UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

APLICACIÓN EN ANDROID PARA CONTROL REMOTO DE UN ORDENADOR CON WINDOWS

Adrián Brito Colomer

Tutor: José Oscar Romero Martínez

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2019-20

Valencia, 25 de noviembre de 2020

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS Gestión ambiental verificada REG.NO. ES-CV- 000030

Resumen

En el presente trabajo de fin de grado se muestra el proceso de desarrollo de una aplicación en Android para el control remoto de un ordenador con Windows. El objetivo principal es poder prescindir de periféricos como teclado y/o ratón, permitiendo que sea la aplicación – siempre con la intervención del usuario - quien simule sus funciones.

Para ello se ha implementado un modelo Cliente-Servidor con comunicación unidireccional que utiliza la tecnología Bluetooth. El Cliente, en este caso la aplicación en el dispositivo Android, se conectará al Servidor -ordenador con Windows – quien permanece a la escucha hasta recibir una petición de conexión. Una vez la conexión Bluetooth haya sido establecida, es el cliente quien enviará ciertos comandos, los cuales serán leídos e interpretados por el Servidor.

Entre las funcionalidades que la aplicación puede simular, cabe destacar que el usuario puede ajustar el volumen con precisión en un rango del uno al cien, controlar el movimiento del ratón mediante la pantalla táctil del Smartphone, o bien escribir en pantalla como si de un teclado físico se tratase.

Resum

Mitjançant el present treball de fi de grau es mostra el procés de desenvolupament d'una aplicació en Android per al control a distancia d'un ordinador amb Windows. El principal objectiu és poder prescindir d'alguns perifèrics com el teclat o el ratolí, i deixar que siga l'aplicació -amb la intervenció de l'usuari- qui puga controlar i simular aquestes funcions.

Per a dur-ho a terme, s'ha implementat un model Client-Servidor amb comunicació unidireccional. El Client, en aquest cas l'aplicació en el dispositiu Android, es connectarà al Servidor -l'ordinador amb Windows- qui roman a l'escolta fins rebre una petició de connexió. Quan la connexió haja sigut establida, és el Client qui enviarà algunes ordres, els quals seran llegits i interpretats pel Servidor.

Entre les funcionalitats que l'aplicació pot simular, cap destacar que l'usuari pot ajustar el volum amb precisió en una escala de l'un al cent, així com controlar el moviment del ratolí mitjançant la pantalla tàctil, o bé escriure per pantalla com si d'un teclat físic es tractés.

Abstract

This final degree project shows the development process of an Android application for remote control of a Windows computer. The main objective is to simulate some peripherals behavior such as keyboard and/or mouse, allowing the application -always with user intervention – to assume that behavior.
For this, a Client-Server model has been implemented with unidirectional communication that uses Bluetooth technology. The Client, in this case the application running on the Android device, will connect to the Server -computer with Windows - which remains listening until it receives a connection request. Once the Bluetooth connection has been established, it is the client who will send certain commands, which will be read and interpreted by the Server.

Among the functionalities that the application can simulate, it should be noted that the user can adjust the volume with precision in a range of one to one hundred, control the movement of the mouse through the touch screen of the Smartphone, or write on the screen as if on a keyboard physical involved.

ÍNDICE

Capítulo 1. Introducción

1.1 Precedentes

Actualmente vivimos en un mundo en que la tecnología forma parte de nuestro día a día. Estamos rodeados de smartphones, tabletas, libros electrónicos, ordenadores portátiles… Se cuentan por decenas los dispositivos electrónicos que una familia normal puede tener en casa.

No es casualidad que el aumento en el número de dispositivos electrónicos vaya ligado al incremento de las actividades que hemos trasladado al mundo digital: hacer la compra por Internet, pedir cita en el médico, consultar las noticias, chatear con gente en la otra parte del mundo o incluso tener una reunión con tu jefe a través de una pantalla son algunos ejemplos de cómo nos hemos adaptado a las nuevas tecnologías.

Son muchas las empresas que no han sabido adaptarse al mundo tecnológico y, por ende, han terminado viéndose superadas por una competencia

a más actualizada, y por un mercado que cada vez demanda más digitalización y automatización de procesos. En cambio, las empresas que sí han sabido dar este "salto" son las que actualmente siguen vivas, o incluso se han 'comido' a su competencia. El ejemplo más claro y quizás más conocido es el de la empresa canadiense Netflix, la cual ha sabido sustituir el famoso modelo de negocio de Blockbuster (que consistía en el alquiler de películas y juegos a través de videoclubs o encargos por correo) por otro modelo en el que se cobra por suscripción, y el cliente tiene todas las películas y series del catálogo disponibles en cualquier momento.

Echando la vista atrás, si algo se puede deducir es que aquello que sobrevive es aquello que se ha sabido adaptar a las nuevas tecnologías.

En cuanto a hábitos, son muchas las personas cuyo dispositivo preferido (detrás del Smartphone) es el ordenador portátil. Sin duda éste otorga una gran versatilidad por sus características generales como la facilidad de transporte, el tamaño de su pantalla o la potencia computacional, entre otras.

En este trabajo de fin de grado se va a hacer uso de esta potencia computacional que todos llevamos en el bolsillo para crear una herramienta capaz de sustituir algunos de los periféricos que comúnmente se utilizan junto a un ordenador. Estos periféricos son principalmente, el ratón y el teclado.

En lo referente a ámbitos de uso, esta herramienta es ideal para ejercer el control sobre una máquina Windows sin tener acceso físico a la misma. Es decir, una vez el servidor se ha iniciado, el usuario puede trasladar el control de la máquina Windows a su Smartphone.

Para ello, la aplicación está dotada de un control táctil (touchpad) el cual imita el panel táctil de un ordenador portátil. También incluye dos botones cuyas acciones asociadas son el clic izquierdo del ratón y el clic derecho del ratón. Para simular la funcionalidad de la rueda del ratón, se han implementado dos botones para los conocidos s*croll up* y s*croll down*. La herramienta también dispone de un pequeño cuadro de texto sobre el que podemos pulsar y se desplegará el teclado en Android. Si escribimos y pulsamos el botón Enviar, la cadena de texto que hayamos escrito se enviará a través del canal de comunicación Bluetooth al Servidor, el cual recibirá dicha cadena de texto y simulará las pulsaciones del teclado correspondientes para escribir dicho texto en pantalla. Por último, se ha incluido en la aplicación un pequeño control de volumen, mediante el cual el usuario puede ajustar el volumen de la máquina Windows a distancia sin necesidad de pulsar la combinación de teclas correspondiente en su teclado.

1.2 Motivación y justificación

Con el presente trabajo de fin de grado, se pretende elaborar un proyecto de complejidad suficiente para poner en práctica los recursos y conocimientos aprendidos durante los años universitarios.

El resultado es una aplicación de Cliente y otra de Servidor que conjuntamente interactúen utilizando un canal de comunicación Bluetooth y ambas sean capaces de ofrecer una experiencia de usuario óptima prescindiendo de ciertos periféricos.

Así mismo, el desarrollo de la aplicación supone un gran reto que superar, pues al principio no se sabía muy bien cómo plantear el trabajo, pero poco a poco se ha ido desgranando el problema en otros problemas más sencillos y a la vez más fáciles de abordar.

También cabe mencionar que además de los problemas previstos, durante el desarrollo del trabajo han ido apareciendo nuevos imprevistos que en un principio eran muy difíciles de prever. En estos casos la metodología ha sido la misma, desgranar el problema principal en otros más simples para tratar de abordarlos por separado de una forma más fácil, siendo capaces así de resolver el problema inicial.

1.3 Objetivos del proyecto

El objetivo principal de este proyecto es desarrollar una aplicación capaz de brindar el control remoto de un ordenador Windows a un smartphone Android, para que el usuario pueda interactuar con el ordenador a cierta distancia.

Esta aplicación resulta útil por ejemplo en el ámbito de una presentación, donde el ordenador está conectado al cable HDMI del proyector y el usuario no tiene fácil acceso físico al ordenador. Utilizando la app, será mucho más sencillo cambiar de vídeo o diapositiva, controlar el volumen del ordenador, hacer una búsqueda rápida en Google, etc.

Cabe destacar que se pretende que, una vez la aplicación esté en marcha, quede a disposición del usuario el control total sobre el ordenador como si estuviese delante del mismo. Es decir, la aplicación deberá ser capaz de realizar todas las funciones anteriores sin que el usuario tenga acceso físico al ordenador (solamente al principio para ejecutar el programa del servidor).

-Establecer comunicación mediante tecnología Bluetooth entre el smartphone y el ordenador.

-Controlar el movimiento del ratón (touchpad).

-Simular clic derecho e izquierdo de ratón.

-Simular pulsaciones de teclado (escribir y borrar caracteres).

-Controlar el volumen del ordenador.

-Deslizarse por la pantalla (*scroll up* y s*croll down*).

Capítulo 2. Metodología

2.1 Distribución en tareas

Tal y como se ha comentado en el apartado de Motivación y Justificación, para abordar el proyecto ha sido necesario descomponer el objetivo principal -desarrollar una aplicación que sea capaz de controlar el ordenador a través de un smartphone mediante Bluetooth- en otros objetivos de menor tamaño y complejidad. Son los siguientes:

-Elegir lenguaje de programación y entorno de desarrollo para Cliente (Java y Android Studio) y para Servidor (Java y NetBeans).

-Desarrollar un programa en el servidor capaz de permanecer a la escucha y aceptar peticiones de conexión Bluetooth.

-Desarrollar un programa en el cliente capaz de encender y apagar el bluetooth, hacer el dispositivo visible, listar los dispositivos vinculados al smartphone y realizar peticiones de conexión estableciendo un canal de comunicación.

-Una vez ya esté establecido el canal de comunicación, el cliente ha de ser capaz de enviar una cadena de caracteres y el servidor de recibirla correctamente y mostrarla en el log.

-La app cliente, al pulsar un botón (LC) envía una cadena de datos predefinida al servidor, el cual identifica dicha cadena de datos, y la interpreta como clic izquierdo de ratón. Entonces simulará un clic izquierdo de ratón en el ordenador. Se repite el proceso para el clic derecho de ratón (RC).

-La app cliente dispone de un recuadro que simula el touchpad. Debe ser capaz de detectar pulsaciones y movimientos dentro de dicho recuadro. También es capaz de codificar dichos movimientos y enviarlos al servidor a través del canal de comunicación. El servidor debe identificar que se trata de un movimiento de ratón, y ejecutar un método que realice el movimiento del ratón en las coordenadas X e Y recibidas.

-La app cliente dispone de dos botones verticales situados uno encima del otro. La función de los mismos es simular el movimiento de la rueda del ratón (*scroll up* y s*croll down*). Si el usuario hace un clic simple sobre el botón, el programa moverá solamente una línea. Si el clic es largo, se moverán tres líneas. Se profundizará en esto más adelante.

-El usuario debe ser capaz de escribir en el ordenador como si tuviese delante un teclado físico. Para ello, la app Cliente dispone de un pequeño recuadro de texto en el que el usuario puede escribir una cadena de hasta 100 caracteres. Al pulsar sobre el botón Enviar, el servidor recibirá dicha cadena de caracteres y simulará pulsaciones de teclado en el orden correspondiente. También se dispone de un botón que borrará un carácter por cada pulsación del mismo.

