Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Control vocal de una vivienda inteligente en Android

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Joaquín Grimalt Camarena

Tutor: Joan Josep Fons Cors

Curso 2013-2014

Resumen

En este proyecto se desarrolla en Android una aplicación que, usando el reconocedor vocal integrado ASR (Android Speech Recognition), permite el control vocal de una vivienda inteligente (encender/apagar luces, subir/bajar persianas, obtener el estado de los dispositivos, etc.).

A este componente se le provee de ciertas capacidades de razonamiento/inferencia, de manera que no es necesario predefinir el tipo de comandos vocales que se pueden usar. Como punto de partida se toma una emulación de vivienda inteligente, con un conjunto de dispositivos y acciones posibles sobre ellos.

La aplicación debe ser lo suficientemente flexible para poder entender, no sólo expresiones predefinidas 'exactas', sino diferentes variantes. Así, por ejemplo, 'encender la bombilla del comedor' o 'dar la luz del comedor', se tratan como equivalentes.

La comunicación con los dispositivos es a través de peticiones REST.

Palabras clave: Android, voz, dispositivo, vivienda inteligente, REST.

Abstract

In this project develops an Android application that using the integrated ASR speech recognizer (Android Speech Recognition) allows voice control of a smart home (switch on/off lights, raise/lower blinds, get the device status, etc.) .

This component provide certain reasoning abilities / inference, so is not be necessary to predefine the type of voice commands that can be used. As a starting point an emulation of a smart home is taken, with a set of devices and possible actions on them.

The application must be flexible enough to understand not only predefined 'exact' expressions, but also different variants. For example, 'switch on the dining room light bulb' or ' turn on the dining room light', are treated as equivalents.

Communication with the devices is made through REST requests.

Keywords: Android, voice, device, smart home, REST.

Tabla de contenidos

1. Introducción

El proyecto “Control vocal de una vivienda inteligente en Android” nace con la idea de desarrollar una aplicación que, ejecutándose sobre un sistema operativo Android, permita controlar mediante la voz dispositivos específicos que se encuentran instalados en una casa inteligente.

1.1 Motivación.

1.1.1 Android.

Hoy en día el sistema operativo Android se ha convertido en el más popular, ocupando gran parte del mercado de los dispositivos móviles. Además, su presencia no se limita únicamente a los teléfonos inteligentes, pudiéndolo también encontrar en mini ordenadores o *tablets*.
A su vez, este tipo de dispositivos están presentes en la mayoría de los hogares y sus capacidades a nivel de hardware son cada día mayores.

La naturaleza de código abierto de Android lo hace especialmente interesante cuando se trata de desarrollar aplicaciones. En nuestro caso, a esta ventaja hay que añadirle la presencia en sus propias APIs (Application Programming Interface) de clases que permiten trabajar con la voz, haciendo posible el desarrollo de este proyecto sin la necesidad de recurrir a librerías de terceros.

Así pues, aprovechemos las posibilidades que Android junto con estos dispositivos nos ofrecen, trasladando el control de los objetos disponibles en una casa domótica de los habituales paneles a una aplicación capaz de reconocer las órdenes pronunciadas por los usuarios.

1.1.2 Domótica.

La domótica es un conjunto de sistemas capaces de automatizar una vivienda, aportando servicios de gestión energética, seguridad, bienestar y comunicación.

Los principales dispositivos de un sistema domótico se pueden clasificar en:

- Controladores, los cuales van a gestionar el sistema según la programación y la información que reciben.
- Actuadores, capaces de ejecutar y/o recibir órdenes del controlador.
- Sensores, para monitorizar el entorno tanto exterior como interior.
- Buses, que son el medio de transmisión de la información.

Según la distribución y la ubicación de los elementos de control podemos distinguir entre cuatro tipos de arquitectura: arquitectura centralizada, arquitectura descentralizada, arquitectura distribuida y arquitectura hibrida o mixta.

Por otra parte, en función del medio empleado en la comunicación podemos encontrar sistemas cableados, sistemas inalámbricos y sistemas mixtos.

Existen distintas tecnologías por las que se puede optar a la hora de implantar sistemas domóticos, destacando entre ellas el estándar de protocolo de comunicación KNX.

1.2 Objetivos del proyecto.

Utilizando el SDK (Software Development Kit) de Android desarrollaremos una aplicación en Java que se ejecutará sobre un dispositivo con Android como sistema operativo y que responderá a las órdenes pronunciadas por el usuario interactuando a través de peticiones REST con dispositivos específicos desplegados en una vivienda inteligente.

El principal objetivo es que la aplicación reconozca las órdenes pronunciadas, identificando el objeto y la acción a ejecutar sobre él, enviando dichas peticiones a la casa inteligente en el formato correcto para que ésta sea capaz de llevarlas a cabo. Además, se debe dotar a la aplicación de cierta flexibilidad a la hora de reconocer esas órdenes evitando la necesidad del uso de órdenes estrictas.

Otro objetivo que marcó el desarrollo del proyecto, una vez conseguido el objetivo principal, es que la aplicación no disponga de interfaz de usuario y que sea un servicio en ejecución que se pueda activar y desactivar también con la voz.

A estos objetivos propios de la naturaleza del proyecto se les debe añadir también el desarrollo de los conocimientos adquiridos durante la carrera, los cuales han permitido la realización de esta aplicación.

1.3 Estructura de la memoria.

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

En el primer capítulo encontramos una pequeña introducción al contenido del documento, presentando a continuación cuál fue la motivación que me llevó a decidirme por este proyecto, así como los objetivos perseguidos con la realización del mismo. Por último, se ofrece una visión general de los contenidos de este documento.

En el segundo capítulo titulado «Propuesta de aplicación», se describen los objetivos de la aplicación, sus características y las necesidades que pretende cubrir. Por otra parte, se informa sobre las líneas de investigación que se han seguido para lograr los objetivos explicando con detalle cada una de ellas.
Por último, se presenta el contexto tecnológico en el cual se desarrolla la aplicación, dando una visión general tanto de Android, como del resto de tecnologías con las que trataremos.

En el tercer capítulo se presenta la estructura del escenario sobre el que se desarrolla la aplicación, exponiendo las diferentes características del mismo y sus funciones. A continuación, se ofrecerá una descripción tanto del tipo de información empleada como de las clases implementadas para su manejo, aportando también información sobre las tecnologías utilizadas.

Conociendo el contexto nos centraremos en el diseño y estructura de la aplicación, explicando el funcionamiento de sus clases y como éstas participan en la consecución de los objetivos finales del software.

Para finalizar el documento, se describen posibles ampliaciones aplicables al proyecto y las conclusiones obtenidas tras la realización de éste.

2. Propuesta de aplicación

2.1 Objetivos de la aplicación.

Se desarrollará una aplicación ligera, que no necesite almacenar información en el dispositivo y que, únicamente, necesite tres permisos especiales: Acceso a Internet, permiso para conocer el estado de la red y permiso para controlar el volumen del audio del dispositivo.

La aplicación debe ser capaz de entender la orden pronunciada y proporcionar una respuesta al usuario informándole sobre la ejecución de la misma. Así mismo, debe poder detectar si le falta algún tipo de información que le impida completar la orden, solicitándosela al usuario.

La aplicación se desarrollará realmente como un servicio en cuyo modo de ejecución, por defecto, no procesará órdenes sino simplemente permanecerá a la escucha a la espera de lo que hemos denominado “frase mágica” que, una vez ha sido pronunciada por el usuario, cambiará el estado del servicio haciendo que comience a procesar las órdenes. El usuario dispondrá de otra “frase mágica” mediante la cual podrá conmutar de nuevo el estado del servicio pasándolo al modo no interactivo.

2.2 Líneas de investigación.

Para la realización de este proyecto se han empleado los conocimientos adquiridos durante la realización de los estudios de Grado en Ingeniería Informática. Siendo necesario completar dichos conocimientos a través de líneas de investigación, referentes a la metodología requerida para el reconocimiento de la voz y el procesado de documentos en los formatos XML y JSON.

2.2.1 Reconocimiento de voz en Android.

El pilar de este proyecto es el uso de la voz, por lo que el primer objetivo de la línea de investigación era conocer las posibilidades que el sistema operativo ofrece para poder procesar nuestras órdenes vocales.
El sistema operativo Android, a partir de su API de nivel 8, ofrece la clase SpeechRecognizer a través de la cual se tiene acceso al servicio de reconocimiento de voz. Conocer su funcionamiento y sus métodos fue el primer paso para empezar a dar forma a la aplicación.

El desarrollo de sencillas aplicaciones de prueba en las que se pudo comprobar el funcionamiento de la clase permitió afianzar lo aprendido y marcar la línea a seguir en el aplicativo final.

Tras disponer de la herramienta que permite sintetizar la voz, se debía conocer las posibilidades que ofrece Android para conseguir enviar una respuesta vocal al usuario.

TextToSpeech es una clase disponible en Android, a partir de su API de nivel 4, que permite sintetizar texto para que pueda ser reproducido por el dispositivo. Se estudió su funcionamiento empleándola en pequeñas aplicaciones de prueba, pudiendo así alcanzar el grado de comprensión necesario para poder integrarla de manera funcional en el proyecto.

2.2.2 Procesado de documentos.

Tras decidir que la información estática del proyecto se iba a introducir a partir de documentos en formato XML, la línea de investigación debía dirigirse a conocer cómo se podida recorrer dichos documentos y extraer su contenido.

Las primeras respuestas llegaron en forma de librerías externas de terceros que ofrecían estas funcionalidades, ante las cuales se realizaron pruebas para ver la viabilidad de estas soluciones en el proyecto. Y, aunque estas librerías podían ofrecer resultados válidos, la idea de incluirlas en el proyecto no convenció, dando un giro en la línea de investigación hacia las propias APIs de Android que evitaría emplear recursos externos.

En este sentido, la respuesta llegó con XmlPullParser, una interfaz que se encuentra disponible en Android a partir de su API de nivel uno y que nos permite recorrer el fichero XML accediendo a los atributos de cada una de sus etiquetas.

Además de la información disponible en ficheros XML, la aplicación debe ser capaz de extraer información de la respuesta en formato JSON que recibe tras realizar una consulta sobre el estado de un objeto de la casa inteligente.

Por la experiencia de las otras búsquedas de información y con la intención de ser fiel a la idea de no incluir librerías externas, el sitio donde iniciar la búsqueda fue directamente la web de desarrolladores de Android, en la que se encuentra el paquete org.json que, mediante sus clases JSONObject, JSONArray y JSONException, proporciona las funcionalidades necesarias para poder obtener los datos de nuestro interés de la respuesta en formato JSON recibida.

2.3 Contexto tecnológico.

2.3.1 Sistema operativo Android.

A día de hoy Android es un sistema operativo altamente conocido y con el que la gran mayoría de usuarios de dispositivos móviles están acostumbrados a trabajar.

Las características que le hacen idóneo para el desarrollo de aplicaciones son dos: su naturaleza *open source* y, por otra parte, que proporciona un conjunto completo de APIs y herramientas de desarrollo, compilación, depuración y emulación.

Como parte negativa encontramos una gran fragmentación de este sistema operativo debido a la existencia de múltiples versiones todavía en uso.

Android emplea un *kernel* Linux, y una máquina virtual propia denominada DVM (Dalvik Virtual Machine).

Los principales componentes que podemos encontrar en una aplicación Android son:

1.- Activity: Componente que define la interfaz de usuario, típicamente asociado a una pantalla de aplicación.

2.- IntenProvider: Componente que responde a notificaciones o cambios de estado. Su activación puede conllevar la creación de un proceso.

3.- Service: Componente sin interface que realiza una tarea en segundo plano.

4.- ContentProvider: Componente que permite a las aplicaciones compartir información.

El ciclo de vida de estos componentes tiene un inicio, cuando Android los instancia en respuesta a un Intent, y un fin, cuando su instancia se destruye, transitando su vida mientras tanto entre los estados activo e inactivo, o en el caso de las actividades, el de visible e invisible para el usuario.

El programador puede controlar este ciclo actuando sobre los siguientes métodos de los componentes:

onCreate(Bundle): Se llama en la creación de la actividad.

onStart( ): Nos indica que la actividad está a punto de hacerse visible para el usuario.

onResume( ): Se llama cuando la actividad va a comenzar a interactuar con el usuario.

onPause( ): Indica que la actividad está a punto de pasar a segundo plano, normalmente porque va a lanzarse a ejecución otra actividad.

onStop( ): La actividad ya no es visible.

onReStart( ): Señala que actividad va a volver a presentarse al usuario tras haberse ocultado.

onDestroy( ): Se activa justo antes de destruir totalmente la actividad.

Todos ellos se pueden sobrescribir para implementar las acciones que nos permitan gestionar el estado de los componentes de la aplicación.

