UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Gestión desde Android de un simulador de control domótico con Raspberry Pi

Presentado por Mario Juez Castrillo
en Universidad de Burgos — 6 de junio de 2018
Tutor: César Represa Pérez

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. César Represa Pérez, profesor del departamento de Ingeniería Electromecánica, área de Tecnología Electrónica.

Expone:

Que el alumno D. Mario Juez Castrillo, con DNI 45573856T, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado Gestión desde Android de un simulador de control domótico con Raspberry Pi.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 6 de junio de 2018

Vº. Bº. del Tutor:

D. César Represa Pérez

Resumen

Se puede decir que la domótica y los edificios inteligentes fueron los primeros avances tecnológicos que empezaron a introducirse en nuestras vidas. Ahora, es el uso de los smartphones el que se ha extendido de tal manera que comienza a ser un elemento indispensable. Este proyecto surgió de la idea de unir estos dos mundos tecnológicos, es decir, controlar un sistema domótico desde nuestro teléfono móvil. En este proyecto se presenta un control domótico simulado sobre una Raspberry Pi y que puede ser controlado desde una aplicación Android. Con dicha aplicación se podrán visualizar y modificar el número de estancias de una vivienda así como los elementos controlables que hay ellas. Además, la aplicación admite el control de la estación domótica por múltiples usuarios. El desarrollo del simulador en la Raspberry Pi se ha llevado a cabo con Python 3 en *Jetbrains Pycharm* y la aplicación con Android en *Android Studio*.

Descriptores

Raspberry Pi, Simulador domótico, Android, Python 3.

Abstract

It can be said that home automation and intelligent buildings were the first technological advances that began to enter our lives. Now, it is the use of smartphones that has spread in such a way that it begins to be an indispensable element. This project arose from the idea of uniting these two technological worlds, that is, controlling a domotic system from our mobile phone. This project presents a simulated domotic control over a Raspberry Pi and can be controlled from an Android application. With this application you can view and modify the number of rooms in a house as well as the controllable elements that are there. In addition, the application supports control of the home automation station by multiple users. The development of the simulator in the Raspberry Pi has been carried out with Python 3 in *Jetbrains Pycharm* and the Android application in *Android Studio*.

Keywords

Raspberry Pi, Domotic simulator, Android, Python 3.

Índice general

Índice de figuras

Índice de tablas

Introducción

Hoy en día la informática se está introduciendo en nuestra vida cotidiana sin darnos mucha cuenta. Todos los días usamos elementos cotidianos que poco a poco se van informatizando o más bien programando para realizar mejor sus tareas y de manera más sencilla. Debido a este cambio, nació la domótica, que es el conjunto de tecnologías aplicadas al control y la automatización inteligente de la vivienda, que permite una gestión eficiente del uso de la energía, que aporta seguridad y confort, además de comunicación entre el usuario y el sistema [4].
Este es un tema que siempre me ha llamado la atención, el cómo es posible interactuar con objetos a distancia sin que nosotros seamos capaces de poder ver un medio de conexión. Nosotros no somos capaces de ver o notar el bluetooth, o incluso el wifi, pero ahí están, rodeándonos sin que nos demos cuenta.
A raíz de todo esto, está poniéndose de moda un nuevo concepto llamado Internet de las Cosas o IoT. El internet de las cosas se refiere a la interconexión digital de objetos cotidianos con Internet, y que actualmente se usa con una denotación de conexión avanzada de dispositivos, sistemas y servicios que va más allá del tradicional M2M (máquina a máquina) y que cubre una amplia variedad de protocolos, dominios y aplicaciones [8].
Esto me motivó a intentar conseguir centralizar todos los elementos cotidianos en nuestra mano, es decir, en nuestro smartphone. Por esto, la idea principal de mi trabajo ha sido trabajar sobre estos temas actuales e intentar crear paso a paso un pequeño simulador que nos permita visualizar como podría funcionar una casa inteligente en la realidad y con qué tecnología se podría conseguir.

Objetivos del proyecto

En este apartado vamos a diferenciar y explicar los objetivos generales del proyecto y los objetivos personales.

2.1. Objetivos generales

- Realizar una aplicación en Android que permita manejar un sistema de control domótico.
- Simular una centralita de control domótico en una RaspberryPi.
- Conectar y desconectar la aplicación en Android del simulador domótico.
- Permitir recrear tu casa añadiendo el número de habitaciones y elementos que se desee.
- Permitir que la aplicación cambie el estado de los elementos de cada habitación.
- Permitir modificar los nombres de las habitaciones y los elementos para que sean más identificables.
- Visualizar el estado de nuestra casa desde una interfaz en la Raspberry Pi.

2.2. Objetivos personales

- Mejorar en el desarrollo de aplicaciones Android y asentar los conocimientos de los que ya disponía.
- Mejorar en el desarrollo en Python y asentar los conocimientos de los que ya disponía.
- Aprender el desarrollo de interfaces gráficas mediante tkinter para Python.
- Mejorar el conocimientos sobre los sockets y las conexiones TCP/IP.