Capítulo 3. Conceptos Previos

3.1 Arquitectura Cliente-Servidor

Para poder entender mejor el funcionamiento de este trabajo es esencial que se explique la arquitectura utilizada, en este caso es el modelo Cliente-Servidor, utilizado en muchos servicios de Internet.

La arquitectura Cliente-Servidor representa la forma en la que los nodos de una red se comunican entre ellos. A cada uno de los nodos se le asigna el rol de Cliente, y a otro el rol de Servidor. El rol asignado no es más que la función que van a desempeñar cada uno de ellos durante la comunicación.

Aunque el modelo Cliente-Servidor es el soporte de la mayor parte de la comunicación a través de nodos de una red, en el presente trabajo vamos a utilizarla en un ámbito de conexión local (LAN) y con sólo un cliente y un servidor.

Es importante destacar que, en el modelo de capas, dicho rol es asignado a nivel de aplicación ya que se hace distinción en el comportamiento de la aplicación cliente y la aplicación servidor.

Un ejemplo son los servidores DNS, donde el navegador del usuario actúa como cliente solicitando respuesta del servidor. En este caso, el cliente necesita resolver una dirección web, por lo que es el servidor quien hará la tarea de resolverlo, y de enviar la respuesta de vuelta al cliente.

Otro ejemplo claro es el mismo funcionamiento de Internet. Lo habitual es que el usuario navegue por Internet visitando distintas páginas web, e incluso distintos recursos en la misma página. En este caso, el servidor está alojado en una dirección web. El cliente envía una petición de algún recurso a dicha dirección, y espera la respuesta por parte del servidor con dicho recurso adjunto. En lo práctico, los recursos suelen ser archivos HTML y CSS que conforman la parte visual de las páginas web.

3.1.1 Componentes de la arquitectura Cliente-Servidor

Para entender mejor esta arquitectura, vamos a definir los componentes básicos que la conforman:

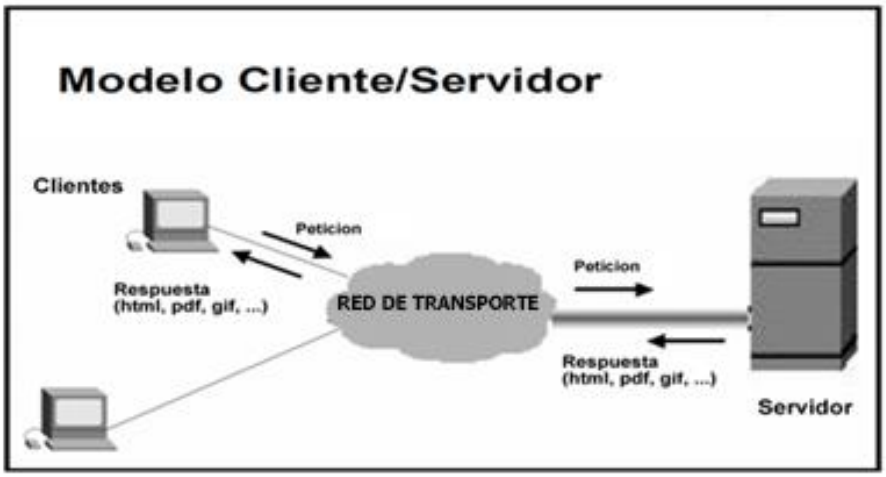

Figura 1. Componentes del modelo Cliente-Servidor.

-Protocolo de comunicación: Es necesario que ambas partes utilicen una tecnología y un protocolo concretos para su correcta comunicación. Existen aplicaciones en las que se lleva a

cabo una negociación del protocolo a utilizar, así como de unas características o parámetros concretos de la comunicación.

-Red de transporte: Generalmente los nodos que se quieren comunicar lo harán a través de una red de dispositivos. Esta red actúa a modo de canal de comunicación entre cliente y servidor.

-Cliente: El cliente es, conceptualmente, quien suele iniciar la conexión y solicita determinados recursos. Comúnmente, el cliente realiza una petición al servidor para establecer una conexión. Una vez la conexión ha sido aceptada y establecida, el cliente puede solicitar recursos al servidor a través del canal de comunicación establecido. El rol de cliente puede tomarlo, por ejemplo, un navegador que solicita un recurso (archivos que conforman la página web) a un servidor web.

-Servidor: El servidor es, conceptualmente, quien atiende las peticiones de clientes, así como el poseedor de recursos. Generalmente el servidor inicia su actividad quedando a la espera de peticiones. Una vez recibe una petición de conexión, el servidor es capaz de recibirla, interpretarla y atenderla. Siguiendo con el ejemplo anterior, el servidor, al recibir la petición de un recurso concreto por parte del cliente, tiene la misión de encapsular dicho recurso en el formato correspondiente (protocolo) y enviarlo al cliente por el canal previamente establecido.

3.2 Tecnología Bluetooth

El archiconocido Bluetooth no es más que una tecnología de comunicación inalámbrica entre dispositivos basada en ondas de radiofrecuencia que operan en la banda ISM de los 2,4 GHz, y que hace posible la transmisión simultánea de datos y voz entre equipos relativamente cercanos.

Esta tecnología ha evolucionado y mejorado en las últimas dos décadas, y hoy en día está presente en la inmensa mayoría de dispositivos electrónicos cuyas necesidades requieren de comunicaciones inalámbricas de corto alcance.

3.2.1 Origen

El origen de la tecnología Bluetooth se remonta a principios del 1994 en Lund, Suecia, como iniciativa de la compañía Ericsson Mobile Communications junto con otras cuatro compañías (Nokia, Intel, IBM y Toshiba). En diciembre de 1999 se unirían compañías como Microsoft y Motorola, además de otras. Todas ellas formaron lo que se conoce como Bluetooth SIG (Special Interest Group) que hoy en día se compone de más de 9000 empresas.

El nombre Bluetooth fue escogido en honor a un rey de origen danés llamado Harald Blatand (diente azul), cuya traducción al inglés es Harald Bluetooth. Este monarca destacó por unificar varios pueblos y reinos noruegos, suecos y daneses en torno al siglo X. De la misma manera, Bluetooth había nacido para unificar diferentes tecnologías como ordenadores, teléfonos móviles, impresoras, dispositivos IoT, etc.

Por otra parte, el logo de esta tecnología se compone de la superposición de las iniciales de este rey, H y B, que en el alfabeto de runas equivalen a las runas Harall y Berkana.

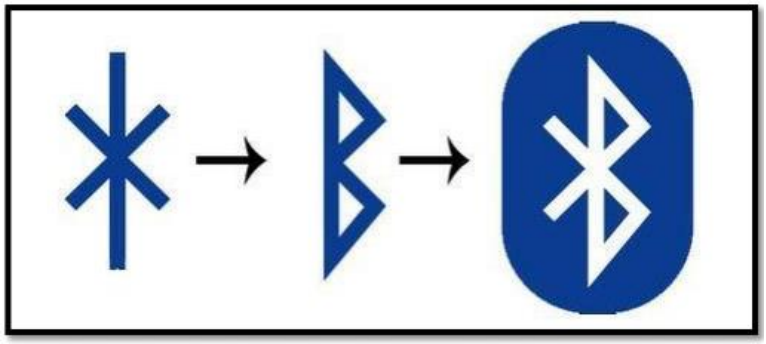

Figura 2. Logo Bluetooth

3.2.2 Características

Las principales características que se perseguían con el desarrollo de esta tecnología eran las siguientes:

-Facilitar comunicaciones inalámbricas entre equipos con diferentes características (figura 3).

-Eliminar cables y conectores entre equipos.

-Crear pequeñas redes inalámbricas para facilitar la sincronización de datos entre varios equipos

-Establecimiento de conexiones con poco gasto energético.

-Bajo coste de producción (se fijó un límite en el gasto de producción de 5 dólares americanos).

Hoy en día esta tecnología ha avanzado notablemente, aumentando el alcance de la comunicación y velocidad de transmisión de datos, que en un principio era de unos 720 kbps hasta los más de 20 Mbps actuales.

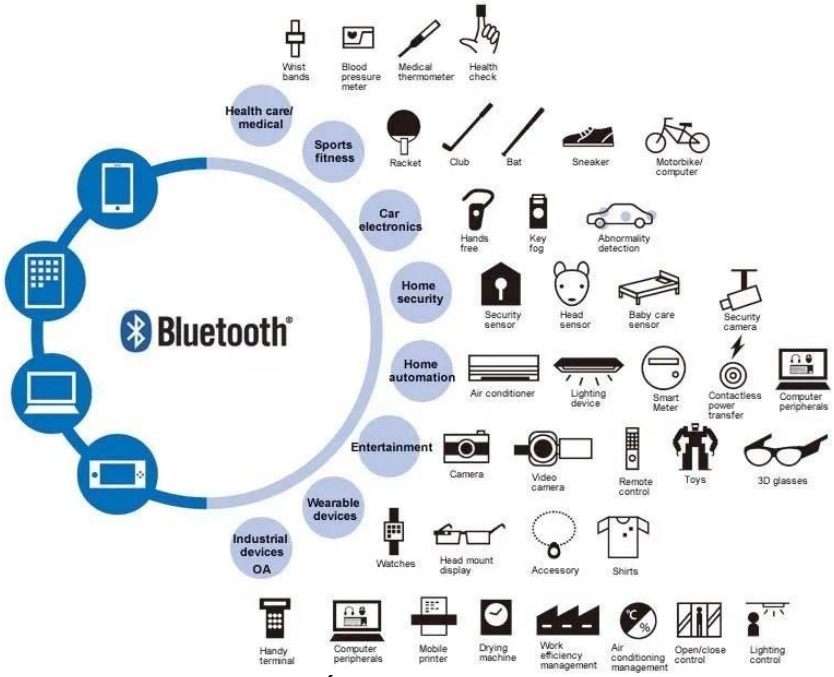

Figura 3. Ámbitos de uso de Bluetooth

3.2.3 Clases

Los equipos que deseen utilizar esta tecnología para comunicarse, deberán encontrarse dentro de un cierto radio de alcance que suele ser relativamente corto, aunque varía según el dispositivo utilizado.

Según la potencia de transmisión del dispositivo, y por ende de su alcance, los dispositivos Bluetooth se clasifican en cuatro clases diferentes:

Clase 1: potencia de consumo medio de 100 mW y alcance de hasta 100 metros.

Clase 2: potencia de consumo medio de 2,5 mW y alcance de hasta 10 metros.

Clase 3: potencia de consumo medio de 1 mW y alcance de hasta 1 metro.

Clase 4: potencia de consumo medio de 0,5 mW y alcance de hasta 0,5 metros.

3.2.4 Versiones

Desde el lanzamiento de la primera versión en 1999 hasta la actualidad, esta tecnología ha experimentado diferentes evoluciones que han traído mejoras como solucionar problemas de conexión, mejorar la velocidad de transmisión o mejorar el alcance, por ejemplo.