Una aplicación Android es básicamente un proceso Linux que contiene distintos tipos de estos componentes. Un proceso cuya destrucción no puede decidirla el usuario sino el sistema, quedando el proceso en memoria incluso cuando el usuario cierre la app que está ejecutando. Solo será eliminado cuando el sistema necesite memoria disponible para la ejecución de otro.

Los tipos de procesos que podemos encontrar son:

Foreground process (proceso en primer plano).

Visible process (proceso visible).

Background process (proceso en segundo plano).

Empty process (proceso vacío).

Service process (proceso de servicio).

La aplicación desarrollada durante la ejecución de este proyecto se basa únicamente en un tipo de componente, el servicio, mediante el cual conseguimos que nuestro aplicativo se ejecute en segundo plano, siendo el propio sistema operativo el que lo mantiene en ejecución.

2.3.2 XML (Extensible Markup Language).

XML es un lenguaje de marcas desarrollado por el World Wide Web Consortium, el cual es utilizado para almacenar datos de forma legible. Su aplicación no se limita únicamente a Internet, sino que se propone como un estándar para el intercambio de información estructurada. Destacando desde el punto de vista de la interoperabilidad y la globalización por ser una tecnología que permite compartir información de manera segura, fiable y fácil.

En este caso se emplea esta tecnología como medio para proveer a nuestra aplicación de los datos referentes a los objetos y órdenes. Se trata de una decisión personal con la idea de que, en posibles ampliaciones futuras, esta parte no se incluya en el aplicativo, situándola en otro componente externo.

Las posibilidades que se han destacado sobre la interoperabilidad que ofrece este lenguaje lo hacen idóneo para este propósito, pudiendo desarrollarse esa futura fuente de datos externos en cualquier otra plataforma diferente a la elegida para el desarrollo de nuestra aplicación.

2.3.3 REST (Representational State Transfer).

REST es un tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar http, permitiéndonos crear servicios y aplicaciones que puedan ser usadas por cualquier dispositivo o cliente que entienda HTTP.

Para el desarrollo del proyecto y, ante la imposibilidad de disponer de una casa domótica con la que interactuar, Joan Fons –profesor tutor del proyecto– proporcionó un servidor de emulación de una casa inteligente que puede ejecutarse en un pc, y que ofrece un API REST a través del cual podemos interactuar con sus recursos.

Esta aplicación REST funciona sobre HTTP en el puerto 8182 de la máquina donde se ejecuta. Siendo el patrón para invocar el servicio:

http::/servidor:8182/devFunc/{DEVFUNC}

Donde servidor es la dirección ip o el nombre de máquina donde corre el servidor, y {DEVFUNC} es el id del recurso con el que queremos interactuar.

Las únicas operaciones permitidas sobre un recurso son:

GET: Para obtener información sobre él.

PUT: Para solicitar la ejecución de alguna operación del recurso. La acción a realizar se indicará en el Payload de la petición.

2.3.4 JSON (JavaScript Object Notation).

JSON es un formato ligero para el intercambio de datos, basado en un subconjunto de la sintaxis de JavaScript.

Es el formato que se empleará durante la comunicación con el API REST de nuestro servidor de SmartHome, por lo que se debe utilizar en las órdenes enviadas mediante peticiones PUT, e interpretarlo en las consultas realizadas mediante peticiones GET.

3. Descripción del escenario y entrada de datos

3.1 Estructura del escenario utilizado.

Debemos, llegado a este punto, definir los diferentes dispositivos que ofrece a fecha de hoy el escenario y cuales soporta la aplicación.

La aplicación deberá ser capaz de comunicarse con los tipos de dispositivos que a continuación se enumeran, estando estos disponibles, como se ha comentado en el apartado anterior, a través de un servidor de emulación de casa inteligente en ejecución en un pc y que proporciona un API REST para poder interactuar con él.

- Bistate: representan “algo” que puede estar en dos estados, como activo/inactivo. Se puede aplicar a luces, calefactores, electrodomésticos, etc…

- ToggleBistate: igual que los anteriores pero permiten conmutar su estado actual.

- Movement: representan “algo” que se puede mover en dos sentidos, como abrir/cerrar pudiendo además parar su movimiento. Se aplica a ventanas, persianas, estores, etc…

- Numeric: representan “algo” con un valor numérico, asociándolo normalmente a sensores de lluvia, de luminosidad, de temperatura, etc…

(El servidor de la casa inteligente nos ofrece varios dispositivos de cada tipo)

Todos ellos disponen de un identificador único el cual emplearemos a la hora de realizar las peticiones. El problema que presenta este identificador es que está muy alejado del lenguaje natural por lo que no podemos utilizarlo a la hora de identificar el objeto mediante el habla (por ejemplo: la luz central de la cocina tiene como id DF-CUINA.IL.CENTRAL).

Por este motivo, se ha optado por emplear lo que podríamos llamar la descripción del objeto. En el ejemplo anterior esa descripción sería “luz cocina central”, diferenciando en ella tres elementos: El objeto en sí que sería la “luz”, su ubicación en la casa que sería la “cocina”, y un valor diferenciador que sería “central”, el cual permite distinguir entre objetos del mismo tipo en la misma ubicación.

En la metodología utilizada van a ser estos *tokens* los utilizados para identificar a los objetos con los que el usuario quiere interactuar.

3.2 Información estática.

En la aplicación debe estar disponible el identificador de todos los objetos con los que queramos interactuar, junto con información que nos permita detectar esos *tokens* o palabras claves, mediante los cuales podremos identificar el objeto a partir de las palabras pronunciadas por el usuario.

Incluir esa información directamente en el código de las clases, hacía que la aplicación fuese poco flexible, por lo que, con el objetivo de dotar a nuestro proyecto de flexibilidad, se optó por incluirla como información estática dentro de ficheros XML que la aplicación procesará en el arranque.

Con esta decisión ampliábamos la modularidad de la aplicación y dejábamos la puerta abierta a poder en un futuro, sacar fuera esa información pudiendo presentársela al usuario para que él pueda introducir nuevos objetos y *tokens*, adaptando así el producto a sus necesidades.

Los ficheros implementados son data.xml y attributes.xml.

Un ejemplo de la información contenida en data.xml se muestra a continuación:

```
<object
    id="DF-CUINA.IL.CENTRAL"
    descripcion="luz cocina central"
    funcionalidad="togglebistate"
    />
```

Todo objeto con el que queramos interactuar en la casa (de entre los que la casa nos ofrece) debe estar dado de alta en este fichero y disponer de los tres atributos que se muestran en el fragmento anterior.

El atributo “id” se incluirá en la petición HTTP dentro de la *url*, ya que el API de la casa inteligente que estamos utilizando requiere de ese dato para poder saber con qué objeto interactuar.

Por otra parte, “descripcion” se empleará para identificar al objeto en la orden pronunciada por el usuario, este proceso se explicará más adelante en esta memoria.

Y por último, “funcionalidad” informa de qué tipo de objeto se trata. Este atributo es necesario para poder saber si la acción que el usuario quiere realizar sobre el objeto se puede realizar (por ejemplo: no se puede abrir una luz, o bajar un enchufe).

El otro fichero de datos que vamos a emplear es attributes.xml en el cual podemos encontramos el siguiente contenido:

```
<attribute
    type="objeto"
    description="luz"
    />

<attribute
    type="lugar"
    description="cocina"
   />

<attribute
    type="atributo"
    description="central"
    />

<attribute
    type="accion"
    description="encender"
    action="biaON"
    />
```

Esta información se va a emplear a la hora de reconocer la orden del usuario.

Las tres primeras etiquetas, mostradas en el ejemplo, son el resultado de desglosar la descripción de uno de los objetos “luz cocina central”, en sus tres componentes “objeto”, “lugar” y “atributo”.

La última etiqueta de la muestra, refleja una acción que el usuario puede solicitar, observándose en ella que en el caso de las acciones es necesario añadir el campo “action” que incluirá la orden real que se enviará a la casa inteligente, ya que las acciones del lenguaje natural no pueden ser empleadas, como se verá más adelante, en las peticiones finales.

3.3 Estructura e implementación de las clases necesarias.

En este apartado, describiremos las clases que componen la estructura creada para albergar la información descrita en el apartado anterior.

Para almacenar la información referente a los dispositivos disponibles en la casa se ha optado por implementar una clase House_object, a través de la cual se podrá crear objetos que contendrán dicha información. Cada uno de estos objetos dispone de tres atributos que se corresponden con los atributos de cada etiqueta “object” del fichero data.xml. Y un cuarto que contendrá una lista con las acciones permitidas sobre ese objeto.

Respecto a la información contenida en attributes.xml y, tras barajar distintas posibilidades, se ha optado por emplear tablas hash a la hora de mantener esta información accesible vía código.

El porqué de esta decisión se debe a que la disposición de esta información en el fichero XML, se adapta perfectamente a la estructura <clave,valor> de estas estructuras de datos, y a que las tablas hash permiten agilizar las búsquedas realizándolas de modo directo a través del campo valor.

Así, dispondremos de dos tablas hash:

-attributes_list: Que contendrá la lista de objetos, lugares y atributos.

-actions_list: Que contendrá la lista de acciones naturales y su orden específica correspondiente.

3.4 Procesado del escenario.

En este capítulo profundizaremos en el procesado de la información contenida en los ficheros XML.

3.4.1 Manipuladores de datos XML.

Los principales tipos de manipuladores de datos XML son:

- DOM (Document Object Model), técnica de análisis basada en árboles que construye el árbol completo del documento en memoria. Permite acceso a todo el documento de forma dinámica.

- SAX (Simple API for XML) basado en el modelo Push para el procesamiento XML, procesa la información por eventos, manipulando cada elemento a un determinado tiempo sin la necesidad de cargar todo el documento en memoria.

- StAX (Streaming API for XML) y XPP (XML Pull Parser), basados en el modelo Pull, que al igual que el anterior está guiado por eventos, con la diferencia de que devuelve dichos eventos a medida que la aplicación los necesita.

3.4.2 Manejador del escenario con XML PullParser.

XPP ha sido la opción empleada para el procesado de los documentos XML de la aplicación.

A través de los eventos podemos recorrer el documento diferenciando entre su principio y su fin, y el principio o el fin de una etiqueta, extrayendo la información que necesitemos.

```
while (tipoevento != XmlPullParser.END_DOCUMENT){

    if ((tipoevento==XmlPullParser.START_TAG)&&(parseado.getName().equals("object")))
    {
        House_object new_object = new House_object();

        new_object.setId(parseado.getAttributeValue(null,"id"));
        new_object.setDescription(parseado.getAttributeValue(null,"descripcion"));
        functionality = parseado.getAttributeValue(null,"funcionalidad");
        new_object.setFunctionality(functionality);
        if (functionality.equals("togglebistate")){
            new_object.setActions(togglebistate_actions);
        }
        if (functionality.equals("bistate"))new_object.setActions(bistate_actions);

        if (functionality.equals("dimmer")) new_object.setActions(dimmer_actions);

        if (functionality.equals("movement")) new_object.setActions(movement_actions);

        if (functionality.equals("numeric")) new_object.setActions(numeric_actions);

        resultado.add(new_object);
        parseado.next();
        tipoevento = parseado.getEventType();
    }
    else{
        parseado.next();
        tipoevento = parseado.getEventType();
    }

}
```

Métodos como getEventType que devuelve el tipo de evento, next que nos permite acceder al evento siguiente, o getAttributeValue que nos permite obtener el valor de un atributo a partir de su identificador, facilitan el procesado de la información.

3.4.3 Inicialización y utilización.

Como la información que vamos a procesar se encuentra en dos ficheros distintos y el tratamiento que se les va a dar va ser diferente, se han definido dos clases mediante las cuales podremos disponer del contenido de dichos ficheros utilizando XmlPullParser.

XML_parse es la clase que va obtener la información sobre los objetos. Por lo que incluye un único método que devolverá una lista de objetos House_object.

Como se ha explicado anteriormente un objeto House_object, además de la información contenida en el XML, debe poseer una lista que contenga las acciones que se pueden ejecutar sobre ese objeto. Esa información se ha dispuesto en el código de la clase, aunque en futuras versiones debería situarse fuera de la aplicación como se ha hecho con el resto de datos.