Conceptos teóricos

Para el correcto entendimiento de este trabajo va a ser necesario explicar o aclarar varios términos de los que se va a hablar.

3.1. Domótica

La domótica [4] es el conjunto de tecnologías aplicadas al control y la automatización inteligente de la vivienda, que permite una gestión eficiente del uso de la energía, que aporta seguridad y confort, además de comunicación entre el usuario y el sistema.
Este concepto es importante para este proyecto porque estamos tratando de imitar una vivienda inteligente, aunque solo tratemos la parte de su iluminación y la comunicación entre el usuario y el sistema.

3.2. Internet de las cosas (IoT)

Internet de las cosas (Internet of Things) [8] es un concepto que se refiere a la interconexión digital de objetos cotidianos con Internet. Alternativamente, Internet de las cosas es la conexión de Internet con más çosas y objetos"que personas.
Quería aclarar este concepto, porque en este trabajo a pesar de usar un servidor como centralita que maneja tu iluminación, actualmente muchas empresas están creando bombillas que contienen su propia conexión a Internet y que pueden ser manejadas a través de una aplicación de móvil.

3.3. Android

Android [6] es un sistema operativo basado en un núcleo Linux. Fue diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes, tabletas y también para relojes inteligentes, televisores y automóviles. La aplicación a desarrollar en este trabajo está enfocada a este sistema operativo, ya que actualmente es el sistema operativo móvil más utilizado del mundo.

3.4. Python

Python [9] es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.
Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.
De todo lo anterior, que es una breve explicación de Python, nos interesa la idea de que sea *multiplataforma*, ya que nos ofrece la posibilidad de ser ejecutado en el sistema operativo Raspbian de una Raspberry Pi, o por ejemplo en un sistema operativo Windows de un ordenador de sobremesa.

3.5. Interfaz gráfica (GUI)

La interfaz gráfica de usuario, conocida también como GUI (del inglés graphical user interface) [7], es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso, consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computador.
En este trabajo, tanto la parte de la aplicación móvil como la de la Raspberry Pi, contarán con una interfaz gráfica. De hecho, todas las aplicaciones Android trabajan de forma gráfica, ya que el usuario no ve su código mientras las maneja. En cambio, la parte de Python 3 en la Raspberry no tiene por qué tener interfaz gráfica y puede ser manejada mediante una terminal. Pero para hacer más cómodo y visible el estado de la casa al usuario, he utilizado la librería *Tkinter* de Python para crear una interfaz gráfica.

3.6. Tkinter

Tkinter [5] forma parte de la biblioteca gráfica Tcl/Tk para el lenguaje de programación Python. Se considera estándar de la interfaz de usuario Python, debido a que viene con su instalación y no es necesario la instalación de librerías externas. Y a pesar de haber otras alternativas como wxPython, PyQt o PySide y PyGTK, he preferido usar esta.

Técnicas y herramientas

4.1. Técnicas

En cuanto al trabajo he seguido una metodología o técnica *Top-down*, porque es la que más se adaptaba a mi forma de trabajar y con la que más cómodo me sentía, por ejemplo, respecto a una metodología *SCRUM*. He intentado llevar un seguimiento de mi trabajo en la plataforma *GitHub*, pero el seguimiento real lo he realizado por escrito en una pizarra día a día y subiendo a GitHub solo los cambios más importantes y completamente funcionales.

4.2. Herramientas

En este apartado voy a explicar brevemente cuáles son las herramientas que he utilizado para realización del trabajo.

4.3. Lenguajes de programación

Python 3

Para la parte de la centralita domótica en Raspberry opté por Python 3 tras pensar sus ventajas respecto a otros. Los aspectos más relevantes para mi elección fueron que viene instalado por defecto con el sistema Raspbian, que es un lenguaje que tiene una sintaxis sencilla para su programación y por último, que es multiplaforma, que eso siempre viene bien en caso de migrar el código a otro sistema.
Estuve también optando por realizar esta parte con Java y la interfaz gráfica con JavaFX, pero uno de los puntos fuertes que tenía Python 3 y Java no, es

que viene preinstalado en el sistema y teniendo en cuenta que pretendemos que los usuarios no tengan que ser expertos en la materia, quería que fuese lo más sencillo posible para su instalación y ejecución.

Una vez decidido el lenguaje de programación, la duda estaba en qué tecnología se usaría para la representación de la interfaz gráfica. Buscando por internet decidí usar la librería *Tkinter* porque ya viene incluida en la mayoría de las instalaciones de Python. Estuve mirando como realizar interfaces gráficas con PyQt5, pero no está soportado en versiones superiores a Python 3.5 y actualmente en la Raspberry Pi estamos usando la versión 3.5.3. Además, PyQt5 funciona creando un fichero .pyw a parte de nuestro fichero .py, lo cual me parecía crear una estructura compleja e innecesaria de directorios, teniendo otras tecnologías más simples.