Aunque actualmente la versión publicada más reciente es Bluetooth 5.2, es Bluetooth 5.0 o Bluetooth 5.1 por la que apuestan la mayoría de aparatos más modernos del mercado, como algunos teléfonos móviles de última generación.

Las versiones de Bluetooth, con algunas de sus particularidades más importantes, son:

- **Bluetooth 1.0**: Fue la primera versión lanzada en 1999 y se usó para transmitir datos, y precisamente por ser el primer paso para la tecnología, tuvo muchos problemas de conectividad y seguridad. Actualmente se encuentra en desuso. Posteriormente se lanzaron las actualizaciones 1.1 y 1.2, las cuales permitían una conexión más rápida (hasta 721 kbps) y la capacidad de detectar otros dispositivos Bluetooth, así como el Adaptative Frequency-hopping (o AFH, Salto de Frecuencia Adaptable, que aumenta la resistencia a las interferencias). Estas dos versiones fueron las primeras en ser reconocidas como estándares de comunicación IEEE, bajo el nombre de IEEE 802.15.1-2002 y IEEE 802.15.1-2005.
- **Bluetooth 2.0**: Esta versión fue lanzada en 2004 y es compatible con Bluetooth 1.2. El avance más importante fue sin duda la actualización a Bluetooth 2.1 + BR/EDR (Basic Rate/Enhanced Data Rate), que permite la transmisión a 1 Mbps en modo BR, y a 2 Mbps en modo EDR. EDR combina las modulaciones GFSK y PSK, esta última con dos variantes (π/4-DQPSK y 8DPSK), lo que permite un menor consumo del dispositivo. También se mejoró la conectividad de cara al usuario, ya que un dispositivo podía detectar a otro y conectarse a él.
- **Bluetooth 3.0**: También conocida como Bluetooth v3.0 + HS (High Speed), esta versión fue lanzada en 2009 y destaca porque supuso un salto de velocidad ofreciendo transferencias teóricas de hasta 24 Mbps. Para ello hace uso del AMP (Alternate MAC/PHY). De este modo se permite que el canal de radio frecuencia sea usado para detección de dispositivos, el establecimiento de conexión y la configuración del perfil. En cambio, para la transferencia de grandes cantidades de datos se utiliza un canal PHY MAC 802.11.
- **Bluetooth 4.0**: Lanzada en 2010 y conocida como Bluetooth v4.0 o BLE (Bluetooth Low-Energy), combina las tres especificaciones anteriores en una (Bluetooth versión 2.1+EDR, Bluetooth 3.0 de alta velocidad y Bluetooth 4.0 de baja energía). Las tres especificaciones se pueden usar de forma combinada o separada según si el dispositivo lo soporta o no. Destaca por ser una versión dirigida a aplicaciones de muy baja potencia alimentados con una pila de botón y por el rápido desarrollo de enlaces entre dispositivos. La característica clave de Bluetooth 4.0+LE es su modo de baja energía, que permite reducir al máximo el consumo energético. Esto cobra especial relevancia en dispositivos del ámbito de la salud que basen sus comunicaciones en esta tecnología.
- **Bluetooth 5.0**: Esta nueva versión presenta una mejora del Bluetooth Low Energy (BLE) ya introducido en la anterior versión. Fue anunciado en 2016 y prometía el doble de velocidad, más fiabilidad y 4 veces más alcance, además de un 800% más de capacidad que su predecesor. La diferencia principal entre ambas versiones es que en BLE 4.0 era difícil la retrocompatibilidad, pero con BLE 5.0 cualquier dispositivo puede utilizar virtualmente el modo de bajo consumo. Se incluyen mecanismos de optimización de cabeceras, permitiendo paquetes de hasta 255 octetos de longitud, aumentando así la carga de los paquetes gracias a que utiliza uno de los 36 canales disponibles como canal de señalización. También se mejora el uso del espectro electromagnético con un sistema llamado SAM (Slot Availability Masks), cuyo fin es evitar interferencias con otras

tecnologías que utilicen bandas cercanas a la ISM de 2,4 GHz, como LTE. También introduce una mejora en el AFH (Adaptative Frequency-Hopping) llamada Channel Selection Algorithm #2. Esta nueva forma de seleccionar el canal por el que se van a enviar datos proporciona secuencias pseudoaleatorias de saltos entre canales, lo que implica que la comunicación se lleva a cabo a través de una amplia selección de canales, lo que hace que Bluetooth 5.0 rinda mejor en medios de radio con interferencias.

3.3 Android

3.3.1 Introducción

Estos últimos años, los teléfonos móviles han evolucionado enormemente. Desde aquellos teléfonos que parecían ladrillos pensados únicamente para realizar llamadas, a los Smartphones de última generación que hoy en día dominan el mercado.

Esta evolución del concepto de teléfono móvil va indudablemente ligada a las necesidades que nuestra sociedad atribuye al uso de los mismos. Actualmente un Smartphone cuenta con decenas (incluso cientos) de funcionalidades, pero entre las más usadas no encontramos las llamadas telefónicas.

La forma en que nos comunicamos ha cambiado y así lo han hecho también los sistemas operativos que se ejecutan en los teléfonos, hasta el punto en que en el mundo hay miles de programadores que trabajan a diario en el desarrollo de los mismos. Más adelante se explicará cómo se ha realizado el desarrollo de la aplicación en Android utilizando Android Studio.

Según un estudio realizado por el IDC (International Data Corporation), en 2010 los dispositivos cuyo sistema operativo era Android representaban el 15,6% de cuota de mercado frente al 86,1% en 2019, alcanzando casi los 1200 millones de dispositivos.

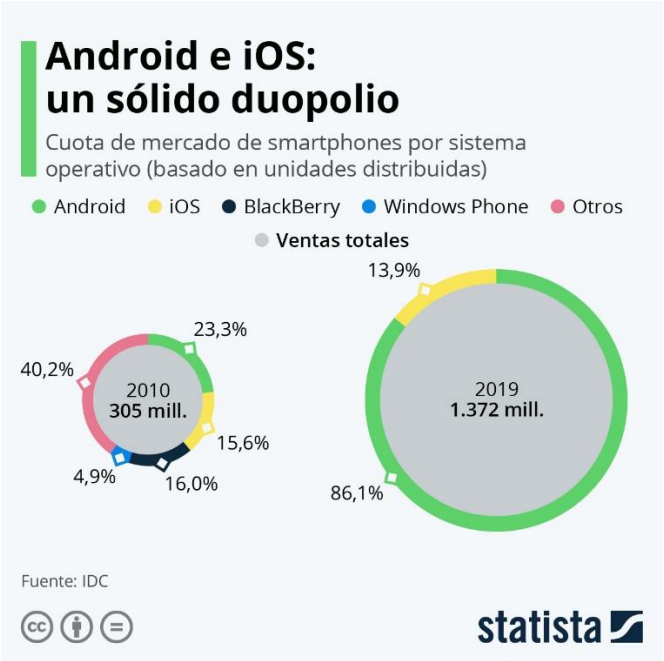

Figura 4. Cuota de mercado según sistema operativo.

Android es un sistema operativo enfocado principalmente a teléfonos móviles y cuyo núcleo (o core) se basa en Linux. Esto implica que este software es de código abierto, lo que permite que haya comunidades de desarrollo de aplicaciones para la plataforma, aumentando así las prestaciones de los dispositivos y, sobre todo, adaptándose a las necesidades del usuario de una forma muy rápida.

Actualmente encontramos variedad de dispositivos que ejecutan sistemas operativos Android, como televisores inteligentes con *Android TV*, smartwatches con *Wear OS* e incluso vehículos con *Android Auto.*

3.3.2 Historia

Android fue fundado en 2003 por Andy Rubin, Rich Miner, Nick Sears y Chris White. El objetivo inicial era crear un sistema operativo para cámaras digitales, pero rápidamente cambiaron la dinámica del equipo, y desarrollando un sistema capaz de competir con Windows Mobile y Symbian.

Sin embargo, no es hasta 2005 cuando Android Inc es adquirida por Google. A partir de aquí empezaría el desarrollo intensivo de la plataforma. En 2007 se anuncia al mercado y en septiembre de 2008 se lanza al mercado americano el primer teléfono con Android 1.0, el *T-Mobile G1* (figura 5).

Figura 5. T-Mobile G1.

En 2007, Apple lanza lo que iba a suponer una revolución en la forma en que consumimos la tecnología, el iPhone de primera generación. Este evento es sin duda un punto de inflexión en el desarrollo de Android, ya que la aparición de una competencia tan firme no hizo más que impulsar a los de Google a mejorar su sistema operativo.

Desde entonces, Android ha ido evolucionando como sistema operativo, pasando por distintas versiones. Algunas de ellas incluían parches de seguridad, y otras traían importantes mejoras del sistema, como una mayor personalización que mejoraría la experiencia de usuario, cambios en la interfaz o más funcionalidades.

En la actualidad, muchas de estas versiones antiguas han quedado obsoletas por el avance del hardware que montan los teléfonos móviles. Por ejemplo, el T-Mobile G1 no puede correr la última versión (Android 11).

3.3.3 Versiones

Desde el lanzamiento de la primera beta de Android en 2008, las diferentes actualizaciones se han ido sucediendo anualmente. Cabe destacar que desde 2009, las versiones recibían nombres especiales siguiendo un orden alfabético.

En la siguiente tabla se relacionan las versiones de Android, el año de publicación y el nombre especial de la versión.

Versión	Año	Nombre
1.0 a 1.4	2008	*Banana Bread*

1.5	2009	*Cupcake*
1.6	2009	*Donut*
2.0 y 2.1	2009	*Eclair*
2.2	2010	*Froyo*
2.3	2010	*Gingerbread*
3.0 a 3.2.6	2011	*Honeycomb*
4.0	2011	*Ice Cream Sandwich*
4.1 a 4.3	2012	*Jelly Bean*
4.4	2013	*KitKat*
5.0	2014	*Lollipop*
6.0	2015	*Marshmallow*
7.0	2016	*Nougat*
8.0	2017	*Oreo*
9.0	2018	*Pie*
10.0	2019	*Android 10*
11.0	2020	*Android 11*

Tabla 1. Clasificación de las versiones Android.

3.4 Android Studio

Android Studio es un IDE (*Integrated Development Environment*, o Entorno de Desarrollo Integrado) oficial para el desarrollo de aplicaciones en Android. La gran ventaja de Android Studio frente otras plataformas de desarrollo es que está 100% especializado en el desarrollo de aplicaciones Android.

Entre sus características cabe destacar su Editor Visual, que permite crear vistas para las actividades de una forma muy intuitiva, simplemente arrastrando y soltando elementos.

También incluye un Analizador de APK's, lo que permite inspeccionar aplicaciones y estudiar su código. Con esto se hace posible hacer nuestras propias modificaciones de apps de terceros, bien sea quitando partes del código que no se necesitan o bien escribiendo nuevo código que añada ciertas funcionalidades. Incluso se pueden analizar APK's que no hayan sido desarrollados con esta herramienta.