Las cuatro listas que corresponden con las acciones permitidas por los cuatro objetos soportados son:

```
List<String> togglebistate_actions = Arrays.asList("biaON","biaOFF","biaTOGGLE");
List<String> bistate_actions= Arrays.asList("biaON","biaOFF");
List<String> movement_actions = Arrays.asList("movaOPEN","movaCLOSE","movaSTOP");
List<String> numeric_actions = Arrays.asList("numaSET");
```

El método principal de la clase XML_parse recibirá un InputStream con el contenido del fichero XML. Esta información se le proporcionará a un objeto XMLPullParser a través del cual se podrá recorrer su contenido, obteniendo los atributos de cada etiqueta y, definiendo con estos, los valores de cada objeto House_object. Cada uno de estos House_object se añadirá a la lista que conforma el resultado devuelto a la clase principal.

```
if ((tipoevento==XmlPullParser.START_TAG)&&(parseado.getName().equals("object")))
{
    House_object new_object = new House_object();

    new_object.setId(parseado.getAttributeValue(null,"id"));
    new_object.setDescription(parseado.getAttributeValue(null,"descripcion"));
    functionality = parseado.getAttributeValue(null,"funcionalidad");
    new_object.setFunctionality(parseado.getAttributeValue(null,"funcionalidad"));
```

El atributo “funcionalidad” va a determinar qué lista de acciones se incluyen en el objeto House_object.

```
if (functionality.equals("togglebistate")){
    new_object.setActions(togglebistate_actions);
}
if (functionality.equals("bistate"))new_object.setActions(bistate_actions);

if (functionality.equals("movement")) new_object.setActions(movement_actions);

if (functionality.equals("numeric")) new_object.setActions(numeric_actions);

resultado.add(new_object);
parseado.next();
tipoevento = parseado.getEventType();
```

Por otra parte, XML_to_Hash es la clase que nos va a permitir procesar la información contenida en el fichero attributes.xml.

El mecanismo que nos permite recorrer el contenido del fichero es el mismo que en la clase anterior, pero el resultado que devolveremos a la clase principal cambia.

Además, necesitamos extraer dos listas diferentes del mismo fichero por lo que se indicará, añadiendo un segundo parámetro en la invocación del método, cuál de ellas estamos solicitando.

Si este parámetro es igual a “atributos” añadiremos los pares “description, action” de todas las etiquetas conformando una tabla hash que nos va a permitir clasificar las palabras reconocidas como acciones, atributos y objetos.

Si, por el contrario, el parámetro es igual a “acciones” solo procesaremos las etiquetas que contengan el atributo “action”, conformando una tabla hash que nos va a permitir saber qué orden específica va asociada a cada acción del lenguaje natural.

4. Arquitectura del servicio

El siguiente apartado tiene como objetivo presentar la arquitectura propuesta detallando el funcionamiento de las clases que la forman.

Al tratarse el producto desarrollado de un servicio, éste no dispone de interfaz gráfica pues la interacción que tendrá el usuario final con la aplicación debe ser completamente vocal.

Viéndolo desde la perspectiva del modelo por capas, la capa de presentación no va a estar presente, disponiendo únicamente de la capa lógica y de la capa de datos.

4.1 Arquitectura por capas.

4.1.1 Capa lógica.

Será la encargada de procesar los datos de entrada, que serán palabras pronunciadas por el usuario, y ejecutar las acciones.

No es necesario mostrar los resultados de la acción ya que estos se verán reflejados en la reacción de los objetos a la orden pero sí que se ofrecerá una respuesta al usuario, en la que se le informe si se ha podido procesar la operación y, en caso contrario, qué problema se ha detectado. Esta respuesta a la orden se proporcionará a través de frases pronunciadas por el dispositivo, en las que se incluirá dicho resultado.

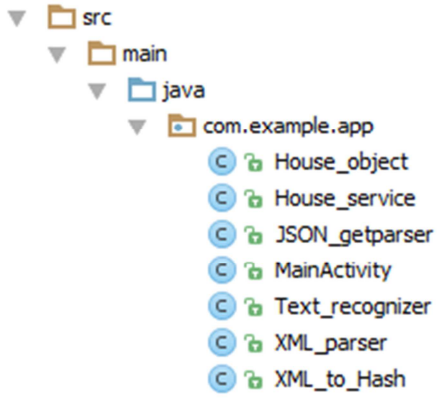
4.1.2 Capa de persistencia.

Debido a que no debemos guardar ningún tipo de información como resultado de los procesos ejecutados en la capa lógica, esta capa solo contendrá los datos incluidos en los ficheros XML detallados en apartados anteriores y las clase encargadas de procesarlos.

4.2 Estructura del servicio.

4.2.1 Directorio src.

Dentro de este directorio encontramos todas las clases que componen nuestra aplicación.



Las clases House_object, XML_parser y XML_to_Hash se han introducido y comentado en el apartado «Estructura e implementación de las clases necesarias» de esta memoria, por ello no entraremos detalladamente en su descripción sino que simplemente recordaremos al lector su cometido: Cargar la información estática en memoria para que pueda ser utilizada durante el ciclo de vida del servicio.

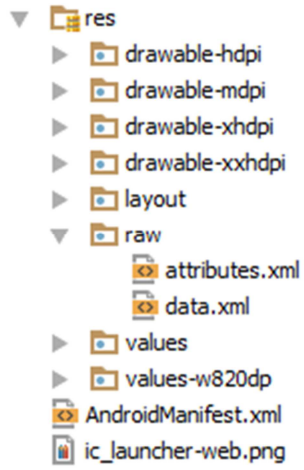
MainActivity es la clase mediante la cual se implementa la actividad principal siendo su única misión lanzar a ejecución el servicio, tras lo cual la actividad finaliza.

El usuario puede requerir conocer el estado de un objeto, o incluso un valor que éste le pueda proporcionar. Para dicho cometido se ha definido la clase JSON_getparser, encargada de recorrer la información recibida como respuesta a una petición de consulta.

House_service y Text_recognizer son las clases que van a llevar el mayor peso del producto final: La primera contiene toda la lógica de la aplicación y permite mantener activa la funcionalidad del servicio. La segunda procesará la frase escuchada, detectando si están presentes los *tokens* definidos en attributes.xml e intentando construir con ellos una descripción que permita identificar un objeto y una acción que ejecutar sobre él.

4.2.2 Directorio raw.

En él se ubicarán los ficheros XML con la información estática.



4.3 Implementación de las clases.

En este apartado se describirán las clases implementadas apoyando la explicación, en caso de que se considere necesario, con fragmentos de código. El lector puede consultar el contenido completo de las clases en el anexo A disponible al final de esta memoria.

4.3.1 MainActivity.

Como ya se ha indicado en apartados anteriores, únicamente lanza el servicio a ejecución.
Si este proceso tiene éxito finaliza su ejecución a través de su propio método finish, y en caso contrario, informa del error.

```
Intent servicio = new Intent(this, House_service.class);

if(startService(servicio)==null)
{
    Toast.makeText(context, "No se ha podido iniciar el servicio", 1).show();
}
else
{
    this.finish();
}
```

4.3.2 JSON_getparser.

Mediante esta clase extraeremos de la información obtenida como respuesta a una petición GET los datos del objeto que resultan de nuestro interés.

La información es recibida por el método principal de esta clase como texto y, tras procesarla, se devuelve el resultado en el mismo formato.

Los datos de interés que debemos extraer son: El tipo de dispositivo, para poder saber cómo interpretar la información, y su estado. Y en el caso de que se trate de un objeto numérico se deberá incluir además sus valores.

```
for (int i=0;i<valuesArray.length();i++){
    if (namesArray.getString(i).toString().equals("currentState")){
        current_state = valuesArray.getString(i);
    }
    else if(namesArray.getString(i).toString().equals("currentValue")){

        JSONObject myjson2 = new JSONObject(valuesArray.getString(i));
        JSONArray  namesArray2 = myjson2.names();
        valuesArray2 = myjson2.toJSONArray(namesArray2);
    }
    else if (namesArray.getString(i).toString().equals("devfunc-type")){
        current_type = valuesArray.getString(i);
    }
}
```

Si el objeto no es de tipo numeric, interpretamos el valor de la etiqueta currentState, en la que se almacena el estado del objeto, para transformarla en información entendible por el usuario

```
if (current_state.equals("bisON"))
    resultado = "encendido";
else if (current_state.equals("bisOFF")){
    resultado = "apagado";
}
else if (current_state.equals("movsMIDDLE")){
    resultado = "en el medio";
}
else if (current_state.equals("movsOPENED")){
    resultado = "arriba";
}
else if (current_state.equals("movsCLOSED")){
    resultado = "abajo";
}
```

Como se observa en la figura, el estado de una luz o de un enchufe está bastante definido, o está apagado o está encendido. Pero no ocurre lo mismo con las persianas o las ventanas.

El API de simulación de la casa inteligente empleada nos proporciona información sobre el porcentaje de apertura o cierre de las ventanas, o sobre el porcentaje de elevación de una persiana. Se planteó incluir esta información en la respuesta pero esto alejaría a la aplicación de la naturalidad que se pretende conseguir durante la comunicación hombre-máquina: Nadie diría la ventana está abierta un 30 por ciento, o la persiana se ha subido un 80 por ciento. Sin embargo, las respuestas “arriba”, “abajo”, o “en el medio”, aunque menos precisas, son más naturales.

Por otra parte, si el tipo del objeto es numeric, significa que la información importante está alojada en otra parte del documento y que, además, debemos procesarla de manera diferente.

```
if (current_type.equals("numeric") && (valuesArray2 != null)){

        if (valuesArray2.length()==2){
            current_unit = valuesArray2.getString(1);
            current_value = valuesArray2.getString(0);

            if (current_unit.equals("oC")){

                resultado = current_value + " grados";

            }
            else if (current_unit.equals("lux")) {

                resultado = current_value + " por ciento de luminosidad".

            }

            else if (current_unit.equals("Km/h")) {

                resultado = current_value + " kilometros por hora";
            }
        }
    }
```

Estos objetos de tipo numeric son sensores de luz, temperatura o velocidad, por lo tanto, además del valor que nos proporcionan se debe analizar la unidad de medida que emplean.

De nuevo, debemos acercar esta información al lenguaje natural detectando dichas unidades y transformándolas en palabras más próximas al usuario. Así, por ejemplo, como se puede ver en el código anterior, el símbolo que emplea el API en el sensor de temperatura “oC” se le devuelve al usuario como “grados”.

4.3.3 Text_recognizer.

Sobre esta clase radica una parte importante del servicio, ya que es la encargada de detectar esos *tokens* o palabras clasificadas, y construir una descripción que nos permita identificar al objeto.

Como se verá en el siguiente apartado, la clase House_service recoge la frase pronunciada por el usuario, crea una instancia de Text_recognizer e invoca a su método principal pasándole como parámetro dicha frase en formato texto.

El reconocedor de texto implementado no interpreta la frase entera como un todo sino que la divide en palabras, intentando detectar las palabras clasificadas que se han introducido en la aplicación a través del fichero attributes.xml.

```
public String recognizer (String text, Hashtable<String,String> diccionario,String object,String locate,String action,String attribute) {

    String[] lista = text.split(" ");
    String result = "";
    objeto = object;
    lugar = locate;
    accion =action;
    atributo = attribute;

    for (int i=0;i < lista.length;i++)
    {
        String encontrado = diccionario.get(lista[i]);

        if (encontrado != null) {

        if ((objeto.equals("vacio"))&& encontrado.equals("objeto")){
            objeto = lista[i];
            /*Podemos tener objetos sin ubicación porque son únicos en la casa*/
            if (objeto.equals("nevera")||objeto.equals("cafetera"))
                lugar="casa";
        }
        else if ((lugar.equals("vacio"))&&encontrado.equals("lugar")) {
            lugar = lista[i];
        }
        else if ((accion.equals("vacio"))&&encontrado.equals("accion")) {
            accion = lista[i];
        }
        else if ((atributo.equals("vacio"))&&encontrado.equals("atributo")){
            atributo = lista[i];
        }

        }

    }
```

Como se puede ver en el fragmento de código anterior, además del texto con la frase, el método recibe una tabla hash que se empleará como diccionario.

Esta tabla, como se explicó en el apartado «Inicialización y utilización», se ha construido a partir de uno de los fichero XML que componen la información estática.

Veamos algunos de los registros que contendrá esta tabla para facilitar la comprensión del código al lector:

<luz,objeto>
<encender,accion>
<cocina,lugar>

Se omitirá, de momento, la explicación referente al uso del resto de parámetros del método para centrarnos en el proceso de reconocimiento.

Utilizando estos dos parámetros, el texto con la frase y la tabla hash diccionario, el proceso de identificación de los *tokens* es sencillo: Primero troceamos la frase utilizando los espacios en blanco como carácter separador, obteniendo un vector de elementos de tipo String, que contiene las palabras pronunciadas. Después recorremos el vector y buscamos en el diccionario si la palabra está presente extrayendo, en caso afirmativo, el tipo de palabra que hemos encontrado. Así ante la frase «*quiero encender la luz central de la cocina*», el método descartará "quiero", "la" y "de", identificando "encender" como acción, "luz" como objeto, y "central" como atributo.