Para la programación de la parte de Raspberry Pi en Python 3 he utilizado la herramienta *Jetbrains Pycharm*.

Android

Para la programación de la aplicación que se usará en el smartphone, he decidido usar *Android* frente *IOS* porque la mayoría de los smartphone actualmente usan *Android*. Además, ya tenía conocimientos básicos sobre la tecnología porque ya había realizado pequeñas aplicaciones antes.

Este proyecto específico está programado con el nivel 23 de la API x86 enfocado para sistemas Lollypop 5.0 o superiores.

Para la programación de la aplicación de este proyecto he utilizado la herramienta *Android Studio* de la que hablaré a continuación.

4.4. Herramientas de desarrollo de software

JetBrains Pycharm (Python 3)

Jetbrains Pycharm es un entorno de desarrollo o *IDE* orientado hacía Python. He optado por esta herramienta y no por otra como por ejemplo *Geany*, porque ya tenía experiencia en ella. Además de ayudar mucho con su autocompletado de código, nos ofrece muchas posibilidades con la instalación de plugíns. Estos plugins nos pueden servir, por ejemplo, para ayudarnos a ver el recubrimiento de código de los test que hemos realizado.

Android Studio (Android)

Android Studio es otro entorno de desarrollo o *IDE* enfocado al desarrollo de aplicaciones en Android. Además de ayudar mucho con su autocompletado como la anterior, también tenemos la opción de instalar plugins. Aunque lo que más me gustaría destacar de esta herramienta es la posibilidad de tener un AVD o *Android Virtual Device*. Un **AVD** se podría entender como un emulador, es decir, podemos elegir entre diferentes tipos de dispositivos, como *smartphone* o *tablets*, y elegir la versión que deseas que se carge en ellos. Gracias a esto, tenemos la posibilidad de hacer funcionar nuestro código sobre un sistema de Android.

Esta parte de la herramienta es la que más uso he dado en el desarrollo de software, ya sea para visualizar la ejecución de tu aplicación, como la ejecución de test instrumentales sobre el sistema Android. Para toda la realización del proyecto he utilizado un emulador del dispositivo **Nexus 5X** con un versión de sistema **Android Oreo 8.1** y con una API nivel 27.

4.5. Herramientas de control de versiones

GitHub

GitHub es una herramienta de control de versiones que nos ayuda a tener disponible nuestro código y ficheros en el repositorio en todo momento. Desde este repositorio podemos acceder a todos los ficheros subidos, y además en cada una de las versiones subidas, lo que nos ayuda a retroceder a una versión anterior en caso de pérdida o fallo. Para realizar de manera más sencilla la visualización de los cambios en el repositorio de trabajo y los tareas pendientes, he utilizado la herramienta de escritorio ***GitHub Desktop***. Con este herramienta, puedo seleccionar fácilmente que archivos deseo subir y realizar tareas básicas de *Git* como **Pull** o **Push**.

ZenHub

ZenHub es una herramienta para la gestión de las tareas de un proyecto y que está totalmente integrada con **GitHub**. Esta herramienta no está disponible desde la herramienta **GitHub Desktop**, pero si desde la web. Esta ha sido una de las herramientas para la planificación temporal y seguimiento del proyecto. Aunque me gustaría aclarar que las tareas diarias y seguimiento de ellas las he realizado a mano sobre un pizarra en la que podía hacer diagramas y dibujos que me han ayudado a su solución.

4.6. Herramientas de documentación

LaTeX

Latex es una herramienta enfocada a la creación de documentos escritos que presenten una alta calidad tipográfica. La memoria y anexos de este proyecto han sido escritos con esta herramienta. Gracias a ella y a la plantilla proporcionada por la universidad solo hay que preocuparse de rellenar su contenido y nos olvidaremos de la problemática de su formateo. Esta herramienta frente a otros editores de texto como *Word* tiene un período de adaptación más alto, pero que conlleva un mejor resultado final.

Aspectos relevantes del desarrollo del proyecto

El inicio de este proyecto comenzó el septiembre, pero he creído conveniente rehacerlo de cero varias veces para mejorar ciertos aspectos desde el principio, darle un nuevo enfoque y no arrastrar errores iniciales. Por ello en este apartado voy a hablar sobre los aspectos relevantes de las dos partes principales del proyecto: *Raspberry Pi* y *Aplicación Android*.

5.1. Raspberry Pi

En cuanto a esta parte vamos a diferencias tres partes importantes del proyecto: *Interfaz*, *Servidor Python* y *Base de datos*.

Interfaz

La interfaz en la Raspberry Pi 3 está desarrollada mediante la librería *Tkinter*. Esta librería viene incorporada con Python 3 y es bastante sencilla de utilizar, pero el problema que tiene con respecto a otras tecnologías, es que sus elementos y posibilidades son limitados. Pero tras investigar, encontré que a partir de la versión 8.5 de *Tkinter* se introdujo un módulo llamado **ttk** que nos permite un nuevo abanico de posibilidades. Entre dichas posibilidades encontramos widgets como *ComboBox*, *Notebook*, *Progressbar*, etc.

