cada mime-mapping corresponde con una extensión de fichero diferente usado por UCH.

```
  <mime-mapping>
              <extension>td</extension>
              <mime-type>application/urc-targetdesc+xml</mime-type>
        </mime-mapping>
        <mime-mapping>
              <extension>uis</extension>
              <mime-type>application/urc-uisocketdesc+xml</mime-
type>
        </mime-mapping>
        <mime-mapping>
              <extension>pret</extension>
              <mime-type>application/urc-pret+xml</mime-type>
        </mime-mapping>
        <mime-mapping>
              <extension>rsheet</extension>
              <mime-type>application/urc-ressheet+xml</mime-type>
        </mime-mapping>
        <mime-mapping>
              <extension>rdir</extension>
              <mime-type>application/urc-resdir+xml</mime-type>
        </mime-mapping>
```

2.3 Instalar SDK de Android en Eclipse

Descargar el SDK http://developer.android.com/sdk/index.html

Download the Android SDK

Welcome Developers! If you are new to the Android SDK, please read the steps below, for an overview of how to set up the SDK.

If you're already using the Android SDK, you should update to the latest tools or platform using the *Android SDK and AVD Manager*, rather than downloading a new SDK starter package. See Adding SDK Components.

Platform	Package	Size	MD5 Checksum
Windows	android-sdk_r16-windows.zip	29562413 bytes	6b926d0c0a871f1a946e65259984701a
	installer_r16-windows.exe (Recommended)	29561554 bytes	3521dda4904886b05980590f83cf3469
Mac OS X (intel)	android-sdk_r16-macosx.zip	26158334 bytes	d1dc2b6f13eed5e3ce5cf26c4e4c47aa
Linux (i386)	android-sdk_r16-linux.tgz	22048174 bytes	3ba457f731d51da3741c29c8830a4583

Ilustración 62 Captura de pantalla de http://developer.android.com/sdk/index.html

Instalar el fichero descargado.

Tras ello, instalar el *plugin* de Eclipse: *ADT Plugin for Eclipse:*

En eclipse, en *Help* > *Install New Software*, acceder a "Add"

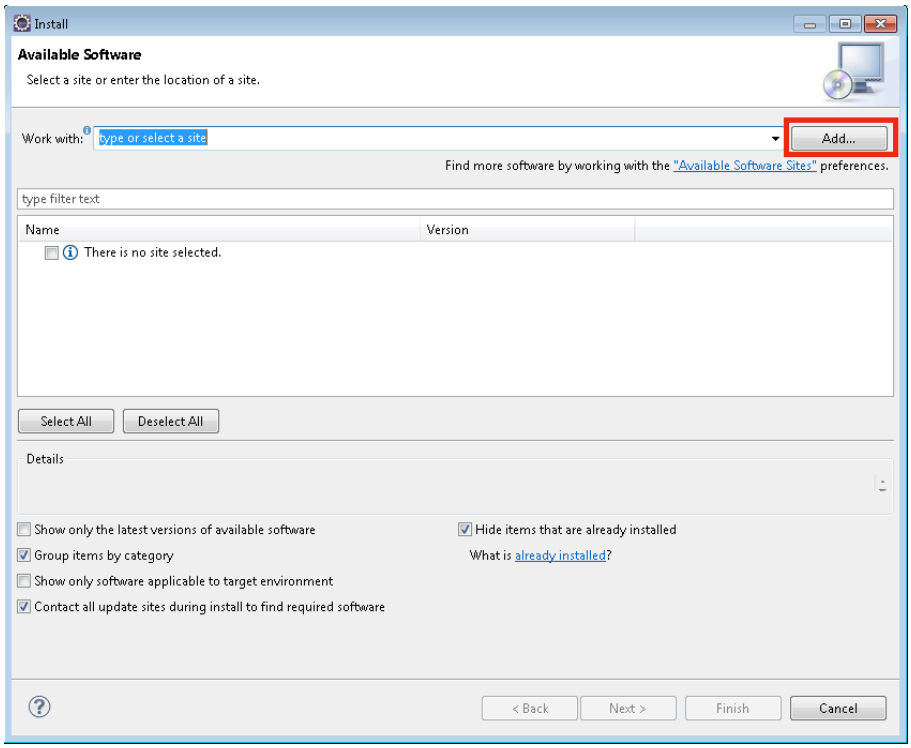


Ilustración 63 Captura de pantalla de la zona de instalación de software en Eclipse

Rellenar los campos, *Name: ADT Plugin y Location:*

https://dl-ssl.google.com/android/eclipse/

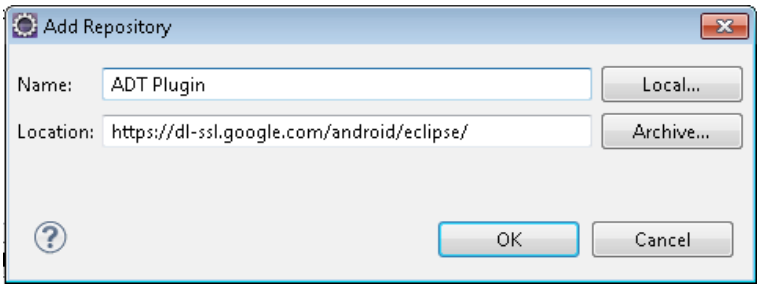


Ilustración 64 Captura de pantalla de los datos a rellenar para la instalación del *plugin* ADT

Pulsamos "OK"

Seleccionar lo que se desee instalar, seleccionar todo y continuar:

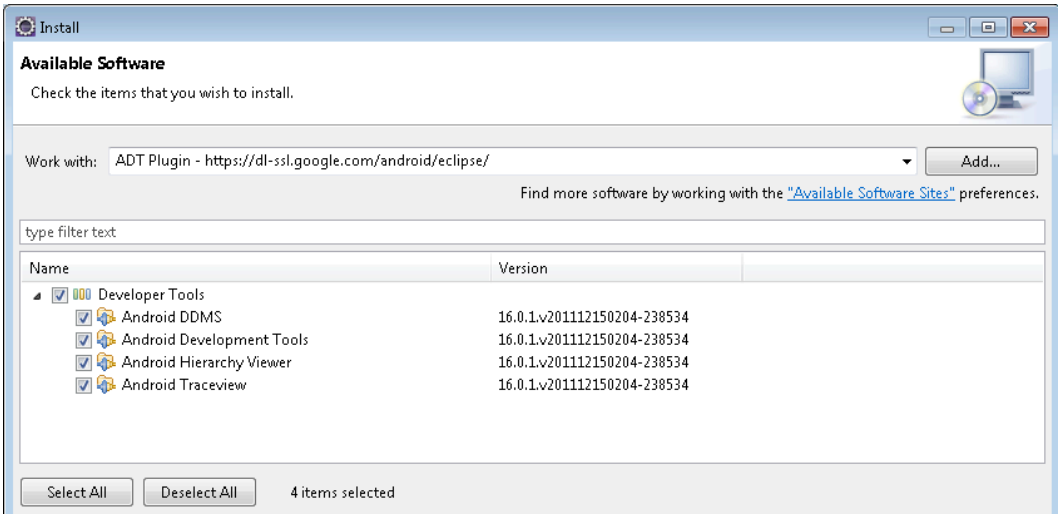


Ilustración 65 Captura de pantalla de los paquetes a instalar del *plugin* ADT

Volver a pulsar *Next*, aceptar los términos y la instalación del *plugin* comienza.

Al finalizar la instalación se debe reiniciar Eclipse.

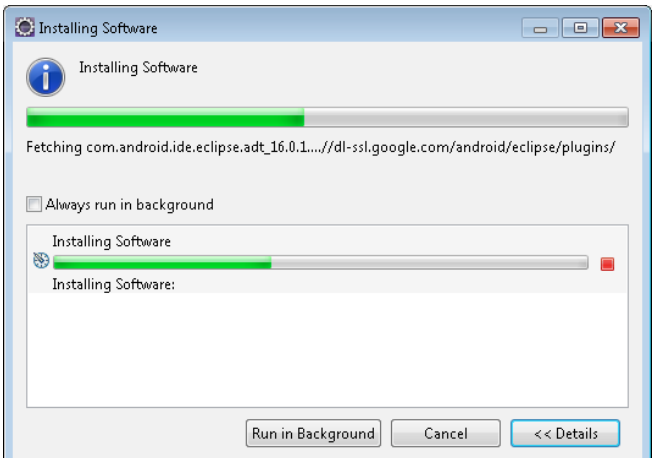


Ilustración 66 Captura de pantalla del proceso de instalación del plugin ADT

Configurando el *plugin* ADT:

Para usar el *plugin*, se debe configurar el *plugin* ADT indicándole donde se tiene instalado el SDK de Android. Para ello ir a *Window* > *Preferences* y hacemos clic en Android. Ahora buscamos donde tenemos la instalación del SDK en *Browse*.