Esta estrategia, basada en palabras clave, dota a la aplicación de flexibilidad ya que no obliga a pronunciar frases fijas con un orden determinado en sus palabras. Por este motivo, frases como «*podrías encender de la cocina la luz central*», «*la luz central de la cocina la quiero encender*», o incluso la frase del ejemplo anterior, llevarían a ejecutar la misma orden sobre el mismo objeto.

Una vez se ha procesado el vector por completo, se puede construir el texto que compondrá el resultado del método.

```
    result = objeto.toString()+" "+lugar.toString()+" "+atributo.toString()+" "+accion.toString();

return result;
}
```

No se realiza ningún control sobre el resultado, estando ese mecanismo implementado en la clase principal House_service que se analizará en el apartado siguiente.

Como se ha comentado anteriormente, el método recibe en su invocación cuatro parámetros más, aparte de la frase y el diccionario.

```
public String recognizer (String text, Hashtable<String,String> diccionario,String object,String locate,String action,String attribute) {

    String[] lista = text.split(" ");
    String result = "";
    objeto = object;
    lugar = locate;
    accion =action;
    atributo = attribute;
```

Las variables que compondrán el resultado del método se inicializan con el valor del parámetro recibido correspondiente por lo que si, por ejemplo, no se pronuncia una palabra que se pueda clasificar como objeto, se devolverá en el resultado el valor recibido como parámetro. Esto nos va a permitir que nuestro servicio mantenga una especie de conversación con el usuario, solicitándole los *tokens* que falten.

Así, si el usuario expresa «*quiero encender un objeto*» tras analizar la frase, en el resultado tan solo dispondremos de la acción "encender", detectando el dispositivo que faltan elementos para poder completar la orden y solicitándoselos al usuario.

Este proceso será explicado con más detalle en el siguiente apartado, destacando únicamente, de momento, la utilidad de esos cuatro parámetros como una especie de memoria sobre los elementos de una orden que todavía no se ha podido determinar.

Dentro de la solución planteada se ha contemplado la posibilidad de que en la casa existan objetos únicos para los que no es necesario indicar su ubicación, siendo habitual, por ejemplo, que una casa disponga únicamente de una cafetera y que ésta esté situada en la cocina, por lo que no debemos obligar al usuario a pronunciar la palabra "cocina" cada vez que quiera que la casa inteligente interactúe con este objeto. Por este motivo durante el reconocimiento si se detectan estos objetos, como son en nuestro caso "nevera" y "cafetera", la variable "lugar", que en otros casos debe de incluirse en la frase, es actualizada por código con el valor "casa".

Es importante destacar que en esta parte del código es donde la elección de tablas hash como estructura de datos para albergar el diccionario cobra más sentido.

A pesar de que este tipo de estructuras se almacenan en memoria durante la ejecución de una aplicación consumiendo este recurso, nos permiten búsquedas directas con una única instrucción evitando recorrer vectores enteros mediante sentencias *for*.

El coste de memoria es inferior al coste temporal que implicaría recorrer los tres vectores: uno para lugares, otro para objetos y otro para acciones, por cada una de las palabras pronunciadas.

4.3.4 House_service.

House_service es la clase principal del proyecto y sobre la que recae el peso del aplicativo. Se trata de una clase que extiende la clase Service y que va a permitir mantener nuestro aplicativo a la escucha pasando de estado latente a interactivo y viceversa, a demanda del usuario.

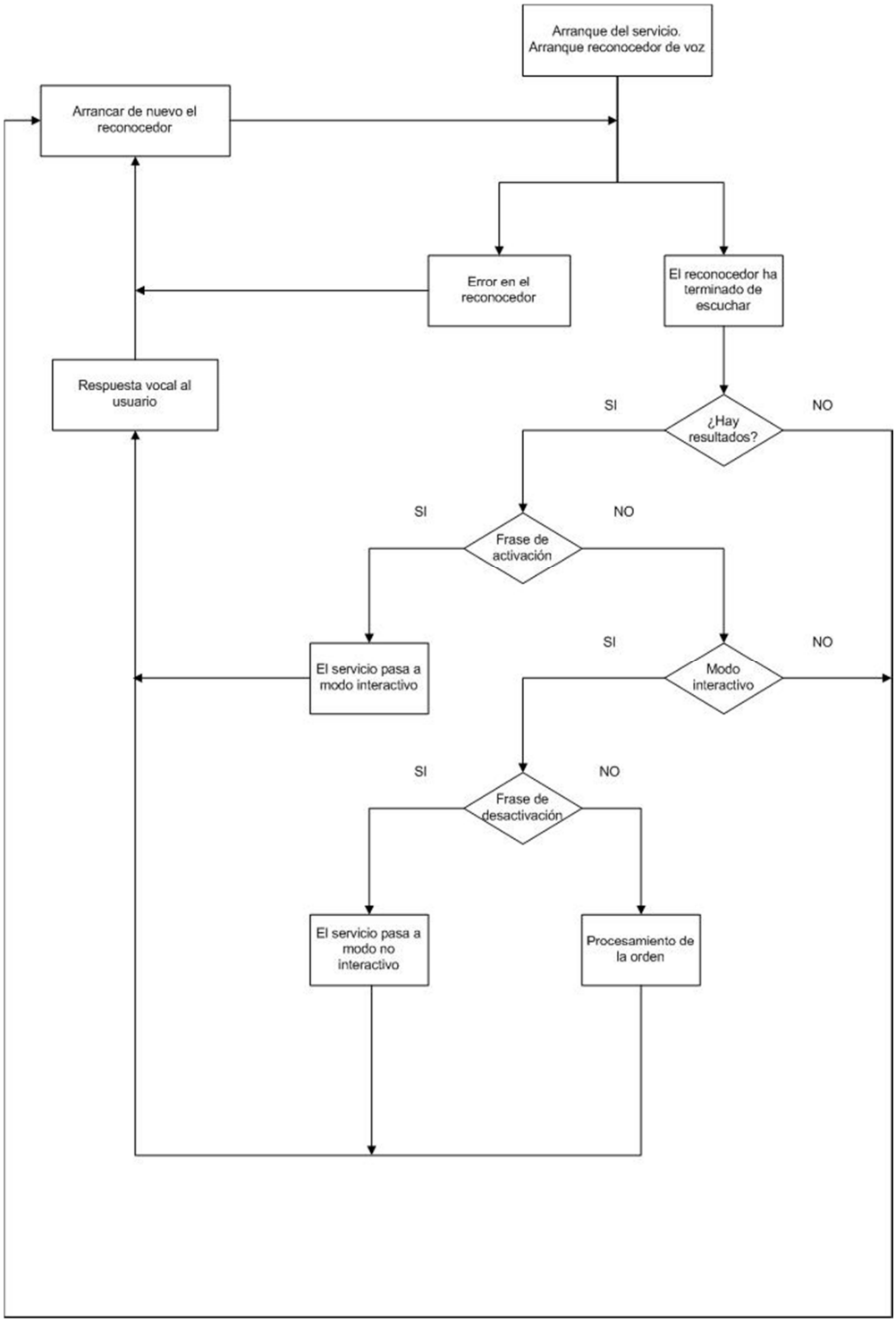
Durante el desarrollo de este proyecto y cuando era necesario empezar a definir el aspecto que iba a tener la aplicación, Joan Fons –profesor tutor del proyecto– en una de las reuniones de seguimiento, planteó la posibilidad de huir de esa parte y desarrollar un servicio con el que se pudiera interactuar únicamente a través de la voz, sin pantalla, sin necesidad de interactuar con el dispositivo empleando las manos. La idea me pareció muy interesante y me recordó como de pequeño "abría los ojos como platos" cuando los tripulantes de la nave "Nostromo" se comunicaban con el ordenador central "Madre" en la película "Alien". Por ese motivo, en la interacción usuario-dispositivo, he incluido palabras que hacen referencia a dicha película.

Espero que el lector me permita esta licencia, y la interprete como lo que es, un simple guiño a una mítica película de ciencia-ficción.

Para poder interactuar mediante la voz con el usuario, el servicio emplea instancias del objeto SpeechRecognizer, disponible en el propio API diecinueve de Android.

Estas instancias, una por ciclo, se pondrán a la escucha, se interpretará su resultado, se destruirán y se volverán a crear durante todo el tiempo que el servicio esté en ejecución. Por tanto, debemos controlar cómo se produce este proceso, decidiendo cuándo se deben destruir, arrancar o detener estos objetos.

Para una mayor comprensión por parte del lector se incluye, a continuación, un diagrama de flujo donde se refleja el ciclo de vida de dicho objeto y en el que podemos observar el funcionamiento general del servicio.



A partir de la imagen anterior se intentará proporcionar al lector una visión general del funcionamiento, incidiendo en cada uno de los procesos en el resto de apartado.

Cuando el servicio arranca por primera vez, crea una instancia de dicho objeto, y la pone a la escucha. Si durante este proceso un error provoca un final anormal de éste, volveremos a crear otra instancia, poniéndola de nuevo en ejecución.

A partir de este momento las frases pronunciadas por el usuario son recogidas por el servicio pero no procesadas, simplemente se analizan en búsqueda de un patrón que corresponda con la frase de activación.
Si dicha frase –que en nuestro caso es "*Hola madre*"– es pronunciada, el estado de nuestro servicio cambiará pasando a modo interactivo.
El usuario será informado de este cambio a través de la frase "*Hola Ripley, ¿en qué puedo ayudarte?*", reproducida por el dispositivo.

En el modo interactivo, las frases pronunciadas son procesadas, interpretándolas y enviando la orden correspondiente a la casa inteligente para que la ejecute. Tanto si dicho proceso se realiza correctamente como si se produce algún error el usuario siempre recibe una respuesta sonora con el resultado.

En cualquier momento, el usuario puede pasar el servicio a modo interactivo pronunciando la frase mágica "*Adiós madre*", recibiendo también una frase de confirmación "*Adiós Ripley*" que le informa de que se le ha entendido.

Después de esta visión general profundizaremos en los métodos principales de la clase, analizando como cada uno de ellos contribuye al funcionamiento global.

4.3.5 StartTask.

Aunque la funcionalidad del método no es muy compleja, ya que su función se limita a arrancar de nuevo el reconocedor, su posición en el ciclo de vida de la actividad del servicio obliga a mencionarlo en los siguientes apartados, por lo que es necesario que el lector conozca de su existencia y funcionalidad para poder comprender de manera más sencilla el resto de los contenidos.

4.3.6 SpeechRecognitionListener.

En esta clase, que implementa la interfaz RecognitionListener, vamos a definir el comportamiento de nuestro reconocedor.

De todos los métodos que nos ofrece la interfaz, solo debemos modificar aquel que se ejecuta cuando se produce un error y aquel que nos permite procesar los resultados.

```
@Override
public void onError(int error){

    /*Si se produce un error hay que relanzar el reconocedor para asegurarnos que el servicio no deje de estar a la escucha*/
    startTask(true);
}
```

En caso de error invocaremos al método startTask, que nos permitirá crear una nueva instancia del objeto SpeechRecognizer para mantener la funcionalidad de escucha.

```
@Override
public void onResults(Bundle results)
{
    /*El reconocedor ha procesado la voz, guardamos el resultado para poder procesarlo*/
    words = results.getStringArrayList(SpeechRecognizer.RESULTS_RECOGNITION);


    assert words != null;
    if (words.get(0) != null){

        if (words.get(0).equals("Hola madre")){ //Si se pronunica la frase de activación el servicio comienza a procesar ordenes
            awaken();
        }
        else if (iAmSpeaking){ //Si ya se ha pronunciado la frase de activación el valor de la variable será verdadero

            if (words.get(0).equals("adiós madre")){ //Esta frase la utiliza el usuario para indicar al servicio que deje de procesar ordenes
                sleep();
            }
            else {
                proceed(words.get(0)); //Se procesa la frase pronunciada
            }
        }
        else {
            startTask(true); //Nos aseguramos de que el reconocedor no deja de estar a la escucha reiniciandlo a través del metodo startTask
        }
    }
    else {

        convertTospeech("No te he entendido"); //Se le comunica al usuario que no se le ha entendido. No debería nunca activarse esta opción

    }
}
```

En el método referente a los resultados es donde vamos a determinar el comportamiento que va a seguir el servicio. Así, si el servicio no está en modo interactivo, tan solo la frase mágica "*Hola madre*" tendrá efecto, invocando al método awaken que cambiará su modo al de escucha activa. Por el contrario, si ya está en modo interactivo, el servicio procesará las frases pronunciadas salvo que ésta sea la otra frase mágica "*Adiós madre*" que cambiará el estado del servicio al modo de escucha no activa.