Leyendo sobre su utilización, comencé a trabajar con un ComboBox rellenándolo con las estancias y teniendo la posibilidad de eliminar la estancia seleccionada en él. Una vez avanzado el proyecto, y tras cambiar los objetivos para que el servidor no tuviese la capacidad de realizar cambios de manera

independiente a la aplicación, me metí con la *Notebook*. Esta ha sido la mejor manera de representar nuestra casa, de manera que cada *pestaña* o *tab* será cada una de las estancias, y dentro del contenido de cada *pestaña* se representa la iluminación. Su creación, modificación y eliminación es bastante sencilla, el único problema se encuentra en la inserción de objetos en el contenido de la pestaña. El problema es que el contenido de la pestaña y la pestaña son cosas diferentes. El contenido de la pestaña es un *Frame* que se acopla a la pestaña en su creación, de tal manera que cuando tratamos de insertar elementos en ella no necesitamos referenciar a la pestaña, sino al *Frame*. Por ello, me ví obligado a mantener una estructura que contiene todos los *Frames* de las pestañas activas, de tal manera que se van añadiendo y eliminando según lo vayando haciendo las pestañas correspondientes.

Un problema similar me surgió para conseguir poder modificar las *labels* o *etiquetas* que representan las bombillas. Estas *etiquetas* están representadas dentro de dichos *Frames*, pero no son accesibles a través de él. Por tanto, me ví de nuevo obligado a crear y mantener una nueva estructura que contiene todas las *etiquetas* o bombillas que tenemos actualmente. Esta segunda estructura es diferente de la primera, ya que la primera era una simple lista y esta es un tabla *hash*. En esta estructura la clave será el nombre de la pestaña, es decir, el de la estancia y el valor, será una lista con los identificadores de las etiquetas de esa pestaña para poder acceder a ellas.

Y por último, hay que tener en cuenta que para la representación de la interfaz, el objeto que la crea se mantiene mediante un bucle infinito. Por tanto, no se deberían realizar inicializaciones ni cambios dentro del código de dicho bucle. Respecto a la interfaz estos son los aspectos más relevantes ya que se intentó simplificar lo más posible su complejidad.

Servidor Python

El servidor Python lo hice con la finalidad de que siempre estuviese escuchando los cambios que le llegan de la aplicación. Lo primero que me sucedió, es que no podía crearlo a la vez que la interfaz, debido a que la interfaz se encuentra en un bucle infinito. Debido a esto, se me ocurrió crearlo al inicio, pero en un *Thread* o *Hilo* independiente.

La otra parte complicada del servidor, ha sido lo que llamo el tratamiento de datos. Por tratamiento de datos me refiero a la interpretación de los mensajes de los clientes de tal manera que produzcan cambios reales en nuestra interfaz. La interpretación de mensajes de realiza de la siguiente manera:

- **Añadir estancia:** + (Añadir) **E** (Estancia) **nombre** (Nombre de la estancia) Ej.: +EHabitación 1
- **Añadir una bombilla:** + (Añadir) **B** (Bombilla) **número** (Posición de la estancia) **nombre** (Nombre) Ej.: +B2Bombilla1
- **Modificar estancia:** * (Modificar) **E** (Estancia) **número** (Posición de la estancia) **nombre** (Nombre al que queremos cambiar) Ej.: *E1Habitación 2
- **Modificar bombilla:** * (Modificar) **B** (Bombilla) **número** (Posición de la estancia) **número** (Posición de la bombilla) **nombre** (Nombre al que queremos cambiar) Ej.: *B02Bombilla Azul
- **Eliminar estancia:** - (Eliminar) **E** (Estancia) **número** (Posición de la estancia) Ej.: -E1
- **Eliminar bombilla:** - (Eliminar) **B** (Bombilla) **número** (Posición de la estancia) **número** (Posición de la bombilla) Ej.: -B12
- **Cambiar estado bombilla:** # (Cambiar) **número** (Estado, 1: Encender 0: Apagar) **número** (Posición de la bombilla en la estancia) **nombre** (Nombre de la estancia) Ej.: #11Habitación 1
- **Borrar base de datos:** / (Borrar la base de datos) Ej.: /
- **Mandar la base de datos actual al cliente:** < (Actualizar) Ej.: <

Para que la función de mandar la base datos al cliente funcione correctamente, mandando un mensaje por cada acción es necesario introducir un pequeño retardo entre mensaje y mensaje. Sino el cliente lo recibirá todo como un mensaje de gran longitud y no podrá tratarlo correctamente.

Al final del proyecto se decidió hacer el servidor fuese multiusuario y que varias personas a la vez pudiesen realizar cambios. Esto llevó algo de trabajo no solo en la parte del servidor, sino en la parte del cliente, que tiene que recibir los cambios, tratarlos en el momento y actualizar su vista.