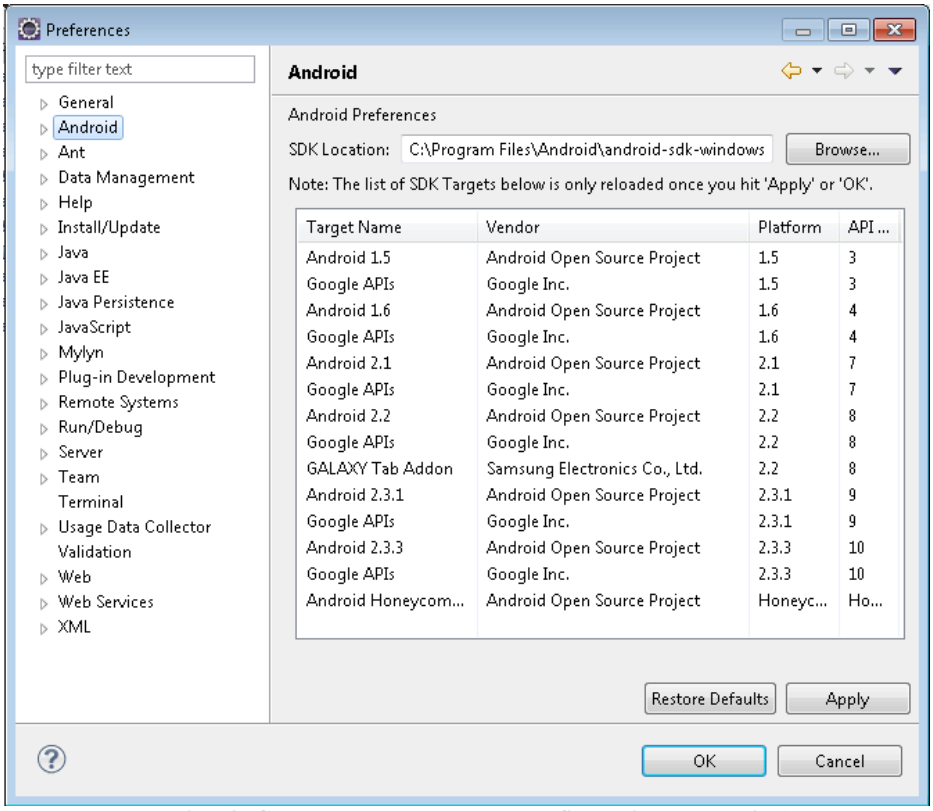


Ilustración 67 Captura de pantalla de la configuración del plugin ADT

Aplicamos y OK.

Ya se puede crear el proyecto Android.

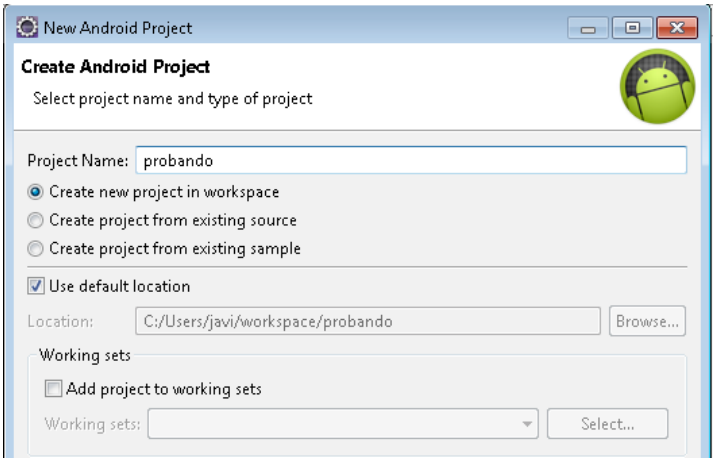


Ilustración 68 Captura de pantalla de la creación de un proyecto Android en Eclipse

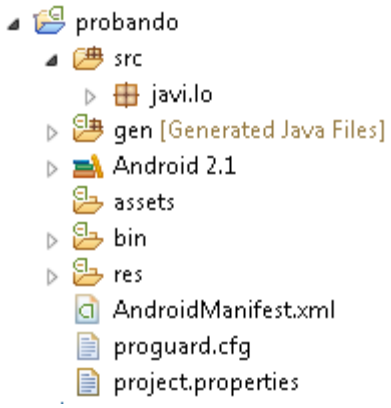


Ilustración 69 Captura de la estructura de un proyecto Android

Anexo 3 Instalación de la aplicación (junto con UCH)

Dependiendo de cómo se tenga configurado el teléfono (idioma) buscaremos la aplicación en el *Market* de Android. Si lo tenemos en castellano lo encontraremos con el nombre "Controlar sistemas UPnP", en Ingles lo encontraremos por "*UPnP control devices*". En ambos casos también se puede encontrar por "fjlogrono".

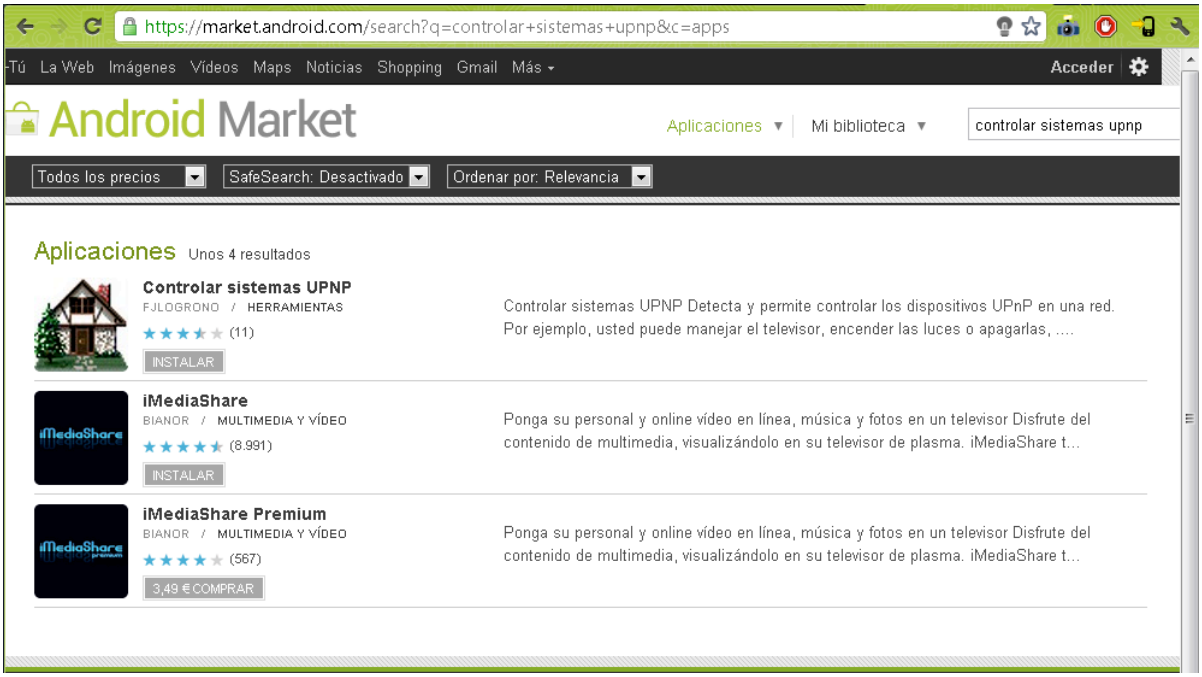


Ilustración 70 Captura de pantalla del market de Android desde PC

Una vez descargada e instalada la aplicación, dentro de ella, en la pantalla principal, vamos a manual. Dentro de la pantalla manual, se nos invita a introducir un E-mail, en el cual recibiremos los pasos para la instalación. Dependiendo del idioma de nuestro dispositivo aparecerá en un idioma u otro. Necesitamos el servidor UCH, por eso se ha metido en un fichero junto a Tomcat. El correo recibido es el siguiente:

Paso 1: Descargar el fichero del siguiente enlace y descomprimirlo.
Enlace: http://depositfiles.com/files/2ygjvc7vh

Paso 2: Editar el fichero apache-tomcat-6.0.30/webapps/UCH/WEB-INF/web.xml, sustituyendo 82.130.206.133 por la IP del PC (cmd-->ipconfig o ifconfig en linux)
Ejemplo: 192.168.1.35:8080 Si se desea utilizar un puerto diferente al 80 (defecto)

Paso 3:
Windows: Ejecutar el fichero /bin/bootstrap.jar
Linux: Ejecutar sh bin/startup.sh

Recordar que la IP que aparezca en el web.xml deberá ser la misma que aparezca en Establecer IP del servidor UCH del menú principal de la aplicación.
javierlogronoaguirre@msn.com

Anexo 4 Como publicar una aplicación en el Market y obtener beneficio

4.1 Como exportar la aplicación Android

Ir a Archivo-Exportar o hacer clic con segundo botón sobre el proyecto y pinchar en Exportar.

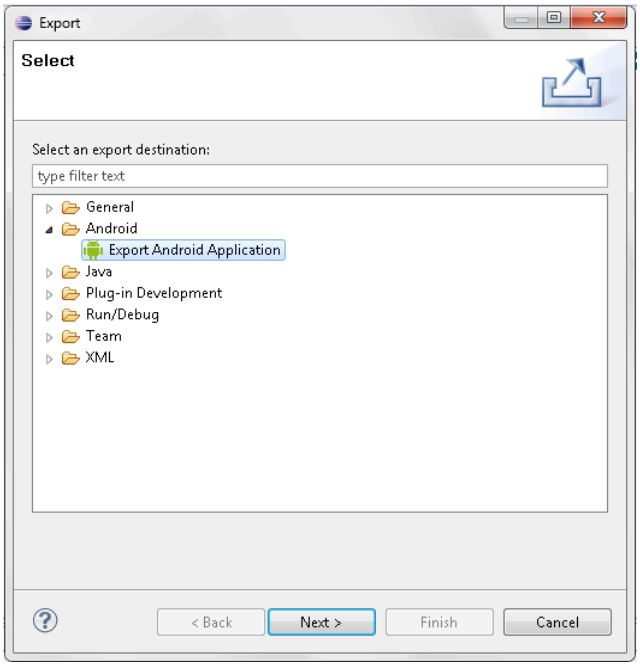


Ilustración 71 Proceso de exportar aplicación (paso 1)

Seleccionar *Export Android Application* de Android, clicar en *Next* y se va a la siguiente ilustración.