4.3.7 ConvertTospeech.

Este método nos va a permitir ofrecer una respuesta sonora al usuario.

Cuando el servicio está en modo interactivo todos los métodos que intervienen en el diagrama de flujo, mostrado más arriba en este mismo capítulo, terminan invocándolo.

El usuario siempre debe tener una respuesta que le permita saber el resultado de su solicitud, por este motivo, la primera sentencia de este método destruye el objeto SpeechRecognizer ya que hasta que no se responda al usuario no debemos seguir escuchando órdenes.

La frase de respuesta la recibiremos como parámetro en formato texto, siendo un objeto TextToSpeech el encargado de que el dispositivo la reproduzca.

Una característica de este tipo de objetos es que la acción de pronunciar la frase se ejecuta en otro hilo diferente al de la clase que lo instancia, por lo que la clase principal continúa su ejecución perdiendo el control sobre el proceso que permite la reproducción. Debido a esto, si no se contrala que la frase de respuesta se haya terminado de pronunciar, ésta se puede solapar con el reconocedor de voz, que la clase principal ha vuelto a poner en ejecución para mantener la funcionalidad del servicio.

Para evitar que se produzca este efecto, la nueva instancia del reconocedor solo debe ejecutarse cuando la reproducción de la repuesta haya terminado. Para conseguirlo utilizamos un *handler* que recibirá el mensaje del hilo secundario cuando el objeto TextToSpeech termine de hablar, siendo el propio *handler* el que arrancará de nuevo el reconocedor invocando al ya conocido método startTask.

4.3.8 Proceed.

Como se ha indicado anteriormente, si el servicio está en modo interactivo y la frase que se pronuncia no es la frase de desactivación, ésta debe de ser procesada.

Proceed es el método que será invocado para iniciar el proceso que transformará la frase pronunciada, en una orden que se pueda ejecutar sobre un objeto de la casa.

En su invocación este método recibe el texto recogido por el reconocedor, y tras procesarlo con la ayuda de la clase Text_recognizer y del método commands, cuyo funcionamiento se detallará en el siguiente apartado, determina según el estado de tres variables locales de la clase principal la acción a ejecutar.

```
/*si hemos procesado la orden correctamente*/
if (!bad_order) {

    /*Devolvemos las variables a su valor original pues ya se ha determinado el objeto y la acción.
    Cuando esta orden sea realizada se le enviara el feedback al usuario, y estas variable volveran a tomar valores durante el proceso
    de reconociento de tokens del servicio*/
    running_object = "vacio";
    running_place = "vacio";
    running_action = "vacio";
    running_attribute = "vacio";


    if (order.equals("consultar")||order.equals("saber")||order.equals("dime")||order.equals("encendida")||order.equals("encendido")){
        Peticion_GET_REST task = new Peticion_GET_REST();
        task.execute();
    }
    /*A cada acción le corresponde una orden, que es la que realmente se envía al objeto, y a parte de ser una acción valida deben de poder ejecutarse sobre el
    objeto seleccionado. Esto lo comprobamo con la variable global action cuyo valor a sido determinado en el metodo commands*/
    else if (!action.equals("error")){
        Toast.makeText(context,action, duration).show();
        Peticion_PUT_REST task = new Peticion_PUT_REST();
        task.execute();
    }
    else {
        /*Error es también una variable global cuyo contenido se va a determinar el metodo commands y que nos va permitir realizar feedback al usuario
        informándole en caso de que se halla producido algún error*/
        convertTospeech(error);
    }

}
else {
    convertTospeech(error);
}
```

Si la orden no se ha podido determinar, la variable booleana bad_order será verdadera por lo que simplemente se informará al usuario del error. En caso contrario, se presentan tres opciones:

1.- A través de la variable order: Se comprueba si la orden es de consulta ejecutando la tarea propia del método GET.

2.- Mediante el valor de la variable action: Se determina si es una acción valida ejecutando la tarea propia del método PUT.

3.- Si el valor de la variable action es “error”: Se informa del problema al usuario.

4.3.9 Commands.

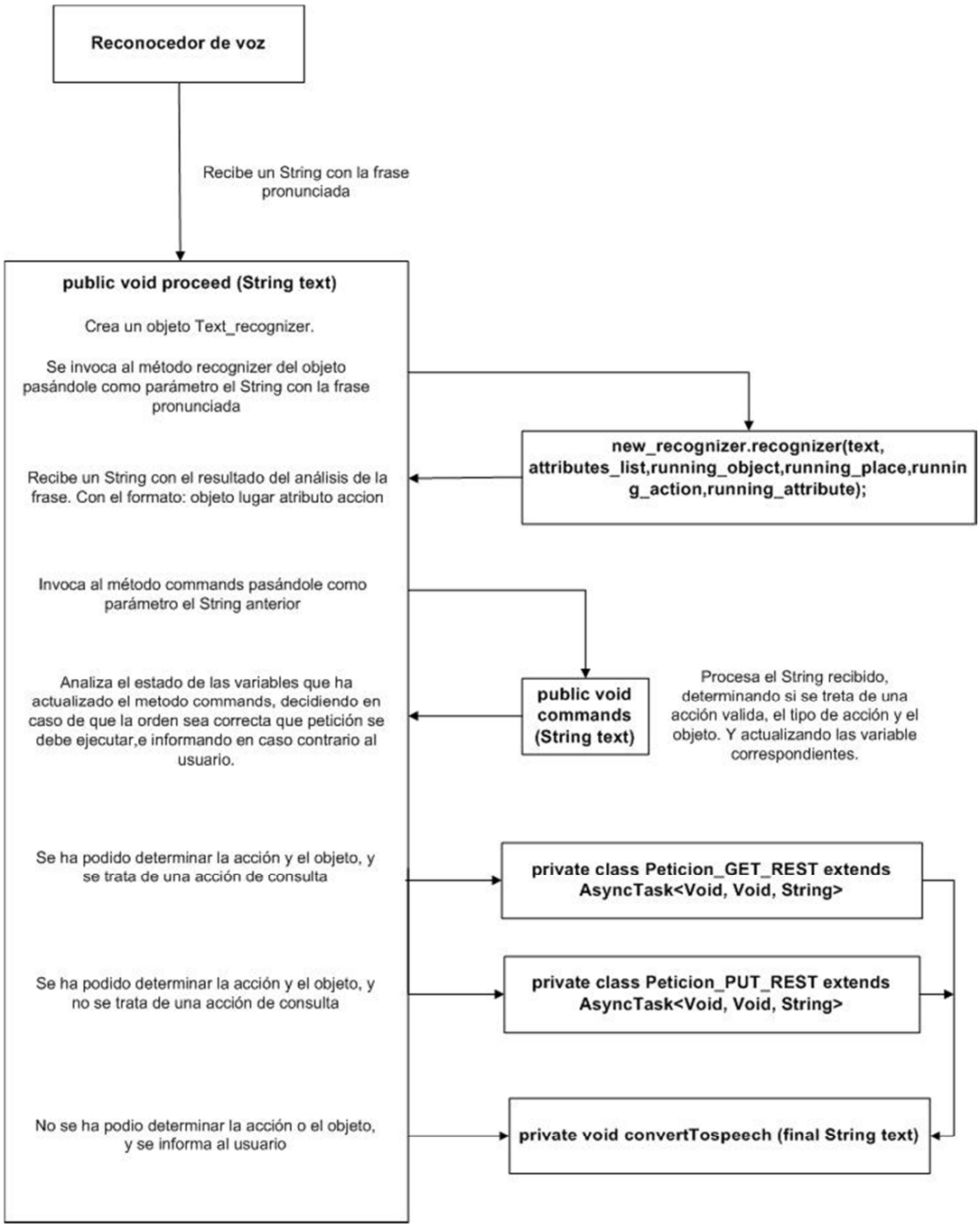
En este apartado se describe el funcionamiento de este método realizando un análisis más detallado debido a su importancia en el proceso de transformación de la orden pronunciada en lenguaje natural al lenguaje específico.

El método recibe como parámetro un texto con la información referente a “objeto” “lugar” “atributo” “acción”.

Tanto la distribución de los valores dentro del texto como el empleo de un espacio para separarlos es fijo y viene determinado por la clase Text_recognizer.

Llegados a este punto, es fácil que el lector haya perdido la visión global del funcionamiento ya que la referencia continua a otros métodos obligan a retroceder o avanzar continuamente en el contenido de la memoria.

El siguiente diagrama intenta solventar este problema, representando un ciclo completo de análisis de una frase para proporcionar al lector una visión global del proceso indicando los métodos que en el intervienen.



El primer paso será, a partir del texto recibido, intentar obtener una descripción que permita determinar de qué objeto se trata, para lo cual el método commands seguirá la siguiente estrategia:

1.- Trocea el texto recibido colocando los valores recibidos en variables correspondientes a "objeto", "lugar", "atributo", y "acción". Se trata de variables locales de la clase principal ya que, como se verá más adelante, su valor debe conservarse durante las sucesivas llamadas al método.

2.- Si las variables de objeto, lugar y acción tienen un valor distinto a "vacío" ya se puede intentar determinar la orden definitiva.

Cabe recordar que en el arranque de la clase principal House_service se ha rellenado una lista con objetos House_object, cada uno de los cuales dispone de un identificador, con la referencia exacta del objeto en la casa inteligente, una descripción que emplearemos para localizar el objeto en la lista, una funcionalidad que nos indica de qué objeto se trata y, por último, una lista de acciones válidas para ese tipo de objeto.

La descripción de cada objeto definida en el fichero data.xml, se compone en su mayoría de dos palabras separadas por espacios que indican el objeto y el lugar, siempre en ese orden. Algunos objetos disponen de un tercer valor en su descripción, que sería el atributo que permite diferenciarlos de otros objetos del mismo tipo situados en el mismo lugar de la casa.

Por otra parte, también existen objetos que, al tratarse de objetos únicos en la casa, solo disponen del valor correspondiente al nombre del objeto en su descripción.

El motivo por el cual, a pesar de la existencia de estos objetos únicos, se ha afirmado que si disponemos de valores distintos de "vacío" para "objeto" y "lugar" podemos construir una descripción válida es porque, como se vió en el apartado correspondiente a la clase Text_recognizer, los objetos únicos son identificados añadiendo el texto "casa" como "lugar" en el resultado.

Veamos un par de ejemplos para poder entender mejor cómo funciona esta parte del código:

Ejemplo 1:

Frase pronunciada por el usuario: "*quiero encender la nevera*".

Texto resultado de Text_recognizer: "nevera casa vacío encender".

Valor de la descripción en commands: "nevera".

Ejemplo 2:

Frase pronunciada por el usuario: "*quiero encender los enchufes del comedor*".

Texto resultado de Text_recognizer: "enchufes comedor vacío encender".

Valor de la descripción en commands: "enchufes comedor."

Por supuesto también se comprueba si el valor de la variable referente al atributo es distinto de "vacío" añadiéndolo a la descripción. Por ejemplo:

Frase pronunciada por el usuario: "*quiero encender la luz central de la cocina*".

Texto resultado de Text_recognizer: "luz cocina central encender".

Valor de la descripción en commands: "luz cocina central".

3.- Si, por el contrario, las variables correspondientes "objeto", lugar" o "acción" contiene alguna de ellas la cadena "vacío", se le debe indicar al usuario que se necesita esa información para poder continuar.

```
bad_order = true;

if (running_object.equals("vacio")){

    error = "Ok Ripley, pero no me has dicho el objeto";
}
else if (running_place.equals("vacio")){

    error = "El objeto, "+running_object+".¿En que lugar de la casa está situado?";

}
else if (running_action.equals("vacio")){

    error = "¿Que quieres que haga con "+running_object+" "+running_place+"?";

}
```

En este caso, la respuesta que se proporciona al usuario no es enviada directamente al método convertToSpeech, ya que el método commands es invocado desde proceed, y es éste el que debe continuar después de que commands termine su ejecución.

Por este motivo las frases de respuesta se almacenan en una variable accesible desde otros métodos cambiando su contenido en función del error detectado.

El valor de las variables que se han actualizado al principio del método no se debe eliminar, ya que almacenan las palabras que ya se han podido identificar, evitando tener que obligar al usuario a que empiece otra vez de nuevo en caso de que falte alguna.

Ilustremos la explicación con un ejemplo:

Usuario: "*Quiero apagar la luz central*".

Servicio: "*El objeto, luz. ¿En qué lugar de la casa está situado?*".

Usuario: "*En la cocina*".

Servicio: "*Orden realizada correctamente*".

Una vez hemos obtenido una descripción con una estructura válida, debemos comprobar si ésta pertenece a algún objeto de nuestra lista. Si la búsqueda del objeto mediante esa descripción tiene éxito, podremos obtener del objeto encontrado su identificador, y la lista de acciones permitidas. El valor del identificador será almacenado en la variable de la clase principal "object".