Base de datos

La base de datos utilizada para mantener la persistencia de los elementos de nuestro hogar ha sido **Sqlite3**. Sqlite es un sistema de gestión de base de datos de pequeño tamaño que permite realizar operaciones básicas de manera

id	nombre
0	Habitación 1
1	Salón 1
2	Cocina 1
3	Baño 1

Tabla 5.1: Ejemplo de datos de la tabla *estancias*.

estancia	nombre	estado
Habitación 1	Bombilla 1	0
Habitación 1	Bombilla 2	0
Salón 1	Bombilla 1	1

Tabla 5.2: Ejemplo de datos de la tabla *elementos*.

id	ip
0	192.168.X.X

Tabla 5.3: Ejemplo de datos de la tabla *tableIp*.

muy sencilla. Entre las opciones de persistencia es la que más me ha gustado debido a su sencillez y funcionamiento. Para mantener la persistencia he creado operaciones básicas como guardar, recuperar, modificar y actualizar valores de las tablas *estancias*, *elementos* y *tableIp*. Podemos observar un ejemplo de cada tabla en las tablas 5.1, 5.2 y 5.3.

5.2. Aplicación Android

La aplicación Android ha sido la parte más complicada de este proyecto, ya que es más la extensa y compleja del proyecto. Por ello vamos a hablar de cierto puntos importantes.

Persistencia

La aplicación esta dividida en tres *Activity* o ventanas diferentes. Uno de los problemas más importantes es sin duda la persistencia de los elementos cuando nos movemos entre ellas. Resulta que cada vez que abandonamos una y nos dirigimos a otra, la primera llama a su método `onDestroy`, y cuando volvemos de la segunda a la primera llama a su método `onCreate`. Esto es lo que se llama ciclo de vida de una *Activity* 5.1. Cuando una ventana no está en la vista actual se destruye y cuando se vuelve a ella se crea de nuevo. El problema está en que nosotros manejamos los elementos dinámicamente, es decir, el método `onCreate` solo nos crea la ventana vacía como la primera vez que abrimos la aplicación, y somos nosotros los que manualmente añadimos más elementos a esa vista. Entonces, cada vez que nos desplazamos entre ventanas, se está llamando al método `onCreate` y nos representa dicha ventana vacía y perdemos los elementos creados dinámicamente.

Para evitar este problema es necesario crear una forma de mantener la persistencia de los elementos. Por ello, opté por crear una base de datos en **Sqlite** como en la parte de Python. Y dentro del método `onCreate` de cada ventana que lo necesite, haremos una llamada a la base de datos para obtener los elementos que teníamos antes.

En la base de datos creada para Android tenemos las tablas *estancias*, *elementos* y *datosServidor*. Podemos observar un ejemplo de cada tabla en las tablas 5.4, 5.5 y 5.6.