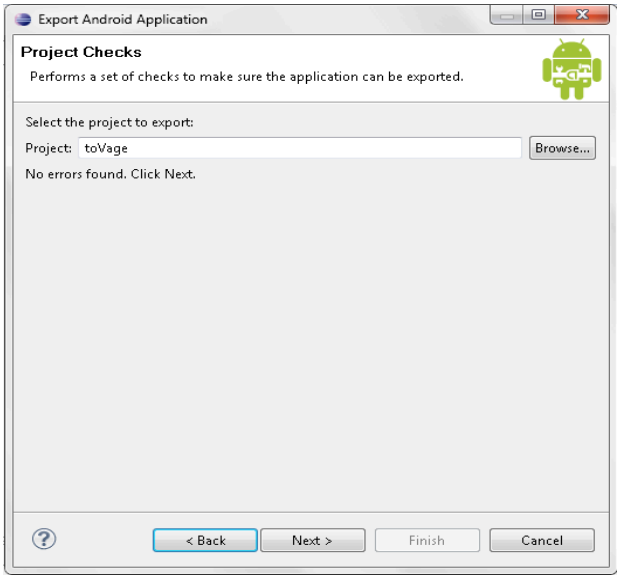


Ilustración 72 Proceso de exportar aplicación (paso 2)

Seleccionar el proyecto que deseamos exportar y clicar en *Next*.

Aparece lo siguiente:

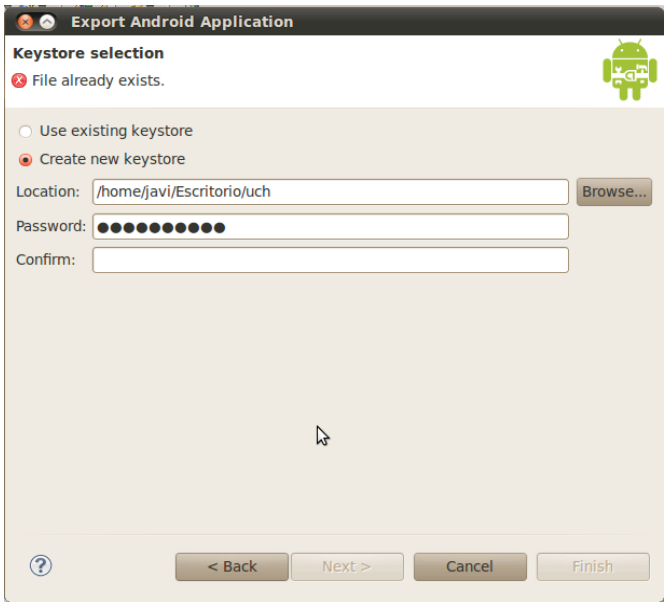
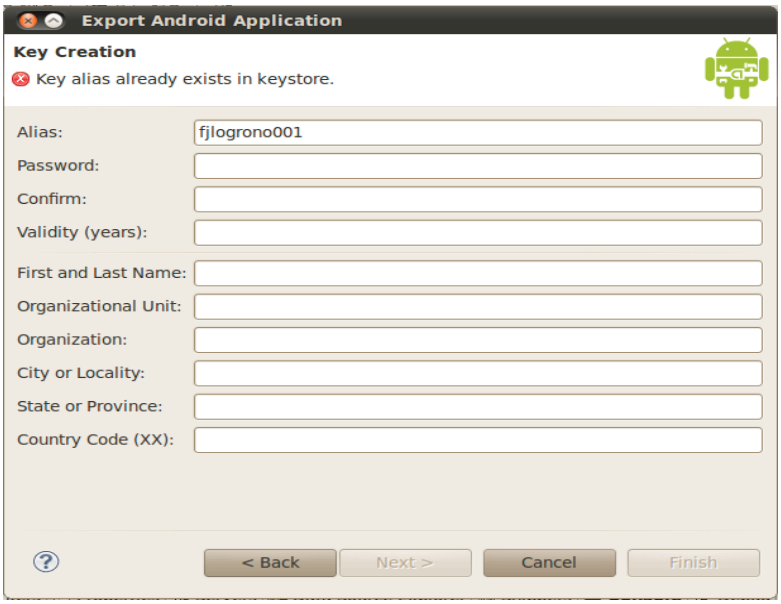


Ilustración 73 Proceso de exportar aplicación (paso 3)

El *keystore* es un fichero usado para gestionar las claves y sus propietarios, es obligatorio para poder exportar la aplicación.

Se debe indicar donde deseamos localizar el fichero e insertar una contraseña como seguridad si es la primera vez que realizamos este paso. Pulsamos en *Next* y vamos a la Ilustracion 76.

Las siguientes veces marcamos "*Use existing keystore*" y pulsamos *Next*. Iremos a Ilustración 77.



Rellenar los campos, no todos son obligatorios. Rellenar al menos los 4 primeros.

Pulsamos en *Next* y aparecemos en la ilustración 77.

Ilustración 74 Proceso de exportar aplicación (paso 4)

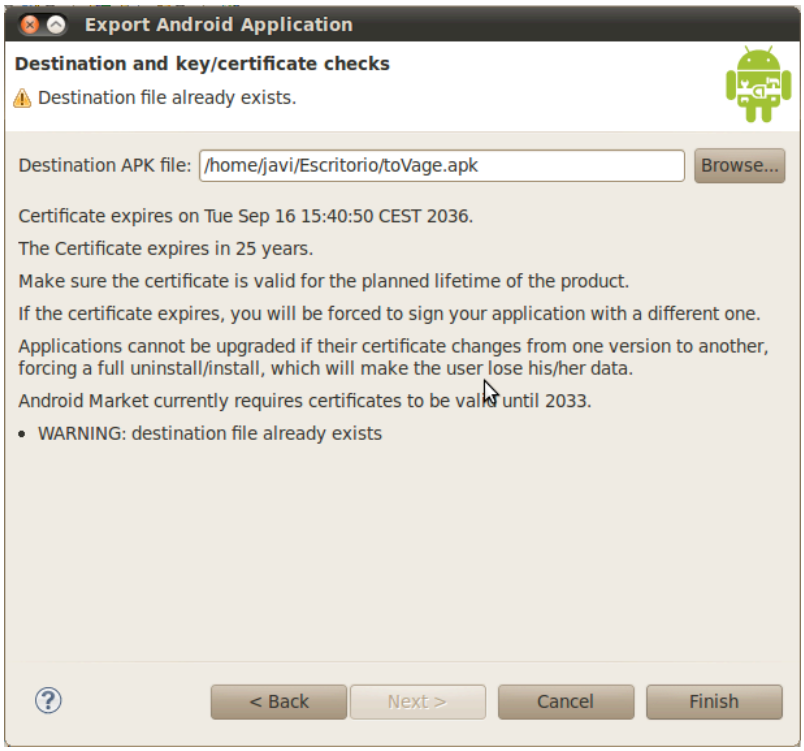


Ilustración 75 Proceso de exportar aplicación (paso 5)

Pulsar en *Finish* y ya tenemos nuestro fichero .apk en el directorio elegido.

4.2 Como publicar la aplicación en el *Market*

4.2.1 Registrarse como desarrollador Google

Para poder subir una aplicación al *Market* de Android se debe registrar previamente como desarrollador Google, para esto se debe tener una cuenta Google (Gmail en mi caso) y pagar 25$ de por vida (unos 18€) mediante Google Checkout[15] , en contrapartida de los desarrolladores de Apple que tienen que pagar 100$ anuales. No solo la diferencia con Apple radica en el precio, también el proceso de aprobación de una aplicación es mucho más costosa para iOS que para Android:

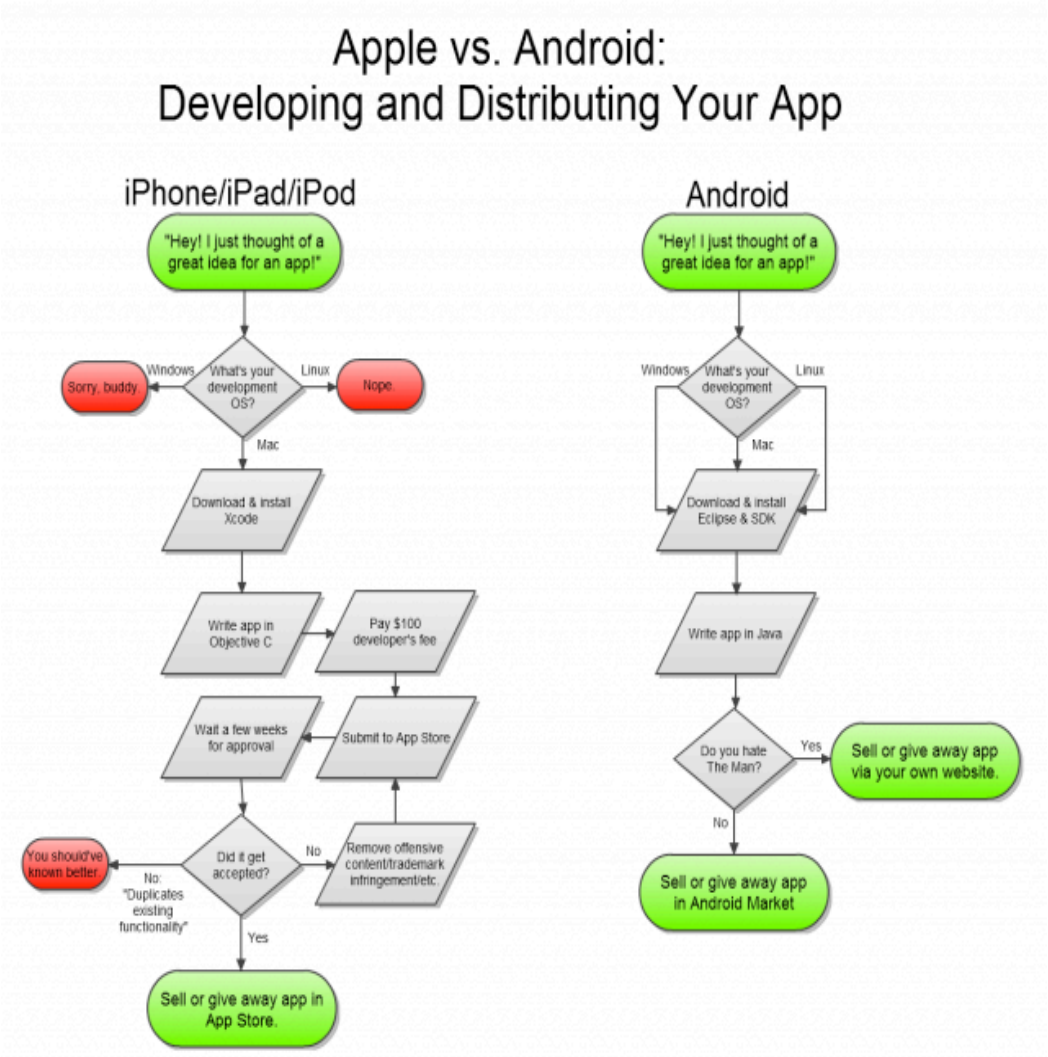


Ilustración 76 Imagen procedente de http://img.androidsis.com/wp-content/uploads/proceso-desarrollo-app.png

[15] Servicio de pago en línea de Google.

Ir a http://developer.android.com/index.html e ir a *Publish*. Aparecerá lo siguiente:

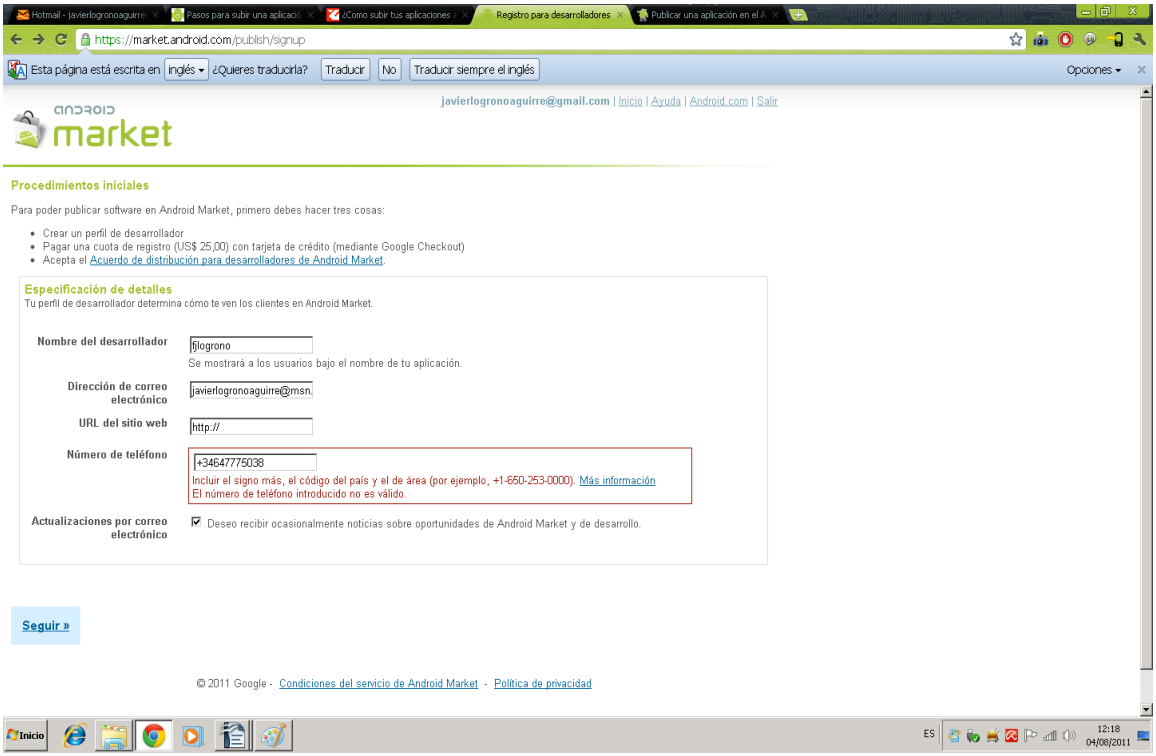


Ilustración 77 Captura del proceso de registro como desarrollador Android (paso 1)

Se debe introducir un nombre de desarrollador, el correo y un número de teléfono. Clicar en "Seguir". Aparecerá la siguiente ilustración, volvemos a clicar en seguir.

Ilustración 78 Captura del proceso de registro como desarrollador Android (paso 2)

Ahora asociar una tarjeta de crédito a la cuenta para poder realizar el pago mediante Google Checkout:

Ilustración 79 Captura del proceso de registro como desarrollador Android (paso 3.1)

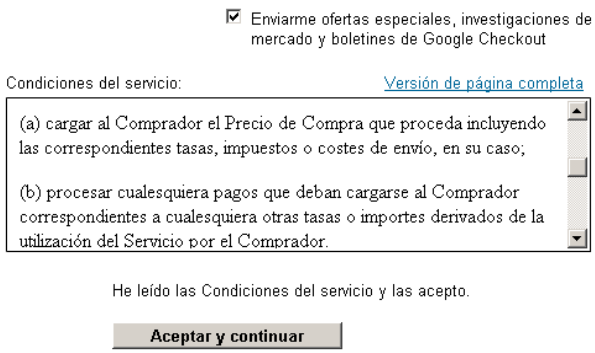


Ilustración 80 Captura del proceso de registro como desarrollador Android (paso 3.2)

Rellenar los campos, aceptar y continuar.

Tras ello se pasa a la pantalla de confirmación del pedido:

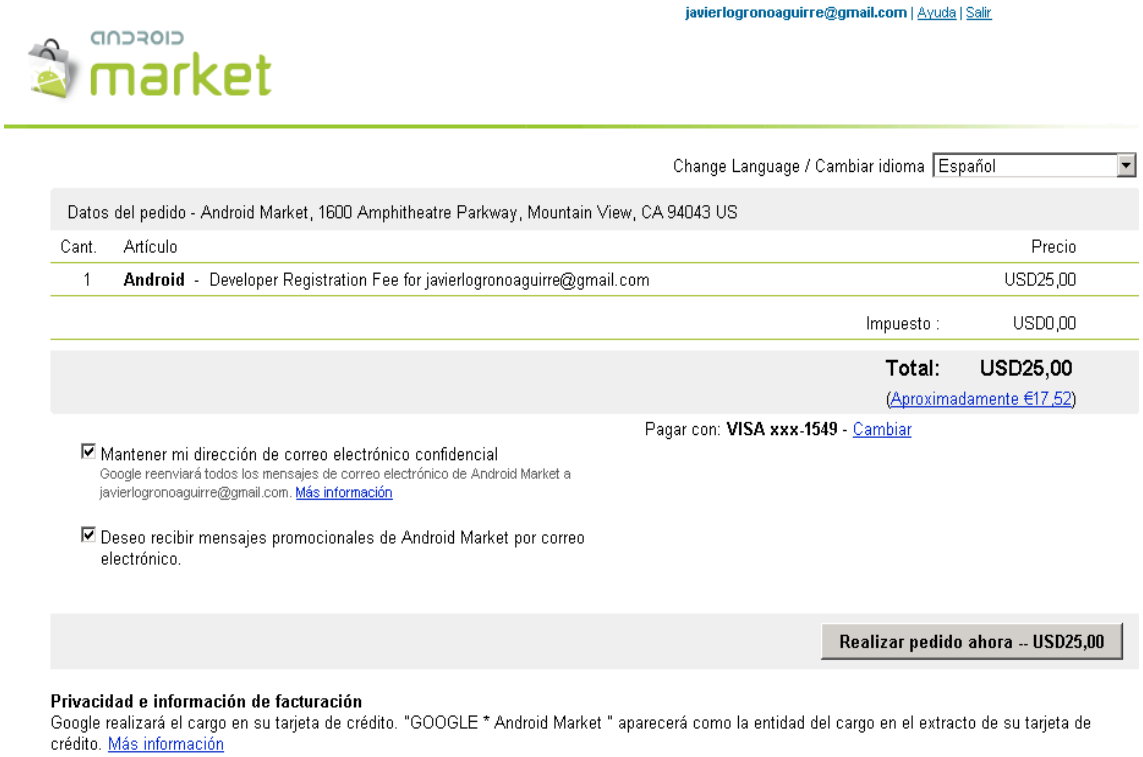


Ilustración 81 Captura del proceso de registro como desarrollador Android (paso 4)