Otra de sus características estrella es conocida como *Fast Emulator*, y es tremendamente útil en el desarrollo de apps para Android. Consiste en una pequeña ventana que emula un terminal Android (se puede seleccionar la marca y el modelo de smartphone, así como la versión de Android que ejecuta) y en él se ejecuta la aplicación que se está desarrollando. Esta herramienta es esencial para realizar simulaciones sin tener un dispositivo Android físicamente presente. También permite a los desarrolladores encontrar fallos o "*bugs*" en la lógica de sus programas.

Por último, también incluye un Editor de Código Inteligente, una característica que facilita el desarrollo del código. Esta herramienta ahorra mucho tiempo ya que cuando se escribe una palabra, automáticamente sugiere el final de la misma, algo indispensable cuando se trabaja con librerías cuyos métodos empiezan por la misma palabra. También es capaz de detectar una mala estructura de código y de sugerir correcciones para la misma.

Capítulo 4. Cliente

4.1 Descripción del programa Cliente

Tal y como se ha mencionado anteriormente, es el smartphone Android quien tomará el rol de Cliente en esta comunicación. Para ello, es esencial que la aplicación Android actúe como tal, y vaya en sincronía con la aplicación del Servidor.

A continuación, se explicará detalladamente cada una de las actividades que se han elaborado, así como las vistas asociadas a cada actividad.

4.1.1 Primera Actividad: MainActivity

MainActivity.java es el nombre por defecto que recibe la primera actividad creada en el proyecto de Android Studio. Al iniciar la aplicación en el Smartphone, es también la primera que se ejecutará. Para ello, se han utilizado las etiquetas en el archivo AndroidManifest.xml que se muestran en la figura 6.

```
<activity android:name=".listarDispositivos"/>
<activity android:name=".touchpad" />
<activity android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Figura 6. Ejemplo de figura.

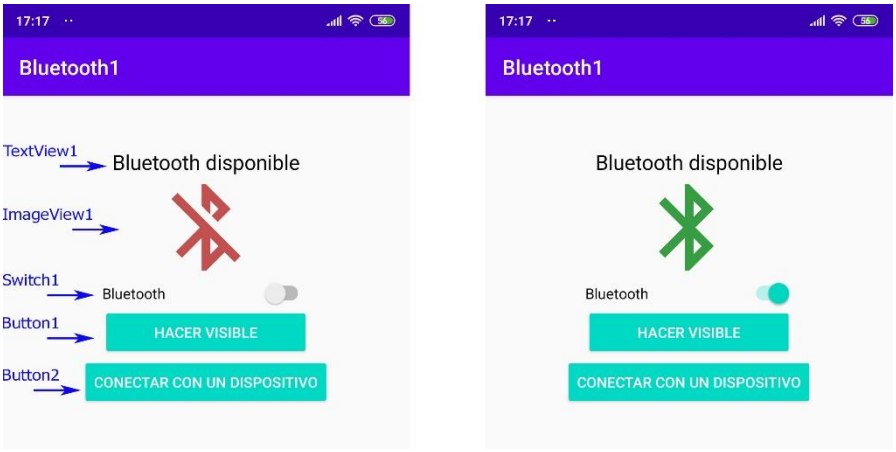

Figura 7. Layout de MainActivity.

Al iniciarse la actividad, se muestra un sencillo *layout* o vista (figura 7) llamado activity_main.xml que contiene los siguientes elementos enumerados a continuación en orden descendente:

- TextView1: Una simple vista de texto que puede tomar dos valores en función de la disponibilidad del adaptador Bluetooth. Si el adaptador se encuentra disponible para su uso, este TextView1 mostrará “Bluetooth Disponible”. En cambio, si no se han otorgado permisos para acceder al adaptador Bluetooth del teléfono o éste no se encuentra disponible, TextView1 tomará el valor de “Bluetooth NO Disponible”.

```
//1.- Comprueba si BT está disponible. Modifica el texto del TV con id: statusBlueTv

mBlueAdapter = BluetoothAdapter.getDefaultAdapter();

if (mBlueAdapter == null){
    mStatusBlueTv.setText("Bluetooth NO disponible");
}
else {
    mStatusBlueTv.setText("Bluetooth disponible");
}
```

Figura 8. Lógica asociada a TextView1.

- ImageView1: Se trata de la imagen del logotipo del Bluetooth cuyo color depende de si el Bluetooth está encendido o no en el dispositivo Android. En el caso de que esté encendido, la imagen se pondrá en verde y, en caso contrario, en rojo. Se pueden apreciar las dos vistas en la figura 7.

```
//2.- Pone imagen verde si BT está ON y rojo si BT está OFF
if (mBlueAdapter.isEnabled()){
    mSwitch1.setChecked(true);
    mBlueIv.setImageResource(R.drawable.ic_action_on);
}
else {
    mSwitch1.setChecked(false);
    mBlueIv.setImageResource(R.drawable.ic_action_off);
}
```

Figura 9. Lógica asociada a ImageView1.

- Switch1: Se trata de un pequeño interruptor que nos permite activar y desactivar el Bluetooth en el dispositivo. Para ello, se ha implementado un *listener* que ejecuta el código del método *onClick()* cuando el usuario activa o desactiva el interruptor. Si el usuario toca el interruptor y el Bluetooth está desactivado, se mostrará un pequeño pop-up con el texto "Encendiendo Bluetooth…". A continuación, se pondrá el ImageView01 en verde y se activará el Bluetooth en el dispositivo.

```
//3.- Listener del switch ON/OFF
mSwitch1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //Si NO esta activo
        if (!mBlueAdapter.isEnabled()){
            showToast( msg: "Encendiendo Bluetooth...");
            mBlueIv.setImageResource(R.drawable.ic_action_on);
            Intent intent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(intent, REQUEST_ENABLE_BT);
        }
        else {
            //Si BT está ON lo desactivamos
            mBlueAdapter.disable();
            showToast( msg: "Apagando Bluetooth...");
            mBlueIv.setImageResource(R.drawable.ic_action_off);
            }
    }
});
```

Figura 10. Lógica asociada a Switch1.

- Button1: La función de este botón es, sencillamente, volver visible al dispositivo. De la misma forma que el interruptor anterior, para detectar pulsaciones en botones se hace uso de un *listener* (Figura 11). Al ser pulsado, primero realizará una comprobación por si el

dispositivo estuviese escaneando. En caso negativo, se ejecutará un *intent* para hacer que el dispositivo sea visible. En caso afirmativo, se pausará dicho escaneo y se procederá a ejecutar el *intent*.

```
//4.- Listener del boton VISIBLE
mDiscoverBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //Nos aseguramos que no está buscando dispositivos
        if(mBlueAdapter.isDiscovering()){
            mBlueAdapter.cancelDiscovery();
        }

        showToast( msg: "Tu dispositivo será visible");
        Intent intent = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
        startActivityForResult(intent, REQUEST_DISCOVER_BT);
    }
});
```

Figura 11. Ejemplo de figura.

- Button2: Al pulsar el botón de “Conectar con un dispositivo”, se ejecutará un *intent* cuya función es iniciar la siguiente actividad, llamada ListarDispositivos que se explica a continuación.

```
//5.- Listener del boton CONECTAR CON UN DISPOSITIVO
getListBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        startActivity(new Intent(getApplicationContext(),listarDispositivos.class));
        finish();
    }
});
```

Figura 12. Ejemplo de figura.

4.1.2 Segunda Actividad: ListarDispositivos

Esta segunda actividad llamada ListarDispositivos tiene una función simple y concreta: mostrar una lista de los dispositivos Bluetooth vinculados al teléfono, para que el usuario pueda seleccionar a cuál de ellos se quiere conectar.

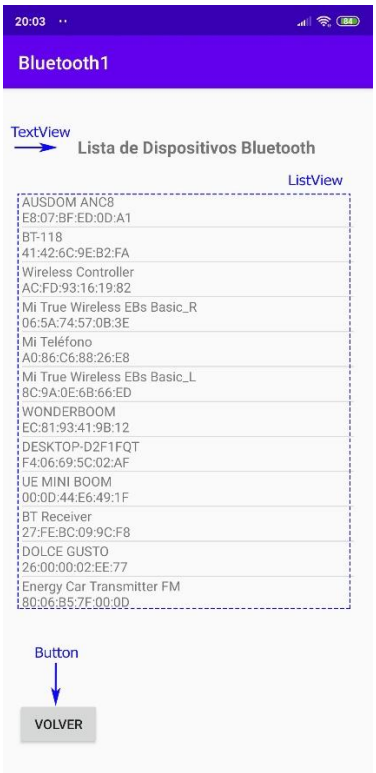

Figura 13. Layout de ListarDispositivos

La estructura de esta actividad es también muy sencilla. Se sobreescriben dos métodos, *onCreate()* y *onResume()*. El primero se ejecutará cuando se inicie la actividad por primera vez, y el segundo se ejecutará cada vez que se vuelva a esta actividad.

En primer lugar, el método *onCreate()* contiene una referencia al *layout* correspondiente de esta actividad (*listaDispositivos.xml*), además de un *listener* para el botón Volver, cuya función es cerrar la actividad actual y volver a la anterior.

En segundo lugar, el método *onResume()* contiene la lógica que permite la visualización de la lista de dispositivos vinculados.

El *layout* de esta actividad consta básicamente de tres elementos:

- TextView: Corresponde al título de la lista. Toma el valor fijo de “Lista Dispositivos Bluetooth”.
- Button: Es el botón que permite cerrar esta actividad y volver a la anterior.
- ListView: Es la lista que se comenta anteriormente. Se muestra el nombre de los dispositivos vinculados al teléfono y su dirección MAC. Para ello, se obtiene la lista de dispositivos emparejados a través del adaptador Bluetooth, haciendo uso del método *getBondedDevices()*. A continuación, se utiliza un bucle *for* para añadir los dispositivos a la lista.

```
mPairedDevicesArrayAdapter = new ArrayAdapter( context: this, R.layout.dispositivos_encontrados);
//Obtenemos adaptador

BtAdapter = BluetoothAdapter.getDefaultAdapter();
//Obtenemos dispositivos emparejados
Set <BluetoothDevice>pairedDevices = BtAdapter.getBondedDevices();

if (pairedDevices.size() > 0)
{   //Bucle para añadir nombre y MAC de cada dispositivo a la lista
    for (BluetoothDevice device : pairedDevices) {
        mPairedDevicesArrayAdapter.add(device.getName() + "\n" + device.getAddress());
    }
}
```

Figura 14. Lógica asociada a ListView.

4.1.3 Tercera Actividad: Touchpad

La tercera actividad llamada Touchpad es la parte funcional de la aplicación. Desempeña una serie de funciones que se detallan a continuación y permite que el usuario tome el control del ordenador. Para ello, la aplicación consta de un proceso inicial orientado a establecer la conexión Bluetooth con el servidor (tomando parámetros de la anterior actividad como el nombre o la MAC), seguido de tres módulos diferenciados que permiten al usuario interactuar con el ordenador. Estos módulos se detallarán más adelante.

Cuando el usuario ha seleccionado el dispositivo al que se quiere conectar, se da paso a esta tercera actividad. Lo primero que hace la actividad en el método *onCreate()* es declarar todos los *listeners* que se verán a continuación. En el método *onResume()* es donde se establece la conexión con el servidor.

4.1.3.1 Creando la conexión con el Servidor

Para crear la conexión, primero se obtienen los parámetros de la actividad anterior que se han mencionado en el párrafo anterior. A continuación, y haciendo uso de un bloque try-catch, se obtiene el adaptador Bluetooth con el método *getDefaultAdapter()*. También se guardarán los detalles del ordenador en un objeto del tipo *BluetoothDevice* haciendo uso del método *getRemoteDevice()* y pasándole la MAC del servidor como parámetro.

Una vez obtenidos el adaptador Bluetooth y el dispositivo vinculado al que se desea conectar, es momento de crear e iniciar un socket. Para ello, se utiliza el método *createRfcommSocketToServiceRecord*(java.util.UUID) que devuelve un objeto socket ya listo para la conexión. Por último, se establece la conexión haciendo uso del método *connect()* y se obtiene el *stream* de salida en una variable *outputStream* mediante el método *Socket.getOutputStream()*.

Una vez establecido el canal de comunicación, cuando el usuario interactúe con alguno de los módulos, el smartphone enviará cadenas de caracteres al servidor a través de dicho canal. Para ello se ha creado un método llamado *escribe()* (figura 15) que utilizarán todos los módulos para comunicarse con el servidor. Este método hace uso de *out*, el objeto *outputStream* que se ha mencionado anteriormente, para obtener y escribir en el *stream* de salida. El método *flush*() se utiliza para forzar la salida del mensaje y que el programa no quede a la espera de más caracteres.