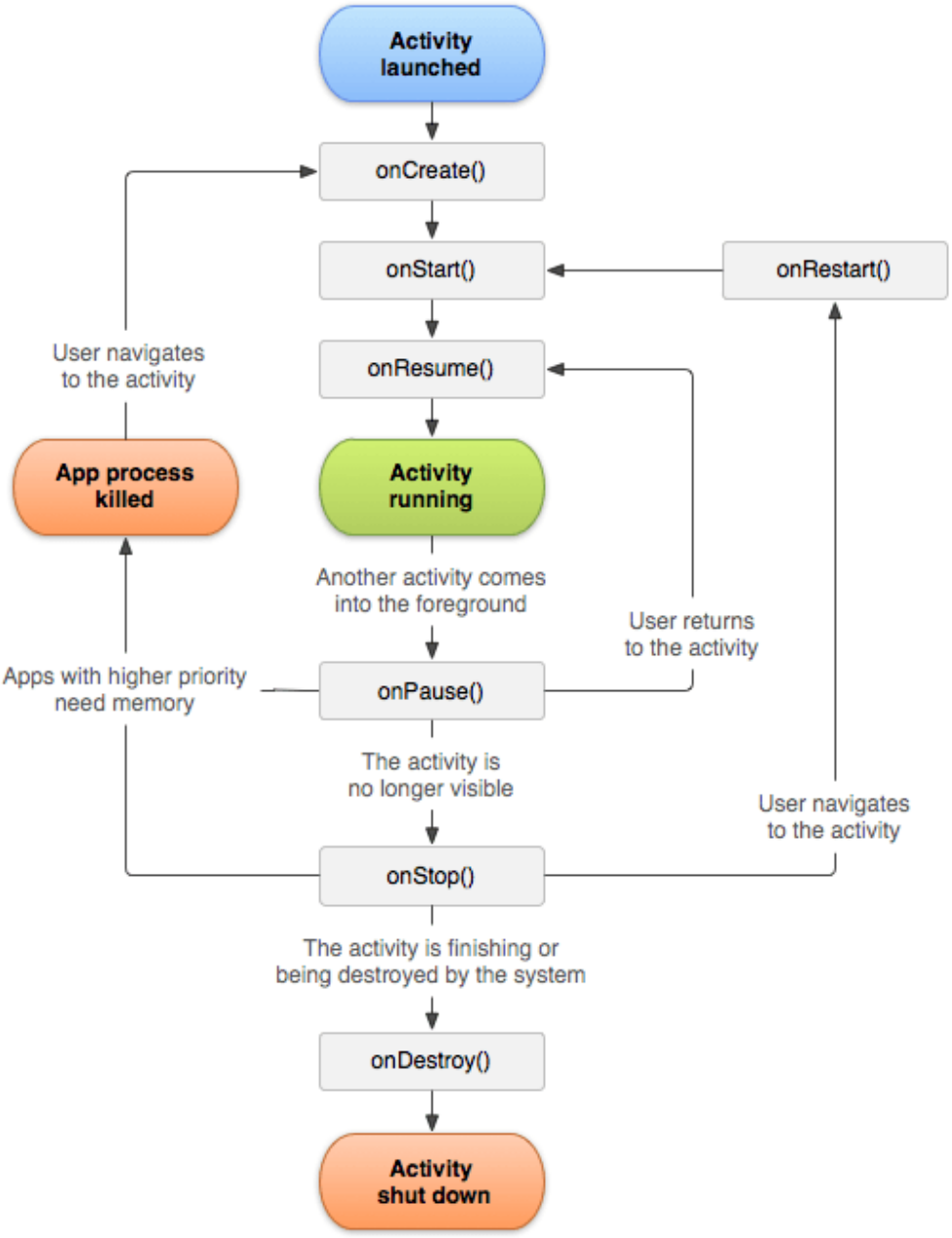

Figura 5.1: Ciclo de vida de una *Activity* en Android.

Visualización

En este apartado visualización quiero hacer referencia a la forma de representación de las estancias y bombillas en la aplicación.

id	nombre
0	Habitación 1
1	Salón 1
2	Cocina 1
3	Baño 1

Tabla 5.4: Ejemplo de datos de la tabla *estancias*.

estancia	nombre	estado
Habitación 1	Bombilla 1	0
Habitación 1	Bombilla 2	0
Salón 1	Bombilla 1	1

Tabla 5.5: Ejemplo de datos de la tabla *elementos*.

id	ip	puerto
0	192.168.X.X	8888

Tabla 5.6: Ejemplo de datos de la tabla *datosServidor*.

Para su correcta representación estaba buscando un método o estructura que me permitiese almacenar, modificar y eliminar de manera sencilla. Entonces se me ocurrió utilizar una *ListView*, es decir, una lista de elementos. Esta estructura tiene sus propios métodos para realizar la agregación, eliminación y modificación, pero yo pretendía tener una *ListView* personalizada. Pretendía que dentro de cada elemento o fila de la *ListView*, se pudiese insertar más elementos como imágenes, botones, interruptores, etc. Debido a esto, no me quedó otra opción que crear una *ListView* personalizada para las estancias y otra totalmente diferente para las bombillas.
Los elementos de la *ListView* son organizados y tratados mediante un adaptador, que es el que representa los elementos en la lista. Por tanto, creé dos clases nuevas que heredasen de la clase *ArrayAdapter* y así personalizarlas con mis preferencias. Una vez creadas, simplemente se crea una *ListView* y se asigna un nuevo adaptador:

```
listView.setAdapter(adaptador);
```

Tras conseguir la estructura personalizada y que los elementos se viesen como quería, necesitaba alguna manera de que el usuario modificase los elementos de forma sencilla. Entonces se me ocurrió crear un menú contextual sobre cada elemento de la lista que permite realizar modificaciones solo sobre ese. Para conseguir que el menú contextual funcione sobre una *ListView* hay que tener en cuenta que en la vista o *layout* de la fila, esté activada la opción que permita mantener pulsado un largo tiempo. Es decir, necesitamos acceder a nuestro adaptador personalizado y añadir lo siguiente:

```
row.setLongClickable(true);
```

Existe otro problema respecto a las *ListView* y es que hay que tener en cuenta que los cambios realizados sobre la vista de cada elemento de la lista no van a permanecer una vez se llame a `NotifyDataSetChanged()`. Este método lo que hace es cargar de nuevo los elementos de donde esta obteniendo la *ListView* los datos. Por tanto, para conseguir añadir nuevos elementos y que tras llamar a este método, no desaparezcan los cambios realizados, en vez de realizar los cambios sobre la vista, tenemos que realizarlos sobre la lista desde la que toma los elementos la *ListView*.

Conexión

Esta es la parte más importante de la aplicación Android y con la que más problemas me he encontrado. Cuando comencé, la manera que me pareció más correcta para enviar datos al servidor era a través de un tarea asíncrona, es decir, es una ejecución independiente de la parte gráfica que estamos viendo. Dentro de ella, abría una conexión mediante un *Socket*, enviaba el mensaje y lo cerraba de nuevo.