Clicar en "Realizar pedido ahora". Ya se ha realizado el pedido, pero tarda un tiempo (máximo 48 horas) en que se confirme el proceso de alta como desarrollador:

Ilustración 82 Captura del proceso de registro como desarrollador Android (paso 5)

Si se intenta acceder a la cuenta de desarrollador Android, al no estar concluido el proceso de alta, nos aparecerá lo siguiente:

Ilustración 83 Captura de la finalización del proceso de registro como desarrollador Android

Cuando nos hayan dado de alta, al entrar en nuestra cuenta de desarrollador, aparecerá la licencia de desarrollador para el mercado de Android:

Ilustración 84 Captura de la finalización del proceso de registro como desarrollador Android 2

Aceptar tras leerlo. Ya se está registrado como desarrolladores Android:

Ilustración 85 Finalización del registro como desarrollador

Subir aplicación o nueva versión de la aplicación:

El proceso de subir una aplicación al *Market* es un poco tedioso. Se debe meter capturas de pantallas de la aplicación desarrollada, un icono, titulo, descripción y cambios recientes en cada idioma para el cual deseemos que esté la aplicación. Lo primero que se debe hacer es subir el *apk*:

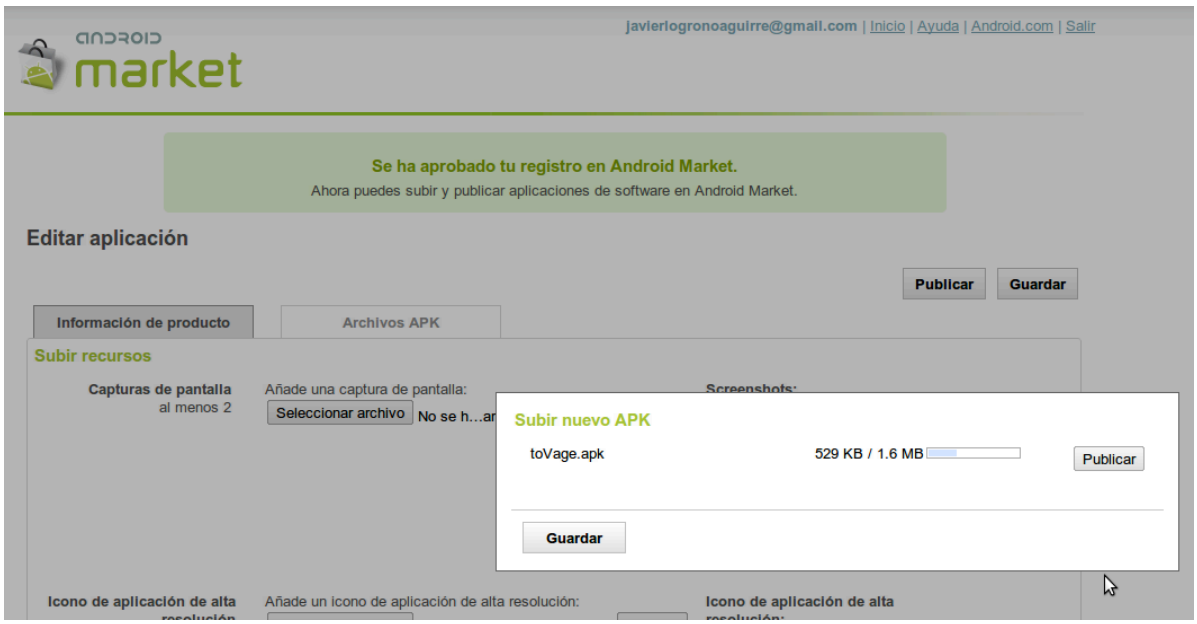


Ilustración 86 Captura del proceso de subir una aplicacion al market (paso 1)

Pedirá la confirmación del *apk* subido, mostrando detalles sobre permisos usados, versión del código, etc.

Ilustración 87 Captura del proceso de subir una aplicación al *Market* (paso 2)

Subir alguna captura, el icono y las descripciones de la aplicación para todos los idiomas deseados, guardamos.

Ilustración 88 Captura del proceso de subir una aplicación al *Market* (paso 3.1)

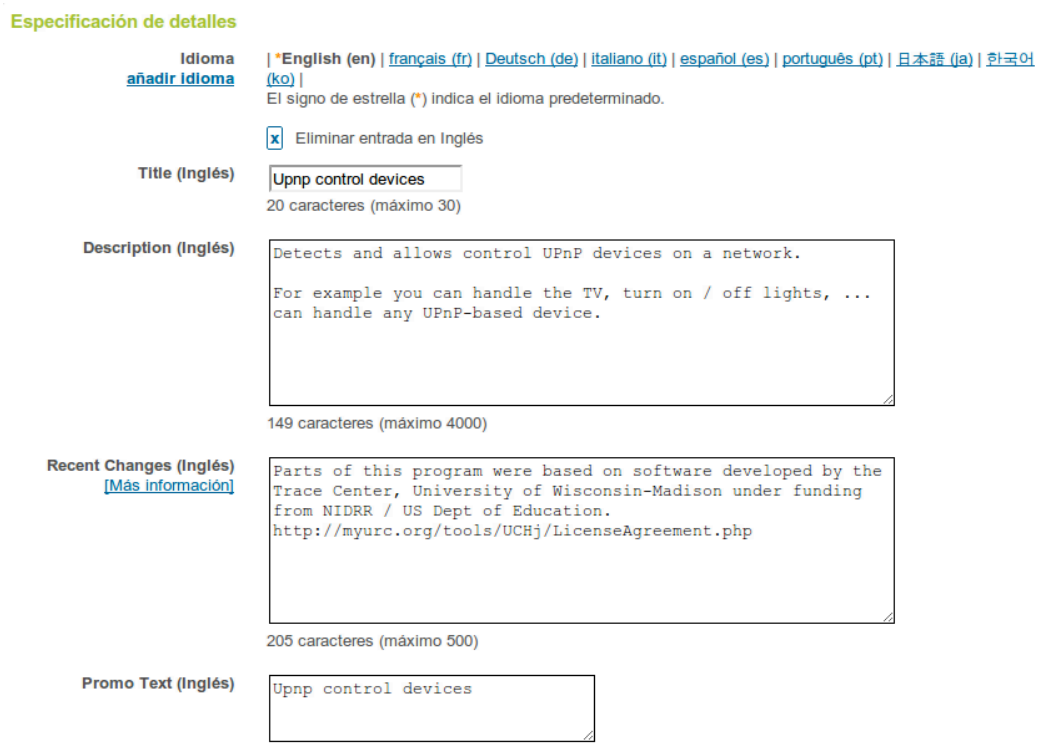


Ilustración 89 Captura del proceso de subir una aplicación al M*arket* (paso 3.2)

Tras rellenar los campos de la aplicación, en nuestra cuenta aparecerá la nueva aplicación subida, mostrando datos de las descargar totales y activas[16], comentarios de usuarios y errores:

Ilustración 90 Captura de detalles de aplicaciones subidas en al *Market* en mi cuenta como desarrollador Android.

Se pueden observar los errores que la aplicación le ha dado a otros usuarios y con gran detalle. Errores clasificados por fecha y tipos:

[16] Una descarga activa es una descarga que no se ha eliminado del dispositivo.

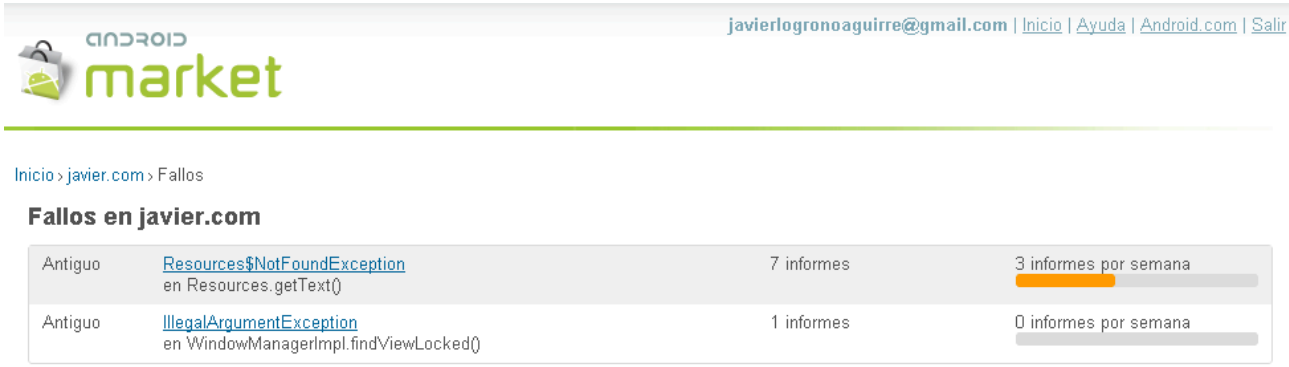


Ilustración 91 Captura de pantalla de los errores producidos en la aplicación desarrollada por los usuarios

Accediendo en cada tipo de fallo nos mostrará su seguimiento de pila con detalle. Esto es de gran ayuda para descubrir nuevos fallos en la aplicación que en las pruebas no aparecen, esto es debido a la multitud de dispositivos entre otras cosas.