```
public void escribe(String mensaje) {
    byte[] buffer = mensaje.getBytes();
    try {
        out = getWriter(btSocket);
        out.write(buffer);
        out.flush();
        Log.d(TAG,  msg: "MENSAJE ENVIADO CON EXITO.");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Figura 15. Ejemplo de figura.

Es interesante remarcar procedimiento que se ha seguido. Llegados a este punto, ya se ha creado la conexión y abierto un canal de comunicación. Ahora es el momento de enumerar y utilizar los

elementos del *layout* de esta actividad, al que se ha llamado activity_touchpad.xml (figura 16), y que permiten al usuario controlar el ordenador.

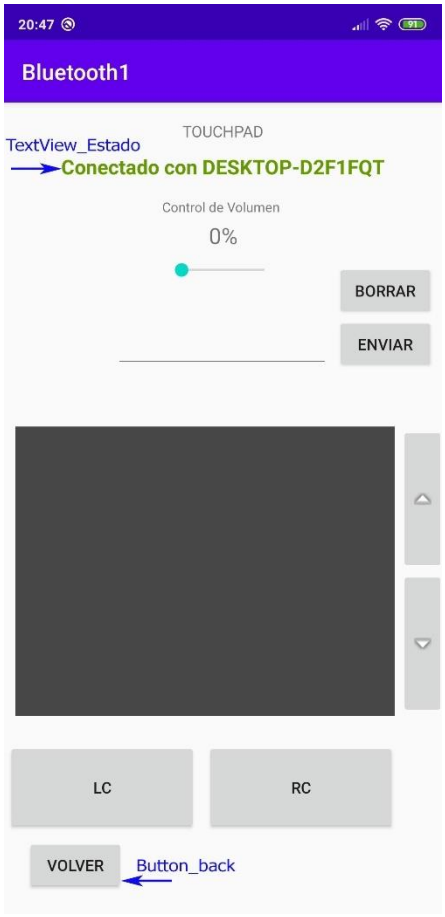

Figura 16. Layout activity_touchpad.xml

La figura 16 muestra la vista de esta actividad. En primer lugar, se enumeran los elementos del *layout* que son independientes de los tres módulos funcionales:

- TextView_Estado: Describe el estado de la conexión. Mostrará el texto "No Conectado" con la fuente en color rojo si el intento de conexión ha fallado. "Conectado con [*nombre*]" y color verde si la conexión ha sido exitosa.
- Button_back: Este botón es idéntico al de la actividad anterior. Su función es cerrar la actividad actual y volver a la anterior.

El resto de elementos del *layout* de esta actividad se han clasificado en 3 grupos o módulos funcionales: el control de volumen, el control de teclado y el control del ratón. A continuación, se profundiza en los elementos y el funcionamiento de cada uno de estos módulos.

4.1.3.2 Control del volumen

El control de volumen permite al usuario interaccionar con la barra para establecer el volumen del ordenador al nivel que desee. Consta de los siguientes elementos:

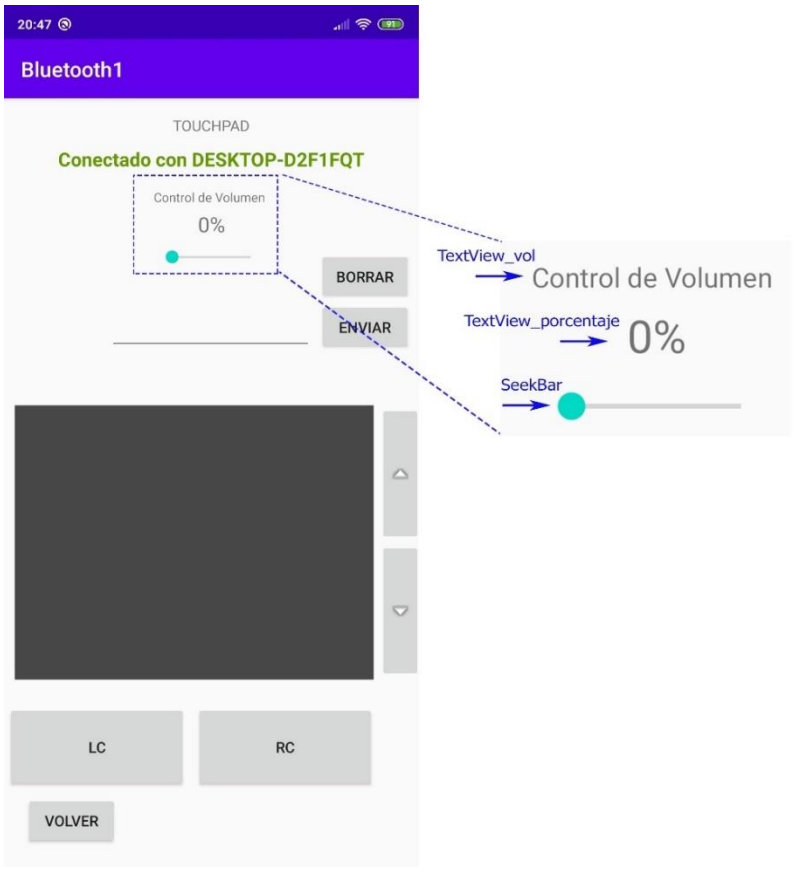

Figura 17. Elementos del control de volumen.

- TextView_vol: Muestra el texto "Control de Volumen".
- TextView_porcentaje: Muestra el volumen en valor numérico del 0% al 100%.
- SeekBar: Es una pequeña barra con la que el usuario puede controlar con precisión el volumen del ordenador.

En lo referente a la lógica de este control de volumen, el método *onCreate()* contiene el *listener* de la *SeekBar* (figura 18). Al crear este listener, Android Studio automáticamente sobreescribe tres métodos:

-*onProgressChanged()*, que detecta cuando cambia la posición de la *SeekBar*. En caso de que el usuario modifique el volumen, se actualizará TextView_porcentaje al nuevo valor haciendo uso de *setText()*.

-*onStartTrackingTouch()*, el cual se deja intacto. No resulta útil para implementar la funcionalidad deseada.

-*onStopTrackingTouch()*, un método muy interesante ya que, una vez el usuario levanta el dedo del control de volumen, nos permite saber este último valor en un rango numérico del 0 al 100. Una vez obtenido dicho valor, se hace uso del método *escribe()* y se envía el número en un *string* de 8 caracteres. El resto de trabajo es cosa del Servidor, quien recibirá el mensaje, obtendrá el número y pondrá el volumen del sistema a dicho valor.

```
seekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {

    @Override
    public void onProgressChanged(SeekBar seekBar, int i, boolean b) {
        porcentaje.setText(""+i+"%");
    }

    @Override
    public void onStartTrackingTouch(SeekBar seekBar) {

    }

    @Override
    public void onStopTrackingTouch(SeekBar seekBar) {

        escribe( mensaje: "+-vol.+"+porcentaje.getText()+"");
        Log.d(TAG,  msg: "SeekBar para en "+porcentaje.getText());
    }
});
```

Figura 18. Listener del SeekBar

4.1.3.3 Control del teclado

El control de teclado es sin duda una de las funcionalidades más importantes en esta aplicación. Permite al usuario escribir y borrar caracteres por pantalla sin necesidad de tener el teclado físico delante, algo muy útil en ciertas situaciones. Hay que tener presente que sólo tendrá sentido utilizar estos controles cuando previamente se ha posicionado el cursor sobre un área que permite la entrada de caracteres, como un formulario o un documento de texto.

A continuación, se enumeran los elementos (figura 19) que conforman esta parte del *layout*, así como una breve explicación de cada uno de ellos.

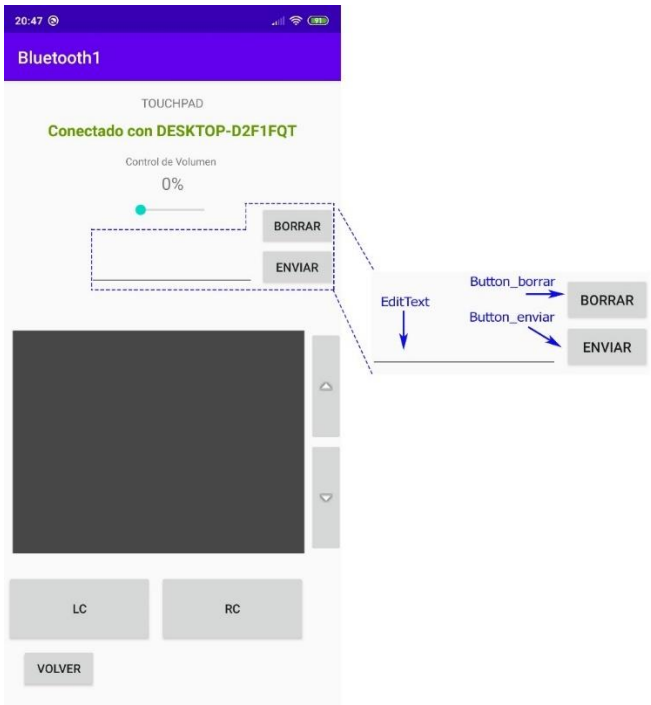

Figura 29. Elementos del control de teclado.

- EditText: Es un pequeño recuadro que, al tocarlo, se despliega el teclado del usuario y se puede escribir en él.
- Button_enviar: Botón que envía la cadena de texto al ordenador para que éste lo escriba por pantalla.

- Button_borrar: Borra un carácter en el ordenador.