Si por el contrario, durante la búsqueda no se encuentra un objeto con esa descripción, debemos de controlar dos posibilidades:

1.- La posición destinada al campo atributo contiene el valor "vacío", por lo que es posible que el usuario no haya incluido ese dato en su frase, así que se le solicita.

```
if (running_attribute.equals("vacio")){
error = "¿Con que "+running_object+" "+running_place+" quieres interactuar?";}
```

En una primera búsqueda se acepta que el valor "atributo" no esté inicializado, ya que existen objetos con solo "objeto" y objetos solo con "objeto lugar" en su descripción.

En el siguiente ejemplo se puede observar el proceso completo:

Usuario: "*Quiero apagar una luz*".

Servicio: "*El objeto luz, ¿en qué lugar de la casa está situado?*".

Usuario: "*En la cocina*".

(Se realiza la búsqueda con la descripción "luz cocina" pero no da resultado, así que se solicita el atributo).

Servicio: "*¿Con qué luz cocina quieres interactuar?*".

Usuario: "*Con la central*".

Servicio: "*Orden realizada correctamente*".

2.- Si por el contrario, la posición destinada al campo atributo no contiene el valor "vacío", y la búsqueda no da resultado, significa que la descripción construida no pertenece a ningún objeto.

La estrategia de reconocimiento se basa en localizar palabras y no frases completas, por lo que puede ocurrir que se detecten todos los *tokens* necesarios para construir una descripción pero ésta no corresponda con ningún objeto.

Si, por ejemplo, el usuario solicita "*quiero apagar el aire central de la cocina*", la descripción se podrá construir, ya que "aire" es un objeto, "central" un atributo, y "cocina" un lugar. Pero no existe en la casa ningún objeto con la descripción "aire cocina central".

Detectado este error, como se puede ver en el fragmento de código siguiente, se actualiza el valor de la variable encargada de reflejarlo. Y las variables que contenían los valores detectados se inicializan como vacías descartando los valores identificados en la orden errónea.

```
else{
    running_object = "vacio";
    running_place = "vacio";
    running_action = "vacio";
    running_attribute = "vacio";
    error = "Lo siento Ripley.Pero ese objeto no existe en la casa";
}
```

Una vez disponemos del identificador del objeto y de su lista de acciones permitidas, podemos obtener a partir del contenido de la variable referente a la acción, que "acción" en lenguaje formal le corresponde y si el objeto la soporta.

El término formal que corresponde con la acción detectada en lenguaje natural se obtiene realizando una búsqueda en la tabla hash action_list, comentada en el apartado referente a la información estática, almacenándolo en la variable local de la clase principal "action".

Y para saber si el objeto soporta la acción que representa el término formal obtenido, simplemente comprobamos si dicho término está contenido en la lista de acciones soportadas por el objeto. Si dicha lista no lo contiene, actualizamos el valor de la variable "action" con la cadena "error" y el de la variable de "error" con la cadena "Orden no aplicable a ese objeto".

Como se ha indicado anteriormente, el método commands no devuelve ningún resultado sino que actualiza variables locales de la clase principal que después interpretarán el resto de métodos que intervienen en el ciclo funcional del servicio.

Una de esas variables es la variable booleana bad_order, la cual es actualizada como verdadera cuando se detecta un error. Así, cuando el método proceed descrito en el apartado anterior, recupera el control tras la ejecución de commands, comprueba si esta variable es verdadera enviando el contenido de la variable "error" a convertToSpeech para informar del problema al usuario.

4.3.10 Peticion_GET_REST

Esta clase que extiende AsyncTask define una tarea asíncrona que será utilizada para enviar la petición GET a la casa inteligente.

Este tipo de petición es de consulta, por lo que para su ejecución tan solo necesitamos el valor de la variable "object" actualizada por el método commands.

Esta tarea es ejecutada desde el método proceed tras comprobar que la orden ha podido ser identificada lo que implica que el contenido de la variable "object" es un identificador de objeto válido.

```
if (order.equals("consultar")||order.equals("saber")||order.equals("dime")||order.equals("encendida")||order.equals("encendido")){
    Peticion_GET_REST task = new Peticion_GET_REST();
    task.execute();
}
/*A cada acción le corresponde una orden, que es la que realmente se envía al objeto,
y aparte de ser una acción valida deben de poder ejecutarse sobre el
objeto seleccionado. Esto lo comprobamos con la variable action cuyo valor
ha sido determinado en el metodo commands*/
else if (!action.equals("error")){
    Toast.makeText(context,action, duration).show();
    Peticion_PUT_REST task = new Peticion_PUT_REST();
    task.execute();
}
else {
    /*Error es también una variable cuyo contenido se va a determinar en el metodo
    commands y que nos va permitir realizar feedback al usuario
    informándole en caso de que se halla producido algún error*/
    convertTospeech(error);
}
```

Añadiendo el contenido de la variable "object" a la *url* base de la casa inteligente construimos la *url* definitiva que emplearemos en la petición.

Un ejemplo de construcción de petición correcta sería:

"http://joaono.ignorelist.com:8182/devFunc/'contenido de object'"

Para poder analizar la respuesta que recibimos tras la petición, empleamos la clase "JSON_parser" cuyo funcionamiento ha sido descrito en el apartado sobre el manejo de la información.

El resultado que proporciona la invocación del método “parseando” de dicha clase se guarda en formato texto entregándolo como respuesta vocal al usuario a través del método convertToSpeech.

4.3.11 Peticion_PUT_REST.

De nuevo se trata de una clase que extiende AsyncTask, por lo que se trata también de una tarea asíncrona que será ejecutada por el método proceed, tras determinar éste que la acción a ejecutar no es de consulta sino que se solicita interactuar con un objeto.

En esta petición, además de la variable “object” necesaria para construir la *url* definitiva que permita el acceso al objeto, se necesita utilizar la variable “action” para conocer la acción en lenguaje formal que se debe ejecutar sobre el objeto.

Para obtener la *url* definitiva tan solo se concatenan en la posición correcta los valores de las dos variables junto con el esquema de una petición estándar.

Un ejemplo sería:

“http://joaono.ignorelist.com:8182/devFunc/’contenido de object’{action:’contenido de action’}”

En el método que se activa tras la ejecución de la tarea, comprobamos si la petición ha sido un éxito informando al usuario con la frase “*Orden realizada correctamente*”, o un fracaso con la frase “*No he podido ejecutar la orden*”.

4.3.12 Awaken.

Sencillo método que se emplea para pasar el servicio a modo de escucha activa.

Lo componen únicamente dos instrucciones: Una que activa la variable booleana “iAmSpeaking”, la cual emplea el servicio para saber si debe procesar las palabras adquiridas por el reconocedor. Y otra orden que, a través del método converToSpeech, saluda al usuario mediante la frase, “*Hola Ripley, ¿en qué puedo ayudarte?*”.

4.3.13 Sleep.

Un método igual de sencillo que el anterior, cuyo cometido es pasar el servicio a modo no interactivo estableciendo el valor de la variable "iAmSpeaking" como false y despidiéndose del usuario con la frase "*Adiós Ripley*".

5. Conclusiones y trabajos futuros

5.1 Valoración personal.

El proyecto realizado ha contribuido a afianzar los conocimientos sobre Java y Android adquiridos durante la realización de la carrera, permitiéndome, a su vez, ampliarlos al contemplar durante su realización aspectos sobre los que hasta el momento no había tenido la oportunidad de trabajar.

Además, el proceso de creación de una aplicación funcional a partir de una idea, encargándose de todo el proceso de desarrollo, ha sido especialmente gratificante.

Por otra parte, prescindir de la parte gráfica ha sido un reto que, además de motivador, ha aumentado mi interés hacia el proyecto.

5.2 Trabajos futuros.

La aplicación se puede continuar desarrollando para conseguir funcionalidades sobre las que se han definido las bases pero que no se han podido terminar de implementar.

En un futuro la parte de información estática debería ser configurable desde el exterior de la aplicación, pudiendo el usuario definir nuevos objetos que se instalen en la casa, o nuevas palabras que le permitan interactuar sobre ellos.
La disposición en el diseño actual de la información en ficheros XML permitiría de una manera relativamente sencilla la implantación de esta nueva funcionalidad.

También desde el punto de vista de la configuración, se debería dar al usuario final la posibilidad de elegir las distintas frases que se emplean en la aplicación, cambiando, por ejemplo, la frase de activación o el saludo inicial.

Por otra parte, se debería aumentar la capacidad de conversación hombre-máquina dotándola de una mayor naturalidad.

6. Referencias

A continuación se detallan las fuentes de información empleadas durante la realización de este proyecto:

MCLAUGHLIN, B. y EDELSON, J. (2012). *Java and XML, 3rd Edition* .California: O'Reilly Media.

MEIER, R. (2012).*Professional Android 4 application development*. Indianapolis: John Willey & Sons.

RIBAS LEQUERICA, J. (2011). *Manual imprescindible de desarrollo de aplicaciones para Android*. Madrid: Anaya Multimedia.

ROMERO MORALES, C., VAZQUEZ SERRANO, F. y DE CASTRO LOZANO, C. (2010). *Domótica e inmótica. Viviendas y edificios inteligentes. 3ª Edició*n. Madrid: Ra-ma.

http://developer.android.com

http://json.og/

http://restfulwebapis.com/

http://stackoverflow.com/

http://source.android.com/

http://www.openhandsetalliance.com

ANEXO A: Código de las clases

XML_parser

```
package com.example.app;

import android.util.Log;

import org.xmlpull.v1.XmlPullParser;
import org.xmlpull.v1.XmlPullParserFactory;

import java.io.InputStream;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class XML_parser {

   public XML_parser () {

   }

   /*Como resultado de parsear el fichero se devuelve una lista de objetos House_object*/
   public List parseando (InputStream fichero) {

      /*La lista que vamos a devolver*/
      List resultado = new ArrayList();
      String functionality;
      List<String> togglebistate_actions = Arrays.asList("biaON","biaOFF","biaTOGGLE");
      List<String> bistate_actions= Arrays.asList("biaON","biaOFF");
      List<String> movement_actions = Arrays.asList("movaOPEN","movaCLOSE","movaSTOP");
      List<String> numeric_actions = Arrays.asList("numaSET");

      try {
         XmlPullParser parseado = XmlPullParserFactory.newInstance().newPullParser();
         parseado.setInput(fichero,null);
         int tipoevento = parseado.getEventType();

         /*Vamos a devolver un ArrayList de objetos House_object*/
         resultado = new ArrayList<House_object>();

            while (tipoevento != XmlPullParser.END_DOCUMENT){

               if ((tipoevento==XmlPullParser.START_TAG)&&(parseado.getName().equals("object")))
               {
                  House_object new_object = new House_object();

                  new_object.setId(parseado.getAttributeValue(null,"id"));
                  new_object.setDescription(parseado.getAttributeValue(null,"descripcion"));
                  functionality = parseado.getAttributeValue(null,"funcionalidad");
                  new_object.setFunctionality(functionality);
                  if (functionality.equals("togglebistate")){
                     new_object.setActions(togglebistate_actions);
                  }
                  if (functionality.equals("bistate"))new_object.setActions(bistate_actions);

                  if (functionality.equals("dimmer")) new_object.setActions(dimmer_actions);

                  if (functionality.equals("movement")) new_object.setActions(movement_actions);

                  if (functionality.equals("numeric")) new_object.setActions(numeric_actions);

                  resultado.add(new_object);
                  parseado.next();
                  tipoevento = parseado.getEventType();
               }
               else{
                  parseado.next();
                  tipoevento = parseado.getEventType();
               }
```

```
                }
            }
        catch (Exception e){
            Log.e("House", "Error lejendo XML desde fichero");
        }

        return resultado;
    }

}
```