Ejemplo del detalle mostrado en un error:

```
Ejemplo de android.content.res.Resources$NotFoundException en javier.com:
java.lang.RuntimeException: Unable to start activity
ComponentInfo{javier.com/javier.com.menuUisDetectadas}:
android.content.res.Resources$NotFoundException: String resource ID
#0x7f050026
at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:1659)
at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:1675)
at android.app.ActivityThread.access$1500(ActivityThread.java:121)
at android.app.ActivityThread$H.handleMessage(ActivityThread.java:943)
at android.os.Handler.dispatchMessage(Handler.java:99)
at android.os.Looper.loop(Looper.java:123)
at android.app.ActivityThread.main(ActivityThread.java:3701)
at java.lang.reflect.Method.invokeNative(Native Method)
at java.lang.reflect.Method.invoke(Method.java:507)
at
com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:862
)
at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:620)
at dalvik.system.NativeStart.main(Native Method)
Caused by: android.content.res.Resources$NotFoundException: String resource ID
#0x7f050026
```

Aparte de la información de los errores, se nos proporciona una serie de estadísticas sobre las descargas: tipos de dispositivos móviles, sistemas operativos, idiomas, países, etc. de los usuarios que descarguen la aplicación:

El día 3 de octubre de 2011 subí la primera versión al *Market*. Para el día 23 de octubre, con la versión 4.0 (errores corregidos) estas son las estadísticas:

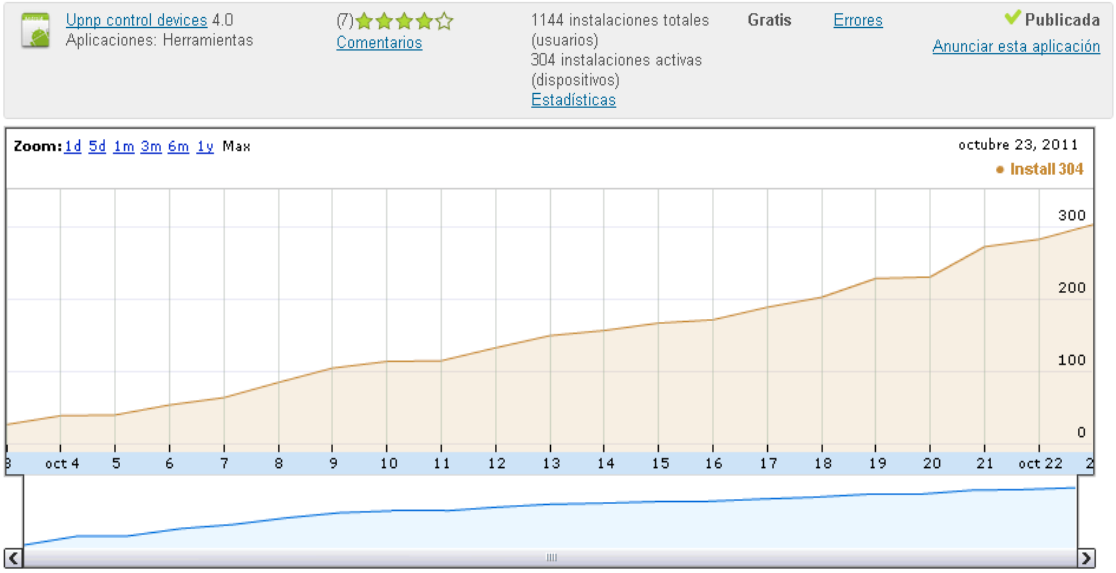
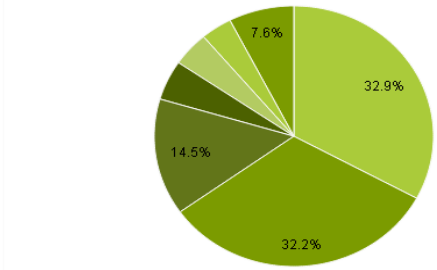


Ilustración 92 Grafica de descargas de la aplicación

Análisis de atributos hasta 23 de octubre de 2011

Versión de plataforma

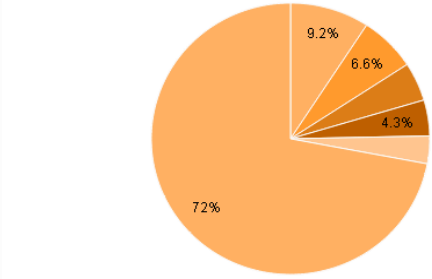


	javier.com	
1	Android 2.2	32,9% (100)
2	Android 2.3.3+	32,2% (98)
3	Android 2.1	14,5% (44)
4	Android 3.1	4,9% (15)
5	Android 1.6	4,3% (13)
6	Android 3.2	3,6% (11)
7	Android 2.3	2,6% (8)
8	Android 3.0	0,3% (1)
9		
10		

Todas las aplicaciones de mi categoría	
Android 2.3.3+	37,5%
Android 2.2	36,6%
Android 2.1	9,2%
Android 1.6	2,1%
Android 1.5	1,4%
Android 3.1	0,7%
Android 3.2	0,5%
Android 2.3	0,4%
Android 3.0	0,1%
Android 1.1	0,0%

Ilustración 93 Grafica de las distintas versiones de Android utilizada por los usuarios de mi aplicación

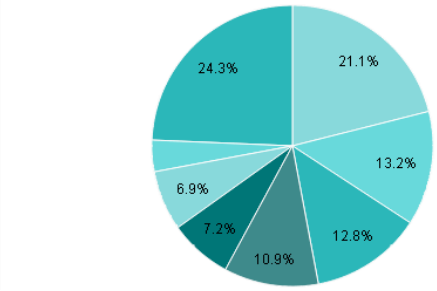
Dispositivo



	javier.com	
1	Samsung Galaxy S2	9,2% (28)
2	Samsung Galaxy S	6,6% (20)
3	Samsung Galaxy Mini	4,6% (14)
4	Samsung Europa	4,3% (13)
5	Acer Picasso	3,3% (10)
6	Samsung Galaxy Ace	2,6% (8)
7	HTC Desire HD	2,3% (7)
8	SEMC Xperia Arc	2,3% (7)
9	Asus EeePad Transformer TF101	2,3% (7)
10	LG Optimus One	2,3% (7)

Ilustración 94 Grafica de los distintos tipos de dispositivos de los usuarios que utilizan mi aplicación

País

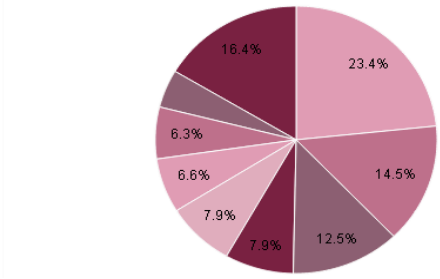


	javier.com	
1	España	21,1% (64)
2	Francia	13,2% (40)
3	Brasil	12,8% (39)
4	México	10,9% (33)
5	Alemania	7,2% (22)
6	Italia	6,9% (21)
7	Estados Unidos	3,6% (11)
8	Suiza	2,3% (7)
9	Chile	2,0% (6)
10	Reino Unido	1,6% (5)

Todas las aplicaciones de mi categoría	
Estados Unidos	35,8%
Corea del Sur	11,1%
Japón	9,7%
Alemania	3,0%
Reino Unido	2,9%
Francia	2,6%
España	1,8%
Taiwán	1,6%
Rusia	1,4%
China	1,4%

Ilustración 95 Grafica de los distintos países de procedencia de los usuarios que utilizan mi aplicación

Idioma



	javier.com	
1	español (España)	23,4% (71)
2	francés (Francia)	14,5% (44)
3	portugués (Brasil)	12,5% (38)
4	español (Estados Unidos)	7,9% (24)
5	alemán (Alemania)	7,9% (24)
6	italiano (Italia)	6,6% (20)
7	español (México)	6,3% (19)
8	inglés (Estados Unidos)	4,6% (14)
9	neerlandés (Países Bajos)	1,6% (5)
10	inglés (Reino Unido)	1,6% (5)

Todas las aplicaciones de mi categoría	
inglés (Estados Unidos)	37,0%
coreano (Corea del Sur)	11,1%
japonés (Japón)	9,6%
inglés (Reino Unido)	5,0%
alemán (Alemania)	3,1%
francés (Francia)	2,8%
chino (Taiwán)	2,0%
español (España)	2,0%
ruso (Rusia)	1,6%
chino (China)	1,4%

Ilustración 96 Grafica de los distintos idiomas de los usuarios que utilizan mi aplicación

Más de 1.100 descargas en 21 días con un 26,57% de instalaciones activas. Muchas de las mejores aplicaciones y juegos están al torno del 40% de instalaciones activas, pero lo más común es estar alrededor del 30%.

4.3 Como obtener beneficio económico

4.3.1 Introducción

Hay varias maneras de obtener beneficio económico de nuestras aplicaciones en Android:

- Aplicación de pago: El desarrollador recibe el 70% del precio que pone a la aplicación, el resto, llamado tarifa de transacción se lo queda Google. Poner publicidad en las aplicaciones de pago no es muy común.
- Aplicación gratuita: El único beneficio se obtiene de la publicidad.
- Botón de donación: Tanto para aplicación de pago como gratuita.

Para el proyecto la he realizado como aplicación gratuita con publicidad.

Como empresa de gestión de publicidad para la aplicación estuve en duda entre dos:

Airpush: la publicidad aparece como notificación y los beneficios por clic son superiores a la otra alternativa. La pega es que existen programas para bloquear la publicidad.

AdMob: Es la que he usado por accesibilidad (clicando en Anunciar esta aplicación nos lleva a nuestra cuenta de admob), pertenece a Google.