Cuando el usuario escribe en el *EditText* y pulsa el botón enviar, se ejecuta el *listener* de este botón, el cual contiene el código que prepara el mensaje a enviar al servidor añadiendo un carácter "$" seguido del mensaje que se quiere enviar, y se envía haciendo uso del método *escribe()* ya mencionado anteriormente. De este modo el servidor reconoce dicho carácter especial e interpreta que a continuación está el mensaje. La cadena que se envía podría ser por ejemplo la siguiente, sin comillas: '$Probando el teclado'

También se puede dar el caso que ya se ha enviado el mensaje y el servidor ya lo ha escrito por pantalla, pero el usuario se ha equivocado y desea borrar un carácter. Para ello se posicionará detrás del carácter que desea borrar y pulsará el botón Borrar. Esta acción ejecuta el *listener* de este botón, y mediante el uso del método *escribe()*, enviará al servidor el mensaje '_borra_' (sin comillas) que el servidor interpretará como una pulsación de la tecla Retroceso, eliminando así el carácter no deseado.

4.1.3.4 Control del ratón

Por último, el control del ratón. Este es sin duda el más importante de los tres, ya que es extremadamente difícil manejar un ordenador si no se tiene este periférico disponible. Esto es precisamente lo que simula este módulo.

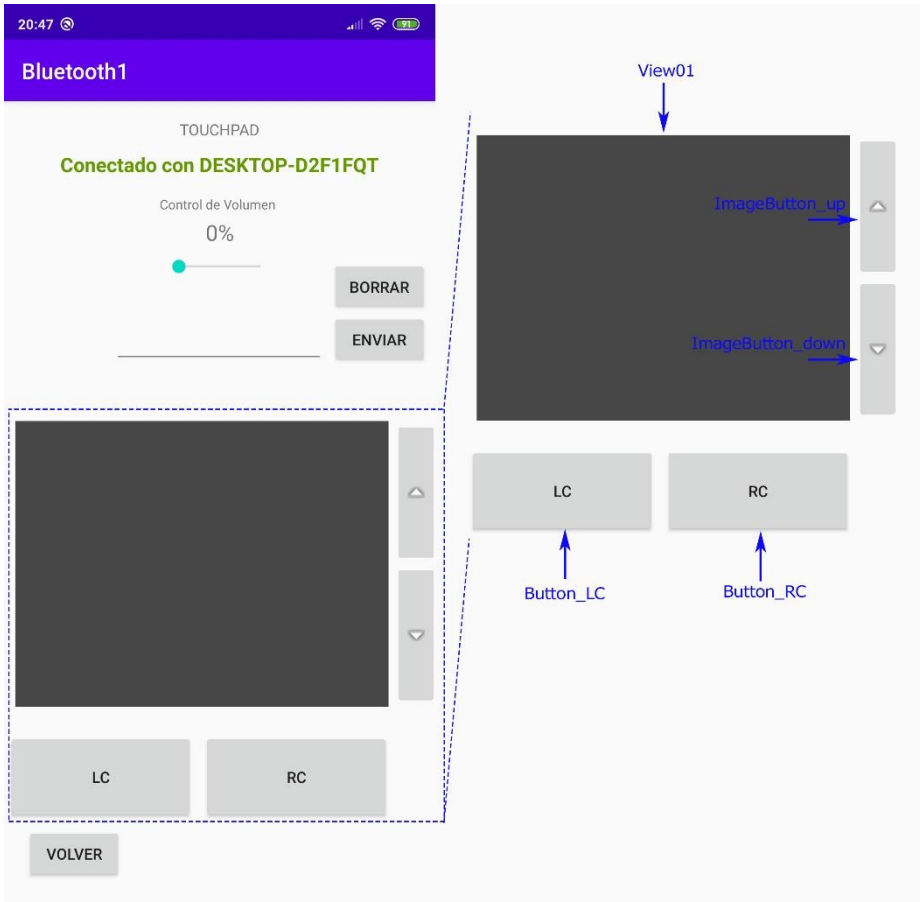

Figura 20. Elementos del control de ratón.

Se distinguen los siguientes 5 elementos:

- View01: Esta vista es un rectángulo de 340dp * 260dp que delimita la superficie táctil del touchpad. El usuario lo utilizará para manejar el movimiento del ratón con precisión.

- ImageButton_up: Una imagen que muestra una flecha hacia arriba. Su función es simular un movimiento hacia arriba con la rueda del ratón, o *scroll up*.
- ImageButton_down: Una imagen que muestra una flecha hacia abajo. Su función es simular un movimiento hacia abajo con la rueda del ratón, o *scroll down*.
- Button_LC: Este botón hace la función del clic izquierdo del ratón.
- Button_RC: Este botón hace la función del clic derecho del ratón.

En lo referente al control del ratón, la parte lógica se sitúa en el código de la actividad Touchpad. Dentro de este módulo, se pueden encontrar tres partes claramente diferenciadas: el movimiento del ratón, los botones y la rueda.

La parte del movimiento del ratón se rige por los eventos que se detecten sobre la *View01* del touchpad. Para ello, en primer lugar se define una instancia del *GestureDetector()* de la clase *java.lang.Object* en un primer método llamado *Gestos()* (figura 21), el cual permite trabajar con eventos.

```
@SuppressLint("NewApi")
public void Gestos() {
    detectTouch = new GestureDetector( context: this, new OnGestureListener()
            {
                @Override
                public boolean onSingleTapUp(MotionEvent e)
                {...}

                @Override
                public void onShowPress(MotionEvent e) { Log.d(TAG, msg: "onShowPress"); }

                @Override
                public boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY)
                {...}

                @Override
                public void onLongPress(MotionEvent e)
                {...}

                @Override
                public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY) {Log.d(TAG, msg: "onFling"); return true;}

                @Override
                public boolean onDown(MotionEvent e) { Log.d(TAG, msg: "onDown");return false; }});
}
```

Figura 21. Método Gestos().

De esta forma el programa será capaz de detectar varios tipos de gestos sobre un elemento de la vista. En concreto, se van a sobrescribir los siguientes métodos que se invocan del detector de gestos:

-onSingleTapUp(), que detecta toques simples sobre el touchpad. Este método actúa cuando el usuario levanta el dedo de la pantalla tras un toque simple. Al detectar un toque o clic, este método envía al servidor una cadena *string* con el código correspondiente al clic izquierdo de ratón. Se envía la siguiente secuencia: 0#12@100. El 0 indica al servidor que se trata de la pulsación de un botón. El #12 indica que se trata de una pulsación normal de corta duración. Por último, @100 indica que la pulsación equivale al botón izquierdo del ratón, aunque se haya pulsado sobre el touchpad. Esta característica dota a la aplicación de mayor usabilidad ya que simula el comportamiento normal de cualquier touchpad de cualquier ordenador.

-onLongPress(), actúa igual que el anterior, pero detectando toques simples sobre el touchpad de mayor duración. En este caso, la cadena *string* que se enviará al servidor es la siguiente secuencia: 0#11@100. Se aprecia que es idéntica a la anterior, pero en este caso identificamos un #11, que indica al servidor que la pulsación es de larga duración.

-onScroll(), que se encarga de detectar desplazamientos sobre el touchpad. Este método es el que registra y codifica dicho desplazamiento, y permite el desplazamiento del ratón. Para ello, utiliza dos eventos *MotionEvent*: en el primer toque del usuario se dispara el primer evento, con el que se obtienen las coordenadas X e Y iniciales del toque. El segundo evento detectará las nuevas coordenadas X e Y. De esta forma, se obtiene la diferencia en X y en Y, y se envía la codificación

de la siguiente forma: 1X-5Y8__. El 1 indica al servidor que se trata de un movimiento en el touchpad, que equivaldrá a un movimiento del ratón. X-5 indica que en el eje X, se ha recorrido una distancia de -5 (5 hacia la izquierda). Del mismo modo, Y8 indica que se ha recorrido una distancia de +8 (8 hacia arriba), y se rellena la cadena con dos __ que suman un total de 8 caracteres.

Resulta interesante destacar que este método se ejecutará continuamente mientras el usuario no levante el dedo del touchpad. De esta forma, tras el envío de datos correspondientes al primer movimiento, se toman las últimas coordenadas X e Y como iniciales y se repite el proceso.

El resto de métodos los dejaremos intactos ya que no resultan relevantes para el desarrollo de esta aplicación.

En segundo lugar, se encuentran los botones del ratón. Esta parte es más sencilla que la anterior, ya que el funcionamiento es mucho más básico: mediante el uso de *listeners*, la actividad puede detectar toques en elementos de las vistas. En este caso, se detectan toques en el botón LC y el botón RC. Como se ha querido diferenciar entre toque simple y toque largo tanto del botón izquierdo como del derecho, se han utilizado 4 *listeners*: dos para el botón izquierdo (uno corto y otro largo) y dos para el derecho (uno corto y otro largo). Cada uno de estos enviará al servidor una cadena de datos distinta.

Botón	Tipo de pulsación	Codificación del mensaje
LC	Corta	0#12@100
LC	Larga	0#11@100
RC	Corta	0#12@101
RC	Larga	0#11@101

Tabla 2. Codificación de mensajes.

En tercer y último lugar, se encuentran los botones laterales. Como ya se ha comentado anteriormente, su función es desplazar la pantalla hacia arriba o hacia abajo, del mismo modo que actúa la rueda central del mouse. Para ello hacen uso del método *Scroll()* que simplemente envía al servidor una cadena de datos dependiendo si se quiere desplazar hacia arriba o hacia abajo.

```
arrowUp.setOnClickListener((view) → {
        Log.d(TAG,  msg: "BTN --> SCROLL UP");
        Scroll( direccion: "arriba");
});
arrowUp.setOnLongClickListener((view) → {
        Log.d(TAG,  msg: "BTN --> LONG SCROLL UP");
        Scroll( direccion: "arriba");
        Scroll( direccion: "arriba");
        Scroll( direccion: "arriba");
        return true;
});
arrowDown.setOnClickListener((view) → {
        Log.d(TAG,  msg: "SCROLL DOWN");
        Scroll( direccion: "abajo");
});
arrowDown.setOnLongClickListener((view) → {
        Log.d(TAG,  msg: "BTN --> LONG SCROLL DOWN");
        Scroll( direccion: "abajo");
        Scroll( direccion: "abajo");
        Scroll( direccion: "abajo");
        return true;
});
```

```
public void Scroll(String direccion){
    if(direccion.equals("arriba")){
        escribe( mensaje: "S1000000");
    } else if(direccion.equals("abajo")){
        escribe( mensaje: "S2000000");
    }
}
```

Figura 22. Listeners de los botones laterales y método Scroll.

Capítulo 5. Servidor

5.1 Descripción del programa servidor

La aplicación cliente explicada en el punto 4 necesita de un servidor que actúe en concordancia con la misma. Para ello, se ha desarrollado un servidor en lenguaje Java, utilizando el IDE *NetBeans*.

El proyecto consta de un solo archivo llamado *BluetoothServer_4.java*, además de la librería que se ha importado para implementar algunas de las funcionalidades necesarias. Esta librería es *BlueCove* (versión 2.1.1) que da soporte al API JSR-82 para Windows

Como la gran mayoría de programas desarrollados en java, en primer lugar, se encuentra la cláusula *package*. Ésta es de vital importancia, ya que permite agrupar todas las clases que se desarrollen en un mismo paquete. A continuación, están las cláusulas *import*. Éstas permiten importar otros paquetes que a su vez contienen otras clases para ser usadas en el paquete actual.