XML_to_Hash

```
package com.example.app;

import android.util.Log;

import org.xmlpull.v1.XmlPullParser;
import org.xmlpull.v1.XmlPullParserFactory;

import java.io.InputStream;
import java.util.Hashtable;


public class XML_to_Hash {

   public XML_to_Hash () {

   }

   /*Como resultado de parsear el fichero se devuelve una tabla Hash de Strings*/
   public Hashtable parseando (InputStream fichero, String opcion) {

      /*La lista que vamos a devolver*/
      Hashtable<String,String> resultado=new Hashtable<String,String>();
      String type;
      String description;
      String action;

      try {
         XmlPullParser parseado = XmlPullParserFactory.newInstance().newPullParser();
         parseado.setInput(fichero,null);
         int tipoevento = parseado.getEventType();

         while (tipoevento != XmlPullParser.END_DOCUMENT){

            if ((tipoevento==XmlPullParser.START_TAG)&&(parseado.getName().equals("attribute")))
            {
               type = parseado.getAttributeValue(null,"type");
               description = parseado.getAttributeValue(null,"description");
               action = parseado.getAttributeValue(null,"action");

               if (opcion.equals("atributos")){
               /*Conseguimos una tabla hash que nos va a permitir clasificar las palabras reconocidas como
                                                      acciones,atributos,y objetos*/
               resultado.put(description,type);
               }
               else if (opcion.equals("acciones")&& action!=null) {
               /*Conseguimos una tabla hash que nos va a permitir saber qué orden va asociada a cada acción*/
               resultado.put(description,action);
               }
               parseado.next();
               tipoevento = parseado.getEventType();
            }
            else{
               parseado.next();
               tipoevento = parseado.getEventType();
            }

         }
      }
      catch (Exception e){
         Log.e("House", "Error lejendo XML desde fichero");
      }

      return resultado;
   }


}
```

JSON_getparser

```
package com.example.app;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

public class JSON_getparser {

   public JSON_getparser() {

   }

   /*Como resultado de parsear el fichero se devuelve el estado del objeto de la casa*/
   public String parseando (String text) throws JSONException {

      /*El String que vamos a devolver*/
      String resultado = null;
      String current_state = null;
      String current_type = null;
      String current_unit;
      String current_value;
      JSONArray valuesArray2 = null;

      try {
         JSONObject myjson = new JSONObject(text);
         JSONArray  namesArray = myjson.names();
         JSONArray valuesArray = myjson.toJSONArray(namesArray);
         for (int i=0;i<valuesArray.length();i++){
            if (namesArray.getString(i).toString().equals("currentState")){
               current_state = valuesArray.getString(i);
            }
            else if(namesArray.getString(i).toString().equals("currentValue")){

               JSONObject myjson2 = new JSONObject(valuesArray.getString(i));
               JSONArray  namesArray2 = myjson2.names();
               valuesArray2 = myjson2.toJSONArray(namesArray2);
            }
            else if (namesArray.getString(i).toString().equals("devfunc-type")){
               current_type = valuesArray.getString(i);
            }
         }
      }
      catch (JSONException e) {
          e.printStackTrace();
      }

      if (current_type.equals("numeric") && (valuesArray2 != null)){

            if (valuesArray2.length()==2){
               current_unit = valuesArray2.getString(1);
               current_value = valuesArray2.getString(0);

               if (current_unit.equals("oC")){

                  resultado = current_value + " grados";

               }
               else if (current_unit.equals("lux")) {

                  resultado = current_value + " por ciento de luminosidad";

               }

               else if (current_unit.equals("Km/h")) {

                  resultado = current_value + " kilómetros por hora";
               }
            }
         }
         else {

            if (current_state.equals("bisON"))
               resultado = "encendido";
            else if (current_state.equals("bisOFF")){
```

```
            resultado = "apagado";
        }
        else if (current_state.equals("movsMIDDLE")){
            resultado = "en el medio";
        }
        else if (current_state.equals("movsOPENED")){
            resultado = "arriba";
        }
        else if (current_state.equals("movsCLOSED")){
            resultado = "abajo";
        }


    }

    if (resultado!=null)
        return resultado;
    else
        return "error";
  }
}
```

Text_recognizer

```
package com.example.app;

import java.util.Hashtable;

public class Text_recognizer {

   String objeto = null;
   String lugar = null;
   String accion = null;
   String atributo = null;

   Text_recognizer () {

   }

   public String recognizer (String text, Hashtable<String,String> diccionario,String object,String locate,String
   action,String attribute) {

      String[] lista = text.split(" ");
      String result = "";
      objeto = object;
      lugar = locate;
      accion =action;
      atributo = attribute;

      for (int i=0;i < lista.length;i++)
      {
         String encontrado = diccionario.get(lista[i]);

         if (encontrado != null) {

         if ((objeto.equals("vacio"))&& encontrado.equals("objeto")){
            objeto = lista[i];
            /*Podemos tener objetos sin ubicación porque son únicos en la casa*/
            if (objeto.equals("nevera")||objeto.equals("cafetera"))
               lugar="casa";
         }
         else if ((lugar.equals("vacio"))&&encontrado.equals("lugar")) {
            lugar = lista[i];
         }
         else if ((accion.equals("vacio"))&&encontrado.equals("accion")) {
            accion = lista[i];
         }
         else if ((atributo.equals("vacio"))&&encontrado.equals("atributo")){
            atributo = lista[i];
         }

         }

      }

      result = objeto.toString()+" "+lugar.toString()+" "+atributo.toString()+" "+accion.toString();

   return result;
   }
}
```

House_object

```
package com.example.app;

import java.util.List;

public class House_object {

   private String id;
   private String description;
   private String functionality;
   private List<String> actions;


   public House_object () {

   }

   public House_object (String id, String description, String functionality){

      this.id = id;
      this.description = description;
      this.functionality = functionality;

   }

   public String getId() {
      return id;
   }

   public void setId(String id) {
      this.id = id;
   }

   public String getDescription() {
      return description;
   }

   public void setDescription(String description) {
      this.description = description;
   }

   public String getFunctionality() {
      return functionality;
   }

   public void setFunctionality(String functionality) {
      this.functionality = functionality;
   }

   public List<String> getActions() {
      return actions;
   }

   public void setActions(List<String> actions) {
      this.actions = actions;
   }



}
```

MainActivity

```
package com.example.app;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.widget.Toast;

public class MainActivity extends Activity {

   public Context context;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      context = getApplicationContext();
      Intent servicio = new Intent(this, House_service.class);

      if(startService(servicio)==null)
      {
         Toast.makeText(context, "No se ha podido iniciar el servicio", 1).show();
      }
      else
      {
         this.finish();
      }
   }
}
```