Ilustración 97 Anunciar la aplicación

Dependiendo de unos usuarios finales u otros se deberá elegir una u otra empresa para obtener mayor beneficio. Si por ejemplo tienes muchos usuarios en EE.UU., mejor elige una empresa de publicidad de dicho país, los ingresos serán mayores.

4.3.2. Registro en AdMob

Para añadir la publicidad de AdMob se sigue los siguientes pasos: Tras pulsar en "Anunciar esta aplicación" nos lleva a la página de AdMob. La primera vez se debe de rellenar los datos bancarios o asociarlo a una cuenta PayPal. Después aparece lo siguiente:

Ilustración 98 Pantalla principal de AdMob

En esta pestaña (Campañas) se pueden crear campañas de publicidad con el presupuesto deseado, pero primero se tiene que registrar la aplicación en AdMob. Para ello clicar en la pestaña Sitios y aplicaciones. La primera vez deberemos de elegir el tipo de aplicación que deseamos publicitar, elijo aplicación Android:

Ilustración 99 Diferentes tipos de aplicaciones que AdMob proporciona publicidad

Tras esto deberemos de introducir los datos de nuestra *app*:

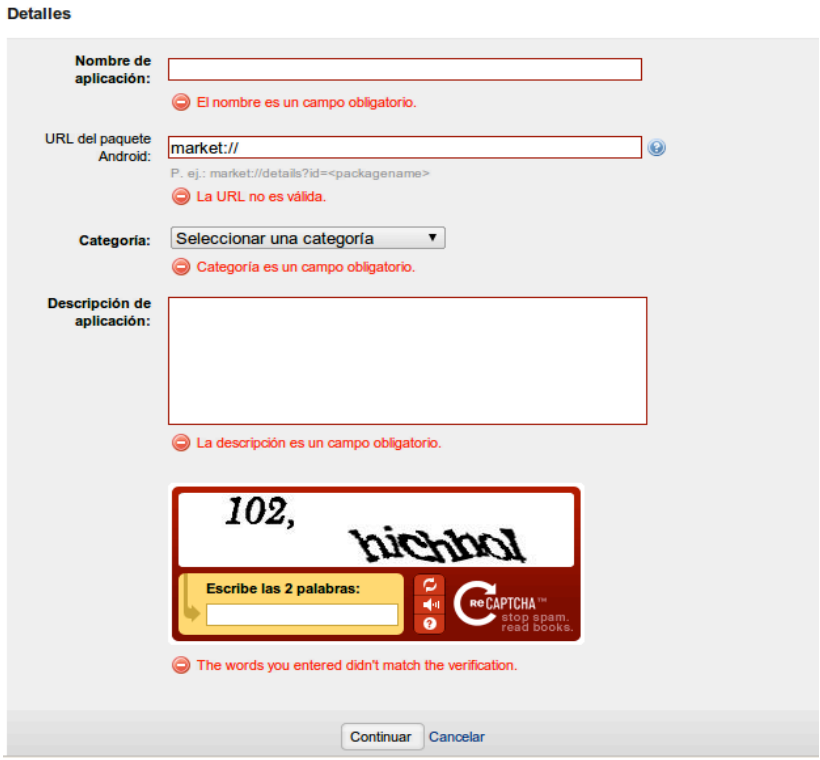


Ilustración 100 Rellenar datos de nuestra aplicación para registrarse en AdMob

La URL de paquete es el paquete que usa el proyecto, en mi caso:
market://details?id=<javier.com>

Ilustración 101 Captura de como insertar la publicidad en nuestra *app*

Clicar en continuar.

Ya está registrada la *app*. Si vamos a "Sitios y aplicaciones" aparecerá un menú con detalles de la cuenta, el beneficio de cada aplicación que tengamos, el porcentaje de impresiones de publicidad a los usuarios y del país de donde proceden, entre otros muchos datos.

En el proyecto, inserte la publicidad el día 4 de Octubre, a día 24 de Octubre los beneficio son de 4.99$, lo que no está nada mal para tan solo 20 días.

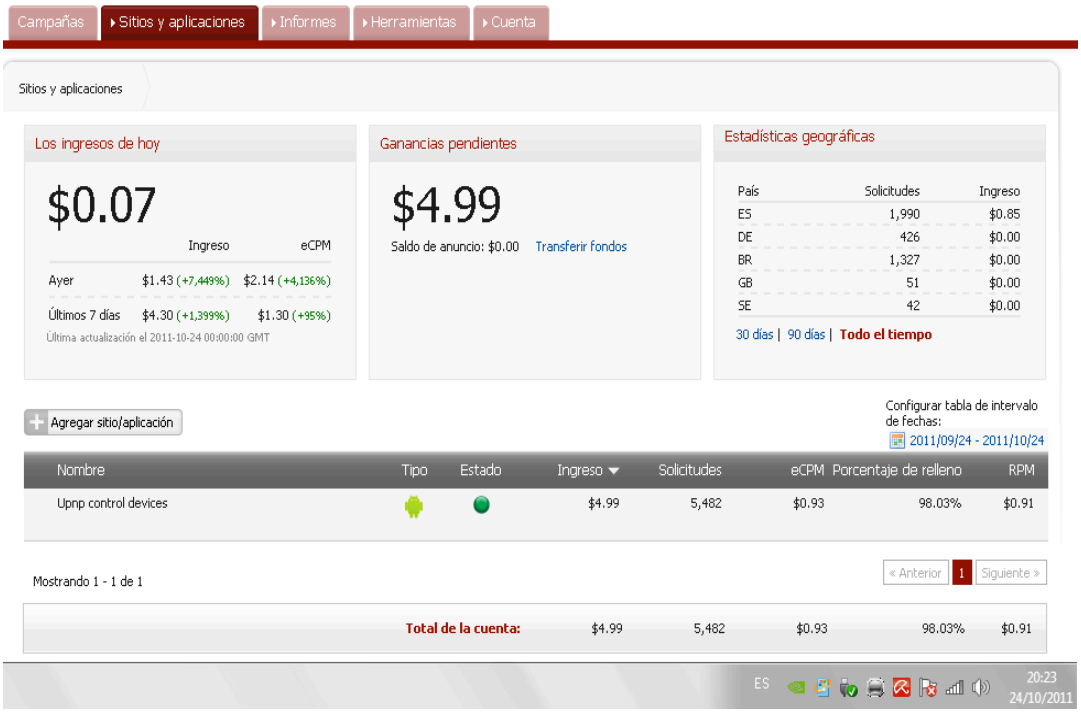


Ilustración 102 Estadísticas de la publicidad de nuestra aplicación

4.3.3. Como insertar la publicidad en la aplicación con AdMob

En la ilustración 105 aparece un enlace con una serie de archivos, donde nos explica cómo montar el SDK AdMob en nuestro proyecto así como la librería Admob.

Ahora bien, al registrar una aplicación en AdMob le asocia un identificador único llamado ID de editor. Este identificador es usado para llevar el control de la publicidad y estará en parte del código. Para conocer nuestro ID, clicamos en "Gestionar aplicación" del menú de aplicaciones de "Sitios y aplicaciones".

Ilustración 103 Como encontrar el ID de editor (paso 1)

El identificador es el código donde pone ID de editor de la ilustración de abajo:

Ilustración 104 Como encontrar el ID de editor (paso 2)

Descargar la librería admob en .jar y lo añadimos a nuestro proyecto como librería externa.

Por otra parte, para que se pueda ver la publicidad, se debe añadir unas líneas de código para cada actividad que muestre la publicidad.

Lo primero en el AndroidManifest.xml, para que tenga permisos de internet debemos añadir:

```
<uses-permission
      android:name="android.permission.INTERNET" />
      <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
para poder usar la librería admob:
<activity android:name="com.google.ads.AdActivity"
          android:configChanges="keyboard|keyboardHidden|orientation"/>
```

Luego por cada actividad que deseemos que aparezca la publicidad, deberemos añadir en su correspondiente *layout* lo siguiente:

```
<LinearLayout
 xmlns:android="http://schemas.android.com/apk/res/android"
          xmlns:ads="http://schemas.android.com/apk/lib/com.google.ads"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:id="@+id/layout" android:orientation="vertical">
   <com.google.ads.AdView android:id="@+id/adView"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                ads:adUnitId="MY_AD_UNIT_ID"
                ads:adSize="BANNER"
                ads:loadAdOnCreate="true"/>
```

Y en la parte de código de la actividad:

```
import com.google.ads.*;
private static final String MY_AD_UNIT_ID = "xxxxxxxxxxxxx"; // ID de editor
Luego en el metodo onCreate:
adView = new AdView(this, AdSize.BANNER, MY_AD_UNIT_ID);
    // Look up the AdView as a resource and load a request.
    // Lookup your LinearLayout assuming it's been given
      // the attribute android:id="@+id/mainLayout"
      LinearLayout layout = (LinearLayout)findViewById(R.id.layout);

      // Add the adView to it
      layout.addView(adView);

      // Initiate a generic request to load it with an ad
      adView.loadAd(new AdRequest());
```

Ya podemos subir la nueva versión de la aplicación, eso sí, recordar incrementar en uno el valor de la versión de la aplicación en el archivo AndroidManifest.xml.

```
android:versionCode="2"
    android:versionName="1.1"
```

Anexo 5 Localización. Como realizar una aplicación multi-idioma

5.1 Introducción. ¿Qué es la localización?

Se llama localización a la forma en que se adecua el software para una zona, gracias a componentes (locales o fuentes) para poder realizar por ejemplo la traducción de textos.