Una vez declaradas las sentencias *package* e *import*, se encuentran las *clases* que conforman el programa.

5.1.1 BluetoothServer_4

BluetoothServer_4 es el nombre que recibe la única clase del programa servidor. Ésta contendrá a su vez varios métodos que realizan distintas funciones y se explicarán más adelante.

```
public class BluetoothServer_4 {

    public final String LEFT_SHORT_CLICK = "0#12@100";
    public final String LEFT_LONG_CLICK = "0#11@100";
    public final String RIGHT_SHORT_CLICK = "0#12@101";
    public final String RIGHT_LONG_CLICK = "0#11@101";
    private String MY_UUID = "00001101000010008000000805F9B34FB";

    public static void main(String[] args) {
        new BluetoothServer_4();
    }

    public BluetoothServer_4() {
        StreamConnectionNotifier mStreamConnectionNotifier = null;

        try {
            mStreamConnectionNotifier = (StreamConnectionNotifier) Connector.open("btspp://localhost:" + MY_UUID);
            System.out.println("Servidor a la espera de peticiones de conexión...");
        } catch (Exception e) {
            e.printStackTrace();
        }

        try {
            while (true) {
                StreamConnection connection = mStreamConnectionNotifier.acceptAndOpen();
                System.out.println("Conexión aceptada. Cliente conectado. ");

                new ClientThread(connection).start();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

Figura 23. Método main() y BluetoothServer_4().

En la figura 1 se aprecia el método BluetoothServer_4(). En él, se diferencian dos bloques *try-catch*. El primero de ellos abrirá una *interface* de la clase *StreamConnectionNotifier* (paquete *javax.microedition.io.Connector*), necesaria para definir las características de la conexión. Se aprecia también que la URL sigue el formato para una conexión del tipo RFCOMM (figura 24).

The URL format for an RFCOMM `StreamConnection`: `btspp:` *//hostname:* `[` *CN* `|` *UUID* `]`; *parameters*

Figura 24. Formato de la URL para conexión RFCOMM.

En este caso, la URL prescinde de *CN*, que hace referencia al *Channel Number* ya que con el UUID es suficiente. Tampoco se utilizará ningún parámetro.

El UUID es el *Universally Unique Identifier*, un número de 128 bits utilizado para establecer conexiones Bluetooth. Este UUID es el mismo tanto en el programa cliente como en el servidor.

Una vez el servidor ya está a la escucha (figura 25), tiene que aceptar las peticiones de conexión entrantes que reciba.

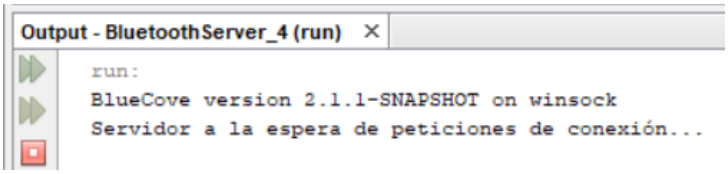

Figura 25. Resultado de la ejecución del programa servidor.

Es aquí donde actúa el segundo bloque *try-catch*. Contiene un bucle *while()* que aceptará siempre la conexión del cliente. Cuando el cliente en su app Android pulse sobre un elemento de la lista (figura 13), en el log del servidor se podrá leer que la conexión ha sido aceptada y el cliente está conectado (figura 26). Entonces se abrirá un hilo *ClientThread()*.

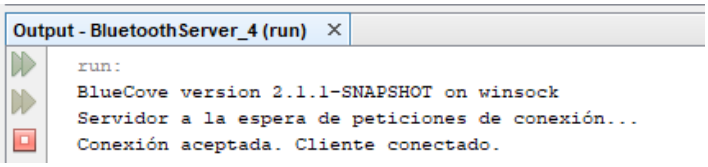

Figura 26. Log del servidor muestra al cliente conectado.

En cuanto al hilo *ClientThread()*, se va a profundizar en el método *run()* del mismo (figura 27), el cual contiene toda la lógica que permite al usuario controlar el ordenador.

```
private class ClientThread extends Thread {
    private StreamConnection mStreamConnection = null;
    public ClientThread(StreamConnection sc) {
        mStreamConnection = sc;
    }

    @Override
    public void run() {
        try {
            Robot robot = new Robot();
            int c = 0;
            byte[] buffer = new byte[100];

            BufferedInputStream bis = new BufferedInputStream(mStreamConnection.openInputStream());

            System.out.println("Leyendo datos...");
```

Figura 27. Primera parte del método run() de ClientThread().

Al inicio del método *run()*, se hace uso de un bucle *try-catch*. Dentro se encuentra en primer lugar la declaración de un objeto de la clase *Robot*, de la que hablaremos más adelante. También se declara un array de bytes que leerá el *buffer* de entrada.

En segundo lugar, se declara el *BufferedInputStream*. Su principal función es abrir un *stream* de entrada para leer los mensajes que envía el cliente, y por otra parte interpretar dichos mensajes y actuar como desee el usuario.

Para leer estos mensajes, se hace uso del método *read()* en el *BufferedInputStream*. Cabe destacar que, el servidor recibe un array de bytes. Cada uno de estos bytes corresponde a un carácter. Por

este motivo es necesario hacer uso del método *Traductor()* (figura 28), un método auxiliar que, dado un array de bytes, devuelve un *string* con el mensaje original.

```
public String Traductor(byte[] b) {

    int lenght = b.length;
    String mensaje = "";

    int msj = 0;
    for (int i = 0; i < lenght; i++) {
        //Pasamos del array de bytes a int
        msj = b[i]; //Obtenemos 72 en msj int.

        //Pasamos 72 a caracter --> de msj int a caracter char: 72-->H
        char caracter = (char) msj;

        mensaje = mensaje + Character.toString(caracter);

    }
    return mensaje;
}
```

Figura 28. Método Traductor()

Cuando el servidor ya ha traducido el mensaje y obtiene el *string* enviado por el usuario, el siguiente paso es saber qué significa dicho mensaje.

Para ello, lo lógico hubiera sido utilizar una sentencia *switch-case* tal y como se hizo al principio, pero por razones desconocidas dicho bucle no funcionaba. Como alternativa se han utilizado varios bucles *if-else* (figura 29) para detectar e interpretar los mensajes.

```
        while (true) {
            c = bis.read(buffer);
            //Pasamos a string el buffer
            String texto = Traductor(buffer);
            buffer = resetBuffer();

            if (texto.startsWith("0")) {  //Llamamos a CLICADOR si el texto empieza por 0.
                System.out.println("CLICK --> " + texto);
                Clicador(robot, texto);
                texto = "";
            } else if (texto.startsWith("1") && texto.contains("X") && texto.contains("Y")) {  //Llamamos a MOVEDOR si el texto empieza por 1.
                // System.out.println("MOVER--> " + texto);
                Movedor(robot, texto);
                texto = "";
            } else if (texto.startsWith("S")) {  //Llamamos a SCROLLER  si el texto empieza por S
                System.out.println("SCROLL--> " + texto);
                Scroller(robot, texto);//pillamos la segunda posición del texto que empieza por S (S1 o S2)
                texto = "";
            } else if (texto.startsWith("$") && texto.length() > 2) { //Si es S1, Scroll UP, si es S2, Scroll DOWN
                String mensaje = texto.substring(1);
                System.out.println("ESCRIBE--> " + mensaje);
                for (int i = 0; i < mensaje.length(); i++) {
                    escribe(robot, mensaje.charAt(i));
                }
            } else if (texto.contains("+-vol.")) {
                String dato = texto.substring(7);
                dato = dato.replaceFirst("%", "");
                System.out.println(dato);
                comando(dato);
            } else if (texto.contains("_borra_")) {
                metodoEscritor(robot, '\b');
            }
            if (c == -1) {
                System.out.println("Conexión finalizada.");
                System.out.println("Buffer: " + buffer);
                break; }
        }
        bis.close();
        mStreamConnection.close();
    } catch (Exception e) {
        e.printStackTrace();
```

Figura 29. Segunda parte del método run() de ClientThread().

En la figura 29 anterior, y una vez obtenido el mensaje en la variable *texto*, empiezan los *if-else*.

En primer lugar, se comprueba si la cadena recibida empieza por '0'. De ser así, significa que el mensaje recibido es un clic de ratón. Entonces se llama al método *Clicador()* (figura 30) el cual diferenciará qué tipo de clic se ha recibido, y hará uso del objeto definido anteriormente de la

clase Robot para emular ese clic. *BUTTON1_MASK* hace referencia al botón izquierdo, y *BUTTON3_MASK* al botón derecho del ratón.

```
public void Clicador(Robot robot, String texto) {
    if (texto.contains("100")) {
        System.out.println("CLIC IZQUIERDO.");
        robot.mousePress(InputEvent.BUTTON1_MASK);
        robot.mouseRelease(InputEvent.BUTTON1_MASK);
    }
    if (texto.contains("101")) {
        System.out.println("CLIC DERECHO.");
        robot.mousePress(InputEvent.BUTTON3_MASK);
        robot.mouseRelease(InputEvent.BUTTON3_MASK);
    }
}
```

Figura 30. Método Clicador()

Siguiendo con la figura 29, si el texto recibido no empieza por '0' se da paso al segundo *bucle if-else*. Este bucle comprueba que el mensaje recibido empiece por '1', lo que indica que se trata de un movimiento del mouse. También se comprueba que la cadena contenga un carácter *X* y otro *Y*. Si la comprobación es correcta, se llamará al método *Movedor()* (figura 31).

```
public void Movedor(Robot robot, String texto) {
    String dX, dY;
    int longi = texto.length(); //Longitud
    System.out.println("longi = " + longi + "\nMOVER--> " + texto);

    int X, Y;

    X = texto.indexOf("X");
    Y = texto.indexOf("Y");

    dX = texto.substring(X + 1, Y);
    System.out.println("Substring X: " + dX);
    dY = texto.substring(Y + 1, texto.indexOf("_"));
    System.out.println("Substring Y: " + dY);

    int ans[] = new int[2];
    ans[0] = Integer.parseInt(dX);
    ans[1] = Integer.parseInt(dY);

    //Falta saber dónde está el puntero
    PointerInfo a = MouseInfo.getPointerInfo();
    Point b = a.getLocation();
    int startX = (int) b.getX();
    int startY = (int) b.getY();

    robot.mouseMove(startX + ans[0], startY + ans[1]);
    texto = null;
}
```

Figura 31. Método Movedor()

El método *Movedor()* utiliza la cadena de texto recibida para detectar las unidades de desplazamiento que el usuario necesita desplazar el ratón. Para ello, hace uso de *substring()* y obtiene el incremento tanto en la coordenada X como en la Y (variables dX y dY).

A continuación se utiliza el método *parseInt()* porque interesa tener estos datos almacenados en variables del tipo *int* y no del tipo *string.*

Por último, se obtiene la posición actual del puntero, y mediante el método *mouseMove()* de la clase *Robot* se consigue desplazarlo las unidades deseadas.

Siguiendo con el método *run()* de *ClientThread()* (figura 29), se llega al método *Scroller()*. Este método es llamado cuando el servidor recibe una cadena de datos que empieza por 'S'. Al llamar a este método, se le pasan como argumentos el objeto de la clase *Robot* y el texto recibido.