House_service

```
package com.example.app;


import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.media.AudioManager;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.speech.RecognitionListener;
import android.speech.RecognizerIntent;
import android.speech.SpeechRecognizer;
import android.speech.tts.TextToSpeech;
import android.speech.tts.UtteranceProgressListener;
import android.widget.Toast;
import org.json.JSONException;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Hashtable;
import java.util.List;
import java.util.Locale;

public class House_service extends Service {


    //Variables referentes al servicio
    protected AudioManager AudioManager; /*Lo utilizaremos para poder controlar el volumen de audio y
    mutear el beep del reconocedor de voz*/

    protected SpeechRecognizer SpeechRecognizer;
    protected Intent SpeechRecognizerIntent;
    private TextToSpeech ttobj;
    protected ArrayList<String> words;
    private boolean iAmSpeaking; /*Lo utilizaremos para detectar que queremos enviar órdenes a la casa*/


    //Variable referentes al reconocimiento
    private List<House_object> objects_list; /*Lista de objetos House_object*/
    private Hashtable<String,String> attributes_list; /*Lista de objetos,lugars y atributos*/
    private Hashtable<String,String> actions_list; /*Lista de acciones y su orden correspondiente*/
    private boolean bad_order = false; /*Lo utilizaremos para saber si ha habido algún error en el
    procesamiento de la orden pronunciada*/
    private String object = null; /*Variable de la clase que se utilizará en varios métodos para definir el objeto de
    la casa con el que vamos a interacturar*/
    private String action = null; /*Variable de la clase que se utilizará en varios métodos para definir la acción
    que se realizará sobre el objeto a interactuar*/
    private String order = null; /*Necesitamos saber de qué orden se trata, si es una consulta hacemos un get y
    si es una accion un put*/
    private String aux_description = null; /*Variable de la clase que se utilizará para localizar el House_object y
    para informar al usuario*/
    private Context context; /*Se utiliza en los Toast*/
    private int duration = Toast.LENGTH_SHORT; /*Se utiliza en los Toast*/
    private HashMap<String,String> ttsParams; /*Se utilizara como parámetro en el para configurar el
    TextToSpeech*/

    /*Estas variables nos van a permitir controlar si falta algún elemento en la orden. Se emplearán en los
    metodos commands y proceed*/
    private String running_object = null;
    private String running_action = null;
    private String running_place = null;
    private String running_attribute = null;
    private String error = null;
```

```
    @Override
    public void onCreate()
    {
        super.onCreate();

        //Inicializaciones referentes al servicio
        AudioManager = (AudioManager) getSystemService(Context.AUDIO_SERVICE);
        SpeechRecognizer = SpeechRecognizer.createSpeechRecognizer(this);
        SpeechRecognizer.setRecognitionListener(new SpeechRecognitionListener());
        SpeechRecognizerIntent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);

SpeechRecognizerIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,RecognizerIntent.LANG
UAGE_MODEL_FREE_FORM);

SpeechRecognizerIntent.putExtra(RecognizerIntent.EXTRA_CALLING_PACKAGE,this.getPackageName());
        SpeechRecognizer.startListening(SpeechRecognizerIntent);

        AudioManager.setStreamVolume(AudioManager.STREAM_MUSIC, 0, 0); /*Silenciamos el volumen del
        dispositivo para evitar que se escuche el beep del reconocedor de voz*/
        iAmSpeaking = false; /*Lo inicializamos como false ya que hasta que no se pronuncie la frase mágica no
        comenzaremos a tratar ordenes*/
        context = getApplicationContext();
        ttsParams = new HashMap<String, String>();


        //Inicializaciones referentes al reconocimiento

        /*Inicializamos las variable con el valor vacío ya que no hay ninguna orden en proceso. Cuando se
        comience a procesar una orden las variable irán obteniendo su valor hasta conseguir que las cuatro estén
        definidas y se pueda determinar el objeto y la acción a realizar*/
        running_object = "vacio";
        running_place = "vacio";
        running_attribute = "vacio";
        running_action = "vacio";

        //Cargamos la lista de objetos de la casa
        InputStream fichero = this.getResources().openRawResource(R.raw.data);
        XML_parser parseador = new XML_parser();
        objects_list = parseador.parseando(fichero);

        //Cargamos la lista de atributos
        InputStream fichero2 = this.getResources().openRawResource(R.raw.attributes);
        XML_to_Hash parseador2 = new XML_to_Hash();
        attributes_list = parseador2.parseando(fichero2,"atributos");

        //Cargamos la lista de ordenes asociada a cada accion
        InputStream fichero3 = this.getResources().openRawResource(R.raw.attributes);
        actions_list = parseador2.parseando(fichero3,"acciones");

    }

    @Override
    public IBinder onBind(Intent arg0) {

        return null;
    }

    @Override
    public void onDestroy(){

        super.onDestroy();
    }

    /*Lo utilizaremos para reiniciar el reconocedor en caso de error o después de haber tratado una orden*/
    public void startTask (boolean destroy_SpeechReconigzer){

        /*Si venimos de un error en el reconocedor o de una orden no procesada porque la frase de activación no
        se ha pronunciado, se invoca el método con true como parámetro para destruir el objeto reconocedor
        antes de instanciar uno nuevo*/
        if (destroy_SpeechReconigzer)
            SpeechRecognizer.destroy();

        SpeechRecognizer = SpeechRecognizer.createSpeechRecognizer(this);
        SpeechRecognizer.setRecognitionListener(new SpeechRecognitionListener());
        SpeechRecognizer.startListening(SpeechRecognizerIntent);
```

```
}

/*Definimos como se comportará nuestro reconocedor*/
protected class SpeechRecognitionListener implements RecognitionListener{

   @Override
   public void onBeginningOfSpeech(){

   }

   @Override
   public void onBufferReceived(byte[] buffer){

   }

   @Override
   public void onEndOfSpeech(){

   }

   @Override
   public void onError(int error){

   /*Si se produce un error hay que relanzar el reconocedor para asegurarnos que el servicio no deje de estar
   a la escucha*/
      startTask(true);
   }

   @Override
   public void onEvent(int eventType, Bundle params){

   }

   @Override
   public void onPartialResults(Bundle partialResults){

   }

   @Override
   public void onReadyForSpeech(Bundle params){

   }

   @Override
   public void onResults(Bundle results)
   {
      /*El reconocedor ha procesado la voz, guardamos el resultado para poder procesarlo*/
      words = results.getStringArrayList(SpeechRecognizer.RESULTS_RECOGNITION);


      assert words != null;
      if (words.get(0) != null){

          if (words.get(0).equals("Hola madre")){ /*Si se pronuncia la frase de activación el servicio comienza
          a procesar ordenes*/
             awaken();
          }
           else if (iAmSpeaking){ /*Si ya se ha pronunciado la frase de activación el valor de la variable será
         verdadero*/

             if (words.get(0).equals("adiós madre")){ /*Esta frase la utiliza el usuario para indicar al servicio
             que deje de procesar ordenes*/
                sleep();
             }
             else {
                proceed(words.get(0)); //Se procesa la frase pronunciada
             }
          }
          else {
          startTask(true); /*Nos aseguramos de que el reconocedor no deja de estar a la escucha   reiniciándolo
          a través del método startTask*/
          }
      }
      else {

      convertTospeech("No te he entendido"); /*Se le comunica al usuario que no se le ha entendido. No
      debería nunca activarse esta opción*/
```

```
        }

    }

    @Override
    public void onRmsChanged(float rmsdB)
    {

    }

}

/*Manejador de mensajes. Cuando el objeto ttobj (TextToSpeech) del método convertTospeech termine de
hablar enviará el mensaje a este manejador*/
private Handler _handler = new Handler(){
    @Override
    public void handleMessage(Message msg){

    bad_order=false;/*Si venimos de una orden errónea (comprobado en el método commands) debemos
    volver a la variable a su estado inicial*/
    AudioManager.setStreamVolume(AudioManager.STREAM_MUSIC, 0, 0); /*Volvemos a mutear para
    evitar de nuevo que se escuche el beep del reconocedor*/

    /*Reiniciamos el SpeechRecognizer pero como ya hemos destruido la anterior instancia (en el método
    convertTospeech pasamos como parámetro  false porque ya no es necesario destruirlo en el método
    startTask*/
    startTask(false);
    }
};

private void convertTospeech (final String text) {

    /*Mientras la aplicación habla no se debe estar reconociendo la voz, por lo que se destruye el objeto
    SpeechRecognizer*/
    SpeechRecognizer.destroy();
    /*Se fija el volumen del audio para que se escuche a la aplicación. Como el objeto SpeechRecognizer se ha
    destruido ya no se escuchará el beep*/
    AudioManager.setStreamVolume(AudioManager.STREAM_MUSIC, 7, 0);

    ttobj=new TextToSpeech(context, new TextToSpeech.OnInitListener() {

        @Override
        public void onInit(int status) {

            /*Si se ha podido crear el objeto*/
            if(status == TextToSpeech.SUCCESS){
                ttobj.setLanguage(Locale.getDefault());
                ttobj.setOnUtteranceProgressListener(new UtteranceProgressListener() {
                    @Override
                    public void onStart(String s) {

                    }

                    @Override
                    public void onDone(String s) {

                        /* Cuando termina de hablar se envía un mensaje al hilo principal, siendo el mensaje recibido
                        por el manejador.  Así conseguimos que el servicio no este lanzando por debajo nuevas
                        instancias del reconocedor mientras la aplicación está hablando.  Solo cuando el ttobj.speak()
                        que se ejecuta en otro hilo diferente del servicio, termine de hablar, el hilo principal continuará
                        ejecutando código*/
                        _handler.sendMessage(_handler.obtainMessage());

                    }

                    @Override
                    public void onError(String s) {

                    }
                });
                /*El objeto ttobj debe tener un KEY_PARAM_UTTRANCE, en este caso "done", para poder
                emplear UtteranceProgressListener().*/
                ttsParams.put(TextToSpeech.Engine.KEY_PARAM_UTTERANCE_ID, "done");
                ttobj.speak(text, TextToSpeech.QUEUE_ADD,ttsParams);

            }
            else {
```

```
            Toast.makeText(context, "Text to Speech no instalado", Toast.LENGTH_SHORT).show();

          }
        }

    });


}

/*La frase mágica ha sido pronunciada y el servicio comienza a procesar ordenes*/
public void awaken () {

    iAmSpeaking = true;
    convertTospeech("Hola Ripley, ¿en que puedo ayudarte?");


}

/*Procesamos las ordenes*/
public void proceed (String text) {

    /*La clase Text_reognizer va a procesar la frase pronunciada, intentando reconocer palabras que se hallan
    definido en  attributes.xml como objeto, lugar, acción, o atributo. Devolviendo el resultado en un string
    con los valores correspondientes a  "objeto lugar atributo accion" o en su defecto la palabra "vacio" como
    valor si no se ha podido reconocer alguno de los tokens*/
    Text_recognizer new_recognizer = new Text_recognizer();
    String resultado = new_recognizer.recognizer(text,
    attributes_list,running_object,running_place,running_action,running_attribute);

     /*Llamamos a un método que intentará a partir del valor de los tokens recibidos en resultado,
     determinar si es posible de que  objeto de la casa se trata y que acción se quiere realizar, dando valor a las
     variable order y action que se emplean a continuación*/
    commands(resultado);

    /*si hemos procesado la orden correctamente*/
    if (!bad_order) {

       /*Devolvemos las variables a su valor original pues ya se ha determinado el objeto y la acción.
      Cuando esta orden sea realizado se le enviara el feedback al usuario, y estas variable volverán a tomar
      valores durante el proceso de reconocimiento de tokens del servicio*/
       running_object = "vacio";
       running_place = "vacio";
       running_action = "vacio";
       running_attribute = "vacio";


      if
      (order.equals("consultar")||order.equals("saber")||order.equals("dime")||order.equals("encendida")||or
      der.equals("encendido")){
          Peticion_GET_REST task = new Peticion_GET_REST();
          task.execute();
       }
      /*A cada acción le corresponde una orden, que es la que realmente se envía al objeto, y aparte de ser una
      acción valida deben de poder ejecutarse sobre el objeto seleccionado. Esto lo comprobamos con la
      variable action cuyo valor ha sido determinado en el método commands*/
       else if (!action.equals("error")){
          Toast.makeText(context,action, duration).show();
          Peticion_PUT_REST task = new Peticion_PUT_REST();
          task.execute();
       }
       else {
      /*Error es también una variable cuyo contenido se va a determinar en el método commands y que nos
      va permitir realizar feedback al usuario
          informándole en caso de que se halla producido algún error*/
          convertTospeech(error);
       }

    }
    else {
       convertTospeech(error);
    }


}
```

```
/*Se ha pronunciado la frase de desactivación, iAmSpeaking se pone a false, se informa al usuario que se le
ha entendido con una frase despedida, estando el servicio a partir de ahora únicamente a la escucha sin
procesar las frases pronunciadas*/
public void sleep () {

   iAmSpeaking = false;
   convertTospeech("Adios, Ripley");
}

/*Este metodo trata el texto obtenido por Text_recognizer, obteniendo si es posible el identificador del
objeto y la orden que se le debe enviar.*/
public void commands (String text) {

    /*text contiene los valores de objeto lugar atributo acción que ha podido identificar la clase
   Text_recognizer instanciada en el método proceed*/
   String[] lista = text.split(" ");

    /*Vamos a buscar un objeto casa que disponga de una descripción que coincida con los valores recibidos.
   Inicializamos la variable a null  para comprobar más adelante si la búsqueda ha tenido éxito
   comprobando que el valor de la variable ha dejado de ser null*/
   object = null;

   /*Todavía no hay un error*/
   error ="";

   /*Estas variable toman los valores recibidos en el String text. Estos valores no se perderán en sucesivas
   llamadas a este método hasta conseguir identificar el objeto y la acción. La primera vez que se invoca este
   método el usuario puede no haber pronunciado todos los elementos necesarios, así que se le indicará que
   añada los que falten*/
   running_object = lista[0];
   running_place = lista[1];
   running_attribute = lista[2];
   running_action = lista[3];

   /*Disponemos de valores para objeto, lugar y acción*/
   if
  ((!running_object.equals("vacio"))&&(!running_place.equals("vacio"))&&(!running_action.equals("vacio"
  ))) {

        /*Si además tenemos atributo ya podemos construir la descripción*/
        if (!running_attribute.equals("vacio")){

           aux_description = running_object+" "+running_place+" "+running_attribute;
        }
        /*Si no hay atributo y el lugar es casa. Se trata de unos objetos especiales únicos en la casa por lo que
        no tiene ni atributo ni lugar en su descripción
        construyendola solo con el valor de objeto*/
        else if (running_place.equals("casa")){

           aux_description = running_object;

        }
        /*Sino puede tratarse de un objeto con solo valores de objeto y lugar en su descripción*/
        else {

           aux_description = running_object+" "+running_place;

        }

      /*Cargamos en la variable order el valor de la accion identificada y recibida como parametro*/
      order = running_action;
      List<String> object_actions = new ArrayList<String>();


       for (House_object new_object : objects_list) {
       /*Si encuentro un objeto en lista de objetos que he obtenido del xml, con una descripción que coincida
       con el valor de aux_description  que he procesado más arriba ej: luz cocina central, guardo su id y las
       acciones que se pueden realizar sobre él para poder interactuar con él*/
          if (new_object.getDescription().equals(aux_description)) {
             object = new_object.getId();
             object_actions = new_object.getActions();

          }
       }
```

```
            /*Si no hemos encontrado el objeto en la lista, es porque se trata de un objeto con atributo y este no se
            ha indicado*/
            if (object == null){

                bad_order = true;
                if (running_attribute.equals("vacio")){
                error = "¿Con que "+running_object+" "+running_place+" quieres interactuar?";
                /*En caso contrario el objeto no existe en la casa*/
                else{
                    running_object = "vacio";
                    running_place = "vacio";
                    running_action = "vacio";
                    running_attribute = "vacio";
                    error = "Lo siento Ripley.Pero ese objeto no existe en la casa";
                }


            }
            else {
            /*Obtengo la accion asociada a la orden*/
            String aux = actions_list.get(order);

            /*Compruebo si el objeto soporta esa acción*/
            if (object_actions.contains(aux)){

                action = aux;


            }
            else {

                action = "error";
                error = "Orden no aplicable a ese objeto";
            }


          }
        }

        else {
            /*Se le solicita al usuario los objetos que falten*/
            bad_order = true;
            Toast.makeText(context,"Faltan objetos", duration).show();
            if (running_object.equals("vacio")){

                error = "Ok Ripley, pero no me has dicho el objeto";
            }
            else if (running_place.equals("vacio")){

                error = "El objeto, "+running_object+".¿En que lugar de la casa está situado?";

            }
            else if (running_action.equals("vacio")){

                error = "¿Que quieres que haga con "+running_object+" "+running_place+"?";

            }

        }
    }

    /*Peticion PUT, en la que trabajaremos con las variables de la clase inicializadas en el método comands
    (object y action).  Se ejecuta en una tarea asincrona*/
    private class Peticion_PUT_REST extends AsyncTask<Void, Void, String> {

        String texto = "error";
        @Override
        protected String doInBackground(Void... params) {

            try{
                URL url = new URL("http://joaono.ignorelist.com:8182/devFunc/"+object);
                HttpURLConnection conexion = (HttpURLConnection) url.openConnection();
                conexion.setRequestMethod("PUT");
                conexion.setDoOutput(true);
                conexion.setRequestProperty("Content-Type", "application/json");
                conexion.setRequestProperty("Accept", "application/json");
                String input = "{action:"+action+"}";
```

```
            OutputStream body = conexion.getOutputStream();
            body.write(input.getBytes());
            body.flush();
            conexion.getInputStream();
            texto = "ok";
          }
          catch (Exception e){
            Toast.makeText(context,"Error de comunicación"+e.getMessage(), duration).show();
          }
          return texto;
        }

        @Override
        protected void onPostExecute(String text) {
          if (text.equals("error")){
            convertTospeech("No he podido ejecutar la orden");
          }
          else if (text.equals("ok")){
            convertTospeech("Orden realizada correctamente");
          }
        }
      }


      /*Peticion GET, en la que trabajaremos con las variables de la clase inicializadas en el metodo comands
      (object y aux_description).
      Se ejecuta en una tarea asincrona*/
      private class Peticion_GET_REST extends AsyncTask<Void, Void, String> {

        @Override
        protected String doInBackground(Void... params) {

          String texto = "";


          try{
            URL urlget = new URL("http://joaono.ignorelist.com:8182/devFunc/"+object);
            HttpURLConnection conexion = (HttpURLConnection) urlget.openConnection();
            conexion.setRequestMethod("GET");
            conexion.setRequestProperty("Accept", "application/json");
            int status;
            status = conexion.getResponseCode();
            if (status == HttpURLConnection.HTTP_OK) {
              BufferedReader reader = new BufferedReader(new
              InputStreamReader(conexion.getInputStream()));
              StringBuilder buffer = new StringBuilder();
              String s;
              while((s=reader.readLine()) != null){
                buffer.append(s);
              }
              JSON_getparser get_state = new JSON_getparser();
              texto = get_state.parseando(buffer.toString());

            }
            else {
              Toast.makeText(context,"Error en la conexion", duration).show();
            }
          }
          catch (IOException e){
            Toast.makeText(context,"Error peticion get"+e.getMessage(), duration).show();
          } catch (JSONException e) {
            e.printStackTrace();
          }
          return texto;
        }

        @Override
        protected void onPostExecute(String text) {
          convertTospeech(aux_description + " " + text);
        }



      }

    }
```