Gracias a ello, la aplicación se convierte en más flexible, pudiendo precargar en tiempo de ejecución varios módulos, que contienen las distintas fuentes que requiere la solicitud elegida en la aplicación cuando esta haya comenzado. Es decir, no se necesita construir múltiples aplicaciones para cargar en una única aplicación varios recursos locales. Puedes poner tus recursos en un modulo de recursos.

También puedes compilar recursos de varias lugares en una única aplicación o módulo. Incluso se permite acceder en tiempo de ejecución al gestor de recursos para poder cambiar los *resources*.

Tras esta breve introducción a la localización vamos a describir un poco como funciona, y como se ha usado en el proyecto centrado en los idiomas, consiguiendo una aplicación multilenguaje, ahorrando mucho tiempo obteniendo mayor eficacia para futuras modificaciones.

5.2 ¿Por qué he elegido la localización como la forma de realizar la aplicación?

Inicialmente para realizar el proyecto se me ocurrió una forma demasiado costosa en cuanto a tiempo, ineficaz y ensucia mucho el código.

Esa solución consistía en que cuando se cambia de idioma mientras se está ejecutando la aplicación, se llama a un método que contiene todos los identificadores de cada elemento por cada idioma, y esto por cada clase o modulo de la aplicación.

Así que muchas de estas palabras se estarán repitiendo y si queremos añadir una palabra, modificarla o incluir un nuevo idioma el trabajo es muy costoso ya que tendríamos que modificar todas las clases.

Una pequeña parte de código de ese método lo pongo como ejemplo, para que se den cuenta del gran problema. No aparecen casi etiquetas ni idiomas, así que imaginen que caos y magnitud puede alcanzar esta solución:

```
public void cambiarIdioma(String idi){
                if(idi=="Euskara"){
        but1.label="Sartzea";
        labNombre.text="Izena";
        labPass.text="pasahitza";
        but2.label="Miatu";
        labTelefono.text="Telefonoak";
        labEmail.text="Email";
        labTitle.text="Titulo";
        msj0.text="Izena sartzea";
        msj1.text="Pasahitza sartzea";
        msj2.text="Email sartzea";
        msj3.text="Telefonoak sartzea";
        alerta="Datua introductua";
  …
    }else if(idi=="English"){
        but1.label="Entry";
        labNombre.text="Name";
        labPass.text="Password";
        but2.label="Registry";
        labTelefono.text="Telephono";
        labEmail.text="Email";
        labTitle.text="Title";
        msj0.text="You must Introduce the name.";
        msj1.text="You must Introduce the pass.";
    …
    }else if(..){
…
    }
}
```

Con la localización, este trabajo se reduce mucho. Por ejemplo para introducir un nuevo idioma introducimos las palabras necesarias para ese idioma en un fichero, cargando el nuevo idioma en tiempo de ejecución cuando se seleccione.

5 3. ¿Cómo funciona?

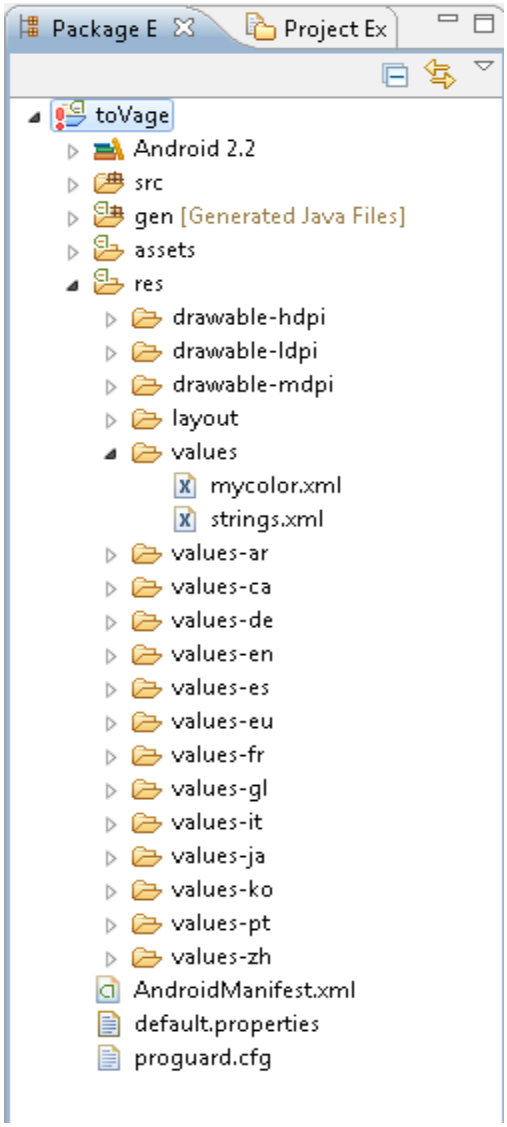


Ilustración 105 Estructura de la aplicación multi-idioma

Un proyecto Android tiene una serie de recursos contenidos en la carpeta /res. Estos recursos pueden ser cadenas de texto, *layouts*, sonidos, gráficos,… cualquier conjunto de datos estáticos. Al lanzar la aplicación, el sistema Android carga estos recursos en función de la configuración del dispositivo.

No es necesario cargar todos los recursos cada vez que deseemos cambiar de configuración en tiempo de ejecución. Cuando cambiamos de idioma en la aplicación, lo que hace Android es cargar una nueva configuración con los recursos que le correspondan al nuevo idioma.

El fichero clave es \res\values\strings.xml, y contiene los textos que aparecen en la aplicación. Si lo que deseamos es que la aplicación aparezca en el idioma del dispositivo, deberemos crear una carpeta por cada idioma del que queramos que esté disponible la aplicación.

La carpeta a crear estará dentro de /res, copiaremos la carpeta *values* con los ficheros que contenga (strings.xml por lo menos, ya que podemos introducir más recursos como imágenes para distintos idiomas), nos aparecerá una ventana indicando que no puede haber dos carpetas con el mismo nombre, por eso pondremos otro con el nombre *values* seguido de un indicador o sufijo del idioma. Ej. values-es

Si por ejemplo en el fichero strings.xml de la carpeta *values* contiene una entrada con el nombre *name* que contenga "*welcome to app*", si deseamos que con el idioma español aparezca en castellano, abriremos el fichero strings.xml de la carpeta values-es y en la entrada *name* introducir "Bienvenido a la aplicación" y guardamos.

Cuando arrancamos la aplicación si tenemos por defecto en el terminal como idioma el español, nos cargara los textos de la carpeta values-es. Por defecto suele ponerse el idioma Inglés en la carpeta *values*.

Si algún elemento de la interfaz gráfica no cambia de idioma es porque esa entrada no está especificada en su fichero Strings.xml de la carpeta values-xx, siendo xx el idioma seleccionado.

5.4 Ejemplo de uso de la localización

Al lanzar una aplicación, el sistema Android carga automáticamente los recursos en función de la configuración del dispositivo, por lo tanto, la aplicación aparecerá en el terminal en el idioma por defecto del usuario.

Pero si se desea que la aplicación aparezca en un idioma distinto al que tiene por defecto el dispositivo, ya sea porque no esté disponible en los idiomas de Android (hay unos 44 idiomas por ahora) o porque simplemente se desea verlo en otro idioma seleccionándolo de una lista de idiomas, se debe de actualizar la configuración de los recursos en tiempo de ejecución.

La aplicación realizada para el proyecto está disponible en 13 idiomas, de los cuales varios no están disponibles en ningún terminal. Por lo tanto se han creado 13 carpetas nuevas (copias de *values*) con el fichero strings.xml traducido a sus respectivos idiomas. La carpeta tendrá el nombre values-xx, donde xx corresponde al identificador del idioma (ISO 639-1).

Una vez montada la estructura, si por ejemplo deseamos que la aplicación se muestre o cambie a Euskera, deberemos de introducir las siguientes líneas en el código.

```
String idioma="eu";
Locale locale= new Locale (idioma);
Locale.setDefault(locale);
Configuration configuración= new Configuration();
configuración.locale=locale;
getBaseContext().getResources().updateConfiguration(configuración,
getBaseContext().getResources().getDisplayMetrics());
```

Con estas líneas forzaremos a la aplicación a cargar el idioma Euskera independientemente del idioma en el que este el terminal.

De esta forma tan sencilla se le da acceso a multitud de usuarios, ya que al acceder al *Market* solo te aparecen las aplicaciones disponibles en tu idioma. Un compañero tenía una aplicación en varios idiomas, pero en cuanto le añadió el idioma Japonés, casi el 50% de las descargas provenían de Japón, expandiendo el mercado.

Anexo 6 Glosario

SatelliteBox: Dispositivo virtual que simula una televisión real.

***Activity* o actividad:** Corresponde con una pantalla de la interfaz del usuario. Una aplicación está formada por una serie de actividades que interactúan entre sí.

***Intent*:** Se les llama intenciones y ofrecen un servicio de paso de mensajes que permite interconectar componentes de la misma o de distintas aplicaciones en Android.

Ej. al arrancar actividad

UPnP: Protocolo que habilita la aparición de nuevo hardware que sería detectado e instalado, sin necesidad de una conexión física con la computadora que lo utiliza. UPnP es un servicio que permite detectar y utilizar automáticamente nuevos dispositivos de una red.

***Market* de Android:** Tienda de aplicaciones para Android, actualmente se conoce por Google Play.