Pronto me dí cuenta de que de esta manera yo no podría recibir mensajes del servidor en cualquier momento y me interesaba. Así que opté por hacer como en la parte del Servidor Python, crear una ejecución independiente mediante un **Thread** y escuchar siempre si llega algún mensaje del servidor. Para conseguir que la ejecución de escucha del servidor funcionase tuve que crear una nueva clase que implementase la interfaz *Runnable* y de está manera meter mi código en el método `run()`. Además, una ventaja que tiene ser una ejecución independiente, es que no se interrumpe a pesar de realizar cambios entre ventanas. Yo quería que el mensaje que llegase fuese mostrado en pantalla y no es posible mostrar un mensaje en pantalla desde una ejecución independiente que no tiene parte gráfica.
Estuve indagando sobre como poder mandar mensajes, incluso objetos, desde un **Thread** a la parte gráfica y me encontré con la clase *Handler* [1]. La clase

Handler permite enviar y procesar mensajes y objetos asociados a la cola de un hilo. Entonces me di cuenta de que la manera de poder transportar el dato *String* obtenido del servidor es encapsulado a través de un objeto *Bundle*.

El método `sendMessage` nos permite poner en la cola un objeto *Message* que contendrá nuestro *Bundle* con nuestros *String* recibido. Una vez nuestro *Bundle* llegue, será procesado por el método `handleMessage`. Entonces lo ideal es sobreescribir ese método en una de las *Activity* para poder realizar los cambios pertinentes sobre el dato recibido.

A la hora de la creación del *Socket*, es mejor optar por crear un *Socket* sin conexión mediante el constructor vacío y luego conectarlo, que intentar instanciar un *Socket* con una ip y un puerto al inicio.
Otro problema respecto a la escucha del servidor es que cada vez que se trataba de conectar a través de la ventana del Servidor se creaba un nuevo hilo y esto daba lugar a que pudiésemos tener más de 5 hilos activos a la vez. Para solucionar este problema traté de usar un patrón de diseño *Singleton*, que haría que solamente hubiese una única conexión y fuese accesible desde cualquier punto de la aplicación. De esta manera podemos realizar la conexión al servidor según entremos a la aplicación desde la ventana principal. No he seguido el patrón de diseño al pie de la letra ya que en él se especifica que el objeto no tiene parámetros, pero el mío si los tiene. Y esto sucede porque a la hora de implementar la posibilidad de ser más usuarios conectados al servidor, necesitaba una manera de saber en que ventana nos encontramos para actualizar sus elementos según el mensaje que llegue.
Además este patrón de diseño ha sido usado debido a que para cerrar la conexión activa con el servidor, necesitamos acceder a la instancia de *RecepcionSocket* que se usó para la creación del hilo. Es decir, de está manera, la conexión que se crea en la ventana principal es posible cerrarla desde la ventana del Servidor, ya que estamos accediendo a la misma instancia *RecepcionSocket* que se usó en la creación.

Por último, quería decir que he tenido muchos problemas a la hora de saber si la conexión ha llegado a conectarse, está activa o cerrada. Es decir, si yo quisiese saber si ahora mismo estamos conectados al servidor, no he encontrado una manera sencilla de averiguarlo. El problema es que los métodos de los que nos provee la clase *Socket* no son nada claros.

- El método `isConnected()` [3] de la clase *Socket* indica que cerrar un *socket* no borra su estado de conexión, lo que significa que este método

devolverá un **True** aunque el *socket* esté cerrado si se ha llegado a conectar alguna vez.

- El método `close()` [2] de la clase *Socket* indica que una vez se ha cerrado el *socket* no estará disponible para nuevas conexiones, es decir, no puede ser reconectado. Por tanto, es necesario crear una nueva instancia del *Socket*.

Esto hace que haya muchos problemas para saber si tenemos una conexión activa con el servidor.

Trabajos relacionados

En este apartado voy a hablar sobre un proyecto parecido a este, pero que está actualmente funcionando.
La empresa Xiaomi tiene actualmente una aplicación como la mía en la tienda *Google Play*, desde la cuál se pueden sincronizar los dispositivos Xiaomi que lo permitan y cambiar parámetros de ellos.

Actualmente, poseo una bombilla de esta marca como la de la imagen 6.2 y cuenta con muchas posibilidades, entre ellas modificar su estado, color, intensidad y modo desde la propia aplicación de Xiaomi. Lo único que se necesita para que funcione es que el interruptor de la corriente esté encendido y que el dispositivo esté sincronizado con la aplicación.

Figura 6.2: Bombilla inteligente de la marca Xiaomi.

La aplicación *Mi Home* de Xiaomi se puede encontrar en *Google Play* en el siguiente enlace https://play.google.com/store/apps/details?id=com.xiaomi.smarthome&hl=es.