```
public void Scroller(Robot r, String texto) {
    if (texto.contains("S1")) { //Scroll Up
        System.out.println("UP");
        r.mouseWheel(-1);
    } else if (texto.contains("S2")) { //Scroll Down
        System.out.println("DOWN");
        r.mouseWheel(1);
    }
}
```

Figura 32. Método Scroller().

En caso de que el texto empiece por 'S' y contenga las subcadenas 'S1' o 'S2', este método moverá una unidad en la dirección indicada la rueda del ratón con *mouseWheel()*, S1 hacia arriba y S2 hacia abajo.

Suponiendo que el mensaje recibido no cumple ninguna de las condiciones anteriores, el programa seguirá ejecutando los *if-else* anidados. La siguiente sentencia evaluará si la cadena empieza por '$' además de tener una longitud mayor a 2. En este caso, el servidor identificará que el usuario quiere escribir un mensaje por pantalla. Para ello se guardará la cadena recibida a partir del carácter en la posición 1, de esta forma se consigue eliminar el '$' que precede al mensaje. A continuación, se recorre todo el mensaje con un bucle for, ejecutando el método *escribe()* en cada una de sus iteraciones. Este nuevo método a su vez hace uso de *metodoEscritor()*. La forma en la que actúan conjuntamente es la siguiente:

```
public void metodoEscritor(Robot robot, int... keyCodes) {
    int length = keyCodes.length;

    for (int i = 0; i < length; i++) {
        robot.keyPress(keyCodes[i]);
    }
    robot.delay(10);
    for (int i = length - 1; i >= 0; i--) {
        robot.keyRelease(keyCodes[i]);
    }
}

public void escribe(Robot robot, char character) {
    switch (character) {
        case 'a':
            metodoEscritor(robot, VK_A);
            break;
        case 'b':
            metodoEscritor(robot, VK_B);
            break;
        case 'c':
            metodoEscritor(robot, VK_C);
            break;
        case 'd':
            metodoEscritor(robot, VK_D);
            break;
        case 'e':
            metodoEscritor(robot, VK_E);
            break;
        case 'f':
            metodoEscritor(robot, VK_F);
```

Figura 33. metodoEscritor() y escribe().

En primer lugar, el método *run()* de *ClientThread()* realizará varias llamadas a *escribe()*, pasándole como parámetros el objeto *Robot* y un carácter del mensaje recibido. Entonces *escribe()* hará uso del *switch-case* y, según qué carácter sea, ejecutará *metodoEscritor()* pasándole una clave de tecla (o *keyCode*). Este *keyCode* es necesario para que el método *keyPress()* funcione simulando una pulsación de la tecla deseada. Tras un pequeño *delay* de 10 milisegundos, se soltará dicha tecla y se habrá escrito un carácter por pantalla. De esta forma es como el programa servidor consigue escribir caracteres por pantalla.

También se puede dar el caso de que el servidor reciba una cadena que empieza por '+-vol.' seguido de un número. Al detectar este inicio de la cadena, el servidor interpreta que la intención del usuario es modificar el volumen del sistema. Entonces ejecutará el método *comando()* al que se le pasará el valor del volumen en *string*.

```
public void comando(String valr) {
    if (valr != null) {
        try {
            String cc = "cmd /K cd c:\\SetVol && SetVol.exe && setvol "+valr.trim()+"";
            System.out.println(cc);
            Process proceso = Runtime.getRuntime().exec(cc);
            BufferedReader br = new BufferedReader(new InputStreamReader(proceso.getInputStream()));
            String resultado = null;
            if ((resultado = br.readLine()) != null) {
                System.out.println(resultado);
            }
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
    }
}
```

Figura 34. Método comando().

La función de este método (figura 34) es modificar el volumen del sistema. Para ello, se preparan los comandos para la línea de comandos de Windows (o *cmd*) para modificar el valor del volumen del sistema. De esta forma, cada vez que el usuario cambie el volumen en el teléfono, se ejecutará este método que también cambiará el volumen en el dispositivo del servidor.

Por último, se realizará la última comprobación antes de desestimar el mensaje. El último caso que podría darse es que se trate de un mensaje para borrar un carácter. De ser así, el servidor detecta que el mensaje contiene la subcadena '_borra_'. Cuando el servidor detecta esta subcadena, ejecuta una llamada al *metodoEscritor()* para que éste simule una pulsación de la tecla de borrado. El resultado sería el mismo que si el usuario pulsa la tecla de borrado en el teclado de su ordenador.

Capítulo 6. Conclusiones y propuesta de trabajo futuro

6.1 Conclusiones

El objetivo inicial de este trabajo era desarrollar una aplicación completamente funcional, tanto del lado del cliente como del servidor, que permitiese al usuario interactuar con el ordenador a pesar de prescindir de ciertos periféricos.

Durante el desarrollo del proyecto, han ido surgiendo muchos imprevistos que no se contemplaban al principio. Es el trabajo de un ingeniero identificar estos problemas, pero sobre todo dar con una solución rápida, fiable y viable para poder seguir adelante. También es de vital importancia asumir que éstos van a aparecer y, por tanto, hay que dedicar un espacio temporal extra en la planificación del trabajo para poder buscar soluciones a estos problemas que todavía no han surgido.

En concreto, uno de estos problemas ha sido la enorme dificultad de desarrollar prácticamente de forma simultánea el cliente y el servidor. Esto se contempló en la fase de planificación, pero sin duda, fue subestimado. Un pequeño cambio en el programa cliente podía hacer que el servidor fuese incapaz de interpretar un solo comando, y viceversa.

Otro de los mayores retos a superar ha sido la curva de aprendizaje del uso de Android Studio. Este IDE es extremadamente completo, y habituarse a él no es tarea sencilla. Por otra parte, la forma de programar en Android es totalmente distinta a la de programar un servidor, por ejemplo. Hay que tener en cuenta los ciclos de vida de las diferentes actividades y, aunque cliente y servidor están programados en el mismo lenguaje, son radicalmente distintos.

Por último, el tercer mayor reto ha sido establecer el canal de comunicación entre las dos aplicaciones e interpretar los mensajes. Al principio, el servidor era capaz de recibir cadenas de texto, pero éstas estaban codificadas. Era crucial entender esta codificación, para poder realizar el proceso inverso y que el servidor mostrase caracteres legibles. Tras varios días de investigación, se pudo resolver esto, y conseguir el resultado deseado.

Tras meses de investigación, documentación, planificación y desarrollo de la aplicación, se puede afirmar que todos los objetivos han sido cumplidos, aunque sin duda, no todos se han conseguido de la forma prevista. Por ejemplo, es relativamente sencillo simular una pulsación del teclado en Java, por lo que uno de los primeros avances fue escribir caracteres por pantalla, pero la combinación de teclas que incrementa o decrementa el volumen no era válida. Se tuvo que cambiar la forma de abordar el problema, y se dio con una solución incluso más completa, ya que el ajuste de volumen actual es mucho más preciso. De un modo similar surgieron problemas con el movimiento del ratón. Al inicio, el programa era demasiado simple y el ratón se movía a saltos, pero con unos pequeños cambios en la forma en que el cliente mandaba las coordenadas al servidor se mejoró la experiencia de uso.

Finalmente, cabe destacar que, además de lograr los objetivos propuestos al inicio, se han adquirido nuevos conocimientos, como la mejora del aspecto visual de una aplicación en Android a través de la edición de *layouts*, algo que hasta el inicio del desarrollo se sabía, pero no con profundidad. O sobre todo la capacidad alumno de idear una propuesta de proyecto, estudiarla y llevarla a cabo.

6.2 Propuesta de trabajo futuro

En cuanto a las funcionalidades que posee la aplicación, hay algunas mejoras que sería interesante implementar de cara al futuro:

-Posibilidad de quitar todos los periféricos. La aplicación podría servir también de monitor de un ordenador. De esta forma, el usuario estaría viendo la pantalla del ordenador en su teléfono Android, y mediante controles táctiles sería capaz de controlarla.

-Utilizar otras tecnologías para la comunicación. En entornos de *LAN*, sería conveniente utilizar la tecnología *WiFi*, ya que proporciona un mayor ancho de banda y mayor alcance, además de un menor consumo de energía y menor latencia.

-Dotar al ordenador de la capacidad de controlar remotamente el *smartphone*. Es decir, el ordenador actuaría como cliente y el teléfono, como servidor.

-Implementar la funcionalidad de transferencia de archivos entre los dispositivos.

-Ampliar la aplicación para otras plataformas. Actualmente la aplicación se limita a entornos con Windows para el servidor, y teléfonos con Android para el cliente. La propuesta sería desarrollar la aplicación para más plataformas, como iOS, MacOS, Linux, etc.

Capítulo 7. Bibliografía

Documentación relacionada con Tecnología Bluetooth:

[1] Franco, D., & Castillo, F. (2009). Comunicaciones Inalámbricas: Bluetooth. *Revista Prisma Tecnológico*, *1*(1), 19-21. Recuperado a partir de https://revistas.utp.ac.pa/index.php/prisma/article/view/412

[2] https://www.bluetooth.com/

[3] https://www.bluetooth.com/specifications/gatt/

[4] https://softwarelab.org/es/bluetooth/

[5] http://www.bluecove.org/

[6] https://es.slideshare.net/JuanLuisMuozArias/estndar-ieee-802-75307058#:~:text=Se%20ha%20publicado%20una%20versi%C3%B3n,IEEE%20802.15.1%2D2005.&text=(WPAN%2FBluetooth)-,5.,ISM%20de%20los%202.4%20GHz.

[7] http://tauja.ujaen.es/bitstream/10953.1/11906/1/Memoria_TFM_Espia_Bluetooth.pdf

[8] https://developer.android.com/guide/topics/connectivity/bluetooth?hl=es-419

[9] https://www.osmosislatina.com/conectividad/bluetooth.htm

Documentación relacionada con Android

[10] https://developer.android.com/guide/topics/ui/ui-events

[11] https://developer.android.com/training/gestures/movement

[12] https://histinf.blogs.upv.es/2012/12/14/android/

[13] https://es.bccrwp.org/solution/android-vs-ios-market-share-worldwide/

[14] https://www.xatakandroid.com/sistema-operativo/historia-y-evolucion-de-android-como-un-sistema-operativo-para-camaras-digitales-acabo-conquistando-los-moviles

[15] https://developer.android.com/studio

Documentación relacionada con programación del servidor

[16] http://keycode.info/

[17] https://www.rlatour.com/setvol/index.html?sc

[18] http://www.sistemamid.com/panel/uploads/biblioteca/1/619/640/641/3792.pdf

[19] http://personales.upv.es/almarab/SDCIR/Tema%20R5%20Otras%20t%C3%A9cnicas%20de%20acceso%20inal%C3%A1mbrico%202011.pdf

[20] https://docs.oracle.com/netbeans/nb82/netbeans/NBDAG/toc.htm