Conclusiones y Líneas de trabajo futuras

7.1. Conclusiones

Para este proyecto voy a hablar sobre conclusión a nivel de proyecto y a nivel personal.

A nivel de proyecto

A nivel de proyecto, al ser una idea que yo propuse esperaba más de el. Esperaba desarrollar todas o casi todas las líneas de trabajo de las que hablaré más adelante, pero que debido a problemas en el desarrollo o el tiempo no se han podido realizar. Aun así, ha sido un proyecto en el que la motivación y la ayuda de mi tutor ha sido fundamental para su desarrollo y estoy muy contento del resultado final.

A nivel personal

A nivel personal, este fue el primer proyecto que tuve en mente para realizar por mi cuenta, pero que gracias a mi tutor he podido desarrollar como mi trabajo final de carrera. Siempre me ha llamado la atención la interconexión entre los dispositivos, como lo consiguen y como es posible trabajar instantáneamente a grandes distancias. Por eso estoy muy contento de haber podido trabajar en un proyecto relacionado con este tema y poder ver de primera mano como se consigue que todo funcione.

7.2. Líneas de trabajo futuras

En este apartado vamos a tratar las líneas de trabajo futuras y los cambios que se podrían añadir para mejorar este proyecto.

Primero nombraré las líneas futuras y cambios y luego hablaré brevemente sobre ellos.

1. No tener que ser necesario estar conectado al mismo Wifi.
2. Añadir más elementos que poder controlar a las estancias.
3. Recepción de notificaciones cuando otro usuario haya realizado un cambio.
4. Añadir elementos generales de una casa como una lavadora o la calefacción.

Primera línea

Este es el mayor inconveniente del proyecto, y es que no es posible modificar el estado de tu casa fuera de ella. Ahora mismo el servidor de la Raspberry Pi se monta sobre una ip privada que solamente es accesible por los dispositivos conectados a esa misma red. No he conseguido poder montar el servidor sobre la ip pública y que de esta manera sea accesible desde otras redes. La posible solución que barajé fue trabajar con la configuración **NAT** y con el redireccionamiento de puertos a través del firewall del router.

Segunda línea

En este proyecto me he centrado tanto en la conexión, visualización y correcta ejecución de todas las partes, que solo he podido añadir iluminación a la centralita domótica. No sería muy complicado añadir más elementos ya que el proyecto está enfocado a ello.

Tercera línea

Esta es una de esas líneas en la que ya he trabajado pero que pospuse para más adelante para trabajar con otras tecnologías. Con los conocimientos que yo tengo, conseguí recibir notificaciones, pero el problema es que la aplicación debe estar abierta para recibirlas. Esto es un problema importante, ya que la esencia de ello es que no tengas la aplicación abierta para recibirlas porque sino ya estarías viendo los cambios.

Cuarta línea

Añadir elementos como puede ser la calefacción y poder programar su encendido y apagado son puntos que darían más nivel a un proyecto como este. Estos elementos generales que no pertenecen a una estancia específica deberían estar en el menú desplegable de la ventana principal, ya que desde ahí serían más accesibles.

Bibliografía

[1] Documentación online android — https://developer.android.com/reference/android/os/Handler. [Internet; descargado 2-mayo-2018].

[2] Documentación online android — https://developer.android.com/reference/java/net/Socket.html#close(). [Internet; descargado 2-mayo-2018].

[3] Documentación online android — https://developer.android.com/reference/java/net/Socket.html#isConnected(). [Internet; descargado 2-mayo-2018].

[4] ASOCIACIÓN ESPAÑOLA DE DOMÓTICA E INMÓTICA. http://www.cedom.es/sobre-domotica/que-es-domotica. [Online; accedido 20-mayo-2018].

[5] Wikipedia. Tkinter — wikipedia, la enciclopedia libre — https://es.wikipedia.org/w/index.php?title=Tkinter&oldid=87914348, 2015. [Internet; descargado 30-mayo-2018].

[6] Wikipedia. Android — wikipedia, la enciclopedia libre — https://es.wikipedia.org/w/index.php?title=Android&oldid=108236887, 2018. [Internet; descargado 30-mayo-2018].

[7] Wikipedia. Interfaz gráfica de usuario — wikipedia, la enciclopedia libre — https://es.wikipedia.org/w/index.php?title=Interfaz_gr%C3%A1fica_de_usuario&oldid=106284851, 2018. [Internet; descargado 30-mayo-2018].

[8] Wikipedia. Internet de las cosas — wikipedia, la enciclopedia libre — https://es.wikipedia.org/w/index.php?title=Internet_

de_las_cosas&oldid=105900761, 2018. [Internet; descargado 6-marzo-2018].

[9] Wikipedia. Python — wikipedia, la enciclopedia libre — https://es.wikipedia.org/w/index.php?title=Python&oldid=107334378, 2018. [Internet; descargado 30-mayo-2018].